

# Maxima User Manual

The Maxima Documentation Team

10-5-2001

# Contents

<b>I</b>	<b>The Maxima Program and Standard Packages</b>	<b>4</b>
<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	An Overview of Available Mathematical Programs . . . . .	5
1.1.1	An Overview of Current Systems . . . . .	6
1.1.2	Free vs. Commercial - Do I get what I pay for? . . . . .	6
1.2	The History of Macsymba . . . . .	7
<b>2</b>	<b>Installing</b>	<b>8</b>
2.1	Installing on Linux . . . . .	8
2.1.1	The Easy Way . . . . .	8
2.1.2	The Hard Way . . . . .	8
2.2	Installing on Windows . . . . .	10
2.2.1	The Easy Way - Installing the Windows Binary . . . . .	10
2.2.2	The Hard Way . . . . .	11
<b>3</b>	<b>Available Interfaces to Maxima</b>	<b>12</b>
3.1	The Terminal Interface . . . . .	12
3.2	The Emacs Interface . . . . .	12
3.3	Xmaxima . . . . .	14
3.4	Symaxx . . . . .	14
3.5	T <sub>E</sub> Xmacs . . . . .	14
<b>4</b>	<b>The Basics - What you need to know to operate in Maxima</b>	<b>19</b>
4.1	The Very Beginning . . . . .	19
4.1.1	Our first Maxima Session . . . . .	19
4.1.2	To Evaluate or Not to Evaluate . . . . .	23
4.1.3	The Concept of Environment - The <code>ev</code> Command . . . . .	23
4.2	Common Operators in Maxima . . . . .	27
4.2.1	Assignment Operators . . . . .	27
<b>5</b>	<b>Trig through Through Calculus</b>	<b>30</b>
<b>6</b>	<b>Advanced Mathematics - ODEs and Beyond</b>	<b>31</b>
<b>7</b>	<b>Matrix Operations and Vectors</b>	<b>32</b>
<b>8</b>	<b>Programming in Maxima</b>	<b>33</b>
<b>9</b>	<b>Graphics and Forms of Output</b>	<b>34</b>

<i>CONTENTS</i>	2
<b>10 Help Systems and Debugging</b>	<b>35</b>
<b>11 Troubleshooting</b>	<b>36</b>
<b>II External, Additional, and Contributed Packages</b>	<b>39</b>
<b>12 The Concept of Packages - Expanding Maxima's Abilities</b>	<b>40</b>
<b>13 Available Packages for Maxima</b>	<b>41</b>

**Examples**

1. First Example	19	6. Evaluation Toggle	23
2. Quitting Maxima	20	7. Basic Use of ev Command	23
3. End of Entry Characters	21	8. ev's Expand Option	24
4. Line Labels	21	9. Float Example	24
5. Labeling an Example Equation	22		

## **Part I**

# **The Maxima Program and Standard Packages**

# Chapter 1

## Introduction

### 1.1 An Overview of Available Mathematical Programs

Maxima is only one of a large number of programs, both free and commercial, which exist to serve the needs of researchers, scientists, and students. Most of these programs are commercial, and some are expensive in the extreme, but much is to be gained by knowing what the right tool is for a job, and when to use it. Hence we will attempt to provide an overview here of what programs exist out there and what they have been designed to do. Whether or not they are suitable for an particular project must be decided on a case by case basis.

#### The Dangers of Computer Algebra

Those considering the use of computers to do mathematics, particularly students, must be warned that these systems are no substitute for hands on work with equations and struggling with concepts. These systems do not build your mathematical intuition, nor will they strengthen your core skills. This will matter a great deal down the road, especially to those of you who wish to break new ground in theoretical mathematics and science. Do not use a computer as a substitute for your basic education.

By the same token, however, proficiency with computers and computer based mathematics is crucial for attacking the many problems which literally cannot be solved by pencil and paper methods. In many cases problems which would take years by hand can be reduced to seconds by powerful computers. Also, in the course of a long derivation, it is sometimes useful for those who have already mastered the fundamentals to do work in these systems as a guard against careless errors, or a faster means than a table of deriving some particular result. Also, in case of an error, fixing the resulting error can often be much quicker and simpler courtesy of a mathematical notebook, which can be reevaluated with the correct parameters in place.

