Numerical Methods, Complexity, and Epistemic Hierarchies

Nicolas Fillion and Sorin Bangu*†

Modern mathematical sciences are hard to imagine without appeal to efficient computational algorithms. We address several conceptual problems arising from this interaction by outlining rival but complementary perspectives on mathematical tractability. More specifically, we articulate three alternative characterizations of the complexity hierarchy of mathematical problems that are themselves based on different understandings of computational constraints. These distinctions resolve the tension between epistemic contexts in which exact solutions can be found and the ones in which they cannot; however, contrary to a persistent myth, we conclude that having an exact solution is not generally more epistemologically beneficial than lacking one.

1. Introduction. In *Extending Ourselves*, Humphreys set two desiderata for a scientifically informed philosophical approach to science: (1) "in dealing with issues concerning the application of mathematical models to the world, as empiricists we should drop the orientation of an ideal agent who is completely free from practical computational constraints of any kind," and yet (2) we should not "restrict ourselves to a minimalist position where what is computable is always referred back to the computational competence of human agents" (Humphreys 2004, 124). While the second desideratum (the rejection of 'minimalism') has potentially transforming consequences for how empiricist philosophers look at science, here we will heed the first one—that we should take seriously the 'computational constraints' confronting the sci-

*To contact the authors, please write to: Nicolas Fillion, Simon Fraser University; e-mail: nfillion@sfu.ca. Sorin Bangu, University of Bergen; e-mail: sorin.bangu@fof.uib.no.

†The current work integrates the two papers we presented at the symposium The Plurality of Numerical Methods in Computer Simulations and Their Philosophical Analysis held at IHPST Paris, November 2011. We thank Julie Jebeile and Anouk Barberousse for organizing the conference. We also thank the audience for this symposium as well as audiences at the WCPA 2014 and the PSA 2014 meetings. In particular, we would like to thank Bob Batterman, Audrey Yap, and Paul Humphreys for very helpful comments.

Philosophy of Science, 82 (December 2015) pp. 941–955. 0031-8248/2015/8205-0018$10.00

entists. Indeed, when it comes to extracting useful information from mathematical equations describing systems of interest, one often fails to notice how significant is the difference between merely proving that a solution exists and proving that a computational route to access it is available.

In addressing the issue of computability, it is essential to draw attention to the important distortions to our understanding of the scientific enterprise that result from identifying the epistemic subject with an 'ideal agent' possessing unlimited calculational resources. Even though this has only recently become the focus of extensive work in philosophy, this epistemic dimension has played a prominent role in the works of the founders of modern computation theory, such as Turing: "the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it . . . is a very useful assumption under many circumstances, but one too easily forgets that it is false" (1950, 451). Since the computational route to the solution can be easier or harder to navigate for different types of mathematical problems, it is natural to organize more or less tractable problems into a complexity hierarchy.

Our article articulates and compares different ways in which the computational complexity hierarchy might be understood. We further emphasize that we must also consider theoretical computational constraints, in addition to the practical ones. As we will show, there are constraints that persist no matter how much computational power is available and regardless of how much the epistemic subjects are idealized. In other words, such constraints are grounded in objective facts about what types of solutions may be obtained for a given problem. To clarify the nature and consequences of such limitations, it is important to understand the relations within the variety of exact solutions (algebraic, elementary, closed form, analytic, etc.) and the methods used to obtain and justify numerical solutions that are typically not exact.

2. Exactly Solvable and Unsolvable Problems. After a mathematical model is generated, the issues of interest come up when the scientist tries to extract useful information from it—or, in scientific parlance, when she tries to 'solve' the model. As is well known, it is only for very simple and drastically simplified models that the equations can be solved exactly (i.e., that an explicit formula can be obtained as a solution).[1] The textbook example is the simple perfect frictionless pendulum, with small amplitude of oscillation. Its equation $\theta'' = -(g/L)\theta$ admits such a solution, namely, $\theta(t) = \theta_{\max}\sin(g/Lt)^{1/2}$. For even slightly less idealized models, such as the simple pendulum of arbitrary amplitude, the equation of motion $\theta'' = -(g/L)\sin\theta$

1. We will clarify what we mean by "solving the equation with an explicit formula" in sec. 2, where types of exact solutions are distinguished.

