What is an Answer? An essay on the Theory of Functions

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
96-035

WHAT IS AN ANSWER?
AN ESSAY ON THE THEORY OF FUNCTIONS

John R. Rice

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR 96-035
June 1996

WHAT IS AN ANSWER? An Essay on the Theory of Functions

John R. Rice
Department of Computer Sciences, Purdue University
West Lafayette IN 47907, U.S.A.

Abstract

Historically the fields of numerical and symbolic methods for scientific applications have been widely separated, nearly isolated from one another. Symbolic (and algebraic) computation has used symbolic methods to obtain symbolic answers to problems. Numerical computation has used numerical methods to obtain numerical answers. The natures of the methods and answers obtained are seen by most to be completely distinct. One goal of this paper is to raise doubts that any such distinction actually exists. The focus is on problems whose answers are functions, e.g., temperature distributions or pressure fields, from phenomena modeled by differential or integral equations. A wide range of mathematical models exist for functions, both from mathematics and computer science. All these theories are candidates to be used to define answers, so we examine the nature of functions that are useful to obtain from either symbolic or numeric computation. We conclude that all these theories for functions fail to model the real world properly.

1. Introduction

This paper considers computing answers to problems whose solutions are functions. Typical example problems are ordinary or partial differential equations, integral equations, etc. These problems arise from mathematical models of a wide variety of physical phenomena. These models have equations whose functional solutions could, in principle, be computed to infinite (unbounded) precision. Over the centuries a vast array of approximate methods have been devised for these problems, e.g., infinite series, asymptotic expansions, finite differences, solving simpler related problems, finite elements, and analog machines. To simplify the notation, we assume all these problems involve two independent variables, x and y, but nothing depends on this assumption in an important way.

The recent increases in computing power have greatly expanded the possibilities of using these solution methods and it is natural to compare them in some general way. In order to make a comparison, we need a definition of an answer that does not depend in any way on the method being used to solve a problem. This is the first issue addressed in this paper. Keep in mind that methods were developed for about two centuries with very, very few problems actually being solved. Thus it should not be a surprise that many of these methods produce mirages instead of answers, i.e., there is no feasible way to obtain useful answers from many of them. The advent of computers has allowed many such problems to be solved and recently it is practical to solve them automatically and in wholesale.

In developing a definition of an answer we are led naturally to consider more carefully what a method is and then what a computable function is. The goal of problem solving is to obtain an answer (which is a function) that can be computed. There exists a rich theory of functions in mathematics and other theories in computer science. These theories are inadequate for the understanding of finding answers to our class of problems. The problem arises because we need to understand both the cost and accuracy of answers obtained. Problems with "simple" answers should be cheap to solve with high accuracy while problems with "complicated" answers might not be. The sticky point turns out to be the meaning of "simple" and "complicated" and the incompatibility of the "theoretical" definitions of complexity with "real world" observations. Recall that the test of validity of a theory is how well it represents the real world and we claim that existing theories are inadequate in meeting this test.

The paper is organized as follows. In Section 2 the kinds of answers are surveyed and discussed followed by a definition of a method. We conclude that numerical and symbolic methods are really the same.

Section 3 presents several possible definitions of an answer and discusses complexity issues. Four views of a function are discussed in Section 4: mathematical, computer science theory, programming languages, and real world. In Section 5 definitions of a function are considered which more closely model the real world. The goals of each definition are discussed. Throughout this material we present examples of the difficulties with existing and potential definitions of answers and functions. Section 6 contains brief remarks on the underlying difficulty that these theories face and note its similarity to the difficulties that have led us not to have a satisfactory theory for finite precision arithmetic.

2. What is an Answer?

An answer to the problem considered here is something that can be evaluated at an arbitrary point. For example, in two variables, it could be $f(x,y)$ or, if the problem has parameters, it could be temp(x, y,eps, thickness, type). These look like ordinary mathematical functions, we next consider various types.

2.1. Symbolic Answers

The simplest of all answers are *exact*, e.g., $f(x,y) = 2x + 4y$. Here we ignore the problems of finite precision arithmetic (round-off), we assume the computations are real numbers exactly. In the final section we briefly discuss the relationship with finite precisions arithmetic. Thus the exact answers are arithmetic expressions in terms of the independent variables, known constants, and known parameters. The computation only involves $+$, $-$, $/$, $\times$, and logical comparisons; these are the only computational operations that a computer (or a human) can do.

