

Re: complexity of numerical software

Source: <http://sci.tech-archive.net/Archive/sci.math.symbolic/2005-12/msg00154.html>

- *From:* Jean-Claude Arbaut <jcarbaut@xxxxxxxxxxxx>
 - *Date:* Fri, 09 Dec 2005 21:03:24 +0100
-

David N. Williams wrote:

This is a repost of a message which appeared at my news server, but hasn't showed up in Google groups for several hours. My apologies if it's actually duplicated.

Jean-Claude Arbaut wrote:

Jaap Spies wrote:

Jean-Claude Arbaut wrote:

Jaap Spies wrote:

David N. Williams wrote in a reaction to carlos@xxxxxxxxxxxx:

I'm really puzzled by your classification of numerical so
I would have put it in the truly challenging category!?

-- David

Re: complexity of numerical software

Numerical software? What kind of numerical software you are talking about?

Maybe it would be interesting to know your definition of "numerical software"...

Yes. We have the same question!

Okay. I thought we were talking about code like SLATEC, NSWC, CMLIB, etc. They have (almost) nothing to do with algorithms to find subsets, which I would classify with discrete or combinatorics algorithms.

SLATEC, etc., are indeed examples of the kind of thing I had in mind.

It's easy to write poor numerical algorithms (with my definition), the usual example is "Numerical Recipes", but I find much more difficult to write *good* (read /accurate and fast/) ones. Just try to write a correctly rounded sine (or even a square root !).

Jaap Spies also wrote:

[...]

Testing and debugging of numeric algorithms is easy. Just check the results.

Re: complexity of numerical software

Actually, I would include testing numerical analysis software in the truly challenging category. A classic example is William Cody's celefant suite of accuracy tests for complex elementary functions. It makes clever use of mathematical identities, but while useful it is still limited, as Cody points out in his ACM article on Algorithm 714. Surely it is nontrivial in general to analyze where inaccuracies for a numerical program are likely to occur, to even begin testing.

Jean-Claude mentioned the square root. Here's a quote (up to math symbols) from one of those truly wonderful old IBM manuals for VS FORTRAN:

Effect of Argument Error

$\epsilon \sim \delta/2$

Accuracy

The accuracy of the algorithm is perfect; for all $(15 \cdot 2^{27})$ nonnegative normalized short floating-point numbers x , SQRT returns the correctly rounded value...

There's clearly nontrivial error analysis underlying IBM's implementation of SQRT,

I saw that in MacOSX implementation of function sqrt, which I suspect comes from IBM: two Newton iteration, plus a correction on the last bits. Everybody can write a Newton iteration, but the remaining part is what makes good algorithms...

which you have to worry about if you don't have access to a trusted implementation like theirs. Then the known relationship between the input error, δ , and the output error, ϵ , maybe gives you some hope of understanding the results of accuracy tests on code that uses SQRT.

Re: complexity of numerical software

And the subtleties of floating-point arithmetic feed in as well.

-- David

Very difficult to write, indeed ;-)

.