# Introduction to Axiom and FriCAS

Alasdair McAndrew

January 2016

## 1 Introduction

Many years ago (2007) I wrote a series of blog posts on Axiom, which you can still find at

`http://bit.ly/1W3sO4L`

These posts were among the first new material about Axiom for some time. Since then the world of software has not stood still, and Axiom and in particular its forks have grown. So this document is both a rewrite of my old introduction, and a brief look at some of the new, and continuing, changes.

To start with, a bit of history[1]. Axiom started life as Scratchpad, which was a system for numeric and symbolic computation, and developed from the early 1970's at IBM. In the 1990's the system was sold to the Numerical Algorithms Group and renamed Axiom. It was never the commercial success than NAG had hoped, so was withdrawn from the market in 2001 and a bit later released as open-source. It has always had a small and devoted group of developers and users, but it has never enjoyed the support of the Big Systems such as Maple and Mathematica, or even MuPAD and Macsyma (now Maxima). Its use of strong typing requires a slightly different mindset of computational thinking than for say Maple and Mathematica—and indeed almost any other system—for which pattern matching provides the basis for much of the symbolic algorithms. However, Axiom has significant strengths. For example it is currently the *only* open-source system to provide a complete implementation of the Risch decision algorithm for integration.

In about 2007, some of the developers, dissatisfied with what they saw as the slow growth of the system, and the direction in which it seemed to be going, split off from Axiom and produced two forks: FriCAS, and OpenAxiom. Of these two forks the first is the most viable, and indeed is being actively developed. Already FriCAS differs from its parent in having a web notebook interface, a web-based front-end to the documentation, and a greater library of special functions. And several of the libraries have been enhanced. FriCAS retains most of its parent's code, and indeed most commands will run equally well on both Axiom and FriCAS and produce the same output. But not all will. In order to bring some sense to the confusion of names, the term "PanAxiom" has been proposed as an umbrella term for all of them. Then "Axiom" refers to the original software, and the forks are referred to by their names.

Given the shared material between the systems though, I will use "Axiom" when referring to generic material, as well as to material specific to the initial system, and "FriCAS" if referring to matter specific to that fork.

[1]A more complete history can be found at `http://www.axiom-developer.org/`

# 2 The basics

## 2.1 Obtaining Axiom/FriCAS

For Axiom first go to `http://www.axiom-developer.org/` for a description and discussion, and then go to `http://www.axiom-developer.org/axiom-website/download.html` to find and download some binaries. Axiom works best under Linux. Under windows you have several choices: you can run a native binary in console mode only, without graphics, or you can run Axiom inside an X-windows server for Windows such as Xming. One interface for Axiom is to use it as a session within the TeXmacs editor[2]. More recently, Axiom has been configured to run in a Docker container: Docker allows lightweight virtualization and the Docker Virtual Machine is a lightweight linux VM which can run within VirtualBox. This means that instead of installing a complete Linux distribution in which to run Axiom, you just install a Docker VM.

FriCAS is found at `http://fricas.sourceforge.net/`; you can download binaries for several systems, or if you prefer you can download the source code and compile it using any Lisp. As with Axiom, FriCAS can run in TeXmacs. It also has some other interfaces, which will be discussed later.

## 2.2 A note on Axiom and its forks

As mentioned earlier, owing to some disagreement among Axiom developers, the original Axiom has spawned two forks: FriCAS, and OpenAxiom. From the users' point of view, there is not much to choose between the three. However, the development models and goals of the three are very different:

**Axiom** "is intended to support educational and research objectives", and aims to be very slow and careful in its development, using literate programming, so the development done now will still be useful and readable in 30 years time. Thus Axiom puts great stress on provable correctness of algorithms, and readability of code. This can make adding new material non-trivial.

**FriCAS** "will use lightweight development, allowing much faster evolution." Also, FriCAS "add hooks which make adding alternative user interfaces easier." FriCAS aims to be more user friendly than its parent, and in particular easier to extend.

**OpenAxiom** "aims at being the open source computer algebra system of choice for research, teaching, engineering, etc." It differs from its parent in technical details.

That being said, there has been quite a lot of development of FriCAS in particular over the past seven or so years: its current developers include some of the best programmers from the initial Axiom team. For example, FriCAS now has quite an extensive library of special functions, as well as a web-based notebook interface, similar to the Sage notebook. The HyperDoc system for managing documentation in a some clunky and old-fashioned (yet undeniably powerful) X-windows system is being replaced with a slicker web interface.

The term "PanAxiom" is now used to mean either the original Axiom, or the user experience of using any of its forks.

One very important aspect of Axiom is that it uses *types* extensively. Everything you do in Axiom takes an input of one of the many types defined in Axiom, and produces an output of a particular type. One of the greatest difficulties for the beginner (well, it was certainly hard for

[2]`http://www.texmacs.org`

me!) is making sense of the types, and ensuring that your input type is commensurate with the mathematics you are trying to do with it. But this is also one of Axiom's greatest strengths.

## 3 Starting off

If you start Axiom or Fricas, let's suppose in a console, you will be see some information on the screen, and then be presented with a prompt, consisting of a number an an ascii arrow, something like this:

```
Checking for foreign routines
AXIOM="/usr/local/lib/fricas/target/x86_64-unknown-linux"
spad-lib="/usr/local/lib/fricas/target/x86_64-unknown-linux/lib/libspad.so"
foreign routines found
openServer result 0
                       FriCAS Computer Algebra System
                         Version: FriCAS 2014-12-18
                   Timestamp: Sat Oct 10 19:29:56 EST 2015
-----------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave FriCAS and return to shell.
-----------------------------------------------------------------------------


(1) ->
```

If you start Axiom instead of FriCAS you'll see much the same as above, but with AXIOM instead of FriCAS. There are a number of commands for interacting with the system environment; these all begin with a right parenthesis. These can be ignored if you like, but in fact they are extremely useful. In particular `)set` provides many functions for changing the appearance of the output; these can be seen by entering `)set output` at the prompt.

But let's start simply: Axiom can be used like a calculator:

```
(1) -> 2+3
```

$5$

```
                                                Type: Positive Integer
(2) -> 17*42
```

$714$

```
                                                Type: Positive Integer
(3) -> %/3
```

$238$

```
                                                Type: Fraction Integer
(4) -> 4^7
```

$16384$

```
                                                Type: Positive Integer
(5) -> 1/2 + 1/3 + 1/4 + 15
```

$\frac{77}{60}$

```
                                                Type: Fraction Integer
```

Axiom has all the standard functions:

```
(6) -> sin(%pi/8.0)
```

$0.3826834323\_6508977173$

Type: Float

```
(7) -> sqrt(2.0)
```

$1.4142135623\_730950488$

Type: Float

```
(8) -> log(3.0)
```

$1.0986122886\_681096914$

Type: Float

```
(9) -> exp(%i*%pi/4.0)
```

$0.7071067811\_865475244 + 0.7071067811\_865475244\%i$

Type: Expression(Complex(Float))

Note that `%pi` produces $\pi$, and `%i` the imaginary unit. Other constants include various infinities, which I'll talk about in a further post.

Note also that the Type given at the end of each computation can be quite long (and there are many longer than these); there is the option to print instead abreviated versions:

```
(10) -> )set output abbreviate on
(10) -> exp(%i*%pi/4.0)
```

$0.7071067811\_865475244 + 0.7071067811\_865475244\%i$

Type: EXPR(COMPLEX(FLOAT))

However, for the moment we shall stick with their full names.

```
(11) -> )set output abbreviate off
```

Note that Axiom, like most computer algebra systems, will attempt to give a result in closed (symbolic) form. To force a floating point output, you need to include a floating point input, or use the numeric function, or coerce your input to be of type Float. (I'll discuss types and coercion in a later post.)

Axiom has no trouble with integers of arbitrary size:

```
(11) -> 2^150
```

$1427247692705959881058285969449495136382746624$

Type: PositiveInteger

```
(12) -> sum(1/k,k=1..100
```

$$\frac{14466636279520351160221518043104131447711}{2788815009188499086581352357412492142272}$$

Type: Union(Expression Integer,...)

