Abstract Geometrical Computation and Computable Analysis

Jérôme Durand-Lose[⋆]

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, B.P. 6759, F-45067 ORLÉANS Cedex 2, France

Abstract. Extended Signal machines are proven able to compute any computable function in the understanding of recursive/computable analysis (CA), here type-2 Turing machines (T2-TM) with signed binary encoding. This relies on an intermediate representation of any real number as an integer (in signed binary) plus an exact value in $(-1, 1)$ which allows to have only finitely many signals present outside of the computation. Extracting a (signed) bit, improving the precision by one bit and iterating the T2-TM only involve standard signal machines.
For exact CA-computations, T2-TM have to deal with an infinite entry and to run through infinitely many iterations to produce an infinite output. This infinite duration can be provided by constructions emulating the black hole model of computation on an *extended* signal machine. Extracting/encoding an infinite sequence of bits is achieved as the limit of the approximation process with a careful handling of accumulations and singularities.

Key-words. Analog computation, Abstract geometrical computation, Computable analysis, Signal machine, Type-2 Turing machine.

1 Introduction

Classical computability deals with integers, finite sequences of letters, and more generally up to countably many discrete values. Does this mean that Analysis and Engineering are totally disconnected from computations? That physical simulation is impossible in silicon? Of course not, this is done everyday.

Dealing with real values is handled in various ways: fixed approximation (π is 3.14), formal manipulation (π is `PI`), unbounded approximation on demand (π is generated by a program that can provide extra digits at any time), interval arithmetics (π is included in $[3.1, 3.2]$)... Following their definition as Cauchy sequences of rational numbers, real numbers can be encoded by infinite decreasing sequences of intervals with rational endpoints, such that their length tends to zero. The whole infinite sequence represents exactly one real at its intersection, but a finite prefix of the sequence is enough to get an approximation, the larger

[⋆] `http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose`, `Jerome.Durand-Lose@univ-orleans.fr` the prefix, the better the approximation. Rational endpoints allow exact coding and manipulation of each interval in classical computation, so that the infinite sequence of intervals can be written as an infinite sequence of symbols. Functions over these infinite sequences provide functions on the reals. A Turing machine is used: the infinite entry is written on one tape and the output is expected on a write-once tape (extra working tapes are available). To entirely process the input and generate the whole output, an infinite number of iterations is needed, but in finitely many an approximation is generated.

This approach, now called *Computable Analysis* (CA) was initiated by Turing [1936], then Grzegorczyk [1957] and is detailed in classical books [Ko, 1991, Weihrauch, 2000]. It was proven equivalent to a variation of Shannon's General Purpose Analog Computer [Bournez et al., 2007]. Another approach to analog computing is to imagine that values and primitives are freely available which leads to the Blum, Shub and Smale (BSS) model [Blum et al., 1989].

In recent years, *Abstract Geometrical Computation* (AGC) has been developed and proven able to simulate the original BSS [Durand-Lose, 2007, 2008]. In the present paper, AGC is shown capable of carrying out any CA-computation. An intermediate representation of real numbers with finitely many signals is provided. This encoding allows to go and return between BSS-encoding and CA-encoding.

Abstract Geometrical Computation is defined by dimensionless signals moving in an Euclidean (continuous) space in continuous time. To each signal corresponds a *meta-signal* which defines its speed. The number of meta-signals is finite. Existing signals are only modified when they collide: they are replaced by new signals according to their meta-signals and *collision rules*. A *signal machine* collects the definition of available meta-signals and collision rules.

In previous works to implement the BSS-model, real numbers are encoded by the distance between two parallel signals (plus two more signals to encode the scale, *i.e.* distance 1). On one side, for CA, this is not so convenient since infinite sequences are expected. On the other side, in the AGC context, infinitely many signals would either occupy the whole infinite space or produce *singularities* (accumulation of signals in a configuration, static) or *accumulations* (of collisions in the space-time diagram, dynamic) — preventing the values from being moved around.

