COMPUTER ASSISTED PROOFS IN ANALYSIS

Oscar E. Lanford III
IHES

Text of a lecture delivered at the International Congress of Mathematicians 1986.

Institut des Hautes Etudes Scientifiques
35, route de Chartres
91440 Bures-sur-Yvette (France)

Mai 1987
IHES/P/87/16

Computer-Assisted Proofs in Analysis

Oscar E. Lanford III

IHES

Computers are useful in many ways in mathematical research. They make it possible, for example, to perform experiments on a wide variety of mathematical objects and to carry out or check complicated algebraic manipulations. In this talk I will describe another service they can provide: The proof of *strict bounds* on the results of (possibly very complicated) computations on real numbers. I will try to illustrate, in a concrete example, how the availability of this kind of help in proving numerical bounds opens up a new way of approaching some qualitative questions which have proved hard to treat by more standard methods.

Interval arithmetic. The techniques to be described here rest on a standard and elementary method of numerical analysis known as *interval arithmetic*. To explain this method, we begin by reviewing the rudiments of how computers perform arithmetic.

Standard computing environments provide two ways of working with numbers; they may be treated either as *integers* or as *floating point numbers*. Since the computations we will discuss deal with general real numbers and not just with integers, integer arithmetic is not directly applicable to them, and we will accordingly concentrate on floating point arithmetic[1]. Floating point numbers are manipulated and stored in a sign-exponent-fraction representation with a fixed number of digits available for exponent and fraction. The details of the representation vary, but any given format is capable of representing only finitely many numbers; we shall refer to these as the *representable* numbers in that format. Elementary arithmetic operations with representable operands often produce results with too many digits to be representable[2]. When this happens, what is normally done is to "round off" the result, i.e., to return a representable number approximating the exact result. In clean computing environments, the returned result may indeed always be the exact result "correctly rounded", i.e., that representable number which is nearest to the exact result, but this is unfortunately far from being universally the case.

Each individual arithmetic operation, then, can—and usually will—introduce some amount of *round-off error*. The user is normally assured of some bound on the amount of round-off error produced in each operation relative to the size of the returned result. It is thus possible, at least in principle, to apply elementary methods of error propagation to derive strict bounds on the error in the result of any given sequence of arithmetic operations. In practice, except for simple computations or ones with particularly transparent structure, these theoretical error estimates are usually too complicated to be feasible.

Interval arithmetic, by contrast, is a method by which the computer is programmed to generate error bounds automatically and mechanically. The idea is to carry through the computation strict upper and lower bounds for all quantities encountered, i.e., intervals guaranteed to contain the corresponding exact quantities. The end points of these intervals are representable numbers. To propagate these error bounds a step at a time it is only necessary to have available procedures for "doing elementary operations ($+$, $-$, $\times$, $\div$) on intervals." To see what this means, consider

[1] A possible alternative approach, which we will not pursue here, is to work with rational numbers, represented as quotients of integers of "arbitrary" precision, on which arithmetic operations can be done exactly.

[2] It is also possible, of course, for a result to be too big or too small to be representable with the limited range of exponents available, but we will ignore these occurrences for the purposes of this schematic review.

the case of multiplication. A procedure for multiplying intervals means one which, given two intervals $[a_1, b_1]$ and $[a_2, b_2]$ with representable endpoints, produces a third interval $[a_3, b_3]$, again with representable endpoints, such that

$$x \in [a_1, b_1] \quad \text{and } y \in [a_2, b_2] \quad \Longrightarrow x \times y \in [a_3, b_3].$$

Once a set of procedures for doing the elementary operations on intervals is available, a program for computing strict upper and lower bounds on the result of any given sequence of arithmetic operations can be constructed in a completely straightforward way by stringing together calls to these procedures.

Here is one way in which a procedure for multiplying intervals can be constructed. First form the four exact products $a_1 \times a_2$, $a_1 \times b_2$, $b_1 \times a_2$, $b_1 \times b_2$. If representable numbers have at most n digits, each of these products has at most $2n$ digits. To construct (the best possible) a_3, find the smallest of these exact products and round down to the next smaller representable number. Similarly, to construct b_3, find the largest and round up.

