Predictions for Scientific Computing Fifty Years From Now

Lloyd N. Trefethen
Oxford University Computing Laboratory
LNT@comlab.ox.ac.uk

This essay is adapted from a talk given June 17, 1998 at the conference "Numerical Analysis and Computers—50 Years of Progress" held at the University of Manchester in commemoration of the 50th anniversary of the Mark 1 computer.

Fifty years is a long, long time in any technological field. In our own field of scientific computing or numerical analysis, think back to 1950. Around the world, numerical problems in 1950 were solved with slide rules and on paper, or with mechanical calculators that had little in common with today's computers. Some of the algorithms we use today were in existence then, but on the whole, the last fifty years have changed numerical computing beyond recognition. The next fifty will do it again.

My remarks consist of twelve predictions. I did not aim for these to orbit around a unifying theme, but that is nevertheless what happened.

1. *We may not be here.*

In the 20th century, everything technological seems to be changing exponentially. This raises a problem. Exponentials do not go on for ever; something happens to them. Now in my opinion, many of the exponentials we are sitting on have not yet started to level off. Here at the beginning of the third millennium, biology is just beginning its great explosion, and although electronics got a head start of a few decades, it is hardly slowing down yet.

The presence of exponentials all around us overshadows any attempt to predict the future. I feel I must dwell for a moment on one of the shadows, one that has nothing specifically to do with computing. In my opinion, our position on an exponential trajectory is evidence that technological civilisations do not last very long. I do not claim that our civilisation must end within fifty years, or five hundred, but I do believe there is reason to doubt it can survive for, say, ten thousand years.

My reasoning has nothing to do with any particular cataclysm that may befall us, such as environmental catastrophe or exhaustion of resources or asteroid impact or biological or nuclear war. The argument is more abstract, and it goes like this. The industrial explosion on earth began just two or three hundred years ago. Now if technological civilisations can last tens of thousands of years, how do you explain the extraordinary coincidence that you were born in the first few generations of this one? — in the very first century of radio, television, light bulbs, telephones, phonographs, lasers, refrigerators, automobiles, airplanes, spacecraft, computers, nuclear power, nuclear weapons, plastics, antibiotics, and genetic engineering?

I believe that the explanation of our special position in history may be that it is not so special after all, because history tends not to last very long. This argument has been called the *Copernican Principle* by J. R. Gott of Princeton University.

There is a second line of evidence, sometimes known as *Fermi's paradox,* that also suggests that technological civilisations are short-lived. The human race is not an outpost of a galactic society; it is a domestic product. How can we explain this if technological civilisations last tens of thousands of years? An ages-old technological civilisation will expand across its galaxy, simply because it can. (Don't ask why, for expanding is what life does. If one species doesn't, another will replace it.) Yet in 100,000 years of expanding at one hundredth the speed of light, a civilisation can spread one thousand light years, a distance encompassing millions of stars. Is it plausible that technological civilisations are so rare as to arise on only one star among millions?

I believe that the explanation of the emptiness out there may be that technological civilisations perish before they start to spread across their galaxy—or that they start spreading, then perish in a cataclysm so great as to take the galaxy with them.

Suddenly the problem of predicting fifty years of scientific computing begins to look easy! Let's get down to it.

2. We'll talk to computers more often than type to them, and they'll respond with pictures more often than numbers.

A big change in the last twenty years has been the arrival of graphical interfaces. When I was a graduate student at Stanford around 1980, we played with some Alto machines donated by Xerox, early workstations featuring windows, icons, mice and pointers, but I thought these were party tricks, too gimmicky to catch on. Today the descendants of the Altos have driven other machines to extinction. It takes no special insight to predict that soon, an equally great change will occur as we take to interacting with computers by speech. It has been a long time coming, but this transformation is now around the corner.

It is good fun to imagine what computer graphics will be like in fifty years. I hardly dare, except to note that three-dimensional virtual reality will be as ordinary as Velcro.

Curiously, though the development of speech and graphics will make our numerical work ever more human in feel, less obviously numerical, the underlying computations will continue to be based on numbers represented digitally to many digits of precision. The digital idea is what makes everything possible, and it is not going to go away. This is one sense in which the scientists and engineers of the future will be further removed from the details of computing than we are, just as we are further removed than were our parents.

3. Numerical computing will be adaptive, iterative, exploratory, intelligent—and the computational power will be beyond your wildest dreams.

