Mathematical Software and Me: A Very Personal Recollection

William Stein

December 2009

I find it difficult for me to write a history of Sage without writing a history of my personal involvement with mathematical software. I loved using calculators since my earliest memories. My father somehow got a mechanical electric adding machine for me when I was quite young, back in the 1970s, and I spent a great deal of time filling ribbons of paper with it. Then he got me an old TI scientific calculator from a yard sale, with a LED readout. At age 11, when I moved from Oregon to Texas, I bought a sliderule, which was pretty exciting for a while. Then in junior high I finally got a real scientific calculator, a little Casio, whose instruction manual I devoured. This was my first inroduction to trigonometry, statistics, and many other bits of computational mathematics. I was prepared though, since I had done elementary school on an unusual work-at-your-pace alternative curriculum, and had already worked through 10th grade level mathematics.

The first mathematical computer software I ever used was Mathematica, back in 1992 on a Windows 3.1 PC, when I was a 17-year old undergraduate at Northern Arizona University in Flagstaff, Arizona.

Naturally, my copy of Mathematica was pirated, since like many students I was extremely poor at the time. The only thing I found it useful for at the time was drawing 3d plots (just for fun), and even then it was frustrating, since there was no way to interactively change the viewpoint. I also obtained a copy of MATHCAD somehow, which I found much more useful than Mathematica. This is perhaps not surprising, because at the time I was a computer engineering undergraduate, so taking courses in Physics, Electrical Engineering, Programming, etc.. I was definitely *not* a mathematics major: I remember once finishing a multivariable calculus class and thinking "this is the last mathematics class I'll ever take"! I also used Maple for about an hour or two in a computer lab for a mathematics course I took, but found it very cumbersome; this was a token 1-hour introduction to math software that professors gave their students, perhaps to justify the grant that paid for the computer lab. At the time, I viewed Maple, Mathematica, and MATHCAD as software that didn't really go beyond scientific calculators in any exciting way.

My next encounter with mathematics software was in early 1994 when I became a mathematics major, after accidentally encountering an abstract algebra

book misfiled under computer science in a used bookstore, and being instantly mesmerized by ideas such as groups, rings, and fields. I was in another small computer lab and stumbled on printouts of the documentation (Northern Arizona University professor) Mike Falk had for Cayley, which was the predecessor to Magma. I was amazed and excited, since the capabilities and ideas in Cayley went so far beyond anything I had thought was even possible in mathematical software before. I'm pretty sure I never got to actually *use* Cayley though, since like now, the software was very expensive and hard to get. Instead, I just spent a lot of time reading the reference manual full of its beautiful examples of computing with groups, rings, and fields, which were the very objects that had enticed me into mathematics in the first place.

In fact, after that brief encounter with Cayley, I didn't touch mathematics software again, or even do any nontrivial computer programming for 3 years. This was because in 1994 I became intensely interested in theoretical mathematics, and spent most of my time during the next 3 years systematically doing exercises in mathematics books, ranging from basic books on linear algebra and combinatorics (which was big in Arizona) to Hartshorne's Algebraic Geometry.

In 1997, I was a graduate student at Berkeley, and didn't want to teach so much, so I landed a job (funded by the NSF VIGRE program) in the department for one year doing programming of curriculum materials for an undergraduate linear algebra course, along with fellow graduate student Tom Insel. Though I had been using Linux for years, I had never thought about free software, or that actual people could contribute to it. I thought of Linux as "Unix that I can install on my own computer", and back then one still paid for Linux by buying a CD, since downloading over a modem was way too slow. Tom and I spent a lot of time working on our project together, and he told me how he had written some software included in Slackware (a Linux distribution), so got free copies of the CD when new versions came out. Hey, anybody can contribute to Linux!

I also remember Tom complaining frequently about how we were forced to program in MATLAB for our project, and he mentioned many other alternatives that would have been better for what we were doing, including Java. We were doing GUI programming, with a tiny, tiny bit of actual mathematics thrown in, and MATLAB's handle-based system for writing graphical users interfaces was really painful. I had a lot of experience with C/C++/Windows 3.1 GUI programming from my computer science undergraduate days, and agreed that MATLAB was pretty awkward for what we were doing at the time. In retrospect, what we did was probably pointless, and perhaps never got used. However, the experience was extremely valuable for both of us, and I'm glad NSF funded it.

In the meantime, my Ph.D. thesis was going nowhere, despite nearly 3 years of graduate school. One day, I heard about a problem Ken Ribet was asking all the graduate students: "Is there a prime number p such that the Hecke algebra at level p is ramified at p?" It's the first research problem I can ever remember hearing that was almost certainly impossible to solve without using a computer. Fellow grad students Janos Csirik (now at D.E. Shaw), Matt Baker (now at Georgia Tech), and I searched and found one paper by Hijikata (?),

I think from the mid 1970s, which gave an algorithm that might allow one to answer the above question for specific p's. But to implement the algorithm, it was necessary to compute class numbers of a huge number of quadratic fields, and none of the mathematics software I had ever heard of until then could do this. Janos and Matt mentioned PARI, and I installed it on my computer. And indeed, it could quickly and easily compute class groups! PARI was also the first free mathematical software I encountered.