We have presented the above algorithm for multiplying intervals simply to show that there is nothing difficult or abstruse, in principle at least, about constructing such procedures. In practice, the method described would probably be replaced by another which is less transparent but more efficient. For example, unless both $a_1 < 0 < b_1$ and $a_2 < 0 < b_2$, it is only necessary to compute two of the four above products. Furthermore, instead of computing the full $2n$-digit products, it may be advantageous to compute n-digit approximations with controlled error—using, for example, floating point hardware—and to enlarge the interval found to compensate for the error. This latter approach is easy to program and produces fast-running code; although it need not give the smallest possible intervals, the slight loss in sharpness in the estimates obtained is not likely to be important. These practical considerations, very important a few years ago, are now fortunately becoming less so. The Institute of Electrical and Electronics Engineers (IEEE) has recently promulgated a standard for floating point arithmetic which provides, among many other things, facilities making it easy to construct interval arithmetic procedures which always produce the best possible results within the limitations imposed by the set of representable numbers. Floating point environments conforming to this standard are becoming more widely available (at least on small computers). This development promises to make good quality and efficient interval arithmetic much more accessible than in the past.

There are a few warnings which need to be issued about interval arithmetic. The first is that, although it does give *correct* bounds, it sometimes gives *excessively pessimistic* ones. It is well known to numerical analysts, for example, that the error estimates given by interval arithmetic applied in a straightforward way to solving simultaneous linear equations can be unrealistically large and that a theoretical error analysis gives much more realistic bounds. We will not pursue this example, since it involves a long and sophisticated analysis, but will discuss briefly a simpler example (due to S. de Gregorio) giving some idea of what can happen. The example is the computation of the sequence (x_n) defined by

$$x_{n+1} = x_n - h \times x_n \quad \text{where } x_0 \neq 0 \text{ and } 0 < h < 1,$$

i.e., the numerical solution of the differential equation $dx/dt = -x$ by Euler's method. It is easy to see that, if interval arithmetic is applied to this computation, the length of the interval obtained for x_{n+1} is at least $1 + h$ times the length of the interval for x_n, and that, even if x_0 and h are intervals of length zero, some x_n will have non-zero length. Thus, the interval obtained for x_n will have length which grows exponentially with n. On the other hand, a simple

analysis shows that, under very weak assumptions about round-off error, the sequence obtained by computing the x_n's in the straightforward way using floating point arithmetic gets and stays very small (as does, of course, the exact sequence). Thus: A theoretical error analysis shows that the computed sequence is a reasonable approximation to the exact sequence, while interval arithmetic gives almost no useful information.

There is another lesson to be learned from this example. The defining formula can be rewritten as

$$x_{n+1} = (1 - h) \times x_n.$$

It is not difficult to see that applying interval arithmetic to this version of the formula *does* give good estimates. This illustrates a general phenomenon: A minor rearrangement in a formula can make a major difference in the sharpness of the estimates obtained by applying interval arithmetic to that formula. I don't know any non-trivial rules for distinguishing between good and bad ways of writing a formula, but the general principle is that it is advantageous to write them with as many cancellations as possible done algebraically rather than numerically.

Another caution: Not all programs doing floating point computation can be translated in a straightforward way into useful interval arithmetic programs producing strict bounds on the results. The translation is easy to do for any program *which contains no branches conditional on the result of comparing two floating point numbers.* Difficulty arises in transcribing a conditional branch because the two intervals being compared may overlap. In principle, this difficulty could be circumvented by following both branches, but repeated conditional branches might then make it necessary to trace very many different routes and thus result in an impractically slow and complex program. Since efficient and robust methods for such tasks as inverting matrices or finding roots of equations are full of conditional branches, this difficulty is a serious one. In the two cases mentioned, however, and in many others like them, there is an effective alternative approach: First do a computation with ordinary floating point arithmetic to find an approximate solution; then use interval arithmetic (together with some kind of perturbation theory) to find an interval around the computed solution which is guaranteed to contain an exact solution. In fact, even when the straightforward direct application of interval arithmetic is feasible, this latter approach often gives much better results.