is not tractable in the same straightforward way. One should then use more sophisticated mathematical machinery to explicitly express an exact solution or instead appeal to numerical methods (such as a Runge-Kutta algorithm) to obtain (hopefully) approximate solutions. In other cases, only such inexact numerical values for the solutions are obtainable by computation, which gives rise to a different epistemic context. The distinction between the two epistemic contexts can be amply illustrated. Here are several well-known scientific examples belonging to each category (Trefethen 2008, 605). Cases in which we can find an explicit formula that solves the problem include solving systems of n linear equations and n unknowns, minimizing n-variable linear functions subject to m linear constraints, and so on. Cases in which no explicit solution is available include finding eigenvalues of $n \times n$ matrices, minimizing functions of several variables, evaluating arbitrary integrals, solving ordinary and partial differential equations, and so on.

It may seem natural to think that the scientist operating in contexts of the second type finds herself in an epistemically disadvantaged position, especially when compared to the circumstances in which explicit exact solutions are available. In this way, this binary distinction—between mathematical problems that afford exact, explicit solutions and those that do not—gives rise to the belief that there is an epistemic hierarchization of the two contexts. It is easy to see why one might presume that possessing exact, explicit expressions for solutions is preferable to not possessing them: we seek these solutions because, presumably, they immediately reveal information about the behavior of systems. Since inexact solutions might fail to do so by not being genuinely informative (when their error is too large), exact solutions are deemed epistemologically superior. The central aim of what follows is to challenge this belief and maintain instead that the difference is not as pronounced as one might think, since even when explicit solutions are available, numerical considerations (typically associated with inexact solutions) cannot be avoided. We argue that the mere existence of an explicit, exact solution does not necessarily improve one's epistemic position, as complications often occur. These complications require that we adopt a perspective on the role of equations (and their solutions) according to which this hierarchization is rather illusory. To this end, we first turn to a closer examination of the two epistemic contexts.

To begin, it is important to understand some nuances about the notion of an exact solution. There are many types of exact solutions to a mathematical problem, affording different epistemic advantages. It is not uncommon to see the phrases "exact solution," "algebraic solution," "analytic solution," and "closed-form solution" used interchangeably in the literature. Typically, these phrases are used to characterize an epistemological context in which exactness prevails (i.e., in which approximations are of no concern). Yet there are important differences between them. Among the phrases above,

"exact" is the most general term that refers to any mathematical objects that satisfies the conditions constitutive of the problem; the other terms denote particular cases of exact solutions arranged as shown in figure 1.

Consider an arbitrary problem that happens to have a unique function as its exact solution (as is the case for "nice" initial-value problems). The problem is said to have an algebraic solution if the solution can be written as a finite combination of algebraic operations. So, the question whether there is an algebraic solution depends on what is in the class of algebraic functions. Note that the property "having an algebraic solution" depends on the existence of an expression of a given type that captures the function that solves the problem (and not only on the existence of a solution). The same can be said for elementary and closed-form solutions. The classes of admissible operations for expressions of the solution are as follows:

- Algebraic expressions admit the following operations: addition, subtraction, multiplication, division, and exponentiation with integral and fractional exponents;
- Elementary expressions admit all elementary algebraic operations, plus exponents and logarithms in general (and so they include trigonometric and inverse trigonometric functions as well);
- Closed-form expressions include all closed-form expressions, plus many other "well-understood functions," in particular the so-called special functions (but not arbitrary limits or integrals).[2]

From their mutual relations, we see that it might be the case that some problems have a closed-form solution without having an elementary solution and that some problems have an elementary solution without having an algebraic solution.

For example, consider a fourth-degree polynomial,

$$p(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4, \tag{1}$$

and a fifth-degree polynomial,

$$q(z) = b_0 + b_1 z + b_2 z^2 + b_3 z^3 + b_4 z^4 + b_5 z^5. \tag{2}$$

In the sixteenth century, Ferrari showed that there are explicit formulas giving the roots of equation (1) in terms of radicals, while no such formulas

2. Borwein and Crandall (2013) review seven different approaches to defining what the class of closed-form solutions contains; importantly, those disagree on the extension of the class. Moreover, as they report (from Weisstein), "an infinite sum would generally not be considered closed form. However, the choice of what to call closed form and what not to is rather arbitrary since a new 'closed-form' function could simply be defined in terms of the infinite sum" (50). The idea is that, at a given stage of development of mathematics, any function that is well understood is to be considered closed form.