So I started coding up the algorithm (the Eichler-Selberg trace formula) in PARI. I had a lot of experience with C++, which is a real programming language with user defined data types, exception handling, templates, etc. I rememember in 1992 carefully reading several C++ book cover-to-cover, and I wrote a large amount of code (video games!) in C++ long, long ago. In contrast, PARI was an immediate shock. This was a language with no local variables, no real scoping, only a couple of builtin types, and for a while I thought (incorrectly) that entire function definitions always had to be on one line, since that was the case in example code I found. But the algebraic number theory algorithms implemented in the internal library were amazing, deep, and very fast. So I implemented the trace formula, and ran it to try to answer Ribet's question. The program did not work—basic consistency checks failed. It turned out that there was a major bug in the algorithm for computing class groups. In particular, the function `qfbclassno`, silently returned wrong answers.

You would think that `qfbclassno` would be fixed by now. But no. It's only frickin' 12 years later!! I just checked right now, and the documentation for PARI *still* says "Important warning. For $D < 0$, this function may give incorrect results when the class group has a low exponent (has many cyclic factors), because implementing Shanks's method in full generality slows it down immensely." This is buried in the documentation. The only change is that I think in 1997 the documentation said that the authors were "too lazy" to implement the full algorithm.[1]

So I worked around that problem, and was able to run the algorithm for all primes up to about 300, but didn't find any primes as in Ribet's question. A few weeks later, at the Arizona Winter School in March 1998 in Tucson, Arizona (`http://math.arizona.edu/~swc/aws/98/98GenlInfo.html`), I mentioned this to Joe Wetherell (who was another Berkeley grad student), while we were walking to lunch, and he told me he had written a program—also in PARI—for computing with modular symbols (following John Cremona's book), with which I might be able to push the computation a little further. He gave me a copy later, and I started playing around with it, and computed the discriminants of all of the Hecke algebras of prime level up to about 500.[2]

Again, I didn't find any examples, and I wrote to Ken Ribet to tell him. Then I hopped on a plane and flew to Cambridge, England, to visit Kevin Buzzard.

Once I got settled in Cambridge, I rechecked my calculations for some reason... and found an example for the prime $p = 389$! Somehow, I had just missed the example in my previous check. I was extremely excited as I wrote to Ken Ribet, with my first ever genuine contribution to research mathematics, which

appears in a big paper Ken published on the Manin-Mumford conjecture—I had shown that his new theorem definitely did not prove that conjecture for all modular curves. I was also hooked on computing modular forms, and started making tables. It was also the height of the mad cow disease scare in England, so I became a vegetarian.

I tried to push the PARI program that Joe Wetherell had given me to make bigger tables, but it very quickly ran out of steam. The main algorithms that the modular symbols algorithm relies on are all manner of linear algebra over the rational numbers, including computation of characteristic polynomials, and kernels of sparse and dense matrices, and also factorization of polynomials over the integers. Despite its first rate algebraic number theory capabilities, PARI was (and still is) terrible at linear algebra with really big matrices over the rational numbers. Also, I found the PARI programming language unbearably naive.

So I spent the summer of 1998 writing a much more general C++ program for computing with modular forms called HECKE. Here it is: `http://wstein.org/Tables/hecke-cpp.html`. If you grab the file `hecke-july99.gz` from that web page, and drop it on just about any Linux box, it should just work[3] Anyway, I spent all my time for many months writing HECKE. This program built on several other C++ math libraries, including LiDIA and NTL. It doesn't use PARI, because using PARI from C is... really weird, and LiDIA/NTL had equivalent functionality at the time. Much of the time I spent on HECKE involved (1) designing algorithms, generalizing work in Cremona's book, etc., and (2) implementing and optimizing algorithms for linear algebra over the rational numbers. For (2), Kevin Buzzard hooked me up with the same code Cremona used, which Cremona had got from some South American student at Cambridge named Luis (?). I then spent a large amount of time optimizing that code for my computations. The actual linear algebra algorithms in that code were very naive compared to the algorithms in Sage now, but they were better than anything available in any other software I was aware of, or in research papers at the time.

I was able to compute many fairly sophisticated tables using HECKE, and it soon became the canonical software for computing with modular forms, since it was the only software generally available for such computations. This is perhaps similar to how NAUTY was for a very long time the canonical software for computing graph automorphism groups. Naturally, I made HECKE freely available on my webpage. I remember once getting an email from Ken Ono, who has probably written over 100 papers on modular forms, many inspired by concrete examples. He had run across HECKE on my web page, installed it, and was totally blown away by the capabilities of HECKE, and how useful it would be for his research. A whole new world had opened up to him, and he told me he was promptly ordering a new fast computer specifically to run HECKE on. I was happy to help. Also, my thesis work was starting to go well, because the computations I was doing was suggesting interesting new (and do-able) mathematics, left and right.

