Open source computer systems in mathematics education

Alasdair McAndrew
Alasdair.McAndrew@vu.edu.au
College of Engineering and Science
Victoria University
PO Box 14428 Melbourne, Victoria
Australia

Abstract

It is axiomatic in mathematics research that all steps of an argument or proof are open to scrutiny. However, a proof based even in part on commercial software is hard to assess, because the source code—and sometimes even the algorithm used - may not be made available. There is the further problem that a reader of the proof may not be able to verify the author's work unless the reader has access to the same software.

For this reason open-source software systems have always enjoyed some use by mathematicians, but only recently have systems of sufficient power and depth become available which can compete with—and in some cases even surpass—commercial systems.

Mathematicians and mathematics educators may gravitate to commercial systems partly because such systems are better marketed, but also in the view that they may enjoy some level of support. But this comes at the cost of initial purchase, plus annual licensing fees. The current state of tertiary funding in much of the world means that for all but the very top tier of universities, the expense of such systems is harder to justify.

For educators, a problem is making the system available to students: it is known that students get the most use from a system when they have unrestricted access to it: at home as well as at their institution. Again, the use of an open-source system makes it trivial to provide access.

This article aims to introduce several very powerful and mature systems: the computer algebra systems Sage, Maxima and Axiom; the numerical systems Octave and Scilab; and the assessment system WeBWorK. We will briefly describe these systems: their history, current status, usage, and comparison with commercial systems. We will also indicate ways in which anybody can be involved in their development. The author will describe his own experiences in using these software systems, and his students' attitudes to them.

There will be no assumption of any particular mathematics beyond undergraduate material, so as far as possible mathematical topics discussed will be those with which an educational audience may be expected to be familiar.

1 Introduction: a personal journey

I have been using mathematical software tools for several decades now, starting with handheld calculators as a student, and moving into computer algebra systems, dynamic geometry software, numerical systems and online assessment systems. Table 1 lists most of the software I have used, either for myself, or with my students:

Computer Algebra Systems	Dynamic Geometry Software	Numeric Software	Assessment Systems	CAS Calculators
Axiom	GeoGebra	GNU Octave	MAA WeBWorK	Casio ClassPad
Derive		Matlab®	Pearson MyMathLab®	HP Prime
Maple®		Scilab	Wiley Assist®	TI-nspire CAS
Mathematica®		Julia		
Maxima		C++		
MuPAD®		Python		
Pari/GP				
Sage				

Table 1: Software used by the author

Some of these systems are commercial, some are open source, others have another licence. For the purposes of this article, we make the following distinctions:

Commercial Software (or *closed-source software*) is software distributed by the developers in executable form only, and for which access to a full and unrestricted version requires the user to pay.

Open source software is distributed free of charge, and with the complete source code, which the user can modify at will. The most extreme open source licence is "GNU CopyLeft" which ensures that not only is the original software free, but so will be any further modifications[1].

Free proprietary software Some software (such as Geogebra) is free "for non-commercial use", but requires payment for commercial use.

There are many variations and shades of gray in the flexibility and openness of software licences, but for the purposes of this paper I shall consider not only GNU Copylefted software, but free proprietary software: basically any software which is available for free with its source code, and which has a licence which allows the user to use a full and unrestricted version. The term FOSS for Free and Open Source Software is much used, although some writers [11] claim that there is a philosophical difference between "free" which is supposed to be completely unrestricted, and "open source" which may or may not have some restrictions. As a matter of simple pragmatism, I am not going to be drawn into this debate.

[1]See `http://www.gnu.org/licenses/licenses.en.html` for the various different licences and their meanings

I started using Maple in one of its earliest versions, and also Mathematica not long after, simply because I had access to them. An early experiment at my university, in which I was a participant, but not the leader, was to use Mathematica in our first year classes. This failed simply because the lead experimenter was too enthusiastic, and expected too much not only of the students, but of the other staff. Some time later we started with Derive, with carefully scaffolded exercises and lab sheets, and which the students seemed to enjoy. We reported on this [8] at the time. Some time later we moved to Maple with a site licence, which we and the students mostly enjoyed. I was discovering then a major problem: my enthusiasm tends to have me ask too much of the students, so that instead of being encouraged to explore they become overwhelmed by all the new commands and their parameters.

Later the site licence became too expensive; this was a time of low student numbers in mathematics, so we moved to a couple of lab licences and individual licences, and later, when the University was going through one of its many organizational clean-ups, we lost those, too.

A third year subject in cryptography which I'd taught initially with Maple required new software, so I spent a year experimenting with Maxima and Axiom. Curiously, in spite of Axiom (under MS Windows) having only a text-based interface, the students didn't seem to mind. As Sage matured I started moving towards it, and for the last few years of this subject's existence we used it exclusively.

We are now using CAS Calculators (TI-nspire CAS and Casio ClassPad) as in my home state of Victoria, Australia, the use of such calculators is mandated in advanced secondary mathematics, so students arrive at the University already with some familiarity with their use. We are using them in both first year subjects and also in a third year subject in numerical methods. Although not free, such calculators are remarkably powerful, and moreover can travel with the students anywhere.

At the same time, we have been experimenting with online assessment systems, starting with Pearson MyMathLab, then Wiley Assist, and now MAA WeBWorK. Both commercial systems have their faults: Pearson required a licence which was only valid for one year—this was a problem as many students take 18 months or more to complete their two core units of mathematics; and Wiley was linked to a single textbook, which meant that any deviation from the textbook would not be supported. Also, their authoring systems seemed very complicated and unfriendly. However, they *did* have very handsome and well-designed user interfaces.

2 Why use open source?

Although the initial cost of open source software (zero!) is sometimes seen as its biggest advantage, this has to be balanced out with the costs of deployment, maintenance, upkeep and upgrades, troubleshooting and support. In a large environment, such as a university, the software will have to be either installed on all laboratory computers, as well as on staff computers, or on a central server. There will be "hidden costs" (in support and maintenance) over the life-cycle of the software. Open source software may well be cheap to install, but it is no cheaper to run, and because there will be no support other than user forums, will require local time and effort to deal with any issues which arise.

