

AI versus Numerical Analysis: Two Forms of Computation

Claes Johnson

KTH Royal Institute of Technology, Stockholm

`claesjohnson@gmail.com`

October 4, 2026

Abstract

A large language model computes with symbols; numerical analysis computes with numbers in digital form. Both are machine computation, both run on the same hardware, and the difference between them is not rigour but what the computed objects denote and how the result is certified. A symbolic computation is certified by derivability: each step follows, and a proof assistant can check that it does. A numerical computation is certified by an error bound that the computation itself produces. The first gives certainty about a statement; the second gives a quantity with a stated accuracy, which can be compared with a measurement.

We take as example the two recent accounts of the incompressible Euler equations. An AI system has produced a proof, formalised in Lean, that smooth compactly supported data can develop infinite velocity gradients in finite time. Adaptive finite element computation produces turbulent solutions past a body whose mean-value outputs — drag, lift, separation — are determined to a computed tolerance and agree with measurement. The first proves *existence without exhibiting an individual case*; the second *computes an individual case* and exhibits nothing general. Each has exactly what the other lacks, neither is the other's refutation, and the pair is instructive about both ways of computing.

We draw two conclusions. Symbolic computation reaches objects that possess exploitable structure — a symmetry, an ansatz, a closed form — and reaches them exactly; it does not reach objects defined by the absence of such structure, of which turbulence is the standing example, and no increase in scale will alter this. And the problem AI now faces, that of trusting a computation no person has followed, is one numerical analysis met fifty years ago and solved, by computing the error alongside the answer. We suggest that a posteriori error control is what symbolic computation presently lacks and what it should take from the older discipline.

1 Two forms of computation

A large language model and a finite element solver are both programs. Both execute on the same processors, both reduce in the end to floating-point arithmetic, and both produce in finite time an output no person could have produced by hand. The distinction between them is not that one is mathematics and the other machinery.

The distinction is in what the computation manipulates and in what it means for the output to be right.

Computation with symbols. A language model operates on tokens — words, formulas, proof steps — and its output is a text. When that text is a proof, correctness means *derivability*: every step follows from previous steps by a rule, and the question of whether it does is decidable by a proof assistant. The verdict is binary. A proof is valid or it is not, and no quantity attaches to it.

Computation with digital numbers. A numerical method operates on numbers held in finite precision over a finite mesh, and its output is a number. Correctness cannot mean derivability, since the computed object is not the exact one and never will be. It means instead a *bound*:

$$|\text{output computed} - \text{output exact}| \leq E,$$

where E is itself computed, from the residual of the computed solution weighted by the solution of an adjoint problem. The verdict is quantitative. The answer is right to within a stated amount, and the amount is part of the answer.

Where the difference lies. It is tempting to rank these, and the ranking usually proposed is that derivability is the stronger notion because it is exact. That is true of the notion and misleading about the practice. A derivation is exact about the statement it derives, and a statement can be exactly established and say nothing about the world. A bound is inexact about a quantity, and a quantity can be compared against a measurement made by somebody else. The two kinds of certainty point in different directions: one inward, to consistency; one outward, to agreement.

And the symbolic layer is numerical underneath. The two forms are not disjoint, and it is as well to say where they meet. A language model is not a symbolic engine in the sense of a term rewriter. It is a numerical one. Tokens are vectors, attention is matrix multiplication, the whole forward pass is dense linear algebra in finite precision, and the weights were obtained by gradient descent on a non-convex objective — which is to say by a numerical method for an optimisation problem, run to a tolerance nobody states. Symbolic computation, as practised by these systems, is *implemented as* numerical computation. At the bottom there is floating-point arithmetic and nothing else, and it is the same arithmetic a finite element solver performs.

So the distinction of this section is one of level and not of kind. What is manipulated at the top is symbols; what happens at the bottom is numbers in digital form. Numerical analysis is in that sense logically prior to the symbolic layer rather than an alternative to it.

