Mysteries of Mathematics and Computation

Michael Shub

I was quite surprised last May when Ted Brown, the Chairman of the Queens College Computer Science Department, called and asked me to give a lecture entitled "The Future of the Theory of Computing." I had just returned from two months in California and hadn't opened my mail yet. So I didn't realize that Ted was asking for a very brief and informal talk. He had written to each member of the Industrial Affiliation Board of the Queens College Computer Science Department asking that they make a small presentation at a meeting to be held at the College in June.

About two years ago I wrote a book review of David Ruelle's book on dynamical systems and bifurcation theory; this made me reflect on what I thought was important in dynamical systems. I was quite happy with the result [20]. So I hesitantly agreed to give the lecture, to have the opportunity for some reflection on the theory of computing, only partly aware of the work it would entail. Among the issues I wanted to consider was the relation of theory and practice — past, present, and future; especially as sources of funding are more and more asking the scientific community to assess the effectiveness of scientific research. After the talk was announced at Queens College, I was asked to give it in my own department and at the Courant Institute, and I revised the talk each time.

Theory is related to practice in at least four ways. For each I will give examples first from computer science and then from mathematics.

1. **Incremental:** Theory and practice incrementally improve together in a well-defined branch of study.

   Here I have in mind the steady improvement of algorithms for data sorting, compiler design, etc.; for mathematics, the steady progress in the numerical solution of differential equations of interest in physics, engineering, and industrial design.
2. **Structural:** Theory provides a language and structure in which to discuss and analyze an existing practice.

   As an outstanding example here we have complexity theory, P- and NP-complete problems, or the application of linguistic and logical constructions to the theory of computer languages and compilers. For mathematics, we may take the organizing effect that dynamical systems has had on ordinary differential equations.
3. **Anticipatory:** For its own internal reasons, theory creates structures which later may be important for practice.

   Here we might point to the spectacular example of Turing's Universal Machines as precursors of the modern computer and to this day as the main theoretical model of the machine. I will return to this

Michael Shub

Michael Shub is a research staff member and manager of the Special Math Studies project in the Mathematical Sciences Department at IBM Research's T. J. Watson Research Center. His wife, Beate Echols, is an importer and collector of Latin American folk art. Michael enjoys assisting her on her buying trips. He met the friend in the photo on such a trip to the Andes Mountains in Venezuela.

later. For mathematics, we have the standard example: the development of Riemannian geometry which was later so useful to Einstein.

On a smaller scale, but much more recently and much closer to home, in fact in my own department, the work of Brian Marcus and Roy Adler on the classification problem in symbolic dynamics was later found useful for coding, and was used to improve the storage capacity of our disks.

4. **Informal:** Theory creates a background which is thought to embody the state of knowledge as a measure of what can be accomplished.

   Here we have Gödel's and Turing's theorems and the conviction that NP-complete problems are hard.

   For mathematics, we can again cite Gödel and Turing, or the unpredictability of chaotic systems.

For each category there are lots of examples, and, of course, the structural, anticipatory, and informal all contribute to the incremental.

Having tied theory to practice, I was partly off the hook as far as the title of my lecture was concerned. The future of theory will depend to a degree on the future of practice, and we will have to wait and see what that is. Currently, the theory of computing is mostly tied to machine design, data management, and the concomitant combinatorial optimization problems. Looking at the table of contents of the Symposium on the Theory of Computing, 1969, I was struck by how much the first symposium seemed like logic or perhaps the theory of computable functions, with emphasis on the complexity of computable functions. By 1992 complexity theory has matured enormously and some new issues appear in the 24th Annual ACM Symposium on the Theory of Computing, such as parallel computing and fault tolerance. But the 1992 STOC emphasis remains on solving combinatorial problems and problems about the logical structure of the machine such as problems of communication.

Over the decades, data processing has occupied most computer time, but the advent of the workstation has put cheap sophisticated computational power on millions of desks. Computational mathematics, even symbolic computational mathematics is exploding. CAD-CAM and emerging computer-video interactive technologies, robotics, and scientific problems such as protein folding will require extensive scientific computation. Parallel and distributed computers will give greatly enhanced computing power. The computer is a tool; numerical algorithms get more sophisticated, partly in response to more sophisticated hardware. We can easily predict that theory will incrementally progress along with practice. The current explosion of work on wavelets is an excellent example.

