Mathematical Research Today and Tomorrow

Viewpoints of Seven Fields Medalists

Lectures given at the Institut d'Estudis Catalans, Barcelona, Spain, June 1991

Editors: C. Casacuberta, M. Castellet

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Symposium on the Current State
and Prospects of Mathematics

Barcelona, June 1991

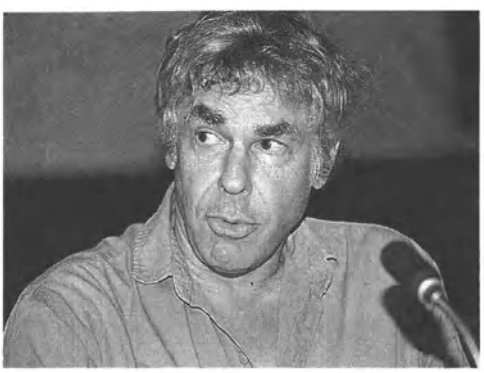
Theory of Computation

by

Stephen Smale

Fields Medal 1966

for his work in differential topology, where he proved the generalized Poincaré conjecture in dimension $n \geq 5$: Every closed n-dimensional manifold homotopy equivalent to the n-dimensional sphere is homeomorphic to it. He introduced the method of handle-bodies to solve this and related problems.



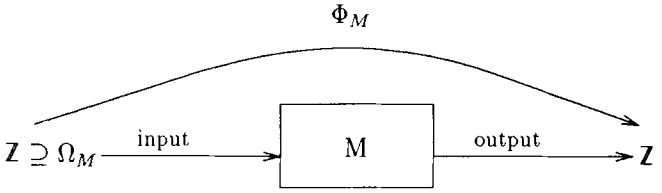
Abstract: It could be said that the modern theory of computation began with Alan Turing in the 1930's. After a period of steady development, work in complexity, specially that of Steve Cook and Richard Karp around 1970, gave a deeper tie of the Turing framework to the practice of the machine. I will discuss an expansion of the above to a theory of computation and complexity over the real numbers (joint work with L. Blum and M. Shub).

Theory of Computation

The reason for a theory of computation, for me in particular, comes from an attempt to understand algorithms in a more systematic way. The notion of algorithm is very old in mathematics; it goes back a couple of thousand years. Mathematicians have talked about algorithms for a long time, but it was not until Gödel that they tried to formalize the notion of algorithm. In Gödel's incompleteness theorem one saw for the first time the limitations of computations or the need to study more clearly what could be done.

To do so, one has to establish more explicitly what an algorithm is, and I think that this became clearer in the way that Turing interpreted Gödel. So let us stop for a moment and look more closely at Turing and his achievements. I think it is fair to say that he laid down the first theory of computation. Perhaps I will be more specific later about what Turing's notion of computation was.

We can take as set of inputs the integers $\mathbf{Z}$, so in the Turing abstraction the input is some integer; perhaps not every integer is allowed but only those that were eventually called the *halting set* Ω_M of the machine M. This is the domain of computation of the machine M. Given an integer in Ω_M, we feed it to the machine M and obtain as output another integer.



There is some kind of mechanism here, described by Turing, which I will later on formalize in my own way. Turing gave different versions of the input set; for instance, finite sequences of zeroes and ones.

Thus Gödel's incompleteness theorem can be stated in the following way.

THEOREM. *There is some set $S \subset \mathbf{Z}$ which is definable in terms of a finite number of polynomial conditions and is not decidable.*

This is Gödel's incompleteness theorem as formulated by Turing. *Not decidable* means that there is no computable function over $\mathbf{Z}$ which is 1 on S and 0 out of S. In other

words, S is *decidable* if its characteristic function is computable by some machine. Thus, Gödel's incompleteness theorem asserts that there exists a set S which is very definable mathematically, yet is not decidable.

This is in some sense the beginning of the theory of computation, which shows the limits of decidability. Eventually, from this evolved a theory for present-day computers. It is from this formulation that it evolved into one of the foundations of computer science. Even a very refined theory of computer science is developed from this: This is complexity theory, which I could say today lies in the center of theoretical computer science, specially after the work of Cook [3] and Karp [7]. They made use of the notion of speed of computation; now the question is not whether a set is decidable, but whether it is decidable in a time that can be affordable by present-day machines, or whether it is a "crackable" problem.