An answer such as $f(x,y) = 3\pi x + 4(y + \gamma)$ is not exact since both π and γ (Euler's constant) are not known exactly. We can call them *almost exact* since the difficulty is obtaining exact values of such constants, a difficulty more closely related to finite precision arithmetic than function theory.

The simplest symbolic answers are expressions involving standard functions, e.g., $f(x,y) = 3\sin(4x) \times \log(x/y)$. We call these *finite symbolic functions.* The symbols "*sin*" and "*log*" are the names of functions that are assumed to be easily computable. However, they cannot, in general, be computed exactly even on a machine with real arithmetic. In fact, these symbols are the names of computations which represent problems which, every one agrees, that are "solved". Thus, in a computer programming language, the execution of a statement which includes $sin(x)$ involves calling a library routine to evaluate it. The *sin* and *log* functions are easily computable but this is not true of special mathematical functions in general. Over the centuries scientists have named dozens of functions, many of which are quite hard to compute.

Example 1. The Gamma Function. Even the common Gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt$$

is quite tricky to evaluate for a general value of x. In [2] there are many representations of $\Gamma(x)$, namely, three integral expressions, one limit, one infinite product, two asymptotic expansions, one continued fraction, and three approximations. There are similar representations for related functions (e.g., the Digamma function and the Incomplete Gamma function) and many formulas relating various values (e.g., $\Gamma(x+1) = x\Gamma(x)$). Along with 10 large pages of symbolic formulas there are 30 pages of tables. Only the asymptotic expansions (e.g., Sterling's formula) can be used as a basis for effectively computing $\Gamma(x)$ in general and even this is a delicate (but computationally cheap) computation. The other representations are not useful for computing $\Gamma(x)$. ●

Finally there are the *infinite symbolic expressions* such as in Example 1 or the following

$$1 + \sum_{i=1}^{\infty} x^i/i!$$

$$f_1(x,y) = \sum_{\substack{i=1\\ j=1}}^{\infty} \frac{(i+j)x^i \sin(jy + x/i)}{(ij)!}$$

$$f_2(x,y) = \sum_{\substack{i=1\\ j=1}}^{\infty} \frac{(1+\sqrt{x+jy})(-1)^{i-j}}{(ij+\pi ix+3jy^2)\log(i+x+2(j+y))} \qquad x,y,\geq 0$$

The first infinite series is the basis for making $exp(x) = e^x$, a standard mathematical function, easily computable. The function $f_1(x,y)$ is also easily computable but $f_2(x,y)$, which appears similar, cannot be computed cheaply at all. Using the arithmetic of actual computers, this formula for $f_2(x,y)$ is useless.

2.2. Numerical Answers.

Many people think of a numerical answer as a table of values, much like those referred to in Example 1 for the Gamma function. However, a table of values is not a function and thus not an answer. But, a table of values *plus* an interpolation formula can be evaluated at any point and is thus a possible answer. The fact that many accept tables as answers might be because many believe that good interpolation formulas are easily obtained. This is not actually so (consider relatively sparse, irregularly spaced data in three dimensions) but obtaining interpolation formulas also is not an issue here.

It appears that a numerical answer depends on ϵ, the accuracy of the answer, explicitly while symbolic answers do not. This appearance is due to the usual notations used and, in fact, both kinds of answers involve ϵ in the same way. To illustrate this, let PDE denote the data (domain, operator, boundary conditions) of a second order elliptic partial differential equation. Then the numerical answer could be expressed as

$$f(x,y) = Interp\ (x, y, 5\text{–}point\text{–}star\ (PDE), \epsilon))$$

where *Interp* is a standard interpolation algorithm and *5-point-star* is a standard finite difference algorithm. A corresponding symbolic answer is

$$f(x,y) = \sum_{\substack{i=1\\ j=1}}^{k(\epsilon)} [coef\ (PDE) x^i y^j]$$

where *coef* is a standard algorithm for Taylor Series coefficients. Both of these answers have an explicit dependence on ϵ and both have a "hidden" two dimensional array of numerical values that is implied once ϵ is set.

