Reproducibility in scientific computing

David H. Bailey  
http://www.davidhbailey.com  
Lawrence Berkeley National Laboratory, USA (retired)  
Computer Science Department, University of California, Davis, USA  
Collaborators:  
Jonathan M. Borwein (CARMA, University of Newcastle, Australia)  
Victoria Stodden (University of Illinois Urbana-Champaign, USA)  

September 29, 2014
Reproducibility in physics

In March 2014, a team of researchers from Harvard University made the dramatic announcement that they had discovered an interesting “twisting” pattern in cosmic microwave background data, measured using their BICEP2 experimental system.

This pattern fit very well with the hypothesized pattern of the most commonly assumed model of “inflation” in the first tiny fraction of a second after the big bang, and thus has been trumpeted as the first experimental evidence of the inflationary cosmology.

But other researchers had difficulty reconstructing the claimed results. Finally, two teams challenged the BICEP2 findings, saying that the results could more readily be explained by dust in the Milky Way.

Reproducibility in biomedicine

The biomedical field has been stung by numerous cases where pharma products look good based on clinical trials, but later disappoint in real-world usage, or the results cannot be reproduced in separate studies. Examples:

▶ In 2004, GlaxoSmithKline acknowledged that while some trials of Paxil found it effective for depression in children, other unpublished studies showed no benefit.

▶ In 2011, Bayer researchers reported that they were able to reproduce the results of only 17 of 67 published studies they examined.

▶ In 2012, Amgen researchers reported that they were able to reproduce the results of only 6 of 53 published cancer studies.

▶ In 2014, a review of Tamiflu found that while it made flu symptoms disappear a bit sooner, it did not stop serious complications or keep people out of the hospital.

These experiences have exposed a fundamental flaw in methodology: Only publicizing the results of successful trials introduces a bias into the results.

The AllTrials movement would require all results to be public: http://www.alltrials.net
High-performance scientific computing: An amazing success story

Performance of the world’s top 500 supercomputers (1994 – present)

Red = #1
Orange = #500
Blue = Sum #1 thru #500

Reproducibility in scientific computing

A December 2012 workshop on reproducibility in computing, held at Brown University in Rhode Island, USA, noted that

*Science is built upon the foundations of theory and experiment validated and improved through open, transparent communication. With the increasingly central role of computation in scientific discovery, this means communicating all details of the computations needed for others to replicate the experiment.*

*... The “reproducible research” movement recognizes that traditional scientific research and publication practices now fall short of this ideal, and encourages all those involved in the production of computational science ... to facilitate and practice really reproducible research.*

Reproducibility in scientific computing: Four key issues

From the ICERM workshop and other recent discussions, four key issues have emerged:

1. The need to institute a culture of reproducibility.
2. The danger of statistical overfitting and other errors in data analysis.
3. The need for greater rigor and forthrightness in performance reporting.
4. The growing concern over numerical reproducibility.

We will deal with each of these items in the remainder of the talk.
Reproducibility in scientific computing:
1. A culture of reproducibility

The ICERM report noted:

*Unfortunately the scientific culture surrounding computational work has evolved in ways that often make it difficult to verify findings, efficiently build on past research, or even to apply the basic tenets of the scientific method to computational procedures. Bench scientists are taught to keep careful lab notebooks documenting all aspects of the materials and methods they use including their negative as well as positive results, but computational work is often done in a much less careful, transparent, or well-documented manner. Often there is no record of the workflow process or the code actually used to obtain the published results, let alone a record of the false starts. This ultimately has a detrimental effect on researchers’ own productivity, their ability to build on past results or participate in community efforts, and the credibility of the research among other scientists and the public.*
Some commonly heard opinions and practices

- “We can’t provide completely full details of our computational techniques — these are our competitive edge over other research teams...”
- “Source code? A student wrote that. It’s not very important...”
- “What was the system environment? We just used what the campus computer center had available at the time...”
- Which version of the source code did we use? I don’t know. After running the code for our conference submission, we immediately started making some additional enhancements, so that is not available...”
Specific cultural issues mentioned in the ICERM report