Stored-program digital computers were invented to do scientific computations. Here the theory, numerical analysis, and the practice seem almost entirely incremental. Wilkinson's invention of backward error analysis and condition number are two exceptions. I would call them structural. What are the ingredients of a structural theory of scientific computation? Let me give you an example of a problem with scientific origin and the difficulties we encounter.

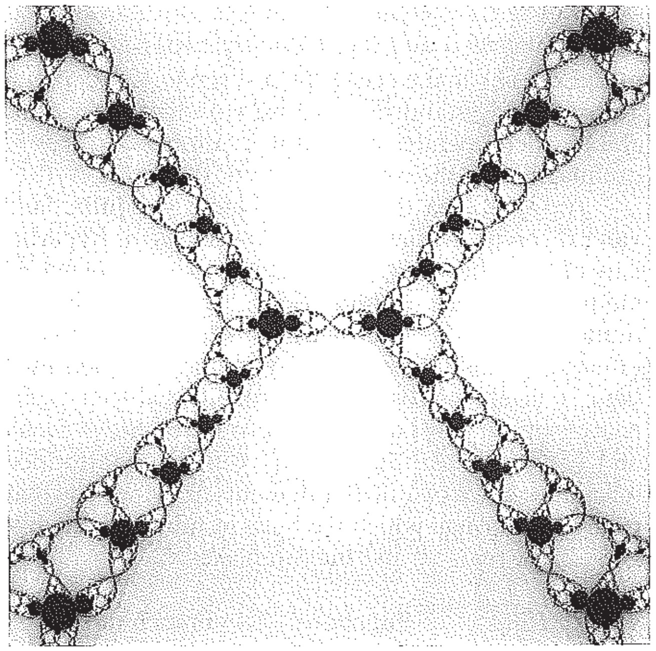


Figure 1.

Computer graphics have proven to be extremely useful tools for the study of low-dimensional dynamical systems. The Lorenz attractor, Julia sets, Mandelbrot set, and Henon attractor are a few of the earliest and most famous examples. If we consider Newton's method for the complex polynomial

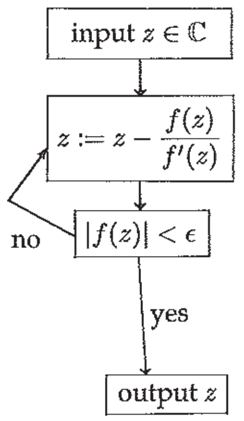
$$f(z) = (z^2 - 1)(z^2 + 0.16),$$

we have the beautiful picture made by Scott Sutherland [1] (Fig. 1; also see the cover of this issue for a color version). We recall that $N_f(z) = z - f(z)/f'(z)$ is a dynamical system on the Riemann sphere. The fixed points of N_f are the roots of f. These are along the imaginary and real axes. The regions in shades of red, green, yellow or blue converge under iteration of N_f to these roots, with the lighter shades converging more quickly. The black region is an open set of points which converge under iteration to two attracting points of period two and, thus, fail to converge to a root. At the boundary between any two hues, including black, is (an approximation of) the Julia set. Its fractal nature is evident in the picture. This computer graphic is an excellent heuristic to understand the dynamics and hence Newton's method for this particular case.

But now let us turn this discussion around and ask ourselves to explain the machine which produced this picture and the picture itself. The Turing machine model does not seem adequate to the task. For starters, fractional dimension and the boundary between open domains do not make sense for a discrete set of points. To truly understand this picture we must posit a machine which operates on complex numbers, so that we can understand the complex analytic dynamics $N_f(z) = z - f(z)/f'(z)$ via the work of Fatou and Julia, Sullivan's theorem on nonwandering domains, the hyperbolicity of the Julia set, approximations to it, etc. Newton's method is but one example. Scientific computation and numerical analysis deal with problems whose natural domains of definition are the real and complex numbers. To be able to discuss computability, efficiency, and complexity in this context, Lenore Blum, Steve Smale, and I [Blum–Shub–Smale [2] (=BSS)] have introduced machines which operate on elements of a ring. The main examples of rings which we have in mind are the integers, $\mathbb{Z}$, the reals, $\mathbb{R}$, and the complexes, $\mathbb{C}$.

Ring	Functions	Branching
$\mathbb{Z}$	Polynomials	≤ 0 or > 0
$\mathbb{R}$	Rational functions	≤ 0 or > 0
$\mathbb{C}$	Rational functions	$= 0$ or $\neq 0$

Our machines, which have come to be known as BSS machines, are essentially flowchart machines, comprising a finite directed graph with five types of nodes: input, output, computation, branch, and a certain fifth node which can access memory. All computations are done on a finite number of variables and there is only one input node. An example of such a machine is a machine to implement Newton's method.



