Symbolic Computation: An (Almost) Indispensible Tool For R & D

Daniel Lichtblau
danl@wolfram.com
Wolfram Research, Inc.
100 Trade Center Dr.
Champaign IL 61820

NSF Workshop:
Future Directions of Symbolic Computation Research and their Applications to the Domain Sciences
University of Rhode Island
April 30−May 1, 2009

ABSTRACT

I hope to explain some of the myriad ways in which symbolic computation, once the province of computational physics and math, is now percolating (sometimes flooding, sometimes trickling, sometimes, alas, reversing course) to the "real world". I also will try to explain why we might believe we have something of value to offer, and what I believe that something to be.

Disclaimer

(1) To state what should be obvious: everything mentioned is my own opinion. I do not speak for my employer (Wolfram Research). And I do not pretend to speak for anyone else in this session, and may offer opinions on what symbolic computation was, or is, that are at odds with the views of others, or with previous versions of myself.

(2) What I state will be, of necessity, heavily influenced by the work I do. This includes 17+ years of development in the kernel of *Mathematica*. That said, many aspects to functionality I describe have a flavor in other programs.

Historically

Symbolic computation, aka computer algebra, aka symbolic manipulation, is a merging of many tools from computational physics and algorithmic algebra. Possibly the main point of commonality between these two streams is that neither was particularly numeric in nature.

In 70’s and 80’s we see evolution toward algorithmic algebra.

- Some important computational achievements were the development of algorithms for:
  - Multivariate polynomial GCDs
  - Polynomial factorization
  - Gröbner bases and triangular sets ==> solving systems of equations
  - Lattice reduction (sounds like a salad dressing at an upscale restaurant)
  - Symbolic integration and summation

Digression for a pop quiz

- Question 1:
  In 1982, when it first appeared (okay, 1981), how many people here regarded lattice reduction as a part of symbolic computation? (My guess: one. Until Kaltofen misbehaves and we send him out of the room.)

- Question 2:
  How many people today would even consider that lattice reduction is not an important part of symbolic computation?

My point being, what we regard as symbolic computation is evolving, mostly growing. I see this as a very healthy reaction to an influx of good algorithms, challenging problems, and adaptability of tools. And toolsmiths.

Important additions along the way

- Acquisition of language
  - Of necessity, every symbolic computation program has developed a bona fide computer language.
- Use of computer graphics
- Numerics
  - I cannot emphasize how important this is, both for general usage and, specifically, for symbolic computation.
  - Since mid−90's we have seen the emergence of hybrid symbolic−numeric computation.
- General purpose interfaces
  - Okay, some people still like a raw "terminal window" interface (I do my *Mathematica* kernel debugging via that type of interface). But for most purposes e.g. getting good graphics, writing a paper (or slide show like this one), etc., the interactive interfaces are quite desirable.
- Interface capabilities to external programs

More recent additions...

- Support for web services
- Support for parallelization
- Importing and exporting from/to a slew of standard data formats
- Integrated Development Environment tools
  - Profiling
  - Debugging
  - Project management
- Configurable GUI interfaces to graphics
  - Really powerful in conjunction with e.g. computational dynamics code
- And— lest we forget— an ever increasing body of "classical" symbolic computation algorithms and functions
  - GCDs over algebraic fields (including prime characteristic case)
  - Limits of exp−log functions
  - Improvements to real solving
  - Better capabilities to utilize some functionality to improve other parts, e.g. using a real solver to locating singularities on an integration path
  - The above just scratch the surface. This is a vibrant field.

What do our programs and technologies offer?

Side remark: Few users give much thought to whether the code under the hood can find common factors in rational functions. Yet they want such functionality even when they do not realize it e.g. for "canonical forms" or simplification of expressions.

- A plethora of built−in mathematical tools
  - Efficient arithmetic (NEVER underestimate the importance of this, or forget how horribly slow we all were in the "bad old days")
  - Symbolic and numeric calculus (this is sweeping a huge amount of technology into one sub−item)
  - Symbolic and numeric linear and nonlinear algebra
  - Symbolic and numeric support for elementary and special functions
  - Number theory functions such as lattice reduction (or is that symbolic computation functionality?)
  - Optimization (linear and nonlinear, unconstrained and constrained, local and global)
  - Statistics functionality
  - Computational geometry (necessary for good graphics, useful in a variety of other ways)

What our programs offer...

- Rich programming languages
  - Can even emulate a Turing machine (okay, maybe that’s not so special...)
  - Necessary for bootstrap development; much development is now done in the language itself, and this offers considerable advantage in terms of development time and later maintainance.
- Interfaces to other software
  - Ability to import from and export to various programs. Quite useful for researchers working with e.g. large data sets, or programs by colleagues in other languages.