That being said, there are still powerful and persuasive reasons to consider open-source software:

1. No vendor lock-in. Lock-in can be insidious: you find you become more and more dependent on a software or a service, to the point where it is almost impossible to change. And then as well as the initial costs, there are yearly licensing costs, as well as possibly extra costs for upgrades, extensions, or packages.

2. Known bugs. Mathematics software is complex and complicated, and users can put great demands on it. No software is bug-free[2] but for a discipline requiring exactness and precision a bug in mathematics software can be disastrous. A recent example involving commercial software [4] has received considerable attention; nobody knows the bugs in any commercial system since their companies don't publicize them. The users simply have to trust that the answers they are getting will be correct[3]. Open-source software developers will maintain a publicly accessible database of known bugs.

3. Communication. A great deal of mathematical writing now, in education as in research, will involve some code samples. I believe that this is a major issue. A discussion about a new way to teach a particular topic, such as modelling, can be used by any educator, following the precepts and ideas presented by the authors. A good recent account of just such an approach is given by Wedelin et al [12]; the article investigates pedagogy, course design, examples of problems. But if the authors decide instead to describe how a computer system is used, and if they use a commercial system, the readership is necessarily limited to those with access to the same system.

   The problem is exacerbated in research, when a paper heavily depends on the use of software. Jacob Neubüser, the initial creator of the GAP system for group theory, claimed in 1993:

   > "You can read Sylow's Theorem and its proof in Huppert's book in the library without even buying the book and then you can use Sylow's Theorem for the rest of your life free of charge, but . . . for many computer algebra systems license fees have to be paid regularly for the total time of their use. In order to protect what you pay for, you do not get the source, but only an executable, i.e. a black box. You can press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case. With this situation two of the most basic rules of conduct in mathematics are violated. In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research [...] means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see?"

   People who are sharing ideas, either in print, or directly, must have a common ground with which to communicate, and this includes an agreed computer system as much as common language. Very few people or institutions can afford the costs of purchasing and maintaining several different commercial systems.

[2]An exception is possibly the typesetting software TEX which is considered to be effectively bug-free.

[3]I was told once that a particular commercial system included the decision that a function whose Taylor series was zero for the first 100 terms was the zero function.

3 A far too brief introduction to some open-source software

We shall look briefly at some of the major players in open-source mathematical software. Each one deserves a longer introduction than possible here.

3.1 Computer Algebra Systems

The "big players" are of course Maple[4] and Mathematica[5], and for engineers Mathcad[6], along with the Matlab Symbolic Toolbox[7]. This toolbox is in fact the software system MuPAD, which used to be made available for free (although never open-source).

All these commercial systems are now mature products: Maple was initially released some 25 years ago and is now at version 18; Mathematica was initially released in 1988 but was based on an earlier program called SMP dating back to 1979; Mathcad and MuPAD go back to 1986 and 1990 respectively.

Open-source systems tend to be newer, or are open-source versions of software which were once commercial. In all cases they are developed by volunteers, sometimes including a small core team. We shall consider:

Axiom. An open-source system which is current version of a commercial system called ScratchPad, developed initially at IBM starting in 1971. It is a very powerful system, and unusually for a CAS makes use of strong typing. It has a very slow development, partly based on small user numbers, and partly based on some decisions by the lead developer. Axiom is supposed to be written using literate programming [7], and to be totally future proof: its algorithms are designed to be provably correct, using formal proof systems.

In its current form Axiom has a GUI which looks very old-fashioned (although it is in fact quite powerful), but which runs only under X-windows, on a Unix-based system. This limits the use of Axiom.

However, there is a fork of Axiom, called FriCAS, the developers of which have made remarkable strides in producing a more pleasant user interface. Currently it is possible to run FriCAS in a browser notebook using the Jupyter web application (written in Python), which provides typeset output. At the time of writing, this is still a work in progress, but is very usable.

Maxima. Sometimes known as *GNU* Maxima; the current open-source incarnation of the Macsyma system, which was developed at MIT between 1968 and 1982. Macsyma was initially renowned for its symbolic power. Macsyma was still being developed and released commercially into the mid 1990's, and the current Maxima is based on the 1982 version of Macsyma which was licensed to the US Department of Energy (and hence known as DOE Macsyma). Thus there was over a decade of commercial development of Macsyma which is not part of (GNU) Maxima.

[4]`http://www.maplesoft.com/products/maple/`

[5]`http://www.wolfram.com/mathematica`

[6]`http://www.ptc.com/engineering-math-software/mathcad`

[7]`http://au.mathworks.com/discovery/mupad.html`

However, Maxima has some very powerful symbolic functionality—especially for calculus—and a large number of added packages, some of which have migrated into "core" Maxima, others of which only exist in a user contributed folder, to be imported only if needed. Such a one is the package for computation with finite fields, of which I was the initial author.

Maxima is written in Lisp, and can be compiled under most current Lisps. Hence Maxima can run as a native application on MS Windows, MacOS, Linux, as well as on portable systems, such as Android.

Sage. This started life as SAGE: Software for Algebraic and Geometry Exploration, initially developed by William Stein at the University of Washington, who was dissatisfied with the commercial algebraic system Magma. Stein's vision was to "redesign the car, not reinvent the wheel", and the initial plan was to construct a sort of "super system" based on currently existing open-source systems, and glue them together using the common language Python, adding new functionality as needed. Sage was initially created in 2004–2005, and is now at version 6.7.

Sage has exploded in use, based in part on its interface—it can use the web-based interfaces of Python, which allow typeset output—and in part on its algebraic strength. Some of its routines have been written and contributed by the world's leading researchers in their fields. There is also a huge world-wide group of developers; people who have contributed to Sage development number in the many hundreds.

