The Numerical Analyst as Computer Science Curmudgeon

Prof. W. Kahan
Elect. Eng. & Computer Sci. Dept., and Math. Dept.

http://www.cs.berkeley.edu/~wkahan

Presented in a 20 min. time-slot on Thurs. 5 Sept. 2002 to the
E.E. & C.S. Dept. Research Fair
acquainting grad. students with the faculty and their research interests.

You probably have heard these conflicting sayings:

0: " Don't sweat the details."

1: " God is in the details."

2: " The Devil is in the details."

Too much of my work in Computer Science arises out of the inordinate impact, upon scientific and engineering computation, of details now controlled by computing professionals most of whom lack experience with numerical computation.

When I started computing in 1953, numerical computations were practically all that computing was about. Now computers have ramified so far beyond Numerical Analysis that it seems like a sliver under the fingernails of Computer Scientists though it is still crucial to other scientists and engineers including several in this department.

. . . .

Aside: Who first said … ?

1: ? G. Flaubert –> "Pope Julius II defending Michaelangelo's Sistine Chapel Roof" ?

2: ? Mies van der Rohe, founder of the Bauhaus School of Architecture ?

I wish I could locate the original sources.

The State of the Art in Numerical Computer Science

Floating-point hardware has become pretty good for both

numerically expert scientists, engineers & statisticians, and
numerically inexpert but otherwise clever programmers,
thanks in part to IEEE Standard 754 for Binary Floating-Point Arithmetic.

BUT

Programming languages generally persist in archaic practices

that prevent the benefits of good floating-point hardware from reaching
numerically inexpert but otherwise clever programmers
and the users of their programs in
business, government, multimedia, games, etc.

These archaic practices were inherited from compromises to which we had to acquiesce when compilers had to fit in into 128 KB of memory and compile programs in one pass:

- •Modern arithmetic capabilities unsupported: Directed rounding, Humane exception handling.
- •Reckless compiler “optimizations” reorder arithmetic ignoring roundoff, among other things.
- •Narrow-minded evaluations of expressions with mixed data-types squander precision, and cripple certain desired kinds of operator overloading.

Some languages and compilers are rather worse for numerical work than others …

Java, and

Microsoft's compilers with Windows NT, 2000 and XP

are *dangerous* to use for floating-point computation.

Why?

See: **"How Java's Floating-Point Hurts Everyone Everywhere"**
(coauthored with Joe Darcy, a former student, now at Sun), and

"Matlab's Loss is Nobody's Gain"

both on my web page.

Three of the dangers will be illustrated here by examples …

Example 1: **Borda's Mouthpiece**, a classical two–dimensional fluid flow

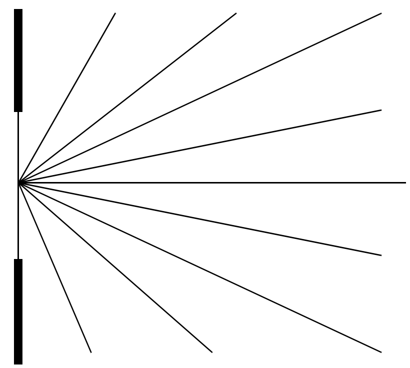
Define complex analytic functions

$$g(z) = z^2 + z \cdot \sqrt{z^2 + 1} \quad , \quad \text{and} \qquad F(z) = 1 + g(z) + \log(g(z)) \quad .$$

Plot the values taken by $F(z)$ as complex variable z runs along eleven rays

$$z = r \cdot \mathrm{i}\ , \quad z = r \cdot e^{4\mathrm{i}\cdot\pi/10}, \quad z = r \cdot e^{3\mathrm{i}\cdot\pi/10}, \quad z = r \cdot e^{2\mathrm{i}\cdot\pi/10}, \quad z = r \cdot e^{\mathrm{i}\cdot\pi/10}, \quad z = r$$

and their Complex Conjugates, taking positive r from near 0 to near $+\infty$.