Then, in 1999, David Kohel—who had been a Berkeley grad student with

me until December 1996—was visiting Berkeley from Sydney, Australia, and told me about implementing algorithms related to his thesis in Magma. I think this was the first I had ever heard of Magma, despite Magma having been around for several years. Magma was expensive and nearly impossible to get unless you knew the right person, since it was sold via informal channels. I think there was an install on the computers in Berkeley, but those computers were ancient vintage 1990-ish Sun workstations, so nobody would seriously try to use them. Anyway, David had implemented code for computing with rational quaternion algebras, and this was the only implementation of that algorithm in the world. Coincidentally, I had extended an old idea of Ribet to come up with a new algorithm for computing Tamagawa numbers of modular abelian varieties (see `http://wstein.org/papers/ants/` and `http://wstein.org/papers/compgrp/`). I really, really wanted to implement my algorithm, because it would allow me to compute all of the invariants in the Birch and Swinnerton-Dyer conjecture (except Sha) for most rank 0 modular abelian varieties, which would be a huge step forward. But my algorithm fundamentally relied on exactly the computations in rational quaternion algebras that David Kohel had implemented in Magma. And that was no small undertaking—it's a complicated algorithm, it takes somebody familiar with the relevant theory months to implement and optimize, and it builds on many other basic capabilities. I had a thesis to finish. David—who was then officially a Magma developer, was able to give me a copy of Magma for my own computer, which had his code in it. Combining all this with HECKE, and copying and pasting, I was the first person ever to systematically compute Tamagawa numbers of general modular abelian varieties (at primes of multiplicative reduction).

So in 1999 David Kohel put me in a situation where I was fundamentally dependent on a closed source non-free program in order to continue my own research. Ironically, during the same afternoon in my apartment in Berkeley, he mentioned the GNU Public License (GPL), and suggested I released HECKE under the GPL. Before that moment, I had never even *heard* of the GPL. I did as he suggested, but had no idea what it meant really. That was perhaps fortunate, since the dependencies of HECKE are NTL and LIDIA, and the NTL license is GPL, but the LIDIA license is GPL incompatible, so technically I guess HECKE can't be distributed. Incidentally, LIDIA is *still* licensed under a GPL-incompatible license, despite many emails I've received suggesting the license would change to GPL—licenses don't change easily, due to having to get agreement from all copyright owners.

I obviously had to actually use Magma at some level in order to do these computations. Despite having a lot of experience with programming, I initially found the Magma language and system extremely hard to learn or do anything with. It was certainly much harder to use initially than PARI. On the one hand, Magma is a massive system, with thousands of commands and thousands of pages of reference manual documenation, but on the other hand there is very little in the way of "introspection", i.e., given an object A, it is hard to get context sensitive help about A. However, one thing was clear: Magma was dramatically better at *dense* linear algebra over the rational numbers than

HECKE. It was a whole different world. We're talking jaw dropping speed. In fact, Magma in the late 1990s on an old computer, was far faster at large linear algebra over the rationals than Maple or Mathematica is *today* on the latest hardware. I would soon find out the reason.

Allan Steel is an enthusiastic Australian, who was an undergraduate at University of Sydney in the early 1990s and fell in love with the Magma project. I guess David Kohel told John Cannon about me in 1999, and Allan happen to be visiting Berkeley for a conference, so Allan and I met. We ended up talking a huge amount over 3 days. Allan answered my every question about the language and the system—usually with an answer about how *he* had implemented it that way for a certain reason. So within a few days I knew Magma well enough to be quite productive in it. Also, Allan gave me some hints about why linear algebra was so fast: together with polynomial and integer arithmetic, asymptotically fast linear algebra was one of his main interests. He explained an exciting array of algorithms, many of which he had developed or—more importantly—made practical. There were dozens of tricks that I had never heard of, such as rational reconstruction, multimodular algorithms, p-adic algorithms, etc., which were far beyond what I had done with HECKE, or seen in any other software. And they meant that it would be possible to push my modular forms computations much farther. All I had to do was rewrite HECKE in Magma.

It was my last year of graduate school at Berkeley, and I probably should have been writing my thesis, but instead John Cannon flew me down to Sydney, Australia, to work with the Magma group for a month and do a complete new implementation of all the algorithms in HECKE on top of Magma. I shared an office with Claus Fieker, a German who has implemented a large amount of the algebraic number theory in Magma, among other things. As I started doing this, I had some serious concerns, including: Magma did not allow users to define their own types (or classes), there is no exception handling, there is no "eval statement", no way for users to write compiled code, Magma is closed source, and Magma is not free. I raised all of these concerns with Cannon and others, and was assured at the time that they would all be addressed really soon, except the free part. Regarding free, they said that I could give copies of Magma to whoever I wanted, so long as I checked with John Cannon first.

I don't remember why exactly, but I remember once during that month in 1999 going on a walk through the park near the U Sydney campus near a pond of ducks, sitting on a bench, and realizing that I was making a huge sacrifice of my freedom as a researcher by going down this path. Magma was not open source—John Cannon had absolute control over the system. Magma was not free. And as a language, Magma was significantly behind C++. It didn't even have a sensible notion of scope, and one added new data types by entering entries in big tables, then recompiling the kernel. I asked Cannon why it was so far behind, and he explained that the grants he was able to secure simply wouldn't pay for language design. The people who supported Magma with funding (mainly granting agencies) would only support implementing and optimizing mathematical algorithms, and the license fees only paid enough to support "maintenance", which mainly meant the person who collected the license fees and distributed

Magma via ftp. Ten years later the Magma language has hardly improved at all. They finally have exception handling, but still no user defined types, etc., etc. The library of functionality implemented in Magma is huge though; the issues with the language didn't stop people from implementing many exciting algorithms, which have supported huge amounts of research in number theory and other areas.