Two things follow, and the second is the point of Section 7. The arithmetic being the same, its difficulties are the same: conditioning, precision, and a dependence on the order of summation which makes a large model’s output irreproducible in the way an unstabilised computation is irreproducible — run it again and the proof may differ. And the numerical layer of these systems carries, at present, no error control whatever. The finite element solver computes a bound on how wrong it is; the transformer computes no such thing, at either level. That is not a limitation of numerical computation. It is a facility of numerical computation which has not been built.

2 The example

The incompressible Euler equations have lately received both treatments, which makes them an unusually clean case.

The symbolic account. An AI system has produced a proof that there exists smooth, compactly supported, divergence-free initial data on $\mathbb{R}^3$ whose smooth Euler solution has finite maximal lifespan: the velocity gradient becomes unbounded and the Beale–Kato–Majda vorticity integral diverges [2]. The proof is fifty-seven pages and is formalised in Lean, so that its correctness is machine-checked. Comparable results were being prepared independently and by hand [3].

What the result delivers is an existence statement. It does not say which data blow up; there is no measure on the set and no claim of genericity. It does not say what the flow looks like. It yields no drag, no lift, no length, no time that could be set beside an experiment. The datum itself is produced as the limit of an iteration and cannot be written down. The result is nonetheless a real advance: a question open for decades is settled, and the settling is exact.

The numerical account. Adaptive finite element computation of the same equations past a body produces turbulent solutions [4, 5, 6]. The velocity stays bounded, the gradients do not, the solution dissipates energy at a rate independent of the viscosity, and the limit of the computed sequence is a weak solution of Euler by the residual control the method carries. Its mean-value outputs are determined: drag to within a few per cent of measurement, the separation line where experiment puts it, the lift curve and the stall.

What this delivers is not a solution. For weak solutions of Euler uniqueness is false, so there is no unique solution to deliver. What is delivered is the quantities that are determined, each with an error bound the computation produces, each open to check by somebody with no interest in the outcome.

The shape of the comparison. Put at its plainest, the difference is between a proof of *existence without an individual case* and the *direct computation of an individual case*.

The symbolic account establishes that a datum with the stated property exists. It does not hand one over. The datum is the limit of an iteration over stages, each adding an oscillation to the one before, and no finite description of it is produced or intended — nothing can be plotted, measured or shown to anybody. What exists is the existence.

The numerical account computes one case: this body, this free stream, this flow, with this drag. It establishes nothing about all data or about any other case. What it produces can be looked at. The wake has a shape, the separation line a position, the drag a value with a bound on its error, and each can be set beside a measurement.

So the two are complementary in the strict sense: each has exactly what the other lacks. Generality without instance, instance without generality. The first is mathematically substantial and physically empty, the second physically substantial and mathematically the weaker. The companion paper [1] argues the fluid mechanics of this and concludes that the singular solutions, whatever their mathematical interest, are not flows. Here we are concerned with what the pair says about the two ways of computing.

3 Both are models of a physical process

Before the differences, the deeper similarity, which is easily missed and which makes the comparison worth making at all. Neither account is a deduction about an abstract object. Both are *sequential processes that march the flow forward*, and both take their instructions from the same equation.

The digital process. General Galerkin steps in time. At each step the forces are computed from the equation — the convective term, the pressure gradient — and velocity and pressure are updated accordingly. What the computer holds is a digital representation of the flow field, and what it does to it at each step is what the equation says is done to a fluid. Running the computation is following the physical process in a representation that happens to be made of numbers, and this is why one can watch it: the separation forms, the vortex tubes appear at the separation line, they stretch and break down, and the wake develops. How and why the flow becomes what it becomes is displayed, step by step, because the steps are the mechanism.

The symbolic process. The construction of [2] is also sequential, and also forward. It produces exact smooth Euler solutions U_j on nested time intervals: at each stage a localised oscillation is added to the parent flow, the pressure Hessian amplifies it along particle trajectories, and the stage times t_j advance toward a finite limit. What is held is a symbolic representation, and what is done to it at each stage is again what the equation licenses. It too displays a mechanism — amplification of an oscillation by the deformation it rides on — and that mechanism is visible in a way no single computation makes it.