- The need to carefully document the full context of computational experiments—system environment, input data, code used, computed results, etc.
- The need to grant open access, whenever possible, to computer code and input datasets, permitting other researchers to rerun the calculation and reproduce the published results.
- The need to save the code and relevant data in a permanent repository.
- The need for journal editors to acknowledge importance of computational details, and to encourage full documentation of the computational techniques, perhaps on an auxiliary website if not in the journal itself.
- The need for reviewers, research institutions and funding agencies to recognize the importance of computing and computing professionals, and to allocate funding for after-the-grant support and repositories.
- The need to encourage publication of negative results—other researchers can often learn from them.
Best practices for publishing research in scientific computing

1. A precise statement of assertions to be made in the paper.
2. The computational approach, and why it constitutes a rigorous test.
3. Complete statements of, or references to, every algorithm employed.
4. Auxiliary software (both research and commercial software).
5. Test environment, including hardware, software and number of processors.
6. Data reduction and statistical analysis methods.
7. Adequacy of precision level and grid resolution.
8. Full statement or summary of experimental results.
9. Verification and validation tests performed.
10. Availability of computer code, input data and output data.
11. Curation: where are code and data available?
12. Instructions for repeating computational experiments.
13. Terms of use and licensing. Ideally code and data “default to open.”
14. Avenues explored and negative findings.
15. Proper citation of all code and data used.
Consequences of high performance

Very high performance computer systems, either massively parallel networks or stand-alone workstations, provide unprecedented capability to process huge datasets and simulate complex natural phenomena.

But there are dangers ahead:

1. Since computer programs can easily explore many variations of a model, given an input dataset, the resulting “optimal” models are often statistically overfit.
2. Performance analyses of these complex systems are fraught with potential error and distortion.
3. Large computations greatly magnify sensitivity to numerical roundoff error, yielding results that are in some cases unreliable and nonreproducible.

We will explore each of these dangers in this talk.
2. Statistical overfitting and data analysis errors

- **Statistical overfitting** means either the development of a model that is fundamentally more complex than the data that it is based on, or else presenting a single “optimal” model, based on an input dataset, not mentioning many other variations of the model that were explored and discarded.

- Overfitting is increasingly common in scientific computing, because the rapidly increasing performance of modern scientific computing systems makes it very easy to explore thousands, millions or even billions of variations of a model.

- Overfitting can be seen as an instance of *selection bias*, namely the well-known effect where a researcher only discloses the results of successful experiments, and ignores or hides results of unsuccessful experiments.

Selection bias

Consider the following absurd financial scheme:

▶ A financial advisor sends letters to $10,240 = 10 \times 2^{10}$ potential clients, with 5120 letters predicting a certain security will go up, and the other half predicting it will go down.

▶ One month later, the advisor sends letters only to the 5120 investors who were previously sent the correct prediction, with 2560 letters predicting a certain security will go up, and the other half predicting it will go down.

▶ The advisor continues this process for 10 months.

▶ The remaining ten investors, having seen ten consecutive spot-on predictions, doubtless would be impressed at the advisor’s remarkable prescience!

This string of ten predictions is a clear instance of selection bias, since the final ten investors are never told of the many other failed predictions.
Statistical overfitting in mathematical finance

Investment strategies derived from computer simulation often look great on paper, but fall flat in practice, due to *backtest overfitting*:

▶ Since present-day high-performance computer systems (or even a workstation) can easily analyze millions of variations of a given investment strategy, each evaluated against a single historical market dataset (a “backtest”), it is quite likely that the “optimal” strategy will be overfit.

▶ Using this “optimal” strategy on other datasets, or in actual market trading, will not reproduce the strategy’s claimed performance — it was “optimized” against idiosyncrasies of the input dataset, and is otherwise ineffective.

▶ Backtest overfitting is widespread in the field, since published schemes are typically not accompanied by any information regarding testing methodology or how many variations of the strategy were explored when developing the strategy.
How easy is it to overfit a strategy derived from backtests? Very!

- If only 2 years of daily market data are available, then no more than 7 strategy variations should be tried.
- If only 5 years of daily market data are available, then no more than 45 strategy variations should be tried.


Demonstration of backtest overfitting

The following three viewgraphs present the results of different steps in an attempt to find an “optimal” investment strategy using a computational search process, based on a simulated (pseudorandom) backtest market dataset.