These are very powerful yet dangerous artifacts. Special care has to be taken in defining the space-time diagram near them. A singularity lasts in the following configurations as long as signals are accumulating there. Anything colliding with it just gets absorbed. Unless a singularity is generated, an accumulation results in one signal (always the same). More complex rules can be devised to handle singularities and accumulations — but are not used here — so as to emulate the (nested) black hole model of computation [Hogarth, 1994, Etesi and Németi, 2002, Durand-Lose, 2006, 2009] (with the use of folding structures) as well as to achieve exact multiplication of real numbers to simulate the BSS model [Durand-Lose, 2008].

To avoid handling infinitely many signals outside of CA-computations, an intermediate representation is used. A real number x is encoded by $n+\varepsilon$ where n is an integer and $-1<\varepsilon<1$; n is encoded in signed binary and ε as a distance between two signals (a pair of signals somewhere in the configuration amounts for distance 1). The encoding used for CA implementation is signed binary with symbols $\overline{\mathtt{1}}$, $\mathtt{0}$, $\mathtt{1}$ plus a decimal point. To go forth and back between the encodings, n is encoded as a sequence of signed bits (and thus does not have to be taken care of) and the first approximation is $(n-1, n+1)$. Then the approximation goes by steps: the interval (a,b) is replaced by $(a, \frac{a+b}{2})$, $(\frac{3a+b}{4}, \frac{a+3b}{4})$ and $(\frac{a+b}{2}, b)$ that corresponds to $\overline{\mathtt{1}}$, $\mathtt{0}$ and $\mathtt{1}$ respectively. (Of course, intervals overlap and the infinite sequence is not unique.) All the geometric constructions to encode and to decode are presented.

The presentation of the implementation of Turing machines (TM) is only illustrated; it is quite straightforward and already done in [Durand-Lose, 2009]. Type-2 TM need infinitely many iterations to complete a computation. To get the results in finite duration, TM and input are embedded into a folding structure as in [Durand-Lose, 2006, 2009]. The output is an infinite convoy of signals, bringing forth singularities. It is shown how this can be handled and used to produce the exact intermediate representation.

Definitions of AGC and CA are gathered in Section 2. Section 3 concentrates on approximation, *i.e.* the intermediate representation as well as how to get/add bits, and the simulation of Turing machines. Section 4 deals with exact computation in finite duration. Section 5 concludes this paper.

2 Definitions

2.1 Abstract Geometrical Computation

A *signal machine* (SM) is defined by (M, S, R) where M (*meta-signals*) is a finite set, S (*speeds*) a mapping from M to $\mathbb{R}$, and R (*collision rules*) a function from the subsets of M of cardinality at least two into subsets of M (all these sets are composed of meta-signals of distinct speeds). A signal machine is *extended* if some meta-signal $\mu_{❄}$ is distinguished to be used as the result of an accumulation.

Each instance of a meta-signal is a *signal* located on the real axis. The mapping S assigns *speeds* to signals. A *collision rule*, $\rho^{-}\!\rightarrow\!\rho^{+}$, defines what emerges ($\rho^{+} \subseteq M$) from the collision of two or more signals ($\rho^{-} \subseteq M$). Since R is a function, SM are deterministic. The *extended value set*, V, is the union of M and R plus three symbols: $\oslash$ for void, ❄ for accumulation, and ✺ for singularity. A *configuration*, c, is a mapping from $\mathbb{R}$ to $M \cup R \cup \{\oslash\}$ such that the set $\{\, x \in \mathbb{R} \,|\, c(x) \neq \oslash \,\}$ is finite. An *extended configuration*, is a mapping from $\mathbb{R}$ to V such that all and only accumulation points of $\{\, x \in \mathbb{R} \,|\, c(x) \neq \oslash, ✺ \,\}$ have the value ✺. The locations of the ✺ are thus defined in a static way: they only depend on the configuration and not on the dynamics.