The two processes are not identical in their sequencing, and the difference should be stated. The digital process steps one flow forward in time. The symbolic process steps through a *sequence of flows* whose initial data converge while the amplification time advances, so it is sequential in the datum as well as in time. The parallel is a parallel and not an identity.

Digital and specific, symbolic and general. With that said, the pair is better understood as two representations of one physical process than as two activities of different kinds. One is digital and specific: a particular flow past a particular body, every quantity a number, every number comparable with a measurement, and nothing established about any other case. The other is symbolic and general: no numbers, nothing to measure, and a statement that holds across a family.

Each therefore makes visible what the other cannot. The digital process shows what this flow does and cannot say what all flows do. The symbolic process exhibits a mechanism valid for a family and cannot show what any member of it looks like. They are models of physics at two levels of representation, and the rest of this paper concerns what follows from the difference in level — not, as it is usually put, from a difference between mathematics and computation.

4 What each form can reach

The division of labour is not a division between rigour and approximation. It follows from what each form of computation requires of its object.

Symbolic computation requires exploitable structure. A proof proceeds by rewriting, and rewriting needs something to grip. Every construction of a singular solution buys its tractability with a structure: self-similarity, which collapses a partial differential equation to a profile and leaves a fixed point; or axisymmetry, which removes a variable; or an iterative scheme whose stages are similar to one another. This is not a criticism. It is what makes the object reachable at all, and the skill lies in finding the structure that a given problem will support.

Numerical computation requires none. A mesh does not care whether the solution has a symmetry. The computation proceeds identically whether the object is a self-similar profile or a turbulent wake, and the cost is set by the resolution rather than by the structure.

Hence the complementarity, and its sharp edge. Objects *with* structure are reachable both ways, and symbolically with greater precision. Objects *without* structure are reachable only numerically. Turbulence is the standing example of the second kind — it is *defined* by carrying structure at every scale with no organising principle to exploit, so there is no ansatz to be found, and the absence is the phenomenon rather than a gap in anybody’s technique.

This bears directly on what AI changes. A language model is a search over symbol sequences, and what it does superlatively is find structures that a human might have found given longer. It therefore amplifies the half of mathematics that was already good at special objects, and amplifies it a great deal. It does not convert the other half. No scale of model makes a turbulent flow analytically constructible, because the obstruction is in the object and not in the searcher.

5 Checking the result, or checking the steps

Neither form of computation can be verified step by step by a person. A finite element computation performs billions of floating-point operations; a fifty-seven page formal development has more inferences than anyone will read. On that both are in the same position, and it is usually where the discussion stops. It should not, because the two escape the position in entirely different ways, and only one of them escapes.

A numerical result can be checked without checking the computation. Given any candidate velocity field, substitute it into the Euler equations and evaluate what is left over:

$$R(u) = \dot{u} + (u \cdot \nabla)u + \nabla p, \quad \nabla \cdot u.$$

If the residual is small in the appropriate weighted sense, the field nearly solves the equations; and weighted by the solution of an adjoint problem it bounds the error in any output of interest. The decisive feature of this test is that it is *independent of how the field was obtained*. It does not inspect the mesh, the solver, the arithmetic or the order of operations. The candidate could have come from a different program, a different method, or a guess, and the test would be the same test. Consequently nobody needs to verify the individual computational steps, and nobody does: an error at the three-millionth operation is not caught by examining that

operation, it is caught because the residual comes out large. The equation itself performs the audit, once, on the answer.

A symbolic result cannot. There is no corresponding operation on a proof. A proof does not have a residual, because what it produces is a statement rather than an object, and a statement cannot be substituted into anything. Its correctness simply *is* the correctness of its steps, so there is no alternative to checking them — which is what a proof assistant does, mechanically and reliably, and at a scale no person matches.

This deserves to be put more carefully, because symbolic computation is not in fact without a residual test. It has one, and it is the strongest of all. Given a candidate in *closed form* — a formula for u and p — one inserts it into the equations and simplifies, and the check is exact: either the residual reduces identically to zero or it does not. No tolerance, no mesh, no adjoint. Symbolic manipulation is very good at this, and a computer algebra system performs it unattended. Verifying a solution in symbolic form is thereby both easier and sharper than verifying one in digital form.