And Axiom can produce numeric output to arbitrary precision. The precision can be set either in number of digits with the `digits` function, or by number of bits with `precision`. Then the number of digits displayed can be set by any of `outputFixed`, `outputFloating` or `outputGeneral`, all of which dispkay the output in slightly different ways. The last two commands here are classic examples of "near integer" computations.

```
(13) -> digits(40);outputGeneral(40);
```

Type: Void

```
(14) -> numeric(%pi)
```
3.1415926535_8979323846_2643383279_502884197

Type: Float

```
(15) -> sqrt(2.0)
```
1.4142135623_7309504880_1688724209_69807857

Type: Float

```
(16) -> exp(%pi*sqrt(163.0)
```
26253741_2640768743.9999999999_9925007259_76

Type: Float

```
(17) -> numeric (log(640320^3+744)/%pi)^2
```
163.0000000000_0000000000_0000000023_216778

Type: Float

Note that it is important to ensure that `digits` $\geq$ `outputGeneral` or else the last few digits displayed will be nonsense:

```
(18) -> digits(5);outputGeneral(20); 1.0/3
```
0.3333339691_162109375

Type: Float

## 4 Algebra

Axiom variables are created using the "colon equals" method of many other computer languages, and user defined functions with a double equals:

```
(19) -> a:=%pi/2
```
$$\frac{\pi}{2}$$

Type: Pi

```
(20) -> b:=2^(1/3)+4^(1/3)
```
$$\sqrt[3]{4} + \sqrt[3]{2}$$

Type: AlgebraicNumber

```
(21) -> p(x)==x^3-6*x-6
```

Type: Void

```
(22) -> p(b)
```
$$3\sqrt[3]{2}\sqrt[3]{4}^2 + (3\sqrt[3]{2}^2 - 6)\sqrt[3]{4} - 6\sqrt[3]{2}$$

Type: AlgebraicNumber

```
(23) -> expand(%)
```
0

Type: AlgebraicNumber

Expansion, factorization and simplification of all sorts of expressions can be accomplished easily:

```
(24) -> expand((x+2*y-3*z)^5
```

$$
\begin{aligned}
&-243\ z^5+(810\ y+405\ x)\ \ z^4+\left(-1080\ y^2-1080\ x\ y-270\ x^2\right)\ \ z^3\\
&+\left(720\ y^3+1080\ x\ y^2+540\ x^2\ y+90\ x^3\right)\ \ z^2\\
&+\left(-240\ y^4-480\ x\ y^3-360\ x^2\ y^2-120\ x^3\ y-15\ x^4\right)\ \ z\\
&+32\ y^5+80\ x\ y^4+80\ x^2\ y^3+40\ x^3\ y^2+10\ x^4\ y+x^5
\end{aligned}
$$

`Type: Polynomial Integer`

```
(25) -> factor(%)
```

$$-(3z-2y-x)^3$$

`Type: Factored Polynomial Integer`

```
(26) -> simplify((x^9-1)/(x^3-1))
```

$$x^6+x^3+1$$

`Type: Expression Integer`

```
(27) -> a:=((sin(x)^2-cos(x)^2)*tan(x)-sin(x)*tan(x)+cos(x))_
/(tan(x)-sec(x))
```

$$\frac{\left(\sin(x)^2-\sin(x)-\cos(x)^2\right)\tan(x)+\cos(x)}{\tan(x)-\sec(x)}$$

`Type: Expression Integer`

```
(28) -> simplify(a)
```

$$-2\cos(x)^2+1$$

`Type: Expression Integer`

Note that in the expansion of the multivariate polynomial above, Axiom treats the result as a polynomial in one variable (in this case $z$), with the coefficients being expressions in the other variables. We shall see later how to obtain a complete expansion, with no brackets.

Axiom has powerful algorithms to solve polynomial equations, systems of polynomials, and linear system. There are in fact several different solve functions:

**solve** Basic solve function: will produce real solutions. The precision of the solution can be adjusted by including an extra parameter

**complexSolve** The complex version of solve—will produce complex solutions, also with an adjustable numerical precision

**radicalSolve** As its name implies, solutions in the form of radicals, if such can be found.

```
(29) -> radicalSolve(p(x)=0,x)
```

$$\left[x=\frac{\left(-\sqrt{-3}+1\right)\sqrt[3]{4}^2-4}{\left(\sqrt{-3}+1\right)\sqrt[3]{4}},x=\frac{\left(-\sqrt{-3}-1\right)\sqrt[3]{4}^2+4}{\left(\sqrt{-3}-1\right)\sqrt[3]{4}},x=\frac{\sqrt[3]{4}^2+2}{\sqrt[3]{4}}\right]$$

`Type: List Equation Expression Integer`

```
(30) -> digits(20);outputGeneral(20);
```

`Type: Void`

```
(31) -> solve(p(x)=0,1.0E-10)
```

$$[x=2.8473221018\_502954394]$$

`Type: List Equation Polynomial Float`

Note that we have asked for a precision of $10^{-10}$, but 20 digits of output. This means that the last half of the output as given here may be discarded as incorrect. We would be better off setting the precision to be no less than the number of output digits:

```
(32) -> digits(30);solve(p(x)=0,1.0e-20)
```

$$[x = 2.8473221018\_630726395]$$

```
                                  Type: List Equation Polynomial Float
(33) -> outputGeneral(10);
                                                            Type: Void
(34) -> complexSolve(p(x)=0,1.0E-10)
```

$$[x = 2.847322102,\ x = -1.423661051 - 0.283606001\%i,$$
$$x = -1.423661051 + 0.283606001\%i]$$

```
                          Type: List Equation Polynomial Complex Float
(35) -> radicalSolve([x^2-y^2=4,x*y=3],[x,y])
```

$$\left[\left[x = \frac{(\sqrt{-13}+2)\sqrt{-\sqrt{13}-2}}{3},\ y = \sqrt{-\sqrt{13}-2}\right],\right.$$
$$\left[x = \frac{(\sqrt{13}-2)\sqrt{-\sqrt{13}-2}}{3},\ y = -\sqrt{-\sqrt{13}-2}\right],$$
$$\left[x = \frac{(\sqrt{13}+2)\sqrt{\sqrt{13}-2}}{3},\ y = \sqrt{\sqrt{13}-2}\right],$$
$$\left.\left[x = \frac{(-\sqrt{13}-2)\sqrt{\sqrt{13}-2}}{3},\ y = -\sqrt{\sqrt{13}-2}\right]\right]$$

```
                           Type: List List Equation Expression Integer
(36) -> solve[x^2-y^2=4,x*y=3],1.0e-10)
```

$$[[y = -1.2671034983,\ x = -2.3676045437],\ [y = 1.2671034983,\ x = 2.3676045437]]$$

```
                             Type: List List Equation Polynomial Float
(37) -> solve([a*x+y-a*z=a^2,x+(a+1)*y-z=a+1,x+2*a*y-a*z=2-a],[x,y,z])
```

$$\left[\left[x = \frac{a^4+2\ a^3-4\ a+2}{a^3-2\ a+1},\ y = \frac{a}{a^2+a-1},\ z = \frac{2\ a^3+2\ a^2-4\ a+1}{a^3-2\ a+1}\right]\right]$$

```
                  Type: List List Equation Fraction Polynomial Integer
```

Note that solve will in general produce numeric solutions, unless the equations are linear. When the precision is given, the list of variables need not be given explicitly, as, according to the documentation: “The list of variables would be redundant information since there can be no parameters for the numerical solver.”

## 5 Calculus

As you would expect, Axiom has some deep calculus abilities. For example, we can start with every beginning student’s love, limits:

```
(38) -> limit((cos(x)-1)/x^2,x=0)
```

$$-\frac{1}{2}$$

```
                    Type: Union(OrderedCompletion Expression Integer,...)
(39) -> limit(x/sqrtx^2-1,x=%plusInfinity)
```

$$1$$

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

Note here the use of `%plusInfinity` which is positive infinity. Axiom also has `%minusInfinity`. Limits are also possible for complex variables, using `complexLimit`, and `%infinity` for complex infinity.

