Computing over the Reals: Where Turing Meets Newton

Lenore Blum

The classical (*Turing*) *theory of computation* has been extraordinarily successful in providing the foundations and framework for theoretical computer science. Yet its dependence on 0s and 1s is fundamentally inadequate for providing such a foundation for modern scientific computation, in which most algorithms—with origins in Newton, Euler, Gauss, et al.—are real number algorithms.

In 1989, Mike Shub, Steve Smale, and I introduced a theory of computation and complexity over an arbitrary ring or field R [BSS89]. If R is $\mathbb{Z}_2 = (\{0,1\}, +, \cdot)$, the classical computer science theory is recovered. If R is the field of real numbers $\mathbb{R}$, *Newton's algorithm*, the paradigm algorithm of numerical analysis, fits naturally into our model of computation.

Complexity classes P, NP and the fundamental question "Does $P = NP$?" can be formulated naturally over an arbitrary ring R. The answer to the fundamental question depends in general on the complexity of deciding feasibility of polynomial systems over R. When R is $\mathbb{Z}_2$, this becomes the classical *satisfiability* problem of Cook–Levin [Cook71, Levin73]. When R is the field of complex numbers $\mathbb{C}$, the answer depends on the complexity of *Hilbert's Nullstellensatz*.

The notion of *reduction* between problems (e.g., between traveling salesman and satisfiability) has been a powerful tool in classical complexity theory. But now, in addition, the *transfer of complexity* results from one domain to another becomes a real possibility. For example, we can ask: Suppose we can show $P = NP$ over $\mathbb{C}$ (using all the mathematics that is natural here). Then, can we conclude that $P = NP$ over another field, such as the algebraic numbers, or even over $\mathbb{Z}_2$? (Answer: Yes and essentially yes.)

In this article, I discuss these results and indicate how basic notions from numerical analysis such as *condition*, round-off, and approximation are being introduced into complexity theory, bringing together ideas germinating from the real calculus of Newton and the discrete computation of computer science. The canonical reference for this material is the book *Complexity and Real Computation* [BCSS98].

Two Traditions of Computation

The two major traditions of the theory of computation have, for the most part, run a parallel nonintersecting course. On the one hand, we have numerical analysis and scientific computation; on the other hand, we have the tradition of computation theory arising from logic and computer science.

Fundamental to both traditions is the notion of *algorithm*. Newton's method is the paradigm example of an algorithm cited most often in numerical analysis texts. The Turing machine is the underlying model of computation given in most computer science texts on algorithms. Yet Newton's

Lenore Blum is Distinguished Career Professor of Computer Science at Carnegie Mellon University. Her email address is `lblum@cs.cmu.edu`.

This article is based on the AWM Noether Lecture given by Lenore Blum at the Joint Mathematics Meetings in San Diego, January 2001.

Supported in part by NSF grant # CCR-0122581. The author is grateful to the Toyota Technological Institute at Chicago for the gift of space-time to write this article.

method is not discussed in these computer science texts, nor are Turing machines mentioned in texts on numerical analysis.

Numerical Analysis/ Scientific Computation	Logic/Computer Science
• Newton's Method, Paradigm Example in Most Numerical Analysis Texts	• Turing Machine, Underlying Model in most Computer Science Texts on Algorithms
• No Mention of Turing Machines*	• No Mention of Newton's Method
• Real & Complex #'s	• 0's & 1's (bits)*
• Math is Continuous	• Math is discrete
• Problems are *Classical*	• Problems are *Newer*
• Major conference: FoCM (Foundation of Computational Mathematics)	• Major conference: FOCS (Foundation of Computer Science)
🚩 * No formal model of computation or systematic Complexity Theory	🚩 * Everything coded by bits, unnatural for problems of numerical analysis.

More fundamental differences arise with the distinct underlying spaces, the mathematics employed, and the problems tackled by each tradition. In numerical analysis and scientific computation, algorithms are generally defined over the reals or complex numbers, and the relevant mathematics is that of the continuum. On the other hand, 0s and 1s are the basic bits of the theory of computation of computer science, and the mathematics employed is generally discrete. The problems of numerical analysts tend to come from the classical tradition of equation solving and the calculus. Those of the computer scientist tend to have more recent combinatorial origins. The highly developed theory of computation and complexity theory of computer science in general is unnatural for analyzing problems arising in numerical analysis, yet no comparable formal theory has emanated from the latter.

One aim of our work is to reconcile the dissonance between these two traditions, perhaps to unify, but most important, to see how perspectives and tools of each can inform the other.

We begin with some *background* and *motivation*, then we present our *unifying model* and *main complexity results,* and, finally, we see *Turing* meet *Newton* and fundamental links introduced.

Background

The motivation for logicians to develop a theory of computation in the 1930s had little to do with computers (think of it, aside from historical artifacts, there were no computers around then). Rather, Gödel, Turing, et al. were groping with the question: "What does it mean for a *problem* or set S ($\subset$ universe X) to be *decidable*?" For example, how can one make precise Hilbert's Tenth Problem?

Example. *Hilbert's Tenth Problem.* Let $X = \{f \in \mathbb{Z}[x_1, \dots, x_n]\, n > 0\}$ and $S = \{f \in X | \exists \zeta \in \mathbb{Z}^n, f(\zeta_1, \dots, \zeta_n) = 0\}$. Is S decidable? That is, can one decide by finite means, given a diophantine polynomial, whether or not it has an integer solution? (Actually, Hilbert's challenge was: Produce such a decision procedure.)

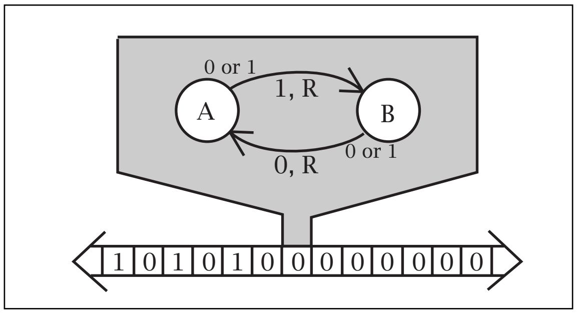
The logicians' subsequent formalization of the notion of *decidability* has had profound consequences.

Definition. A set $S (\subset X)$ is *decidable* if its characteristic function χ_S (with values 1 on S, 0 on $X - S$) is computable (in finite time) by a *machine*.

Such a 0-1 valued machine is called a *decision procedure* for S. On *input* $x \in X$ it answers the question "Is $x \in S$?" with *output* 1 if YES and 0 if NO. Here X is Σ^*, the set of finite but unbounded sequences over a finite set Σ. We will also allow X to be a decidable subset of Σ^* (e.g., as would be the case for the set of all diophantine polynomials embedded in $\{0, 1\}^*$ via some natural coding). N.B. Σ^* is countable.

To complete the definition, many seemingly different machines were proposed. What has been striking is that all gave rise to the *exact same class* of "computable" functions. This gives rise to the belief, known as *Church's Thesis*, that the *computable functions* form a natural class and any informal notion of procedure or *algorithm* can be realized within any of the formal settings.