For appropriately chosen ϵ, this machine could be used together with a counter as a subroutine to produce the illustration on the cover of this issue.

Part of the reason to have a machine defined over various rings is to have the power of mathematical analysis available for problems over $\mathbb{R}$ or $\mathbb{C}$, while having available simultaneously the well-developed theory of computability and complexity over $\mathbb{Z}$. One of our first results, motivated by recursive function theory, was on nondecidability. Given a subset of the inputs, $S \subset I$, a machine M is said to decide S, and S is said to be decidable, if on input $x \in I$, M outputs yes if $x \in S$ and no if $x \notin S$.

THEOREM 1 (BSS). *If $f(z) = \sum_{j=0}^{d} a_j z^j$ is a complex polynomial with three or more distinct roots, then no machine can decide for input $z \in \mathbb{C}$ if Newton's method will converge to a root of f, i.e., if $f(N_f^k(z)) \to 0$ as $k \to \infty$.*

Thus, the approximation to the Julia set in Figure 1 can only be an approximation.

For problems which are decidable, we have those decidable in polynomial cost in the input size. This is called the class P of decision problems. In order to define P we need two more pieces of data for our three basic examples:

Ring	Input Size	Cost
$\mathbb{Z}$	Bit	Bit
$\mathbb{R}$	Dimension	Algebraic
$\mathbb{C}$	Dimension	Algebraic

The class NP is roughly that class of decision problems which can be solved in polynomial cost in the input size. The classes P and NP are defined in analogy with the classes of Cook [3] and agree for $\mathbb{Z}$. The basic problem, does P $\neq$ NP? now makes sense in all three contexts. The importance of the class NP over $\mathbb{Z}$ is largely due to the great number of NP-complete problems of Cook [3] and Karp [4]; see Garey and Johnson [5]. Recall that a problem is NP-complete if any other problem in NP admits a polynomial cost reduction to it.

THEOREM 2 (BSS). *The following problems are NP-complete:*

Over $\mathbb{Z}$:

(a) *Given $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and a bound $b \in \mathbb{Z}$, is there a point $(x_1, \ldots, x_n) \in \mathbb{Z}^n$ such that $f(x_1, \ldots, x_n) = 0$ and $\Sigma\, x_i^2 \leq b^2$? (Bounded Hilbert's 10th)*

(b) *Given $n \times m$ integer matrix A, integer vectors b, c, and an integer k, is there an integer point x such that $Ax \leq b$ and $cx \geq k$? (Integer Linear Programming)*

Over $\mathbb{R}$:

(c) *Given $f\colon \mathbb{R}^n \to \mathbb{R}$ a polynomial of degree 4, is there an $x \in \mathbb{R}^n$ such that $f(x) = 0$? (Four feasibility)*

Over $\mathbb{C}$:

(d) *Given a system of polynomials $f_1, \ldots, f_k$ where $f_j\colon \mathbb{C}^n \to \mathbb{C}$, is there a common root of the f_j, i.e., an $x \in \mathbb{C}^n$ such that $f_j(x) = 0$ for $j = 1, \ldots, k$? (Hilbert's Nullstellensatz)*

The problems over $\mathbb{Z}$ are well known to be *NP*-complete. Using BSS machines, we give an integrated proof for (a), (c), and (d); (b) is only slightly more difficult.

In search problems over $\mathbb{R}$, we are asked not only to decide if a problem has a solution but to approximate a solution if one exists. Even to solve $x^2 - a = 0$, i.e., to compute $\sqrt{a}$ up to ϵ, we see that more and more algebraic operations are needed as $\epsilon \to 0$, since $\sqrt{a}$ is not a rational function of a.

Thus, the degree of approximation must be part of the input size of the problem. The situation for search problems in the presence of input or computational error is even more problematical. The condition of the problem, i.e., the sensitivity of the solution to perturbation of the data, will have to play a role. Even for two simultaneous linear equations, the solutions are not continuous in the data, so in the presence of error we have infinitely badly conditioned problems.

Steve Smale and I have recently done an analysis of homotopy methods for Bezout's theorem [6–8] which measures the number of steps in terms of the condition of the homotopy, gives a geometric interpretation to the condition, and gives probability estimates that a problem may be ill-conditioned.

Here is an example of our results for Bezout's problem: Find the solution lines of n homogeneous equations of degrees $d_1, \ldots, d_n$ in $n+1$ complex variables. We let $\mathcal{H}_{(d)}$ be the vector space of such systems, $(d) = (d_1, \ldots, d_n)$.

