Unrealistic Models for Realistic Computations:[*]

How Idealisations Help Represent Mathematical Structures and Found Scientific Computing

Philippos Papayannopoulos

Sidney M. Edelstein Centre for History and Philosophy of Science Technology and Medicine, The Hebrew University of Jerusalem.

Abstract

We examine two very different approaches to formalising real computation, commonly referred to as "Computable Analysis" and "the BSS approach". The main models of computation underlying these approaches —bit computation (or Type-2 Effectivity) and BSS, respectively— have also been put forward as appropriate foundations for scientific computing. The two frameworks offer useful computability and complexity results about problems whose underlying domain is an uncountable space (such as $\mathbb{R}$ or $\mathbb{C}$). Since typically the problems dealt with in physical sciences, applied mathematics, economics, and engineering are also defined in uncountable domains, it is fitting that we choose between these two approaches a foundational framework for scientific computing. However, the models are incompatible as to their results. What is more, the BSS model is highly idealised and unrealistic; yet, it is the *de facto* implicit model in various areas of computational mathematics, with virtually no problems for the everyday practice.

This paper serves three purposes. First, we attempt to delineate what the goal of developing foundations for scientific computing exactly is. We distinguish between two very different interpretations of that goal, and on the separate basis of each one, we put forward answers about the appropriateness of each framework. Second, we provide an account of the fruitfulness and wide use of BSS, despite its unrealistic assumptions. Third, according to one of our proposed interpretations of the scope of foundations, the target domain of both models is a certain mathematical structure (namely, floating-point arithmetic). In a clear sense, then, we are using idealised models to study a purely mathematical structure (actually a class of such structures). The third purpose is to point out and explain this intriguing (perhaps unique) phenomenon and attempt to make connections with the typical case of idealised models of empirical domains.

[*]Preprint of an article published in *Synthese*: https://doi.org/10.1007/s11229-020-02654-8

Keywords BSS/Real-RAM Model; Computable Analysis; Bit computation; Foundations of Scientific Computing; Idealisations; Real Complexity; Unrealistic Models; Well-posed, Ill-posed Physical Problems; Stability; Floating-Point Arithmetic

1 Introduction

In a (2004) volume of the *Notices of the AMS*, an article titled "Computing over the Reals: Where Turing Meets Newton" by L. Blum appeared, presenting an algebraic model of computation for real-valued functions. The article was a summary of previous work by a group of mathematicians, aiming to offer a foundation for scientific computing. Two years later, another article, titled "Computing over the Reals: Foundations for Scientific Computing" (2006) by M. Braverman and S. Cook, appeared in the same journal. As its title suggests, it shared similar goals to the former work: to articulate a model of computation for real functions, which can also offer a useful foundation for scientific computing. Nevertheless, the two articles described models that are fundamentally incompatible.

Why is this important? It is almost a cliché to say that computers are nowadays used in almost every scientific lab, and that their results are presumed trustworthy. Running of simulations is a prominent use, but other uses are merely computations of approximate solutions to differential equations that describe physical systems. Thus, the issue is of great interest for theoretical, but even more so practical, reasons. From a theoretical perspective, we need a mathematical framework within which we will be able to classify problems of the computational sciences according to whether they are easy, hard, or impossible to compute. More specifically, we need to be able to say whether a problem has any solutions that are in principle computable (or whether we need to make do with *approximate* solutions), and if so, what amount of computational resources these solutions require (typically expressed as the needed amount of time and storage). From a practical perspective, we also need to be able to compare the different *methods* (algorithms) available for solving (or approximating) a given problem. This latter comparison is typically made on the basis of a method's efficiency (and accuracy, in the case of approximations), which translates into the method's time cost (and the error of its approximate output). It should be noted that an additional factor that may further complicate this task is the inevitable rounding (and other types of) errors that arise whenever any such methods are to be implemented in digital computers.

Now, one can ask why the well-developed theories of computability and computational complexity, from mainstream computer science, do not suffice for the aforementioned goals. The quick answer is that those theories are concerned with problems defined over countable, discrete domains, such as the non-negative integers. On the other hand, physical sciences, applied mathematics, economics, and engineering are typically concerned with problems defined over uncountable, continuous domains, such as the real or the complex numbers. Therefore, we need theories whose target domain is primarily computation of functions defined in uncountable spaces.

Unfortunately (but also interestingly), there is no universally accepted framework for studying the computability and complexity of problems defined in uncountable spaces. In fact, two very different traditions of computation have emerged, which have run a parallel, non-intersecting course. Both traditions try to understand the nature of computation, to prove novel results, and to offer foundations for scientific computing; however, they have developed incompatible frameworks, in the sense that they characterise the same problems differently with regard to their computability and complexity. The two frameworks are, namely, the so-called *BSS-model* and *Computable Analysis* (CA) (aka 'Type-2 Effectivity' (TTE)), presented in the two articles mentioned in the beginning.

We briefly present the two approaches in section (2), along with examples of their incompatibility. In section (3), we discuss some informal considerations that might offer some guidance on how to go about choosing between them. Perhaps, the approach that satisfies more of our intuitive desiderata would be the one to accept. It turns out, though, that each approach captures successfully certain desired properties, but also fails in some other aspects that would intuitively be highly desirable.

We claimed that the whole issue is of practical interest also because it pertains to algorithms that are to be executed by digital computers (e.g., in scientific labs). Nevertheless, a mathematically designed algorithm is not flawlessly implemented by a machine, for in practice rounding errors are introduced, which may also accumulate. This is because generally a real number cannot be represented in its entirety by a finite string of symbols, and hence needs to be "truncated" to some finite approximation; the same goes for every output of an operation on such numbers, which subsequently has to be rounded too.

The inevitable occurrence and accumulation of errors, then, owing to limitations of implementation, puts us at a crossroads when it comes to developing foundations for scientific computing. Should practical considerations, such as the accumulation of errors or the properties of different representations, be taken into account for the appropriateness of a theory that provides the much-needed foundations, and, if so, to what extent? We claim that different ways of approaching this dilemma influence accordingly the choice between the rival theories (TTE vs. BSS) of real computation. We thus distinguish between two possible interpretations of what 'foundations of scientific computing' can be about. The first concerns the development of a fundamental universal framework that regiments the notion of 'effective computation over uncountable spaces', expanding its universally accepted counterpart for computation over countable domains (i.e., Turing computability). The second interpretation concerns the development of a framework for comparing and analysing algorithms that practitioners develop with practical limitations of implementation in mind. That is, practitioners know that today's computers typically work with floating-point arithmetic (FPA). So, they often take this into account and focus on constructing and comparing algorithms that minimise FPA round-offs. However, FPA is a mathematical structure with peculiar properties, and very difficult to work with in practice. Therefore, analysing algorithmic costs and complexity problems can be extremely cumbersome if one takes as one's underlying framework the mathematical properties of FPA systems (let alone that many such properties vary according to the particular representation format; see appendix). Thus, according to our second interpretation, an appropriate 'foundation for scientific computing' would mean an easier, unified model to work with, such that it would be general enough to serve as a *foundation*, but, at the same time, specific enough to faithfully *model* FP algorithms. We elaborate on these matters in section (4).

With these distinctions and clarifications in mind, we are subsequently able to explain, in section (5), the following, apparently paradoxical, situation. The BSS framework is a highly idealised and unrealistic model of computation, which assumes that the computing machine can always compare and operate upon any two real numbers (even infinite irrationals) in just one step. Of course, this is impossible in reality, since even just calculating an irrational number (e.g., π) in its entirety would require infinite time; let alone adding it to another infinite number (e.g., e). Furthermore, real computers work with floating-point numbers, as already said, which are rational and finite (see appendix). Therefore, BSS has been heavily criticised by many researchers in computer science and mathematics, on the grounds of its unrealistic assumptions. Nevertheless, it is successfully used as the implicit model in various areas of mathematics, such as numerical analysis, computational geometry, information-based complexity, and others. We investigate the apparently unexpected success of BSS, including exact conditions for its appropriateness as a model of FP algorithms; when these conditions are not met, Type-2 Effectivity (TTE) seems to be the only appropriate framework.

In sum, if the goal of developing foundations for scientific computing is construed in the first way (a fundamental theoretical framework that also explicates 'effective computation'), we argue that computable analysis (CA) is the appropriate approach. But if the goal is to be understood in the second way (with implementational considerations in mind), we argue for a pragmatic approach to the problem of choice: the appropriateness of each model depends on the nature of the particular problem at hand. BSS can be a more tractable and useful model for "higher-level" analyses, whereas the bit-computation model (a practice-oriented formulation of TTE) is preferable for more "fine-grained" analyses. These arguments are discussed in section (6).

Finally, it may be observed that the proposed pragmatic approach is reminiscent of some cases in empirical science, where we likewise use incompatible representations of a single system, differently idealised and in accordance with contextual needs (e.g., a fluid may be represented as continuous or as discrete, according to the desired level of analysis). In the final section (7), we take this analogy further. We discuss some essential conditions that reliable mathematical representations of physical systems generally ought to satisfy, and we suggest that, although in our case we are concerned with the modelling of *mathematical* systems (computations in FPA), the situation is still to a large extent analogous.

2 The Different Approaches

To begin, we briefly examine the two approaches, CA and BSS.

2.1 Computable Analysis

Computable analysis, as a systematic attempt to pin down those parts of mathematical analysis that have a computational meaning (in the sense of being calculable by following a mechanical method), appears as early as the parallel systematic development of ordinary computability theory. Turing, in his seminal (1936) paper, was primarily concerned with continuous mathematics and, more specifically, with the computable *real* numbers. The paper begins:

> The "computable" numbers may be described briefly as the *real* numbers whose expressions as a decimal are calculable by finite means. (1936, 230; emphasis added).

As a careful study of the paper shows, Turing was not really after describing the computable integer-valued functions per se, despite the later typical textbook presentations of his theory. Although he did provide a quick characterisation of such functions, this was just a *working* definition, as an intermediate step towards defining the computable *real-valued* functions (see, in particular, pp. 254-5 in 1936).
But even a few decades earlier, É. Borel had provided an informal characterisation of the computable real numbers and functions as well:

> We say that a number α is computable if, given a natural number n, we can obtain a rational number that differs from α by at most $\frac{1}{n}$.
>
> [..]
>
> I intentionally leave aside the practical length of operations, which can be shorter or longer; the essential point is that each operation can be executed in finite time with a safe method that is unambiguous. (Borel, 1912)[1]

It was after Turing's work, though, that questions of computable results in real analysis became an intensive area of research. The (independent) work of Grzegorczyk (1955) and Lacombe (1955) gave rise to rich developments, and to slightly different approaches (based, e.g., on recursive functions, embodied in Pour-El and Richards 1989, or on oracle Turing machines, embodied in Ko 1991). However, the earliest systematic studies in this area probably go back to 1937-1939, in the work of Banach and Mazur, which, however, was published several decades later (Mazur, 1963) because of WWII.[2]

[1] Quoted and translated in Avigad and Brattka (2014).

[2] See Avigad and Brattka (2014), for a wonderfully written history of the development of real computability.

In its standardly accepted contemporary formulation, computable analysis extends the classical model of a Turing machine to the domain of the real numbers and functions. The standard reference today is Weihrauch (2000). Friendlier and shorter expositions can be found in Braverman and Cook (2006), Braverman (2005), and Brattka et al. (2008). For a brief exposition of the theory and a philosophical discussion of its conceptual underpinnings, see Pégny (2016).

The main problem facing us when we move from computing over the integers to computing over the reals has to do with representation. Representation is essential to computation, since a computing agent (Turing machine or other) standardly manipulates symbols from some symbolic system.[3] Nevertheless, when we move to an uncountable domain, such as $\mathbb{R}$, we still have only countably many names available to denote uncountably many entities.

There are two possible ways to go about tackling this problem. The first would be to consider instead of the whole domain $\mathbb{R}$ of real numbers just the subfield $\mathbb{R}_C$ of computable reals. In fact, that was both Borel's and Turing's approach.[4] In this way, we deal with only a countable space and the problem of representation is overcome by fixing an appropriate coding scheme for Turing machines, and taking as a name for a computable real the code number of the TM that produces it when starting on a blank tape. This approach led to the Russian school of computability (also reflected in the Russian school of constructive mathematics), represented in its most complete form in Aberth (1980).