A (resp. extended) *space-time diagram* is a mapping from an interval of $\mathbb{R}$ (representing the time) into (resp. extended) configurations. A signal corresponding to a meta-signal μ at a position x, *i.e.* $c(x) = \mu$, is moving uniformly with constant speed $S(\mu)$. A signal must start (resp. end) in the initial (resp. final) configuration or in a collision. At a $\rho^- \!\to\! \rho^+$ collision, signals corresponding to the meta-signals in ρ^- (resp. ρ^+) must end (resp. start) and no other signal should be present. There is a ❄ if and only if there is no ✺ and collisions are accumulating (from before, this one is dynamic). A ❄ immediately turns into a (regular) $\mu_{\text{❄}}$ signal, so that a $\mu_{\text{❄}}$ signal can also result from an accumulation. Continuation rules could be designed to distinguish more cases, but this is useless here.

Space-time diagrams are represented with time increasing upward. The traces of signals are line segments whose directions are defined by $(S(.), 1)$ (1 is the temporal coordinate) so that the speed is the inverse of the slope. Collisions correspond to the common extremities of these segments.

2.2 Computable Analysis

A *type-2 Turing machine* (T2-TM) is a regular Turing machine (TM) such that the entry is an infinite sequence of symbols written on a read-only tape. The output is also expected to be an infinite sequence of symbols written on a write once tape (each cell can only be written once). The TM has an extra work tape on which it can freely read and write. It needs infinitely many iterations to read a whole input and write a whole output, but after finitely many iterations, a finite part of the entry is read and a finite part of the output is generated. Since the output is write-once, anything written never changes and it converges to the infinite output according to the prefix topology.

To link this machinery to analysis, a representation of real numbers by infinite sequences should be provided. The larger the prefix of the representation is read, the more should be known on the encoded real number x, ultimately x should be perfectly known and distinguished from any other real number. The standard representation of a real number x is by any decreasing sequence of open intervals with rational ends such that their intersection reduces to $\{x\}$. The infinite sequence is then just the self-delimiting concatenation of the naming of the intervals. In the present paper, an equivalent representation is used.

Let $\Sigma = \{\bullet, \overline{1}, 0, 1\}$, the *signed binary representation*, $\rho_{\text{sb}} :\subseteq \Sigma^\omega \longrightarrow \mathbb{R}$, (from [Weihrauch, 2000, Def. 7.2.4 p. 206]) is defined only for infinite sequences with only one dot ($\bullet$) by:

$$n_0 \bullet d_1 d_2 d_3 \dots d_n \dots \longmapsto \nu_{\text{sb}}(n_0) + \sum_{1 \leq i} \frac{d_i}{2^i}$$

where $n_0 \in \{\overline{1}, 0, 1\}^*$, $d_i \in \{\overline{1}, 0, 1\}$ and ν_{sb} is a naming of natural integers signed in base 2 ($\overline{1}$ representing -1). Only the open intervals $I_{n,k} = (\frac{n-1}{2^k}, \frac{n+1}{2^k})$ $(n, k \in \mathbb{N})$ are considered; this family of intervals generates the usual topology on $\mathbb{R}$.

3 Approximation

This section deals with finite run of T2-TM. After defining the intermediate representation of real numbers and the extraction of signed bits, TM implementation is presented. The section ends with approximating the intermediate representation from a sequence of signed bits.

3.1 Intermediate representation and decoding

In the signed binary representation, there are finitely many symbols before the dot and then an infinite sequence of symbols in $\{\overline{1}, 0, 1\}$. The first part, including the dot, is encoded by signals, one signal for each symbol. The second part represents a real number, ε, in $(-1, 1)$ as a signed binary infinite sequence. This number is represented in AGC, by the distance between two signals (there exist two signals somewhere representing the scale, *i.e.* their distance is 1).