As you can see, by tweaking some very basic parameters (entry date, sell date, length of time in market, stop-loss, etc. — there are many thousands of such combinations), the computer search process can fit and “predict” the underlying dataset quite well, and yet not possess any fundamental “skill” for any other dataset.

You can try this simulator yourself at: http://datagrid.lbl.gov/backtest

Initial strategy applied to input (In Sample) dataset

$$(\text{Iter, Entry, Holding, Stop, Side}) = (23, 1, 1, -10, -1)$$

SR = 0.06  PSR = 0.13  Freq = 23.79
Final optimized strategy applied to input (In Sample) dataset

(Iter, Entry, Holding, Stop, Side) = (6376, 14, 16, -2, 1)

SR = 1.32  PSR = 2.89  Freq = 160.33
Final optimized strategy applied to new (Out of Sample) dataset

\[(\text{Entry, Holding, Stop, Side})=(14, 1, -2, 16)\]

\[\text{SR}=-0.03 \quad \text{PSR}=-0.06 \quad \text{Freq}=161.13\]
Analysis of the optimization results

- Note that the Sharpe ratio ("SR," a standard measure of investment performance) increases from 0.06 for the initial strategy to 1.32 (fairly good) for the final optimized strategy.
- But when the final "optimized" strategy is applied to the new (Out of Sample) dataset, the Sharpe ratio is -0.03, indicating a completely ineffective strategy.
- Any attempt to reproduce the promising-looking results obtained from the first dataset on any other dataset (or in real-life finance) will be disappointed.

Backtest overfitting is now thought to be a chief reason that many new financial products and strategies look great on paper (based on backtest evaluations) but then fall flat when actually fielded in the market.
Reproducibility in scientific computing:
3. Performance reporting practices

High-performance computing, 1990-1994:

▶ Many new parallel systems had been introduced; each claimed theirs was best.
▶ Many researchers were excited about the potential of highly parallel systems.
▶ Few standard benchmarks and testing methodologies had been established.
▶ It was hard to reproduce published performance results; much confusion reigned.
▶ Overall, the level of rigor and peer review in the field was rather low.

In response, in 1991 DHB published a humorous essay ”Twelve Ways to Fool the Masses,” poking fun at some of the abuses. Since abuses continued, DHB presented a talk at Supercomputing 1992 and published a paper with specific examples.

“Twelve ways to fool the masses in highly parallel computing”

1. Quote 32-bit performance results, not 64-bit results, but don’t mention this in paper.
2. Present performance figures for an inner kernel, then represent these figures as the performance of the entire application.
3. Quietly employ assembly code and other low-level language constructs.
4. Scale up the problem size with the number of processors, but omit any mention of this.
5. Quote performance results projected to a full system.
6. Compare your results against scalar, unoptimized code on conventional systems.
7. When run times are compared, compare with an old code on an obsolete system.
9. Quote performance as processor utilization, parallel speedups or Mflop/s per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and animated videos, and don’t discuss performance.
In some published papers and conference presentations, performance results on small-sized parallel systems were linearly scaled to full-sized systems, without even clearly disclosing this fact.

Example: 8,192-CPU performance results were linearly scaled to 65,536-CPU results, simply by multiplying by eight.

Excuse: “We can’t afford a full-sized system.”

This and the other examples mentioned in the next few viewgraphs are presented in:

In many cases, inefficient algorithms were employed for the highly parallel implementation, requiring many more operations, thus producing artificially high Mflop/s rates:

- Numerous researchers cited parallel PDE performance based explicit schemes, where implicit schemes were known to be much better. *Excuse: Explicit schemes “run better” on the researchers’ parallel system.*

- One paper cited performance for computing a 3D discrete Fourier transform by direct evaluation of the defining formula \(8n^2\) operations), rather than by using a fast Fourier transform \(5n \log_2 n\). *Excuse: Direct computation was “more appropriate” for the architecture being analyzed.*

Both examples violate a rule of professional performance reporting, namely to base the operation count (when computing Mflop/s or Gflop/s rates) on the best practical serial algorithm, no matter what scheme was actually used on the parallel system.
1992 paper: Not actually performing a claimed computation

Abstract of published paper: “The current Connection Machine implementation runs at 300-800 Mflop/s on a full [64K] CM-2, or at the speed of a single processor of a Cray-2 on 1/4 of a CM-2.”