The fundamental question is

$$P \neq NP?$$

This is a very famous conjecture and it is the most important new problem in mathematics in the last half of this century; it is only 20 years old. To me it is the most beautiful new problem in mathematics. Very hard to solve, a very fine notion coming from this theory of Cook and Karp.

So we have a very active subject in this area but there is something that is missing. I have talked about the need for a notion of definable algorithm, yet the algorithms mathematicians have used for a couple of thousand years at least do not fit into this framework. The algorithms we are talking about have to do with real numbers, and specially since the time of Newton they have had to do with differential equations, nonlinear systems, etc. We see the notion of the real numbers $\mathbf{R}$ is central.

Newton's method to me is a paradigm of a great classical algorithm like the procedure of the Greeks for finding square roots, and it does not fit naturally into this framework, because the framework is quite discrete and to fit Newton's method into it requires destroying geometric concepts. One can do this in a very cumbersome way —I find this a very destructive way— to deal with the algorithms of continuous mathematics with Turing machines.

Indeed, some work has now been done to adapt the Turing machine framework to deal with real numbers. Let me mention two such attempts. One of them is recursive analysis, which initially was worked out by Ker-I-Ko and Harvey Friedman [5] and the main name connected with it is Marian Pour-El. She worked with Richards [8] in developing a kind of real number analysis based on Turing machines. There has been very extensive work on this which deals with partial differential equations, and the way to deal with real numbers in this context is to consider a real number s defined by its decimal expansion

$$s = 1.2378\ldots.$$

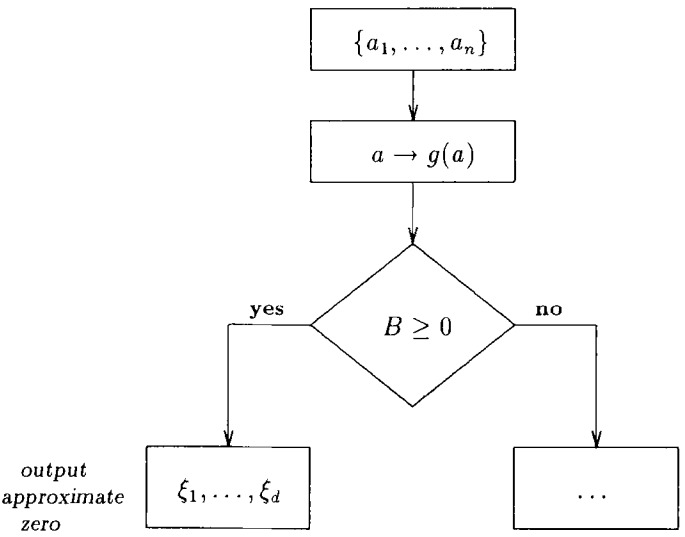
A real number is computable in this sense if there exists a Turing machine which says that the first digit is 1, the second 2, the third 3, and so on, with the decimal point in the appropriate place. So a computable real number is given by a Turing machine. These work with computable real numbers and eventually provide a very successful theory. Similarly there is the notion of *interval arithmetic* from R. E. Moore (see [1]). In some ways it is close to the work of Pour-El but in a quite different direction. Thus, the foundations are probably being laid for a theory of computation over the real numbers.

Now, continuing from a very different point of view, I will devise a notion of computation taking the real numbers as something axiomatically given. So, a real number to me is something not given by a decimal expansion; a real number is just like a real number that we use in geometry or analysis, something which is given by its properties and not by its decimal expansion. Eventually, I will talk about a notion of computability over the real numbers which takes this point of view. There one thinks of inputing a real number not as its decimal expansion but as an abstract entity in its own right.

Some mathematicians and computer scientists have trouble with the idea that a machine takes as input an arbitrary real number. I wrote a paper [13] on precisely this point, saying that here one idealizes, as in physics Newton idealized the atomistic universe —making it a continuum— in order to use differential equations. One can idealize the machine itself by conceiving it as allowing an arbitrary real number as input, but I am not going to argue about this point today.

In a preliminary phase, I was concerned with the problem of root finding for polynomials for many years. In that process I faced the kind of objects known as *tame machines.* It is not a theory of computation, but just a preliminary.