The integral part is represented by parallel vertical signals ($\overline{1}$, 0 or 1) that get transformed into $\overline{1}_L$, 0_L or 1_L on receiving get (the signal sent to extract the next bit). The dot is treated similarly. These are not addressed in this paper.

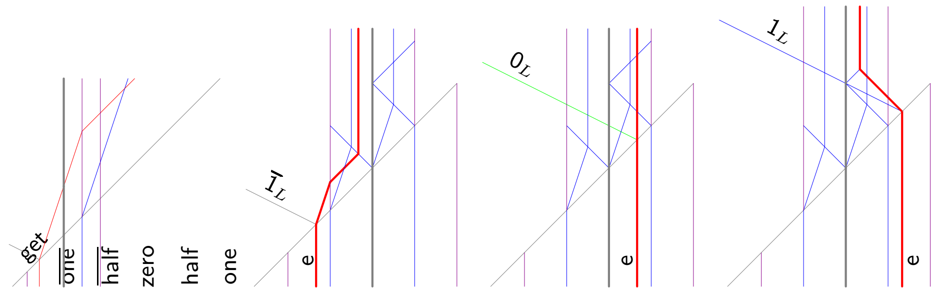
The two signals to represent ε are zero and e. More signals are present: $\overline{\text{one}}$, $\overline{\text{half}}$, half and one at position respectively -1, $-\frac{1}{2}$, $\frac{1}{2}$ and 1 (position 0 is the one of zero). Signal e is between $\overline{\text{one}}$ and one, it might be superposed with $\overline{\text{half}}$ or zero, in such a case a different meta-signal is used (these technical details are omitted from now on).

Signed bits are extracted one after the other and companion signals are prepared for next extraction: $\overline{\text{one}}$, $\overline{\text{half}}$, zero, half and one are scaled by one half and e is translated in order to correspond to the remainder. The structure on Fig. 1(a) is used for this. The three main cases are represented on Fig. 1 where signal zero (in the middle) and the one handling e are drawn with thick lines. If $-1 < \varepsilon < -\frac{1}{2^n}$ then the extracted bit is $\overline{1}$ (signal $\overline{1}_L$ exits on the left) and ε is replaced by $\varepsilon + \frac{1}{2^n}$ (e is shifted by the distance from $\overline{\text{one}}$ to $\overline{\text{half}}$). If $-\frac{1}{2^n} < \varepsilon < \frac{1}{2^n}$ then the extracted bit is 0 (signal 0_L) and ε is kept (e is not shifted). Except for the geometric construction, the last case is symmetric to the first one. Arbitrarily, if $-\frac{1}{2} < x < 0$, then the extracted bit is 0 although it could have been $\overline{1}$, similarly for $0 < x < \frac{1}{2}$.

There is no room in the paper to detail all the meta-signals and rules. The corrections of the various constructions rely on simple geometry. All have been checked and implemented in Java to generate the pictures.

3.2 Computable Analysis

A type-2 Turing machine is nothing but a regular TM that is supposed never to halt with two special purpose tapes: one initially totally filled — input — and one for writing once — output. On the input tape, the head cannot change the read symbol nor move left (*i.e.* back to the start). On the output tape, the head either rewrites the blank symbol or moves right. In the transition table,



(a) empty structure (b) $-1 < \varepsilon < -\frac{1}{2}$ (c) $-\frac{1}{2} < \varepsilon < \frac{1}{2}$ (d) $\frac{1}{2} < \varepsilon < 1$

Fig. 1. Extracting a single signed bit.

transitions that read on the input can be clearly identified. Reading is replaced by sending a get signal to the intermediate structure and waiting for the signed bit. The same is done with writing, in this case, signals encoding signed bits are sent on the side but no acknowledge/returned value is expected (next subsection presents how to deal with them).