Adaptive numerical computing is one of the glories of the computer age. Gauss quadrature was invented two centuries ago, but adaptive quadrature didn't arrive until the 1960s. Adaptive ODE solvers came soon after, and turned the solution of most ordinary differential equations into the use of a black box. Partial differential equations are not yet boxed in black, but the trend is in that direction. As time goes by, adaptivity managed by the computer's intelligence becomes more and more widespread. Computers are not as wise as people, but they can explore a forest of possibilities faster than we can. In fifty years, this is how most numerical problems will be solved. We will tell the machine what we want, and the machine, an intelligent control system sitting atop an encyclopaedia of numerical methods, will juggle computational options at incomprehensible speed until it has solved the problem to the accuracy required. Then it will give us the answer; and if we insist, it may even tell us something of how it got there.

The power unleashed by this kind of computing will be vast. Large parts of physical reality will be simulated in real time before our eyes, with effects so far beyond what the men of 1950 could envision that the word "computation" may begin to seem old-fashioned and drop out of use.

When computations are all intelligent, when everything is embedded in a control loop, the mathematical landscape will change. One distinction that means a great deal to us today is that, broadly speaking, linear problems can be solved in one pass, but nonlinear ones require iteration. In fifty years, when everything is embedded in an iterative loop anyway, this difference will have diminished. For the same reason, today's big distinction between forward and inverse problems will have faded too.

My next prediction is a corollary.

4. Determinism in numerical computing will be gone.

Recently our family rented a car for a holiday. One evening we wanted to look at the stars, which meant turning off the dome light. We couldn't figure out how to do it! A decade ago, closing the doors and flipping a switch would have sufficed, but nowadays, cars are more intelligent. In some, the light stays on for a fixed period after you close the doors, and in ours, the situation was even more complicated. There was an interlock with the engine, plus some additional intelligence that we never got to the bottom of. Eventually we got the light off, but we were not quite sure how we had done it, or if we could do it the same way again.

Have you noticed how many of our machines behave this way? Photocopiers used to be deterministic, but nowadays they have complicated arrays of internal states. The first copy may come out in landscape orientation, but the second in portrait, if the machine decides in-between that it ought to change modes. Typewriters used to be predictable too: you knew what would happen when you pressed a key. Nowadays, in Word or LaTeX, changing one character of input may alter the whole document in startling ways. Why, at motorway rest stops, even toilets are intelligent devices now whose states of mind we don't fully understand, and when you're finished with the toilet, you have two further negotiations to undertake with the intelligent sink and the intelligent hand drier!

What's true of toilets will be true of numerical computations. In fifty years, though the answers you get will be accurate without fail to the prescribed precision, you will not expect to duplicate them exactly if you solve the problem a second time. I don't see how this loss of determinism can be stopped. Of course, from a technical point of view, it would be easy to make our machines deterministic by simply leaving out all that intelligence. However, we will not do this, for intelligence is too powerful.

In the last fifty years, the great message communicated to scientists and engineers was that it is unreasonable to ask for exactness in numerical computation. In the next fifty, they will learn not to ask for repeatability, either.

5. The importance of floating point arithmetic will be undiminished.

So much will change in fifty years that it is refreshing to to predict some continuity. One thing that I believe will last is floating point arithmetic. Of course, the details will change, and in particular, word lengths will continue their progression from 16 to 32 to 64 to 128 bits and beyond, as sequences of computations become longer and require more accuracy to contain accumulation of errors. Conceivably we might even switch to hardware based on a logarithmic representation of numbers. But I believe the two defining features of floating point arithmetic will persist: relative rather than absolute magnitudes, and rounding of all intermediate operations.

Outside the numerical analysis community, some people feel that floating point arithmetic is an anachronism, a 1950s kludge that is destined to be cast aside as machines become more sophisticated. Computers may have been born as number crunchers, the feeling goes, but now that they are fast enough to do arbitrary symbolic manipulations, we must move to a higher plane. In truth, no amount of computer power will change the fact that most numerical problems cannot be solved symbolically. You have to make approximations, and floating point arithmetic is the best general-purpose approximation idea ever devised. It will persist, but get hidden deeper in the machine.

6. Linear systems of equations will be solved in $O(N^{2+\epsilon})$ flops.

Matrix computations as performed on machines around the world typically require $O(N^3)$ floating point operations—"flops"—where N is the dimension of the problem. This statement applies exactly for computing inverses, determinants, and solutions of systems of equations, and it applies approximately for eigenvalues and singular values. But all of these problems involve only $O(N^2)$ inputs, and as machines get faster, it is increasingly aggravating that $O(N^3)$ operations should be needed to solve them.