We take as input now the coefficients $\{a_0, a_1, \ldots, a_d\}$ of a complex polynomial f of degree d, and we think of it over the real numbers, i.e., each a_i is given by its real and imaginary parts. Thus, we think of this as the input and we describe the computation in the language of flowcharts. Then comes a box describing the computations. We replace a by $g(a)$, where g is a rational function. This vector of numbers $[a_0, a_1, \ldots, a_d]$ can be considered as a state and in this step this state is transformed by a rational function. Then we can put down and answer the question of whether some coordinate of the state is less than or equal to 0. Depending on the outcome of this comparison, we continue along the corresponding branch.



So we go down the tree in this way, and eventually we may output the approximate

zeroes $\{\xi_1, \ldots, \xi_d\}$ of f up to some ε.

In fact this is a good way to express an algorithm for solving this equation. We next ask: How about these nodes which branch? To what extent are they necessary?

The topological complexity of a problem in general is the minimum number of these branch nodes for any machine which solves the problem.

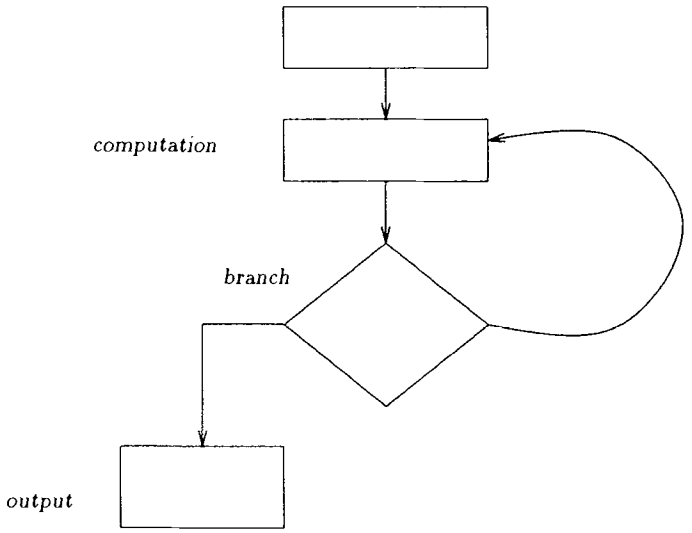
I am not being completely precise about what resolving a problem means. One can imagine this example as a prototype of the general situation of a machine that solves problems. In particular, for this problem of finding the zeroes of polynomials we have the notion of its *topological complexity.* And the theorem [12] is as follows:

THEOREM. *The topological complexity of the root finding problem is greater than or equal to* $\log d$.

So the topological complexity increases with the degree, and the proof of this theorem actually is not so easy; it uses the cohomology of the braid group worked by Fuchs in Russia [6]. Subsequently, Vasiliev [14] extended this bound to $\simeq d$. So the answer eventually emerged that the topological complexity grows linearly with d and this is a sharp bound.

One notes that the Turing machine framework could never deal with this way of looking at all possible algorithms, even in this limited class. There is no useful way of thinking about it in terms of Turing machines, whereas using this kind of tree we were able to give necessary conditions on all algorithms, what we call *lower bound theorems.* There is some early work dealing with this kind of tree, but this is the first time we have obtained topological complexity results using algebraic topology.

Then, shortly after this, we did a joint work with Lenore Blum and Mike Shub [2] and developed this into a complete theory of computation over the reals by allowing loops. We certainly increased the computational power of tame machines by allowing loops to give a notion of computation in general. This situation is reflected in the next picture.



So here we have also

- an input space $\mathbf{R}^l$,
- an output space $\mathbf{R}^k$,
- and also a state space $\mathcal{S} = \mathbf{R}^j$ for things happening inside the machine.

If l, k, j are finite, this essentially defines a machine. We can take here an oriented graph where the nodes are computation nodes given by rational functions, branch nodes given by inequalities, and input and output defined inside accordingly. At each computation node there is a single output, a single branch going out of the node. A decision node has two. Remember that the number of input branches is arbitrary except for the input node where nothing comes in and there is a branch going out, and an output node, where nothing comes out.

This gives a theory of computation for finitely dimensional input and output spaces motivated directly by the flowcharts used in scientific computation. Yet the full theory will have to allow $l, k, j = \infty$ and we will have to have a little more technical process to access far out coordinates, but this is the idea.

This model gives an algebraic flavour to the process of computation. We defined this not only over the real numbers but over any ordered ring, eventually any ring. In particular, if we take the ring to be $\mathbf{Z}$, the input space to be a subset of $\mathbf{Z}$, the output space again $\mathbf{Z}$, and the state space $\mathbf{Z}^\infty$, we obtain Turing theory, and so this extends the Turing theory of computation.