Uses of these programs

- Classrooms (in first year college math, at least since 80’s)
- Math & physics research (since 70’s, and even some in 60’s)
  - Automated theorem proving in geometry comes to mind as a huge success story in the field of symbolic computation. It fueled algorithm development and refined notions of what symbolic computation can do, what are the questions to ask, etc.
  - University algebra is now commonly taught with an eye toward computational methods, and even efficiency in those methods.

Uses...(more recently)

- Finance community (quants, financial modellers/analysts, actuarial work)
- Engineering R&D (automakers, jet manufacturers, chemical and electronics companies)
- Life sciences (medical imaging, compartmental models, cell/tissue/population dynamics, pharmaceutical design)
- Computer science (algorithm development and analysis)
- Government and university labs (avionics, biophysics, chemistry, data analysis, environmental science, fishery management, ..., materials sciences, nanotechnology, operations research, physics, ..., zoology)
- Economics/econometrics departments and organizations
- Other social sciences, modelling and simulation, insurance modelling, law analysis
- Arts and music (sculpture design, graphic art, electronic composition,...)
- Literature (okay, not really in literature, but certainly in mathematical text analysis)

Mathematica−specific use

- www.wolfram.com / solutions
  - Provides a wealth of information about domain−specific usage in industrial and related settings.
- demonstrations.wolfram.com
  - Various visualizations created with, but not requiring, *Mathematica*. Many arise in domain sciences.

What the Symbolic Computation community has to offer

- Experienced developers, development tools, etc.
  - We now have numerous professors, in departments around the world, training grad students in algorithm development.
  - We see an increased emphasis on software engineering, both in the academic world and more especially in the development sector (commercial and otherwise).
  - Program development, commercial and noncommercial, seems to be healthy.
- A growing experience base with "real world" problems.
- A growing user base (important for spreading the experience base). Their success is good for everyone (that is, everyone who thinks of R&D as a "good thing").

What the community can offer...

- Increasing availability of quality software numeric libraries e.g. Lapack, PHCPack, ...
  - We do not write all of these, but as a group have put them to good use.
  - These are important in their own right: numeric linear algebra, for example, is used in more ways than any of us can imagine.
  - Also they dovetail nicely with symbolic computation: LinBox and hybrid polynomial algebra methods can make use of them.
- Availability of quality "exact" math libraries e.g. GMP (and MPIR, soon if not already), LinBox, ...
  - From point of view of general purpose math programs, these are something of a common good that can be made available to the user base.
- A confluence of interests between those of us in symbolic computation, and the user community in domain sciences and elsewhere.

What the community can offer...

- We are well positioned to assist users, in effect passing along our expertise to the domain sciences.
  - Anectodally, I would guess this comprises 10−15% of my own work day, in responding to a myriad of questions that come through our Tech Support, Usenet users' group, and private email.
  - I often know little to nothing of the users' areas. In the past several months I have taken on problems in cartography (quickly determine if a given pixel resides in Canada), an intricate economic−and−climate model based on that of Nordhaus (the challenge: make it crank through a few hundred iterations), a mixed two−level optimization problem from chemistry, extraction of a planar slice from a complicated surface, and maybe another dozen or more I have forgotten.
- Once in a while, pursuit of our own research interests leads to applicable functionality. A few examples from my own backyard:
  - A Wolfram Research colleague received "best paper" award at ISSAC 2008 in recognition of some work he has done in real solving.
  - A silly computational number theory problem lead to some of our integer programming capabilities.
  - Development of numeric Gröbner bases, more or less for the challenge of doing so, lead a few years later to a global solver for numeric polynomial systems.

What are the failures of our field?

- Historically, an over−emphasis on pure "symbolic" functionality as opposed to numerics, user interface issues, programming language design and function, and more.
  - Happily, that seems to be in the past (declares victory, goes home).
- We have fallen quite short of the mark in convincing the potential user base that what we offer can increase productivity of researchers in virtually all fields that use computer programs.
  - I would classify this as the single largest failing in our field.
  - Again, this provides us with a challenge: Figure out those things we are not doing well, improve them, package them as "symbolic computation" if need be, and show people how to use them.
- As a field we have been slow to recognize, and support development of, "common good" libraries (GMP would be an example of one such). There are places where it simply does not make sense to reinvent wheels several times over.

Failures...