The Feigenbaum fixed point. We turn now to a discussion, by example, of what is involved in using interval arithmetic to prove a qualitative mathematical result. The example to be discussed, which was the first application of the set of ideas being described here (see Lanford[6]), is the proof of the following

THEOREM. *The Feigenbaum-Cvitanović functional equation*

$$g(x) = -\frac{1}{\lambda} g \circ g(-\lambda x)$$

admits an even analytic solution

$$g(x) = 1 - 1.5276\cdots x^2 + 0.1048\cdots x^4 + 0.026\cdots x^6 + \cdots$$

We are not going to discuss here the application of this theorem to dynamical systems theory—see the contributions of Eckmann and Sullivan to these proceedings—but we will begin with a few orienting remarks about the functional equation.

First of all, it has a scale invariance: If $g(x)$ is a solution, so is $\gamma g(x/\gamma)$ for any non-zero constant γ. Hence, if we are interested only in solutions with $g(0) \neq 0$, there is no loss of

generality in assuming that $g(0) = 1$. Once this normalization condition is imposed, putting $x = 0$ in the equation gives $\lambda = -g(1)$. Hence, we can rewrite the functional equation as

$$g(x) = \frac{1}{g(1)} g \circ g(g(1)x) \equiv \mathcal{T}g(x),$$

i.e., g is a fixed point for the operator $\mathcal{T}$.

It is a fact of experience that any reasonable approach to solving this equation numerically works. For example, take n distinct points in $(0, 1]$ and look for a polynomial of degree n in x^2 which is 1 at 0 and satisfies the equation at these n points. This is a system of n non-linear equations in n unknowns to which Newton's method can be applied. In practice, Newton's method converges for any reasonable initial guess; what it converges to doesn't depend much on the choice of points; and the approximate solution obtained in this way shows every sign of converging as n goes to ∞.

This certainly suggests that the equation does have a solution. Nevertheless, it has turned out to be surprisingly difficult to prove the existence of this solution by standard qualitative methods. The first proof of existence used estimates proved by computer; it is this proof which will be described here. Subsequently, several successively less computational proofs have been given, and the most recent of these, due to H. Epstein[3], makes no use at all of a computer or even of a calculator. Thus, for the *existence* of a solution, computer assistance is no longer necessary. For applications to dynamical systems theory, however, it is not enough to know that the operator $\mathcal{T}$ has a fixed point g; one also needs to know that the derivative $D\mathcal{T}(g)$ of $\mathcal{T}$ at g is hyperbolic with one-dimensional expanding subspace. The original computer-assisted proof established this along with existence. Although a certain amount of progress has been made on this question by qualitative methods, no complete conceptual proof has so far been given.

The first step in proving the existence of g is to convert the functional equation to a fixed point problem for a *contraction*. (The operator $\mathcal{T}$ itself is not contractive near g; it has one expanding direction.) We use a simplified version of Newton's method. Newton's method itself for solving $\mathcal{T}g = g$ amounts to searching for a fixed point of

$$f \longmapsto f - [D\mathcal{T}(f) - 1]^{-1}[\mathcal{T}(f) - f]$$

by iteration. We simplify this iteration by replacing $D\mathcal{T}(f)$ by a fixed, explicit linear operator Γ, and we observe that a fixed point of $\mathcal{T}$ is the same thing as a fixed point of

$$\Phi(f) = f - [\Gamma - 1]^{-1}[\mathcal{T}(f) - f].$$

A simple calculation shows, moreover, that

$$D\Phi(f) = [\Gamma - 1]^{-1}[\Gamma - D\mathcal{T}(f)],$$

and from this it is easy to see that

To show that Φ maps the ball of radius ρ about g_0 contractively into itself, it suffices to show that

E1. $\|[\Gamma - 1]^{-1}[\Gamma - D\mathcal{T}(f)]\| \leq \kappa < 1$ for $\|f - g_0\| \leq \rho$.

E2. $\|\Phi(g_0) - g_0\| \leq \rho(1 - \kappa)$.