But just as a computer can guard against human error, the human must not trust the computer unquestioningly. All of these systems have limits, and when those limits are reached it is quite possible for bizarre errors to result, or in some cases answers which are actually wrong, to say nothing of the fact that the people who programmed these systems were human, and make mistakes. To illustrate the limits of computer algebra systems, we note a discussion on the Maxima developers list in which it was pointed out that when given the integral (give example from list, after getting permission). Had the person who wished to learn the result blindly trusted most of the systems in question, he would have been misled. So remember to think about the results you are given. The computer is not always necessarily right, and even if it is the answer is not necessarily complete.

### 1.1.1 An Overview of Current Systems

Painted with a broad brush, mathematical software can be broken into three major categories: Numerical, Symbolic, and Special Purpose. These categories are not set in stone; symbolic packages are capable of some numerical work, for example. Special Purpose systems, by definition, tend to incorporate whatever abilities are needed to attack their particular problem.

#### Numerical Mathematical Software

There are many players in this game, since numerical computation has many immense practical benefits and is quite often used to approach real world problems. The premier general solution in the free software world is probably GNU Octave (<http://www.octave.org>). (Talk about octave.) In the commercial world, Matlab (<http://www.matlab.com>) is a lead player, and is more or less the standard by which Numerical packages are judged. (Talk about Matlab.) (Mathcad? Others?)

#### Symbolic Mathematical Software

There are many fewer players here, partially since symbolic mathematics are not particularly necessary or feasible for many applications, and the difficulty involved with producing such software is considerable. There are four or five significant contenders. The commercial leaders in this field are Mathematica (<http://www.wolfram.com>) and Maple (<http://www.?.>), both very respectable programs. There is also Reduce, Mupad, and (?), which have a somewhat smaller following. Macsyma, until 1998 or thereabouts, was also a player in the commercial symbolic software game - unfortunately the company was sold at that time and the product is now off the market. (See the history of Macsyma.) Maxima, by far the oldest and most mature of the open source symbolic software offerings, is based off of the original code base of Macsyma and has many of the same core abilities as Macsyma, although it is lacking in the refinement and graphical touches. Since Macsyma has vanished from the commercial market, it is hoped that a vigorous effort to improve Maxima, of which this document is a part, will allow former Macsyma users to begin making use of Maxima instead. There are many other projects with some symbolic capabilities out there, but none have as yet shown any sign of being able to challenge Maxima as a general purpose symbolic computer algebra system. Some of the new efforts are yacas (<http://>), etc.

#### Special Purpose Mathematical Software

(Form, Pari, etc.)

### 1.1.2 Free vs. Commercial - Do I get what I pay for?

One of the all time popular sayings when it comes to purchases is “You get what you pay for.” This is accurate to a point when it comes to software, but fortunately only to a point. Traditional economics doesn’t really deal with the concept of open source programs. We will attempt to explain why you might actually prefer the open source solution, and why there are also good reasons for going with the commercial product.

#### Free Software

The phenomena of open source software is based on the desire of computer savvy individuals who are skilled in computer coding wishing to share their code with the world. The reasons are as varied as the individuals - many are convinced that the massive peer review made possible by the availability of the source code to their software enables them to write better code, and are not interested in making money; some wish to give back to the community that has given them access to free tools; some may only add a feature or two to an already existing program to achieve a

goal, and over time abilities accumulate. Whatever the reason, the Linux and \*BSD communities have shown that the concept can work, and work well.

But the question remains - do I get what I pay for? The question primarily revolves around what you expect when you use software. If you do not expect guaranteed support, printed manuals, etc. then you get more than what you pay for, and free software is a great bargain. Many businesses have begun to discover this bargain, and are taking advantage of it. The drawback is that if you want support, it is not guaranteed. Often people are quite helpful, but they are not commercially obligated to support and maintain their product. So if that is what you want, you should not use free software, at least not without contracting someone to maintain it. If you want a new feature you are free to suggest it, but unless you add it yourself it is not guaranteed to be there. And often in order to distribute an improved version of the program, you must distributed your changes as freely as the original code was distributed. This is a problem for some commercial interests.