The second possibility, which became the prominent practice in Europe and North America, is to consider $\mathbb{R}$ in its entirety. This is the approach we mainly consider in this paper and which we hereafter mean by 'Computable analysis' (CA), following standard practice. According to this approach, we use approximations as names for real numbers. More specifically, a number is represented by an infinite sequence of digits, which the longer it is the better it approximates the exact real value of the number. Informally then, a function is computable if, given a "good" rational approximation of the input, there is a Turing program that yields a "good" rational approximation of the output.

2.1.1 The model (aka 'Type-2 Effectivity' or 'Bit-computation')

In classical computability theory, Turing computation of a function $f :\subseteq \mathbb{N}^n \to \mathbb{N}$ can be formulated as computation of string functions $f :\subseteq (\Sigma^*)^n \to \Sigma^*$, where Σ^* is the set of all finite words over a non-empty finite alphabet Σ. Similarly, here we consider a Turing machine[5]

[3] See, Papayannopoulos (2018, ch.4) for a detailed account of that view.

[4] "We say that a function is computable if its value is computable for any *computable* value of the variable. In other words, if α *is a computable number*, one has to know how to compute the value of $f(\alpha)$ with precision $\frac{1}{n}$ for any n." (Borel 1912, quote from Avigad and Brattka 2014; emphasis added).
And, also:
"We cannot define general computable functions of a real variable, since there is no general method of describing a real number, but we can define a computable function of a computable variable." (Turing, 1936, 254)

[5] Often referred to as a 'Type-2 Turing Machine', and the theory as 'Type-2 Effectivity' (abbrv. 'TTE').

operating over strings which are the representations of real numbers. It is convenient to work with dyadic rationals, from $\mathbb{D} = \{\frac{a}{2^b} : a \in \mathbb{Z}, b \in \mathbb{N}\}$, because they always have finite binary expansions. Then, we can represent a real number x on the tape of the machine by a sequence of dyadics $\varphi(1), \varphi(2), \ldots$ approaching x, such that $|x - \varphi(m)| \leqslant 2^{-m}$ ($m \in \mathbb{N}$); for example, $\frac{1}{2}, \frac{2}{4}, \frac{3}{8}, \frac{6}{16}, \frac{11}{32}, \ldots$ as a representation of the number $\frac{1}{3}$.

In the interest of brevity, we restrict ourselves to an informal presentation involving oracle Turing machines, since a more formal treatment would require a lengthier discussion of naming systems (e.g., by employing Cauchy sequences) and representation mappings. The advantage of an oracle-based formulation is twofold: it avoids a treatment based on infinite computations, since writing down the representation of a real number as a complete sequence of dyadics would take infinitely long, and it also offers a natural way to separate the complexity of computing the input x from the complexity of computing on x given as a parameter (Braverman, 2005).

We equip the Turing machine with an oracle. An oracle is a kind of a "black box" that, at any given step of the computation, can be queried by the TM to provide the value of a function $\varphi : \mathbb{N} \to \mathbb{D}$, in one step. Assume a function $f :\subseteq \mathbb{R} \to \mathbb{R}$ and that we want to compute $f(x)$, for some $x \in dom(f)$. We denote by $\bar{x}$ the input on the machine's tape (i.e., the representation of x), and by $\bar{y}$ the output (the representation of $f(x)$). Therefore, $\bar{x}, \bar{y}$ refer to finite sequences of dyadics.[6]

- We first tell the TM (as input) an integer n (context-dependent), such that the computed output $\bar{y}$ must be within the error 2^{-n}; that is, $|f(x) - \bar{y}| \leqslant 2^{-n}$.
- The machine M refers to the oracle to obtain $\bar{x} = \varphi(m)$ such that $|x - \bar{x}| = |x - \varphi(m)| \leqslant 2^{-m}$. We allow unlimited questions from M to the oracle, and the parameter m may depend on the answers to these questions.
- On input $\bar{x}$, M writes $\bar{y}$ on its output tape, after finitely many steps.

We will say that a function $f :\subseteq \mathbb{R} \to \mathbb{R}$ is TTE-computable (or CA-computable) if there is an oracle Turing machine that can compute it in the way described above. Generalization to multivariate functions $f :\subseteq \mathbb{R}^k \to \mathbb{R}$ is straightforward, by allowing k oracles, $\varphi_1(m), \varphi_2(m), \ldots, \varphi_k(m)$ that can be queried with respect to m.

On the complexity side, the cost (running time) of M is the worst-case cost that a computation with M can take given a valid input and precision n.[7]

[6]For example, on an alphabet containing at least $\{0,1,\#\}$, if x is $\frac{1}{3}$, $\bar{x}$ could be a finite string of the form:

$$\#b(\frac{1}{2})\#b(\frac{2}{4})\#b(\frac{3}{8})\#b(\frac{6}{16})\#\ldots$$

where $b(k)$ means the binary expansion of k.

[7]The above model is sometimes modified as follows. We assume that M has access, via its input tape, to a rational sequence that rapidly converges to x. Furthermore, we require from the machine to produce not only one rational approximation to $f(x)$ with some precision n, but approximations for every $n \in \mathbb{N}$; that is,

2.2 The BSS Model

The BSS model is a model of computation not exclusively over continuous spaces, but over any arbitrary field or ring R. If R is $\mathbb{Z}_2 =< \{0,1\},+,\times >$, then the model becomes reduced to classical computability theory. The standard reference is Blum et al. (1997), but briefer expositions can also be found in Blum (2004), Smale (1990), and Cucker (1999).

The model is based on the notion of a *machine M over R* (where R is a commutative, possibly ordered, ring or field). A finite-dimensional machine has an *input* and *output* space associated with it, a two-way *infinite* tape with cells, and, similarly to Turing machines, a *read-write* head. The machine's *program* is a finite directed graph with five types of nodes, linked by operations or *next node* mappings.

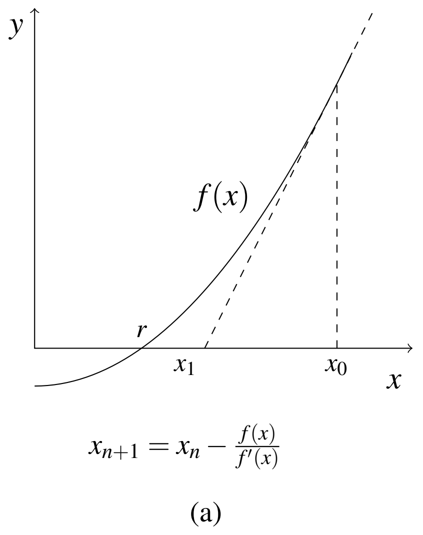
More specifically, the top node represents the input stage and the last node the output stage. At each other stage, passage from a node to another is made either by means of a rational computation or by means of a comparison step whose branches may lead to a new node or a previous one. The last kind of node is a *shift node*, associated with shifting the head one cell to the right or to the left. The machine computes a function $f :\subseteq R^k \to R^l$, if on an input $x \in dom(f)$ it outputs $f(x) \in R^l$. The inputs can be thought of as strings of elements of R of the form: $...000x_1x_2x_3..x_k000...$ The machine decides a set $A \subset R^k$, if it computes a characteristic function for it; for example, $X_A(x) = 1$, if $x \in A$, and $X_A(x) = 0$ otherwise. For most cases in numerical analysis, the space R can be thought of as either $\mathbb{R}$ or $\mathbb{C}$. See fig.1 for an example.

The model can additionally be extended to infinite-dimensional cases for computing functions $f :\subseteq R^\infty \to R^\infty$, where the underlying structures $R^\infty := \bigcup_{n\in\mathbb{N}^*} R^n$. In this case, the two-way infinite tape allows for inputs that are sequences of elements of R of arbitrary length. For the purposes of this paper, it is adequate to restrict our focus to the finite-dimensional case. In doing so, we can equivalently think of BSS-machines as a generalisation of Random Access Machines from the natural numbers to real numbers (whence comes the often used, alternative name 'Real-RAM' for the same model).

Within this framework, the authors prove several results about the decidability and complexity of sets and problems over $\mathbb{R}$ and $\mathbb{C}$ (e.g., that the Mandelbrot set is undecidable over $\mathbb{R}$). But, as mentioned already, a crucial idealisation that Blum et al. employ is that the machine M is able to manipulate the *exact value* of any real number it is operating on. Real numbers are viewed as unanalyzed entities in the algebra R, and algebraic operations and comparisons are each counted as one unit of work; for instance, if $R = \mathbb{R}$, it takes one step to add, subtract, multiply, divide, and compare any two real numbers, even irrationals.

Regarding complexity considerations, the size $S(x)$ of an input $x \in R^k$ is $S(x) = k$. The

to produce a rational sequence that rapidly converges to $f(x)$. We then get an extensionally equivalent model, which dispenses with the extra input n. This modified model is in effect the one considered in Weihrauch (2000). Note that in this modification, it is necessary that the output tape is one-way, so that although a computation may need an infinite number of steps to be completed, we are nevertheless assured that after a finite number of steps we have always available a correct first approximation of the output (hence the equivalence with the oracle formulation).



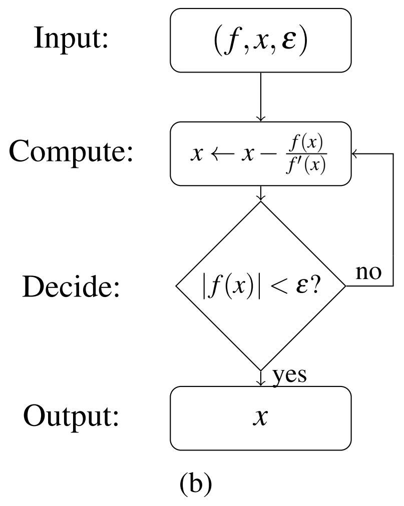


Figure 1: (a) Newton's method for approximating a root of a real polynomial $f(x)$. (b) A simplified version of a BSS program for implementing the method (next and shift nodes are omitted). Input and output spaces are both $\mathbb{R}$. The set $A \subset \mathbb{R}$ whose elements are those x for which the machine halts after a number of steps, yielding an x_i such that $|f(x)| < \varepsilon$, is often called the 'Halting set' of the machine. Crucially, comparisons and arithmetical operations are assumed computable in this model (in one step).

computational cost (the running time of the program) is defined as the number of operations, comparisons, and movements —that is, as the number of nodes traversed from input to output when the machine starts with input x. The cost is expressed as a function of the input size, k, analogously to ordinary complexity theory, thus allowing for the definition of corresponding complexity classes for real (or complex) number problems, such as the classes $\mathbf{P}_R$ and $\mathbf{NP}_R$, when $R = \mathbb{R}, \mathbb{C}, \mathbb{Z}_2$ (or any other field or ring). If $R = \mathbb{Z}_2$ (the ring of two elements), the input has the form of a string consisting of n digits, and its size $S(x)$ is the bit size of the string. In this case, the computational cost reduces to the bit cost, and the theory becomes equivalent to the classical complexity theory, which is based on Turing machines (e.g., we recover the classical complexity classes: $\mathbf{P}_{\mathbb{Z}_2} = \mathbf{P}$, and $\mathbf{NP}_{\mathbb{Z}_2} = \mathbf{NP}$).

2.3 Incompatibility between the two main approaches

There is a precise sense in which the two main approaches are incompatible. Almost all of the functions characterised as computable by BSS are non-computable by CA, and vice-versa (see, also, Weihrauch 2000, ch.9).

More precisely, a major result in CA is the following:

Every computable function must be continuous.

In other words, discontinuous functions are not computable (at least, near the discontinuity points). This result has a proof under any particular formalisation of computable analysis (be it in terms of Type-2 Turing machines, or recursive functions, or the topology of the computed spaces, etc.), and it was even pointed out by Borel (1912) in his informal discussion of computable functions.[8] Intuitively, it holds because if we're computing an approximation to a value $f(x)$, we want this approximation to be a good one for all points near x, since the latter is also given by an approximation.