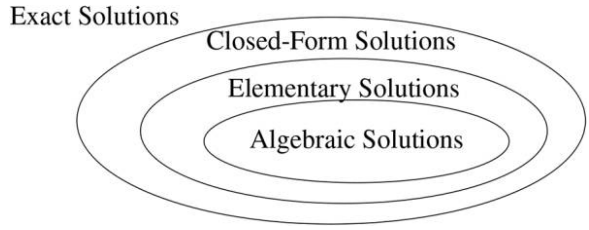

Figure 1.

exist for the roots of equation (2) (as proved by Abel and Ruffini at the turn of the nineteenth century). Thus, one might think that finding the roots of p is a different kind of task than that of finding the roots of q.

In technical terms, we say that whereas the quartic has an *elementary* solution, the quintic does not. The Abel-Ruffini Impossibility Theorem shows that fifth-degree polynomials generally have no algebraic solutions (i.e., they have no solutions expressible with algebraic expressions, which include radicals). That is of course not the same as saying that there is no solution, because the existence of a solution is guaranteed by the Fundamental Theorem of Algebra. Importantly, the Impossibility Theorem states that the solution cannot be expressed using a form that is particularly convenient for the sake of calculations. Moreover, it does not only say that no such expression has been found so far; rather, it says that no such expression will ever be found. Yet, there is an exact solution, and it turns out to be closed form since it can be expressed in terms of Jacobi elliptic functions (as shown by Hermite in the mid-nineteenth century). So, we can write down an explicit formula for the roots, provided that we use special functions in this expression (and so it is not elementary).

This being said, a look at how such roots are obtained in practice today reveals that solving equations (1) and (2) is not that different after all. If an engineer wants to find the roots of either of these polynomials, she will use a computer. But, while the computer cannot use an explicit (elementary) formula in the case of q, it does not necessarily use one in the case of p—since it might turn out that the use of such a formula slows down the actual computation of the result or provides spurious roots due to numerical instability (we return to this point later).[3] Thus, in practice, the possession of an explicit formula does not immediately translate into an epistemic advantage.

As we have indicated above with the example of the pendulum, some simple physical systems do not have closed-form solutions. The perfect pen-

3. For instance, in virtually any numerical analysis text, one finds an argument explaining that computing roots of quadratics using the quadratic formula is ill advised, for it may lead to what is known as 'catastrophic cancellation'.

dulum with small angle of oscillation is described by a differential equation that turns out not to have an elementary solution.[4] However, it does have an exact solution, in terms of Jacobi elliptic functions, which is closed form. Be that as it may, instead of being satisfied with this exact nonelementary solution, physicists often approximate the problem by taking the limit $\theta \to 0$. Then, the model equation reduces to the simple harmonic oscillator that has the simple elementary solution $\theta(t) = c_1 \sin \omega t + c_2 \cos \omega t$, where $\omega = \theta'$. Interestingly, if we take a simple harmonic oscillator and then add a linear factor to the model equation, we can again have a situation that has no elementary solution. This is also easy to imagine in a physical setup. If you consider a mass attached to a Hookean spring, the model equation would be $x'' = -x$ (suppose the stiffness k is 1), a simple harmonic oscillator. However, in real systems, stiffness is not constant. We can try to understand what would happen if the stiffness increased linearly with time, so that $x'' + tx = 0$. It turns out that the solution of this model equation is the Airy function, which cannot be expressed as an elementary function. Thus, small changes in the physical circumstances can drastically alter the kind of solution afforded by the model equations.[5]

When there is an exact solution, but no elementary solution, it is necessary to rely in some way on infinite series representation of the solution to evaluate it at some time. With respect to calculations, the difficulty with infinite series representations is that we cannot sum an infinite number of terms. It then seems that we can evaluate the solution to arbitrary accuracy by using increasingly long (but finite) truncated series. An interesting situation arises when we have a perfectly good analytic solution in the form of a uniformly convergent Taylor series, which converges so slowly that it ends up being of no practical use for computation. The Airy function mentioned above is a good example of this. Numerically, even if the series converges for all x, it might be of little practical use, since the theoretical uniform convergence might not translate to success in numerical contexts.