Hence, to prove the existence of a fixed point g, all we have to do is to find g_0, Γ, ρ, and κ so that these two inequalities hold. Here, roughly, is how we choose them: We take for g_0 an accurate

approximate fixed point; we then take Γ to be a good enough approximation to $D\mathcal{T}(g_0)$ to make the estimate (E1) hold at least for $f = g_0$ and ρ small enough so that it continues to hold on the ball of radius ρ about g_0. Estimate (E2) can then be expected to hold provided that g_0 was a good enough approximate fixed point. Both inequalities are proved with the aid of computer estimates; for purposes of exposition, we will concentrate on the first of them.

The above analysis is abstract and general; it applies in any Banach space. The next step will be to chose the space in which we work. There is first of all a trivial reduction. The candidates f for the fixed point satisfy some conditions: They are even functions taking the value 1 at 0. We want to make a space with these conditions built in. Thus, we write

$$f(x) = 1 + x^2 h(x^2).$$

Replacing f by h as the "independent variable" is simply a linear change of variable, and we can reasonably allow h to vary over an open set in some Banach space. The operator $\mathcal{T}$ acting on functions f induces an operator acting on the corresponding h's, which we will also denote by $\mathcal{T}$.

We are looking for an *analytic* fixed point, and so we will work in a space of analytic functions. In fact, we choose to work in a space of functions $h(t)$ analytic on a disk of radius 2.5 about 1. This choice cannot be justified on general grounds; it is made on the basis of a careful (but "non-rigorous") numerical study of the fixed point (together with considerations of convenience). Evidently, we would not have chosen this domain if we had not had good reason to believe that the fixed point whose existence we were trying to prove is analytic on $\Omega = \{x : |x^2 - 1| < 2.5\}$. This domain has another, less obvious, property which plays an essential role in making our argument work: The mapping $x \mapsto g(-\lambda x)$ sends a neighborhood of $\bar{\Omega}$ into Ω. One consequence is that the right hand side of the functional equation

$$g(x) = -\frac{1}{\lambda} g \circ g(-\lambda x)$$

gives an analytic continuation of g from Ω to a strictly larger domain, and control over g on Ω automatically gives control on this larger domain. A second consequence is that $\mathcal{T}$ is differentiable on a neighborhood of g and the derivative $D\mathcal{T}(f)$ is a compact operator if f is close enough to g on Ω.

Finally, we choose a norm; we put

$$\|h\| = \sum_{n=0}^{\infty} |h_n| \quad \text{where} \quad h(t) = \sum_{n=0}^{\infty} h_n \left(\frac{t-1}{2.5}\right)^n.$$

The principal reason for using this ℓ^1 norm, rather than the supremum norm, is that it makes it easy to estimate norms of operators: If T is a linear operator, then

$$\|T\| = \sup_n \|Te_n\| \quad \text{where} \quad e_n(t) = \left(\frac{t-1}{2.5}\right)^n.$$

We now choose an approximation Γ to $D\mathcal{T}(g)$ which, in the basis (e_n), is represented by a matrix with only finitely many non-zero entries. The choice of Γ, again, is guided by numerical computation. What we need to do, then, is to estimate

$$\sup_n \|[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n\|$$

for all f in some given (small) ball about an approximate fixed point g_0. We do this in two steps: We make a relatively crude estimate on $\|[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n\|$ valid for all sufficiently large n; we then estimate this quantity by detailed computation for the finitely many n's not covered by the large-n estimate. Both estimates require the use of the computer. We will concentrate on the second step; the first is simpler, but depends on the specific form of $D\mathcal{T}(f)$.

For any given n, and an explicit Γ, it is not difficult to write an explicit formula of $[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n$. This formula is complicated but is built up in a straightforward way from a sequence of elementary operations on functions—such operations as pointwise sums and products, composition, etc. A convenient way to organize the programming of bounds on $\|[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n\|$ is to devise a data structure, generalizing intervals for real numbers, giving a finite representation for a class of regions in function space, and to construct procedures for estimating the results of elementary operations on these regions. The formula for $[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n$ can then simply be transcribed into a sequence of calls to these procedures to get a program for estimating this quantity.