A consequence of this is that the following relations are all non CA-computable:

$$\{(x,y) \in \mathbb{R}^2 | x = y\},\ \{(x,y) \in \mathbb{R}^2 | x < y\},\ \{(x,y) \in \mathbb{R}^2 | x \leqslant y\}$$

The equality relation, for example, is not computable, intuitively, because a TM with two equal real numbers on its input tape(s) would not be able to decide within any finite number of steps —i.e., after having read only a finite prefix of the inputs— that the two numbers are equal and do not differ at some later digit, not yet reached by the machine's head.[9] Equivalently, the corresponding function is not continuous. But if the two numbers *are* different, a decision is possible. Differently put, the equality relation is (negatively) semi-decidable (computably enumerable).[10] Similarly, the total order relation is not generally computable; however, it can be decided for two non-equal numbers. Formally, these results are proved either by showing that the decidability of the above relations imply computability of the halting problem, or by topological reasoning based on the discontinuity of the corresponding functions; for the latter, see, e.g., Weihrauch (2000) and Brattka et al. (2008).[11]

In the BSS/Real RAM approach, on the other hand, there's no similar continuity requirement. That means that functions seemingly easy to define, such as step or floor/ceiling functions, although not CA-computable, are BSS-computable.

Furthermore, since every non-trivial branching node in the program of a BSS machine introduces a point of discontinuity,[12] most BSS-computable functions are not CA-computable (Weihrauch, 2000). Conversely, functions such as square roots or exponentials are not BSS-computable, whereas they're unproblematic in CA.[13] This is intuitively the case because

[8]"A function cannot be computable, if it is not continuous at all computable values of the variable." (Quoted in Avigad and Brattka 2014)

[9]Two implicit assumptions here are that the same naming system is used for both inputs, and that the output tape is one-way.

[10]This too was informally pointed out by Borel (1912): "The first problem in the theory of computable numbers is the problem of equality of two such numbers. If two computable numbers are unequal, this can obviously be noticed by computing both with sufficient precision, but in general it will not be known *a priori*." (Quoted in Avigad and Brattka 2014)

[11]If instead of the entire $\mathbb{R}$, only the computable reals $\mathbb{R}_C$ are considered, then the undecidability of equality comes as a consequence of Rice's theorem.

[12]Recall, for example, how the BSS-decidability of a set was defined in sec. 2.2, based on computing its discontinuous characteristic function.

[13]Almost all commonly encountered continuous functions in analysis are CA-computable on an appropriate domain, including, for example, polynomials, trigonometric functions, $\frac{1}{x}$, e^x, $\log x$, $\sqrt[n]{x}$, $|x|$, etc., and compositions thereof.

generally all intermediate computations performed by a BSS machine are rational functions of the inputs and built-in constants; that is, algebraic fractions with polynomials as numerators and denominators.[14] Of course, such functions can often be *approximated* by BSS-computable functions; for example, there is some BSS-computable function $f :\subseteq \mathbb{R}^2 \to \mathbb{R}$, such that $|f(x,\varepsilon) - \sqrt{x}| < \varepsilon$ for any $x, \varepsilon > 0$.[15]

All this creates a strong incompatibility between the two main approaches. How can we go about choosing one over the other? Since we are talking about models that in a sense purport to capture mathematical facts, one attempt might be to test both models against some basic intuitions; that is, intuitions regarding what operations and functions should be computable or not, and with what difficulty.

3 Intuitions

Whenever we are to choose a model of computation for modelling some particular domain of computations, we have certain pre-theoretic intuitions that we want our chosen model to satisfy. What are those intuitions in our case?

A first attempt might be that a theory of computability about a mathematical area should try to characterise as 'computable' at least the fundamental operations used in the area (Gherardi, 2008). The four basic arithmetical operations $(\pm, \times, \div)$ are generally computable in both the BBS and computable analysis (CA) approaches. However, n-th-roots are not BSS-computable, as we saw, whereas they are unproblematic for CA. On the other hand, although comparisons $(>, <, \geq, \leq)$ and identity are not computable in CA, they are defined as computable in BSS, something that seems desirable for any theory which regiments numerical algorithms. This is because comparisons between real numbers, as elements of an ordered set, are taken as basic and are built into many such algorithms (consider, as an example, a classic bisection, root-finding method).

A further factor of consideration is what functions each model characterises as computable. We have already pointed out that only continuous functions are CA-computable. This is an inevitable consequence, necessary (actually *constitutive*) of *any* framework in which computation is understood as an effective process over uncountable domains —that is, it can in principle be simulated by pen and paper. Of course the uncomputability of the equality relation stems from the same conceptual reasons.[16] This is however an undesir-

[14]For a formal proof of the non BSS-computability of exponentials, see, e.g., Brattka (2000, 8). See also Weihrauch (2000, 265) for the sketch of a similar proof. See Calvert et al. (2011), for various results of uncomputability in BSS, including transcendental and algebraic functions.

[15]For example, by using a machine implementing Newton's method (fig.1), to approximate the zero of the function $f(x) = x - a^2$, and obtain $\sqrt{x} \approx a$. Nevertheless, such approximate algorithms may turn out problematic in case of compositions of functions, with a known desired output precision (e.g., $f(x) = \sqrt{5 + \sqrt{x}}$, with precision $\varepsilon > 0$), since we need to be able to find the required precision of the input based on ε (sec. 2.1.1). This is more straightforward in bit-computation (see Braverman and Cook 2006, 322 for an example) than in BSS.

[16]Recall that Borel had arrived at the same conclusions mainly by informal reasoning (see, fn.8,10).

able feature for the purposes of analysing numerical algorithms, full of comparisons, and so it is avoided in BSS by fiat. Furthermore, as said already, transcendental functions — such as exponential, logarithmic, and trigonometric ones— that cannot be locally expressible as algebraic functions are not BSS-computable. Whenever a particular problem involves such functions as parts of the investigated algorithms, their computability and fixed cost are stipulated. Again, this might be problematic if someone is interested in capturing what is effectively (un)computable in principle, by an idealised agent (machine or otherwise) following a mechanical method; yet, it is fruitful for analysing algorithms. Another difference is that any constant function $f(x) = c$ ($c \in \mathbb{R}$) is BSS-computable, since polynomials are BSS-computable and a constant is trivially a polynomial (so just ignore the input x and output c). This is conceptually suspicious from the point of view of effective computation, since all real numbers but denumerably many are not computable by a Turing machine (there are denumerably many TMs but uncountably many real numbers). Furthermore, it is possible to construct a particular real number c such that computing it with arbitrary precision would entail solving the Halting problem, which is accepted to be uncomputable by both traditions.

Another proposed criterion for evaluating a theory of computability might be how well it matches our intuitive notions of 'easy', 'hard', and 'impossible' computation (Braverman, 2005). The usual arithmetic operations, for instance, should be characterised as 'easy' by any model. Arguably, the same goes for every function found in a common calculator, such as $\sin x$. Also, problems that are known to be easy, hard, or impossible, in the discrete case might be expected to fall under the same categories in the continuous case. For example, the Travelling Salesman Problem, which is NP-complete in the discrete case, or other known hard problems, such as the N-body problem or the Navier-Stokes equations, are expected to be considered hard in the continuous case too (Braverman and Cook 2006; however, see the forward by R. Karp in Blum et al. 1997). The fact that, for BSS, $f(x) = c$ is always computable for any $c \in \mathbb{R}$ in one step goes *prima facie* against the "criterion" we are discussing here. In order for a machine to compute $f(x) = c$, for some $c \in \mathbb{R}$, it is enough to ignore the input x, and start computing approximations to c as described for example in Turing (1936). Then, we would expect intuitively that computing (say) π would be more difficult than computing (say) $\frac{1}{4}$. And, as said before, there are numbers $c \in \mathbb{R}$ that are even impossible to compute. However, BSS does not distinguish between these cases, categorising any function of the form $f(x) = c$ as 'easily computable' (one-step operation), since a real number can always be stored in a register, being just an element from $\mathbb{R}$.

Finally, there is another intuitive aspect that is captured quite differently by the two approaches. The cost of a computational method is expressed in computer science and mathematics as a function of the input size. In Turing machine models, including CA, the input size is measured in bits (word size). This means that the total costs of particular operations, and fully fledged algorithms, are dependent on the bit size of the input(s) size. Nevertheless, the way modern-day machines implement algorithms does not involve any corresponding notion of varying size of a real number (see appendix). So, for the purposes of comparing different available algorithms for the same problem, and given that they are meant to be machine-implemented, it may be intuitively natural to focus the attention on the number of operations and comparisons per se, as the dominant parameters of the total cost; at least in those cases where the considered problem is not too sensitive to the initial data (well-conditioned problem).[17] To make the point clearer, let us consider a well-conditioned problem.[18] A slight perturbation of the value of the input, say, from 1 to $1 + (\frac{1}{2})^n$, would cause the input bit size to grow from 1 to n+1 (Blum, 2004). For large n, this can lead the analysis astray, because although the input size would change considerably, due to the small perturbation, the actual complexity class should not change if the problem is well-conditioned. Thus, within the CA framework we run a risk of getting misled, if such circumstances are not taken into account when evaluating costs of algorithms and complexity classes of problems.

4 Towards a foundational framework

The point of the previous discussion is that neither of the two approaches satisfies *all* of our pre-theoretic expectations about a computational model over the reals. Nevertheless, both frameworks have been fruitful in terms of their results. Bit-computation is a natural extension of the original Turing model, and is grounded in plausible and intuitive assumptions, such as those of Turing machines *approximating* the exact values of the computable reals by writing/erasing symbols from a finite alphabet, etc. Thus, we think that CA appropriately regiments the pre-theoretic idea of 'effective computation', as applies to continuous spaces.[19,20] In a sense, this fact, by itself, anticipates the fecundity of this model. On the other hand, the successful applicability of BSS to more than one mathematical area —despite the fact that it is not a model concerned with explicating 'effective computation' per se, and given that it is grounded in assumptions and idealisations that have been criticised as unnatural and unrealistic[21]— begs for a philosophical explanation. In fact, besides numerical analysis,

[17]Although the motivation here is based on the properties of modern-day FPA implementations, interestingly, Turing himself had also regarded as useful a similar cost measure, based on the number of operations. The following quotation is from (1948) —the same paper where he also introduces the notion of *condition number* (long before FPA was implemented in actual computers).

> 'It is convenient to have a measure of the amount of work involved in a computing process, even though a very crude one [..] We might, for instance, count the number of additions, subtractions, multiplications, divisions, ..." (Turing, 1948, 288).

'

[18]We explain what that means in sec.5.

[19]By 'effective computation' we mean computation performed by an idealised agent (machine or otherwise), following a systematic method and having unlimited space and time. Such a computation should be in principle simulable by pen-and-paper.

[20]We do not offer further arguments for *that* claim here, besides ascertaining that it is a generalised version of classical Turing computability, which admittedly regiments 'effectivity'. We refer, however, the unconvinced reader to the discussion in Pégny (2016) for more conceptual arguments to that effect.

[21]See, for example, Weihrauch (2000, ch.9), for a standard criticism.

BSS and similar models[22] are also helpfully applied to computational geometry, information-based complexity, and algebraic complexity. These all are fields concerned with algorithms, and a good number of practitioners in them completely ignore foundational approaches to algorithms in terms of Turing machine models, such as CA. Wozniakowski's comment is characteristic:[23]

> Before we go on, we pause for a moment and ask why so many people are using the real number model, and why there are so few complaints about this model. In fact, with a little exaggeration, one can say that only some theoreticians are unhappy with the real number model, whereas many practitioners do not experience any problem with it. (Woźniakowski, 1999, 452)

What makes BSS –and the other similar models– so applicable? The question is even more germane, if one accepts our interpretation that these models are not concerned with regimenting the notion of 'effective computation' per se. We propose an account in the following sections, but the argument involves two stages. The first is to distinguish different possibilities about what the scope and meaning of 'foundations for scientific computing' can be, and the second is to examine how each tradition responds to each such possibility. But first, we take a look at what the mathematicians behind BSS themselves take their goal to be. We think that interesting insights can be gained by this.

4.1 BSS and similar models as idealised representations

A stated goal of this model is to offer a foundation of scientific computing, by examining the notion of 'computation' itself. Blum et al. (1997, 22) write: "There is a sense in which this work is a study of the laws of computation". And they continue:

> [W]e write not from the point of view of the engineer who looks for a good algorithm [..] The perspective is more like that of a physicist, trying to understand the laws of scientific computation. Idealizations are appropriate, but such *idealizations should carry basic truths*. (Blum et al. 1997, 22; emphasis added)

Now, as this paper aims to point out, this foundational pursuit, besides presenting an interest on its own, also provides a distinct example of employing idealisations to mathematically describe a domain which is in fact mathematical, viz. the "laws of computation".[24] A

[22]Other similar models are the *Real-RAM* and *Real Number* model. In fact, all these three models are practically equivalent, and the difference is mostly terminological. The former term is typically found in the computational geometry literature, whereas the latter typically in information-based complexity.