2.3. What is a Method?

Above we gave a name to a numerical method to make a numerical answer look like a symbolic answer. This might appear to be "cheating", but it is, in fact, a standard procedure in mathematics. The names "*sin*", "*log*", and "*exp*" stand for methods (e.g., infinite series with truncation) to compute these functions. We have stated already that an answer is something that can be evaluated, let us say "something" here means at least an algorithm in the sense of a computer program. Accepting the imprecision of defining an algorithm, we can make a definition as follows.

DEFINITION 1. A method for solving a problem is an algorithm that computes an algorithm for an answer to the problem.

Of course, we are considering a class of problems without exact answers, so we can make this definition more precise as follows:

DEFINITION 1A. Consider a problem with defining data D and an accuracy specification ϵ. Then a method is an algorithm which takes (D, ϵ) as input and computes an algorithm $f = f(x, y) = f(x, y, D, \epsilon)$ so that (a) f can be evaluated for any (x, y) in the problem domain, and (b) the difference between $f(x, y)$ and the exact solution of the problem is at most ϵ.

The algorithm used to obtain $f(x, y)$ can be of any type; symbolic or numerical. But what is the difference between symbolic and numerical methods? In programming language circles one tends to think as algorithms expressed in Lisp as being symbolic and those expressed in Fortran as being numerical. Yet it is a basic result in computer science that everything that can be computed in Lisp can also be computed in Fortran and vice versa. Examining typical methods, one might say that numerical methods tend to approximate the continuum "early" while symbolic methods tend to do it "late".

We conclude that the distinction between symbolic and numerical answers is only apparent and not real. It is more a matter of the culture of the methodology. One can hope with symbolic solutions that some useful information can be obtained without evaluating any functions at all. This does happen in especially simple cases, but as one considers more complex problems these occurances becomes very rare. One finds that looking at 10 pages of polynomials is just as informative (or uninformative) as looking at 10 pages of numbers. That is, the first step in numerical methods is usually to discretize the continuum while for symbolic methods the discretization (numerical evaluation of expressions) is often delayed as long as possible.

3. Possible Definitions of an Answer

The first definition of an answer is taken directly from the initial, informal statement of Section 2.

DEFINITION 2. An answer is an algorithm $f(x,y)$ that can be evaluated for every (x,y) in the problem domain.

Recall that by algorithm we mean $f(x,y)$ can be computed by a program.

In theory this is a viable definition but in practice it is unacceptable. The solution of these problems is a surface and an answer must be able to produce the whole thing, as least approximately. That is, one must be able to plot the solution. An answer from Definition 2 can take unbounded time to evaluate as (x,y) varies. Thus there must be some uniformity in the time to compute specific values. This holistic nature of answers leads to the second definition.

DEFINITION 3. Let $\delta \geq \epsilon$, be the evaluation accuracy. An answer is an algorithm $f(x,y)$ that can be evaluated with accuracy δ and with a fixed cost for every (x,y) in the problem domain.

This definition does not imply anything about the effect of problem parameters on the cost. It is unrealistic to ask for the cost to be independent of parameters in all cases, many physical phenomena exhibit wide ranges in complexity as parameters change.

The principal motivation for defining answers precisely is so that one can compare methods on the basis of cost, both the computational cost of the method itself and the computational cost of evaluating a solution. It is plausible that evaluation can be quite efficient using a "table plus interpolation" approach. Even the effects of making ϵ

smaller are modest; the table might grow in size and/or the interpolation formula grow in complexity but the growth normally should be rather slow.

The table plus interpolation approach seems simple but what about problems whose solutions are truly and essentially non-uniform in nature. Real world problems have phenomena like turbulence, singularities, boundary layers, and chaos all of which make uniform evaluation cost more difficult. The next definition takes such situations into account.

DEFINITION 4. Let $\delta \geq \epsilon$ be the evaluation accuracy. An answer is an algorithm $f(x,y)$ that can be evaluated with accuracy δ and with cost commensurate with the complexity of $f(x,y)$ for every (x,y) in the problem domain.

This definition sounds good and ordinary scientists and engineers can understand it. Yet it has the imprecise phrase "cost commensurate with the complexity of $f(x,y)$". The difficulty is that there are intuitive and theoretical notions here which do not match well.

