Chebfun as a software project

Nick Trefethen, University of Oxford

Andrew Wiles Building
opened September 2013

Chebfun as a software project

Nick Trefethen, University of Oxford

+ Toby Driscoll, Nick Hale, Asgeir Birkisson, Anthony Austin, Alex Townsend, …

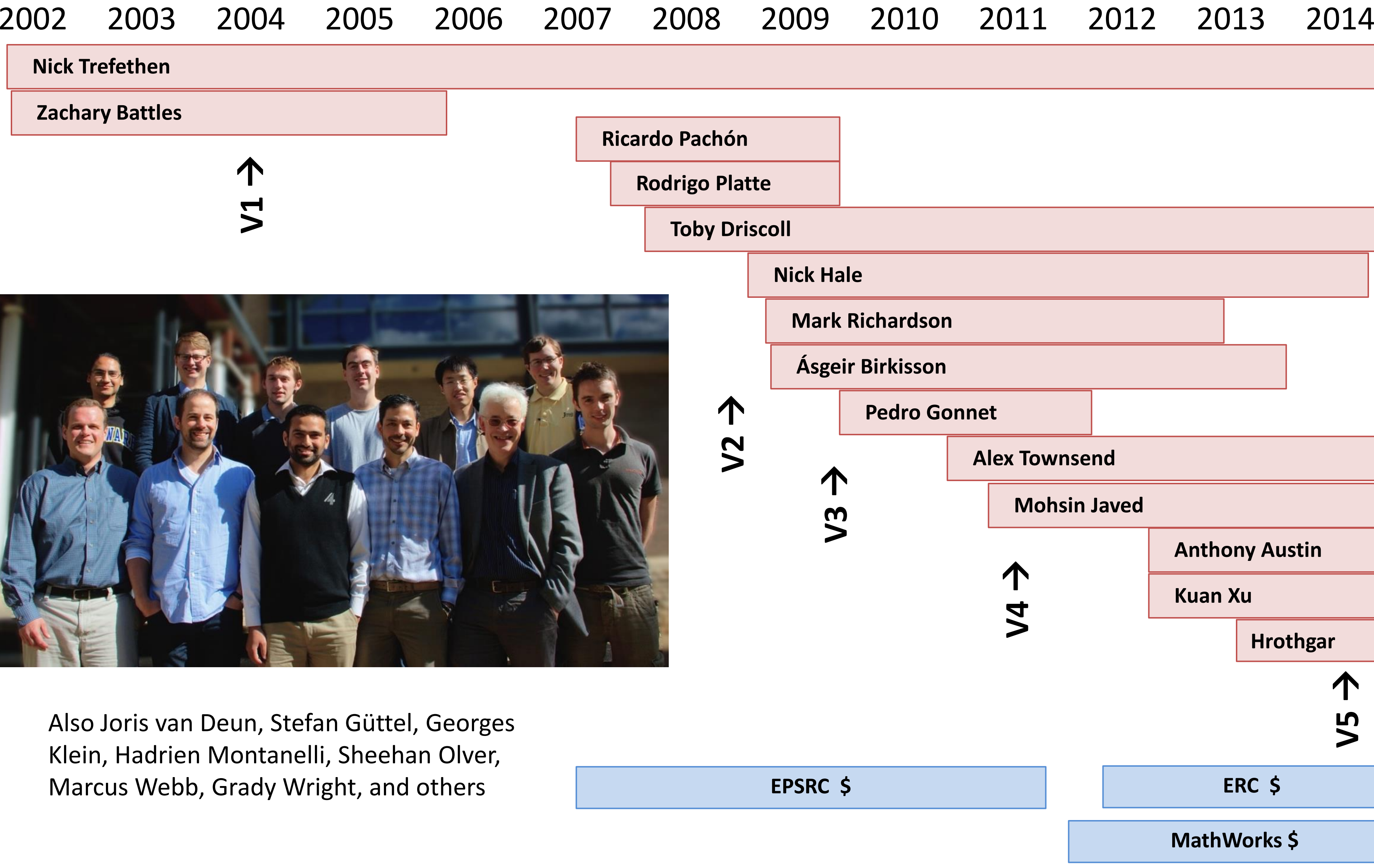
[Floating point arithmetic has been a triumphant success for computing with real numbers. The idea behind Chebfun is to do something analogous for computing with functions.