Strassen showed in 1968 that the $O(N^3)$ barrier could be breached. He devised a recursive algorithm whose running time was $O(N^{\log_2 7})$, approximately $O(N^{2.81})$, and subsequent improvements by Coppersmith, Winograd and others have brought the exponent down to 2.376. However, the algorithms in question involve constants so large that they are impractical, and they have had little effect on scientific computing. As a result, the problem of speeding up matrix computations is viewed by many numerical analysts as a theoretical distraction. This is a strange attitude to take to the most conspicuous unsolved problem in our field! Of course, it may be that there is some reason why no practical algorithm can ever be found, but we certainly do not know that today. A "fast matrix inverse" may be possible, perhaps one with complexity $O(N^2 \log N)$ or

$O(N^2 \log^2 N)$, and discovering it would change everything.

In 1985 I made a bet with Peter Alfeld of the University of Utah that a matrix algorithm with complexity $0(N^{2+\epsilon})$ for any $\epsilon > 0$ would be found within ten years. None was, and I gave Alfeld a check for \$100. We renewed our bet, however, to 2005, and in that year I will renew it again if necessary. One morning, with luck, the headlines will appear. I think fifty years should be long enough.

7. Multipole methods and their descendants will be ubiquitous.

The conjugate gradient and Lanczos algorithms were invented around 1950, and their story is a curious one. Nowadays we have no doubt as to what these methods are good for: they are matrix iterations, which for certain structured matrices bring those $O(N^3)$ operation counts down to $O(N^2)$ or even better. Though there are constants hidden in the "O", these methods are often much faster than Gaussian elimination and its relatives when N is large.

What is curious is that Hestenes, Stiefel, Lanczos and the rest didn't see this coming. In the 1950s, N was too small for conjugate gradients and Lanczos yet to be competitive, but all the mathematical pieces were in place. These men knew something of the convergence properties of their iterations, enough to have been able to predict that eventually, as machines grew faster, they must beat the competition. Yet they seem not to have made this prediction. A numerical analyst writing an essay like this one in 1960 might not have mentioned conjugate gradients at all.

It is with this history in mind that I mention multipole methods, by which I mean methods related to the recent algorithms of Rokhlin and Greengard for N-body problems and integral equations. Times have changed, and we are all asymptotickers. When multipole methods were being invented in the 1980s, they were competitive in 2D but not 3D. Yet Rokhlin and Greengard saw immediately that these techniques reduced operation counts from $O(N^2)$ to $O(N)$, give or take a logarithmic factor, so how could they not win in the long run? And so they will.

The success of multipole methods will exemplify a general trend. As time goes by, large-scale numerical computations rely more on approximate algorithms, even for problems that might in principle be solved exactly in a finite number of steps. Approximate algorithms are more robust than exact ones, and they are also often faster.

8. Breakthroughs will have occurred in matrix preconditioners, spectral methods, and time stepping for partial differential equations.

It is hard not to be optimistic about merely technical hurdles. The business of matrix preconditioners is vitally important, but it is a jungle these days—surely improvements are in store! Spectral methods for PDEs are in a similar state—remarkably powerful, but varying awkwardly from one application to the next. Order is needed here, and it will come. As for time-stepping, this is the old problems of stiffness, reasonably well in hand for ODEs but still unsolved in a general way for PDEs. To this day, the CFL restriction constrains our computations all across the range of science and engineering. To get around this constraint, time steps are taken smaller than we would wish, huge matrix problems are solved at great cost, and physically important terms are thrown away just because they are too hard to implement. The CFL condition will not disappear, but new weapons will be devised to help us in the day-to-day struggle against it.

9. The dream of seamless interoperability will have been achieved.

Users and onlookers complain year after year, why is so much human intervention needed to get from the whiteboard to the solution? Why does one computer program have to be written for the grid generator, another for the discretisation, and another for the linear algebra, requiring interfaces all along the way with repeated opportunities for human error? Why are symbolic and numerical calculations separate? Why can't our ideas and tools blend together into a seamless interoperable system? Well, of course, they can, and getting there is merely an engineering problem. Fifty years from now, the grids and the solvers will have been coupled—and humans will more and more rarely catch sight of actual numbers in the course of doing science.

10. The problem of massively parallel computing will have been blown open by ideas related to the human brain.

The information revolution is well underway, but the revolution in understanding the human brain has not arrived yet. Some key idea is missing.