Thus, to the extent that we need concrete numerical values, calculability is crucial. Calculability is conceptually most straightforward when we have an expression that we can evaluate, and this obtains when we have finite expressions capturing solutions (i.e., when we have algebraic or perhaps elementary solutions). The requirement of exactness is insufficient to the extent that it allows for solutions that cannot be expressed finitarily. However, as we discuss below, even finitarily representable solutions do not guarantee that no problems will arise in actual calculation.

4. Another famous example of this situation is the global solution of the n-body problem provided by Wang (1990). It is analytic but does not have a closed-form representation.

5. Another interesting perturbation of the equation leads to Duffing's equation, which also has a character that supports our argument.

The upshot of this discussion is this: when only qualitative behavior is of interest, exact solutions are not very important. But, when quantitative information is required, exact solutions will in general not give us a straightforward recipe to obtain numbers. This recipe would be within reach, however, if the exact solution could be captured by (or given in the form of) an algebraic or elementary expression. This is why, even when a problem is susceptible of receiving an exact solution, applied mathematicians often approximate the description of the system in order to derive model equations that have a closed-form solution or, even better, elementary or algebraic solutions. However, this implies that, for small changes in our description of the system, the character of the solutions can change significantly. But, given that mathematical modeling is an activity practiced in a context where uncertainty is always present, this means that our emphasis on an exact solution will not, in general, guarantee the computability of accurate numerical results from explicit exact solutions. These considerations lead us to a more flexible and inclusive way of dealing with the extraction of quantitative information from mathematical representations inspired by the works of numerical analysts, as well as to different ways of characterizing the hierarchy of computational complexity.

3. Computational Cost and Numerical Stability. In order to characterize alternative, complementary ways of understanding complexity hierarchies, we distinguish among three forms of equivalence of mathematical problems. We will call them *mathematical* equivalence, *computational* equivalence, and *numerical* equivalence.[6] To grasp the difference between the first two notions of equivalence, consider the following expression, known as the *sample variance* of a set of n values:

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2, \tag{3}$$

where $\bar{x}$ is, as usual, the arithmetic mean of the n values, $\bar{x} = (1/n)\sum_{i=1}^{n} x_i$. Calculating the variance using expression (3) involves passing through the set of data twice (first to compute $\bar{x}$, then to accumulate the sum of squares). Another formula is sometimes considered as a replacement for equation (3):

$$s_n^2 = \left(\frac{1}{n-1}\right)\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]. \tag{4}$$

Formulas (3) and (4) are of course mathematically equivalent, since the graphs of the two right-hand-side formulas are identical, as can be shown by

6. The distinction between mathematical and numerical equivalence is a standard one in numerical analysis. See, e.g., Dahlquist and Bjorck (1974, 48).

elementary manipulations. But expression (4) seems to offer a computational advantage: to compute it, we pass through the data only once. We then say that (3) and (4) are not computationally equivalent: (4) is, from this perspective, preferable to (3).

Typically, such computational advantages are measured by counting the number of arithmetic operations; in numerical contexts, this is known as the "flop count," where 'flop' stands for 'floating-point operation'. Formula (3) necessitates $n - 1$ additions and one division by n to compute $\bar{x}$, n subtractions and n multiplications for the squaring, $n - 1$ operations for the sum of squares, and one division by $n - 1$, for a total of $4n$ operations. But, as one can easily check, formula (4) requires only $3n + 2$ operations, so we see that the computational cost of formula (3) is larger than that of (4). Therefore, although (3) and (4) are, strictly speaking, not computationally equivalent, both computational costs are linear functions, so the difference between the two is often of no practical importance, even for very large collections of data, since modern computers can process gigaflops (10^9 flops) per second.

Sometimes, however, the difference between computational costs is very important indeed. What matters is the difference between orders of computational cost. The two methods to compute the sample variance above had a linear cost, which we denote by $O(n)$. But if we instead considered an algorithm having a quadratic cost ($O(n^2)$), a cubic cost ($O(n^3)$), or more generally a polynomial cost ($O(n^k)$) or even an exponential cost ($O(c^n)$), the difference would be more dramatic (see fig. 2). Such considerations lead to a second way of characterizing a hierarchy of complexity of mathematical problems.