Specialized systems. There are many open-source systems for specialized mathematics work: CoCoA for commutative algebra, GAP for group theory, Macaulay2 for algebraic geometry, Pari/GP for number theory, SINGULAR for polynomials and algebra. Such systems are generally only used by researchers in their fields, and I will not discuss them in this article. However, most of them are obtainable from within Sage.

We might reasonably ask—how powerful are these systems, and how can their power be measured? In the past CAS's would be compared by throwing a vast selection of problems: algebraic, numeric, graphical, from all areas of mathematics, and see which of them solves the most, and the most quickly. Probably the greatest selection of problems was developed by Michael Wester in the 1990's [13], but there are also others.

However, in terms of comparing CAS's for educational use, we might be more interested in their ability to give simple answers, to make sensible decisions regarding simplification, to be fairly robust against student errors in input, have a comfortable GUI, and produce typeset output and nice graphs. Most teaching at undergraduate level—where we might want to use a CAS—involves mostly simple mathematics. We are unlikely to use a CAS for teaching algebraic topology or complex analysis for example (although specialized systems do exist).

Here are some examples of the systems Axiom, Maxima and Sage.

Axiom

Naturally, all numeric operations are supported, and operations on arbitrary precision real numbers, and integers of arbitrary length:

```
(1) -> binomial(200,100)
```

$$\begin{array}{l}102952500135414432972975880320401986757210925381077648234849059575923333\backslash\\ 237265195859833659551897649295156404859750677412 0\end{array}$$

Type: `PositiveInteger`

Axiom can perform factorization over arbitrary fields: `POLY PF` is an abbreviation for `Polynomials` over `PrimeField`—the Galois field over a prime number.

```
(2) -> p := x^6+x^4+x^2+1
```

$$x^6+x^4+x^2+1$$

Type: `Polynomial(Integer)`

```
(3) -> factor(p)
```

$$(x^2+1)(x^4+1)$$

Type: `Factored(Polynomial(Integer))`

```
(4) -> factor(p::POLY PF 3)
```

$$(x^2+1)(x^2+x+2)(x^2+2x+2)$$

Type: `Factored(Polynomial(PrimeField(3)))`

```
(5) -> factor(p::POLY PF 17)
```

$$(x+2)(x+4)(x+8)(x+9)(x+13)(x+15)$$

Type: `Factored(Polynomial(PrimeField(17)))`

Axiom supports the full Risch algorithm for symbolic integration of elementary functions (the only open source system to do so), and can solve all of the integrals given by Charlwood [2], for example:

```
(6) -> integrate(asin(sqrt(x+1)-sqrt(x)),x)
```

$$\frac{(3\sqrt{x+1}+\sqrt{x})\sqrt{2\sqrt{x}\,\sqrt{x+1}-2x}+(8x+3)\operatorname{asin}(\sqrt{x+1}-\sqrt{x})}{8}$$

Type: `Union(Expression(Integer),...)`

Finally, a bit of linear algebra: a matrix and the Cayley-Hamilton theorem, starting with a little command to produce random numbers:

```
(7) -> r() == randnum(10)+5
```

Type: `Void`

```
(8) -> M := matrix [[r() for i in 1..3] for j in 1..3]
```

$$\begin{pmatrix}10 & 5 & 9\\ 5 & 7 & 5\\ 6 & 7 & 13\end{pmatrix}$$

Type: `Matrix(Integer)`

```
(9) -> cp := characteristicPolynomial(M,x)
```

$$-x^3+30x^2-177x+322$$

Type: `Polynomial(Integer)`

```
(10) -> eval(cp, x = M)
```

$$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$$

Type: `Polynomial(SquareMatrix(3,Integer))`

Axiom is *strongly typed*: as you see every object in Axiom belongs to a domain which precisely describes the operations available. This is both an enormous strength, and a cause of difficulty for learners—and is also partly responsible for the slowness of its development. In the last decade Axiom has been forked into a new system FriCAS[8]which aims to have a faster development cycle than Axiom, and be easier to extend. The examples for this article were in fact obtained using FriCAS, compiled from source code.

Maxima

Maxima's strength is its algebra and calculus: it can't integrate *all* Risch-integrable functions (as Axiom can), but it has greater breadth. Partly this is because it has a greater library of functions than Axiom.
For example:

```
(%i1)  integrate(x*sin(x^3),x);
(%o1)
```

$$\frac{i\,\mathrm{gamma\_incomplete}\left(\frac{2}{3}, i\,x^3\right) - i\,\mathrm{gamma\_incomplete}\left(\frac{2}{3}, -i\,x^3\right)}{6}$$

Maxima also has lots of different commands for converting expressions. Here for example, we can obtain the trigonometric solution of a cubic polynomial equation:

```
(%i2)  p: x^3-3*x-1$
(%i3)  sol: solve(p=0,x);
(%o3)
```

$$\left[x = \left(-\frac{\sqrt{3}\,i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{\frac{\sqrt{3}\,i}{2} - \frac{1}{2}}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}}, x = \left(\frac{\sqrt{3}\,i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{-\frac{\sqrt{3}\,i}{2} - \frac{1}{2}}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}}, x = \left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{1}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}}\right]$$

```
(%i4)  map(lambda([x],trigsimp(rectform(x))),sol);
(%o4)
```

$$\left[x = \sqrt{3}\,\sin\left(\frac{\pi}{9}\right) - \cos\left(\frac{\pi}{9}\right), x = -\sqrt{3}\,\sin\left(\frac{\pi}{9}\right) - \cos\left(\frac{\pi}{9}\right), x = 2\,\cos\left(\frac{\pi}{9}\right)\right]$$