We can now say that a Turing computable function is one which is given on some Ω of the machine, a domain of inputs, by following the flow of the machine and doing what is said at each node.

$$\{admissible\ inputs\} = \Omega_M \xrightarrow{\Phi_M} \mathbf{R}^k$$

And this essentially is a complete picture of what we mean by *computable function.* A function Φ_M defined by a machine going from the admissible inputs or the halting set of the machine to the output set.

And it is precisely equivalent —or practically so— to the notion of *Turing computable* in the case when the ring is the ring of integers.

We have developed for this model notions of computability itself; we have, for instance, shown the existence of universal machines. We have a complexity theory and the problem "$P \neq NP$?" is also defined over these rings; for example, the theory for $\mathbf{R}$ or $\mathbf{C}$ possesses universal or NP-complete problems just as in the case of Cook and Karp.

An NP-complete problem over $\mathbf{C}$ (a machine over $\mathbf{C}$ is like one over $\mathbf{R}$ except that the branch nodes just ask "$\neq 0$?") is the following:

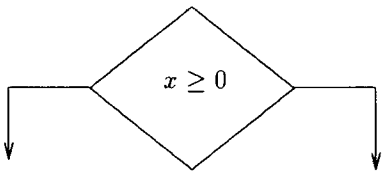
Does a system of quadratic polynomials have a zero?

The idea of the reduction is to have more polynomials than variables. So it is an open question whether there is a machine that can decide in polynomial time if there is such a zero. All this is written very carefully in our paper.

In Barcelona, Felipe Cucker [4] gave an analog for the real numbers of the arithmetical hierarchy of classical recursion theory. There have been developments in different

directions in the theory of computation of these machines from the point of view both of complexity theory and of computability.

There has also been a lot of controversy and criticism. Let me deal with one main point, making some comments on two sharp critiques by Pour-El and Moore. Our theory of computation is very different from their two theories. In a way this is more or less the basis of their criticism. It has to do with the branching



We branch according to whether one of the coordinates of the state space is greater than or equal to 0. This is in some respects one of the most controversial elements in the kind of machines we have, because the question is that an actual machine cannot do this. Given a number, for example

$$0.0000\ldots00\ldots,$$

it may or may not have a one after that eventually. If it never has a one, and we input it to an actual machine, we can never decide this question. If it does have a one, we wait long enough and we can decide it.

So we have a problem here when branching at ≥ 0, or equivalently at $= 0$, and this is the focus of attack of both Pour-El and Moore. Let me give an example here.

Both of their theories of computation lead to a notion of computable function which is continuous. Every computable function here is continuous. Even in a strong sense: They have to be constructively continuous.

Now the clue to this lies in the philosophy of thinking about the real numbers as abstractly given, and choosing the idealization of the right machines. For example, in scientific computation this is the kind of computation carried out traditionally by algorithms like Newton's method. One does test if something is ≥ 0, then do this, if not do something else.

Moreover, the need for these branchings is given by our earlier results on topological complexity. Topological complexity states that if one wants to find zeroes of polynomials then one has to branch, and the number of branchings in the machine is given approximately by the degree. Even to approximately solve the fundamental theorem of algebra one needs to branch. So I would imply that these two theories of computation do not lead even to an approximate solution of the fundamental theorem of algebra.

Here I would refer to a letter I received from Moore a year ago. I do not intend to dwell here on my opinion that numerical analysis and scientifical computing have weak foundations. Moore is the main developer of interval arithmetic and he wrote that "There are foundations for scientifical computation. More than 2000 papers and dozens of books. I invite you to read all of Aberth's book" [1]; it is a book that Moore even sent to me. He said "It will open your eyes to a whole new world." So I opened the book —actually a few weeks ago— and read on page 34 of the book (called *Precise Numerical Analysis*) "The problem of deciding whether two computable real numbers are equal is therefore a computational problem one should avoid." But problem 3.1 on this book reads: "Given

two numbers a, b decide whether $a = b$." Later, on page 62, Aberth says: "Solve the problem 6.1: Find k decimals for the real and imaginary parts of the zeroes of a polynomial of positive degree." But the answer to this solvable problem —the fundamental theorem of algebra— needs to pass through d versions of this single problem that "one should avoid."