I sat down on that park bench, and realized what a dangerous path I was taking in giving up so much freedom so early in my career. I resolved at that moment not to do it. At that moment I started designing what would eventually become Sage. I started thinking about the language I would implement (I had taken a course in writing interpreters when I was a student), about implementing all the linear algebra algorithms Allan had hinted at (but given *no* details), etc. I then realized that if I did this, I would have to do it by myself, since almost everybody I knew used Magma, and would consider my plan too difficult and pointless. I wouldn't get to do number theory for years. My spirit broke. And Cannon told me that many of my issues with Magma would be addressed within a year.

I spent the next 5 years writing and using Magma. I gave dozens and dozens of talks all over, and convinced anybody who wanted to do computations with modular forms that Magma was the way to go. I gave away free copies of Magma (with John Cannon's official blessing), taught undergraduate courses using Magma, and was generally very productive. Also in 2003, I had a real job and money, so I started forming this philosophy of software, where I would judge what software I used purely based on capability and functionality, and not on price or openness. I started using Microsoft Windows fulltime, since it best supported my PDA and had the widest range of software. I bought some $500 (with educational discount) suite of Adobe software for video editing, photos, vector graphics, etc. And of course I used Magma. I was a well paid academic at a well-endowed university (Harvard), and wanted the best that money could buy.

In 2002, William Randolph Hearst III also donated money to Harvard to buy me a cluster computers, and by 2003, I wanted to easily script running lots of computations in parallel. Since Magma didn't have any parallel capabilities, I stumbled on some language called "Python" (Version 2.3), which looked a lot like Magma, but was designed for general purpose scripting. I started using it to run many computations in parallel on that cluster. It was the best tool I could find for the job.

I also started working much harder on making the number theory data I was computing with Magma available online, and naturally I turned to Python. Dimitar Jetchev (a Harvard undergrad) and I wrote Python code to make the data easily queryable via a web interface, and also wrote code that made it so one could do computations in Magma (and PARI in some cases) over the web. One incarnation of this is hosted on the Magma website: `http://magma.maths.usyd.edu.au/calc/`.

As I learned about Python, a funny thing happened. I had by this point developed a large list of issues with Magma. For example, the documentation

and examples in the Magma reference manual aren't automatically tested to ensure they give the claimed output, and they often get out of sync with the actual code. Python is in many ways similar to Magma – the language itself feels somewhat similar, and it has the same "batteries included" philosophy. The surprising thing was that Python had solved the dozens of major problems I had with Magma! I was excited about this in 2003, and so during my next (and last) trip to work with the Magma group in Sydney, and started trying to incorporate these solutions into my new Magma code, and to explain what I had learned to John and others. Right after I returned from Sydney, I recall excitedly explaining all of this to David Goldschmidt at a reception at an AMS meeting.

I was quite surprised when a month or two later, in early 2004, John Cannon really soundly rejected my ideas and even had one of his employees rewrite my new modular abelian varieties code to get rid of them. In retrospect, now that I run a large software development project, I can understand why he didn't get what I was doing. But I was really suprised then. Second, I went to a big Magma conference at IHP in Paris, where Manjul Bharghava (a young professor at Princeton), me, and many other people gave some series of talks. I recall listening to Manjul's talk as he described a research problem he was working on, and during his talk he explained that the whole thrust of his research was seriously stymmied because Magma is closed source. He needed to adapt some relatively minor part of the some algorithm in Magma related to quadratic forms, and simply couldn't due to it not being in the interpreter level of Magma. This just didn't seem right.

I also had lunch with John Cannon during that workshop, where he explained some big plans he had to edit a huge sequence of volumes about the mathematics behind algorithms implemented in Magma. I suggested that it would be nice if these were freely available, and he did not think that would be possible. He made good on his idea, and I actually published a paper in the first such volume: `http://wstein.org/papers/bsdmagma/`. I wrote that paper in Windows using Microsoft Word (and converted to LaTeX using WinEdt, a non-free LaTeX frontend, only at the very end)!

Finally, at that IHP workshop I learned two other things. First, I learned that a workshop is an incredibly efficient way to develop mathematical software, far more efficient than the Magma model of hiring people for months at a time and flying them to Sydney, and certainly vastly more efficient than what Maple, Mathematica, and Matlab do, which costs literally costs hundreds of millions of dollars per year. Second, I recall overhearing conversations about the Magma language during tea breaks by some locals who were interested in the workshop, but had not drank the Magma koolaid, and in these conversations it became clear that I wasn't the only person that found the Magma language to be deficient.

I started reading slashdot in early 2004, mainly for the interesting tech news, and the comments kept mentioning "open source", which was honestly something I had paid almost no attention to until then. Intrigued I decided to look around and see how open source mathematics software had done since I had

abandoned it in 1999. NTL was no longer being actively developed, the LiDIA project was nearly dead, PARI hadn't changed much (except to break all my old code), but with some more algorithms for relative number fields. So in five years, the situation with the open source number theory software environment had got worse. I realized that I was probably partly to blame, having tried to convince every number theorist I knew to use Magma, and often given them free access to ensure they could. I had helped hook a generation.