Here is how we might solve the differential equation

$$\frac{d^2y}{dx^2} - 6\frac{dy}{dx} + 13y = 5\sin(2x), \qquad y(0) = 1, y'(0) = -4.$$

by first setting up the equation, then solving it in general, and finally entering the initial values:

```
(%i5)  de:'diff(y,x,2)-6*'diff(y,x)+13*y=5*sin(2*x);
(%o5)
```

$$\frac{d^2}{d\,x^2}\,y - 6\,\left(\frac{d}{d\,x}\,y\right) + 13\,y = 5\,\sin\left(2\,x\right)$$

[8]http://fricas.sourceforge.net/

```
(%i6)  sol:ode2(de,y,x);
(%o6)
```

$$y = e^{3\,x}\,\left(\%k1\,\sin\left(2\,x\right) + \%k2\,\cos\left(2\,x\right)\right) + \frac{3\,\sin\left(2\,x\right) + 4\,\cos\left(2\,x\right)}{15}$$

```
(%i7)  ic2(sol,x=0,y=1,'diff(y,x)=-4);
(%o7)
```

$$y = \frac{3\,\sin\left(2\,x\right) + 4\,\cos\left(2\,x\right)}{15} + e^{3\,x}\,\left(\frac{11\,\cos\left(2\,x\right)}{15} - \frac{33\,\sin\left(2\,x\right)}{10}\right)$$

Sage

Sage contains a great many open source mathematical systems within it (including Maxima); there are also interfaces to other systems. If you have access to Maple or Mathematica, you can use them within Sage. This means that you can use the output of a Maple computation for instance, as input to a Sage command.

Sage is also built on top of Python, so you have all the power of Python (and its libraries). This makes Sage the easiest of all systems for programming.

Sage uses types, but not to the same extent as Axiom; much of Sage's computations are performed using pattern matching. Sage's typing is considered to be *duck typing*: an object is not defined to be of a particular type, but only in terms of the operations available for it. To the casual user duck typing looks like strong typing[9] but in fact it is much weaker.

Here's how Sage can be used to produce Heron's formula for the area of a triangle. We start with the standard result that the area of a triangle with vertices $(p_1, q_1), (p_2, q_2), (p_n, q_n)$ has area K, defined using the absolute value of a determinant:

$$K = \frac{1}{2}\,\mathrm{abs}\left(\begin{vmatrix} p_1 & q_1 & 1 \\ p_2 & q_2 & 1 \\ p_3 & q_3 & 1 \end{vmatrix}\right)$$

```
sage:  var('p1,q1,p2,q2,p3,q3')
sage:  M = matrix([[p1,q1,1],[p2,q2,1],[p3,q3,1]])
sage:  area = M.det()/2
sage:  P.<p1,q1,p2,q2,p3,q3,a,b,c,K> = PolynomialRing(QQ)
sage:  edges = [(p1-q1)^2+(p2-q2)^2-a^2,(p2-p3)^2+(q2-q3)^2-b^2,\
....:  (p3-p1)^2+(q3-q1)^2-c^2]
sage:  Id = P.ideal([K-area]+edges)
sage:  E = Id.elimination_ideal([p1,p2,p3,q1,q2,q3])
sage:  E.gen(0)
```

$$a^4 - 2a^2b^2 + b^4 - 2a^2c^2 - 2b^2c^2 + c^4 + 16K^2$$

```
sage:  factor(E.gen(0)-16*K^2)
```

$$(-a + b - c)(-a + b + c)(a + b - c)(a + b + c)$$

[9]James Whitcomb Riley (1849–1916): "When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

Taking the absolute value produces

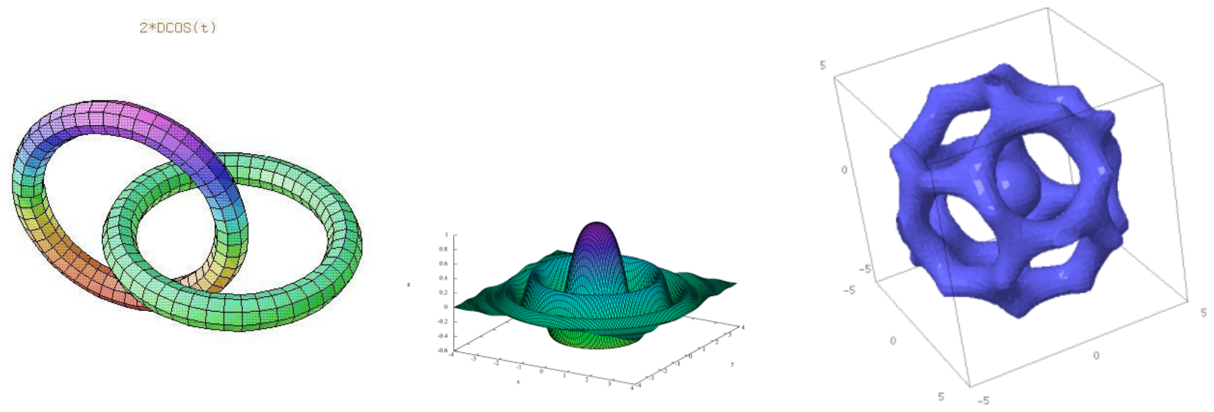
$$\begin{aligned} K^2 &= \frac{1}{16}(a-b+c)(-a+b+c)(a+b-c)(a+b+c) \\ &= \left(\frac{a-b+c}{2}\right)\left(\frac{-a+b+c}{2}\right)\left(\frac{a+b-c}{2}\right)\left(\frac{a+b+c}{2}\right) \end{aligned}$$

which can be written in the more usual way using $s = (a+b+c)/2$. The computation above includes the `elimination_ideal` method, which in fact uses Gröbner bases computations to eliminate variables from a set of polynomial equations. Sage's Gröbner basis routines are by far the strongest of any open source system; also the user has the option of choosing between different algorithms. The default behaviour is to devolve such computations to the subsystem Singular [3], but the user has neither to be aware of Singular nor of the algorithms being used.

A good account of the philosophy of Sage is given by Eröcal and Stein [5].