Marian Pour-El very kindly sent me a review that she has given of our paper in the Journal of Symbolic Logic [9], in which she says that it is a very good, highly developed theory of computability over the reals. In the review she confirms that in her theory she can only produce continuous functions and so she cannot solve the fundamental theorem of algebra, not even approximately.

What I will now do is to pass on to something which relates to this problem of NP-completeness if only a little indirectly. This is work done jointly with Mike Shub in the last few months. It is an example of an algorithm which fits into our framework. But it is a simple algorithm, so the fact that it is an algorithm in our strict sense is secondary. It is the problem of the complexity analysis of Bézout's theorem. Let me say a little bit about what this is. The situation we look at is as follows: We have a polynomial system

$$f : \mathbf{C}^n \to \mathbf{C}^n$$

of n polynomials in n variables of degrees $d_1, \ldots, d_n$ respectively. One wants to find an algorithm and analize its speed for solving the equation

$$f(z) = 0\,;$$

not to produce a solution but to analize how much time it takes. The idea is to make complexity analysis on this.

The work done so far on polynomial equation solving can be summarized by dividing it into two parts: one is Newton's method as the basic algorithm —it is essentially the method used by the Greeks for finding square roots— and the other method is elimination theory, a very algebraic method; it works over arbitrary fields. In the first one we are using some kind of norm or metric, so it is metric-oriented. It is the method of choice of numerical analysts. The second is probably the method that would be chosen by a computer scientist. My own inclination is to the first side. Numerical analysts have a better focus on the problem. They do not have a complexity theory or any kind of foundation, but they have a better instinct about how to solve this problem.

In any case, what we use is some global version of Newton's method to solve Bézout's theorem. That is, we use Newton's method to follow a path in the space of polynomials.

All these are homogeneized. It is more elegant to think entirely in terms of homogeneous coordinates and projections

$$\mathcal{P}_{(d)} = \{f : \mathbf{C}^n \to \mathbf{C}^n\}$$

and

$$\mathcal{H}_{(d)} = \{f : \mathbf{C}^{n+1} \to \mathbf{C}^n \ \text{ homogeneous of degree } d\}\,,$$

with

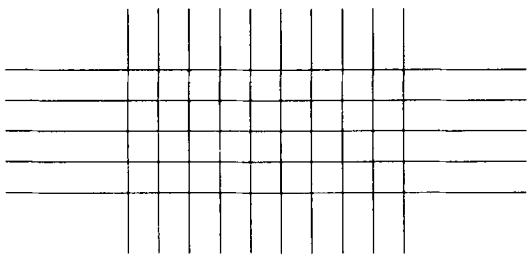
$$\mathcal{P}_{(d)} \equiv \mathcal{H}_{(d)}\,.$$

Thus we are going to work in projective space, and the main thing we will analize is a projective version of Newton's method which is due to M. Shub [11]. The previous work

has been done mostly in one variable on this problem, using Newton's method to solve this; I spent many years doing that. Jim Renegar [10] has some extension to n variables.

What we want to do here is give a very conceptual process. All the ideas are lying there; we can try to understand the best, most elegant ways of looking at algorithms to solve this. In this way perhaps eventually we will be able to see more clearly the problem "$P \neq NP$?" over the real numbers or the complexes. I hope it will eventually shed some light on the practical problem "$P \neq NP$?"

Given the space $\mathcal{H}_{(d)}$ we consider a function f_0 for which we know the zeroes. The zeroes of f_0 could be given by a set of intersections in a grid so we get a set of equally spaced zeroes



This could be the initial element in $\mathcal{H}_{(d)}$ for which we know the answer, and we simply homotope that back

$$f_t = tf + (1-t)f_0\,,$$

where t goes from 0 to 1. Let us denote by $\mathcal{F}$ the curve f_t in the space $\mathcal{H}_{(d)}$.

We try to trace the zeroes, which we know for f_0, to give the answer for f_1. This seems to be a very good method that has been used for the last decade or two. It embodies some kind of global Newton's method. The idea is to consider some sequence t_i and apply Newton's method to the function $f_{t_{i+1}}$, starting from some approximation X_i of the solutions for f_{t_i}

$$N_{f_{t_{i+1}}}(X_i) = X_{i+1}$$

where, here, $N_f(X)$ stands for applying Newton's method to solve f starting with the initial guess X. In the projective space $\mathbf{P}_n(\mathbf{R})$ we can see the paths given by the solutions X_t of f_t and the algorithm provides a sequence of points X_i following this path very closely.