In 1970, Yuri Matijasevich answered Hilbert's Tenth Problem in the negative by showing the associated characteristic function χ_S is not Turing computable and hence, by Church's Thesis, no procedure exists for deciding the solvability in integers of diophantine equations. (Following the program mapped out earlier by Martin Davis, Hi-



The Turing Machine (with simple operations, finite number of states, finite program, and infinite tape) provides a mathematical foundation for the classical Theory of Computation (originated by logicians in the 1930s and 40s) and Complexity Theory (originated by computer scientists in the 1960s and 70s). The program of this 2-state TM instructs the machine "if in state A with a 0 or 1 in the current scanned cell, then print a 1, move R(ight) and go into state B; if in state B, print 0, move R and go into state A." That the long-term behavior of this machine is discernible is not the norm. (TM picture, courtesy of Bryan Clair.)

lary Putnam, and Julia Robinson, Matijasevich needed only to show the existence of a diophantine relation of exponential growth [DavisMatijasevicRobinson76].)

Since the initial startling results of the 1930s of the undecidability of the true statements of arithmetic and the halting problem for Turing machines, logicians have focused attention on classifying problems as to their decidability or undecidability. Considerable attention was placed on studying the hierarchy of the undecidable. Decidable problems, particularly finite ones, held little interest. In the 1960s and 70s, computer scientists started to realize that not all such decidable problems were alike; indeed, some seemed to defy feasible solution by machine.

Example. *The SATisfiability Problem* (SAT). Here,

$$X = \{f | f : \mathbb{Z}_2^n \to \mathbb{Z}_2\}$$

is the set of Boolean functions and

$$S = \{f \in X | \exists \zeta \in \mathbb{Z}_2^n, f(\zeta_1, \dots, \zeta_n) = 0\}.$$

It is assumed X is embedded in $\{0,1\}^*$ via some *natural* encoding. Systematically testing all possible 2^n arguments for a given Boolean function f clearly yields a decision procedure for S. This procedure takes an exponential number of basic operations in the worst case. We do not know if SAT is *tractable*, i.e., if there is a *polynomial time* decision procedure for S.

Definition. The *decision problem* (X, S) is in *class P* if the characteristic function χ_S is *polynomial time* computable, i.e., computable by a *polynomial time* Turing machine. A *polynomial time* Turing machine is one that halts in $c(\mathit{size}\ x)^k$ Turing operations for some fixed $c, k \geq 0$ and all inputs $x \in X$. Here *size* x is the length of the sequence x, i.e., the bit length if $\Sigma = \{0,1\}$.

As was the case for computable functions, the polynomial time functions, the class P, and subsequently defined *complexity classes*, form natural classes independent of machine.[1] Thus again, computer scientists have confidence they are working with a very natural class of functions and feel justified employing their favorite model.

In the early 1970s Steve Cook and Leonid Levin [Cook71, Levin73] independently formulated and answered the question about the tractability of SAT with another question: Does $P = NP$?[2]

A decision problem $(X, S) \in \mathit{classNP}$ if for each $x \in S$ there is a polynomial time verification of this fact. Later we will formalize the definition of NP, but meantime we note that SAT $\in NP$: If a Boolean function $f \in S$, then there is a *witness* $\zeta \in \mathbb{Z}_2^n$ that provides, together with the computation of f on argument $\zeta = (\zeta_1, \dots, \zeta_n)$ producing value 0, a polynomial time verification. If $f \notin S$, then no such witness will do.

The significance of the $P = NP$? question became clear when Dick Karp showed that the tractability of each of twenty-one seemingly unrelated problems was equivalent to the tractability of SAT [Karp72]. The number of such problems known today is legion.

As did the earlier questions about decidability, the $P = NP$? question has had profound consequence. The apparent dichotomy between the classes P and NP has been the underpinning of some of the most important applications of complexity theory, such as to cryptography and secure communication. Particularly appealing here is the idea of using hard problems to our advantage.

Thus the classical Turing tradition has yielded a highly developed and rich (invariant) theory of computation and complexity with essential applications to computation—and deep interesting questions. Why do we want a new model of computation?

Motivation for Model

Decidability over the Continuum: Is the Mandelbrot Set Decidable?

> Now we witnessed a certain extraordinarily complicated-looking set, namely the Mandelbrot set. Although the rules which provide its definition are surprisingly simple,[3] the set itself exhibits an endless variety of highly elaborate structure. *Could this be an example of a non-recursive* [*i.e. undecidable*] *set, truly exhibited before our mortal eyes*?[4]
>
> —Roger Penrose, *The Emperor's New Mind* [Penrose89].

Classically, decidability is defined only for countable sets. The Mandelbrot set is uncountable. So the Mandelbrot set is not decidable classically. But clearly this is not a satisfactory argument. So how do we reasonably address Penrose's question?

From time to time, logicians and computer scientists *do* look at problems over the reals or complex numbers. One approach has been through "recursive analysis," which has its origins in Turing's seminal 1936 paper [Turing36–37].[5] In the first paragraph, Turing defines "a number [to be] computable

[1] *A main proviso is that integers are not coded in unary.*

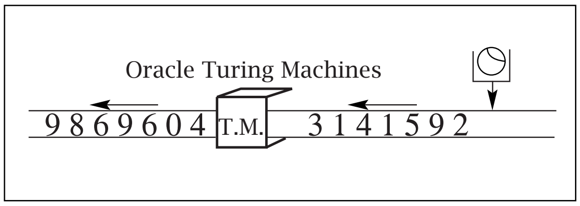
[2] *I am rephrasing their result. The usual statement is: SAT is NP-complete.*

[3] *The Mandelbrot set M is the set of all parameters $c \in \mathbb{C}$, such that the orbit of 0 under the quadratic map $p_c(z) = z^2 + c$ remains bounded.*

[4] *Emphasis mine.*

[5] *Here Turing first defines* automatic machines *and shows the undecidability of the halting problem.*

if its decimal can be written down by a machine." Turing later defines computable functions of computable real numbers.[6] One can then imagine an *oracle* Turing machine that, when fed a real number by oracle, decimal by decimal, outputs a real number decimal by decimal. A refinement of this notion forms the basis of *recursive analysis.*



Another tack taken by computer scientists is what might be called the "rational number model" approach. The approach is not formalized, but its reasoning goes as follows:

A. Machines are finite.

B. Finite approximations of reals are rationals.

C. $\therefore$ We are really looking at problems over the rationals.

If we are totally naïve here, we quickly run into trouble. The rational skeleton of the curve $x^3 + y^3 = 1$ on the positive quadrant is hardly informative.[7]

We have even more serious concerns when this approach is used in complexity theory. Computer scientists measure complexity as a function of input word size (in bits). But small perturbations (of input) can cause large differences in word size. For example, a small perturbation of an input 1 to $1 + 1/2^n$ causes the word size to grow from 1 to $n + 1$. Thus an algorithm that is polynomial time according to the discrete model definition would be allowed to take considerably more time on a perturbed input than on a given input. If the problem instance were well conditioned, this clearly would not be acceptable. An issue here is that the Euclidean metric is very different from the bit metric. Another is *condition.*

Not paying attention to these issues has caused both incompleteness in the analysis and confusion in the comparison of different algorithms over the reals. The comparison of competing algorithms for the Linear Programming Problem provides a case in point. We shall return to this example again.

Penrose explores similar scenarios for posing his question but in the end "... is left with the strong feeling that the correct viewpoint has not yet been arrived at."