4. What is a Function?

Four views of functions and how to measure their complexity are considered. The real world is represented by scientists who use functions naturally and have an intuitive idea of what it means for a function to be "simple and well behaved" or "complicated". They just examine plots of the functions. It is these people who are the experimentalists; their views determine the success of theoretical models of functions and their complexity. We are, after all, discussing abstract models of real things; these models should not produce nonsense and must be compatible with real world observations.

4.1. Mathematical Models of Functions

Mathematics has a very elaborate structure and theory for functions. This theory has evolved over centuries and moved away from its experimental roots to be very non-constructive. Thus this theory admits such irrelevancies as: (a) non-constructible functions like $g(x) = 0$ on the rationals and $g(x) = 1$ on the irrationals, (b) theorems that are true except for sets of measure zero — and the set of all real world functions is of measure zero. However, the essential shortfall of this theory is the fact that it does not properly abstract the real world concept of a simple, well behaved function.

Mathematics measures complexity of functions in two equivalent ways: (1) using smoothness as measured by the number of continuous derivatives a function has, (2) using polynomials as prototypes and measuring complexity by the speed (in terms of error as a function of polynomial degree) of approximating functions by polynomials. That these measures are equivalent follows from the classical results in Dunham Jackson's 1911 thesis [4], see [6] or Chapter 5 of Vol. 1 of [11] for complete presentations of this theory. With this approach, the simplest or smoothness common class of functions besides polynomials are the entire functions. Entire functions are infinitely differentiable everywhere and are approximated exponentially fast by polynomials. Yet it is well known that given any $\delta > 0$ and any functional curve of thickness δ drawn on the interval $[0, 1]$, say, there is an entire function entirely inside this curve. In other words, all real world functions are simple and well behaved!

Example 2. The Function $1/(1+x^2)$

Consider $f(x) = 1/(1+x^2)$ on the interval $[-5, 5]$. It defines a very smooth, simple curve; any reasonable measure of complexity of functions must indicate that this is a simple function. It varies between 0 and 1 with only two inflection points. Its slope changes sign at $x = 0$, has two intervals of monoticity, and varies between -0.5 and 0.5. Its second derivative varies between -2 and 2. Let us measure how the speed of convergence of approximating f by polynomials and how well the polynomials resemble it. That this function is "troublesome" for polynomials was discovered by Runge a century ago when he showed that the polynomial interpolants to f on equi-spaced points do not converge.

It is well known that the interpolants of f at the Chebyshev points $x_i = -10\cos(\pi(2i+1)/(2N+2))$, $i = 0, 1, 2, \ldots, N$ provide polynomials of degree N which converge to f at the fastest possible rate. The following table provides some data about these polynomials. Note that, for $N = 20$, the errors for f' and f'' are 61% and 636%, respectively, of the maximum values for f. For the values of f near the ends the errors are 1000% and 56,500%, respectively.

	Degree N				
	4	6	10	16	20
Polynomials shape					
– Changes in sign of slope	4	6	10	10	10
– Number of inflection points	2	4	8	14	16
Errors at ends of interval					
– f value	0.165	0.102	0.044	0.013	0.006
– f' value	0.687	0.922	0.949	0.545	0.308
– f'' value	0.99	2.76	7.71	12.76	12.71

We see that while the difference between f and polynomials is not huge, it is not as small as we would expect for a really simple function. Much more disturbing is that the nature and shape of the polynomials do not at all resemble those of $1/(1+x^2)$. ●

Many mathematicians would claim this is an unfair conclusion, that one must do more than use this classification of functions and use norms of derivatives as well. They would say, for example, that a simple and well behaved function is one that has a "fourth" derivative bounded by, for example, 3.5. There are two comments to make this alternative. First, the natural classification system of rough to smooth is not being used. Second, no one has an intuitive idea of what it means to have a fourth derivative bounded by 3.5. Further, the choice of "fourth" is quite arbitrary.