[23]Wozniakowski, coming from the area of information-based complexity, uses the term 'real number model'.

[24]One could retort here that ordinary complexity theory is grounded in idealisations as well, apparent in the asymptotic formalism describing costs and complexity classes. I think, however, that this would be a mistaken take on this practice, and that asymptotic results should rather be understood as *approximations*. There is a conceptual difference between approximations and idealisations, but a discussion of it would be beyond the scope of this paper. For illuminating clarifications, one can see Norton (2012).

BSS-machine, for example, is assumed able to store and manipulate exact real values. But how can we know that such an idealisation does carry some of the "basic truths" that Blum et al. (1997) are asking for? After all, most of the criticism against this model, as already discussed, has focused on this very assumption. Interestingly, even the authors themselves admit some hesitation in adopting such an idealisation. We ask for the reader's indulgence in quoting such a lengthy excerpt as the following, but we think that many interesting points for the upcoming discussion are contained in it.

> [W]e are led to expanding the theoretical model of the machine to allow real numbers as inputs. *There has been great hesitation to do this* because of the digital nature of the computer. Here we might learn a lesson from the history of science. At the time of Newton, scientists assumed that the world was atomistic [..] Newton accepted that picture according to which all matter is composed of indivisible particles, a finite number in each bounded region. On the other hand, Newton's mathematics was continuous as was Euclid's. [..] It was a substantial problem for Newton to reconcile the discrete world with the continuous mathematics. The resolution was produced by analyzing the effect of replacing an object (e.g., the earth) by a finite number of particles, then making a better approximation with a larger number of particles. In the limit, the mathematics becomes continuous. [..]
>
> Now our suggestion is that *the modern digital computer could be idealized* in the same way that Newton idealized his discrete universe. The *machine numbers* are rational numbers, finite in number, but they fill up a bounded set of real numbers (e.g., between -1000 and 1000) sufficiently densely that *viewing the computer as manipulating real numbers is a reasonable idealization*, at least in a number of contexts.
>
> Moreover, if one regards *computer-graphical output* [..] and asks to *describe the machine that made these pictures*, one is driven to the idealization of machines that work on real or complex numbers in order to give a coherent explanation of these pictures. For a wide variety of scientific computations the continuous mathematics that the machine is simulating is the correct vehicle for analyzing the operation of the machine itself.
>
> These reasonings give some justification for taking as a model for scientific computation a machine model that accepts real numbers as inputs. Of course a great many issues such as round-off error must be dealt with. Moreover the ultimate justification is: does the model developed this way give new insights and understanding to the use of the big machines? To attempt to show this is a major goal of our work. (Blum et al. 1997, 23-24; emphasis added)

From the above it seems that the authors themselves consider two modes of justification for the idealisations underlying their approach. One is based on theoretical reasoning focused on the target (modelled) domain; in particular, properties of the actual ("modern digital") computers that perform scientific computations. The other is grounded in pragmatic considerations; mainly, the ultimate fruitfulness of the model at hand, in terms of the understanding and results gained. Since the latter is rather an internal mathematical criterion, we believe that the verdict on its fulfilment is up to the mathematical community itself. Here, our concern is with the former criterion, which is little discussed by the authors themselves; that is, *to what extent, and by virtue of what, the chosen idealisations are appropriate for the domain at hand.* This is important because it pertains to the fundamental issue of how strictly false models can provide reliable results and capture faithfully aspects of the domain under study. But first we need to clarify *what the domain exactly is* in the case of scientific computing. We discuss this in the following section, 4.2.

A second interesting issue, arising from the above quotation, is the analogy the authors draw with empirical science and the use of idealised representations therein. The authors keep the analogy minimal; in effect, the only similarity they point out between the two cases is the usefulness of idealisations. However, I submit that the analogy can go further. This is the topic of section 7.

4.2 What is even meant by 'foundations for scientific computing'?

What does it mean to develop foundations for scientific computing? I suggest that there are two ways of understanding this goal. One way is to aim at a framework within which we can prove fundamental results about what can ultimately and in principle be computed by a computer, machine, or idealised agent, following an *effective* procedure and given enough space and time. The tradition of computable analysis, based on the groundwork laid by Turing's (1936) work, and continued in the works of Lacombe (1955), Mazur (1963), Grzegorczyk (1955), Pour-El and Richards (1989), and Weihrauch (2000), is in accordance with such a goal. Then, to the extent that such a framework is so fundamental as to be an accepted explication of the informal notion of 'effective computation', any results proven within it about the (un)computability of certain problems and functions would be transferred —via some kind of extended *Uncountable* Church-Turing thesis[25]— to what digital conventional computers (and idealised agents) can do as well.[26] This would definitely be considered as a foundation of computing then, and would exhibit a degree of robustness and implementation-invariance analogous to the one enjoyed by classical computability and complexity theories, which are also model/implementation-independent.[27] And *a fortiori* it would provide a foundational framework for the narrower domain of application that is referred to as 'scientific computing' by the modern-day mathematicians, scientists, and engineers.

[25]Similarly to the classical case of discrete computability theory.

[26]We ignore conceptual complications arising from the possibility of hypercomputers, or other non-conventional forms of computing here.

[27]Their reliance on Turing machine formalisms is presumably harmless, for a tacit assumption is the validity of the Church-Turing thesis.

On the other hand, another possible route —another interpretation of the discussed goal, that is— would be to pay special attention to what modern-day practitioners mean by 'scientific computing'. We will not attempt a complete (or even incomplete!) definition of the term here but some general characterisation. To this end, it may be useful to look at David Keyes's entry 'Computational science' in *The Princeton Companion to Applied Mathematics* (Higham et al., 2015). In the 'Definitions' section of the entry, Keyes writes:

> [S]cientific computing is the study of techniques in the intersection of many discipline-specific fields of computational science. Computational chemistry, computational physics, computational biology, computational finance, and all of the flavors of computational engineering (chemical, civil, electrical, mechanical, etc.) depend on a common set of techniques connecting the conceptualization of a system *to its realization on a computational platform*. These techniques—such as representing continuous governing equations with a discrete set of basis functions, *encoding this representation for digital hardware*, integrating or solving the discrete system, *estimating the error in the result, adapting the computational approximation,* [..]— span diverse applications from science and engineering and are the elements of scientific computing. (336; emphasis added)

By 'computational science', in the above quote, Keyes means "the systematic study of the structure and behavior of natural and human-engineered systems accomplished by computational means" (335). The interesting aspect here is that under this understanding of the term, the fact that any computing technique is to be represented in and performed by digital computers is taken into account from the outset, during the development stage (see the emphasised parts).[28] Thus, this second interpretation of the scope of 'foundations for scientific computing' concerns developing a practical framework within which we are able to mathematically analyse and compare computational methods, in advance, and *under the (tacit) assumption* that they are meant to be implemented by modern-day digital machines.[29] The consequence of this is that definitions and other means of regimenting informal notions, within this framework, need be responsible to the actual capacities and limitations of real computers. This is not to say that any such foundational framework should be built upon assumptions about very specific features of computer design and architecture, such as, say, whether a binary or a ternary number representation is implemented by the machine, or whether single or double precision FPA is used.[30] Although even the mere consideration of the programming language influences the exact form of a computation, any such model would be so specific and cumbersome as to be practically useless and easily outdated. Nevertheless, a degree of attention

[28]As a historical aside, it seems that the seminal articles initiating the field of scientific computing, by means of introducing thorough discussions of computer errors, were von Neumann and Goldstine (1947), and Turing (1948).

[29]Such a framework would also need to provide an appropriate formal definition of 'algorithm', for in order to prove *lower bounds* of computational costs (and thereby characterise complexity classes), we need to generalise over all possible algorithms.

[30]Thanks to an anonymous referee for pressing on this point.

to *some* essential features (e.g., that computers use rational numbers of fixed bit size, and thereby introduce representational and round-off errors) is necessary, for if such features are not taken into account they can render the theoretical model far too idealised with respect to the target domain. Of course, as in any case of modelling, what to consider and what to leave out is anything but a trivial matter.

Importantly, arguments that seem to imply this second understanding of the search for foundations are found in papers from both traditions of computation. For example, the long quotation by Blum et al. (1997) (sec. 4.1) shows that the authors do consider properties and capabilities of modern computers as a yardstick by which to evaluate their model's fecundity. Additionally, one can also see Blum (2004), Cucker (1999), Novak (1995), Smale (1997), and other papers of the same tradition for discussions —and extensions of the BSS model— concerned with treatment of round-off errors.[31] Additionally, although computable analysis, in the spirit of works cited in the beginning of this section, has not been concerned with implementational issues,[32] the formulation of this approach as an additional candidate for a foundational framework for scientific computing in Braverman and Cook (2006) likewise involves arguments that regard modern computational practice as a yardstick of success or failure.[33] For example: "A weakness of the BSS approach as a model of scientific computing is that uncomputability results do not *correspond to computing practice* in the case $R = \mathbb{R}$." (319; added emphasis). And later:

> In the bit model there is a nice definition of decidability (bit-computability) for bounded subsets [.. T]he idea is that the set is bit-computable *if some computer program can draw it on a computer screen*. [..]
>
> [..] In the rest of the paper we concentrate on the bit model, because we believe that *this is the most accurate abstraction of how computers are used* to solve problems over the reals. (319-320; emphasis added)

So, assuming our second (practice-oriented) interpretation of what developing foundations for scientific computing amounts to, what kinds of computations performed by modern-day computers should be considered and modelled? First, we need to distinguish between two different ways in which computers are employed in scientific computing. The first has to do with *symbolic computations*. The mathematical underpinning of this practice is *computer algebra*; an area of research concerned with the development, analysis, and implementation of *algebraic algorithms*. The second major use of computers, within scientific computing, is *numerical computations*, and the underlying mathematical area is *numerical analysis*; concerned with the development of *numerical algorithms* (e.g., the Newton method illustrated in

[31] One can also see the concern with implementational issues reflected in how the research in the fields of numerical analysis, computational geometry, and information-based complexity have changed through the last decades and become much more implementation-focused. Textbook presentations make this turn apparent too.

[32] For example, there is no discussion of representation or round-off errors in Weihrauch (2000).

[33] We are always referring to this particular formulation whenever we use the term 'bit-computability' in this paper.

fig.1). The former area studies computations with symbols representing mathematical concepts, such as polynomials, algebraic numbers, trigonometric functions, integrals, and so on. In contrast, the latter area studies numerical calculations focused, for example, on solving (sytems of) equations, data interpolation, optimisation problems etc.[34] Algebraic algorithms generally receive as their inputs formulas or symbolic expressions of numbers, and the same holds for their outputs. Therefore, their outputs are not approximations of numbers, and their computations –in contrast with numerical computations– are exact and not subject to errors. Put differently, a symbolic algebra system manipulates mathematical expressions analytically, in a similar fashion to how a student would solve a calculus problem by pen and paper. To give an example of the differences between the two kinds of computing, suppose we are solving a system of two equations. Whereas the output of a numerical method might be, say, $(-0.5773502...,0.25)$, the output of the corresponding symbolic computation would yield $(-\frac{\sqrt{3}}{3},\frac{1}{4})$. As another toy example, a symbolic computation of $\frac{\pi}{4}+\frac{\pi}{6}$ would return $\frac{5\pi}{12}$ as output, while a numeric computation would return 1.3090...

With this distinction in mind, we can return to the last question. It seems safe to assume that the quest for foundations of scientific computing, at least in the way pursued by both traditions and advocated in the two papers of Blum (2004) and Braverman and Cook (2006), primarily focuses on numerical computations. This should not be a surprising claim. From the outset, the foundational study of computation has been concerned with computability of numbers and functions. Borel (1912) and Turing (1936) (and others later) were concerned with computable real numbers and functions; Church (1936), Gödel, Herbrand and Kleene were concerned with number-theoretic (partial) recursive functions. Both traditions discussed here are concerned with computation of real numbers and functions as well. For that reason, we will also restrict our attention, for the rest of the paper, to *numerical* computations.

To conclude the (rather long) discussion of this and the previous section: the way we interpret the goal of developing foundations for scientific computing matters, for it ultimately determines what the target domain of our models is. Accordingly, identifying the target domain bears on which properties of that domain have to be considered in our formal model and which can safely be abstracted away and idealised. But, as we have already said, this is not always a straightforward task (in the apt words of F. Cucker (1999, 106), "models of computation are not sold in hardware stores").