The Mandelbrot example is perhaps too exotic to draw generalizations. We turn now to a decision problem ubiquitous in mathematics.

The Hilbert Nullstellensatz/R

Given a system of polynomial equations over a ring R, $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$. Is there $\zeta \in R^n$, such that $f_1(\zeta) = \dots = f_m(\zeta) = 0$?

We call the corresponding decision problem over R, HN_R. If R is $\mathbb{Z}_2, \mathbb{Z}$, or the rational numbers $\mathbb{Q}$, HN_R fits naturally into the Turing formalism (via bit coding of integers and rationals). The corresponding decision problem over $\mathbb{Z}_2$ is essentially SAT (decidable but not known to be in P), over $\mathbb{Z}$ it is Hilbert's Tenth Problem (undecidable) and over $\mathbb{Q}$ it is not known to be decidable or undecidable. If R is the real field $\mathbb{R}$ or complex numbers $\mathbb{C}$, then HN_R does not fit naturally into the Turing formalism.

An even simpler example is the high school algorithm for deciding whether or not a real polynomial $ax^2 + bx + c$ has a real root. We just check if the discriminant $b^2 - 4ac \geq 0$. We do not stipulate that a, b, and c be rational or be fed to us bit by bit—or question if we can tell whether or not a real number equals 0. We just work with the basic arithmetic operations and comparisons of an ordered ring or field.

More generally, we have perfectly good algorithms for deciding HN_R over $\mathbb{R}$ and $\mathbb{C}$. Recall:

Hilbert's Nullstellensatz, HN [Hilbert1893]

Given $f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n) \in \mathbb{C}[x_1, \dots, x_n]$. Then, $f_1 = \dots = f_m = 0$ is *not* solvable over $\mathbb{C} \Leftrightarrow$

$$(*) \qquad 1 = \Sigma g_i f_i, \text{ for some polynomials } g_i \in \mathbb{C}[x_1, \dots, x_n].$$

This theorem provides a *semidecision procedure* for the complement of HN_C: Given $f_1, \dots, f_m \in \mathbb{C}[x_1, \dots, x_n]$, systematically search for g_i's to solve (*). If found, then $f_1 = \dots = f_m = 0$ is *not* solvable over $\mathbb{C}$ and so output 0. (The search can be done by considering, for each successive D, general polynomials g_i of degree D with indeterminate coefficients. Checking if there exist coefficients satisfying (*) reduces to solving $\sim D^n$ linear equations over $\mathbb{C}$.)

However, if $f_1 = \dots = f_m = 0$ is solvable, then no such g_i's will ever be found. Fortunately, we have *stopping rules.* In 1926, Grete Hermann, a student of Emmy Noether, gave an effective upper bound D $(= d \uparrow (2^n))$ where $d = \max(3, \deg f_i))$ on the degrees of the g_i's that one need consider [Hermann26].[8] If no solution is found for generic g_i's

[6]*Although Turing states in this paper that "a development of the theory of functions of a real variable expressed in terms of computable numbers" would be forthcoming, I am not aware of any further such development by him in this direction.*

[7]*By Fermat, the only rational points on the curve are* $(0, 1)$ *and* $(1, 0)$.

[8]*By Brownawell and Kollar, we only need check the case* $D = d^n$, *which by Masser and Philippon is optimal. (See [Yger01] for discussion and [KPS01] for refinements.)*

of degree D, then none exists, and so we can output 1.

This *decision procedure*, using arithmetic operations and comparisons on complex numbers, inspires us to take a different tack. Rather than forcing artificial coding of problems into bits, we propose a model of computation that computes over a ring (or field), using basic algebraic operations and comparisons on elements of the ring (or field). Thus we have our *first motivation* for proposing a new model.

Algorithms of Numerical Analysis

Our *next motivation* comes directly from the tradition of numerical analysis. We start with the paper, "Rounding-off errors in matrix processes" in the *Quarterly Journal of Mechanics and Applied Mathematics*, vol. I, 1948, pp. 287–308 [Turing48]. Written by Alan Turing while he was preoccupied with solving large systems of equations, this paper is quite well known to numerical analysts but almost unknown by logicians and computer scientists. Its implicit model of computation is more closely related to the former than the latter.

In the first section of his paper, Turing considers the "measures of work in a process":

> It is convenient to have a *measure of the amount of work involved in a computing process*, even though it be a very crude one.... We might, for instance, *count the number of additions, subtractions, multiplications, divisions, recordings of numbers*...[9]

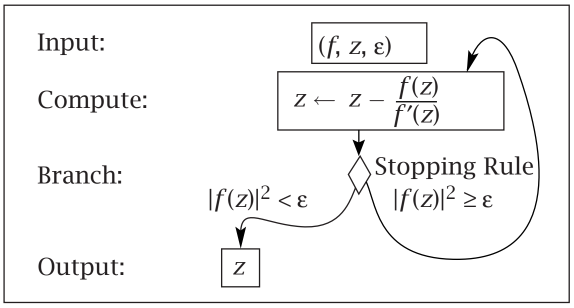
From this point of view, it is again natural to start with a model of computation in which real numbers are viewed as entities, and algebraic operations and comparisons, as well as simple accessing, are each counted as a unit of work. We will return again to this paper to motivate refinements of these initial "measures of work."

We also want a model of computation that is more natural for *describing* algorithms of numerical analysis, such as Newton's method for finding zeros of polynomials. Here, given a polynomial $f(z)$ over $\mathbb{R}$ or $\mathbb{C}$, the basic operation is the Newton map, $N_f(z) = z - f(z)/f'(z)$, which is iterated until the current value satisfies some proscribed stopping rule. Translating to bit operations would wipe out the natural structure of this algorithm.

Does *P = NP*?

In order to gain new perspective and access additional tools, mathematicians often find it profitable to view problems in a broader framework

[9]*Emphasis mine.*



A "machine" for executing Newton's method.

than originally posed. We are thus *motivated* to follow this path for the $P = NP$? problem.

The Model: Machines over a Ring or Field *R* [BSS89]

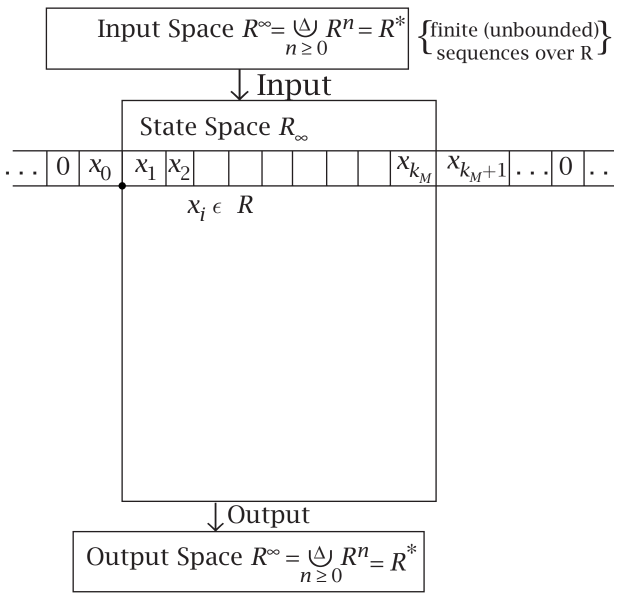
We suppose R is a commutative ring or field (possibly ordered) with unit. A *machine M over R* has the following properties:

Associated with M is an *input space* and an *output space*, both R^∞ (the disjoint union of R^n, $n \geq 0$). At the top level, our machine M is similar to a Turing machine. M has a 2-way *infinite tape* divided into *cells* and a *read-write* head that can *view* a fixed number k_M of contiguous cells at a time.