What that check requires is a specific symbolic object to insert. Here there is none. The datum of the construction is the limit of an iteration over stages and has no closed form; there is no expression to substitute, nothing to simplify, and no identity to reduce to zero. Nor is the numerical route open, since no finite stage of the iteration exhibits the property — the blowup belongs to the limit. So *both* residual tests fail, the symbolic and the digital, and they fail for the same reason and not for two: there is no individual object, in either form, to put into the equations.

The two observations of this paper are therefore one observation. The construction proves existence without exhibiting an individual case, and an individual case is exactly what any check of the result, symbolic or numerical, would have to be applied to. *Because there is no instance, there is nothing to insert; because there is nothing to insert, only the steps remain.*

What Lean settles and what it leaves. A proof assistant establishes that the steps follow from the stated definitions. It does not establish that the stated definitions say what was intended, and that question cannot be mechanised — somebody has to read the formal statement and agree that it is the theorem. For a development of this length, with its own hierarchy of definitions, that reading is itself substantial work, and it is the part no machine does. So the verification burden is reduced and relocated rather than removed, and what remains of it falls on a human mathematician who must in the end understand something.

The asymmetry is therefore not that one discipline is careful and the other careless. It is that the numerical computation is answerable to an external object — the equation, and beyond it the measurement — which audits the result wholesale, while the symbolic computation is answerable only to its own steps, which must be audited one at a time.

Stated as two kinds of certainty, and without rancour: *a formal proof is internally certain.* The assistant guarantees that the steps follow. It guarantees nothing about whether the statement proved is about anything, and here the statement is about a datum that cannot be prepared and a flow no instrument will see. The certainty is real and it is certainty of consistency. *A numerical computation is externally checkable.* Its output is a number with a tolerance, and the number can be set beside a measurement somebody else made at a

facility with no stake in the outcome. The High Lift Prediction Workshop and the high-order workshops exist for precisely this: predictions are submitted before the measurements are released. That is an accountability a proof does not have and does not need, and a computation does need.

6 What is a mathematical proof?

The question cannot be avoided here, and it is the episode rather than this paper that raises it. Two notions have long been carried together and have now come apart.

A proof as certificate. A finite object which establishes a statement — and, the part usually left unsaid, one that can be *checked more cheaply than it was found*. That asymmetry is what makes proof useful: finding is hard, checking is easier, and so a result can be transmitted without being rediscovered by each person who receives it.

A proof as explanation. An argument conveying *why* a statement holds, which leaves the reader able to see it, to use the idea elsewhere, and to recognise the same mechanism in another problem. This is what is meant in calling a proof good rather than merely correct.

In human mathematics the two coincided closely enough that they were not distinguished. A proof short enough to check was generally short enough to understand. De Millo, Lipton and Perlis argued in 1979 that the coincidence was not incidental [7]: theorems are accepted through a social process of being read, reused and internalised, and formal verification would disappoint because it supplies the certificate while dispensing with the process. A fifty-seven page machine-checked development is that argument made actual.

And what is an explanation? The notion needs a test, or it will carry whatever weight is wanted of it. We propose the plainest available: *an explanation tells you what the result depends on*. It lets you say which hypotheses are load-bearing, what a perturbation would do, and why the outcome is this one and not another. On that test two things follow, and neither is what the symmetry of the previous paragraphs suggests.

A specific solution can be inspected, and inspection has produced symbolic understanding. What G2 exhibits is a particular flow, held in full: every field at every step, open to being plotted, sectioned, differenced and stared at. That is the raw material of explanation, and in this case the explanation was duly extracted. The resolution of d’Alembert’s paradox [5] and the account of flight [6] were both obtained by *looking at* computed turbulent solutions — seeing where the instability acted, what it generated, and which part of the surface carried the drag — and both are stated symbolically, as arguments about stability and about vorticity, not as tables of numbers. The computation did not replace the analysis. It supplied the object the analysis was about. A specific solution open to inspection is thus a route into symbolic understanding, and a travelled one.