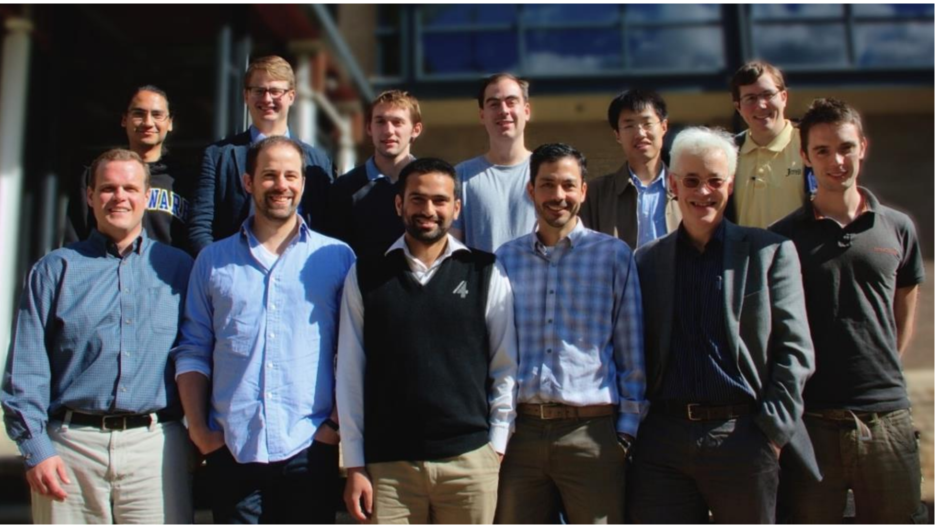
If x and y are 64-bit numbers, then xy has more like 128 bits. Floating point arithmetic rounds this to 64.

Analogously, Chebfun might represent two functions f and g by Chebyshev expansions of degrees m and n. Then mathematically, fg has degree m+n, but we truncate the expansion down to 16 digits.

Brief demonstration.]

CHEBFUN TIMELINE





Also Joris van Deun, Stefan Güttel, Georges Klein, Hadrien Montanelli, Sheehan Olver, Marcus Webb, Grady Wright, and others

Twenty questions

Q1. In-house project? Found a company? Open source?

Concern about the long term ruled out #1.

Lack of interest ruled out #2.
(Would it have been different in the USA or Korea?)

So we decided on #3, open source:

- code freely available
- open-source license
- open to outside developers
- no claim to long-term ownership

Q2. Which software licence?

We were tempted by GPL.

People told us “If it’s GPL, industry won’t touch it.”

We went with BSD.

It took a year to persuade Oxford’s bureaucrats to let us do this.

So far, no commercial or licensing issues have arisen.

Q3. Which programming language?

Chebfun’s starting idea was MATLAB-based:
overload MATLAB’s vectors/matrices to functions/operators.
MATLAB users are our obvious constituency.

A lot of people don’t like MATLAB. They talk about

• speed • language features • commercial product

Parts of “Core Chebfun” have been implemented by various people in C, Python, Octave, Maxima, and Julia.

The future?

Q4. How do we make money from Chebfun?

We don't.

In principle we could found a software consultancy.
Maybe one day?

Q5. How do we get funding to support our team?

Funding software is always a challenge.
(In Britain, funding anything is a challenge.)

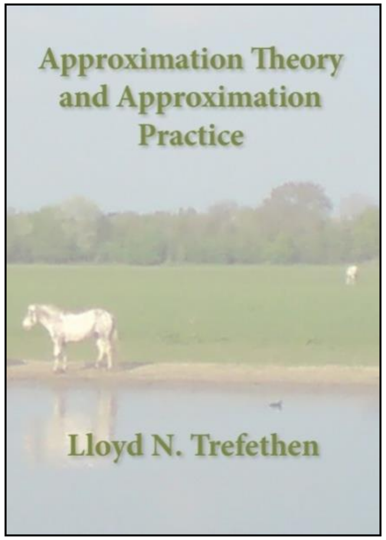
EPSRC CS 2007-11:	1 student, 2 postdocs, 1 long-term visitor
MathWorks 2011-15:	2 postdocs, 1 remote collaborator
EU ERC 2012-17*:	4 students, 3 postdocs, 2 long-term visitors
Self-funded:	5 students, 3 postdocs

* research, not software

Q6. How do people get credit for software work?

A universal problem. Software “doesn’t count”.

But informally, it can count greatly if your name is well associated with the software. Easy for me; harder for students and postdocs.



Our imperfect solution: make sure everybody publishes papers along the way.

The good news: the software project helps guide people toward work of genuine value.