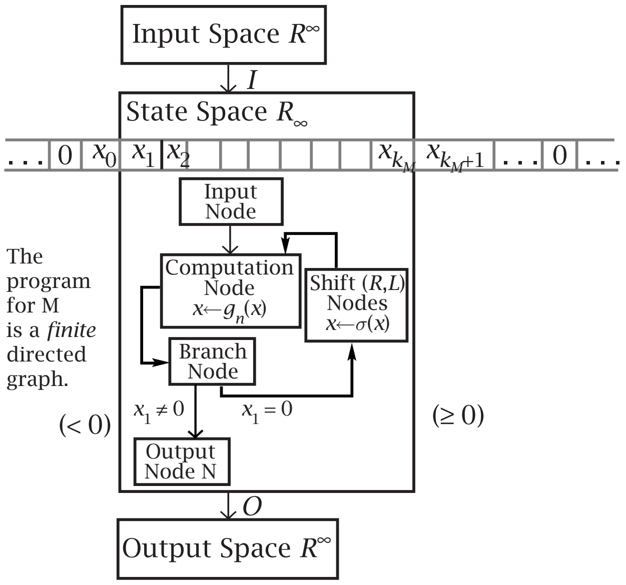
Internal to M is its *program*, a finite directed graph with 5 types of nodes, each with associated *operations* and *next node maps*:

- The operation g_ι associated with the *input node* ι takes elements $x = (x_1, \ldots, x_k)$ from the input space R^∞ and puts each x_i ($i = 1, \ldots, k$) in successive tape cells, starting with the leftmost one in M's view. There is a *unique* next node $\iota' \neq \iota$.
- Each *computation node* η has a built-in polynomial or rational map $g_\eta : R^n \to R^m$ with $n, m \leq k_M$.[10] Given elements $x_1, \ldots, x_n$ in the first n cells of M's view, the associated operation, also called g_η, puts $g_\eta^j(x_1, \ldots, x_n)$ in the jth cell in M's view ($j = 1, \ldots, m$). There is a unique next node η'.
- For each *branch node* η, the associated operation is the identity. There are *two* possible next nodes η'_L and η'_R depending on the leftmost element x_1 in M's view. If $x_1 = 0$ (≥ 0, if R is ordered), then $\eta' = \eta'_R$. If $x_1 \neq 0$ ($x_1 < 0$), then $\eta' = \eta'_L$.
- For each *shift node* σ, the associated map is the identity and there is a *unique* next node σ'. Right shift nodes σ_R shift M's view one cell to the right, left shift nodes σ_L shift one cell to the left.
- The operation g_N associated with the *output node N outputs* (by projection) the contents of the tape into R^∞. N has *no* next node.

[10]*Machines over R can thus have a* finite number of built-in constants *from R*.



Input Space $R^\infty = \biguplus_{n\geq 0} R^n = R^*$
finite (unbounded) sequences over R
Input
State Space R_∞
$x_i \in R$
Output
Output Space $R^\infty = \biguplus_{n\geq 0} R^n = R^*$



A machine over a ring or field *R*, top-level and internal views.

The *computable functions over R* are the *input-output* maps ϕ_M of machines M over R. Thus, for $x \in R^\infty$, $\phi_M(x)$ is defined if the output node N is reachable by following M's program on input x. If so, $\phi_M(x)$ is the *output* $y \in R^\infty$.

Although the machine's view at any time is finite, the shift nodes enable the machine to read and operate on inputs from R^n for all n, and thus we can model algorithms that are defined uniformly for inputs of any dimension. Noting this, we can also construct universal (programmable) machines over R. (We do not use Gödel coding. The machine program itself is (essentially) its own code.)

If R is $\mathbb{Z}_2$, we recover the classical theory of computation (and complexity, as we shall see). We also note that Newton's method is naturally implemented by a machine over $\mathbb{R}$.

Let's return now to questions of decidability over $\mathbb{R}$ and $\mathbb{C}$, which were so problematic before.

Definition. A problem *over R* is a pair, (X, X_{yes}), where $X_{\text{yes}} \subseteq X \subseteq R^\infty$. X consists of the *problem instances*, X_{yes}, the *yes-instances*.

For HN_R,

$$X = \{f = (f_1, \ldots, f_m) | f_i \in R[x_1, \ldots, x_n]\}, m, n > 0 \quad \text{and}$$

$$X_{\text{yes}} = \{f \in X | \exists \zeta \in R^n, \; f_i(\zeta_1, \ldots, \zeta_n) = 0, i = 1, \ldots, m\}.$$

Finite polynomial systems over R can be coded as elements of R^∞ (by systematically listing coefficients); thus X can be viewed as a subspace of R^∞.

Definition. A problem over R, (X, X_{yes}), is *decidable* if the characteristic function of X_{yes} in X is computable over R.

Thus, in this framework, we can state problems over $\mathbb{R}$ and $\mathbb{C}$ (or any ring or field) and ask questions of decidability. The algorithm presented earlier, based on Hilbert's Nullstellensatz with effective bounds, is easily converted to a decision machine over $\mathbb{C}$, and so $\text{HN}_{\mathbb{C}}$ is decidable over $\mathbb{C}$. Similarly, $\text{HN}_{\mathbb{R}}$ is decidable over $\mathbb{R}$ (by Seidenberg's elimination theory [Seid54]).

We can also now formally state, and answer, Penrose's question about the Mandelbrot set M. Here X is $\mathbb{R}^2$ and X_{yes} is M. (In order to allow algorithms that compare magnitudes, we are viewing $\mathbb{C}$ as $\mathbb{R}^2$.)

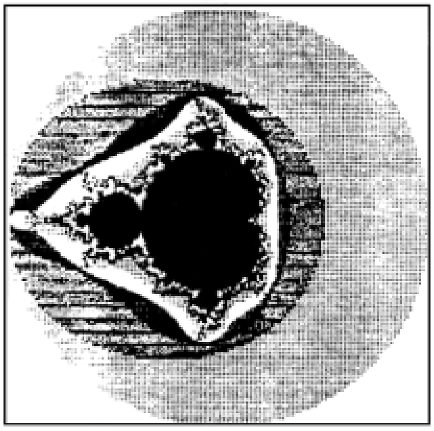
Theorem [BlumSmale93]. The Mandelbrot set M is undecidable over $\mathbb{R}$.

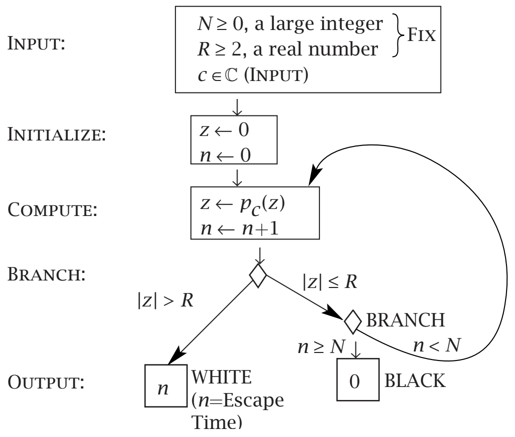
The proof uses the fact that the boundary of a closed semidecidable set in $\mathbb{R}^2$ has Hausdorff dimension at most 1, whereas the Hausdorff dimension of the boundary of the Mandelbrot set is 2 [Shishikura91].