The adjoint solution gives a second and quantitative form of the same thing. It is a sensitivity: it says which parts of the flow the output depends on and how strongly, which is counterfactual dependence with a number attached. Whatever else is obscure about a turbulent computation, what its drag depends on is not.

Principles of a construction are not a specific case. The symbolic account does offer something real here, and it should be credited: it gives the principles by which such a flow can be built — an oscillation amplified along trajectories by the pressure Hessian, scales chosen and the construction iterated. That is a mechanism and it is intelligible. What it does not give is any instance of it to inspect. There is no flow to plot, no field to section, nothing to look at and be surprised by. So the understanding available stops where the principles stop: one learns how such a solution is assembled, and nothing about what the equations do to a fluid, because no fluid is exhibited.

The asymmetry on explanation is therefore the one on checking, seen again from the other side. A specific case can be inspected, and inspection yields understanding which may then be put in symbolic form. Principles without a case can be read and followed, and there the matter rests.

6.1 How far a result descends

The distinction usually reached for here is constructive against non-constructive, and it is the wrong one: it is an axis on which the construction in question does well. Its datum is the limit of an explicit iteration with a convergent modulus, so it can be approximated to any accuracy named, and calling the result non-constructive would be false.

What is true is finer, and it is not binary. Results descend to differing depths, and the useful grading is how far each gets from *there is one to here it is, measure it*.

1. **Exists, no witness.** The disproof of the Mertens conjecture [9] establishes that a counterexample exists and exhibits none; so far as we know none has been exhibited since. Entirely respectable, and a great deal of good mathematics lives here.
2. **Witness as a limit, the property belonging to the limit alone.** The Euler blowup. The datum is approximable, and *no finite stage of the approximation blows up*, the property being the limit's.
3. **Witness computable to usable accuracy, the approximation possessing the property.** Most of numerical mathematics, including the turbulent solutions here.
4. **Witness exhibited explicitly.** Weierstrass's nowhere-differentiable function; Enflo's Banach space without the approximation property [8].
5. **Witness self-verifying.** Lander and Parkin's $27^5 + 84^5 + 110^5 + 133^5 = 144^5$, two lines long [10], refuting a conjecture of Euler's. The object checks the claim, so the computer search that found it needs no scrutiny whatever.
6. **Witness physically realisable and measurable.** A cylinder in a stream, a drag coefficient, a wind tunnel.

Every argument of this paper is a statement about rungs. The residual test of Section 5 requires rung 3 or above — something must exist to insert into the equations. Inspection, and the understanding got by it, requires rung 4: an object to look at. The wholesale audit which makes step-checking unnecessary is rung 5 in its purest form. Physics is rung 6.

The Euler blowup sits at rung 2. That is respectable mathematics, and it is the lowest rung from which nothing can be inserted, nothing plotted and nothing measured. The complaint is not that the result is non-constructive, which it is not, but that it descends one rung short of where any check on a result, as opposed to a check on its derivation, becomes possible.

Length is not the difficulty, and the record shows it. It is worth recording what proofs have cost before, since fifty-seven pages is not remarkable.

Lander–Parkin, Euler’s sum-of-powers conjecture	1966	≈ 2 lines
Weierstrass, nowhere-differentiable function	1872	≈ 2 pp
Enflo, approximation problem	1973	≈ 9 pp
Odlyzko–te Riele, Mertens conjecture	1985	≈ 23 pp
Isett, Onsager’s conjecture	2018	≈ 93 pp
Enflo, invariant subspace problem	1987	≈ 100 pp
Feit–Thompson, odd order theorem	1963	255 pp
OpenAI, Euler blowup	2026	57 pp
Appel–Haken, four colour theorem	1976	≈ 400 pp + machine
Hales, Kepler conjecture	1998	≈ 300 pp + machine
Classification of finite simple groups	1955–2004	$\approx 10^4$ pp

Mathematics has lived with unsurveyable proofs since 1976 and with a theorem nobody has read in full for longer. Length is therefore not the complaint, and the four colour theorem is not usually held to be no theorem. Feit–Thompson runs to an entire journal issue and was absorbed, because there are ideas in it which compress. Enflo’s counterexample is nine pages and was hard for years, because it is a construction with no shortcut — and it is at rung 4, so one can at least hold the object. What is new in the present case is neither length nor machine assistance. It is a proof with no compressing idea available to a reader and no instance available to anyone.