About this time I was also writing an elementary number theory book (which eventually became this book: `http://wstein.org/ent/`). I had planned to have a chapter about number theory using each of Mathematica, PARI, Magma, and Maple. I had the first three programs, but didn't have access to Maple. Somebody suggested that being a faculty member and writing a book should be a good argument for Maple to send me a free copy, so I contacted them. A person from Maplesoft called me back, and explained that though I was writing a book with a chapter on Maple, they could not give me a free copy. However, they could give me the special academic discount of 500 dollars. I asked if he could do better, and he called me back the next day and said: "If you can get 4 of your colleagues to also buy Maple at 250 dollars/each then I can sell you Maple for 250 dollars." I was offended, so I "obtained" a "trial" copy, and started writing my chapter anyways. I started trying all the same things as I had easily done in PARI and Magma, e.g., checking primality of numbers, etc. I was totally surprised to find that Maple was terrible, being massively slower than Pari, Magma or Mathematica for most elementary number theory computations relevant to my book. (Maplesoft was bought by some Japanese company a few months ago, by the way.)

I also installed Linux in a virtual machine (under Windows), to see what all the fuss was about, and found I started using Linux all the time instead of Windows, because the software was better (even Magma runs much better under Linux than Windows). I deleted Windows and installed Linux. I was also starting to definitively realize that my huge list of problems with Magma would never, ever get resolved, and was getting increasingly frustrated by these problems because Python didn't have them.

I started talking a lot with Thomas Barnet-Lamb about a crazy idea to create a new open source math software system with readable implementations of algorithms, and nothing hidden in some stupid proprietary layer. Thomas was then a first year Harvard grad student who had won some international computer programming competition, so I figured he would enjoy talking about software. I also talked a lot with Dylan Thurston about this crazy idea; Dylan had started grad school at the same time as me at Berkeley, graduated the same time, and had the same first two jobs as me, was also an Assistant Professor. Both Thomas and Dylan gave me many ideas for programming languages to consider, including OCaml (which Thomas liked), Haskell (which Dylan was a huge fan of), etc. After having used Magma for years, with its highly optimized algorithms, I desperately needed a fast language. But I also wanted a language that was easy to read, and that mathematicians could pick up without too much trouble, since I wanted people like Manjul to someday use this system and not

have their research cut off. And I knew from experience that unreadable source code is no better than closed source.

I'm not going to go into negatives of any languages. Though I used Python a lot, for a long time I didn't consider it seriously at all for this crazy project, since I tried implementing some basic arithmetic algorithms in Python and found that they were vastly too slow to compete with Magma (or C). I had also tried quite hard to use SWIG to make C++ available in Python, but SWIG is extremely frustring, and has horrible performance (due to multiple layers of wrapping), at least compared to what Magma could do.

In October 2004, I was flying back from Europe (the Paris Magma conference) and started reading the Python/C API reference manual straight through. I realized that Python is far, far more than just an interpreter. It is a C library that implements everything you need, and has a well defined and well documented API. I did some sample benchmarks on the plane, and found not surprisingly that I could write code as extensions to Python that was just as fast as anything one could write for Magma by modifying the Magma kernel, since under the hood, both were written in C. Also, on the flight, I realized that because the Python/C interface uses reference counting, it would be vastly easier to write the C extensions I would need using some sort of language I would design. I got home and somehow stumbled onto Pyrex, which was exactly what I was *planning* to write. I tried it out, did benchmarks, and realized that I had a winner.

With Pyrex and Python, I could implement algorithms and make them as fast as anything in Magma, assuming I could figure out the right algorithm. Moreover, the dozens of issues I had with Magma, many of which were simply a function of them not having the resources to do language development, were already solved in Python. And Python would continue to move forward with no work from me. It was mid-2004 and because of Python, the overall software ecosystem was much better than in 1999, despite open source number theory software having not moved forward much.

I started going to (and sometimes hosting) the Boston Python user group meetings, which was quite large, and gave me much useful feedback. And I decided it was time to move past my test and prototype stage and get to work. My plan, as I had explained it to Thomas, was to create a complete new system from the ground up using Python + Pyrex. All the code would have an easy to read Python implementation that was well documented, in some cases there would be a much faster Pyrex implementation of the same code, etc. With my naive plan in hand, I sat down with the main elliptic curves file of the PARI source code, and started to translate.

I think I made it through one function. Where some might have doggedly persisted for years with such an approach, I quickly ran out of patience. In fact, when it comes to software and programming I can be extremely impatient. I realized that my entire plan was insane, and would take too long. I had discussions with Thomas, Dylan, and others, and everybody I knew who was seriously into number theory computation was using Magma, so I realized that I was going to have to do this entire project myself. So I realized translating

was doomed. Somehow, even with all my experience, I had massively underestimated the complexity of the algorithmic edifice that is any serious mathematical software system.

I read the PARI C API reference, and used Pyrex to write a wrapper so that I could call some basic PARI functions from Python. I implemented basic rational and integer types using Pyrex and GMP, and the performance was reasonable. One day, I was using Matplotlib (a Python library) to draw some plots for Barry Mazur that involved explicit computation with the incomplete Gamma function, and was frustrated because neither PARI nor Magma had an implementation of this special function at the time. Harvard had a Mathematica site license, so I had a copy of Mathematica, and I wrote code using the pexpect Python library to hold open a single Mathematica session and use it to compute the incomplete Gamma function. Problem solved. This was when the interfaces between Sage and other mathematics software systems was born.