It turns out that the complement of the Mandelbrot set M is *semidecidable* over $\mathbb{R}$. To see this, a little arithmetic shows that $M = \{c \in \mathbb{C}$ such that the sequence $c, c^2 + c, (c^2 + c)^2 + c, \ldots$ stays within the circle of radius $2\}$. Hence, if at some point the orbit of 0 under the map $z^2 + c$ escapes the circle of radius 2, we can be *certain* that c is in the complement of M. One can use this fact to "draw" the Mandelbrot set.

Complexity Classes and Theory over a Ring *R*

Following the classical tradition, we measure *time* (or *cost*) as a function of input word size. Suppose $x \in R^\infty$. We define $size(x)$ to be the vector length of x, thus $size(x) = n$ if $x \in R^n$. For machine M over R and input x, we define $T_M(x)$ to be the number of nodes traversed from input to output when M is input x. T_M is our measure of *time* or *cost*. So, if $R = \mathbb{Z}_2$ $size(x)$ is the *bit size* of x, and $T_M(x)$ is the *bit cost* of the computation.





Machine to "draw" the Mandelbrot set *M* (Fact: *M* = {c $\in \mathbb{C}$ | $|p_c^n(0)| \geq 2$, all n > 0} where $p_c(z) = z^2 + c$)

Definition. $(X, X_{\text{yes}}) \in P_R$ (*class P over R*) if there exists a decision machine M and a polynomial $p \in \mathbb{Z}[x]$ such that for each $x \in X$, $T_M(x) < p(\text{size}(x))$. M is called a *polynomial time* (*decision*) *machine*.

Definition. $(X, X_{\text{yes}}) \in NP_R$ (*class NP over R*) if there exists $(Y, Y_{\text{yes}}) \in P_R$ and a polynomial $p \in \mathbb{Z}[x]$ such that, for each $x \in X$, $x \in X_{\text{yes}} \Leftrightarrow \exists$ *witness* $w \in R^{p(\text{size}(x))}$ such that $(x, w) \in Y_{\text{yes}}$.

In the above, we generally require that $(R^\infty, X) \in P_R$. Then, if $R = \mathbb{Z}_2$, we recover the *classical* complexity classes, that is, class P over $\mathbb{Z}_2$ is the classical class P, and class NP over $\mathbb{Z}_2$ is the classical *class NP*. Whenever we omit subscripts in the following, we are referring to the case when $R = \mathbb{Z}_2$, the classical case.

Example. $\text{HN}_R \in \text{NP}_R$. To see this let

$$Y = \{(f, \zeta) | f = (f_1, \dots, f_m), f_i \in R[x_1, \dots, x_n], \zeta \in R^n, m, n > 0\}$$

and $Y_{\text{yes}} = \{(f, \zeta) \in Y | f_i(\zeta_1, \dots, \zeta_n) = 0, i = 1, \dots, m\}$.

Then, since polynomial evaluation is a polynomial time computation over R,[11] we have $(Y, Y_{\text{yes}}) \in P_R$ over R. Also, given $f \in X$, $f = (f_1, \dots, f_m)$, $f_i \in R[x_1, \dots, x_n]$, $f \in X_{\text{yes}} \Leftrightarrow \exists \zeta \in R^n$ such that $f_i(\zeta_1, \dots, \zeta_n) = 0$, $i = 1, \dots, m$, that is, $(f, \zeta) \in Y_{\text{yes}}$.

We note that there is an exponential time decision machine for HN over $\mathbb{Z}_2$, but as mentioned earlier, we do not know if HN $\in P$. On the other hand, since $\text{HN}_{\mathbb{Z}}$ is not decidable (over $\mathbb{Z}$), it certainly is not in $P_{\mathbb{Z}}$, thus $P_{\mathbb{Z}} \neq NP_{\mathbb{Z}}$.

Main Complexity Results

We recall that

Theorem [Cook71, Levin73].

$$P = NP \Leftrightarrow \text{SAT} \in P \text{ and}$$

Theorem [Karp72].

$$P = NP \Leftrightarrow \text{TSP}^{12} \in P \Leftrightarrow X \in P,$$

where X = Hamilton Circuit or any of nineteen other problems.

We prove

Theorem [BSS89].

$$P_R = NP_R \Leftrightarrow \text{HN}_R \in P_R \text{ for } R = \mathbb{Z}_2, \mathbb{R}, \mathbb{C},$$

or for any field R, unordered or ordered.

To prove this theorem, we show that given any problem $(X, X_{\text{yes}}) \in \text{NP}_R$ and instance $x \in X$, we can code x (in polynomial time) as a polynomial system f_x over R such that $x \in X_{\text{yes}} \iff f_x$ has a zero over R. In other words, we give a *polynomial time reduction* from any problem in class NP_R into HN_R. This is done by writing down the equations for the *computing endomorphism* of an NP_R machine.

So, HN_R is a universal *NP-complete* problem. We know that $\text{HN}_R \in EXP_R$ for $R = \mathbb{R}$ and $\mathbb{C}$. That is, there are exponential time algorithms for deciding the solvability of polynomial systems over $\mathbb{R}$ and $\mathbb{C}$ [Renegar92, BPR96]. But, again, no polynomial time algorithms are known.

So, in addition to the classical $P = NP$? question, we pose two new ones: Is $P_{\mathbb{R}} = NP_{\mathbb{R}}$? Is $P_{\mathbb{C}} = NP_{\mathbb{C}}$?

Understanding the complexity of the Hilbert Nullstellensatz thus plays a central role in complexity theory over R.[13]

Transfer Principles

In the preface to *Complexity and Real Computation* [BCSS98], Dick Karp speculates about the transferability of complexity results from one domain to another:

[11] *It should be clear what formal definitions we are supposing here. The proof requires a construction.*

[12] *The Traveling Salesman Problem (TSP) is generally stated as a search problem: Find the shortest (cheapest) path traversing all nodes. To view TSP as a decision problem, we introduce bounds: Given k is there a path of length (cost) at most k traversing all nodes?*

[13] *Toward this goal, a sequence of five papers by Mike Shub and Steve Smale on the related Bezout's Theorem presents a comprehensive analysis of the complexity of solving polynomial systems approximately and probabilistically.*

> It is interesting to speculate as to whether the questions of *whether* $P_{\mathbb{R}} = NP_{\mathbb{R}}$ and *whether* $P_{\mathbb{C}} = NP_{\mathbb{C}}$ are related to each other and to the *classical* P versus NP question. . . .I am inclined to think that the three questions are very different and need to be attacked independently.. . .[14]

One reason for the skepticism is that over $\mathbb{R}$ or $\mathbb{C}$ one can quickly build numbers of large magnitude, for example by successive squaring. And over $\mathbb{R}$ or $\mathbb{C}$, arithmetic operations on numbers of any magnitude can be done in one step. Hence, polynomial time machines over $\mathbb{R}$ or $\mathbb{C}$ might be able to decide inherently hard discrete problems by "cheating", e.g., by quickly coding up an exponential amount of information within large numbers and subsequently getting an exponential amount of essential bit operations accomplished quickly.