Consider such a case, namely, the problem of finding the determinant of a matrix $A \in \mathbb{R}^{n\times n}$, which arises very often in science. We consider two methods. The first method—a recursive method known as Laplacian determinant expansion by minors, the method taught in introductory linear algebra—uses this formula to compute the determinant:

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} M_{ij}, \tag{5}$$

where i is any row along which we expand, and M_{ij} is the determinant of the $(n - 1) \times (n - 1)$ matrix obtained by crossing out row i and column j. This sum requires a linear number of operations; that is, the cost is a function $ax + b$, or simply $O(n)$. But this requires computing M_{ij}, which is itself the determinant of an $(n - 1) \times (n - 1)$ matrix, so that the summation for this number will require $O(n - 1)$ operations. Again, it will involve computing the determinant of $(n - 2) \times (n - 2)$ matrices, which costs $O(n - 2)$ operations. So, including all the steps of the recursion, we have a cost of

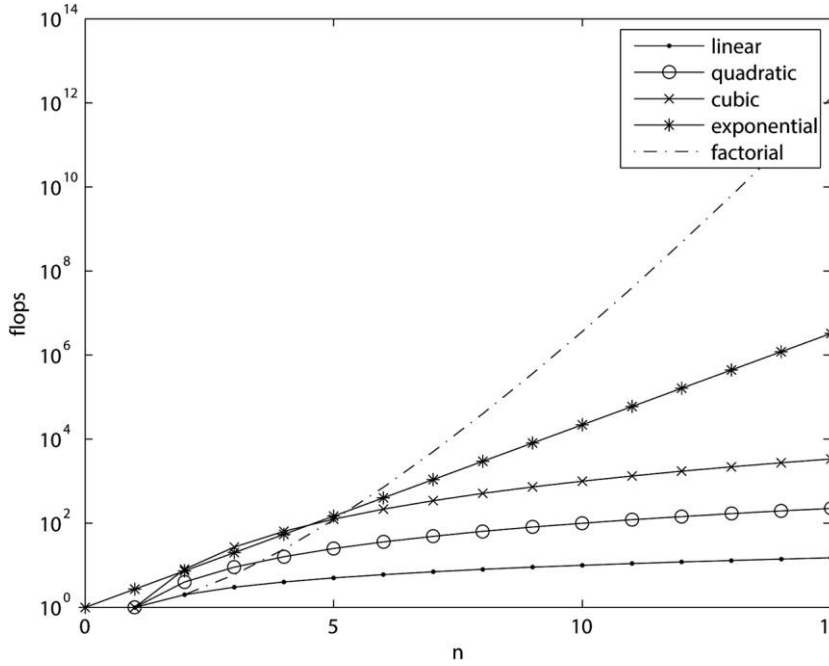

Figure 2. Logarithmic plot of different orders of computational cost.

$O(n \times (n-1) \ldots 2 \times 1) = O(n!)$, and factorial cost is even larger than exponential. Clearly, for even a small system, this method will be prohibitively computationally expansive. However, one could use a more affordable strategy—Gaussian elimination—to transform the matrix A into an upper-triangular matrix B. Note that the elementary transformations involved in the transformation do not change the determinant, so that the problem of finding $\det(A)$ and $\det(B)$ are mathematically equivalent. However, $\det(B)$ is simple to find: it is the product of the n diagonal entries. Thus, since the reduction to a transformed matrix uses $O(n^3)$ operations (see, e.g., Corless and Fillion 2013, chap. 4), we have an algorithm using cubic cost. This is hugely inferior to the cost of the Laplacian row expansion method. For instance, for a small 12×12 matrix, the latter method requires about 1,700 flops, while the Laplacian expansion method requires about 500 million flops. These are certainly not computationally equivalent.

As we have seen, mathematical equivalence regards the calculations as abstract operations, while computational equivalence takes into account some practical computational constraints. Yet the notion of numerical equivalence goes even deeper into these constraints, and, in fact, this is the notion that underlies modern approaches to scientific computing (as opposed to Turing-

style computation and complexity theory). In fact, there is a computing tradition grounded in numerical analysis that significantly differs from and complements the computation theory that is more familiar to philosophers of science (see, e.g., Blum 2004). Below we emphasize key aspects of the concept of numerical stability and how it leads to a different characterization of computational complexity.