Example 3. Approximating Physical Data

A test of the modeling capability of polynomials is reported in [7] using real world (physical) functions. Over 60 one dimensional functions were selected from various sources for scientific data (e.g., Handbook for Physics and Chemistry). All of these functions were simple and well behaved in the intuitive sense. These were approximated by various mathematical forms including polynomials. Over 50% of these functions could not be approximated satisfactorily by polynomials. Not satisfactory is defined to mean one of the following: (a) The polynomial approximated the data with satisfactory accuracy but had high frequency, low amplitude oscillations not present in the data. (b) Satisfactory accuracy was not obtained with polynomials of degree less than 100 or so. In other words, the mathematical measure (low polynomial degree required) for simplicity did not correspond to the real world, intuitive measure for simplicity. This experience has been confirmed many times over the years by this author but not reported in the literature.

Example 4. The Power of Adaptive Numerical Methods

It is now well known that adaptive numerical methods are extremely effective (efficient) for a variety of problems (e.g., quadrature, approximation, solving differential equations). See [1], [8] for example results. Yet there are theorems in the mathematics literature which seem to prove that this is not true. These theorems go as follows: Let S be the set of all functions f with $||f^{(4)}|| \leq 4$ (we use the 4s for concreteness). Then, with probability 1.0, an adaptive method applied to integrate f is no better than a known non-adaptive method. We use integration of f for this example but the result and method of proof apply more generally. To understand how such a theorem could be true mathematically and yet wrong in the real world, we divide S into two parts. Let S_1 be the f in S whose fourth derivatives are everywhere discontinuous and S_2 the remainder of S. We know that S_2 is of measure zero in S so the theorem is true if it is true for f in S_1. It is not too hard to prove the result for S_1 as adaptive methods do not work well for functions with bad behavior *everywhere*. It is their nature to identify locations of bad behavior of f and to do something special (and good) there. Now we can understand this apparent contradiction, the real world functions are *all* in the set of measure zero where this theorem's result does not apply. ●

These three examples show that the mathematics theory of functions does not correctly abstract the intuitive notion of a simple and well behaved function.

4.2. Theoretical Computer Science Model of Functions

Theoretical computer science has the concept of a computable function and one could hope that this would serve as a good model for the functions of interest here. However, these functions do not compute with real numbers; rather the computation of $f(x)$ takes in the digits of x and produces (one at a time) the digits in the expansion of f. This model is not useful for real world computation and has some unexpected aspects. For example, every computable function is continuous! So every computable function is infinitely differentiable. This is due to the fact that a question like "Is $x > 1/7$" is undecidable and hence not contained in any program for a computable function. This model also excludes the function $f(x) = 1$ if $x > 0$, $f(x) = 0$ if $x \leq 0$. Clearly this model is not useful here.

4.3. Programming Languages Models of Functions

Most high level programming languages appear to have functions in them and one could hope that these would serve as good models for the functions of interest here. However, functions are not very compatible with the discrete view of computation popular in theoretical programming languages. Indeed, high level programming languages severely restrict the nature of functions; they are not ordinary variables to be manipulated in a natural mathematical way. Interestingly, part of the problem is that computer languages tend to expand the nature of functions so as to allow side effects. That is, a computer program evaluating $sin(3.62)$ could cause printing, files to be created or destroyed, or the computation to be stopped altogether. Such behavior is not part of the traditional view of functions and not needed for our purposes. However, it is not so easy to prevent a computer program for a function from having side effects.

The problems considered here cause another difficulty for programming languages. Computing their answers is not compatible with the idea that the results of a computer program are either correct or incorrect. In our context, answers only have different levels of accuracy, levels that often cannot even be measured easily or even reliably.

More seriously, these models of functions provide no measures of simplicity or behavior related to answers of the problems at hand.

5. Possible Definitions/Structures for (Real World) Functions

The view that counts is that of the problem solvers, the scientists and engineers who use functions. There is little interest or awareness among them of measure theory, Turing machines, or Ada. There is interest and awareness among them about real world functions with properties like "smooth", "well behaved", "singular", "boundary layers", "unstable", etc., even though these terms have imprecise meanings. A straightforward approach to functions would be to say these are what computers compute. More precisely we would have:

DEFINITION 5. A function f is a piecewise rational expression using (finite) total N of constants (polynomial coefficients and breakpoint locations). The complexity of f is N.