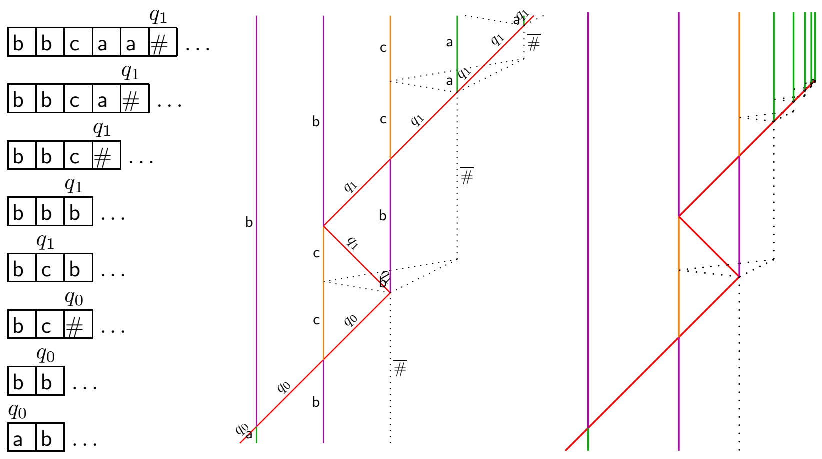
Implementing a TM in AGC has already been done in Durand-Lose [2009]. One work tape is enough; read and write operations are carried out by signals sent on the side (an answer is expected for reading). The simulation cannot be used straightaway because on an infinite run, the tape might extend infinitely, so that with previous construction, the room used for the tape becomes infinite. Since space is continuous, it is pretty easy to put less and less space between signals amounting for the cells of the tape. Indeed, a geometric decreasing is used so that each signal is half the distance from the last than the distance to the last from the previous one. This is illustrated on Fig. 2 with the first iterations of a TM, the initial implementation and the one within bounded space.

These signals might go accumulating if the head is moving right forever. But in a proper CA-computation, it is not the case: there must be infinitely many readings and each one needs signals to go forth and back from the tape to the intermediate structure. Since this distance is bounded away from zero, each reading requires a minimal time. There is no accumulation whatsoever in a normal computation. A bounded space is used by the T2-TM simulation and the encoded real number. There is always finitely many signals in any configuration.

In case of multiple input (for example for an addition), it is easy to set one after the other on the right and to have a distinct get for each entry.

3.3 Approximating the intermediate representation

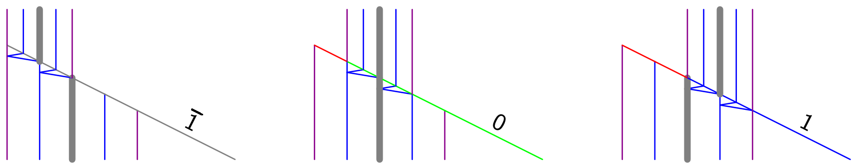
Since only approximation is concerned in this Section, only a tightening bound on ε is generated. The first bits for the integral part are encoded as they are in a geometric positioning to ensure that they remain in a bounded space. Only the fractional part needs careful attention. At start, there are five signals to encode



(a) TM computation (b) [Durand-Lose, 2009, Fig. 6] (c) finite space version (with more transitions)

Fig. 2. Simulation of a Turing machine.