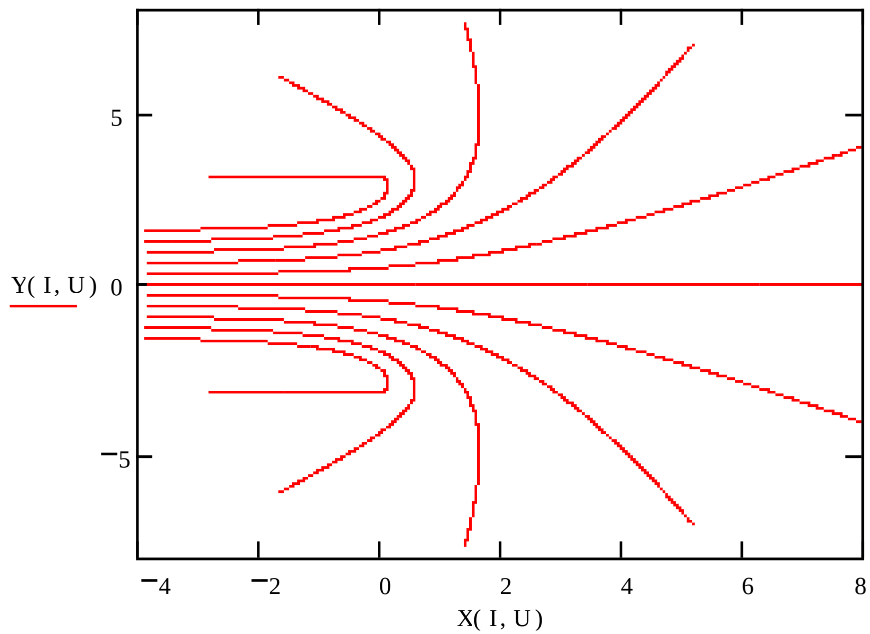


These rays are streamlines of an ideal fluid flowing in the right half-plane into a sink at the origin. The left half-plane is filled with air flowing into the sink. The vertical axis is a free boundary; its darker parts are walls inserted into the flow without changing it. The function $F(z)$ maps this flow *conformally* to a flow with the sink moved to $-\infty$ and the walls, pivoting around their innermost ends, turned into the left half-plane but kept straight to form the parallel walls of a long channel. (Perhaps the Physics is idealized excessively, but that doesn't matter here.)

The expected picture, " Borda's Mouthpiece," should show eleven streamlines of an ideal fluid flowing into a channel under pressure so high that the fluid's surface tears free from the inside of the channel.