We concentrate on the incidence variety $V \subset P(\mathcal{H}_{(d)}) \times P(n)$, where $P(\mathcal{H}_{(d)})$ is the projective space of $\mathcal{H}_{(d)}$, $P(n)$ is the projective space of $\mathbb{C}^{n+1}$, and $V = \{(f,x) | f(x) = 0\}$. The unitary group acts on $\mathbb{C}^{n+1}$ with the usual Hermitian structure and induces an action on $\mathcal{H}_{(d)}$. We take a unitarily invariant Hermitian structure on $\mathcal{H}_{(d)}$ first considered by Kostlan [9]. Thus, we have the Fubini–Study metric on $P(\mathcal{H}_{(d)})$ and $P(n)$ and an induced metric on V. We replace Newton's method by projective Newton introduced in [10]. The classical condition of (f,x) where $f(x) = 0$ is determined by the norm of the inverse of the derivative of f at x. We denote the projective version of this number by $\mu(f,x)$. For a homotopy $F = f_t$, let $\mu(F)$ be the sup of $\mu(f_t, x_t)$ over all $f_t(x_t) = 0$.

THEOREM (Bez I). *The number of steps of projective Newton's method sufficient to follow a homotopy is $\leq cLD^{3/2}\mu^2$, where c is a constant < 10, $D = \max\ d_i$, and L is the length of the homotopy.*

There is also a one-root version of this theorem. Thus, if the condition of a problem is taken into account, the number of steps may depend modestly on the condition. The unitary group is crucial to our analysis. Next, we give a geometric interpretation to the condition number, generalizing work of Demmel [11, 12].

Let $\Sigma' = \{(f,x) \in V | \text{rank } Df(x) < n\}$. Σ' is the generalization of the univariate variety of (f,x) such that x is a multiple root of f. We define a vertical distance ρ to Σ' and prove.

THEOREM (Bez I). *For $(f,x) \in V$,*

$$\mu(f,x) = \frac{1}{\rho((f,x), \Sigma')}.$$

Finally, we estimate the volume of the badly conditioned problems.

THEOREM (Bez II). *The probability for $(f,x) \in V$ that $\rho(f,x) < \rho_0 < 1/\sqrt{n}$ is $\leq KnN\rho_0^2$, where $N = \dim\ \mathcal{H}_{(d)}$ and K is a small constant.*

In order to prove this volume estimate we exploit the two projections

$$\begin{tikzcd} & V \subset P(\mathcal{H}_{(d)}) \times P(n) \arrow[dl] \arrow[dr] & \\ P(\mathcal{H}_{(d)}) & & P(n) \end{tikzcd}$$

When we apply the technique to *real* homogeneous systems with the induced Riemannian structure, we have

THEOREM (Bez II). *The average number of real roots of real homogeneous systems is the square root of the number of complex roots, i.e., $\mathcal{D}^{1/2}$, where $\mathcal{D} = \Pi d_i$ is the Bezout number.*

For all the d_i equal, this is a result of Kostlan [13]. It stands in distinction to the univariate result of Kac [14] which asymptotically gives $(2/\pi)\ln\ d$, where d is the degree. The difference is the role of the unitary group.

This is but a contribution to a theory of the complexity of equation-solving. Finding adequate, realistic models and a good body of results for a complexity theory of scientific computation remains a great problem.

Now I would like to consider the theory and practice from a completely different perspective. I have pointed to Turing's Universal Machine as a spectacular example of theory developed for its own sake applied to practice. Turing and von Neumann are both frequently credited with the invention of the stored-program computer. But two engineers, J. P. Eckert and J. W. Mauchly at the Moore School of the University of Pennsylvania, had built the Eniac and were planning a follow-up machine.

Von Neumann consulted on the project and wrote a draft of a report on the Edvac which was distributed under von Neumann's name alone [14]. Joel Shurkin [15] writes:

The paper has become, arguably, the single most important document in the field. Almost every history written about the computer, or about von Neumann, credits him with the idea of a stored-memory computer. Among scientists, computers are to this day known as von Neumann machines.

Von Neumann was not the originator of the stored-program computer. As we have seen, the idea for such a machine was being discussed at the Moore School a year before von Neumann arrived on the scene, and Eckert had written a memo on the subject almost six months before von Neumann had even heard about the Moore School project.

Shurkin goes on to quote H. H. Goldstine about the report.