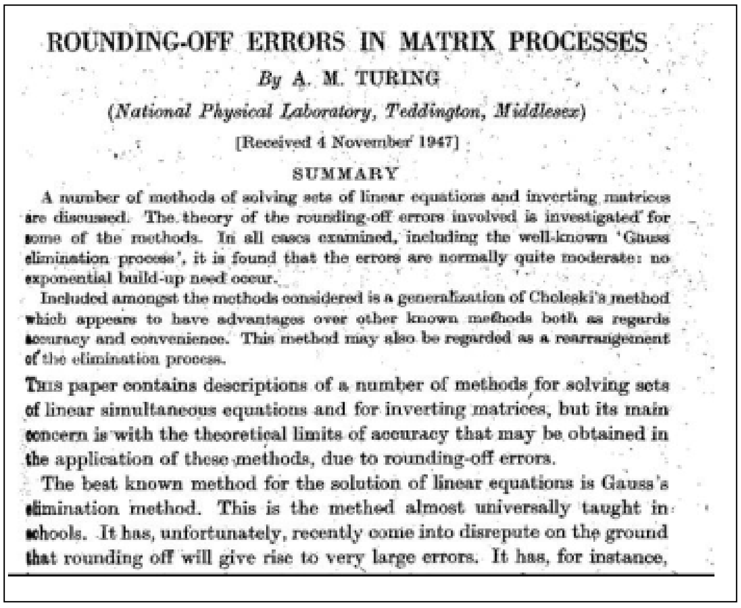
In our book we present a number of *transfer results* for complexity theory, one of which transfers the $P_R = NP_R$? question across algebraically closed fields of characteristic 0.

Theorem [BCSS96/98]. $P_{\mathbb{C}} = NP_{\mathbb{C}} \Leftrightarrow P_K = NP_K$, where K is any algebraically closed field of characteristic 0, for example the field of algebraic numbers.

The transfer of the $P_R = NP_R$? problem from the algebraic numbers to the complexes had been proved earlier by Christian Michaux, using *model theoretic* techniques [Michaux94]. The other direction uses *number theory*. Here we show how a machine over $\mathbb{C}$ with built in algebraically independent constants can be simulated by a machine that has no such constants and that takes the same amount of time (up to a polynomial factor) to compute. We call this result the *elimination of constants*. The constants in a machine come into play at branch nodes where a decision is to be made as to whether or not the current x_1, which is a polynomial in the machine constants, is equal to 0. This polynomial is not presented to us in the standard form, but rather by a composition of the polynomials along the computation path, so we cannot in general tell quickly enough if the coefficients are all zero. Instead, we use the *Witness Theorem* to quickly generate algebraic *witnesses* with the property that, if the original constants are replaced by these witnesses, and the resulting evaluation is 0, then the original polynomial is 0. (The theory of *heights* comes into play here.)

Shortly after our book went to press, Steve Smale realized (after talking to Manuel Blum) that standard computer science arguments could yield a transfer result from $\mathbb{C}$ to the classical setting [Smale2000].

[14] *Emphasis mine.*


ROUNDING-OFF ERRORS IN MATRIX PROCESSES

By A. M. TURING

(*National Physical Laboratory, Teddington, Middlesex*)

[Received 4 November 1947]

SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known 'Gauss elimination process', it is found that the errors are normally quite moderate: no exponential build-up need occur.

Included amongst the methods considered is a generalization of Choleski's method which appears to have advantages over other known methods both as regards accuracy and convenience. This method may also be regarded as a rearrangement of the elimination process.

This paper contains descriptions of a number of methods for solving sets of linear simultaneous equations and for inverting matrices, but its main concern is with the theoretical limits of accuracy that may be obtained in the application of these methods, due to rounding-off errors.

The best known method for the solution of linear equations is Gauss's elimination method. This is the method almost universally taught in schools. It has, unfortunately, recently come into disrepute on the ground that rounding off will give rise to very large errors. It has, for instance,

From the *Quarterly Journal of Mechanics and Applied Mathematics*, vol. I, 1948, p. 287.

Let BPP be the class of problems over $\mathbb{Z}_2$ that can be solved in bounded error ($< 1/2$) probabilistic polynomial time. Class BPP is a practical modification of class P. Repeating a BPP algorithm k times produces a polynomial time algorithm with probability of error $< 1/2^k$. For example, BPP algorithms for Primality (testing) were known and used well before it was known that Primality was in class P.

Theorem. $P_{\mathbb{C}} = NP_{\mathbb{C}} \Rightarrow BPP \supseteq NP$.

The idea of the proof goes as follows. First we note that by adding polynomials of the form $x(x-1)$ to an instance $f \in$ HN we get an *equivalent* instance $f^* \in \mathrm{HN}_{\mathbb{C}}$. Then, any polynomial decision machine M over $\mathbb{C}$ for $\mathrm{HN}_{\mathbb{C}}$ will decide the solvability of f^* and hence the solvability of f over $\mathbb{Z}_2$. The trouble is, M might have a finite number of built-in complex constants. As before, they come into play at branch nodes where a decision is to be made as to whether or not a polynomial in these constants, presented by composition, is equal to 0. Rather than generate witnesses as before to eliminate constants, the decision is now made by probabilistically replacing these constants by a small number (by Schwartz's Lemma) of small numbers (by the Prime Number Theorem). Thus, given a polynomial time machine M over $\mathbb{C}$ for $\mathrm{HN}_{\mathbb{C}}$, we could construct a probabilistic polynomial time machine for HN.

Transfer results provide important connections between the two approaches to computing.[15] Underlying connections derive from the uniform distribution of rational data of bounded input length

[15] *Transfer results are also known for questions regarding other complexity classes such as* $PSPACE$ *[Koiran02].*

with respect to the probability distribution of condition numbers of numerical analysis [CMPS02].

Introducing Condition, Accuracy, and Round-off into Complexity Theory: Where Turing Meets Newton

We now return to the Turing paper on rounding-off errors referred to earlier [Turing48].

It is here that the notion of *condition* (of a linear system) was originally introduced. An illustrative example is presented on page 297:

$$\begin{aligned} 1.4x + 0.9y &= 2.7 \\ 0.8x + 1.7y &= -1.2 \end{aligned} \tag{8.1}$$

$$\begin{aligned} -0.786x + 1.709y &= -1.173 \\ -0.8x + 1.7y &= -1.2 \end{aligned} \tag{8.2}$$

> The set of equations (8.2) is fully equivalent to (8.1)[16], but clearly if we attempt to solve (8.2) by numerical methods involving rounding-off errors we are almost certain to get much less accuracy than if we worked with equations (8.1)...
>
> We should describe the equations (8.2) as an *ill-conditioned* set, or, at any rate, as ill-conditioned compared with (8.1).[17] It is characteristic of ill-conditioned sets of equations that small percentage errors in the coefficients given may lead to large percentage errors in the solution.