We have put forward two possible interpretations of this goal. For the purposes of the next section, which are to formulate an account of the success of BSS, we will assume the second proposed interpretation: that is, any foundational framework should be answerable to actual

[34]The distinction is in reality not as clear-cut as presented here. For example, computer algebra involves representations of numbers as well. Perhaps, the most broadly known computer algebra systems today are MATHEMATICA and MAPLE, whereas MATLAB is perhaps the most well-known environment for numerical calculations. Of course, many more such systems exist, and recently many of them are enhanced with packages that allow them to cross boundaries and perform both kinds of computations (MATLAB being a recent case in point.)

practice and have as an implicit assumption the intended machine-implementability of the developed algorithms. Within this context then, what properties of the target system should be considered? Perhaps the most important such property is that actual computers store and work with finite representations of real numbers, which are of *fixed size*; namely, floating-point numbers (cf., also, Blum et al.'s quotation, on p.15).[35] This property does influence the way mathematicians design *numerical* algorithms, and bears on the actual costs of machine computations. Another property is that the majority of problems in the actual practice of scientific computing cannot be solved exactly, and the developed numerical algorithms are such that they approximate the exact solutions up to some predetermined accuracy, $\varepsilon > 0$, by means of iterating certain steps (cf., Newton's method, fig.1).

In the next section (5), we delve into how all these features are captured or neglected in BSS and bit-computation models.

We note in passing that we do not attempt in this paper a definite answer to the question of what the correct interpretation of the goal of scientific computing is. Rather, we assess the appropriateness of each model *conditionally* on what the goals may be taken to be by the mathematical community itself.

5 Why BSS turns out to be so successful as a model of machine-computations

BSS (or Real-RAM) analyses computations between values that are assumed (i) real, (ii) exact, (iii) able to be added, subtracted, multiplied, divided, and compared at *unit cost*.[36] Nonetheless, given that this and similar models are actually successful, this might seem to fly in the face of the fact that today's digital computers, almost universally, implement numerical computations with floating-point arithmetic (FPA). This is because floating-point computations are between values that are (i) rational, (ii) rounded and of finite precision, (iii) added, subtracted, multiplied, divided, and compared at some fixed cost.[37]

For comparison, we also mention that bit-computations (within CA) are between values that are (i) rational (ii) of arbitrary precision (iii) added, subtracted, multiplied, and divided (but not compared) at a cost that is dependent on the size of the operands and the desired precision. We summarise these features in table 1.

In what follows, we will be examining all these three features separately, in order to understand under what circumstances BSS is reliable, despite seemingly unrealistic idealisations.

The first difference between BSS and FPA —i.e., that the computed numbers are real in the

[35]Symbolic algorithms, on the other hand, are not implemented in floating-point arithmetic.

[36]Hereafter, we consider BSS computations over $R = \mathbb{R}$.

[37]This may seem even more surprising, given some of this arithmetic's bizarre properties, mentioned in the appendix.

Model	**Values**	**Precision**	**Operations and costs**
BSS	Real	Infinite, exact	Addition, subtraction, multiplication, division, comparison, at *unit* cost
FPA	Rational	Finite, limited	Addition, subtraction, multiplication, division, comparison, at *fixed* cost
Bit-comp.	Rational	Finite, arbitrary	Addition, subtraction, multiplication, division (no comparison), at cost *dependent on the size of the operands*

Table 1: Comparison between the two models and floating-point arithmetic

former, though rational in the latter— can be theoretically "alleviated" as follows. Floating-point numbers are most commonly defined as integers multiplied by positive or negative powers of 2. This means that all FP numbers represented this way are dyadic rationals; that is, elements of $\mathbb{D} = \{\frac{a}{2^b} : a \in \mathbb{Z}, b \in \mathbb{N}\}$. The set $\mathbb{D}$ enjoys some properties that make it very convenient for computational models. It is closed under addition, subtraction and multiplication. It is also dense in $\mathbb{R}$; that is, for any real number r, any neighbourhood of r contains at least one dyadic rational in it. Furthermore, any real r can be approximated by a dyadic of the form $\lfloor 2^k x \rfloor / 2^k$ with arbitrary precision. Finally, the dyadic rationals are precisely those numbers whose binary expansion is finite. Interestingly, some operations between dyadics can be performed exactly, even in FPA, under certain circumstances (e.g., when there's no underflow). For example, dyadic rationals can be added or divided by two without round-off, a property that can make a bisection algorithm have much fewer round-off errors (Woźniakowski, 1999). Of course, some formalisations of bit-computation models also employ dyadic rationals to represent reals (see, e.g., Braverman 2005), and so they exploit the merits of this system as well.

The second contrast between BSS and FPA —namely, the precision of the numbers being computed upon— is the one that makes the success of BSS in modelling FPA most remarkable. In the course of an FPA computation, there are representation and round-off errors that may accumulate. Nevertheless, most researchers in scientific computing do not experience significant difference between the two models in most (but not all) cases. Can we identify reasons that account for the fact that BSS can reliably model FPA computations, despite that it abstracts away from representation and round-off errors and idealises FP numbers as reals? The discussions in Woźniakowski (1999) and Cucker (1999) can be helpful for the question at hand.[38]

[38]Woźniakowski's discussion is about the use of the Real Number model in information-based complexity, but his results bear on the context of our discussion as well. We heavily draw upon his discussion, and Cucker's

Woźniakowski posits two conditions that should be met in order for BSS/Real-RAM to be a reliable model with respect to real-world computations. The first is that the machine-algorithm we intend to model via BSS is a *stable* one. What is meant by that? More often than not, we do not have access to exact data, but only to approximations within some error bounds; often due to limitations on measurement precision. Furthermore, just by inputting the initial data we typically introduce additional representation errors. If we assume, after this point, exact computations without any round-off errors, then that means that we can consider the output of the computation as an exact result (since there are no in-between introduced errors) of a slightly perturbed problem (since only the input was slightly changed). However, in reality, FPA computations introduce intermediate round-offs as well. Intuitively then, the stability assumption is that the (FPA-)algorithm, overall, does not magnify the initial small errors. Put differently, a stable algorithm is one that "returns results that are about as accurate as the problem and the resources available allow" (Corless and Fillion, 2013, 29). This idea is expressed schematically in fig.2. A precise definition of "stability", though, depends on the specific problem under consideration.

Now, assuming stable FPA algorithms, we can reliably use BSS to analyse their costs and obtain lower bounds for characterising complexity classes.[39] This is mainly because of two reasons. First, a lower bound, obtained by BSS for idealised machines with no round-offs, would be a lower bound for a machine with round-offs as well. Finding in BSS that a problem cannot be solved with less than, e.g., $\Omega(n^2)$ operations in BSS, allows us to safely assume that in the presence of round-offs there would not exist an algorithm with cost, say, $O(n)$.[40] The second reason is that in both BSS and FPA the cost of a particular operation does not depend on the bit size of operands. We return to this point at the end of this section.

We said that our stable algorithm solves (almost exactly) a slightly perturbed problem; that is, a problem with slightly perturbed initial data. We need in general to be able to measure how much the result of the output is affected by the perturbations of the input. This sensitivity of the solution to small changes in the data is called the 'conditioning' of the problem and a measure of that is the *condition number*, $\mu(x)$, of the particular problem. A problem that is not too sensitive to initial perturbations (small $\mu(x)$) is said to be *well-conditioned*. A problem

(1999), in what follows.

[39]We remind the reader that the cost of a numerical *algorithm*, in general, is measured by the sum cost of all particular operation costs, expressed as a function of some input size *n*, in asymptotic form. If we take the worst-case cost for any particular algorithm that solves a particular problem, and then the minimum algorithmic cost (of known and unknown algorithms), then we can identify this cost with the complexity of the problem. We note in passing here the unfortunate fact about the lack of standard terminology in the literature, which can cause confusion. Many practitioners use the term 'complexity' both for algorithms and problems. Others, distinguish between saying that an algorithm has a *time complexity*, whereas a problem belongs in a *complexity class*. Whatever the terminology one may adopt, one should always bear in mind that there is conceptual difference between the two cases. Here, we choose to talk about the '*cost*' of an algorithm, and the '*complexity*' of a *problem*.

[40]See, e.g., Cucker (1999, sec.3). Extensions of the BSS model, allowing for machines with round-off errors, have also been proposed; see, e.g., Cucker and Smale (1999). Here, we only deal with exact BSS machines.

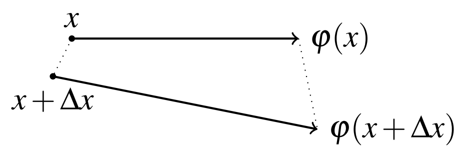


Figure 2: A stable algorithm, on perturbed input $(x+\Delta x)$ and in the presence of round-off errors, solves a slightly perturbed problem $\varphi(x+\Delta x)$.

with large $\mu(x)$ is *ill-conditioned*.[41] Conditioning is important in our context, because it affects complexity. This intuitively makes sense, of course: a problem which is very sensitive to input errors would require much more precision in its input (hence, more resources) in order to obtain useful results. Conversely, we would desire a measure of *loss* of precision of the output (relative error), given the loss of precision of the input.

An illuminating, classic example of how conditioning affects the discussion is solving a linear system of n equations $Ax = b$, where A is an $n \times n$ invertible real matrix, and $b \in \mathbb{R}^n$. The input here is b and the output is x. Letting $b+\Delta b$ be the perturbed input, and $x+\Delta x$ the perturbed output, the problem becomes:

$$A(x+\Delta x) = (b+\Delta b)$$

In this particular case, the condition number κ of the problem depends on properties of A, and is defined as:

$$\kappa(A) = \| A \| \| A^{-1} \|$$

where $\| A \|$ is the *operator norm* of A:[42]

$$\| A \| = \max_{\|x\|=1} \| A(x) \|$$

From the above, it can be quickly shown that the condition number links in a direct manner the relative output error to the input relative error:[43]

$$\frac{\| \Delta x \|}{\| x \|} \leqslant \kappa(A) \frac{\| \Delta b \|}{\| b \|} \tag{1}$$

[41]It is important to keep in mind the distinction that ‘stability’ is a property of an *algorithm*, whereas ‘conditioning’ is a property of the *problem*. If a problem is ill-conditioned, a stable algorithm is not enough to save the situation; the computation error will be large. But if a problem is well-conditioned, then all stable algorithms will give results with small errors.

[42]The operator norm of a (matrix) linear operator expresses the largest value by which the operator A stretches the vector x. In this case, it is the square root of the largest eigenvalue of the symmetric matrix $A^T A$. The definition of the condition number of the problem $Ax = b$ is due to Turing (1948).

[43]The derivation is quick and straightforward; see, e.g., Cucker (1999, 114).

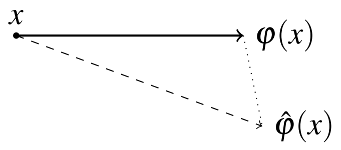


Figure 3: A change of perspective, where we consider the exact solution of a close problem $\hat{\varphi}(x)$.

If A has errors too, a similar but longer formula can be shown to connect $\kappa(A)$ with the relative errors as well (see, Cucker 1999).

From the above, it can be seen that $\kappa(A)$ provides the worst-case estimate of relative error of x as a function of the relative error of b. If we take the base-2 logarithm of both sides of eq.(5), and recall that the number of bits needed to represent a number n is $\lfloor \log_2 n \rfloor + 1$, we see that $\log_2 \kappa(A)$ expresses a worst-case estimate for the *loss of precision* in solving the problem (Cucker, 1999; Blum, 2004). This shows again that conditioning matters. It also indicates that the base-2 logarithm of the condition number can provide a better-suited parameter than bit-word size for problem instances over $\mathbb{R}$ (Blum, 2004, 19), and remedy some of the potentially problematic cases discussed at the end of sec.3 (p.13).

Nevertheless, mathematical problems like the system above, which admit of exact solutions, are not typical in scientific computing. As already mentioned (sec.4.2), more often than not, the encountered problems are such that they admit of only approximate solutions, typically by means of iterative algorithms, and up to some predetermined acceptable error ε (context-dependent). Furthermore, in practice, round-off errors appear too. This complicates things a little, but a helping hand is lent by *backward error analysis*. We only give the gist here.[44] The idea involves a slight change of perspective, according to which, instead of regarding the computed solution as an approximate solution to our original problem φ (with error ε), we instead regard it as an exact solution to a close problem $\hat{\varphi}$ (fig.3). We can then look for some *perturbed* input which would have the same effect on the output; that is, we seek some *backward* error Δx, such that $\hat{\varphi}(x) = \varphi(x + \Delta x)$ (fig.4).