Another fact of scientific life is that the problem of massively parallel computing is stalled. For decades it has seemed plain that eventually, serial computers must run up against the constraints of the speed of light and the size of atoms, at which point further increases in power must come about through parallelism. Yet parallel computing nowadays is a clumsy business, bogged down in communication problems, nowhere near as advanced as everyone expected a decade ago.

I believe that the dream of parallel computing will be fulfilled. And it is hard to avoid the thought that if parallel computing and the human brain are both on the agenda, the two revolutions in store will somehow be linked. Brain researchers will make discoveries that transform our methods of parallel computing; or computer scientists will make discoveries that unlock the secrets of the brain; or, just as likely, the two fields will change in tandem, perhaps during an astonishing ten years of upheaval. The upheaval could begin tomorrow, or it might take another generation, but it will come before 2050.

Meanwhile, another revolution in biology is already happening: the working out of DNA/RNA genomes and their implications. Every organism from virus to man is specified by a program written in the alphabet of the nucleotides. Since Watson and Crick, we have known this must be true, and in 1995, the first genome of a free-standing organism was sequenced. Since then, dozens more have followed, with the human genome itself now nearly complete, and everything in biology, from development to drug design, is being reinvented as we watch. If I give you the sequence `KPSGCGEQNMINFYPNVL` in the standard code for the amino acids, this is enough for you to determine in a few seconds that I am speaking of an α-macroglobulin proteinase inhibitor of *Octopus vulgaris*, and to locate related enzymes in ten other species. Just point your browser to `http://www.ncbi.nlm.nih.gov` and run *blastp*.

I believe that this drama has implications for computing.

11. Our methods of programming will have been blown open by ideas related to genomes and natural selection.

Genetic programs and computer programs are strangely analogous. Both are absolutely precise digital codes, and no other codes that we know of have anything like the complexity of these two, with the size of a genome being of roughly the same order of magnitude (3×10^9 nucleotides for *Homo sapiens*) as the size of an operating system (2×10^9 bits for Windows 98). As a generation of engineers grows up with genomics, thinking digitally about the evolution of life on earth, our methods of computer programming will change. (Some ideas in this direction are already with us.) Traditionally, computer programs are written in a different way from biological ones. There's a programmer in the loop, an intelligence, which gives computer programs a logical structure that biological programs lack (not to mention comments!). Yet it is notable that nowadays, large-scale software systems are too big to be understood in detail by any individual, let alone mechanically analysed or verified, and indeed, the process of industrial software design already seems as close to evolution by natural selection as to mathematical logic. Software at a place like Microsoft is generated by an unending process of experiment and test, code and correct, a process in which individual human intelligences seem less important than they used to. Software systems evolve from one generation to the next, and they are never perfect, but they work. The process is repugnant to some computer scientists, but it is scalable and unstoppable.

Finally, a prediction that is not really a prediction, just a pious wish.

12. If we start thinking now, maybe we can cook up a good name for our field!

* * *

Table 1 lists some highlights from the history of scientific computing. Its attempt to extrapolate to the future summarises some of the thoughts I have expressed in this essay.

When I looked at this collection of predictions, I was startled to see that a theme emerges from them. Some are what one might call purely technical. The others, however, those marked by asterisks, suggest a trend:

Human beings will be removed from the loop. (*)

I find I have envisioned an unsettling future, a future in which humans, though still the taskmasters of computers, are no longer much involved in the details of getting the tasks done. Fifty years from now, it is hard to imagine that our machines will still be dim enough to benefit much from our assistance. Sketch your needs to the machine, and then—well, you might as well go have a cup of coffee.

That's my report from 2000, down here on the exponential.

Table 1. Some past and future developments in scientific computing. The asterisks mark items summarised by $(*)$.

Before 1940
Newton's method
Gaussian elimination
Gauss quadrature
least-squares fitting
Adams and Runge–Kutta formulas
Richardson extrapolation

1940–1970
floating point arithmetic
Fortran
finite differences
finite elements
simplex algorithm
Monte Carlo
orthogonal linear algebra
splines
FFT

1970–2000
quasi-Newton iterations
adaptivity
stiff ODE solvers
software libraries
Matlab
multigrid
sparse and iterative linear algebra
spectral methods
interior point methods
wavelets

2000–2048
linear algebra in $O(N^{2+\epsilon})$ flops
multipole methods
breakthroughs in preconditioners, spectral methods, time stepping for PDE
* speech and graphics everywhere
* fully intelligent, adaptive numerics
* loss of determinism
* seamless interoperability
* massively parallel computing made possible by ideas related to the human brain
* new programming methods made possible by ideas related to natural selection