In January 2005, I was at the AMS meeting in Atlanta, Georgia, hacking on my code, and David Joyner walked up to me and asked what I was doing. Until then, I had not shown my Python/Pyrex math software project to people. There were a few reasons, including feeling sure that it was massively too difficult to pull off, that working on something like this would seriously piss off John Cannon, etc. But feeling brave, I showed David what I was doing and I was surprised that he found it interesting. I promised to post a copy online, which he could download.

David Joyner is the first to admit that it's a good idea to make software easy for him! So I had to make it easy for him to download and install my program, which I called Manin at the time (after one of my favorite mathematicians). My target audience wasn't "Debian"; it wasn't "Python programmers"; it wasn't elite hackers—it was David Joyner. I had to make this program trivial for him to install, work with, develop, etc. I thought about how it had literally taken me huge amounts of time just to build Python, GMP, PARI, etc. all from source in a directory for development, and realized that there was no way in hell David would get anywhere on Manin if I told him that his first step was to figure out how to build all those programs, then get back to me. So I setup something that would do it all automatically in a self contained way. He tried it, it "just worked", and he got really excited and started writing code for Manin. David is a coding theorist, and wanted group theory and coding theory functionality in Manin, but didn't want to write it all himself from scratch, so he asked in email about making Manin and GAP talk to each other somehow. I showed him my pexpect code for controlling Mathematica, and he adapted it to create a GAP interface.

David also works at the US Naval Academy where he evidently teaches a lot of Calculus and Differential Equations courses. He was having so much fun with Manin, he asked about adding something to do symbolic calculus to Manin. This was 2005, and I personally had never used any symbolic calculus software aside from Mathematica and Maple 12 years earlier, since I viewed computational symbolic calculus as pretty much irrelevant for most computational number theory, and the Calculus I had taught never required a computer since computers

weren't allowed on exams. (I now think no computational technique should be a priori viewed as irrelevant to research in number theory!) So I asked David about the available open source options, and he said they were Axiom and Maxima, neither of which I had ever heard of. I can't remember how we chose Maxima instead of Axiom, but it was some combination of Maxima being easier to build, easier to understand, and having about 1000 times as many users. In any case, like with PARI, GMP, and Python, I added Maxima and GAP to the Manin distribution. I also changed the name from Manin to SAGE = Software for Algebra and Geometry Experimentation.

David also convinced me Sage needed commutative algebra. At first, he talked to people and tried to implement everything from scratch, but even the resulting arithmetic was dreadfully slow. Groebner basis would be a nightmare waiting on the horizon. We were both impatient, so we decided to try to find an open source program already out here, and just use it. There were two choices: Macaulay 2 and Singular, which had a lot of overlap in functionality. For what we wanted—basic commutative algebra—they both did all we needed. Singular built from source easily in a few minutes on every computer I cared about. Macaulay 2 was ridiculously hard to build and took a long time. I think based mostly on that, we chose Singular. Also, it was encouraging that Singular had a relatively large development team, and did better in some benchmarks I tried.

At the same time as all this, I was traveling a lot and interviewing for tons of tenure/tenure track jobs all over the place. I got some job offers with tenure, and suddenly had the crazy idea that if I worked fulltime on SAGE for a year two, my career could not be destroyed. This really encouraged me. My number theory research slowed a lot, and I spent all my extra time on SAGE for a while.

Remember David Kohel, who six years ago in 1999 first introduced me to Magma? It turns out that like me he spent years and years writing a large library of software on top of Magma for number theory and cryptography research. However, at some point he had a fairly serious falling out with the Magma group, whose details I will omit. Suffice to say, like me he was motivated to at least look for other options. He started building and using Sage, and started doing huge amounts of work on Sage as well, e.g., introducing morphisms and categories systematically into Sage, and implementing tons of code related to elliptic curves, algebraic varieties, etc. David Kohel was a Biologist as an undergraduate and has an amazing eye for general structure. He also had many technical issues with Magma, which were mostly different than mine. For example, he felt that the Magma developers had made a mistake with the design of morphisms, and he didn't want Sage to make the same mistakes. And he was right to worry, since for things I didn't care too much about, I would usually just copy Magma... or as David would say, "copy Magma's mistakes".

I moved to San Diego and Joe Wetherell who first introduced me to modular symbols in 1997 also got involved in Sage development, though mostly from the conversation point of view. Joe had long ago quit grad school to start a software company in the early 1990s, then retired from that and went back to grad school, so he had a fairly mature perspective on software development, and he knew a huge amount about number theory and optimized algorithms. So 2005 was a

long, long year in which David Kohel in Australia, David Joyner in Maryland, and me in San Diego, wrote code.

At the end of 2005, the three of us had written a ton of code, integrated numerous components togethers, and finally had something. On December 6, Jaap Spies mentioned Sage on the sci.math.symbolic newsgroup, in response to which some guy named Richard Fateman declared Sage a curiosity and made some unencouraging assertions about the way the world works (regarding users, funding, etc.):

`http://mathforum.org/kb/message.jspa?messageID=4132045&tstart=0`

I was certain deep down inside that Sage would fail anyways, that what we wanted to do with Sage was totally impossible, so Fateman's comments couldn't discourage me further. I just didn't care that Sage was doomed. I couldn't help pushing further.