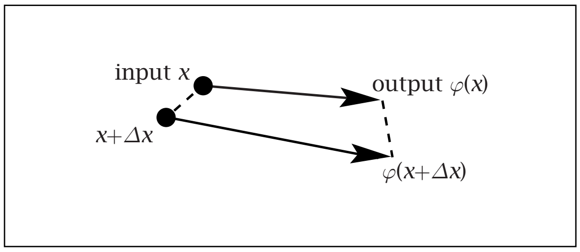
Turing's notion of condition, clearly inspired by Newton's derivative, links both traditions, particularly when considering questions of complexity.

Condition Numbers and Complexity

The *condition* of a problem (instance) measures how small perturbations of the input will alter the output. Introducing the notion of condition provides an important link between the two traditions of computing.

Definition (Turing). Suppose A is an $n \times n$ real matrix and $b \in \mathbb{R}^n$. The *condition number* of the linear system, $Ax = b$, is given by $\kappa(A) = \|A\|\,\|A_1\|$. Here $\|A\| = \max\{|Ay|/|y| | y \neq 0\}$ is the *operator norm* with respect to the *Euclidean norm* $|\;|$.

We note that $\kappa(A)$ is the worst-case relative condition for solving the system $Ax = b$ for x. Thus $\log^+ \kappa(A)$ provides a worst-case lower bound for the *loss of precision* in solving the system.[18] For



With appropriate norm, the ratio $\|\varphi(x+\Delta x) - \varphi(x)\|/\|\Delta x\|$, or *relative* ratio $(\|\varphi(x+\Delta x) - \varphi(x)\|/\|\varphi(x)\|)/(\|\Delta x\|/\|x\|)$ indicates the condition of an instance. If the quotient is large, the instance is *ill-conditioned* and so requires more accuracy, and hence more resources, to compute with small error.

computational purposes, ill-conditioned problem instances will generally require more input precision than well-conditioned instances.

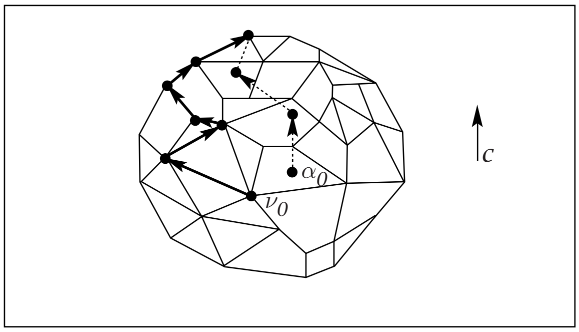
During the 1980s a number of people gave estimates on the average loss of precision for (solving) linear systems over $\mathbb{R}$.

Theorem [Edelman88]. Average $\log^+ \kappa(A) \sim \log n$.

The *log of condition* provides an *intrinsic* parameter to replace arbitrarily chosen bit *input word sizes* for *problem instances* where the underlying mathematical spaces are $\mathbb{R}$ or $\mathbb{C}$. And so a focus for complexity theory is to formulate and understand measures of condition.

My favorite example for illustrating the issues raised and the resolutions proposed is the Linear Programming Problem over R (LPP_R) alluded to earlier, where R is $\mathbb{Q}$ or $\mathbb{R}$. An instance of the LPP_R is to maximize a linear function $c \cdot x$ subject to the constraints $Ax \leq b$, $x \geq 0$, or to conclude no such maximum exists. The *data* here is (A, b, c), where A is an $m \times n$ matrix over R, $b \in R^m$ and $c \in R^n$.

Competing algorithms for solving the LPP_R are often posed and analyzed using distinct models of computation. (Also, there are various equivalent



The simplex method for the LPP optimizes by traversing vertices. The newer interior point methods follow a trajectory of centers.

[16] *The third equation is the second plus .01 times the first.*

[17] *Emphasis mine.*

[18] $\log^+ x = \log x$ *for* $x \geq 1$, *otherwise* $\log^+ x = 0$.

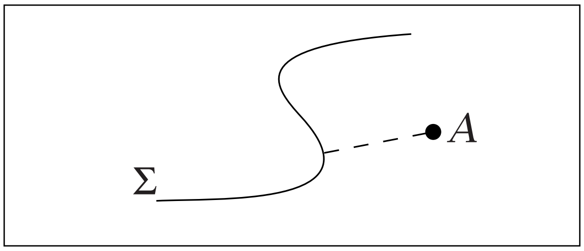
mathematical formulations of the LPP_R, not necessarily equivalent with respect to complexity theory.)

The *simplex* algorithm [Dantzig47/90], a long-time method of choice, is an algebraic algorithm that in the worst case takes exponential (in *n* and *m*) number of steps [KleeMinty72] given exact arithmetic. For rational inputs, simplex also is exponential in the input word size (given in bits) in the bit model.

On the other hand, for rational inputs, the *ellipsoid* algorithm [Khachiyan79] is polynomial time in the input word size in the bit model. But, as an algorithm over $\mathbb{R}$, i.e., allowing exact arithmetic, it is not finite in general. The same is true of the newer *interior point* algorithms.[19]

It would seem more natural, and appropriate, to analyze the complexity of algorithms for the LPP_R with respect to an *intrinsic input word size*. Hence we are motivated to define a measure of the condition of a linear program [Blum90]. Jim Renegar [Renegar95] was the first to propose such a condition number in this context. His definition is inspired by the 1936 theorem of Eckart and Young.

Theorem [EckartYoung36]. For a real matrix A, $\kappa(A) \sim 1/d_F(A,\Sigma)$ where Σ is space of *ill-posed* problem instances, i.e., Σ is the space of *noninvertible* matrices. (Here the relative distance d_F is with respect to the *Frobenius norm* $\|A\|_F = \sqrt{\Sigma a_{ij}^2}$.)



We now consider the linear programming problem over $\mathbb{R}$ in the form: Given $Ax = b$, $x \geq 0$, find a *feasible* solution or declare there is none.

Definition [Renegar95]. The *condition number* of a linear program over $\mathbb{R}$ with data (A,b) is given by $C(A,b) = \|(A,b)\|/d((A,b),\Sigma_{m,n})$. (Here, $\Sigma_{m,n}$ is the boundary of the feasible pairs (A,b), and both the operator norm $\|\ \|$ and the distance d are with respect to the respective Euclidean norms.)

Renegar proposes an interior point algorithm and analyzes it with respect to parameters: n, m, the *loss of precision*, and the *desired accuracy of solution*.

Theorem [Renegar Interior Point Algorithm].[20] If the linear program is feasible, the number of iterations to produce an ε-approximation to a feasible point is polynomial in n, m, $\log^+ C(A,b)$ and $|\log \varepsilon|$.

Felipe Cucker and Javier Peña propose an algorithm and add *round-off error* as a parameter for the complexity analysis [CuckerPena02].

Theorem [Cucker-Peña Algorithm with Round-Off]. If the linear program is feasible, the bit cost to produce an ε-approximation to a feasible point is $O((m+n)^{3.5}(\log(m+n)+\log^+ C(A,b)+|\log\varepsilon|)^3)$. The finest precision required is a *round-off* unit of $1/c(m+n)^{12}C(A,b)^2$.

While condition, approximation and round-off help bridge the combinatorial and continuous approaches to the design and analyses of linear programming algorithms, basic connections and complexity questions *remain open*.