After limits there are derivatives, implemented in Axiom with D. A third optional parameter gives the order of the derivative:

```
(40) -> D(sqrt(x^2-1),x)
```

$$\frac{x}{\sqrt{x^2-1}}$$

```
                                             Type: Expression Integer
(41) -> D(sqrt(x^2-1),x,4)
```

$$\frac{-12\,x^3-3}{(x^6-3\,x^4+3\,x^2-1)\sqrt{x^2-1}}$$

```
                                             Type: Expression Integer
```

Integration is performed with the `integrate` command. Sometimes Axiom gets worried that an integrand may have a pole in the region of integration, in which case the addition of the string `"noPole"` will alleviate its fears:

```
(42) -> integrate(sqrt(x^2-3),x)
```

$$\frac{\left(6x\sqrt{x^2-3}-6x^2+9\right)\log\left(\sqrt{x^2-3}-x\right)+\left(-2x^3+3x\right)\sqrt{x^2-3}+2x^4-6x^2}{4x\sqrt{x^2-3}-4x^2+6}$$

```
                                 Type: Union(Expression Integer,...)
(43) -> integrate(sqrt(x^2-3),x=2..4,"noPole")
```

$$\frac{\left(24\sqrt{13}-87\right)\log\left(-8\sqrt{13}+29\right)-264\sqrt{13}+948}{32\sqrt{13}-116}$$

```
           Type: Union(f1; OrderedCompletion Expression Integer,...)
(44) -> numeric(%)
```

$$4.8157033664\_708774417$$

```
                                                         Type: Float
```

Axiom is the only open-source system which includes an implementation of the complete Risch decision algorithm for systematic integration. This means that the example first given by Manuel Bronstein:

$$\int\frac{x}{\sqrt{x^4+10x^2-96x-71}}\,dx$$

is managed easily by Axiom:

```
(45) -> integrate(x/sqrt(x^4+10*x^2-96*x-71),x)
```

$$-\frac{1}{8}\ln\Big(\left(x^6+15x^4-80x^3+27x^2-528x+781\right)\sqrt{x^4+10x^2-96x-71} - \left(x^8+20x^6-128x^5+54x^4-1408x^3+3124x^2+10001\right)\Big)$$

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

Note also that FriCAS has a larger library of special functions than Axiom, so is able to integrate a wider class of functions. For example, this integration:

```
(46) -> integrate(x*sin(x^3),x);
(47) -> normalize(%)
```

$$\frac{-\sqrt[6]{-1}^{4}\,\Gamma\left(\frac{2}{3},x^3\sqrt[6]{-1}^{3}\right)+\Gamma\left(\frac{2}{3},-x^3\sqrt[6]{-1}^{3}\right)}{6\sqrt[6]{-1}^{3}}$$

```
                                           Type: Expression(Integer)
```

Now for some series. There are several different commands for producing a power series, of which `series` is the most general. The following three commands show some of the power of Axiom's type system. Since the first two outputs are of type `UnivariatePuiseuxSeries`, arithmetic on them will produce an output also of that type. Thus the division command produces the series for $\tan(x)$:

```
(48) -> s:=series(sin(x),x=0)
```

$$x-\frac{1}{6}x^3+\frac{1}{120}x^5-\frac{1}{5040}x^7+\frac{1}{362880}x^9-\frac{1}{39916800}x^{11}+O(x^{12})$$

```
                              Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(49) -> c:=series(cos(x),x=0)
```

$$1-\frac{1}{2}x^2+\frac{1}{24}x^4-\frac{1}{720}x^6+\frac{1}{40320}x^8-\frac{1}{3628800}x^{10}+O(x^{11})$$

```
                              Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(50) -> s/c
```

$$x+\frac{1}{3}x^3+\frac{2}{15}x^5+\frac{17}{315}x^7+\frac{62}{2835}x^9+\frac{1382}{155925}x^{11}+O(x^{12})$$

```
                              Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
```

As an example of Axiom's calculus power, let's use the series command as a generating function. In particular we shall construct the Euler polynomials $E_n(x)$ defined by

$$\frac{2e^{xt}}{e^t+1}=\sum_{n=0}^{\infty}E_n(x)\frac{t^n}{n!}.$$

So first create the series, and then extract a particular polynomial from it:

```
(51) -> E:=series(2*exp(x*t)/(exp(t)+1),t=0)
```

$$\begin{aligned}&1+\frac{2\,x-1}{2}t+\frac{x^2-x}{2}t^2+\frac{4\,x^3-6\,x^2+1}{24}t^3+\frac{x^4-2\,x^3+x}{24}t^4\\&+\frac{2\,x^5-5\,x^4+5\,x^2-1}{240}t^5+\frac{x^6-3\,x^5+5\,x^3-3\,x}{720}x^6\\&+\frac{8\,x^7-28\,x^6+70\,x^4-84\,x^2+17}{40320}t^7\\&+\frac{x^8-4\,x^7+14\,x^5-28\,x^3+17\,x}{40320}t^8\\&+\frac{2\,x^9-9\,x^8+42\,x^6-126\,x^4+153\,x^2-31}{725760}t^9\\&+\frac{x^{10}-5\,x^9+30\,x^7-126\,x^5+255\,x^3-155\,x}{3628800}t^{10}+O(t^{11})\end{aligned}$$

```
                              Type: UnivariatePuiseuxSeries(Expression Integer,t,0)
(52) -> coefficient(E,10)*factorial(10)
```

$$x^{10}-5\,x^9+30\,x^7-126\,x^5+255\,x^3-155\,x$$

```
                                                         Type: Expression Integer
```

Finally, some differential equations. Note the use of the `solve` command—since the equation to be solved is a differential equation, `solve` produces an appropriate solution. The main thing to note here is the prior definition of the unknown function as an operator:

```
(53) -> y:=operator 'y;

                                                              Type: BasicOperator
```

```
(54) -> de:=D(y(x),x)-y(x)-2*x^2
```
$$y'(x) - y(x) - 2x^2$$

Type: Expression Integer

```
(55) -> solve(de=0,y,x=0,[1])
```
$$5e^x - 2x^2 - 4x - 4$$

Type: Union(Expression Integer,...)

```
(56) -> de2:=D(y(x),x)-x^2-y(x)^2
```
$$y'(x) - y(x)^2 - x^2$$

Type: Expression Integer

```
(57) -> seriesSolve(de2=0,y,x=0,[1])
```
$$1 + x + x^2 + \frac{4}{3}x^3 + \frac{7}{6}x^4 + \frac{6}{5}x^5 + \frac{37}{30}x^6 + \frac{404}{315}x^7 + \frac{369}{280}x^8 + \frac{428}{315}x^9 + \frac{1961}{1400}x^{10} + O(x^{11})$$

Type: UnivariateTaylorSeries(Expression Integer,x,0)

Note that the solution method sometimes gives results which require a bit more work:

```
(58) -> de3:=D(y(x),x,2)-5*D(y(x),x)+6*y(x)-3*x^2-x-4*exp(2*x)
```
$$y''(x) - 5y'(x) - 4e^{2x} + 6y(x) - 3x^2 - x$$

Type: Expression Integer

```
(59) -> sol:=solve(de3,y,x)
```
$$\left[\text{particular} = \frac{(-24(e^x)^2 - 6x^2 - 6x - 2)e^{3x} - 24x(e^x)^3e^{2x} + (9x^2 + 12x + 6)(e^x)^3}{6(e^x)^3}, \text{basis} = [e^{3x}, e^{2x}]\right]$$

Type: Union(Record(particular: Expression(Integer),basis: List(Expression(Integer))),...)

We can now try to simplify the particular solution:

```
(60) -> normalize(sol.particular)
```
$$\frac{(-24x - 24)(e^x)^2 + 3x^2 + 6x + 4}{6}$$

Type: Expression Integer

But notice that this particular solution still has an $e^{2x}$ term in it, which is part of the homogeneous solution. Although this is not mathematically wrong, it may be confusing. So:

```
(61) -> normalize(sol.particular)+exp(2*x)/4)
```
$$\frac{(-48x - 45)(e^x)^2 + 6x^2 + 12x + 8}{12}$$

Type: Expression Integer

## 6 Lists and Matrices

Lists are a standard data structure in most CAS's, and Axiom is no exception. A list is delineated with square brackets, and can either be defined by listing it in full, or by iterating a function over a range of integers.