Where this leaves the two computations. The formal proof is a certificate and not an explanation. Its checking asymmetry survives for machines — the assistant verifies faster than the system searched — and is lost for people. The numerical computation with an a posteriori bound is likewise a certificate and not an explanation, and its asymmetry is the more extreme of the two: hours to compute, a single residual evaluation to check.

They are in the same category. Both certify; neither explains.

That is awkward for the conventional ranking, and the awkwardness runs both ways. *If proof means certificate*, then an a posteriori error estimate is a proof — it establishes $|\text{computed} - \text{exact}| \leq E$, which is a theorem about the exact solution, obtained by computation — and the distinction between what is proved and what is merely computed dissolves. *If proof means explanation*, then the fifty-seven page development is not a proof either, and what has been produced is a certificate that something is so, with nobody the wiser as to why.

We do not claim to settle which notion is the right one, and we doubt it is settled by argument. We observe only that the two are at present applied selectively: the certificate standard to the machine proof, which is accepted, and the explanation standard to the computation, which is not. Whichever notion is adopted, it should be adopted for both.

7 What symbolic computation might take from the older discipline

That a result may have to be believed on the strength of a procedure rather than of an understanding is not a new condition. It is the one numerical analysis has worked in since the 1950s, and the discipline's response was not to argue about whether machine results count. It was to make them accountable: to compute, alongside the answer, a bound on how wrong the answer may be, and to refine where the bound says the error lives. That is the content of a posteriori error estimation and adaptivity, and it is why a computed drag coefficient is a scientific statement rather than an opinion produced by a program.

The central practical problem now facing AI in mathematics and elsewhere is this: a system produces an output that is too long to check and too important to trust blindly. That is, exactly, the problem numerical analysis faced when machines first outran hand calculation, and it has a developed answer.

The answer has three parts, and all three transfer in principle.

Compute the error, not merely the answer. A finite element computation returns a solution and a bound on the error in each output of interest. The bound comes from the residual — what the computed solution fails to satisfy — weighted by the solution of an adjoint problem which measures how much that failure matters for the output in question. The structure is general: a residual, a sensitivity, and their product. A language model has a residual in the same sense — the respects in which its output fails to satisfy the conditions it should — and nothing at present weights it.

Let the error direct the effort. Adaptivity refines where the bound says the error is, not uniformly. The analogue for a symbolic system is to concentrate verification where the proof is weakest rather than uniformly over its length, which requires knowing where that is, which requires the weighting above.

Report what is *not* determined. An honest a posteriori estimate sometimes returns a large bound, and that is information: the computation does not determine this output, and no refinement of this kind will make it do so. A system that cannot return that verdict about its own output is not yet accountable, whatever its success rate.

That the machinery is numerical underneath makes this less of an analogy than it may appear. The quantities a posteriori estimation works with — a residual, a sensitivity, their product — are numerical quantities, and a language model is a numerical object. The tools would not have to be carried across a gap; they would have to be applied one level down from where the symbols are.

We put this forward as the constructive half of the comparison. The debate about whether machine proof is mathematics will be settled the way the corresponding debate about machine computation was settled, which is not by argument but by the development of error control. Numerical analysis has fifty years of that, and it was built for precisely the situation symbolic computation is now in.

8 Understanding, and two ways to it

Symbolic computation and numerical computation are both machine computation, and at the bottom the first is implemented in the second. They differ in the level of representation — symbols against numbers in digital form — and in how a result is certified: by derivability, checkable step by step, against an error bound the computation itself produces. Both are sequential processes which march a flow forward on instructions from the same equation, and both are therefore models of a physical process rather than activities of different kinds.