Graphics and Interfaces

All of these systems support graphics in two and three dimensions: functions defined explicitly, implicitly, parametrically, special shapes such as polyhedra, graphs and networks, and all with some sort of interactivity. So the user can move a three dimensional shape about to obtain the best view of it, zoom in and out, change colours, change rendering. Figure 1 shows examples of graphics generated in each of the systems.



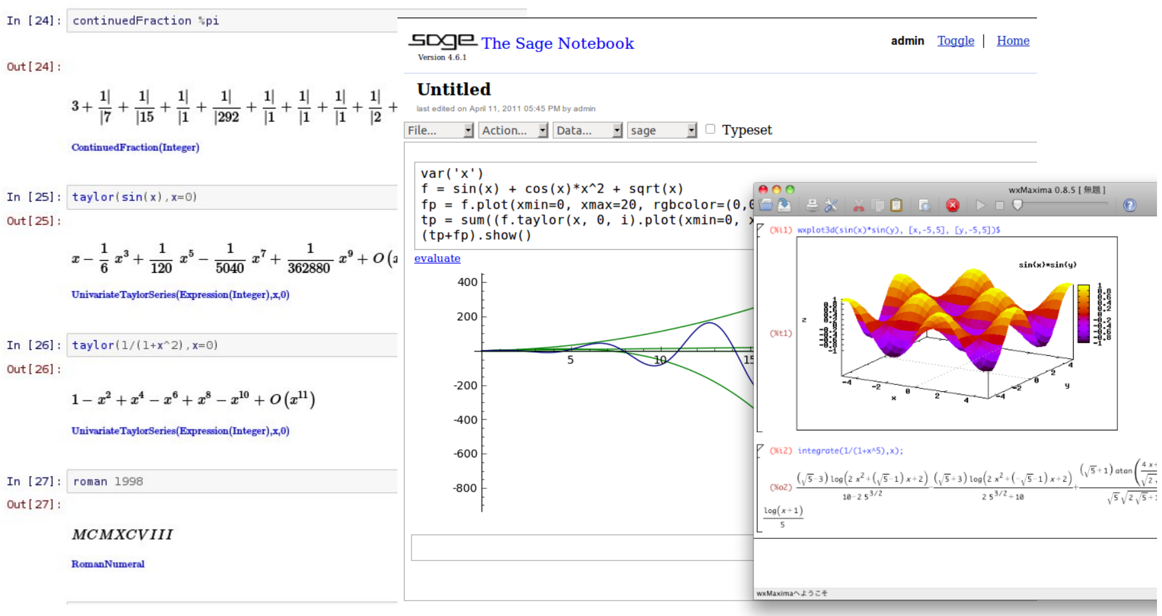
(a) Axiom (b) Maxima (c) Sage

Figure 1: Graphics examples

One of the limitations of open source systems has been in the interfaces: elegant publishable notebook interfaces with floating palettes do not exist in the open-source world. However, there have been recent advances: Maxima has long had its wxMaxima interface, which runs under many different operating systems, and both Sage and now FriCAS can run in a browser using the iPython system. Figure 2 shows examples of the interfaces of each system. And of course all systems can run in a console, without graphics, and without typeset output.

3.2 Numeric Software

In this section we shall briefly investigate numeric software. The standard commercial offering is Matlab® which is beloved of engineers the world over, with Mathcad® a close second. The



```
In [24]: continuedFraction %pi
Out[24]:
```

$$3+\frac{1|}{|7}+\frac{1|}{|15}+\frac{1|}{|1}+\frac{1|}{|292}+\frac{1|}{|1}+\frac{1|}{|1}+\frac{1|}{|1}+\frac{1|}{|2}+$$

ContinuedFraction(Integer)

```
In [25]: taylor(sin(x),x=0)
Out[25]:
```

$$x-\frac{1}{6}x^3+\frac{1}{120}x^5-\frac{1}{5040}x^7+\frac{1}{362880}x^9+O($$

UnivariateTaylorSeries(Expression(Integer),x,0)

```
In [26]: taylor(1/(1+x^2),x=0)
Out[26]:
```

$$1-x^2+x^4-x^6+x^8-x^{10}+O(x^{11})$$

UnivariateTaylorSeries(Expression(Integer),x,0)

```
In [27]: roman 1998
Out[27]:
```

MCMXCVIII

RomanNumeral

```
var('x')
f = sin(x) + cos(x)*x^2 + sqrt(x)
fp = f.plot(xmin=0, xmax=20, rgbcolor=(0,0
tp = sum((f.taylor(x, 0, i).plot(xmin=0, x
(tp+fp).show()
```

```
(%i1) wxplot3d(sin(x)*sin(y), [x,-5,5], [y,-5,5])$
(%t1)
(%i2) integrate(1/(1+x^5),x);
```

$$(\%o2)\ \frac{(\sqrt{5}-3)\log(2x^2+(\sqrt{5}-1)x+2)}{10-2\,5^{3/2}}-\frac{(\sqrt{5}+3)\log(2x^2+(-\sqrt{5}-1)x+2)}{2\,5^{3/2}+10}+\frac{(\sqrt{5}+1)\operatorname{atan}\left(\frac{4x+}{\sqrt{2}}\right)}{\sqrt{5}\sqrt{2\sqrt{5}+}}$$

$$\frac{\log(x+1)}{5}$$

(a) FriCAS (b) Sage (c) Maxima

Figure 2: Interfaces

two main open-source contenders are GNU Octave and Scilab. GNU Octave is designed to be Matlab-compatible, with some small differences. Programs written in Matlab, as long as they don't rely on specific extra toolboxes, should run with very little modification on Octave. Scilab is not so concerned with compatibility, although much of its syntax is similar to that of Matlab. Scilab also comes with a graphical editor called Xcos to design and simulate dynamical systems; similar in some ways to Matlab's Simulink. Octave does not have such a graphical subsystem, although most of this simulation can be achieved by other means.