John Cannon found out about Sage, maybe as a result of the postings on sci.math.symbolic, and right before Christmas in 2005 he sent me this email:

```
-----------------------------------------------------------------
Date: Mon, 19 Dec 2005 16:54:09 -0800
From: "John Cannon" <john@maths.usyd.edu.au>
Subject: Magma calculator
William,

This is to formally advise you that your permission to run a
general-purpose calculator based on Magma ends on Dec 31,
2005. This was originally set up at your request so students
in your courses at Harvard could have easy access to Magma.

Please confirm receipt of this letter.
Wishing you a happy Christmas,
John
-----------------------------------------------------------------
```

This single email seriously scared me. Though I was working on Sage very hard for nearly a year at this point, I honestly didn't then expect Sage to really be able to replace Magma for me. Magma was the commercially funded result of fulltime work over decades (really starting in 1973 with the first version of Cayley). The amount of work to get from what I had with Sage in December 2005 to what I had with Magma in December 2000, was still absolutely momentous. I didn't even know if it could be done by a single human being. Moreover, as far as I could tell many of the critical linear algebras algorithms I needed (to make the difference between a calculation taking a minute or a year) existed only in the secret kernel of Magma and Allan Steel's head, and they were going to stay locked there forever as far as I could tell. For example, in June 2004 (before Sage existed), Allan and I were together at the ANTS VI conference. I started asking Allan to explain some of the algorithms, and he would explain things to a point, but not nearly enough to do an actual implementation. And he gave me this look, like he knew I was trying to get something out of him.

Isn't it weird that mathematics can be done that way? In 2004, almost everybody in the world doing serious computations with elliptic curves, modular forms, etc., were using Magma. Magma was the industry standard, Magma had won for the forseable future. David Kohel and I were a big reason why. And yet what kind of mathematics is it, when much of my work fundamentally depends on a bunch of secret algorithms? That's just insane. Moreover, it turns out that these algorithms I'm alluding to are really beautiful (and they are now standard and in some cases better than what's in Magma, in my opinion, thanks to great work of people like Giesbrecht, Perent, Kaltofen, Storjohann, Saunders, Albrecht, etc.).

Anyway, John Cannon's email above seriously scared me. I wasn't in any way confident that Sage would ever replace Magma for my work and teaching, and I had big plans involving interactive mathematical web pages. These plans were temporarily on hold as I was drawn into Sage. But there were still there. What John did with that email is tell me, in no uncertain terms, that if I was going to create those interactive mathematical web pages, they couldn't depend on Magma. "This is to formally advise you that your permission to run a general-purpose calculator based on Magma ends." I was scared. It was also the first time I saw just how much power John Cannon had over my life and over my dreams. That email was sent on a whim. I hadn't got any official permission to run that Magma calculator for a specific amount of time (just open ended permission). What John made crystal clear to me was that he could destroy my entire longterm plans on a whim. I looked around for other options, and there just weren't any. Sage *had* to succeed. But still I was certain that it just wasn't humanly possible, given that I had to do almost all the work, with limited funding and time.

At this time I had an NSF grant, and also startup money at UCSD, hence I could rebudget some of my NSF grant. David Joyner suggested that we run a "Sage Days", which I guess was named after the East Coast Computer Algebra Days (ECCAD). David and I organized the first one, and David did an amazing job inviting a great cast of speakers, including Steve Linton (of GAP), Sebastian Pauli (of KANT), etc., and Joe Buhler who also lives in San Diego made sure we scheduled the workshop so that a lot of people would show up. We had Sage Days in early February, and I released Sage version 1.0 during my talk, which started the workshop. The talks went well, people were extremely enthusiastic about Sage, the coding sprints were intense: the first version of Sagetex was written, and the current sophisticated GAP interface was written then by Steven Linton, Kiran Kedlaya and David Roe wrote that Macaulay 2 interface, and I had the first spark of insight about how to create the Sage Notebook, after watching a talk by Robert Kern about some failed attempt to give IPython a notebook interface. That was the first time I realized a notebook style interface had some value. And Gonzalo Tornaria got us to finally start using revision control for our source (!), which meant way more people could easily contribute. (I had used revision control before with Magma, but with Sage I had been just taking snapshots regularly.)

As a direct result of Sage Days 1, development picked up. Then I moved

to University of Washington (Seattle) two months later. Somehow, during the summer of 2006 I was invited by MSRI to run a 2-week summer workshop on modular forms for about 40 graduate students. I invited David Kohel and a few other speakers. At this point, honestly some aspects of Sage sucked. I had written a massive amount of code, for really wide range of things. Power series, fraction fields, number fields, modular symbols, linear algebra, etc. I probably should have just used Magma for the workshop, and indeed at least one speaker entirely did. But I didn't... and this was a turning point. Some of the students, such as David Harvey (grad student at Harvard), Robert Bradshaw (Seattle), Craig Citro, and many others, became highly interested in fixing the numerous flaws they ran into with Sage. After the talks, we had huge all night coding sprints in the dorm lobby. Students constantly asked me questions about how to do things in Sage, and my answer was usually: "It's easy. Implement it and send me a patch!" They made a t-shirt for the conference with this quote on it.