This report represents a masterful analysis and synthesis by him of all the thinking that had gone into the EDVAC from the fall of 1944 through the spring of 1945. Not everything in there is his, but the crucial parts are. . . .

It is obvious that von Neumann, by writing his report, crystallized thinking in the field of computers as no other person ever did. He was, among all members of the group at the Moore School, the indispensable one . . . only von Neumann was essential to the entire task.

Compare this with S. Frankel quoted by B. Randell:

Many people have acclaimed von Neumann as the 'father of the computer' (in a modern sense of the term) but I am sure that he would never have made that mistake himself. He might well be called the midwife, perhaps, but he firmly emphasized to me, and to others I am sure, that the fundamental conception is owing to Turing—insofar as not anticipated by Babbage, Lovelace and others. In my view von Neumann's essential role was in making the world aware of these fundamental concepts introduced by Turing and of the development work carried out in the Moore School and elsewhere.

I can imagine that von Neumann understood Eckert's idea better than Eckert because he certainly knew Turing's work. This would be consistent with Goldstine's quote and Frankel's. I have not tried to check the original documents. Two things are clear here, though: the enormous difficulty of tracing the intellectual origin of ideas in practice, and the value of having someone who really understands them when it comes time to apply them. In these tight economic times, as theoreticians we must not lose our confidence in the value of good theoretical work, even when it seems distant from practice in the near term.

This is how I concluded my talk at Queens College in June. The other two talks were not scheduled until October. I put my material aside, afraid to get too involved in this historical question without the skills or training of a historian, and certainly happy to be concentrating on the mathematical papers I was writing. But I could not put out of my mind the question of whether von Neumann knew Turing's Universal Machine. The assertion was plausible enough. Turing had spent two years at Princeton after solving the *Entscheidungsproblem*. Von Neumann had even asked him to be his assistant. But there was a very disturbing feature to the assertion. The Edvac report mentioned McCulloch and Pitts, who had invented neural nets in a 1943 paper, but there was no mention of Turing! It is true that McCulloch and Pitts [16] mention Turing machines and assert that their neural nets give the same computable functions. "This is of interest as affording a psychological justification of the Turing definition of computability and its equivalents, Church's λ-definability and Kleene's primitive recursiveness: If any number can be computed by an organism, it is computable by these definitions and conversely" (p. 35 of reprint). But no mention is made of Universal Machines and Turing's paper is not referenced.

Besides Frankel, Goldstine [17], p. 174, had the following to say about von Neumann:

It was his training in formal logics that made him very much aware of and interested in a result which foreshadowed the modern computer. This was contained in independent papers published by Emil L. Post and Alan M. Turing in 1936. Post taught at City College of New York and Turing was an Englishman studying at Princeton University (1936–1938). Each of them conceived of what is now called an automaton and described it in similar, mechanistic terms. The men worked independently and in ignorance of each other. There is no doubt that von Neumann was thoroughly aware of Turing's work but apparently not of Post's.

In 1937, von Neumann wrote a letter of recommendation about Turing which is quoted and commented on in Hodge's biography of Turing [18].

June 1, 1937
Sir,
Mr. A. M. Turing has informed me that he is applying for a Proctor [sic] Visiting Fellowship to Princeton University from Cambridge for the academic year 1937–1938. I should like to support his application and to inform you that I know Mr. Turing very well from previous years: during the last term of 1935, when I was a visiting professor in Cambridge, and during 1936–1937, which year Mr. Turing has spent in Princeton. I had opportunity to observe his scientific work. He has done good work in branches of mathematics in which I am interested, namely: theory of almost periodic functions, and theory of continuous groups.

I think that he is a most deserving candidate for the Proctor Fellowship, and I should be very glad if you should find it possible to award one to him.

I am, Respectfully, John von Neumann

Last week I called Goldstine and read him von Neumann's letter. He found it incredible. He was of the opinion that "Johnny would have mentioned the work if he had known it." He also said that in the times of the Edvac or the early stages of the IAS project, Turing's work was never mentioned. Only later when von Neumann lectured on automata theory did Turing's work come up. Goldstine had learned of Turing from von Neumann at some point and had read the work, but he wasn't sure when. Some other evidence that von Neumann knew Turing's work was recorded in Hodge's 1983 biography: reminiscences of Ulam from 1938, 1939, and of Frankel, who recalls that von Neumann pointed out Turing's work to him in 1943 or 1944. On the other hand Hodge relates that von Neumann claimed not to have read another paper in logic after Gödel's 1931 theorem. So another plausible reconstruction of reality is that von Neumann did not know Turing's Universal Machine at the time of the Edvac report, but when he learned of it he graciously gave him credit.