Given a list X, its i-th element is obtained with X.i. The first examples show the creation of a list, and some operations on it:

```
(62) -> L:=[invmod(4*i,97) for i in 1..20]
```

$$[73, 85, 89, 91, 34, 93, 52, 94, 62, 17, 86, 95, 28, 26, 76, 47, 10, 31, 60, 57]$$

```
                                   Type: List(Union(DoubleFloat,Integer))
(63) -> [L.(2*i) for i in 1..10]
```

$$[85, 91, 93, 94, 17, 95, 26, 47, 31, 57]$$

```
                                   Type: List(Union(DoubleFloat,Integer))
(64) -> reduce(+,L)
```

$$1206$$

```
                                                   Type: PositiveInteger
(65) -> reduce(*,L)
```

$$18439941734502408142997068523520000$$

```
                                                   Type: PositiveInteger
(66) -> reduce(max, L)
```

$$95$$

```
                                                   Type: PositiveInteger
(67) -> map(x+->x^2+1,L)
```

$$[5330.0, 7226.0, 7922.0, 8282.0, 1157.0, 8650.0, 2705.0, 8837.0, 3845.0, 290.0, 7397.0, 9026.0, 785.0, 677.0, 5777.0, 2210.0, 101.0, 962.0, 3601.0, 3250.0]$$

```
                                                 Type: List(DoubleFloat)
```

There are a few things to note here:

- `reduce` extends a binary operation to a list, working its way through the list taking an extra element each time.
- The final operation shows an example of an "anonymous function"; that is, a function which hasn't been given a name. The use of the `+->` is useful when you want to use a function "on the fly" or when you are only to going to need it once. The last operation could have been done with

  ```
  f(x)==x^2+1
  map(f,L)
  ```

  but that would require defining the function first.

Matrices are defined as lists of lists. All arithmetic is supported, and inversion, if the matrix is square and non-singular.

```
(68) -> A:=matrix [[1,2,3],[4,5,6],[7,8,10]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

```
                                                   Type: Matrix(Integer)
```

```
(69) -> B:=matrix [[-3,2,1],[4,-5,6],[-9,8,7]]
```

$$\begin{bmatrix} -3 & 2 & 1 \\ 4 & -5 & 6 \\ -9 & 8 & 7 \end{bmatrix}$$

```
                                                     Type: Matrix(Integer)
(70) -> 2*A+3*B
```

$$\begin{bmatrix} -7 & 10 & 9 \\ 20 & -5 & 30 \\ -13 & 40 & 41 \end{bmatrix}$$

```
                                                     Type: Matrix(Integer)
(71) -> A^3*B^2
```

$$\begin{bmatrix} -50712 & 43176 & 83952 \\ -114342 & 97353 & 189264 \\ -189300 & 161174 & 313336 \end{bmatrix}$$

```
                                                     Type: Matrix(Integer)
(72) -> B^-1
```

$$\begin{bmatrix} -\frac{83}{72} & -\frac{1}{12} & \frac{17}{72} \\ -\frac{41}{36} & -\frac{1}{6} & \frac{11}{36} \\ -\frac{13}{72} & \frac{1}{12} & \frac{7}{72} \end{bmatrix}$$

```
                                           Type: Matrix(Fraction(Integer))
```

We can also obtain this last in numeric form:

```
(73) -> digits(6);outputGeneral(6);
                                                                Type: Void
(74) -> map(numeric,B^-1)
```

$$\begin{bmatrix} -1.15278 & -0.0833333 & 0.236111 \\ -1.13889 & -0.166667 & 0.305556 \\ -0.180556 & 0.0833333 & 0.0972222 \end{bmatrix}$$

```
                                                       Type: Matrix(Float)
```

Axiom can do all the things you'd expect on matrices: find the determinant and permanent, eigenvalues and eigenvectors, row-echelon form, null space... To finish here, a little example of the Cayley-Hamilton theorem, and a solution of a set of linear equations, using matrices:

```
(75) -> cp:=characteristicPolynomial(B,x)
```

$$-x^3 - x^2 + 88x + 72$$

```
                                                 Type: Polynomial(Integer)
(76) -> eval(cp,x=B)
```

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

```
                               Type: Polynomial SquareMatrix(3,Integer)
(77) -> solve(B,[-1,21,11])
```

$$[\text{particular} = [2,1,3], \text{basis} = [[0,0,0]]]$$

```
      Type: Record(particular: Union(Vector Fraction Integer,''failed''),basis:
```

```
List Vector Fraction Integer)
```

The output of the last command is given using Axiom's `Record` data structure. To isolate individual parts of this output, we refer to them by name. So to obtain the $y$ value (the second value), we would use

```
(78) -> %.particular.2

        1

                                                   Type: Fraction(Integer)
```

Here the `particular` is a particular example of a solution, and `basis` the basis for the set of solutions. This is an easy and elegant way of describing multiple solutions to a linear system.

## 7 Graphics

Moving right on, we now provide a brief introduction to Axiom's graphics.

To obtain graphics, you need to run Axiom in Linux, or in windows within an X-windows system, such as Xming. At the time of writing, there is no native windows graphics subsystem for Axiom. You can also run FriCAS in a web-based notebook interface, using the Python system Jupyter. This system provides typeset output, and also embedded graphics.

Having said that, Axiom's graphics are sophisticated and powerful. In the plane you can draw:

- graphs of functions $y = f(x)$
- parametric graphs $(x(t), y(t))$
- graphs defined algebraically $p(x, y) = 0$
- graphs defined with polar, bipolar, elliptic or parabolic coordinates

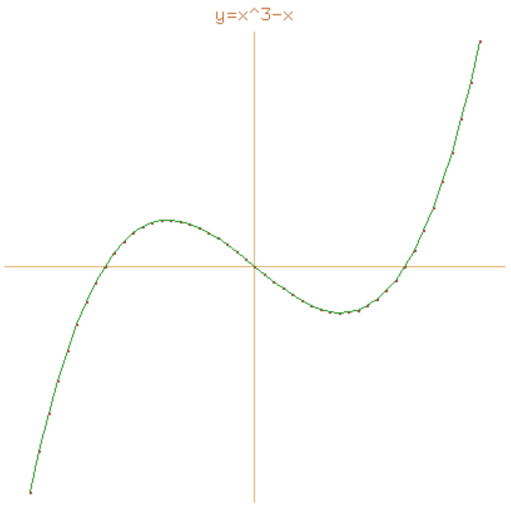
In three-dimensions, you can have

- surfaces defined by $z = f(x, y)$
- parametric space curves $(x(t), y(t), z(t))$
- surfaces defined parametrically $(x(s, t), y(s, t), z(s, t))$
- curves or surfaces defined with other coordinate systems, including cylindrical, spherical, paraboloidal, conical

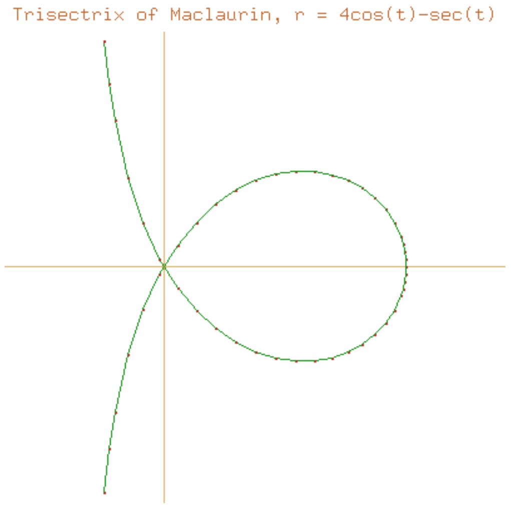
The command to produce any such graphic is `draw`. The graphic can be changed in numerous ways, either by adding a parameter to the draw command, or by using the associated control panel.

Here are two elementary examples: plane curves, one the graph of a cubic polynomial in cartesian coordinates, the other a trisectrix given in polar coordinates. The commands for each are

```
(79) -> draw(x^3-x,x=-1.5..1.5,title=="y=x^3-x");
                                              Type: TwoDimensionalViewport
(80) -> tr(x)==4*cos(x)-sec(x)
                                                                Type: Void
```



Graph of a cubic polynomial



Graph of a trisectrix

Figure 1: Graphs of plane curves in Axiom