Next we started planning Sage Days 2 in Seattle in late 2006. This second Sage Days was also well attended and resulted in major fundamental development directions. For example, David Harvey led a charge to redesign the coercion model and David Roe got obsessed with implementing every model imaginable of the p-adics (this still isn't really done over 3 years later). Sebastian Pauli gave a talk in which he explained what anybody who takes an algebraic number theory course knows (or pays attention to Weil), which is that there is a number field and function field analogy and that all the algorithms carry over. Guess what—Magma has a sophisticated implementation of all the relevant algorithms in the function field case, due to work of Florian Hess that built on work of the KANT group and others, and PARI has absolutely nothing for algebraic number theory over functions fields. Sebastian explained that in fact Magma is the only program in the world that provides both sides of this analogy, I think hoping that we would do something about this problem. (Now it is 2009, and still nothing at all has happened—Magma is the only program in the world for the function field half of algebraic number theory. Gees.)

We had about 13 talks on the first day of Sage Days 2. At the end of the day, David Savitt (a student of Richard Taylor, and now a professor in Arizona) looked at me and declared me insane.

After Sage Days 2, I spent over 2 very, very painful months implementing David Harvey's proposed coercion model. I learned (or rather, remembered) how difficult certain types of changes to a large interrelated library can be. Also, a student from San Diego—Alex Clemesha—followed me to Seattle, and I paid him fulltime to work on Sage using my startup money. He implemented 2d graphics for mathematicians (instead of scientists, which is what matplotlib provides), and he also helped a lot with the first version of the Sage Notebook. In fact, he was a big Mathematica user before he started using Sage, and he really missed the Mathematica Notebook, so he wanted something similar in Sage. When he used Mathematica, he had a job programming webpages using webMathematica, so really wanted something that combined the notebook idea with a webpage. We came up with various ideas. Then I hired an undergraduate, Tom Boothby, who had just quit a 6-year career in web programming to go back

to college. Together, the three of us figured out how to write AJAX applications, and the first version of the notebook was born.

It was a controversial decision at the time to write a webapp instead of a traditional local GUI application. There were many reasons we made this choice, but for me it was mainly because (1) I had written some serious local GUI applications before and knew that they are not easy to write, not portable, and hard to build from source, and (2) I had tried out wxMaxima (the local GUI maxima interface) and was just totally shocked to see how bad it was, due to having to reinvent the wheel—they would have to implement font dialogs, tabs, everything from scratch; in contrast, with a web application much of that comes for free. So my motivation was entirely to create a desktop application quickly. That it happen to later make it possible for people to collaborate easily, use a Sage notebook over the web, etc., is a nice bonus. And, it's clear by now that web applications (like Facebook, Gmail, etc.) are extremely popular now, and will only get way more popular in the future.

In 2007, the Sage project started really picking up steam. Bobby Moretti, another UW undergraduate, got obsessed with making it possible to actually do symbolic calculus in Sage itself. Until Bobby's code was added to Sage in mid 2007, absolutely all symbolic Calculus in Sage had to be done via explicit unnatural calls to Maxima, and involved embarassing and confusing conversions. Bobby, me and others spent a lot of work designing Sage's first symbolic calculus interface, and Bobby wrote a pure Python "proof of concept" reference implementation that used Maxima via a pseudotty behind the scenes for everything. This took him months, and probably had a negative impact on all other aspects of his life. But he heroically pulled it off. It went into Sage and changed things dramatically—suddenly, Sage could actually be used for some undergraduate courses. This increased interest in Sage dramatically.

A few months later, in November 2007, Sage was nominated for the Trophees du Libre, and Martin Albrecht presented Sage at the meeting for finalists. We won first place in the Scientific Software category. This resulted in a blitz of publicity (e.g., several slashdot articles, and articles in papers around the world in many languages), and greatly increased the number of Sage downloads.

Around this time we also have Sage Days 5 at the Clay Mathematics Institute, and Craig Citro convinces us to switch to a 100% peer review and 100% doctest policy on all new Sage code. Also, I hire Michael Abshoff to do release management for one year, which temporarily frees me up to work more on coding, grant proposals, and my own research.

There is of course much, much more to this story. But it's too recent, and sometimes a story shouldn't be told until enough time has elapsed.

1 Endnotes

Notes

[1]In the 1990s the function was `classno` instead at that time. I found in `tutorial.tex` from `http://pari.math.u-bordeaux.fr/pub/pari/unix/OLD/pari-1.39a.tar.gz`: `Type classno(−10007). GP tells us that the result is 77. However, you may have noticed in the explanation above that the result is only usually correct. This is because the implementers of the algorithm have been lazy and have not put the complete Shanks algorithm in PARI, causing it to fail in certain very rare cases. In practice, it is almost always correct, and the much more powerful buchimag program, which` *is* `complete, can give confirmation.`

[2]That program lives on here, in case you're interested: `http://wstein.org/Tables/heckegp.html`. And it still works (here with GP 2.4.3)!!!

```
? M37 = modsym(37,+1);
1. Generating M-symbols (0 ms)
2. Hashing M-symbols (2 ms)
3. Quotienting out by relations (3 ms)
4. Computing the kernel of delta (0 ms)
Total time..................... (5 ms)
? factor(charpoly(T(2,M37)))
%27 =
```

[3]Here is Hecke which I just tried on a Core 2:

```
tarte% ./hecke-july99
HECKE Version 0.4, Copyright (C) 1999 William A. Stein
HECKE comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; read the included COPYING file for details.

   HECKE: Modular Forms Calculator Version 0.4 (July 9, 1999)

   William Stein
Send bug reports and suggestions to was@math.berkeley.edu.
Type ? for help.
```