In numerical terms, computational limitations are often given by specifying the number of digits available to represent numbers; this is made essential by the fact that digital computers are unable to handle the infinite number of decimals that a real or complex number may have. Thus, real and complex numbers are replaced by finite "machine numbers."[7]

For the purposes of a simple example, suppose that we work within the confines of what is called '8-digit fixed-point arithmetic'. That is, we have only 8 digits available to represent numbers, so numbers longer than 8 digits get chopped, which gives rise to a round-off error. Under this constraint, let us consider the following set of three values: $x_1 = 10{,}000$, $x_2 = 10{,}001$, and $x_3 = 10{,}002$. For these values, formula (3) computes $s_3^2 = +1$, and formula (4) computes $s_3^2 = 0$. As is immediately evident, this discrepancy is due to the fact that chopping discards the last digits in some cases. This shows that, even if (3) and (4) are mathematically equivalent and almost computationally equivalent in the sense given above, they are not numerically equivalent.

This situation raises interesting conceptual questions: What is the value of s_3^2 after all? We noted that in certain circumstances (a large amount of data), two mathematically equivalent expressions are associated with different computational speeds—and in these circumstances (4) is to be preferred to (3) (although, as we saw, the difference in practice is not very significant). In yet other circumstances (8-digit arithmetic and data of a certain type), the numerical equivalence of the two expressions is lost, despite their mathematical equivalence. Therefore, one must advance a principled reason to select the 'right' expression when calculating concrete values. What is this principle then?

Before answering the question, note that modern scientific computing generally uses floating-point arithmetic and not fixed-point arithmetic as in the example above.[8] If we work in, say, a standard 16-digit floating-point

7. This has important consequences that cannot be neglected; indeed, in limited-precision arithmetic, many field laws do not hold true, e.g., associativity of addition and cancellation of multiplication. Thus, a computer arithmetic has a different algebraic structure.

8. See, e.g., the first appendix and the first chapter of Corless and Fillion (2013). The main advantage of floating-point numbers over fixed-point numbers is that they cover a broader range of approximate real values.

arithmetic, there are design standards that guarantee that basic operations such as addition, subtraction, multiplication, division, square root, and so on, are correctly rounded.[9] In other words, for any basic real-valued operation $*$: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$,[10] it will be the case that

$$fl(x * y) = x * y + \Delta = x * y(1 + \delta), \tag{6}$$

where Δ is a round-off error that is at most half the unit in the last place,[11] and δ is the equivalent relative error given by

$$\delta = \frac{\Delta}{x * y} = \frac{fl(x * y) - x * y}{x * y}. \tag{7}$$

So, the calculated floating-point value of such basic operations will be within $1 \pm 1.1 \times 10^{-16}$ of the exact value. When multiple operations are chained together, we obtain a *cumulative round-off error* in terms of the individual round-off errors. If implementing a problem in floating point using a certain algorithm (or formula) results in a relatively small cumulative round-off error, then we say that the algorithm or formula is *numerically stable*. So, the notion of numerical equivalence necessitates that we consider sets of problems/functions and their relations, and not only a single function written in mathematically equivalent forms.

Let us examine a simple problem that will exemplify how one chooses between solutions obtained from numerically inequivalent problems. Its careful analysis will point toward the general principle underlying the analysis of scientific computation. Suppose then that we are in the happy circumstance that we can solve the model equations of a physical process, and we obtain an explicit solution f:[12]

$$f(x) = x(\sqrt{x + 1} - \sqrt{x}). \tag{8}$$

Suppose, further, that for some reason we are interested in calculating the value $f(500)$. If we had unlimited computational resources, such a value would be given in the form of an infinite array of decimals. However, this

9. One such standard that is widespread in the software industry is the IEEE-754 standard.

10. The point generalizes to operations of arbitrary (but finite) arity and to complex numbers.

11. The unit in the last place, denoted *ulp*, is a constant determined by the parameters of a system of floating-point arithmetic. In a standard 16-digit floating-point system using a binary basis, it is about 1.1×10^{-16}. Another related quantity is also used, namely, the "machine epsilon."