So, what kind of results can we expect here? What kind of things may we prove? Here is the main theorem. The question is how many iterative steps are necessary, how many t_i, in such a way that we can follow this path very closely, and our result is

THEOREM. *The number of steps is bounded above by*

$$\frac{\hat{\alpha} D^{3/2}}{\rho^2}$$

where $\hat{\alpha}$ is a universal constant (given by a set of equations which can be solved itself by Newton's method) which is approximately $1/16$, D is $\max\{d_1, \ldots, d_n\}$ and ρ is the distance from the arc joining f_0 and f_1 to the discriminant variety.

It should be recalled that the discriminant variety is the subset of $\mathcal{H}_{(d)}$ of all singular polynomial systems. It is the variety of polynomial systems which are degenerate at some zero. And this is an algebraic variety that we shall call Σ. The theorem says that what is crucial are not the coefficients of f. They do not even enter. In fact, not even the dimension comes directly here; this is even dimension-free. But what is crucial here is the distance ρ between $\mathcal{F}$ and the discriminant variety Σ. This is the crucial factor —the only factor— in estimating the complexity for finding the zeroes of a polynomial system. Now we have to make a little caveat here because we are not finding the zeroes of every polynomial system. There may be a continuum of zeroes and then we cannot do this. So we have to put some kind of condition, let us say to solve $f + \varepsilon$ where ε is a small polynomial. This is the thing we solve. We cannot find the solution of arbitrary polynomial systems; there may be a continuum of solutions, but for some deformation we can find the zeroes in a very exact sense.

Now, the great problem to me is: To what extent is the term $D^{3/2}$ necessary?

While we have no proof of this, we suspect that the D itself could be eliminated from the formula. For a polynomial in one variable, this is the fundamental theorem of algebra, and we show that we can take off the $3/2$ to get D. This is what we have done in the last months; the proof is in handwritten form. Since last week we believe that we can eliminate the D in the one variable case, but this uses the theory of Schlicht functions, which is only available for one variable. There is no theory of Bieberbach conjecture for more than one variable. If it is true, if it is D-free, if we can do this, then one can find for example one zero of a polynomial in one variable in a universal number of steps, say one hundred.

References

[1] O. Aberth, *Precise Numerical Analysis*, Brown Publishers, Dubuque, Iowa, 1988.

[2] 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. (N.S.)* **21** (1989), no. 1, 1–46.

[3] S. A. Cook, The complexity of theorem-proving procedures, Proceedings 3rd ACM STOC (1983), 80–86.

[4] F. Cucker, The arithmetical hierarchy over the reals, to appear in *J. Logic Comput.*

[5] H. Friedman and K. Ko, Computational complexity of real functions, *Theoret. Comput. Sci.* **20** (1986), 323–352.

[6] D. Fuchs, Cohomologies of the braid group mod 2, *Functional Anal. Appl.* **4** (1970), 143–151.

[7] R. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, R. Miller and J. Thatcher (eds.), Plenum Press, New York, 1972, 85–104.

[8] M. B. Pour-El and I. Richards, Computability and noncomputability in classical analysis, *Trans. Amer. Math. Soc.* **275** (1983), 539–560.

[9] M. Pour-El, Review of [2], to appear in *J. Symbolic Logic.*

[10] J. Renegar, On the efficiency of Newton's method in approximating all the zeroes of a system of complex polynomials, *Math. Oper. Res.* **12** (1987), 121–148.

[11] M. Shub, Some remarks on Bézout's theorem and complexity theory, to appear in *Proceedings of the Smalefest*, M. Hirsch, J. Marsden and M. Shub (eds.).

[12] S. Smale, On the topology of algorithms I, *J. Complexity* **3** (1987), 81–89.

[13] S. Smale, Some remarks on the foundations of numerical analysis, *SIAM Rev.* **32** (1990), no. 2, 211–220.

[14] V. Vasiliev, Cohomology of the braid group and the complexity of algorithms, to appear in *Proceedings of the Smalefest*, M. Hirsch, J. Marsden and M. Shub (eds.).

Stephen Smale
Mathematics Department
University of California
Berkeley, California 94720
USA

Transcribed from the videotape of the talk by Felipe Cucker, Francesc Rosselló and Álvaro Vinacua; revised by the author.