Alternatively, one could require that the rational expressions be of fixed degree and form. Accepting this definition in general would discard about 25,000 person centuries of research in mathematics. Many good ideas and results from mathematics would be lost and it is not clear that the overall situation would be improved.

5.1. Attributes of a Useful Definition for Functions

Actually, the need is not so much for new definitions of functions but rather for a new structure (classification system, measure of complexity) for functions that better models the real world. Thus the first attribute desired for a new definition/structure for functions is:

ATTRIBUTE 1. The definition/structure must model the properties of real world expressions such as simplicity versus complexity in shape and cost of evaluation at a point.

Although a new definition/structure must be computationally oriented, it should not be intimately involved with any particular model of discrete arithmetic. The underlying model of arithmetic is thus the four arithmetic operations: $+$, $-$, $\times$, $\div$, and logical comparison: $=$, $>$, $<$, applied to real numbers (these need not be precisely the same real numbers as used now in mathematics). The goal is to separate the issues of function theory from those of models for the computational number system. Thus the second attribute is

ATTRIBUTE 2. The definition/structure is based on arithmetic and comparisons of real numbers.

One source of the current unsatisfactory structure in the mathematical theory of functions is the process of completing and closing the set of functions. These processes greatly simplify the theory and allow for elegant mental models. They also bring in functions which exist only as mental models; they cannot be computed nor do they have analogs in the real world. A few such functions could be tolerated but, in fact, these artificial functions can dominate in the theory. The set of real world functions is of measure zero in the current mathematical theory and thus there is the danger of proving theorems which look like they apply to all functions when they actually apply to no real world functions. We have

ATTRIBUTE 3. The definition/structure need not be closed or complete in the usual mathematical sense.

Two of the principal goals of theoretical computing are to:

- prove that algorithms are correct or have certain properties.
- find optimal (or nearly optimal) algorithms (from a given class of algorithms) for solving various problems.

These goals are very hard to achieve even for many problems where questions about modeling the real world do not exist. They are especially difficult in numerical analysis because of the weaknesses in models for arithmetic and functions. These goals are nevertheless of equally great importance in numerical analysis and we have

ATTRIBUTE 4. The definition/structure allows for realistic analysis of the correctness and efficiency of algorithms for solving problems involving functions.

Example 3 show one form of unrealistic analysis that results from the current mathematical theory of functions. Another form of unrealistic analysis permeates numerical analysis now and is discussed in the next section.

5.2. Verifiable Hypotheses for Numerical Analysis

About 20 years ago I heard a talk where it was pointed out that the hypotheses of many, if not most, theorems in numerical analysis are unverifiable. The speaker then went on to question the value of theorems whose hypotheses cannot be verified. He even observed that really striking theorems can be established within a theory that allows such hypotheses in a general way. The person giving the talk attributed the observation to a third person. In the past five years I have searched (not too vigorously) for the origin of this idea or its discussion in the literature. The two persons whom I recall being involved say they have never heard of the idea.

Many theorems in numerical analysis have hypotheses like

If $f(x,y)$ has four continuous derivatives then...

This is unverifiable from the mathematics point of view, there is no theoretical or practical way to test this property. From the point of view of computable function theory, it is a tautology. Every computable function is infinitely differentiable. We ignore the fact that the arithmetic on real computers makes every function discontinuous as the usefulness of numerical analysis depends on keeping function theory isolated in some way from computer hardware arithmetic. This does not automatically make the theorems in numerical analysis nonsense, many theorems have the hypotheses in the form of

...then the error is bounded by "stuff" $\times ||f^{(m)}(x,y)||$

which uses the norm of the mth derivative. At least here one immediately sees that one cannot use the theorem if one does not know the value of the norm involved.

I pose a number of rhetorical questions which I think illustrate that the users of numerical analysis (and perhaps the Professors of Numerical Analysis also) pay little attention to hypotheses of such theorems:

- How many times have you calculated or estimated the value of norm of the fourth derivative?
- Suppose you were given plots of 10 functions and asked to estimate the norm of their fourth derivative? What is the average accuracy you would expect your estimates to have?
- Do you know a reasonably reliable way to compute estimates of the norm of the fourth derivative of a given function?

- Do you know a reasonably reliable way to compute estimates of the norm of the fourth derivative of a function defined by a given second order ordinary differential equation?