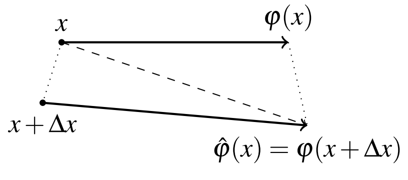


Figure 4: Schematic representation of backward error analysis. The absolute backward error is Δx and the absolute forward error is $\varepsilon = \hat{\varphi}(x) - \varphi(x)$ (in norm).

A great benefit of this method is that by finding an appropriate backward error, we in

[44]For a thorough introduction to backward analysis, one can see, e.g., Corless and Fillion (2013).

effect subsume all kinds of errors in the actual computation (measurement, representation, round-off, etc.) under the, intentionally assumed, backward error itself. Furthermore, this single error in the close problem, $\hat{\varphi}$, is treated as merely a perturbation of the initial data. Thus, not only is the analysis facilitated by dealing with only a single kind of error, but, quite importantly, we can draw on results and techniques from the rich area of perturbation theory.

Now, since in most applications we know in advance a tolerable ε, dependent on the context,[45] we are mainly interested in knowing whether the actual computational forward error is going to exceed ε or not. The strength of this approach is that if we know how well-conditioned the problem is, and if we can estimate a bound for the backward error, then we obtain a bound for the forward error as well. More precisely, in the general case of a problem with condition number $\mu(x)$, the following general inequality holds (Cucker, 1999):

$$\frac{\parallel \varphi(x) - \hat{\varphi}(x) \parallel}{\parallel \varphi(x) \parallel} \lesssim \mu(x) \frac{\parallel \Delta x \parallel}{\parallel x \parallel} \tag{2}$$

That is, the relative forward error is approximately bounded by the relative backward error multiplied by the condition number. Note that both the actual backward and forward error depend on properties of the algorithm as well as on properties of the particular FPA system.

The upshot of it all is that the conditioning of the problem at hand bears significantly on the output errors. Furthermore, in the ubiquitous case of iterative algorithms that are meant to approximate a solution up to a predetermined ε, the cost of such algorithms also may depend on the input size *and* ε itself (contrary to the exact cases, where it depends solely on the input size). As a result, the cost of an iterative algorithm depends on the condition of the inputs as well. Since space limitations do not allow a discussion of relevant examples here, the reader is referred to Cucker (1999, sec.8) and Smale (1997), for several examples of problems and precise results, where costs depend *both* on condition and ε. For a complexity theory of problems over $\mathbb{R}$, the study of conditioning, then, becomes essential.

We are now ready to return back to the two conditions posed by Woźniakowski (1999) in order for BSS to be a reliable model for real-world computations, implemented by FPA algorithms. Recall that the first condition was that we model a stable algorithm.

Consider the general case of a problem with condition $\mu(x)$ that is not solved exactly, but for which we have a stable algorithm that approximates the solution within some error bound ε. Assume also that this algorithm is to be implemented in an FPA system with round-off unit u. Woźniakowski's second condition relates these parameters via the following relation (1999, 455):

$$C \cdot u \cdot \mu(x) \leqslant \varepsilon \tag{3}$$

[45] Typically, within the interval $[10^{-8}, 10^{-2}]$ for most applications (Woźniakowski, 1999).

The additional parameter C, called the *accumulation constant of round-off errors*, expresses the accumulation of round-off errors due to the algorithm; it is often a polynomial in the number of inputs, and is expected to be of low degree, since the algorithm is assumed stable.[46] The unit round-off, u, depends on the parameters of the particular system of FPA used (single or double precision, size of significand, etc.; see appendix).

So, whenever (3) and stability hold, we can expect that the total errors of the actual computation will not exceed the accepted error ε. The FPA implementation of the algorithm is stable, thereby yielding a small forward error. As Woźniakowski says, this error is bounded by $C \cdot u \cdot \mu(x)$.[47] Therefore, as long as (3) is satisfied, the error is also bounded by ε, which is the approximation assumed in BSS. Thus, the actual error in FPA is kept within the BSS ε-approximation, thereby rendering our result in the latter faithful to the FPA output.

Finally, going back to table 1 and the comparison between BSS, FPA, and bit-computation, the third important characteristic is the cost of the operations. In this respect, BSS is closer to actual FPA computations than the bit-computation model. This is because in both cases the cost of operations is fixed; that is, it is not dependent on the *size* of the operands. So one only needs to replace the unit costs calculated in BSS with the actual fixed costs of operation in FPA. This is a big difference with Turing-machine–based models, such as bit-computation, where the size of operands matters. Since costs of algorithms are expressed as functions of input sizes, in the case of a Turing machine (with an oracle), the input includes the desired precision n of the output, which though may affect the input bit size (dependent on m; see, sec.2.1.1). This is not the case in BSS, where although ε may also be part of the input and affect the computing costs (p.25), it does not however affect the size of the input itself.

5.1 When the idealisations fail

What happens when (3) is not satisfied? One way to deal with that is to increase the precision of FPA; that is, to reduce u. But if the problem is not well-conditioned, then (3) might not be satisfied still. In that case, a more "fine-structured" model is necessary. The bit-computation model will be the appropriate framework then, since its results are more sensitive to the available accuracy of the approximation to the initial data. Of course, all this presupposes that the problems of conditioning discussed in sec.3 (p.13) have been taken into account.

For more on how cost and complexity are analysed in bit-computation, one can see Braverman (2005), Ko (1991), or Weihrauch (2000).

[46] See, Woźniakowski (1999).

[47] To see this, consider eq.(2), and recall that the backward relative error in this case comprises the relative error due to FP rounding (bounded by u; eq.(5) in the appendix) and the relative error due to the stable algorithm itself (bounded by C).

6 Which foundational framework then?

The proposal of this paper, as discussed in sec.4.2, is that the answer to this question depends on how one understands the term "foundations for scientific computing"; in a sense, on whether one puts the emphasis on the '*computing*' or on the '*scientific*' part.

If the quest is for a theoretical framework that allows for rigorous results about what elements and functions of continuous spaces can in principle be computed, and how easily, then the appropriate framework is that provided by computable analysis. Such a framework regiments the informal notion of 'effective computation' as regards uncountable spaces. An extended form of the Church-Turing thesis can be accepted (for lack of a better name, let us call it the 'Uncountable Church-Turing Thesis', UCTT), which explicates the informal idea of effectively computable functions over uncountable domains by extensionally identifying them with functions computable by a Type-2 Turing machine (under admissible representations). By UCTT, then, such a framework *a fortiori* delimits what scientific computers can do and with what difficulty. That is, TTE can be regarded as *the* fundamental theory of computing. Nevertheless, for common applications, this fundamentality comes with a cost; that of studying costs (pardon the pun) and lower/upper bounds for common-or-garden algorithms and problems arising in the computational sciences.

On the other hand, if the quest is for a framework that is from the outset practice oriented, under the tacit assumption that modern-day scientific computing is performed by digital computers (using FPA), the answer to the problem of choice that we put forward here is a pragmatic one.[48] The discussion in the previous section shows that BSS is better-suited for modelling FPA algorithms and their costs, at least under "standard conditions". Whereas CA can be regarded as the fundamental theory of computation, BSS can be regarded as a theory at a higher level of abstraction, convenient for certain purposes. In addition, BSS also captures something of the way mathematicians think of algorithms at the initial stages of their development: they initially assume that the algorithms operate over exact real numbers (e.g., Gauss elimination, bisection algorithms, etc.) without presupposing some specific representation

[48]To put the contrast between the two interpretations in a different, maybe clearer, way: consider the possibility that usable quantum computers are finally developed and put to commercial use. Then, of their many initial applications, scientific computing is expected to be a prominent one (e.g., simulating quantum systems). A foundational framework for scientific computing in the first sense, then, would subsume this subfield too. Since, quantum computers do not go beyond Turing computability limits, (un)computability and complexity results proven in the context of Type-2 Effectivity will be relevant to quantum scientific computations as well. On the other hand, a foundational framework in the second sense would be rather irrelevant for quantum scientific computations, since the latter may not employ anything similar to FPA, or exhibit round-offs arising from digital representation (it remains to see what they will actually be using, and what their round-offs will look like).

scheme.[49] This approach makes the first, coarse comparison between different algorithms (in terms of costs) easier. After leaving out the most costly ones, a more fine-grained comparison, taking into account round-offs and other errors follows. Thus, it can be said that the implicit model of computation —which defines what steps are primitive, and what thus counts as an algorithm— is, at the first stage, something along the lines of BSS/Real-RAM. But as long as the conditions of *stability* and eq.(3) also hold, BSS is pertinent to *all* stages, due to increased mathematical tractability. The "highly unrealistic" idealisations of BSS is the price to pay for manageable analyses.

Now, if our particular problem demands analyses of a finer structure, either because particular applications require smaller ε, or because the problem at hand is too sensitive to perturbations in the initial data, then bit-computation is the model to go with. Here, the difficulty of cost analyses of typical (e.g., iterative) algorithms is the price to pay for more realistic assumptions. Nevertheless, the use of bit-computation to analyse any algorithm and problem encountered in scientific computing (and not only ill-conditioned ones), although possible in principle (Woźniakowski, 1999), would be rather inefficient and unmotivated, for the everyday practitioner.

7 An analogy with scientific representation

Our proposed solution to the dispute over the proper way to found scientific computing over the reals, in terms of fundamental and higher-level models, is suggestive of analogous situations of modelling in empirical science.

There is extensive discussion in philosophy of science about idealised models and "inconsistent" scientific representations in cases where we employ different, or even incompatible, descriptions of certain systems. To give a much-cited example,[50] in fluid dynamics we often mathematically represent water as a continuous medium (by means of, e.g., the Euler or the Navier-Stokes equations). Nevertheless, we know that water is in fact discrete, consisting of molecules, and we do represent it as such within other contexts (e.g., in a QM or chemistry class). Arguably, then, the choice of the appropriate representation turns on pragmatic considerations, related to how much one is willing to make a trade-off between manageability

[49] As a toy example, consider the high school algorithm for finding the roots of the quadratic $ax^2+bx+c=0$:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \tag{4}$$

In developing an algorithm like this, there is no initial assumption to the effect that a, b, and c are rational, or floating-point numbers, or Cauchy sequences of rationals, etc. The algorithm is thought of as if it's operating over the exact reals, as elements of the field $\mathbb{R}$. Representational concerns arise at a later stage. (After all, when Newton, Gauss, et al. developed their methods, there were no digital computers around.) For a detailed discussion of this practice, and its implications for the conceptualisation of 'algorithm' in logic and mathematics, see, Papayannopoulos (2018, ch.2-3).

[50] See, e.g., Maddy (1997, 1992) and Pincock (2012, 2014).

of calculations and accuracy of representation, according to the needs of the specific application. Strictly false representations are used even though no scientist would really hold that the continuous representation of water is actually true.

The proposal is then that a similar case can be made for the rival approaches to real computability. And I submit that the particular issue of the foundations of scientific computing is also an interesting case even for discussions about unrealistic models in the context of empirical science, for it provides a unique example in which both the model and the target domain are mathematical; thereby showing that unrealistic idealisations can be appropriate even for modelling aspects of mathematics itself.[51]

In the last part of this paper, we attempt to take the analogy between the two cases even further and pin down some assumptions that in a sense are common to both instances of modelling.

7.1 Idealisations in scientific representation and the principle of regularity

Scientific representation involves mathematical models that are meant to describe the state and evolution of certain physical systems in time. Such models predict experimental outcomes, which are to be obtained and/or tested by means of experimental apparatus and measurements. The continuous mathematics, in which these models are often couched, expresses relations between real values which correspond to the magnitudes of physical properties. In the theory then, knowledge of physical quantities is assumed exact. In practice, we never have exact knowledge either of the state of the physical system or of the experimental apparatus; only approximate knowledge within some error is available to us, no matter how meticulously we try to eliminate errors of any kind. This means that a mathematical description is by its nature already an idealised picture of what we know about a target system.