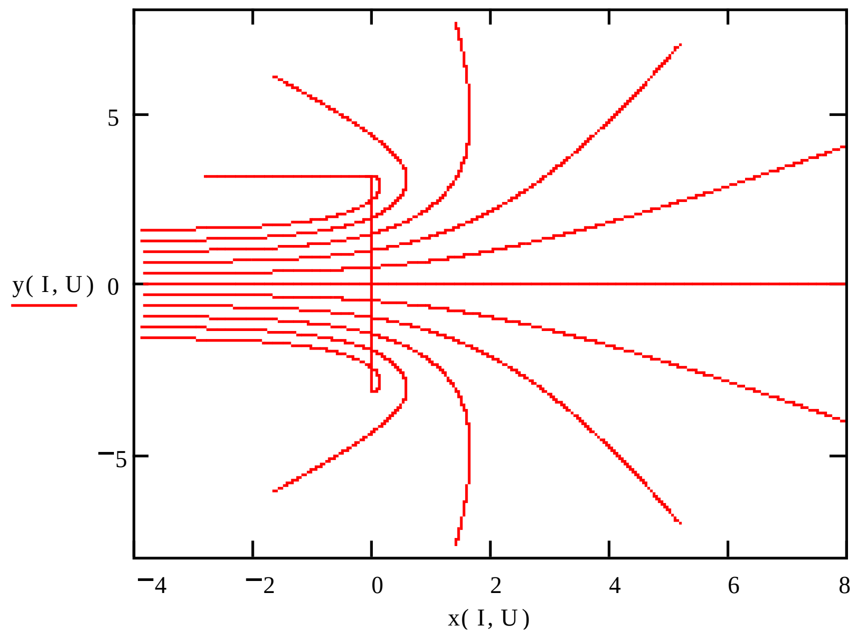
Borda’s Mouthpiece

Correctly plotted Streamlines



Plotted using C99–like *Complex* and *Imaginary*

Streamlines should not cut across each other **!**

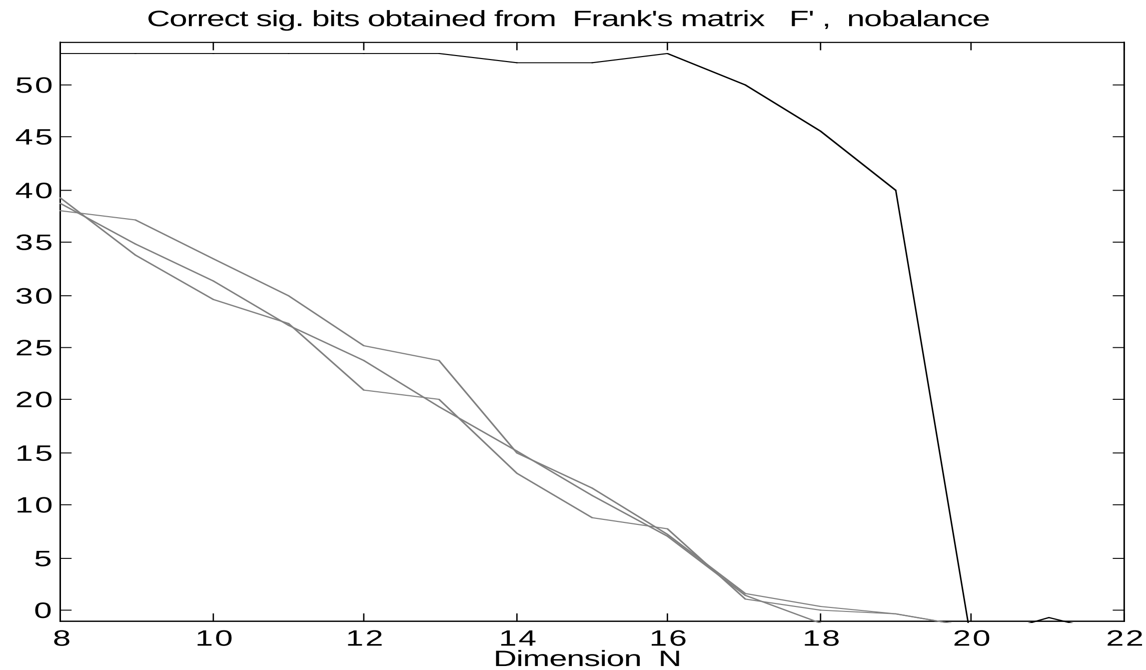


Misplotted using Java/Fortran–like *Complex*

An *Ideal Fluid* under high pressure escapes to the left through a channel with straight horizontal sides. Inside the channel, the flow's boundary is *free,*— it does not touch the channel walls. But when –0 is mishandled, as Fortran-style Complex arithmetic must mishandle it, that streamline of the flow along and underneath the lower channel wall is misplotted across the inner mouth of the channel and, though it does not show above, also as a short segment in the upper wall at its inside end. Both plots come from the same simple program using different Complex Class libraries, first with and second without an *Imaginary* Class introduced into C99 mostly through the efforts of Jim Thomas, a former student here, now with H-P.

Example 2: **Iterative Refinement of Computed Eigenvalues/vectors**

Accuracy in sig. bits is plotted here against the dimensions of a family of test matrices:



Legend: ——— `Refineig` on 680x0-Mac or Intel-PC & old Matlab on Windows 98 or earlier

·-·-·-· `Refineig` on Power-Mac or newer Matlab on PC with Windows NT and later

. . . . . . `eig` unrefined results are inaccurate on all the machines mentioned above.

Old 680x0-based Macs and older versions of Windows on PCs yield better accuracy **!**

Why? *For compatibility with DEC Alpha* **!** Windows NT/2000/XP disable Pentium's extra-precise (11 extra sig. bits) registers intended to accumulate matrix products . And Apple switched Macs to a RISC architecture with inadvertently inferior floating-point.

Example 3: **A programming Joke**

Removal of algebraically redundant parentheses corrects a programmer’s mistake:
(Usually, in floating-point expressions, such parentheses are best left in place; this is an exception.)

“ `C=(F-32)*(5/9)` ” gets the wrong result; can you see why?
“ `C=(F-32)*5/9` ” gets the right result,
converting Fahrenheit `F` to Celsius `C` .
(See comp.lang.java.help for 1997/07/02 .)

An archaic programming language convention about mixed-type expressions invites that kind of error.

This convention is a mistake, not a joke.

This convention also impedes techniques like …
- •Exploitation of Interval Arithmetic to help assess and control numerical uncertainty
- •Arithmetic of arbitrarily high-precision variable at run-time

that would enhance greatly the reliability of floating-point software generated by programmers innocent of exposure to floating-point error-analyses.

It imposes error-prone redundant references to multiple coordinate systems unnecesarily upon geometrical computations using operator-overloaded object-oriented linear algebra.

Now that we have seen enough examples of mysterious misbehavior,

What needs doing?

Among students planning to specialize in programming languages and compiler technology, we need some to learn enough about

- Numerical Analysis (Math. 128, 221, 228), particularly about
- Error-Analysis (Math. 273…), and also about
- Computer System Support for Sci. and Eng. Computation (CS 279)

that they can participate competently in a slow but steady evolution towards more humane programming environments safer for numerical work.

“Think not of Duty nor Indulgence;
think about Self-Defence.”

See “Miscalculating Area and Angles of a Needle-like Triangle” on my web page.