12. Matthews and Fink (1999, 28) use this example to illustrate loss of significant figures, but we use it instead to make a more general point about numerical stability.

value is simply not available to a finite epistemic agent computing it on a finite physical machine. So, within the current context, the question is in fact what is the value $f(500)$, given certain computational constraints. We can call the value supposedly obtainable by an ideal (Platonist, if you will) mathematician the 'true' value, but we have to think about what the actual, computable value is, under the given constraints. Suppose our constraint is that we work in a 6-digit floating-point arithmetic in base 10. Then, the computed value would be

$$f(500) = 500(\sqrt{501} - \sqrt{500}) = 500(22.3830 - 22.3607) = 11.1500. \quad (9)$$

Let us introduce the notation $\varphi_c(\alpha)$ standing for the 'computable' value of φ at point α, subject to constraints C. Then, $f_c(500) = 11.1500$.

At this point, suppose we discover a different expression g,

$$g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}, \quad (10)$$

which, as is immediately clear, is mathematically equivalent to f. In fact, we may regard it as merely an accident that we first expressed the solution as a computation of the expression for f. The solution could have been obtained in the form g to begin with. We can put this in terms of epistemic symmetry: imagine that Fred and Ginger solve the problem in parallel, perhaps in different rooms, and that Fred writes down the solution as f while Ginger obtains it as g. Epistemically speaking, there seems to be perfect symmetry between them. Yet, a simple calculation shows that when Ginger computes $g_c(500)$, she gets

$$g_c(500) = \frac{500}{\sqrt{501} + \sqrt{500}} = \frac{500}{22.3830 + 22.3607} = \frac{500}{44.7437} = 11.1748. \quad (11)$$

The problem we face now is of the type introduced above: which of the two values, 11.1500 or 11.1748, should we use (e.g., to build a bridge)?

Recall the aim we stated at the outset: to raise a challenge to the belief that it is preferable to have an analytic solution available because, allegedly, in this case we do not need to appeal to error-theoretic considerations. We just made such an assumption, that we are in the happy circumstance of being able to know one exact expression for the solution of the differential equation of interest: f. The point of this example is to show that this was not good enough. When it came to actually calculating values at points of interest, we noticed that this expression of the solution can be questioned. The lesson is that the supposedly superior epistemic situation of possessing

an exact explicit solution is in fact on many occasions more precarious—and it is exactly considerations of numerical nature that show this.

This example can be generalized. Suppose, again, that function f is the solution to our model. We also want the physical information encoded in this function; that is, we would like to know the values of this function at various points τ. As limited epistemic agents, we are always under certain computational limitations C, so we have to calculate $f_c(\tau)$. The problem in its most general form is that we are not immediately justified to work with this value, since other values exist—and, therefore, we need a principled way to choose among them. We know that other values exist since there are other functions mathematically equivalent to f, and, what is even more disconcerting, we can produce them at will, by simple manipulations of one of them.

The general nature of the difficulty is then clear: given a solution f, we can specify an infinite class of functions g_k such that the following conditions obtain:

1. for all k, g_k is mathematically equivalent to f;
2. it is possible that at least one of them, call it g_i, will be computationally inequivalent to f;
3. it is possible that at least one of them, call it g_i, is such that $g_{i_c}(\tau) \neq f_c(\tau)$.[13]

Thus, if we construct a complexity hierarchy based on the notion of numerical stability, it will differ in important respects from the two hierarchies mentioned earlier. We want to use a computed solution from an expression that is as computationally simple as possible while being numerically stable.

If we return to the alternative expressions of equations (8) and (10), we use the method of equation (6) to determine whether the expression is numerically stable. If we write the floating-point implementation of (8) and want to find an equivalent real function with error terms, we get

$$\begin{aligned} f_C(x) &= fl(x(\sqrt{x+1} - \sqrt{x})) = x \times fl(\sqrt{x+1} - \sqrt{x}) \times (1 + \delta_1) \\ &= x(fl(\sqrt{x+1}) - fl(\sqrt{x}))(1 + \delta_2)(1 + \delta_1) \\ &= x(\sqrt{x+1}(1 + \delta_3) - \sqrt{x}(1 + \delta_4))(1 + \delta_2)(1 + \delta_1). \end{aligned} \tag{12}$$

13. With a characterization of approximate solutions in terms of solutions of nearby problems, as found, e.g., in Fillion and Corless (2014), we could even say in addition that $g_{i_c}(\tau) \approx f_c(\tau)$.