As an amateur, I've gone about as far as I can. I wonder if historians can settle the question. Hank Tropp tells me I'm opening a can of worms. Does this change my evaluation of the anticipatory value of theory in this example? Not really. In any case, Turing's work did anticipate the modern computer and is enormously important as a model of computation. Von Neumann's background in logic was surely an important ingredient to his work on the Edvac. One thing that becomes even clearer is the incredible difficulty of tracing the intellectual origins of ideas in practice.

When I gave this talk at the Courant Institute, Martin Davis gave me a copy of his excellent scholarly article [19]. Martin was more certain that von Neumann knew about the Universal Machine. We agreed that proof, as mathematicians like to have it, is hard to find.

References

1. Sutherland, S., Finding roots of complex polynomials with Newton's method, Preprint, Institute for Math. Sciences, SUNY, Stony Brook, 1989.
2. Blum, L., Shub, M. and Smale, S., On a theory of computation and complexity over the real numbers: NP-completeness, Recursive functions and Universal Machines, *Bull. Amer. Math. Soc.* (*New Series*). Also abstracted in the *IEEE 1988 FOCS* **21** (1989), 1–46.
3. Cook, S. A., The complexity of theorem-proving procedures, Proceedings 3rd ACM STOC, 1989, 151–158.
4. Karp, R. M., *Reducibility among combinatorial problems*, in *Complexity of Computer Computations*, (R. E. Miller, and J. W. Thatcher, eds.), Plenum, New York, 1972, 85–104.
5. Garey, M. and Johnson D., *Computers and Intractability*, Freeman, San Francisco, 1979.
6. Shub, M. and Smale, S., Complexity of Bezout's Theorem I: Geometrical aspects, *J. Amer. Math. Soc.* **6** (1993), 459–501.
7. Shub, M. and Smale, S., Complexity of Bezout's Theorem II: Volumes and probabilities, *Computational Algebraic Geometry* (F. Eysette & A. Galliger, eds.) Progress in Mathematics, Vol. 109, Birkhäuser Boston, (1993), 267–285.
8. Shub, M. and Smale, S., Complexity of Bezout's Theorem III: Condition number and packing, *J. Complexity* 9, (1993), 4–14.
9. Kostlan, E., Random polynomials and the statistical fundamental theorem of algebra. Preprint, University of Hawaii, 1987.
10. Shub, M., Some remarks on Bezout's Theorem and complexity theory, in *From Topology to Computation*, Proceedings of the Smalefest (M. Hirsch, J. Marsden and M. Shub, eds.) Springer, 1993, 443–455.
11. Demmel, J., On condition numbers and the distance to the nearest ill-posed problem, *Numer. Math.* **51** (1987), 251–289.
12. Demmel, J., The probability that a numerical analysis problem is difficult, *Math. Comp.* **50** (1988), 449–480.
13. Kostlan, E., On the distribution of the roots of random polynomials, in *From Topology to Computation*, Proceedings of the Smalefest, (M. Hirsch, J. Marsden and M. Shub, eds.) Springer, 1993, 419–432.
14. Kac, M., On the average number of real roots of a random algebraic equation, *Bull. Amer. Math. Soc.* **49** (1943), 314–320.
15. Shurkin, J., *Engines of the Mind*, W. W. Norton, New York, 1984.
16. McCulloch, W. S. and Pitts, W., A logical calculus of the ideas imminent in nervous activity, *Bull. Math. Biophys.* **5** (1943), 115–133. Reprinted in McCulloch, W. S., *Embodiments of Mind*, MIT Press, 1988.
17. Goldstine, H. H., *The Computer from Pascal to von Neumann* Princeton University Press, Princeton, NJ, 1972.
18. Hodges, A., *Alan Turing: The Enigma*, Simon & Schuster, New York, 1983.
19. Davis, M., Logic and the Origin of Modern Computers, in *The Universal Turing Machine. A Half Century Survey* (Rolf Hecken, eds.), Verlag Kammerer & Unverzagt, Hamburg-Berlin; Oxford University Press, Oxford, 1988.
20. Shub, M., Book review of *Elements of differentiable dynamics and bifurcation theory* by David Ruelle, *Bull. Amer. Math. Soc.* (New Series) **24** (1991), 199–211.

IBM
P.O. Box 318
Yorktown Heights, NY 10598 USA