- Excerpt from text: “This computation requires 568 iterations (taking 272 seconds) on a 16K Connection Machine.”
  *In other words, the computation was run on a 16K system, not on a 64K system; the figures cited in the Abstract were merely multiplied by four.*

- Excerpt from text: “In contrast, a Convex C210 requires 909 seconds to compute this example. Experience indicates that for a wide range of problems, a C210 is about 1/4 the speed of a single processor Cray-2.”
  *In other words, the computation mentioned in the Abstract was not actually run on a Cray-2; instead, it was run on a Convex system, and a questionable rule-of-thumb scaling factor was used to produce the Cray-2 rate.*
1992 paper: Performance plot [parallel (lower) vs vector (upper)]
1992 paper: Data for performance plot

<table>
<thead>
<tr>
<th>Problem size (x axis)</th>
<th>Parallel system run time</th>
<th>Vector system run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>8:18</td>
<td>0:16</td>
</tr>
<tr>
<td>40</td>
<td>9:11</td>
<td>0:26</td>
</tr>
<tr>
<td>80</td>
<td>11:59</td>
<td>0:57</td>
</tr>
<tr>
<td>160</td>
<td>15:07</td>
<td>2:11</td>
</tr>
<tr>
<td>990</td>
<td>21:32</td>
<td>19:00</td>
</tr>
<tr>
<td>9600</td>
<td>31:36</td>
<td>3:11:50*</td>
</tr>
</tbody>
</table>

Details in text of paper:
- In last entry, the 3:11:50 figure is an “estimate.”
- The vector system code is “not optimized.”

Note that the parallel system is actually slower than the vector system for all cases, except for the last (estimated) entry. Also, except for the last entry, all real data in the graph is in the lower left corner.
Fast forward to 2014: New ways to fool the masses

- Citing performance rates for a run with only one processor core active in a shared-memory multi-core node, producing artificially inflated performance (since there is no shared memory interference) and wasting resources (since most cores are idle).
  - Example: Citing performance on “1024 cores,” even though the code was run on 1024 multicore nodes, using only one core per node, with 15 out of 16 cores idle.

- Claiming that since one is using a graphics processing unit (GPU) system, that efficient parallel algorithms must be discarded in favor of more basic algorithms.

- Citing performance rates only for a core algorithm (such as FFT or linear system solution), even though full-scale applications have been run on the system.

- Listing only the best performance figure in the paper, even though the run was made numerous times (recall the experience of pharmaceutical tests).

- Employing special hardware, operating system or compiler settings that are not appropriate for real-world production usage.
Reproducibility in scientific computing:

4. Numerical reproducibility

The ICERM report noted:

Numerical round-off error and numerical differences are greatly magnified as computational simulations are scaled up to run on highly parallel systems. As a result, it is increasingly difficult to determine whether a code has been correctly ported to a new system, because computational results quickly diverge from standard benchmark cases. And it is doubly difficult for other researchers, using independently written codes and distinct computer systems, to reproduce published results.

Sensitivity to numerical round-off error

Many applications routinely use either 32-bit or 64-bit IEEE arithmetic, and employ fairly simple algorithms, assuming that all is well. But problems can arise.

Particularly vulnerable are:

1. Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors — numerical sensitivities are greatly magnified.
2. Certain applications with highly ill-conditioned linear systems.
3. Large summations, especially those involving cancellations.
4. Long-time, iterative simulations (such as molecular dynamics or climate models).
5. Computations to resolve small-scale phenomena.
6. Studies in computational physics or experimental mathematics often require huge precision levels.

The 2012 discovery of the Higgs boson at the ATLAS experiment in the LHC relied crucially on the ability to track charged particles with exquisite precision (10 microns over a 10m length) and high reliability (over 99% of roughly 1000 charged particles per collision correctly identified).

- Software: 5 millions line of C++ and python code, developed by roughly 2000 physicists and engineers over 15 years.
- Recently, in an attempt to speed up the calculation, researchers found that merely changing the underlying math library resulted in some collisions being missed or misidentified.