- Something I have heard from a few people at conferences: we have adopted too much of a "mathematical" attitude, at the expense of sound computer, computational, and engineering science.
  - One indication, anecdotal though it may be, is that twenty or so years ago the field was more heavily populated with physicists than is the case today.
  - More anecdote: I have heard from four or five people over the years that they will not submit work to publications in symbolic computation (due to poor treatment in past, I gather. Been there.) Much as I would prefer that people evolve thicker skins, the possibility remains that we might be losing good researchers.
- We have sometimes let differences over arcane issues obscure the fact that users in most application domains could care less.
  - Our time would be better spent insulting one another’s parentage than software development efforts.
  - There is but little to gain from the open/free vs. commercial/proprietary squabble that sometimes raises its tiny head. People want good software, not arguments over whether some of it might conceivably be provided via commercial products.
  - Squandering potential resources in bickering over what is, for most end users, obscurata, will not further development of good software. I’ve yet to see this debate do harm to the commercial players, but it seems not to be helping the academic side, on which much future research depends.

Other weak areas

We all have bugs in need of fixing. Everyone knows this (discussion thereof is even something of a cottage industry in one Usenet group). But why do we appear to have so many relative to other types of software?

- Mathematical code tends to have tremendous level of interdependency. For example...
  - Symbolic integration sits on top of rational function manipulation, limit and series functionality, real solving, and more.
  - Limits and series tend to use one another (that at least, has been my experience, although perhaps this is more specific to *Mathematica* than other programs).
  - Complicated graphis may require serious computational geometry under the hood
  - One is always to some extent at the mercy of library code, even when it is better than what we ourselves might separately write.

Weak areas...

- Speed. While improvements are coming along, many of us remain slow due to interpreter speed, deficiencies in sound just−in−time compilation, etc.
- Mathematical algorithms are sometimes complicated, hence easy to mess up in edge cases.
  - Many "standard" algorithms are "easy". But we all want code that is faster, and often go beyond standard to gain that order−of−magnitude improvement (e.g. in going from Euclidean algorithm to asymptotically fast gcd). Lemme tell you, it’s easy to incur the wrath of the bug wraith, when you go all out for speed.
- Mathematical functionality is in some places simply incomplete. That is, we have methods for handling some problems, but cannot always tell (at least, not without substantial work) when they might fail. Finding path singularities of elliptic integrals is an example where this can happen.

Weak areas...

- In the commercial sector (and perhaps most heavily in *Mathematica*), there has been far too little emphasis on apportionment of academic credit to those who develop the algorithms. But we need this because...
  - It is only fair.
  - Those who develop good algorithms then are further encouraged to do more of that. And the recognition of their work makes it easier to get grants, tenure, and all those little things that make their research continue.
  - Those who use the software, then have references to which they can go, should they be so inclined. Some people require details regarding how the software works, particularly when they rely upon it for work that goes into scientific publication.

Future tasks

What can we do?

- Improve our software systems engineering so that we better understand and can more readily track down and kill, or better still, prevent, these bugs.
- Encourage development and maintainance of "common good" libraries such as for bignum arithmetic, linear algebra, etc.
  - Should this be extended to polynomial algebra or other areas? I’m not sure but do sometimes wonder about this.
- Improve documentation in regards to areas for which functionality, and algorithm development as a whole, are incomplete at best.
  - Definite integration comes immediately to mind. It is the La Brea Tar Pit of symbolic computation.
- Obviously the field will be well served by some of us continuing to develope symbolic computation, in the classical sense.
- Figure out the important open problems for the next generation in Symbolic Computation to tackle.
- While we are seeing improved communication with numerics and applications domain experts, there is a need for still more.

Future...

- In the commercial sector, it is important that we continue to incorporate some of the notable work from the academic sector.
  - Impossible to include everything.
  - Different R&D teams will have different priorities.It is increasingly clear that we must be attentive to user base, both actual and potential.
  - Sometimes this means saying "no".
  - Sometimes it means less exciting development work that one might like.
  - Sometimes, no, often, it means working on program documentation.
  - It certainly means paying attention to bugs.
  - But overall, we need to keep thinking about what is useful in the literature (both new and old), and strive to put it to good use.
- Improve speed
  - Probably bad to sacrifice language semantics.
  - But can often be done on a subset of the language, in a way that improves capabilities for a diverse set of uses and users.

Summary

Always more work to do...

- Over past two+ decades Symbolic Computation has seen a tremendous transfer of technology to the private and non−academic, non−commerical sectors.
  - Availability in a plethora of scientific and other Research and Development environments means we have considerable leveraging of our community’s research efforts.
- We do not develop a plethora of "killer apps" in this game...
  - Rather, we amass a large repertoire of generally useful capabilities. Much of the development is, or at least seems to be, incremental.
  - The importance is in putting them together for general usage, and then teaching people how to use the software. With hundreds of thousands of users doing sophisticated R&D, a case can readily be made that...
- ...but these programs are themselves the Killer Applications.