What they deliver differs as the level does. A proof of blowup establishes existence without exhibiting an individual case; a computation exhibits an individual case and establishes nothing general. Each has exactly what the other lacks.

Understanding is the end, and there are two ways to it. If the goal of all this is understanding rather than certification, then the two accounts are better seen as two modes of explanation than as rivals. Explanation by the *particular case*: here is the flow, watch it separate, see the vortex tubes form at the separation line and stretch, see which part of the surface carries the drag. Explanation by the *governing principle*: here is the mechanism by which such a solution can be built, and it holds across a family. Both are genuine and neither reduces to the other.

What the particular case offers that a principle cannot is *surprise* — something not put in and not expected, which is where new principles come from. Nobody derived the three-dimensional rotational separation of [5] before seeing it; it was found by inspecting computed solutions, and only then stated as an argument. What the principle offers that a case cannot is *transfer*: what to expect where nothing has been computed.

Science advances by cycling between them — compute or observe, inspect, formulate, predict, test — and on that reading the two forms of computation occupy two phases of one process rather than two sides of a dispute.

But the cycle is not symmetric. Only the phase that produces instances touches the world. A principle which never descends to an individual case can be checked against nothing but its own derivation: there is no object to insert into the equations, nothing to plot, nothing to measure, and nothing a fluid could contradict. That is the asymmetry this paper has been documenting from several directions, and they are one asymmetry. No instance, no residual; no instance, no inspection; no instance, no surprise.

So the proposal of Section 7 is not a complaint but the obvious next move. Symbolic computation has arrived at the condition numerical analysis has worked in for seventy years — results too long to follow, believed on the strength of a procedure — and numerical analysis answered it by computing the error alongside the answer and refining where it lives. The machinery is numerical in both cases, so the answer transfers. What it would buy is not rigour, which is not in short supply, but accountability: a system able to report what it has *not* determined, which is the beginning of being answerable to something outside itself.

References

- [1] J. Hoffman and C. Johnson, *Incompressible Euler flow: turbulence vs blow-up*, companion paper, 2026.
- [2] OpenAI, *Finite time blowup for the Euler equation*, preprint, 2026, 57 pp. <https://cdn.openai.com/pdf/315b36cd-ec98-4023-8342-93345194ece1/euler.pdf>
- [3] T. Buckmaster and L. Alpöge, *Statement*, 2026. <https://cims.nyu.edu/~tristanb/statement.pdf>
- [4] J. Hoffman and C. Johnson, *Computational Turbulent Incompressible Flow*, Applied Mathematics: Body and Soul, vol. 4, Springer, 2007.
- [5] J. Hoffman and C. Johnson, *Resolution of d'Alembert's paradox*, J. Math. Fluid Mech. **12** (2010) 321–334.
- [6] J. Hoffman, J. Jansson and C. Johnson, *New theory of flight*, J. Math. Fluid Mech. **18** (2016) 219–241.
- [7] R. A. De Millo, R. J. Lipton and A. J. Perlis, *Social processes and proofs of theorems and programs*, Comm. ACM **22** (1979) 271–280.
- [8] P. Enflo, *A counterexample to the approximation problem in Banach spaces*, Acta Math. **130** (1973) 309–317.
- [9] A. M. Odlyzko and H. J. J. te Riele, *Disproof of the Mertens conjecture*, J. reine angew. Math. **357** (1985) 138–160.
- [10] L. J. Lander and T. R. Parkin, *Counterexample to Euler's conjecture on sums of like powers*, Bull. Amer. Math. Soc. **72** (1966) 1079. The paper is two sentences long; the search ran on a CDC 6600.
- [11] K. Eriksson, D. Estep, P. Hansbo and C. Johnson, *Computational Differential Equations*, Cambridge University Press, 1996.
- [12] A. Logg, *Automation of computational mathematical modeling*, Ph.D. thesis, Chalmers University of Technology, 2004.
- [13] A. Logg, K.-A. Mardal and G. N. Wells (eds.), *Automated Solution of Differential Equations by the Finite Element Method*, Lecture Notes in Computational Science and Engineering 84, Springer, 2012.