Questions:

- How serious are these numerical difficulties?
- How can they be tracked down?
- How can the library be maintained, producing numerically reliable results?
Solutions to numerical sensitivity and reproducibility problems

1. Employ an expert numerical analyst to re-examine every algorithm employed in the computation to ensure that only the most stable known schemes are being used.

2. Carefully analyze every step of the computation to ascertain the level of numerical sensitivity at each step.

3. Employ interval arithmetic for large portions of the application (which greatly increases run time and code complexity).

4. Employ higher-precision arithmetic, assisted with some “smart” tools to help determine where extra precision is needed and where it is not.

Item #4 is the only practical solution.
Numerical analysis expertise among U.C. Berkeley graduates

Of the 2010 U.C. Berkeley graduating class, 870 were in disciplines likely to require technical computing:

- Division of Mathematical and Physical Sciences (Math, Physics, Statistics).
- College of Chemistry.
- College of Engineering (including Computer Science).

Other fields (not counted) that will likely involve significant computing:

- Biology, geology, medicine, economics, psychology, sociology.

Enrollment in numerical analysis courses:


Conclusion: Fewer than 2% of Berkeley graduates who will do technical computing have had rigorous training in numerical analysis!
Enhancing reproducibility with high-precision arithmetic

Problem: Find the arc length of the irregular function
\[ g(x) = x + \sum_{0 \leq k \leq 10} 2^{-k} \sin(2^k x), \]
over the interval \((0, \pi)\) (using \(10^7\) abscissa points).

▶ If this computation is done with ordinary double precision arithmetic, the calculation takes 2.59 seconds and yields the result 7.073157029008510.
▶ If it is done using all double-double arithmetic (31-digit accuracy), it takes 47.39 seconds seconds and yields the result 7.073157029007832.
▶ But if only the summation is changed to double-double, the result is identical to the double-double result (to 15 digits), yet the computation only takes 3.47 seconds.

Graph of \(g(x) = x + \sum_{0 \leq k \leq 10} 2^{-k} \sin(2^k x),\) over \((0, \pi)\).
Aren’t 64 bits enough?

There has been considerable resistance in the scientific computing community to the notion that more than 64-bit arithmetic is not only useful, but may even be essential in some scientific computations. Why?

▶ Many are persuaded that physical reality fundamentally does not require high precision, beyond, say, the limits of 64-bit IEEE arithmetic.

▶ Many believe that numerical sensitivity problems are always due to the usage of inferior algorithms.

▶ Some researchers regard the usage of high-precision arithmetic as “cheating” or “sinful” — because it is too easy?

▶ Until a few years ago, easy-to-use high-precision arithmetic software facilities were not widely available to experiment with, or the computational cost was too great.

Enough excuses!

Problem: Find a polynomial to fit the data \((1, 1048579, 16777489, 84941299, 268501249, 655751251, 1360635409, 2523398179, 4311748609)\) for arguments \(0, 1, \cdots, 8\). The usual approach is to solve the linear system:

\[
\begin{bmatrix}
\sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \cdots & \sum_{k=1}^{n} x_k^{n+1} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{k=1}^{n} x_k^n & \sum_{k=1}^{n} x_k^{n+1} & \cdots & \sum_{k=1}^{n} x_k^{2n}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
\vdots \\
a_n
\end{bmatrix} =
\begin{bmatrix}
\sum_{k=1}^{n} y_k \\
\sum_{k=1}^{n} x_k y_k \\
\vdots \\
\sum_{k=1}^{n} x_k^n y_k
\end{bmatrix}
\]

A 64-bit computation (e.g., using Matlab, Linpack or LAPACK) fails to find the correct polynomial in this instance, even if one rounds results to nearest integer.

However, if Linpack routines are converted to use double-double arithmetic (31-digit accuracy), the above computation quickly produces the correct polynomial:

\[
f(x) = 1 + 1048577x^4 + x^8 = 1 + (2^{20} + 1)x^4 + x^8
\]
Algorithm changes versus double-double?
Double-double is the pragmatic choice

The result on the previous page can be obtained with double precision using Lagrange interpolation or the Demmel-Koev algorithm. But few scientists, outside of expert numerical analysts, are aware of these schemes.