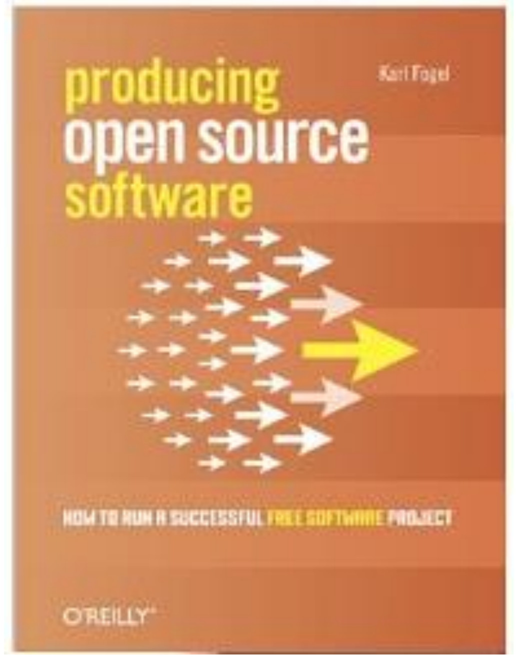
REF2014
Research Excellence Framework

Q7. How do we learn about open source software?

The youngsters on the team have grown up with this stuff.

As for me, I was stuck in Seattle by Eyjafjallajökull in 2010. William ("Sage") Stein persuaded me Chebfun should go open-source.

Stein recommended a marvelous book by Karl Fogel, freely available online.



Q8. Where do we host the code?

2003-2008: just 2-4 people, no version control

2008-2014: svn at Oxford

2014- : GitHub

GitHub is 6 years old and hosts >10 million repositories.
It is also our bug and issue tracker.

→ https://github.com

Q9. What’s the coding and review process?

Until 2013, we just wrote code and that was that.

Then we introduced style conventions and code review.
Standard GitHub mode: pull request ⟶ review ⟶ merge.

(We’ve had long discussions about spacing and CamelCase.)

User’s Guide and Examples collection are also on GitHub, but don’t use pull requests and merges. Instead, I manage them.

Q10. How about testing?

A fantastic change when we introduced “chebtest” in 2008!
It ran in under a minute.

This grew to several minutes, then 15. We introduced an automated chebtest each night. If something fails, we get email.

We try to test all codes, but are not completely systematic.

Last week, we added the user’s guide, the Examples collection, and *Approx. Theory and Approx. Practice* to the nightly run.

So far, we just test on one platform, since in principle MATLAB is platform-independent. Anomalies arise, however.

Q11. Who responds to help requests? Who fixes bugs?

Help requests go to help@chebfun.org.
Bugs are reported to help, or on the tracker.
Who responds?

We haven't solved this satisfactorily.
We've been reluctant to set up a ticket system.

Q12. How do we make a good web site?

Our solution has been to have one person do it: Hrothgar.
Hosted at GitHub. Python is used as a static site generator.

→ www.chebfun.org

Q13. How do we generate documentation?

I’ve written most of the user’s guide.
Most of our developers don’t much enjoy writing.

Individual developers write the help text in each file.

Should one try to automate some of this, doxygen-style?
(doxygen itself isn’t adapted to MATLAB.)