A consequence of this discrepancy between mathematical representation and physical reality is the following. In order for the former to be of some applicability to concrete cases, it needs to be sufficiently insensitive to the fact that no perfect information is ever possible for the physical state at hand. For example, a solution to a differential equation that would be so sensitive to the input data as to yield considerably different outcomes whenever — even *arbitrarily* small— variations in the data occur, would be of no help for numerical

[51]There is a precise sense in which FPA is a mathematical structure; i.e., has an underlying set $\mathbb{F}$ endowed with specific operations and functions. See the appendix for a very brief exposition.

predictions.[52] Courant and Hilbert (1962) describe this specific condition as one of the three requirements that a mathematical problem should satisfy in order to be a "properly posed" problem for physical reality.[53]

> [A] mathematical problem cannot be considered as realistically corresponding to physical phenomena unless a variation of the given data in a sufficiently small range leads to an arbitrary small change in the solution. This requirement of "stability" is not only essential for meaningful problems in mathematical1 physics, but also for approximation methods. (Courant and Hilbert, 1962, 227)

---

[52]What could such an equation look like? A typical example is the two-dimensional Laplace equation: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$. Given determinate boundary conditions along the the y-axis, $u(0,y)$ and $\frac{\partial u}{\partial x}(0,x)$, a unique solution $u(x,y)$ can be obtained. More specifically, if we let: $u(0,y) = 0$ and $\frac{\partial u}{\partial x}(0,y) = g_n(y) = \frac{\sin(ny)}{n}$, the unique solution for the n^{th} problem ($n = 1,2,3,..$), is:

$$u(x,y) = \frac{1}{2n^2}(e^{nx} - e^{-nx})\sin(ny)$$

The initial data $u(0,y)$ and $\frac{\partial u}{\partial x}(0,x) = g_n(y)$ can be kept in this case arbitrarily close to zero while n grows very large. Assume a determinate x ($\neq 0$); say, $x = 1$. The solution becomes:

$$u(1,y) = \frac{1}{2n^2}(e^{n} - e^{-n})\sin(ny)$$

In this sort of situation, then, the solution can become extremely large —by means of increasing n as much as we want, thereby exploding the term $\frac{e^n}{n}$— while at the same time keeping $|g_n(y)| = |\frac{\sin(ny)}{n}| < \varepsilon$ for any $\varepsilon > 0$ and any y (and even assuming $u(0,y) = 0$). This means that *no matter how accurate a measurement of the data could be*, if we don't know the *exact* value of $\frac{\partial u}{\partial x}(0,y)$, it is impossible to approximate the solution $u(1,y)$ to any degree of accuracy. This is an example of an ill-posed problem, and it's far worse than a chaotic system. In the words of J. Hadamard:

> "[..T]he mere replacing of the value zero for [$g_n(y)$] by the (however small) value [$\frac{\sin(ny)}{n}$] changes the solution not by very small but by very great quantities. Everything takes place, physically speaking, as if the knowledge of [the initial] data would not determine the unknown function. This shows how very differently things behave in this case and in those which correspond to physical questions. If a physical phenomenon were to be dependent on such an analytical problem as [the one here] it would appear to us as being governed by pure chance.. (Hadamard, 1923, 38)

By the word 'knowledge' in this quote, Hadamard refers to experimental knowledge, within a certain approximation. In his analysis of the problem, he defines different *orders of continuity*, and pinpoints the source of this problem to the fact that the solution $u(x,y)$ of the problem is *not continuous in the data* $u(0,y)$ and $g_n(y)$ of any order at all.

I took this example from Myrvold (1995). It is originally due to Hadamard (1923), and was later discussed in Courant and Hilbert (1962) too, in the context of characterising well-posed problems in mathematical physics. As Myrvold notices, the construction by Pour-El and Richards (1981) of the well-known Turing-uncomputable solution of the wave equation with computable initial data, exploits the same fact that underlies Hadamard's example; namely, that the solution is not continuously dependent on the initial data.

[53]The other two are that a solution exists and is uniquely determined. Courant and Hilbert present Hadamard's example as well, as a case of an ill-posed problem.

What Courant and Hilbert mean by the 'requirement of stability' here,[54] Tisza (1963), in a similar vein, refers to as 'the principle of regularity':

> Experiment provides us with the values of continuous variables within a certain accuracy. However, the analysis of mathematical physics would be extremely cumbersome and lose much of its precision if the finite width of continuous parameters corresponding to empirical quantities were to be observed at every step. In actual practice, the segments of finite widths are sharpened into definite points of the continuum for the purposes of the analysis. However, the results obtained have no physical meaning unless they are sufficiently insensitive to the actual unsharpness of the continuous parameters.
>
> Mathematical solutions satisfying this requirement will be called *regular*. [..] The requirement that mathematical solutions be regular in order to have a physical meaning will prove to be of great importance and deserves to be called a principle, the *principle of regularity*. (1963, 159; emphasis in original)

As it is discussed in Myrvold (1994, 1995), the point of this principle (and of Courant and Hilbert's "stability") is *not* to provide some a priori reasoning to the effect that no physical systems exhibit discontinuous dependence on initial conditions or that 'nature does not make jumps'.[55] Rather, the point is that "even if *natura facit saltum*, the quantities about which we can make reliable *quantitative* predictions will be found in the regions where Nature refrains from leaping" (1995, 39; emphasis added). In passing, we note that even Poincaré, in (1902), had formulated some qualitative requirements of a similar flavour.[56]

We think that the above considerations —which unfortunately have gone rather unnoticed in contemporary debates about scientific representation— offer some insights into understanding why, in the general case,[57] approximate and/or idealised models may be reliable. In a successful mathematical representation, the points of the continuum as resulted from the "sharpening" of the error intervals into definite points (as Tisza puts it) are insensitive

[54]We think that the context here does not allow for misinterpretation between the sense of 'stability' in C&H's quote (which refers to a problem) and the sense of 'numerical stability' used earlier, which is a property of algorithms.

[55]Courant and Hilbert, for example, mention non-linear systems and quantum theory as cases in which studied problems reflect real phenomena, even though they are not properly-posed in the above sense.

[56]"[T]here are those [hypotheses] that are quite natural and without which we could hardly do. It is difficult not to suppose that the influence of very distant bodies is quite negligible, that small movements obey a linear law, that *the effect is a continuous function of the cause*. [..] All these hypotheses form, so to speak, the common foundation of all theories in mathematical physics." (Poincaré, 1902, 524; emphasis added).

[57]Again, the aim of this discussion is not to delimit what meaningful representations in empirical science are. As Courant and Hilbert (1962, 230) say, there exist not well-posed problems whose study however can be of physical significance (a classic example might be the inverse heat equation). Still, however, for the purposes of this paper, we think it adequate to restrict attention to the success of idealised models that are concerned with well-posed problems.

to the actual "unsharpness" of the experimentally obtained parameters. Hence, any theoretical definite point of the representation will be a good approximation for all actual values within a neighbourhood around it. Small variations within the neighbourhood do not render the theoretical point of the model too far removed from the actual one; it will still be a good approximation.[58] As long as these conditions hold, then, we can make sense of fruitful scientific representations that seem even highly idealised. Considering, for instance, the gravitational force at some far distance from a *point mass* ($F(m,r) = G\frac{Mm}{r^2}$), instead of a mass with dimensions, gives a "sharpened" result that is still within the neighbourhood of the actual "unsharpened" value for the physical 3D object, and furthermore a good approximation for any distance r since the problem is well-posed (in the earlier Courant-Hilbert sense). Of course, what precisely makes a 'neighbourhood' (or 'far distance', in this example) is context-dependent, and varies with the intended scale of analysis (how coarse- or fine-grained we need it to be). Clearly, the same account applies to cases where we use continuous models to study purely discrete phenomena, such as continuous parameters and functions for populations of species.

We suggest that the cases in which BSS is a successful model can be seen as analogous; even though both model and target domain are now mathematical.[59] When BSS (or Real-RAM) is used to analyse the cost of specific algorithms (and, based on that, the complexity of certain problems), it can be seen as an idealised representation of an FPA computation by a digital computer. If we are using BSS to model a stable FPA algorithm for a well-conditioned problem, implemented with small unit round-off u (i.e., the two Woźniakowski conditions), then *ex hypothesi* we are modelling a system that solves a close problem to the theoretical one and with good approximation (the target system "does not make jumps", so to speak). Due to eq.(3), each FP (output) value is bounded by ε, which is the radius of the BSS (output) value neighbourhood, meaning that, in a sense, the BSS output has a "continuous dependence" on the FPA output. (If we image a mapping between the two outputs, the image of variations in the FPA output will be within any variation ε in the BSS output.)

On the other hand, assume we are using BSS to model the computation of the solution to some problem which critically depends on the initial data (the target system does make jumps, that is), such as the following function:

$$f(x) := \begin{cases} 1 & x \in \mathbb{Q} \\ 0 & x \notin \mathbb{Q} \text{ but } x^2 \in \mathbb{Q} \\ \text{div} & \text{otherwise} \end{cases}$$

[58]Or if, like Poincaré (1902), one is comfortable with cause-effect terminology, the "effect" depends continuously on the "cause" (see, fn.56).

[59]For example, the floating-point numbers can be seen as the "analogue" of experimental values. The latter are always measured with some error, which corresponds to the *representation error* of floating-point numbers, since in both cases a real value gets approximated and represented by a rational one. Round-off errors can be considered to corresponded to propagation of errors in experimental physics; that is, uncertainties in *derived* quantities caused by initial uncertainties in the experimentally measured quantities.

This function is computable in BSS. Hence, our BSS representation of this computation by a BSS-machine, although mathematically correct[60], does not realistically correspond to any *computational* phenomena (paraphrasing Courant and Hilbert here) —or does not have any *computational* meaning (paraphrasing Tisza)— since no realistic physically computing machine (or idealised human agent)[61] can effectively compute such a function (it is Turing-uncomputable).

8 Final Remarks

We examined two rival models of computation, which, among others goals, aim to offer foundations for scientific computing. We put forward two different ways of interpreting such a pursuit, each one having different implications for the question of what exactly the target domain of such models should be. Based on our proposal, we discussed the appropriateness of each model with respect to each particular interpretation. This was the first goal of this paper.[62]

Adjudicating on the appropriateness of rival idealised models with respect to a certain target domain requires —besides clarifying what the target domain exactly is— a closer investigation into the conceptual underpinnings of the employed idealisations in each model. The idealisations used in the Turing machine model (e.g., unlimited, or sometimes even infinite, tape) have been well-discussed in the philosophical literature already. Our second goal, then, was to account for the fruitfulness and justification of the —more controversial and more criticised— idealisations in BSS, given that this model has been around, in an equivalent form, since the early ’80s.[63]

In examining the controversy around the choice of the appropriate real computability model, one comes to see that although the modelled domain in this case is actually mathematical, one can nonetheless identify interesting similarities to idealised modelling of physical domains. As a third main goal of this work, then, we attempted to pinpoint common conditions that can be seen as underlying a great variety of idealised-though-reliable mathematical representations, either within empirical science or within mathematics itself.

[60]In analogy with the Hadamard example, in fn.52, which too is mathematically correct.

[61]Assuming the validity of UCTT and ignoring the possibility of hypercomputation.

[62]We also hoped to achieve a further, indirect goal: that of drawing attention to a debate that has long been going on within the mathematics and computer science communities, yet remains, by and large, unnoticed by the philosophical community.

[63]To the author’s knowledge, the earliest explicit description of this model was in Preparata and Shamos (1985), under the name ‘Real-RAM’ (though, according to the Wikipedia entry “Real-RAM”, the model was originally formulated in 1978 in Shamos’s PhD thesis).

Appendix: Floating-point arithmetic

Physical computing requires ways of representing the entities operated upon. When you are computing on a napkin the generous tip you should leave with your restaurant bill, you typically represent the added numbers in decimal notation. If you are trying to solve (perhaps on the same napkin) a physics problem with larger numbers (e.g., Avogadro's number), you might employ scientific notation, in powers of 10, as it is convenient for computing only with the significant figures. Accordingly, digital, silicon-based computing requires representation of numbers in some convenient system. Many possible systems exist (see, e.g., chapter 4 in Knuth 1997), but in modern computers, the near ubiquitous representation system is Floating-Point Arithmetic (FPA).