We thus see that the worst case scenario is

$$f_C(x) = f(x)(1 + \delta)^3, \tag{13}$$

where δ is the maximum relative rounding error dictated by the size of the unit in the last place for this system of floating-point arithmetic (in this case, with 6 digits and $\alpha = 500$, this is about $\delta = 20^{-4}$). What about g? We have

$$\begin{aligned} g_C(x) &= fl\left(\frac{x}{\sqrt{x+1} + \sqrt{x}}\right) = \frac{x}{fl(\sqrt{x+1} + \sqrt{x})}(1 + \delta_1) \\ &= \frac{x}{(\sqrt{x+1}(1 + \delta_2) + \sqrt{x}(1 + \delta_3))(1 + \delta_4)}(1 + \delta_1). \end{aligned} \tag{14}$$

Here, the worst case scenario is

$$g_C(\alpha) = g(\alpha)(1 + \delta), \tag{15}$$

which is much more robust to round-off errors of maximum magnitudes δ. Thus, we can conclude that, once implemented in floating-point arithmetic, g solves a very near problem, but f does so to a lesser extent. But, as we have noted, many other candidate functions could come up against g; would it undermine our confidence that g gave us a reliable answer? No, since the error factor for g is of order $(1 + \delta)$, so that no other candidate could have an error factor of lower order.

As we see, the argument is quite general and does not depend on the specifics of a particular system of floating-point arithmetic. In fact, the values δ could be equally well thought of as physical perturbations or measurement errors, instead of round-off errors. This is why the expression g will be more information conducive than f, both in a numerical and in a physical sense. As a result, we conclude that, even if Fred and Ginger started from epistemically symmetric contexts, their results have asymmetric values. In other words, we see that the epistemic symmetry in the mathematical context does not carry over to the numerical context.

4. Conclusion. When we do not have an exact solution to a mathematical problem, we see that it is important to consider various forms of stability, to determine the similarity between modified problems, and to determine the robustness of our original reference problem to perturbations. By such forms of analysis, we establish that inexact solutions are entirely satisfactory (or not, as the case may be), given a certain modeling context. More precisely, we seek to show that the computational error engendered by a numerical method is small in comparison to the systemic modeling error that we know to be present for physical reasons. But even for exact solutions, the robustness or sensitivity to such factors has to be established. As a

result, the alleged superiority of exact solutions is mitigated, provided that one is interested in scientific modeling. This fact has important consequences for the way that the hierarchy of complexity of mathematical problems should be conceived, as it incorporates a relationship with nearby problems in terms of both computational complexity and numerical stability.

REFERENCES

Blum, L. 2004. "Computing over the Reals: Where Turing Meets Newton." *Notices of the American Mathematical Society* 51 (9): 1024–37.

Borwein, J., and R. Crandall. 2013. "Closed Forms: What They Are and Why We Care." *Notices of the American Mathematical Society* 60 (1): 50–65.

Corless, R. M., and N. Fillion. 2013. *A Graduate Introduction to Numerical Methods*. New York: Springer.

Dahlquist, G., and A. Bjorck. 1974. *Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall. Translated from the 1969 Swedish edition.

Fillion, N., and R. M. Corless. 2014. "On the Epistemological Analysis of Modeling and Computational Error in the Mathematical Sciences." *Synthese* 191:1451–67.

Humphreys, P. 2004. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. New York: Oxford University Press.

Matthews, J., and K. Fink. 1999. *Numerical Methods Using MATLAB*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall.

Trefethen, L. N. 2008. "Numerical Analysis." In *The Princeton Companion to Mathematics*, ed. T. Gowers, J. Barrow-Green, and I. Leader. Princeton, NJ: Princeton University Press.

Turing, A. 1950. "Computing Machinery and Intelligence." *Mind* 59 (236): 433–60.

Wang, Q. 1990. "The Global Solution of the *n*-Body Problem." *Celestial Mechanics and Dynamical Astronomy* 50 (1): 73–88.