```
(81) ->   draw(tr(x),coordinates==polar,x=-1.5..1.5,toScale==true,_
          clip==[-1..3,-3..3],_
          title=="Trisectrix of Maclaurin, r = 4cos(t)-sec(t)")
                                             Type: TwoDimensionalViewport
```

These curves are shown in figure 1.

In Axiom there are options for changing the colour of the curve, the number of data points used to ploat the curve, whether those points are shown or now, and whether the axes are to be shown, the plot drawn at scale, or with the $x$ and $y$ values clipped. These can be included in the `draw` command as extra parameters.

Associated with any such graphic is a control panel, which can be displayed by clicking on the graphic, and is shown in figure 2. The panel allows you to change the size and position of the graphic, whether or not you want points shown (the default is to show points on the curve), whether or not you want a bounding box, or axes, and many other adjustments.

Now for some three-dimensional surfaces, each one given in the form $y = f(x, y)$. The first is Scherk's minimal surface, defined implicitly by $e^z \cos x = \cos y$, or explicitly as $z = \log(\cos(y)/\cos(x))$, and the second is the "monkey saddle" surface defined by $z = x^3 - 3xy^2$. The commands to produce them are

```
(82) -> draw(log(abs(cos(y)/cos(x))),x=-1.5..1.5,y=-1.5..1.5);
                                           Type: ThreeDimensionalViewport
(83) -> draw(x^3-3*x*y^2,x=-1..1,y=-1..1);
                                           Type: ThreeDimensionalViewport
```

and they are shown in figure 3

As for 2D graphics, there is a control panel for 3D graphics, which allows you to change much of the appearance of the graphic, as well as its orientation, as shown on the left in figure 4. A second control panel, accessed from the first, allows you to change the bounds of the graphic, and its perspective; this is shown on the right in figure 4.

It's quite easy to build a complicated graphic by plotting several surfaces or shapes on the one set of axes. The following commands illustrate how to set up a graphic consisting of two

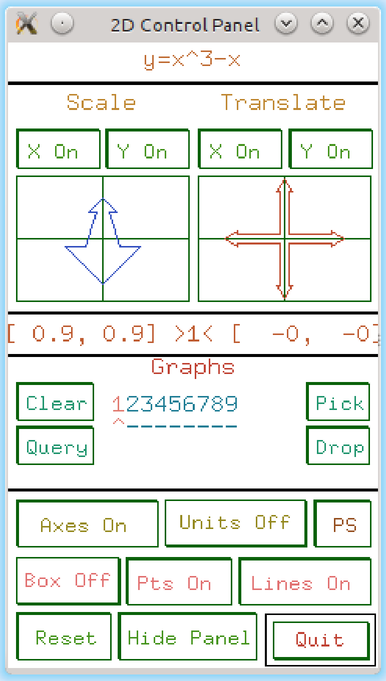


Figure 2: 2D control panel for Axiom graphics

interlocked tori:

```
(84) -> s:=create3Space()$(ThreeSpace DFLOAT)
```

3-Space with 0 components

Type: ThreeSpace DoubleFloat

```
(85) -> draw(curve(0,2*cos(t)+2,2*sin(t)),t=0..2*%pi,tubeRadius==0.3,space==s)
```

ThreeDimensionalViewport: "0"

Type: ThreeDimensional Viewport

```
(86) -> draw(curve(2*cos(t),2*sin(t),0),t=0..2*%pi,tubeRadius==0.3,_
        space==s, title=="Interlocked Tori")
```

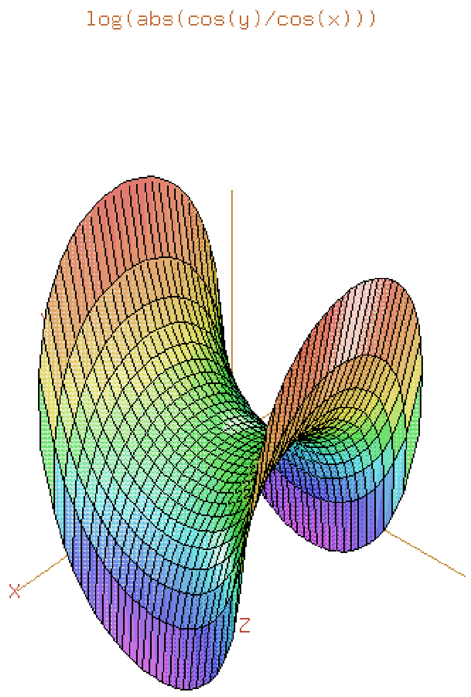
ThreeDimensionalViewport: "Interlocked Tori"

Type: ThreeDimensional Viewport

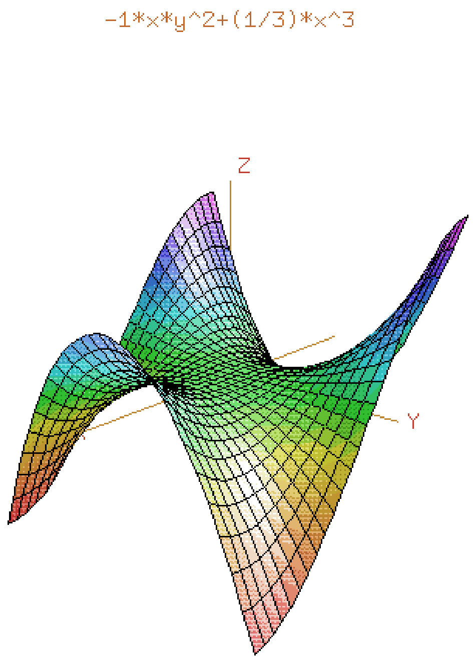
and the result is shown in figure 5.

Finally, a couple of knots; first a trefoil knot, constructed parametrically with

$$x(t) = \sin(t) + 2\sin(2t)$$
$$y(t) = \sin(3t)$$
$$z(t) = \cos(t) - 2\cos(2t)$$



Scherk's minimal surface



Monkey saddle

Figure 3: Graphs of surfaces in Axiom

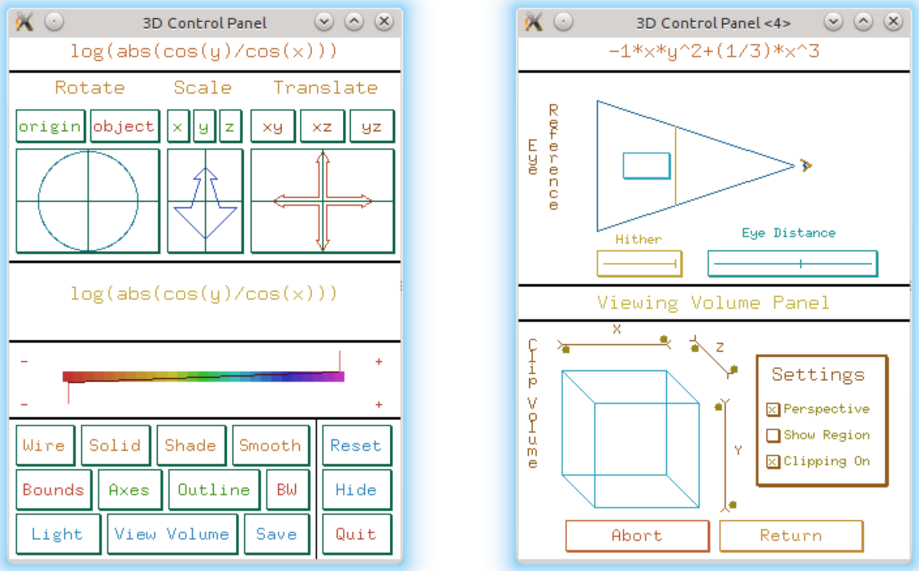


Figure 4: 3D control panels for Axiom graphics

and a figure of eight knot with

$$x(t) = (2 + \cos(4t))\cos(6t)$$
$$y(t) = (2 + \cos(4t))\sin(6t)$$
$$z(t) = 2\sin(8t)$$

(I found these equations at `http://www.infradead.org/~wmp/proj_knots.html`.) They can be written into Axiom as