To show the power of these systems, we shall solve a simple numerical problem: to fit the SIR model of disease spread to data of an influenza outbreak in an English school. This is a well-known case study; the number of infected students from day one to 14 of the outbreak were

$$3,\ 6,\ 25,\ 73,\ 222,\ 294,\ 258,\ 237,\ 191,\ 125,\ 69,\ 27,\ 11,\ 4$$

with no deaths. Thus the total population remained constant, and so the disease model:

$$\frac{dS}{dt}=-\beta IS,\ \frac{dI}{dt}=\beta IS-\gamma I,\ \frac{dR}{dt}=\gamma I$$

where S, I, R are the number of susceptible, infected, and recovered persons respectively, should fit the data, for appropriately chosen values of β and γ.

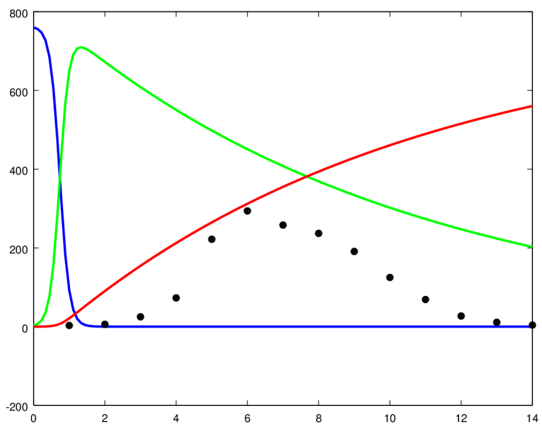
With Octave, the first step is to create a function to model the differential equations:

```
function xdot = f (x, t)
    global B;
    global G;
    xdot = zeros(3,1);
    xdot(1) = -B*x(1)*x(2);
    xdot(2) = B*x(1)*x(2)-G*x(2);
    xdot(3) = G*x(2);
endfunction
```

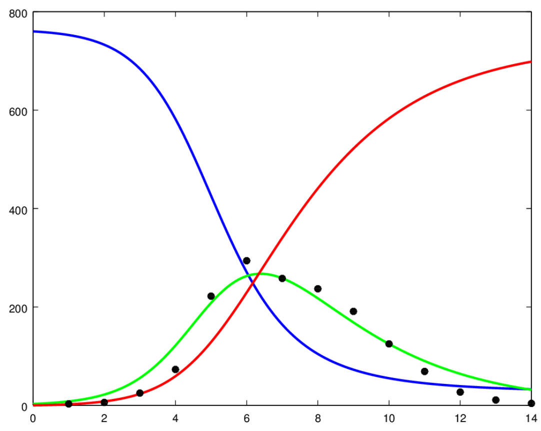
Then the `lsode` function can be used to provide a numerical solution. As an example, we shall use the parameters $\beta = 0.01$ and $\gamma = 0.1$ (which we shall refer to as `B` and `G`):

```
octave:  t = linspace(0,14,127);
octave:  data = [ 6 25 73 222 294 258 237 191 125 69 27 11 4];
octave:  B = 0.01; G = 0.1;
octave:  y = lsode("f",[760;3;0],t);
octave:  S = y(:,1);I=y(:,2),R=y(:,3);
octave:  plot(t,S,"b","linewidth",2,t,I,"g","linewidth",2,...
> t,R,"r","linewidth",2,[1:14],data,"*k","markersize",10)
```

The plot is shown on the left in figure 3. Note that the green curve—representing the infected numbers, is a very poor fit for the actual data.



$\beta = 0.01, \gamma = 0.1$



β, γ minimizing difference

Figure 3: The SIR model with different parameters

To find the parameters that best fit the data, first we need a function which produces a sum of squares between the data and the computed I values:

```
function out = ss(b)
    global B;
    global G;
    B = b(1);G=b(2);
    t = linspace(0,14,127);
    y = lsode("f",[760;3;0],t);
    data = [3 6 25 73 222 294 258 237 191 125 69 27 11 4]';
    out = sum((data-y(10:9:127,2)).^2);
endfunction
```

Now we can use the `nelder_mead_min` function from Octave's "Optim" package:

```
octave:  nelder_mead_min(@(x) ss(x),[0.001;0.001])

ans =

   0.0018868
   0.4192210
```

Now if these parameters are used for β and γ in the model, the resulting graph is shown on the right in figure 3, and the curve representing the infected numbers is a very good fit to the initial data.

The programs and commands for Scilab are nearly identical. First the programs:

```
function ydot = f(t,y)
  global('B', 'G')
  ydot(1) = -B*y(1)*y(2);
  ydot(2) = B*y(1)*y(2)-G*y(2)
  ydot(3) = G*y(2);
endfunction
```

```
function out = ss(b)
  global('B','G')
  B = b(1);G=b(2);
  t = linspace(0,14,127);
  x = ode([760;3;0],[0;0;0],t,f);
  data = [3 6 25 73 222 294 258 237 191 125 69...
  27 11 4];
  out = sum((data-x(2,10:9:127)).^2);
endfunction
```

and the commands used:

```
-->t = linspace(0,14,127)
-->B=0.01; G=0.1;
-->x = ode([760;3;0],[0;0;0],t,f);
-->data = [3 6 25 73 222 294 258 237 191 125 69 27 11 4]
-->plot(t,x(1,:),'r',t,x(2,:),'g',t,x(3,:),'b',[1:14],data,'.k')
-->fminsearch(ss,[0.001;0.001])
 ans =

    0.0018869    0.4192225
```

There are some excellent discussions and comparisons of numeric tools [1, 6, 10] which test both free and commercial software against a variety of numerical and computational problems.

3.3 Assessment tools

For assessing large student cohorts, online assessment systems are deservedly popular. They have their faults: it is very hard to marking a student's working, and so most of the time the system will just check a final answer. For this reason they can't as yet take the place of well designed written assessment, where the marker can concentrate as much on the student's logic as on the arithmetic or algebraic correctness.