Besides, even these schemes fail for higher-degree problems. For example: 
\((1, 134217731, 8589938753, 97845255883, 549772595201, 2097396156251, 6264239146561, 15804422886323, 35253091827713, 71611233653971, 135217729000001, 240913322581691, 409688091758593)\)

is generated by:
\[f(x) = 1 + 134217729x^6 + x^{12} = 1 + (2^{27} + 1)x^6 + x^{12}\]

In contrast, a straightforward Linpack scheme, implemented with double-double arithmetic, works fine for this and a wide range of similar problems.
Free software for high-precision computation


2. GMP. Produced by a volunteer effort and distributed under the GNU license. http://gmplib.org.


5. MPFUN90. Similar to ARPREC, but is written entirely in Fortran-90 and provides only a Fortran-90 interface. http://crd.lbl.gov/~dhbailey/mpdist.

Objective: Develop software facilities to find and ameliorate numerical anomalies in large-scale computations:

- Facilities to test the level of numerical accuracy required for an application.
- Facilities to delimit the portions of code that are inaccurate.
- Facilities to search the space of possible code modifications.
- Facilities to repair numerical difficulties, including usage of high-precision arithmetic.
- Facilities to navigate through a hierarchy of precision levels (32-bit, 64-bit, 80-bit or higher as needed).

The current version of this tool is known as “Precimonious.”

Some applications where high precision is useful or essential

1. Planetary orbit calculations (32 digits).
2. Supernova simulations (32–64 digits).
3. Climate modeling (32 digits).
4. Coulomb n-body atomic system simulations (32–120 digits).
5. Schrodinger solutions for lithium and helium atoms (32 digits).
7. Scattering amplitudes of fundamental particles (32 digits).
8. Discrete dynamical systems (32 digits).
9. Theory of nonlinear oscillators (64 digits).
10. The Taylor algorithm for ODEs (100–600 digits).
11. Ising integrals from mathematical physics (100–1000 digits).
12. Problems in experimental mathematics (100–50,000 digits).

Researchers at LBNL have used quad-double arithmetic to solve for non-local thermodynamic equilibrium populations of iron and other atoms in the atmospheres of supernovas. 

Iron may exist in several species, so it is necessary to solve for all species simultaneously. 

Since the relative population of any state from the dominant state is proportional to the exponential of the ionization energy, the dynamic range of these values can be very large, and cancellations may occur. 

Quad-double arithmetic (62 digits) was employed to reduce these errors.

Climate modeling: High-precision for reproducibility

- Climate and weather simulations are fundamentally chaotic: if microscopic changes are made to the current state, soon the future state is quite different.
- In practice, computational results are altered even if minor changes are made to the code or the system.
- This numerical variation is a major nuisance for code maintenance.
- He and Ding found that by using double-double arithmetic in two key inner loops, most of this numerical variation disappeared.

Experimental mathematics:
Discovering new mathematical results by computer

Methodology:

1. Compute various mathematical entities (limits, infinite series sums, definite integrals, etc.) to high precision, typically 100–10,000 digits.

2. Use algorithms such as “PSLQ” to recognize these numerical values in terms of well-known mathematical constants.

3. When results are found experimentally, seek formal mathematical proofs of the discovered relations.

Many results have recently been found using this methodology, both in pure mathematics and in mathematical physics.

“If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics.” – Kurt Godel
The PSLQ integer relation algorithm

Let \((x_n)\) be a given vector of real numbers. An integer relation algorithm either finds integers \((a_n)\) such that

\[ a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0 \]

(to within the “epsilon” of the arithmetic being used), or else finds bounds within which no relation can exist.

The “PSLQ” algorithm of mathematician-sculptor Helaman Ferguson is the most widely used integer relation algorithm.

Integer relation detection requires very high precision (at least \(n \times d\) digits, where \(d\) is the size in digits of the largest \(a_k\)), both in the input data and in the operation of the algorithm.