```
(87) -> draw(curve(sin(t)+2*sin(2*t),sin(3*t),cos(t)-2*cos(2*t)),_
        tubePoints==19,t=0..2*%pi,tubeRadius==0.3,title=="Trefoil knot")
        ThreeDimensionalViewport : "Trefoil knot"
                                         Type: ThreeDimensionalViewport
(88) -> draw(curve((2+cos(4*t))*cos(6*t),(2+cos(4*t))*sin(6*t),2*sin(8*t)),_
        tubePoints==19,t=-%pi..%pi,tubeRadius==0.3,title=="Figure of eight knot")

        ThreeDimensionalViewport : "Figure of eight knot"
                                         Type: ThreeDimensionalViewport
```

and are shown in figure 6.

## 8 Types

So far, Axiom does most of the calculation we would expect of any computer algebra system. But now we look at an aspect of Axiom which sets it apart from other systems: its use of types. In Axiom, "type" is another word for "domain", or "domain of computation", which is a method

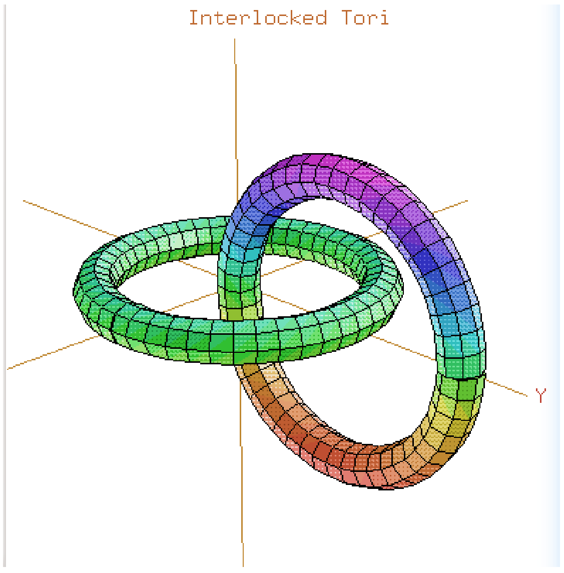


Figure 5: Interlocked tori

of organizing all objects. Objects include standard mathematical constructs such as matrices, functions, integers, abelian groups; data structures such as lists and arrays; as well as constructs built up from other constructs. Thus in Axiom we can create and use the type: "List of square matrices of polynomials over integer fractions". The type of the operand indicates the type of the operation: a simple multiplication * will have a different interpretation depending on whether the operands are integers, real numbers (floats), matrices, elements of a finite field, or many other types.

There are three special symbols which are used when dealing with types:

**::** This is used for converting from one type to another. In the form

```
object::type
```

it takes the object object and converts it to type type.

**$** This means to take the operation as defined for objects of the given type. It is used in the form

```
(operand1 operation operand2)$type
```

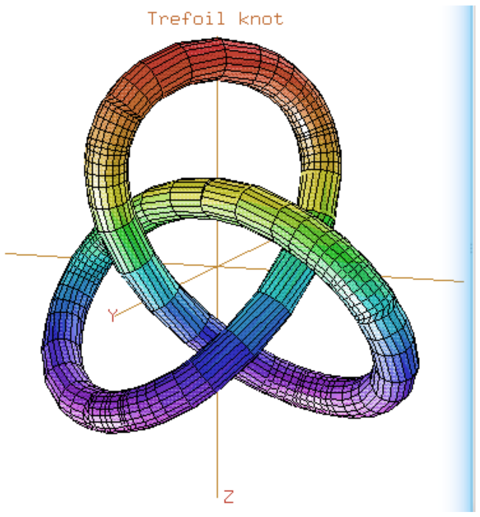
**@** This means to choose the operation (and possibly convert the types of the objects, if needed) so that the result is of the given type.

As examples of the use and misuse of these symbols, let's investigate a modular power:

$$a^n \bmod p$$

Such operations form much of the basis of modern cryptography. Start by defining a prime $p$ and an exponent $n$:

```
(89) -> reseed(1)
                                                    Type: Void
```



Trefoil knots

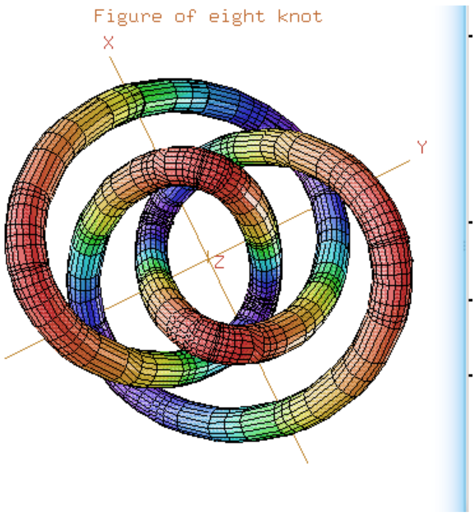


Figure-of-eight knot

Figure 6: Knots in Axiom

```
(90) -> p:=nextPrime(randnum(10^10))
        8537081929
                                         Type: PositiveInteger
(91) -> n:=randnum(10^10
        1840983108
                                         Type: PositiveInteger
```

The use of `reseed` at the beginning ensures that if you try this, you will obtain the same random numbers. Now we will attempt to compute the above modular power with $a = 7$:

```
(92) -> a:=7;
                                         Type: PositiveInteger
(93) -> (a^n)@ZMOD(p)
        3213930103
                                   Type: IntegerMod(8537081929)
(94) -> (a::ZMOD(p))^n
        3213930103
                                   Type: IntegerMod(8537081929)
```

In each of these commands, the power operation is performed using its modular version, which is both highly efficient and only uses small numbers. However:

```
(95) -> (a^n)::ZMOD(p)
```

will just hang, and produce no result, and possibly even cause Axiom to crash. This command is actually asked Axiom to first compute `a^n` (which is a huge number), and then reduce that number modulo $p$.

Here's another example of type conversion, this time using polynomials over prime fields. The abbreviation for the finite field of residues modulo a prime $p$ is `PF(p)`.

```
(96) -> p:=1+x+x^2+x^3+x^4
```
$$x^4+x^3+x^2+x+1$$

Type: Polynomial Integer

```
(97) -> factor(p)
```
$$x^4+x^3+x^2+x+1$$

Type: Factored Polynomial Integer

```
(98) -> factor(p::POLY PF 5)
```
$$(x+4)^5$$

Type: Factored Polynomial PrimeField 5

```
(99) -> factor(p::POLY PF 11)
```
$$(x+2)(x+6)(x+7)(x+8)$$

Type: Factored Polynomial PrimeField 11

```
(100) -> factor(p::POLY PF 19)
```
$$(x^2+5x+1)(x^2+15x+1)$$

Type: Factored Polynomial PrimeField 19

Notice that we needed to change the type of `p` to `POLY PF p`: that is, to Polynomials over the Prime Field $p$. If we tried to write, for example

```
factor(p::PF 5)
```

we would obtain the error

```
    Cannot convert the value from type Polynomial(Integer) to PrimeField
(5).
```

Now some examples with matrices:

```
(101) -> A:=matrix [[1,2,3],[4,5,6],[7,8,10]]
```
$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 10\end{bmatrix}$$

Type: Matrix Integer

```
(102) -> A^(-1)
```
$$\begin{bmatrix}-\frac{2}{3} & -\frac{4}{3} & 1\\ -\frac{2}{3} & \frac{11}{3} & -2\\ 1 & -2 & 1\end{bmatrix}$$

Type: Matrix Fraction Integer

```
(103) -> (A::MATRIX PF 11)^(-1)
```
$$\begin{bmatrix}3 & 6 & 1\\ 3 & 0 & 9\\ 1 & 9 & 1\end{bmatrix}$$

Type: Matrix PrimeField 11

```
(104) -> (A::MATRIX PF 101)^(-1)
```
$$\begin{bmatrix}50 & 54 & 30\\ 95 & 88 & 71\\ 4 & 33 & 8\end{bmatrix}$$

Type: Matrix PrimeField 101