Roughly, we specify these regions in function space by giving upper and lower bounds on Taylor series coefficients up to some fixed order together with an upper bound on the sum of the absolute values of all coefficients of higher order. That is: We first we pick an N, the degree up to which the coefficients will be bounded individually. Once N has been fixed, regions are specified by giving N pairs of representable numbers:

$$\ell_0 \le u_0, \ell_1 \le u_1, \ldots, \ell_{N-1} \le u_{N-1},$$

together with a non-negative representable number ϵ. The corresponding region, which we denote by $\mathcal{U}(\ell_0, u_0, \ldots, \ell_{N-1}, u_{N-1}, \epsilon)$ is the set of functions h with Taylor series

$$h(t) = \sum_{n=0}^{\infty} h_n \left(\frac{t-1}{2.5}\right)^n$$

such that

$$\ell_i \le h_i \le u_i \quad \text{for } i = 0, \ldots, N-1, \text{ and } \quad \sum_{n=N}^{\infty} |h_n| \le \epsilon$$

(In fact, although this representation would work, a more complicated one giving better estimates is usually used in practice.)

The sorts of procedures needed to estimate the results of elementary operations on these regions can be illustrated by the pointwise product. We need a procedure which, given two of these representable regions $\mathcal{U}_1$ and $\mathcal{U}_2$, produces a third $\mathcal{U}_3$ such that

$$u_1 \in \mathcal{U}_1 \text{ and } u_2 \in \mathcal{U}_2 \Longrightarrow u_1 \times u_2 \in u_3.$$

Constructing such a procedure, starting from ordinary interval arithmetic, is straightforward in principle and not very difficult in practice. Once these procedures are available for a set of about ten elementary operation on functions, it is easy to write a program estimating $\|[\Gamma - 1]^{-1}[D\mathcal{T}(f) - \Gamma]e_n\|$ for any given n, and this, as we have already noted, is just what is needed to complete the particular branch of the proof which we have been describing.

Other applications. Strict bounds proved by computer with the aid of interval arithmetic have been applied to a variety of problems besides the one described in the preceding section. Among these are:

1. Existence and properties of solutions of other functional equations arising from renormalization group analyses in dynamical systems theory: Eckmann, Koch, and Wittwer[1], Eckmann and Wittwer[2], Mestel[10], Lanford and de la Llave[7].
2. Existence of a non-Gaussian fixed point for a hierarchical lattice field theory: Koch and Wittwer[5].
3. Stability of a semi-relativistic quantum mechanical model for matter (via estimates on the behavior of solutions of some explicit ordinary differential equations): Fefferman and de la Llave[4].
4. Non-existence of invariant circles for area preserving maps: MacKay and Percival[9].
5. Realistic estimates on the sizes of Siegel linearization domains: de la Llave and Rana[8].

Concluding remarks. I would like to close with some general remarks about these techniques. Should an argument like the one described above be regarded as a valid proof? There are a number of reasons for reluctance to accept proofs using these methods. One of them arises from the widespread misconception that a "computer proof" is nothing more than a numerical experiment, i.e., that errors are controlled by empirical or heuristic methods rather than by strict mathematics. I hope that the preceding discussion has made it clear that this is not the case. It seems to me that the argument outlined above would certainly be accepted a valid proof if all the verifications were carried out by hand (provided, of course, that all details in the analysis can be filled in and that no mistakes are made in carrying out the verifications). There remain, nevertheless, a number of genuine issues.

First: Using a computer to perform verifications raises practical problems of reliability. Computers are complicated devices, and many things can go wrong. It is certainly not possible to give absolute assurances that errors cannot happen. Nevertheless, with reasonable care, it is possible to reduce the *probability* of error to a very low level.