Decrease of $\log_{10}(\min |y_i|)$ in multipair PSLQ run
The BBP formula for $\pi$

In 1996, a computer program, utilizing the PSLQ algorithm and high-precision (200-digit) computation, discovered this new formula for $\pi$:

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

This formula permits one to compute binary (or hexadecimal) digits of $\pi$ beginning at an arbitrary starting position, using a very simple scheme that requires only standard 64-bit or 128-bit arithmetic.

In 2004, Borwein, Galway and Borwein proved that no base-$n$ formulas of this type exist for $\pi$, except when $n = 2^m$.

BBP-type formulas (discovered with PSLQ) are now known for numerous other mathematical constants.

Poisson potential functions associated with lattice sums

Lattice sums arising from the Poisson equation have been studied widely in mathematical physics and also in image processing. We numerically discovered, and then proved, that for rational \((x, y)\), the two-dimensional Poisson potential function satisfies

\[
\phi_2(x, y) = \frac{1}{\pi^2} \sum_{m,n \text{ odd}} \frac{\cos(m\pi x) \cos(n\pi y)}{m^2 + n^2} = \frac{1}{\pi} \log \alpha
\]

where \(\alpha\) is an \textit{algebraic number}, i.e., the root of an integer polynomial

\[
0 = a_0 + a_1 \alpha + a_2 \alpha^2 + \cdots + a_n \alpha^n
\]

The minimal polynomials for these \(\alpha\) were found by PSLQ calculations, with the \((n + 1)\)-long vector \((1, \alpha, \alpha^2, \cdots, \alpha^n)\) as input, where \(\alpha = \exp(8\pi \phi_2(x, y))\). PSLQ returns the vector of integer coefficients \((a_0, a_1, a_2, \cdots, a_n)\) as output.


Samples of minimal polynomials found by PSLQ

\[ k \quad \text{Minimal polynomial for } \exp(8\pi \phi_2(1/k, 1/k)) \]

5 \[ 1 + 52\alpha - 26\alpha^2 - 12\alpha^3 + \alpha^4 \]
6 \[ 1 - 28\alpha + 6\alpha^2 - 28\alpha^3 + \alpha^4 \]
7 \[ -1 + 196\alpha + 1302\alpha^2 - 14756\alpha^3 + 15673\alpha^4 + 42168\alpha^5 - 111916\alpha^6 + 82264\alpha^7 \]
\[ -35231\alpha^8 + 19852\alpha^9 - 2954\alpha^{10} - 308\alpha^{11} + 7\alpha^{12} \]
8 \[ 1 - 88\alpha + 92\alpha^2 - 872\alpha^3 + 1990\alpha^4 - 872\alpha^5 + 92\alpha^6 - 88\alpha^7 + \alpha^8 \]
9 \[ -1 + 534\alpha + 10923\alpha^2 - 342864\alpha^3 + 2304684\alpha^4 - 7820712\alpha^5 + 13729068\alpha^6 \]
\[ -22321584\alpha^7 + 39775986\alpha^8 - 44431044\alpha^9 + 19899882\alpha^{10} + 3546576\alpha^{11} \]
\[ -8458020\alpha^{12} + 4009176\alpha^{13} - 273348\alpha^{14} + 121392\alpha^{15} \]
\[ -11385\alpha^{16} - 342\alpha^{17} + 3\alpha^{18} \]
10 \[ 1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8 \]

For \( k = 32 \), the minimal polynomial has degree 128, with individual coefficients ranging up to 56 digits long. This PSLQ computation required 10,000-digit precision.

Other polynomials required up to 50,000-digit precision.
How much precision is enough?

- For any of these high-precision computations, there is no known way to know a priori what precision level should be used.
- Current ad-hoc approach: Increase precision until a consistent, credible-looking result is achieved, then reduce until one finds roughly the minimum precision needed.
- Clearly research is needed to better understand this process.
Summary

Reproducibility looms as a major challenge for the future of high-performance computing:

1. The need to institute a culture of reproducibility.
2. The danger of statistical overfitting and other errors in data analysis.
3. The need for greater rigor and forthrightness in performance reporting.
4. The growing concern over numerical reproducibility.

Technical and organizational solutions are available, but will the community adopt them?

This talk is available at http://www.davidhbailey.com/dhbtalks/dhb-reprod.pdf.