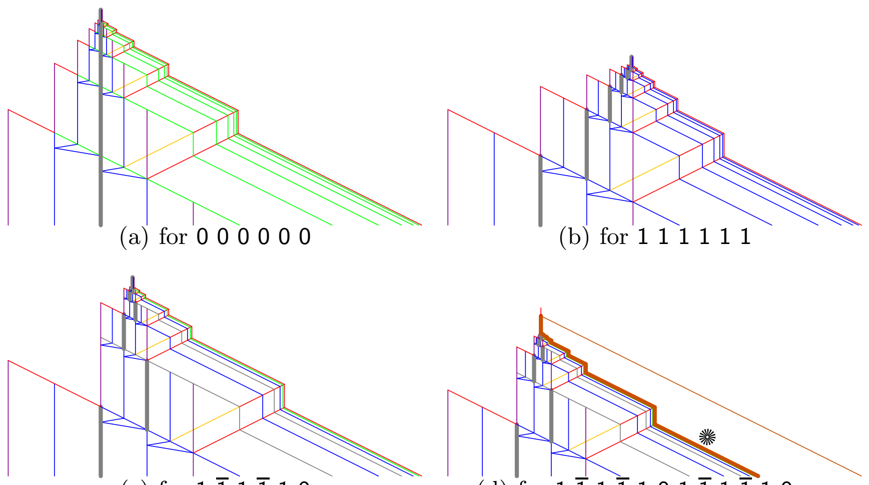
-1, $-\frac{1}{2}$, 0, $\frac{1}{2}$, and 1 at their correct relative positions (obtained for example by copying) and the approximation is up to 1. The problem is then to locate the position of e: it is somewhere between signals $\overline{\texttt{one}}$ and one. Each incoming bit allows to scale down the interval. Figure 3 shows the three cases.



(a) for $\overline{1}$ (b) for 0 (c) for 1

Fig. 3. Tightening the intermediate representation.

For exact values, next section considers infinite incoming sequences of signed bits in bounded space. Considering an incoming finite sequence of signed bits or *convoy*, the tightening for one bit should not mess with the one of another bit so delays have to be provided. Delaying is done by replacing the incoming signals by null speed signals (recording the bits). This replacement is carried out by some *toggle* signal. When the delay has elapsed, another toggle signal is sent to restore their movement. The first toggle is launched when the first remaining signal (the new one) is generated, the second toggle at the second one (the new half) as illustrated on Fig. 4 where an extra signal is added on the right to stop the toggles.


(a) for 0 0 0 0 0 0 (b) for 1 1 1 1 1 1

(c) for 1 $\overline{1}$ 1 $\overline{1}$ 1 0 (d) for 1 $\overline{1}$ 1 $\overline{1}$ 1 0 1 $\overline{1}$ 1 $\overline{1}$ 1 0 . . .

Fig. 4. Management of three finite convoys and an infinite one.

4 Exact computation

4.1 Folding and extended signal machines

Except for the output signals, the Type-2 Turing machine and the input use a bounded portion of space during the whole computation. There exist constructions to fold into a bounded space and bounded time a spatially bounded infinite time computation. They can be used to fold the computation without interfering with the output signals: by running through the collision rules and cancelling any action that the folding structure would have on them, they become insensible to the folding. Inside the structure, the computation is scaled down, generating an infinite acceleration (used to emulate the black hole in Durand-Lose [2006, 2009]). This way in finite time, the whole infinite output is generated and exits the structure as displayed on Fig. 5.

Inside the folding structure, the Turing machine has an infinite time ahead of it. The structure *preserves ratios* in the understanding that there are infinitely many pieces at different scales that reform the original space-time diagram after rescaling and translating. This means that even though the intermediate representation suffers multiple rescaling, it still generates the same infinite sequence, and the TM the same output.

The output is an infinite convoy in bounded space. This uses the extended context. There is a companion singularity ✺ (on the top of it).

4.2 CA-representation to intermediate representation

Considering the whole structure: no accumulation ❄ results from the folding structure nor from any "turn" from the infinite convoy because each time in the

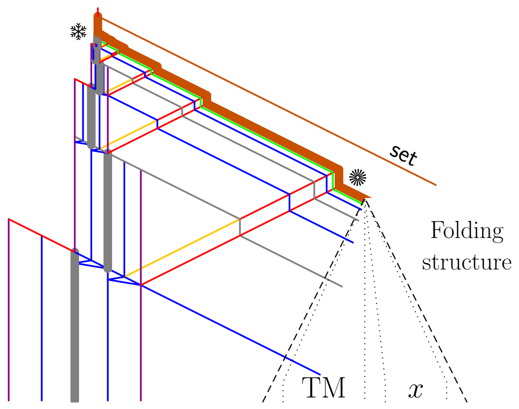


Fig. 5. Global structure.