By 'representation', in this context, we mean a mapping $\delta :\subseteq R \to \Sigma^*$ from a set of numbers (e.g., the reals, $R = \mathbb{R}$) to a set of finite strings Σ^* over a finite alphabet Σ. A possible string from Σ^* can be of the form: $d_n d_{n-1} ... d_1 d_0$ ($d_i \in \Sigma$). Additionally, it can also allow for decimal points, and be of the form:

$$d_n d_{n-1} ... d_1 d_0 . d_{-1} d_{-2} ... d_{-p+1} d_{-p}$$

In floating-point arithmetic, numbers are represented in a way that is structurally reminiscent of scientific notation; that is, in the form:

$$\text{significan}d \times \text{base}^{\text{exponent}}$$

For example, assuming base 10, the number 152853.5047, can be represented in the above form as 1.528535047×10^5. Also, if the base is fixed and known in advance, then there is no need to include a representation of it as part of the actual FP string used by the system. The representation of the number, then, can comprise just two parts: the *significand* and the *exponent* (for example: 1.528535047|5).

More precisely, then, an n-bit FP word consists of two parts: a k-bit word, the *significand* (aka *mantissa*, or *coefficient*), representing a signed rational number, and an l-bit word ($k + l = n$), the *exponent* (aka *characteristic*, or *scale*), representing a signed integer.

In concrete applications, there are two main ways of representing a positive or negative number, either in the significand or in the exponent. The first is to use a designated bit to denote the sign and the remaining available bits for the magnitude. The second is to use all the available bits for the magnitude, and add a bias.

The most widely used system of FPA today is the IEEE-754 standard. This standard requires that the significand is normalised as a number of the form (Corless and Fillion, 2013)

$$\pm 1.d_{-1} d_{-2} ... d_{-p+1} d_{-p}$$

That is, there is one bit used for the sign and the rest for the magnitude. However, since the first digit is always 1, it can be implied and hidden, and hence all the bits (except the

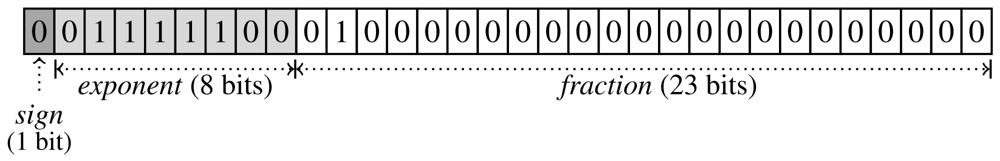


Figure 5: Single-precision (32-bit) floating-point representation (base 2) of the number 0.15625, in accordance with the IEEE-754 standard. The first bit signifies the sign of the significand, the next 8 bits are assigned to the representation of the exponent, and the last 23 bits are assigned to the fractional part of the significand.

one for the sign) can be used for the fractional part of the number. For the exponent, on the other hand, the standard requires a biased representation. Furthermore, IEEE-754 defines five formats, in terms of precision, the most common of which are *single* precision (32-bit) and *double* precision (64-bit).

To make things clearer, we present in figure 5 a 32-bit example. In this precision format, the exponent can be either an 8-bit signed integer from -128 to 127, or an 8-bit unsigned integer from 0 to 255 (i.e., biased); IEEE-754 requires the latter.

An important concept, with respect to the particular system, is the *unit round-off, u*. It is meant to express an *upper bound on the relative error* due to the representation and round-off errors of the particular FP system. A related term is the *machine epsilon, ε*. Unfortunately, there are no standard definitions of these ubiquitous terms, and approaches vary, even between the numerical analysis and the numerical software literature. In the former tradition, the two terms are often used interchangeably to refer to the maximum relative error due to the particular FP representation. In the latter, ε is defined as the difference between 1 and the next larger floating-point number (equivalently, the spacing of floating-point numbers when the exponent is zero; see, e.g., Corless and Fillion 2013, 805). In this case, the unit round-off is defined as $\varepsilon/2$, where rounding to the nearest is assumed. In any case, u is meant as the maximum relative error due to FP representation (and round-offs).

It is clear that only countably many reals can be represented exactly by means of floating-point systems. The finite set $\mathbb{F}$ of *floating-point numbers* is the set of all numbers that have an *exact* floating-point representation. More precisely, for some set Σ^*_{fl} of floating-point words, and some floating-point representation mapping $\delta_{fl} :\subseteq \mathbb{R} \to \Sigma^*_{fl}$,

$$\mathbb{F} = \{x \in \mathbb{R} \mid \delta_{fl}(x) = s,\ s \in \Sigma^*_{fl}\}$$

Let $\mathrm{fl}(x) \in \mathbb{F}$ be the floating-point number to which some real number $x \in \mathbb{R}$ is rounded. Then, the following holds for the unit round-off, u ($0 < u < 1$):

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leqslant u \tag{5}$$

The exact operation $\mathrm{fl} :\subset \mathbb{R} \to \mathbb{F}$ is determined by the particular FPA system employed.

In the body of this paper, we have repeatedly asserted that the considered models of computation are meant to model FPA as a mathematical structure. We obtain such a structure by endowing the set $\mathbb{F}$ with (counterparts of) the basic arithmetic operations $(\pm, \times, \div)$, along with some additional ones related to rounding, such as $\mathrm{fl}(x)$. The resulted structure exhibits peculiar mathematical properties. For instance, addition is generally not associative or distributive, the set $\mathbb{F}$ is not closed under addition, and it is neither a field, nor a ring or a group. These (and other) bizarre properties mainly occur due to the accumulation of representation and round-off errors. However, the exact mathematical properties depend on the specific representation format used. Therefore, significant effort has been put into developing satisfactory formats, in which the implementations of the above operations are precisely specified, and the accumulation of round-offs and representation errors are held at a minimum. The IEEE-754 standard, with double (64-bit) precision, guarantees many desirable properties, whence its popularity. It is exactly these messy details and cumbersome practicalities that make the majority of numerical analysts, and other practitioners, turn to easier models, such as BSS and Real-RAM.[64]

Acknowledgements

I am indebted to Wayne Myrvold for discussions on some of the topics dealt with here, and to Alexander Regueiro for thoroughly reading this paper and giving advice on its final form. I have also been greatly benefited from the comments and suggestions of two anonymous referees of *Synthese*. Part of this research was carried out while I held a post-doctoral fellowship at the Sidney M. Edelstein Centre for History and Philosophy of Science Technology and Medicine, at the Hebrew University of Jerusalem.

References

Aberth, O. (1980). *Computable Anlysis*. McGraw-Hill.

Avigad, J. and V. Brattka (2014). Computability and analysis: the legacy of Alan Turing. In R. Downey (Ed.), *Turing's Legacy: Developments from Turing's Ideas in Logic*, pp. 1–47. Cambridge University Press.

Blum, L. (2004). Computing over the reals: Where Turing meets Newton. *Notices of the AMS 51*(9), 1024–1034.

[64]This appendix provided only a very sketchy description of FPA, omitting important concepts related to errors, such as undeflow, overflow, rounding operations, and others. We refer the interested reader to various online resources as well as to the very illuminating discussion in Corless and Fillion (2013), upon which this appendix has also extensively drawn.

Blum, L., F. Cucker, M. Shub, and S. Smale (1997). *Complexity and Real Computation*. Springer Science.

Borel, É. (1912). Le calcul des intégrales définies. *Journal de Mathématiques pures et appliquées 8*, 159–210.

Brattka, V. (2000). The emperor's new recursiveness: The epigraph of the exponential function in two models of computability. *Words, languages and combinatorics 3*, 63–72.

Brattka, V., P. Hertling, and K. Weihrauch (2008). A tutorial on computable analysis. In S. B. Cooper, B. Löwe, and A. Sorbi (Eds.), *New Computational Paradigms: Changing Conceptions of What is Computable*, pp. 425–491. New York, NY: Springer New York.

Braverman, M. (2005). On the complexity of real functions. In *Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pp. 155–164.

Braverman, M. and S. Cook (2006). Computing over the reals: Foundations for scientific computing. *Notices of the AMS 53*(3), 318–329.

Calvert, W., K. Kramer, and R. Miller (2011). Noncomputable functions in the Blum-Shub-Smale model. *Logical Methods in Computer Science 7*(2), 1–20.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics 58*(2), 345–363.

Corless, R. M. and N. Fillion (2013). *A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis*. Springer-Verlag New York.

Courant, R. and D. Hilbert (1962). *Methods of Mathematical Physics. Volume II: Partial Differential Equations*. John Wiley & Sons.

Cucker, F. (1999). Real computations with fake numbers. In J. Wiedermann, P. van Emde Boas, and M. Nielsen (Eds.), *Automata, Languages and Programming*, pp. 55–73. Springer Berlin Heidelberg.

Cucker, F. and S. Smale (1999). Complexity estimates depending on condition and round-off error. *J. ACM 46*(1), 113–184.

Gherardi, G. (2008). Computability and incomputability of differential equations. In R. Lupacchini and G. Corsi (Eds.), *Deduction, Computation, Experiment*, pp. 223–242. Springer, Milano.

Grzegorczyk, A. (1955). Computable functionals. *Fund. Math 42*(19553), 168–202.

Hadamard, J. (1923). *Lectures on Cauchy's Problem in Linear Partial Differential Equations*. New Haven Yale University Press.

Higham, N. J., M. R. Dennis, P. Glendinning, P. A. Martin, F. Santosa, and J. Tanner (2015). *The Princeton Companion to Applied Mathematics*. Princeton University Press.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Ko, K.-I. (1991). *Complexity Theory of Real Functions*. Cambridge, MA, USA: Birkhauser Boston Inc.

Lacombe, D. (1955). Extension de la notion de fonction recursive aux fonctions dune ou plusieurs variables reelles. *Comptes Rendus Hebdomadaires Des Seances De L' Academie Des Sciences 240*(26), 2478–2480.

Maddy, P. (1992). Indispensability and practice. *The Journal of Philosophy 89*(6), 275–289.

Maddy, P. (1997). *Naturalism in Mathematics*. Oxford University Press.

Mazur, S. (1963). *Computable Analysis*. Instytut Matematyczny Polskiej Akademi Nauk.

Myrvold, W. C. (1994). *Constructivism, Computability, and Physical Theories*. Ph. D. thesis, Boston University.

Myrvold, W. C. (1995). Computability in quantum mechanics. In W. D. Pauli-Schimanovich, E. Köhler, and F. Stadler (Eds.), *Vienna Circle Institute Yearbook*, pp. 33–46. Kluwer Academic Publishers.

Norton, J. D. (2012). Approximation and idealization: Why the difference matters. *Philosophy of Science 79*(2), 207–232.

Novak, E. (1995). The real number model in numerical analysis. *Journal of Complexity 11*(1), 57 – 73.

Papayannopoulos, P. (2018). *Computing, Modelling, and Scientific Practice: Foundational Analyses and Limitations*. Ph. D. thesis, University Of Western Ontario.

Pégny, M. (2016). How to make a meaningful comparison of models: The Church–Turing thesis over the reals. *Minds and Machines 26*(4), 359–388.

Pincock, C. (2012). *Mathematics and Scientific Representation*. Oxford University Press.

Pincock, C. (2014). How to avoid inconsistent idealizations. *Synthese 191*(13), 2957–2972.

Poincaré, H. (1902). Relations between experimental physics and mathematical physics. *The Monist 12*(4), 516–543.

Pour-El, M. B. and I. Richards (1981). The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics 39*(3), 215–239.

Pour-El, M. B. and I. Richards (1989). *Computability in Analysis and Physics*. Berlin: Springer-Verlag.

Preparata, F. P. and M. I. Shamos (1985). *Computational Geometry: An Introduction*. New York, NY, USA: Springer-Verlag New York, Inc.

Smale, S. (1990). Some remarks on the foundations of numerical analysis. *SIAM Rev. 32*(2), 211–220.

Smale, S. (1997). Complexity theory and numerical analysis. *Acta Numerica 6*, 523–551.

Tisza, L. (1963). The conceptual structure of physics. *Rev. Mod. Phys. 35*, 151–184.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society 42*(1), 230–265.

Turing, A. M. (1948). Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics 1*(1), 287–308.

von Neumann, J. and H. H. Goldstine (1947). Numerical inverting of matrices of high order. *Bull. Amer. Math. Soc. 53*(11), 1021–1099.

Weihrauch, K. (2000). *Computable Analysis: An Introduction*. Springer Science.

Woźniakowski, H. (1999). Why does information-based complexity use the real number model? *Theoretical Computer Science 219*(1), 451 – 465.