One goal of a new structure for functions is to produce theorems with hypotheses that are more frequently verifiable. Such hypotheses should include properties like

- *Characteristic lengths.* These are lengths over which the function behavior is truly simple.
- *Number of pieces.* An explicit count for functions defined using logical operations.
- *Symbolic/Geometric properties.* Pieces might be convex, have three inflection points, be polynomials of degree three, etc.
- *Number of singularities.* The types might also be included.
- *Characteristic sizes.* The size of the domain of definition, the size of the function, etc.
- *Uncertainty measure.* This might be zero for some functions that define a problem, it is unlikely to be zero for the computed results.

Further, the properties should be naturally measurable and visualized. Thus, $||f(x)||$ is acceptable while $||f^{(4)}(x)||$ is not; one knows what $||f||$ means intuitively and can usually estimate it well. These properties are representative of the information that actually exists (or can be expected to be readily obtained) for science and engineering computations.

An example of hypothesis properties is illustrated in Figure 1. Here we have a function of size two on a domain of size four which has four convex pieces, a characteristic length of 0.5, one singularity and an accuracy (uncertainty) of 0.01. Note that the properties illustrated here include several geometric items. Geometric information is commonly available about functions and often provides very useful insight into the behavior and nature of functions. Unfortunately, we do not have well developed measures, tools, etc., for many important geometric properties.

Example 5. A Theorem with Verifiable Hypotheses

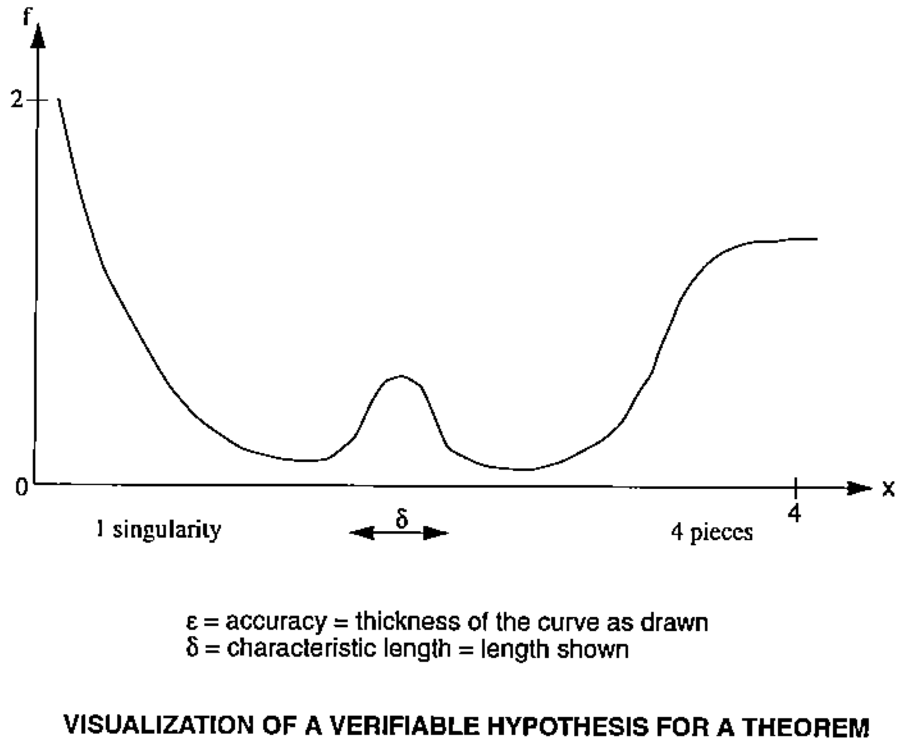

Figure 5.1: Plot of a function with verifiable properties. It has size 2, domain size 4, four convex pieces, one singularity, characteristic length of 0.5 and uncertainty of 0.01.

In [9] a broad class of adaptive algorithms for numerical integration is defined and analyzed. This class is called the metalgorithm. The hypotheses used are examples of verifiable hypotheses as discussed here. An example result is given here with the assumptions about the metalgorithm omitted except as they involve function properties; these other assumptions are rather lengthy and stating them would require describing many details of the metalgorithm.