```
(105) -> A*%
```

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Type: `Matrix PrimeField 101`

Note that again we had to give the full type of A. If we tried to convert A to type "`PF 11`" Axiom would return an error message saying that the particular conversion is not possible.

Recall earlier on that an expansion of a multivariate polynomial treats the result as a polynomial in one variable, whose coefficients are polynomials in the other variables:

```
(106) -> expand((x-2*y+3*z)^3
```

$$27z^3 + (-54y + 27x)z^2 + (36y^2 - 36xy + 9x^2)z - 8y^3 + 12xy^2 - 6x^2y + x^3$$

Type: `Polynomial(Integer)`

To obtain a "full" expansion, we have to use the `DistributedMultivariatePolynomial` type, which is abbreviated as `DMP`:

```
(107) -> expand((x-2*y+3*z)^3)::DMP([x,y,z],INT)
```

$$x^3 - 6xy^2 + 9x^2z + 12xy^2 - 36xyz + 27xz^2 - 8y^3 + 36y^2z - 54yz^2 + 27z^3$$

Type: `DistributedMultivariatePolynomial([x,y,z],Integer)`

Finally, some examples of "package calling"—calling a command with the same name (in this case `random`) but from different packages. The first command requests a random integer; the second a random residue from the prime field $\mathbb{Z}/31\mathbb{Z}$, then random element from the finite field $\mathbb{Z}_2[x]/(x^6 + x + 1)$, and a random square matrix over a prime field:

```
(108) -> random()$INT
```

$$26799862$$

Type: `Integer`

```
(109) -> random()$PF 31
```

$$7$$

Type: `PrimeField(31)`

```
(110) -> random()$FFP(PF 2,x^6+x+1
```

$$\%A^5 + \%A^3 + \%A^2$$

Type: `FiniteFieldExtensionByPolynomial(PrimeField(2),?^6+?+1)`

```
(111) -> random()$SQMATRIX(4,PF 31)
```

$$\begin{bmatrix}22 & 12 & 7 & 16\\ 5 & 19 & 16 & 22\\ 1 & 24 & 20 & 8\\ 8 & 3 & 18 & 28\end{bmatrix}$$

Type: `SquareMatrix(4,PrimeField(31))`

## 9 Programming

Axiom has a full and complete, and well-structured programming language. Rather than describe it in detail, I'll just make a few general remarks, and provide a few examples.

### A simple function

Here's an example of a simple program, to solve an equation numerically by the bisection method. We implement the following algorithm:

```
**Require:** function $f(x)$, values $a$,$b$ for which $f(a)f(b) < 0$, a tolerance $\epsilon$
  **while** $|f(a)| > \epsilon$ **do**
      $c = (a + b)/2$
      **if** $\mathrm{sgn}(f(a)) = \mathrm{sgn}(f(c))$ **then**
          $a = c$
      **else**
          $b = c$
      **end if**
  **end while**
  **return** $a$
```

For this algorithm at least, the program is very similar in Axiom:

```
bisect(f,a,b,eps) ==
  local c
  while abs(f(a))>eps repeat
    c:=numeric (a+b)/2
    if sign(f(a))=sign(f(c)) then
      a:=c
    else
      b:=c
  return(a)
```

Note that for ease of exposition I've done no sign checking on the values of $f(a)$ and $f(b)$. This little program can be saved in a file called `bisect.input` and read into Axiom with the command

```
     )read bisect
```

And now it can be used:

```
(112) -> bisect(x+->x^5+x^2-1,0,1,1.0e-12)
```

$$0.8087306004\_7929357097$$

Type: Float

Note the use of anonymous functions.

Axiom programs are made of blocks, which are groups of statements all with the same indentation, or alternatively delineated with parentheses; the statements being separated with semicolons. The above program could be entered as

```
bisect2(f,a,b,eps) == (while abs(f(a))>eps repeat (c:=numeric (a+b)/2;
if sign(f(a))=sign(f(c)) then a:=c else b:=c),return a)
```

This is equally correct, but less readable.

Axiom contains the usual programming constructs: loops with for and while, branching, and recursion. The above program shows several of these.

## An algebraic problem

If you factorize $x^n - 1$ for a few values of $n$, each factor seems to have non-zero coefficients only of 1 or $-1$. This might lead us to conjecture that this is the case for all $n$. We can write some functions and programs to test this. First, the maximum coefficient of a polynomial:

```
maxCoeff(p) ==
  reduce(max,map(abs,coefficients(p)))
```

Next, the maximum coefficient taken over all factors of a polynomial:

```
maxFactCoeff(q) ==
  reduce(max,[maxCoeff(nthFactor(q,j)) for j in 1..numberOfFactors(q)])
```

Finally, the “driver” program which simply lists all values of $n$ for which the factorization of $x^n - 1$ contains a factor one of whose coefficients has an absolute value greater than 1:

```
testCoeff() ==
  for k in 1.. repeat
    if maxFactCoeff(factor(x^(k::PI)-1))>1 then print(k)
```

Put all these into a single file, called say `maxFactor.input`, read it into Axiom with

```
(113) -> )read maxFactor
```

and run it with

```
(114) -> testCoeff()
```

You will obtain the following values:

```
105
165
195
210
255
273
285
315
330
345
357
```

and so on, until either you hit CTRL-c to abort the program, or you hit your computer with a heavy brick. (I suggest the first method).

This problem can be equivalently stated in terms of the coefficients of cyclotomic polynomials, and in fact the sequence generated above is sequence A013590 in the Online Encyclopaedia of Integer Sequences: “Orders of cyclotomic polynomials containing a coefficient >= 2.”

## 10 Documentation

Like all Computer Algebra Systems, Axiom has both online documentation, and various other web pages devoted to its description.

### HyperDoc

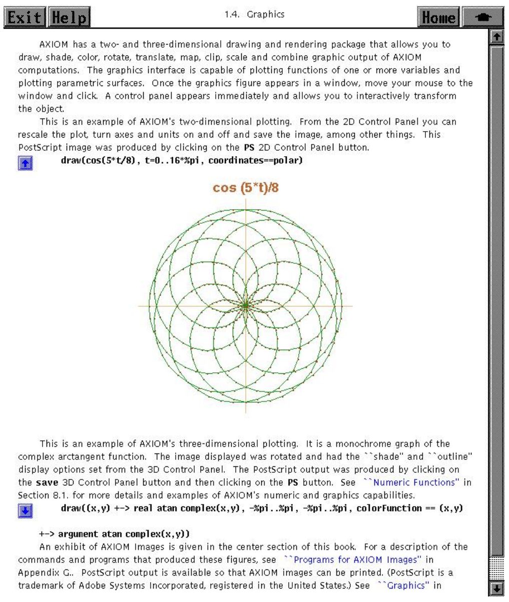
If you are using Axiom under linux, or under Windows with the Xming X-server, you will have access to the HyperDoc help browser. Although this can look a little old-fashioned (in terms of its widgets), it is extraordinarily powerful, and very useful. It contains:

- the complete text of the book ”Axiom: The Scientific Computation System” by Richard Jenks and Robert Sutor (Springer, 1992), in a convenient hypertext format
- masses of examples

- a useful search facility
- a browse system, where you can browse Axiom by commands, or libraries. For example, you can find all the commands which apply to finite fields, what they do, and how they can be used.

HyperDoc is started automatically when you start Axiom. Although HyperDoc can be used as a stand-alone help browser, it connects with Axiom using a socket, so it can run Axiom commands.

Figure 7 shows a couple of screenshots; one with graphics, one with algebra.



AXIOM has a two- and three-dimensional drawing and rendering package that allows you to draw, shade, color, rotate, translate, map, clip, scale and combine graphic output of AXIOM computations. The graphics interface is capable of plotting functions of one or more variables and plotting parametric surfaces. Once the graphics figure appears in a window, move your mouse to the window and click. A control panel appears immediately and allows you to interactively transform the object.

This is an example of AXIOM's two-dimensional plotting. From the 2D Control Panel you can rescale the plot, turn axes and units on and off and save the image, among other things. This PostScript image was produced by clicking on the **PS** 2D Control Panel button.