Second: There is a problem about how such a proof can be communicated and to what extent the result is reproducible, notably on computers different from the one on which it was first given. The communication problem can be solved by communicating the program, typically written in a high-level language. The reproducibility question has two answers, a formal one and an informal one. The strict formal answer is to specify completely the details of the floating point environment and the interval arithmetic operations. If this is done, it is possible in principle—albeit perhaps outrageously difficult in practice—to repeat the verification on a different computer[3]. The informal answer is that many results, such as the existence of the Feigenbaum fixed point, can be made to follow from estimates which are not very critical. In such a case, the high-level language program can be expected to verify these estimates successfully independent of the details of the floating point environment in which it is run. The program is thus a sketch of the proof; verifications on different computers are different ways of filling in the details. The expectation that any reasonable approach will succeed in proving the estimates is of course only heuristic; if it is not realized, it is always possible to fall back on the complete specification. It is, after all, only necessary to prove the desired estimates with *one* correct set of interval arithmetic operations.

Finally: It is possible to take the position that the only acceptable proofs are those in which all steps are carried out by humans, and thus to reject all computer assisted proofs. This position seems to me to be perfectly defensible. Like other restrictive views of what techniques should be accepted in mathematics, however, it has to be weighed against the benefits derived from

[3]This kind of strict reproducibility becomes in fact quite practical if both computers provide a set of floating point operations conforming to the abovementioned IEEE standard.

the use of these techniques. For example: Proving the existence and basic properties of the Feigenbaum fixed point was not an end in itself; it served as a starting point for other analyses, carried out by standard qualitative methods. In this case, the computer-assisted proof served to eliminate a bottleneck which would otherwise have prevented further progress in this area for many years.

Deciding to accept a computer-assisted proof as valid does not mean that one has to be *satisfied* with it. Proofs vary in the extent to which they are conceptual or computational. Computer assisted proofs are certainly on the extreme computational end of this scale. It seems to me that it is always desirable to replace a computational proof by a more conceptual one, and especially so when the computational proof requires the assistance of a computer.

It is worth noting, nevertheless, a difference between a computer-assisted proof and a computational proof carried out by hand. In a computer-assisted proof, there is a clean separation between the conceptual part of the analysis and the part which is simply a mechanical verification. Furthermore, the steps to be performed in the mechanical part of the proof must be specified completely and unambiguously. It is thus possible to read and understand the theoretical analysis establishing that a certain computation suffices to prove some desired result without actually going through that computation. While this can also be true of computational proofs done by hand, it need not be and in practice there is a strong tendancy to mix the analysis and the computation. Thus, a carefully constructed computer assisted proof may be more transparent than a computational proof done by hand even if the amount of computation done in the former is much larger.

References

1. J.-P. Eckmann, H. Koch, and P. Wittwer, *A computer-assisted proof of universality for area preserving maps*, Memoirs AMS **47** (1984).
2. J.-P. Eckmann and P. Wittwer, "Computer Methods and Borel Summability Applied to Feigenbaum's Equation," Springer Lecture Notes in Physics 227, Springer-Verlag, Berlin, 1985.
3. H. Epstein, *New proofs of the existence of the Feigenbaum functions*, Commun. Math. Phys. **106** (1986), 395–426.
4. C. Fefferman and R. de la Llave, *Relativistic stability of matter I*, Preprint, Mathematics Department, Princeton University, 1985.
5. H. Koch and P. Wittwer, *A non-Gaussian renormalization group fixed point for hierarchical scalar lattice field theories*, Commun. Math. Phys. **106** (1986), 495–532.
6. O. E. Lanford III, *A computer-assisted proof of the Feigenbaum conjectures*, Bull.Amer.Math.Soc., New Series **6** (1982), 427–434.
7. O. E. Lanford III and R. de la Llave, *Solution of the functional equation for critical circle mappings with golden ratio rotation number*, In preparation.
8. R. de la Llave and D. Rana, *Proof of accurate bound in small denominator problems*, Preprint, Mathematics Department, Princeton University, 1986.
9. R. S. MacKay and I. C. Percival, *Converse KAM: theory and practice*, Commun. Math. Phys. **98** (1985), 469–512.
10. B. D. Mestel, *A computer assisted proof of universality for cubic critical maps of the circle with golden mean rotation number*, Ph.D. Thesis, Mathematics Department, University of Warwick, 1985.

91440 Bures-sur-Yvette France