In particular: Is $\mathrm{LPP}_{\mathbb{R}} \in P_{\mathbb{R}}$? Even more, is LPP *strongly polynomial*? That is, is there a polynomial time algorithm over $\mathbb{R}$ for $\mathrm{LPP}_{\mathbb{R}}$ that is also polynomial time with respect to bit cost on rational input data?[21]

In conclusion, I have endeavored to give an idea of how machines over the reals—tempered with condition, approximation, round-off, and probability[22]—enable us to combine tools and traditions of theoretical computer science with tools and traditions of numerical analysis to help better understand the nature and complexity of computation.

References

[BPR96] S. Basu, R. Pollack, and M.-F. Roy, On the combinatorial and algebraic complexity of quantifier-elimination, *J. ACM* **43** (1996), 1002–1045.

[Blum90] L. Blum, Lectures on a theory of computation and complexity over the reals (or an arbitrary ring), *Lectures in the Sciences of Complexity II*, (E. Jen, ed.), Addison-Wesley, 1–47, (1990).

[Blum91] ____ , A theory of computation and complexity over the real numbers, *Proceedings of the International Congress of Mathematicians*, 1990,

[19]*We emphasize this* distinction. *Simplex is a finite algorithm over $\mathbb{R}$ and over the rationals in both the exact arithmetic model and the bit model. The newer interior point algorithms are finite* only *over the rationals in the bit*

[20]*By giving the simplest results here, I am hardly doing justice to the full extent of Renegar's and others' work in this area but hope this discussion will spur the reader to investigate this area more fully.*

[21]*Recall that in each case, i.e., over $\mathbb{R}$ and in the bit model, complexity is a function of input word size. However, over $\mathbb{R}$, the input size for a rational linear program is the vector length of the input (approximately $m \times n$), whereas in the bit model it is the bit length (approximately $m \times n \times k$, where k is the maximum height of the coefficients).*

[22]*For more on probabilistic algorithms and probabilistic analysis see [BCSS98]. For more results on complexity estimates depending on condition and round-off error, see [CS98].*

Kyoto, (I. Satake, ed.), Springer-Verlag, 1492-1507, (1991).

[BCSS96] L. Blum, F. Cucker, M. Shub, and S. Smale, Algebraic settings for the problem P=NP?, The Mathematics of Numerical Analysis, Volume 32, *Lectures in Applied Mathematics*, (J. Renegar, M. Shub, and S. Smale, eds.), Amer. Math. Soc., (1996), 125-144.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer-Verlag (1998).

[BSS89] 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, *Bulletin of the AMS* **21** (July 1989), No. 1, 1–46.

[BlumSmale93] L. Blum and S. Smale, The Gödel incompleteness theorem and decidability over a ring, *From Topology to Computation: Proceedings of the Smalefest*, (M. Hirsch, J. Marsden, and M. Shub, eds.), 321–339 (1990).

[CMPS02] D. Castro, J. L. Montaña, L. M. Pardo, and J. San Martin, The distribution of condition numbers of rational data of bounded bit length, *Foundations of Computational Mathematics* **2** (1) (2002), 1–52.

[Cook71] S. Cook, The complexity of theorem-proving procedures, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 151–158, (1971).

[CuckerPena02] F. Cucker, and J. Peña, A primal-dual algorithm for solving polyhedral conic systems with a finite-precision machine, *SIAM Journal on Optimization* **12** (2002), 522–554.

[CS98] F. Cucker and S. Smale, Complexity estimates depending on condition and round-off error, *Journal of the ACM* **46** (1999), 113–184.

[Dantzig47/90] G. B. Dantzig, Origins of the simplex method, *A History of Scientific Computing*, (S. G. Nash, ed.), ACM Press, New York, 141–151, (1990).

[DavisMatijasevicRobinson76] M. Davis, Y. Matijasevic, and J. Robinson, Hilbert's Tenth Problem: Diophantine Equations: Positive Aspects of a Negative Solution, *Proceedings of Symposia in Pure Mathematics*, vol. 28, 323–378, (1976); reprinted in *The Collected Works of Julia Robinson*, (S. Feferman, ed.), Amer. Math. Soc., 269–378, (1996).

[Edelman88] A. Edelman, Eigenvalues and condition numbers of random matrices, *SIAM J. Matrix Anal. Appl.* **9** (1988), 543–560.

[EckartYoung36] C. Eckart and G. Young, The approximation of one matrix by another of lower rank, *Psychometrika* **1** (1936), 211–218.

[Hilbert1893] D. Hilbert, Über die vollen Invariantensysteme, *Math. Ann.* **42** (1893), 313–373.

[Hermann26] G. Hermann, Die Frage der endlich vielen Schritte in der Theorie der Polynomideale, *Math. Ann.* **95** (1926), 736–788.

[Kachiyan79] L. G. Khachiyan, A polynomial algorithm in linear programming, *Soviet Math. Dokl.* **20** (1979), 191–194.

[Karp72] R. M. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations,(* R. E. Miller and J. W. Thatcher, eds.), 85–103, Plenum Press, (1972).

[KleeMinty72] V. Klee. and G. J. Minty, How good is the simplex algorithm?, *Inequalities, III* (O. Shisha, ed.) 159–175, Academic Press (1972).

[Koiran 02] P. Koiran, Transfer theorems via sign conditions, *Information Processing Letters* **81** (2002), 65–69 .

[KPS01] T. Krick, L. M. Pardo, and M. Sombra, Sharp estimates for the arithmetic Nullstellensatz, *Duke Mathematical Journal* **109** (2001), 521–598 .

[Levin73] L. A. Levin, Universal search problems, *Problems Inform. Transmission* **9** (1973), 265–266.

[Michaux94] C. Michaux, $P \neq NP$ over the nonstandard reals implies $P \neq NP$ over $\mathbb{R}$, *Theoretical Computer Science* **133** (1994), 95–104.

[Penrose89] R. Penrose, *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, Oxford University Press (1989).

[Renegar92] J. Renegar, On the computational complexity and the first order theory of the reals, Parts I-III, *Journal Symbolic Comp.* **13** (1992), 255–352.

[Renegar95] J. Renegar, Incorporating condition numbers into the complexity theory of linear programming, *SIAM J. Optim.* **5** (1995), 506–524.

[Seid54] A. Seidenberg, A new decision method for elementary algebra, *Ann. of Math* **60** (1954), 365–374.

[Shishikura91/98] M. Shishikura, The Hausdorff dimension of the boundary of the Mandelbrot set and Julia sets, *Ann. of Math.* **147** (2) (1998), 225–267. (Preprint, SUNY at Stony Brook, IMS 1991/70.)

[Smale2000] S. Smale, Mathematical Problems for the Next Century, *Mathematics: Frontiers and Perspectives 2000*, (V. Arnold, M. Atiyah, P. Lax, and B. Mazur, eds.), Amer. Math. Soc., Providence, RI, 2000.

[Turing36-37] A. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. Lond. Math. Soc.* (2) **42** (1936-7), 230–265; correction ibid. **43** (1937), 544–546.

[Turing48] A. Turing, Rounding-off errors in matrix processes, *Quart. J. Mech. Appl. Math.* **1** (1948), 287–308.

[Yger01] A. Yger, The diversity of mathematics with respect to a problem in logic, `http://www.math.u-bordeaux.fr/~yger/exposes.html`.