```
draw(cos(5*t/8), t=0..16*%pi, coordinates==polar)
```

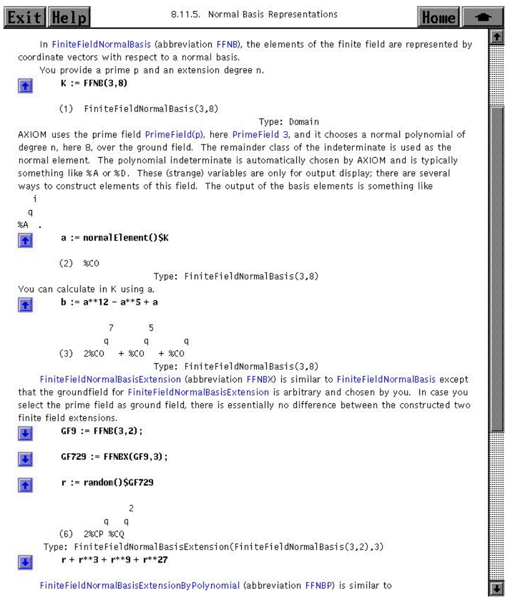
This is an example of AXIOM's three-dimensional plotting. It is a monochrome graph of the complex arctangent function. The image displayed was rotated and had the ``shade'' and ``outline'' display options set from the 3D Control Panel. The PostScript output was produced by clicking on the **save** 3D Control Panel button and then clicking on the **PS** button. See ``Numeric Functions'' in Section 8.1. for more details and examples of AXIOM's numeric and graphics capabilities.

```
draw((x,y) +-> real atan complex(x,y), -%pi..%pi, -%pi..%pi, colorFunction == (x,y)

+-> argument atan complex(x,y))
```

An exhibit of AXIOM Images is given in the center section of this book. For a description of the commands and programs that produced these figures, see ``Programs for AXIOM Images'' in Appendix G.. PostScript output is available so that AXIOM images can be printed. (PostScript is a trademark of Adobe Systems Incorporated, registered in the United States.) See ``Graphics'' in

Graphics in HyperDoc



In FiniteFieldNormalBasis (abbreviation FFNB), the elements of the finite field are represented by coordinate vectors with respect to a normal basis.

You provide a prime p and an extension degree n.

```
K := FFNB(3,8)

(1)  FiniteFieldNormalBasis(3,8)
                                    Type: Domain
```

AXIOM uses the prime field PrimeField(p), here PrimeField 3, and it chooses a normal polynomial of degree n, here 8, over the ground field. The remainder class of the indeterminate is used as the normal element. The polynomial indeterminate is automatically chosen by AXIOM and is typically something like %A or %D. These (strange) variables are only for output display; there are several ways to construct elements of this field. The output of the basis elements is something like

```
  i
 q
%A  .
```

```
a := normalElement()$K

(2)  %CO
                     Type: FiniteFieldNormalBasis(3,8)
```

You can calculate in K using a.

```
b := a**12 - a**5 + a

           7       5
          q       q       q
(3)  2%CO   + %CO   + %CO
                     Type: FiniteFieldNormalBasis(3,8)
```

FiniteFieldNormalBasisExtension (abbreviation FFNBX) is similar to FiniteFieldNormalBasis except that the groundfield for FiniteFieldNormalBasisExtension is arbitrary and chosen by you. In case you select the prime field as ground field, there is essentially no difference between the constructed two finite field extensions.

```
GF9 := FFNB(3,2);

GF729 := FFNBX(GF9,3);

r := random()$GF729

               2
          q   q
(6)  2%CP %CQ
Type: FiniteFieldNormalBasisExtension(FiniteFieldNormalBasis(3,2),3)
r + r**3 + r**9 + r**27
```

FiniteFieldNormalBasisExtensionByPolynomial (abbreviation FFNBP) is similar to

Algebra in HyperDoc

Figure 7: Hyperdoc

## Other help from within Axiom

If you only have a console based Axiom, then there is limited help available from within Axiom itself. However, there are a few commands you can use to provide some information. Suppose, for example, you wish to list all operations containing the string "diff". Then you enter

```
)wh op diff
```

which is an abbreviation for

```
)what operations diff
```

and receive the output

```
Operations whose names satisfy the above pattern(s):

diff difference differentialVariables differentiate exteriorDifferential
```

```
mergeDifference setDifference symmetricDifference totalDifferential

To get more information about an operation such as differentiate,
issue the command

)display op differentiate
```

To list domains, categories or packages, you can write

```
)wh do diff
)wh ca diff
)wh pa diff
```

The command

```
)wh th diff
```

which is an abbreviation for

```
)what things diff
```

lists everything which contains the substring “`diff`”.

Now suppose we want to find out about the operation ”difference”:

```
)d op difference
```

which is an abbreviation for

```
)display operation difference
```

and which produces

```
There are 2 exposed functions called difference :
   [1] (D,D1) -> D from D if D has SETAGG D1 and D1 has SETCAT
   [2] (D,D) -> D from D if D has SETAGG D1 and D1 has SETCAT
```

If you want to find out what ”difference” actually does, without using HyperDoc, and from within Axiom itself, you can’t.

However, a very handy command is show, which when applied to a domain lists all operations defined on that domain. Here for exmaple, we find out about Cliiford alegbras, whose type has abbreviation CLIF:

```
)sh CLIF
```

with output

```
CliffordAlgebra(n: PositiveInteger,K: Field,Q: QuadraticForm(n,K))
 is a constructor
 Abbreviation for CliffordAlgebra is CLIF
 This constructor is exposed in this frame.
 Issue )edit /usr/local/axiom/mnt/linux/../../src/algebra/CLIF.spad
to see source code for CLIF
```

```
------------------------------ Operations ------------------------------
 ?*? : (%,K) -> %                     ?*? : (K,%) -> %
 ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
 ?+? : (%,%) -> %                     ?-? : (%,%) -> %
 -? : % -> %                          ?/? : (%,K) -> %
 ?=? : (%,%) -> Boolean               1 : () -> %
 0 : () -> %                          ?^? : (%,PositiveInteger) -> %
 coerce : K -> %                      coerce : Integer -> %
 coerce : % -> OutputForm             dimension : () -> CardinalNumber
 e : PositiveInteger -> %             hash : % -> SingleInteger
 latex : % -> String                  one? : % -> Boolean
 recip : % -> Union(%,"failed")       sample : () -> %
 zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 characteristic : () -> NonNegativeInteger
 coefficient : (%,List PositiveInteger) -> K
 monomial : (K,List PositiveInteger) -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
```

It seems that we can add, subtract, multiply and divide elements in a Clifford algebra, as well as raising an element to an integer power. As well as other operations.

## Local help outside Axiom

If you have a complete Axiom system, or have compiled it from scratch and also compiled the documentation, you'll find some useful files in the

```
/usr/share/doc/axiom-doc
```

directory, of which the most important and useful are:

**rosetta.dvi** This is the "Rosetta Stone" and provides a "collection of synonyms for various operations in the computer algebra systems Axiom, Derive, GAP, Gmp, DoCon, Macsyma, Magnus, Maxima, Maple, Mathematica, MuPAD, Octave, Pari, Reduce, Scilab, Sumit and Yacas." Without being comprehensive, it is nonetheless an incredibly useful document, especially if you're coming to a new CAS after experience with another.

**book.dvi** This is the Jenks/Sutor book, with additions—1136 pages! Not for printing!

**bookvol1.dvi** This is a Tutorial volume, and a very good starting place for learning about Axiom. It exists also in printed form, available from lulu.com.

## Online help

There is also some very good material online. First off is an xhtml version of the Jenks/Sutor book, at `http://axiom-wiki.newsynthesis.org/uploads/contents.xhtml`

Complete pdf files of all Axiom user documentation are available at `http://www.axiom-developer.org/axiom-website/documentation.html` This page also contains a link to the beginnings of a new browser-based interface to the HyperDoc material.

For tutorials, the best one in English is by Martin Dunstan, from as long ago as 1996, at `http://www.dcs.st-and.ac.uk/~mnd/documentation/axiom_tutorial`.

There are some excellent tutorials in French, by Daniel Augot at `http://www-rocq.inria.fr/codes/Daniel.Augot/axiom_intro.pdf` and by Christophe Conil and Quentin Carpent at `http://la.riverotte.free.fr/axiom`.

Now - get out there and learn about Axiom!