Consider a function $f(x)$ with singularies $S = \{s_i | i = 1, 2, \ldots, R\}$ and set $w(x) = \Pi(x - s_i)$, $i = 1, 2, \ldots, R$. We say that $f(x)$ has p continuous derivatives except for a finite number of algebraic singularities if: (i) If x_0 is not in S then $f^{(p)}(x)$ is continuous in a neighborhood of x_0. (ii) There are constants K and $\alpha > 0$ so that $|f^{(p)}(x)| \leq K|w(x)|^{\alpha - p}$. Let If be the exact integral of $f(x)$ and $Q_N f$ be the estimate produced by the metalgorithm using N evaluations of $f(x)$. Then we have

Theorem 7 [9]. *Consider a function $f(x)$ with characteristic length $\lambda(f)$ and with p continuous derivatives except for a finite number of algebraic singularities. Consider the metalgorithm with the error bound used only when the length of the subinterval involved is less than $\lambda(f)$. Let EPS be the error bound of the metalgorithm. Then we have $|If - Q_N f| \leq EPS \leq O(1/N^p)$.*

This result is then applied to three existing algorithms [3, 5, 10] to prove quickly their convergence. One of these [10] is a trapezoidal rule algorithm (and thus has $p = 2$) whose entire description and assumptions involve only geometric terms (convex, concave, inflection and cusp) with no derivatives or singularities. ●

6. Conclusion

An argument has been given that new definitions and/or structure are needed for the theory of functions. The fact is that the current theories for functions fail to model the real world properly. The difficulty is much easier to identify than to correct. The inherent difficulty is shared in computation for both functions and arithmetic. In both cases we have a basic core of experience that is simple, familiar, and well understood. Things in this core are excellent models for the real world. For functions this core consists of low degree (≤ 4) polynomials, low order (≤ 2) differentiation, and a few special functions ($sin, log, ...$). For arithmetic this core consists of numbers with a few digits (≤ 6), integers, and a few special numbers (π, e, $\sqrt{2}$, $\ldots$).

Then in both cases we have logically complete and mathematically eloquent theories that includes everything. These theories are so big that the real world analogs in them are of negligible relative size. The theories are dominated by artificial entities and, in the case of function theory, the real world is not modeled properly.

What is needed (and very hard to create) is something in between. We must be able to "cut off" infinitely, give up the elegance of closed and complete systems, and yet be able to consider and analyze "arbitrarily" large, complicated or accurate computations.

References

[1] A. Chen and John R. Rice, On grid refinement at point singularities for h-p methods, *Intl. J. Num. Meth. Engr.*, **33**, (1992), pp. 39–57.

[2] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Vol. 55, Nat. Bureau Standards, Applied Mathematics Series, 1964.

[3] Carl W. deBoor, CADRE: An algorithm for numerical quadrature. In *Mathematical Software* (J. Rice, ed.) Academic Press, NY (1971), pp. 417–449.

[4] Dunham Jackson, *Über die Genauigkeit der Annäherung stetiger Funktionen durch ganze rationale Funktionen*, Ph.D. Thesis, Göttingen, 1911.

[5] James N. Lyness, SQUANK (Simpson quadrature used adaptively-noise killed), Algorithm 379, *J. Assoc. Comp. Mach.*, **16**, (1969), pp. 260–263.

[6] John R. Rice, *The Approximation of Functions*, Addison Wesley, Reading, MA, Vol. 1, 1964 and Vol. 2, 1969.

[7] John R. Rice, Approximation formulas for physical data, *Pyrodynamics*, **6** (1968), pp. 231–256.

[8] John R. Rice, On adaptive piecewise polynomial approximation. In *Theory of Approximation with Applications* (A. Law and B. Sahney, eds.), Academic Press (1976), pp. 359–386.

[9] John R. Rice, A metalgorithm for adaptive quadrature, *J. Assoc. Comp. Mach.*, **22**, (1975), pp. 61–82.

[10] John R. Rice, An educational adaptive quadrature algorithm, *SIGNUM Newsletter*, **8**, (1973), pp. 27–41.

[11] A.F. Timan, *Theory of Approximation of Functions of a Real Variable*, MacMillan, New York, 1963.