→ www.chebfun.org

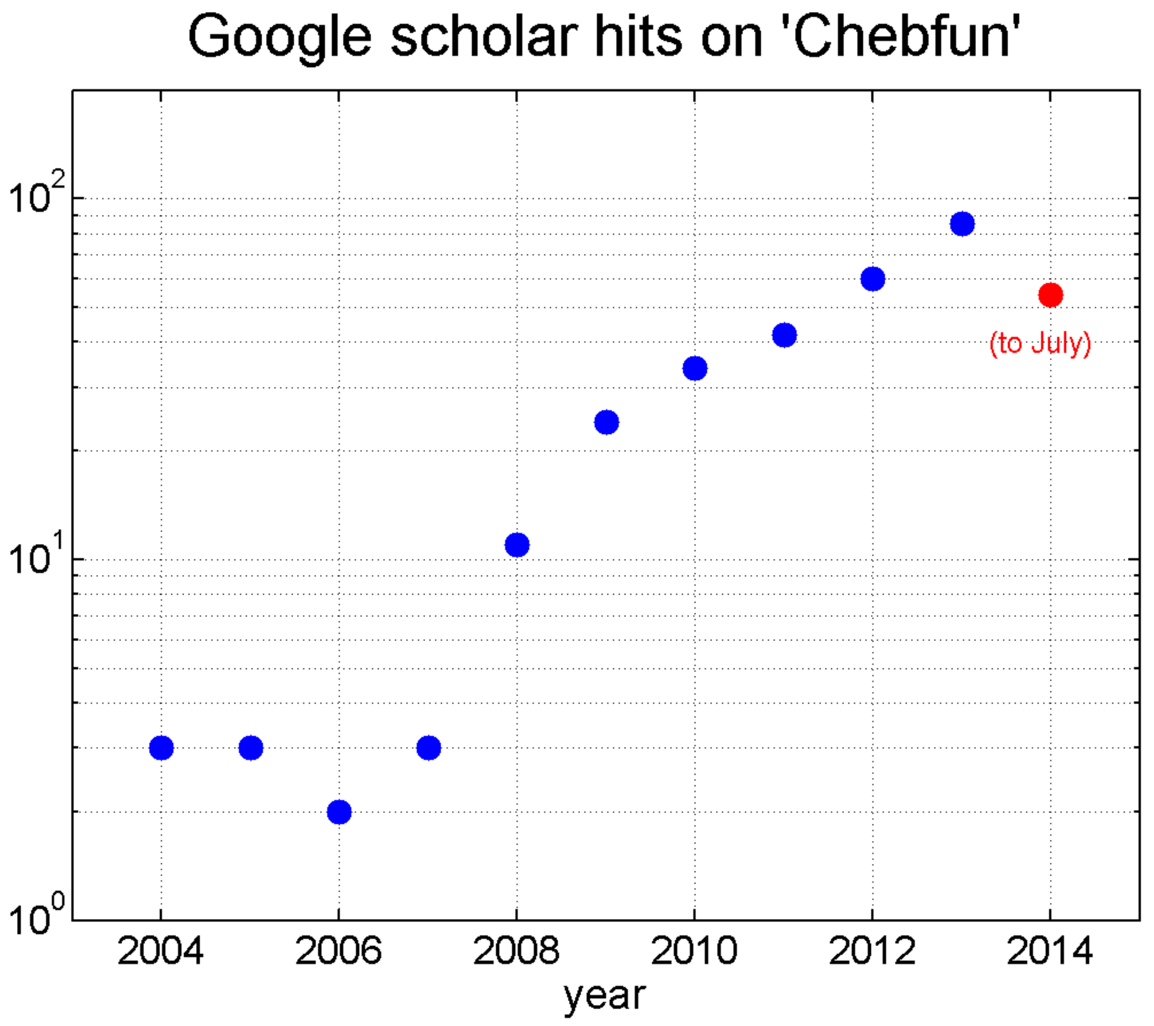
Q14. How do we get feedback from users?

Badly, so far!

You don’t have to register to download Chebfun. So we have no database of users and we know little about them.

≈20000 downloads so far, but how many are regular users? What do people do with Chebfun? Our knowledge is anecdotal.

Should we set up discussion mailing lists? A wiki?

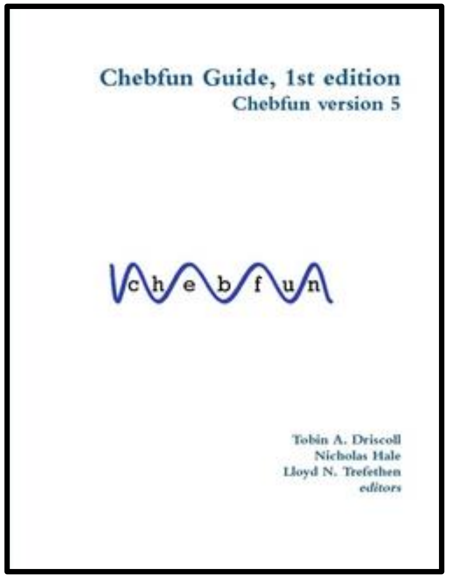

Google scholar hits on 'Chebfun'
10^2
10^1
10^0
2004
2006
2008
2010
2012
2014
year
(to July)

Q15. How should people cite Chebfun?

Some software projects ask you to cite a key paper, others a book, others the software itself (which journals don't like). But who is the author, in a team of dozens?

Our (recent) solution: package the user's guide as a freely-available .pdf book with three editors (Driscoll, Hale, Trefethen). Individual chapters may have any authors.

So we ask users to cite Driscoll, Hale, and Trefethen, eds., *The Chebfun Guide*.


Chebfun Guide, 1st edition
Chebfun version 5

Tobin A. Driscoll
Nicholas Hale
Lloyd N. Trefethen
editors

Q16. How do we communicate among ourselves?

The “Chebfun Team”: 8-10 of us at Oxford, one at Delaware (Toby Driscoll), one at Stellenbosch (Nick Hale). We have a private email address for the team.

The people at Oxford share offices and talk a lot. Everybody is constantly emailing and using the tracker.

Once a week we have a 75-minute meeting with a regular agenda and informal minutes. Delaware and Stellenbosch connect by Google Hangout. Far from ideal.