configuration, infinitely many signals accumulate there; so ✺ prevents ❄ even though collisions are accumulating. The intermediate structure also produces an accumulation of collisions; but this time, there is no signal around it. So that one ❄ is generated and immediately turns to $\mu_{❄}$. This $\mu_{❄}$ is located exactly where e should be and the speed of $\mu_{❄}$ is chosen null. This is illustrated on figures 4(d) and 5.

A signal set is sent to turn $\mu_{❄}$ into e so that the two signals are distinct. Sending this signal is not complicated. Indeed the space-time location of the last accumulation is bounded easily: the ✺ path starts from the top of the folding structure (this location is known from construction), then the movement of the infinite convoy is the sum of a left translation and the sum of a geometric series.

5 Conclusion

Theorem 1. *With a proper handling of accumulations of collisions and of signals, it is possible to compute any function of computable analysis. Moreover, even if the computation involves infinitely many signals present simultaneously in a bounded space, there are finitely many signals in the initial and final configurations.*

There is no hidden oracle. Although speeds may be any real and thus encode information, only a few integral values are used. And apart from intermediate representation of ε, the distance between signals are also proportional with rational ratio. Nevertheless such "extra information" could be provided to compute at different level of CA-hierarchies.

Not only does the intermediate representation use finitely many signals, but is also almost directly reusable to do analog computation in the Blum-Shub-Smale understanding (as presented in [Durand-Lose, 2007, 2008]). This leads to consider even more powerful analog computational system since CA and BSS are not comparable, *e.g.* analytic machines [Chadzelek and Hotz, 1999].

It should also be possible to use higher order accumulation to hyper-compute as is done for the BSS model in Durand-Lose [2009] and to do some real hyper-computing [Ziegler, 2007] (although, just by being able to do linear BSS computation without the extended context, or with the intermediate representation, it can already compute the sign in finite time!).

Bibliography

L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21(1):1–46, 1989.

O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.

T. Chadzelek and G. Hotz. Analytic machines. *Theoret. Comp. Sci.*, 219(1-2):151–167, 1999. doi: 10.1016/S0304-3975(98)00287-4.

J. Durand-Lose. Abstract geometrical computation 1: Embedding black hole computations with rational numbers. *Fund. Inf.*, 74(4):491–510, 2006.

J. Durand-Lose. Abstract geometrical computation and the linear Blum, Shub and Smale model. In S. B. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE '07)*, number 4497 in LNCS, pages 238–247. Springer, 2007.

J. Durand-Lose. Abstract geometrical computation with accumulations: Beyond the Blum, Shub and Smale model. In A. Beckmann, C. Dimitracopoulos, and B. Löwe, editors, *Logic and Theory of Algorithms, 4th Conf. Computability in Europe (CiE '08) (abstracts and extended abstracts of unpublished papers)*, pages 107–116. University of Athens, 2008.

J. Durand-Lose. Abstract geometrical computation 3: Black holes for classical and analog computing. *Nat. Comput.*, 2009. doi: 10.1007/s11047-009-9117-0.

G. Etesi and I. Németi. Non-Turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.

A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–77, 1957.

M. L. Hogarth. Non-Turing computers and non-Turing computability. In *Biennial Meeting of the Philosophy of Science Association*, pages 126–138, 1994.

K.-I. Ko. *Computational Complexity of Real Functions.* Birkhäuser, 1991.

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

K. Weihrauch. *Introduction to computable analysis.* Texts in Theoretical computer science. Springer, Berlin, 2000.

M. Ziegler. (Short) Survey of real hypercomputation. In S. B. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe, CiE '07*, volume 4497 of *LNCS*, pages 809–824. Springer, 2007.