The most popular systems at the moment seem to be the commercial Pearson MyMathLab, Wiley Assist, Maple TA, and the open-source STACK and WeBWorK. The first two commercial systems are provided by academic publishers, and have the advantage of slick interfaces. However they also have certain disadvantages:

1. Pearson provide a year's access with any purchase of a text. However, students who take more than a year to complete their mathematics studies either have to miss out, or purchase a new licence. Pearson also only provides access to a limited number of texts for questions.

2. Wiley provides their online assessment free to students, but it will be based around the one text chosen by the lecturer. A teacher can't choose questions from a different text to set for the students. So as with Pearson, the available question bank can be limited for some questions.

3. Both systems have poor answer checkers. MyMathLab seems to work on string matching, which is highly restrictive. Both systems also have somewhat unintuitive question editors.

We have found that MAA WeBWorK satisfies our needs very well. It provides a very large number of different question types: textual, symbolic, numeric, graphic, along with multiple choice and pair matching. It has a question composer which although not easy, is quite manageable for a user with experience of a markup language. It also has a vast question bank, containing many tens of thousands of questions from all areas of elementary undergraduate mathematics. It can run off any well set-up server, and although its installation is not trivial, it is very clearly described on the website. Also the user forums (which include the developers) are active, and the developers seem quite happy to answer questions of users and administrators.

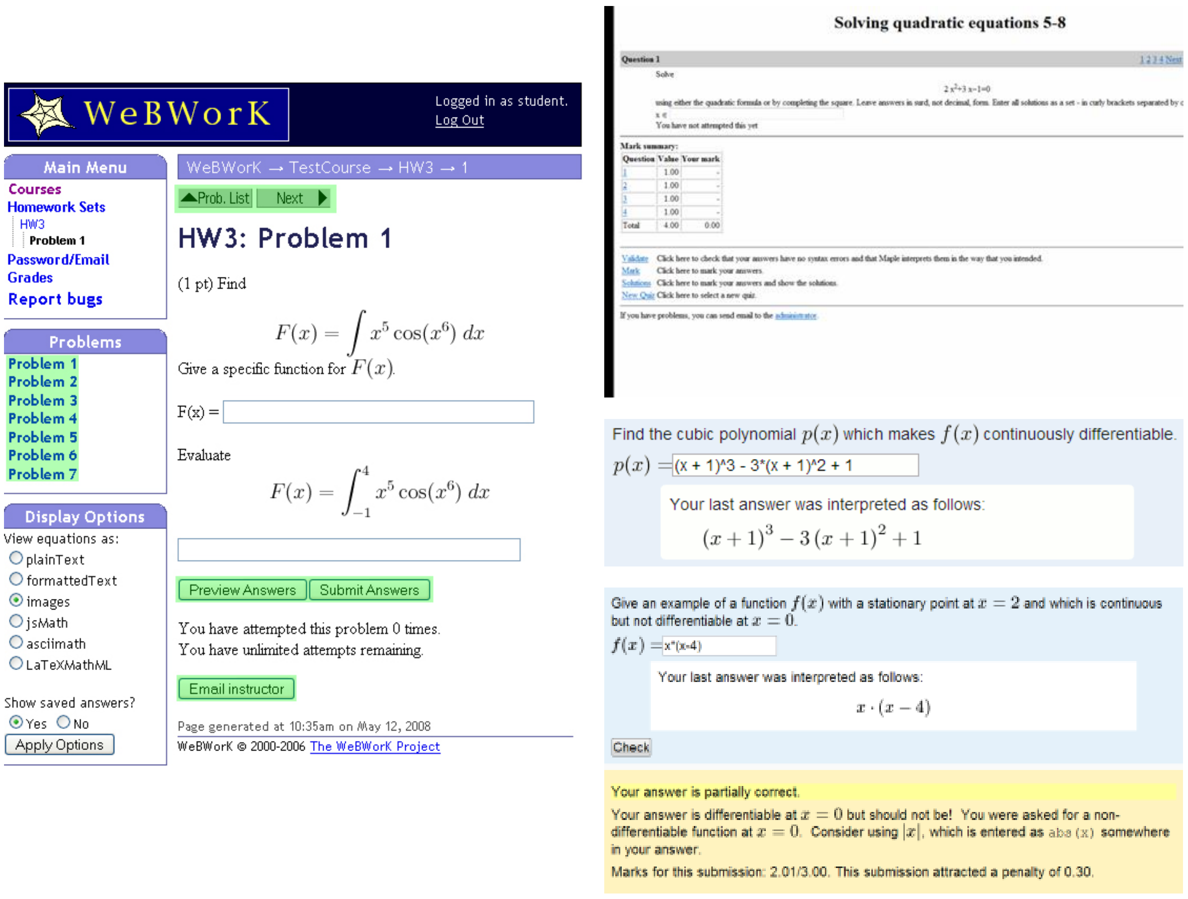
Figure 4 shows several screenshots from WeBWorK and from STACK. More information can be found at their respective websites, or in the excellent text by the author of STACK [9].

4 Conclusions

In this brief article we have only scratched the surface of the open source world, and looked at a few products. We have not touched on dynamic geometry software, of which Geogebra, C.A.R/CarMetal and Cinderella are the principal current free offerings. In each of the three areas we discussed: computer algebra systems, numeric software, assessment, there are many other products. There are also programming languages designed for particular mathematical use, or language libraries, such as SymPy for Python (which includes excellent symbolic processing, as well as geometry), and also for Python the numeric and scientific libraries SciPy and NumPy. Users of C or C++ can use the GNU Multiple Precision Library (which has wrappers for other languages), or PARI. Julia is a new language designed to have the power of Matlab and the speed of C. Thus the user is spoiled for choice.

Unless there are very specific requirements which can only be met by a commercial system, I see no need for mathematics educators not to strongly support open source software. Having some software in common means that it is far easier to share ideas; whether teaching or research. However, because there are often different products to choose it does not necessarily mean that any one person will be experienced in more than one. However, removing the price element means that downloading (and sometimes not even that) and experimenting can be done at no cost. This is not the case with material involving commercial software.