We try for the meetings to blend research and software management. Not always easy.

Q17. How do we attract outside developers?

This side of the project is completely unformed (we just went public on GitHub June 21). We hope to build it in the upcoming months.

Developer docs? Wiki? List of open tasks?

We’ve had nibbles of interest from outsiders, and so far, we haven’t known how to be very welcoming.

Q18. Should one plan ahead?

We haven’t much, so far. No Chebfun requirements analysis!

- The project was initiated almost on a whim (2002)
- Linear ODEs came along unplanned (a key idea in 2008)
- Extension to nonlinear ODEs also unexpected (key idea 2009)
- 2D was a vague ambition until we made it happen (2012)
- Periodic function representations arrived casually (2014)

The prospect of widening out to a developer community makes the question of planning more pressing.

Q19. What do we fight about?

Caricature of what the white-haired professor wants:
Beauty, elegance, simplicity! Clear and exciting concepts! Short, memorable command names. Good writing and good layout. Consistency with MATLAB. Backward compatibility with earlier Chebfun versions.
The project must be led by a team with a vision.

Caricature of what the young developers want:
Speed, functionality, power! Up-to-date user and developer environments! Functionality is more important than concepts. Backward compatibility may be too restrictive. MATLAB is a dinosaur. Everything must be online and democratic.
Leadership and vision aren’t scalable.

Q20. What’s our management structure?

At present, our procedures are not spelled out.
The project is run by the Chebfun team, notionally a democracy, but some are more equal than others.

What should my personal role be?
I am 30 years more experienced than the others, and pay most of their salaries. How can/should this situation evolve?

What role should “the team” have as we attract outsiders?
This really troubles me. An open-source project needs open communication. Yet it would be crazy to ignore the fact that 8-10 of us have special expertise and are sharing a “Chebyshev hallway” at Oxford. And I believe in vision.

Please, now and for the rest of the week,
tell me your thoughts.

Comments and suggestions by ICMS participants, p. 1

Jon Borwein, Newcastle. I should make sure I have considered the question: what will happen if I am hit by a bus tomorrow?

Matthew Sherritt, Stanford. For our weekly meetings, we should try Google Hangout for video plus Mumble for audio. They've had good luck with this.

Gert-Martin Greuel, Kaiserslautern and zbMATH. A group at zbMATH (= Zentralblatt, creators of swMATH) organized a discussion on a unified way to cite software. No convergence yet, but a proposal may be on its way.

Eric Berberich, Saarbrücken. The CGAL group has faced very similar challenges. They have formally spelled out Rules, though it's hard to love them. The group is run by 12 insiders, but this is rather large; 7 would be better. One can find discussions of optimal committee size on the web.

Jon Borwein, Newcastle. Commercial projects are very different from academic ones, with plusses as well as minuses. For example, you can fire people! — and ensure that things don't hang around on the bug tracker.

Comments and suggestions by ICMS participants, p. 2

Martin von Gagern, Munich. His experience with Sage persuades him that one thing is more important than all others: having clear information at the web site about how to contribute. Big non-commercial projects inevitably have many loose strands. Sage has 1500 open issues on the tracker, 500 open bugs.

John Abbott, Genoa. The lack of credit one gets for software is catastrophic for many careers.

Jon Hauenstein, Notre Dame. It's true, however, that a good software project can focus one's research and publications marvelously. So software can be decidedly positive in a career.

Wolfgang Windsteiger, Linz. Like Chebfun, the Theorema project has just been completely rewritten. Like Chebfun also, it is an open-source project written in a commercial language (Mathematica). Various people told him this would rule out a GPL license, but as far as he can determine, GPL is still ok.

Michael Kohlhase, Jacobs U. Clarity and accessibility aren't so important. If the software is exciting, people will do what it takes to get involved.

Comments and suggestions by ICMS participants, p. 3

Andrew Sommese, Notre Dame. Bertini, 150K lines of code, is about to be completely rewritten and move from C to C++. They have a clearly defined core team of 5 people (Sommese, Wampler, Bates, Hauenstein, Brake).

Martin von Gagern, Munich. Though anybody can issue a pull request, there must be a core group that makes final decisions about what to include. For documentation, the Sage group uses sphinx (like doxygen, but for Python).

Michael Kohlhase, Jacobs U. A software project can have multiple licenses, and indeed, a mature project normally has this: e.g. GPL for ordinary users, plus special software versions with commercial licenses for sales to companies. What matters in enabling such things is who has the *copyright.*

Various people. Chebfun looks already well advanced as an open-source project, and the web site is outstanding.