We also note that bugs in open-source software are known and made publicly available: any system will keep a list of known bugs, and possibly also a timeline for fixing them. Nobody can know what bugs exist in commercial software. In fact, sometimes you can't even know much



HW3: Problem 1

(1 pt) Find

$$F(x) = \int x^5 \cos(x^6)\, dx$$

Give a specific function for $F(x)$.

F(x) =

Evaluate

$$F(x) = \int_{-1}^{4} x^5 \cos(x^6)\, dx$$

You have attempted this problem 0 times.
You have unlimited attempts remaining.

View equations as:
plainText
formattedText
[✓ images]
jsMath
asciimath
LaTeXMathML

Show saved answers?
[✓ Yes] No

Page generated at 10:35am on May 12, 2008
WeBWorK © 2000-2006 The WeBWorK Project

Solving quadratic equations 5-8

Question 1

Solve

$$2x^2+3x-1=0$$

using either the quadratic formula or by completing the square. Leave answers in surd, not decimal, form. Enter all solutions as a set - in curly brackets separated by c

$x \in$

You have not attempted this yet

Mark summary:

Question	Value	Your mark
1	1.00	-
2	1.00	-
3	1.00	-
4	1.00	-
Total	4.00	0.00

Validate Click here to check that your answers have no syntax errors and that Maple interprets them in the way that you intended.
Mark Click here to mark your answers.
Solutions Click here to mark your answers and show the solutions.
New Quiz Click here to select a new quiz.

If you have problems, you can send email to the administrator.

Find the cubic polynomial $p(x)$ which makes $f(x)$ continuously differentiable.

$p(x) =$ (x + 1)^3 - 3*(x + 1)^2 + 1

Your last answer was interpreted as follows:

$$(x+1)^3 - 3\,(x+1)^2 + 1$$

Give an example of a function $f(x)$ with a stationary point at $x = 2$ and which is continuous but not differentiable at $x = 0$.

$f(x) =$ x*(x-4)

Your last answer was interpreted as follows:

$$x \cdot (x-4)$$

Your answer is partially correct.

Your answer is differentiable at $x = 0$ but should not be! You were asked for a non-differentiable function at $x = 0$. Consider using $|x|$, which is entered as abs(x) somewhere in your answer.

Marks for this submission: 2.01/3.00. This submission attracted a penalty of 0.30.

Figure 4: Online assessment screenshots

about the algorithms being used; the software documentation making it clear that you should just know how to use the system, not how the systems obtains its results[10]. This means you have to place implicit trust in a closed "black box" system. You might never want to check out the code in your system, but the very fact that the code is open and available for checking provides a level of trust.

Finally, of you find some software you like but which doesn't support a particular area of mathematics—you can code it up yourself (or set it for students as a project)—and submit it to the project. Much modern open-source software contains material which started life as student projects, some at high school.

In many ways open-source software for mathematics is still in its starting phase: it will take a long time before any single system has the power and the usability of the big current commercial products. But you may well find that a carefully chosen suite of products, can be matched to fit your own requirements. And also, you will have the excitement of knowing that you are working at the cutting edge of mathematical software.

[10]`https://reference.wolfram.com/language/tutorial/WhyYouDoNotUsuallyNeedToKnowAboutInternals.html`

References

[1] Eliana S de Almeida, Antonio C Medeiros, and Alejandro C Frery. "How good are MatLab, Octave and Scilab for computational modelling?" In: *Computational & Applied Mathematics* 31.3 (2012), pp. 523–538.

[2] Kevin Charlwood. "Integration on Computer Algebra Systems". In: *Electronic Journal of Mathematics and Technology* (2008).

[3] Wolfram Decker et al. SINGULAR *4-0-2 — A computer algebra system for polynomial computations*. `http://www.singular.uni-kl.de`. 2015.

[4] Antonio J Durán, Mario Pérez, and Juan L Varona. "The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?" In: *Notices of the AMS* 61.10 (2014).

[5] Burçin Eröcal and William Stein. "The Sage Project: Unifying free mathematical software to create a viable alternative to Magma, Maple, Mathematica and MATLAB". In: *Mathematical Software–ICMS 2010*. Springer, 2010, pp. 12–27.

[6] Themistoklis Glavelis, Nikolaos Ploskas, and Nikolaos Samaras. "A computational evaluation of some free mathematical software for scientific computing". In: *Journal of Computational Science* 1.3 (2010), pp. 150–158.

[7] Donald Ervin Knuth. "Literate programming". In: *The Computer Journal* 27.2 (1984), pp. 97–111.

[8] Alasdair McAndrew et al. "Using a computer algebra system to facilitate the transition from secondary to tertiary mathematics". In: *World Conference on Computers in Education VI Abstracts*. 1995, p. 287.

[9] Chris Sangwin. *Computer Aided Assessment of Mathematics*. OUP Oxford, 2013.

[10] Neeraj Sharma and Matthias K. Gobbert. *A comparative evaluation of Matlab, Octave, FreeMat, and Scilab for research and teaching*. Tech. rep. available at `http://ciep.itam.mx/~rtorres/progdin/comparative-evaluation-of-matlab-octave-scilab-freemat.pdf`. Department of Mathematics & Statistics, University of Maryland, 2010.

[11] Richard Stallman. *Why Open Source misses the point of Free Software*. Accessed September 11, 2015. 2015. URL: `https://www.gnu.org/philosophy/open-source-misses-the-point.html`.

[12] Dag Wedelin et al. "Investigating and developing engineering students mathematical modelling and problem-solving skills". In: *European Journal of Engineering Education* ahead-of-print (2015), pp. 1–16.

[13] Michael Wester. "A critique of the mathematical abilities of CA systems". In: *Computer Algebra Systems: A Practical Guide* 16 (1999), p. 436.