### **Commercial Software**

Commercial software, on the other hand, is supported by a company which is obligated to listen to your complaints. You get printed documentation and a number to call. Often, particularly in the case of mathematical software, the interfaces on the commercial programs tend to be smoother and more friendly. The learning curve, being less steep, is often less expensive to climb time-wise and people who don't like to train will find it worth a little money to avoid the steep curves.

The catch is, of course, that if you should wish to make a change, fix a bug, add a feature, check how something is done, etc. you can't do it. In academic work openness has long been a staple of the process - how you got a result is extremely important. In the case of a commercial product, they have a vested interest in you're not knowing exactly how they did what they did - they are making money off of those methods. This can lead to problems when doing research which will tend to push the limits of the system.

So the trade-off is clear. As usual, the choice depends on the situation.

### **Why is Maxima free?**

Maxima is free partially because of it's heritage, coming from an academic environment, and partially to keep it alive. The audience for symbolic computation is small in any case, and the available mindshare for Maxima is smaller still, with the field being dominated by the likes of Mathematica and Maple and the last major commercial effort from this code base having vanished. There are advantages to using Maxima for a number of problems, and those who have invested a large amount of time learning it would like to see it remain alive. As a free product, as long as there is enough interest to keep the project going the program will remain active. Part time helpers and coders can do wonders when commercial efforts can't be supported. And even if the project should fade, individuals can freely use and extend the program on their own. So we hope you will take a look, and like what you see. Welcome to the world of Maxima.

## **1.2 The History of Macsyma**

Who can write this?

# Chapter 2

## Installing

### 2.1 Installing on Linux

#### 2.1.1 The Easy Way

If you are using Debian Linux, there are packages for Maxima in the Debian archives. Redhat users can find packages at rpmfind.net and mirrors. To the best of my knowledge these packages are all compiled against GNU Common Lisp (GCL). GCL does not access the readline library, which makes using Maxima on the terminal somewhat cumbersome. There are various interfaces which address this issue, but in the case of these packages the left arrow key will not take you back to the middle of an expression and let you edit just what you want - you will have to delete everything back to the point where you made your error. See the interfaces section for more about this, or if you really want to use maxima in the terminal I'd recommend compiling against CLISP.

#### 2.1.2 The Hard Way

Compiling Maxima on Linux should go reasonably smoothly, although some distributions seem to have problems with the makefiles. Redhat 7.1 is known to be a bit flaky in this regard.

##### Compiling with GCL

GCL is the version of Lisp Maxima has been maintained on for many years by William Schelter, and thus the default system on which to build Maxima. Maxima will typically be made to work with GCL, and then be fixed to work with other Lisps. If in doubt, start here, but again be warned that readline support is not present in GCL.

The first step in this process is to compile GCL. The files from the compiling of GCL are needed when building Maxima against it, so first download and decompress the most current version of GCL. Compiling it should be simple:

```
cd $HOME/gcl
./configure
make
```

Once that is complete, download and decompress Maxima. Now the process gets a little more complicated - you will have to hand edit the file `configure`. What follows is the first part of the file, with the parts you must edit in bold:

```
#!/bin/sh
# edit next few lines
```

```

# GCLDIR should be where gcl was built, and the o/*.o lsp/*.o etc must be
# there to link with maxima
GCLDIR=/home/cliff/gcl-2.3.8
# the directory where this file is, and where you will build maxima
# could use
MAXDIR=/home/cliff/maxima-5.5
MAXDIR=`pwd`
# where to put some maxima .el files
EMACS_SITE_LISP=/usr/lib/emacs/site-lisp
# determines where to install
# PREFIX_DIR=/usr/local puts things in /usr/local/lib/maxima-x.y
# and /usr/local/bin
PREFIX_DIR=/usr/local
INFO_DIR=${PREFIX_DIR}/info
MAN_DIR=${PREFIX_DIR}/man/man1
##### end editing #####

```

Once this is correctly done, run the configure script: `./configure`. If this succeeds without any problems, run `make`. This could be a long process, depending on your machine. Once it is done, and if no errors result, run `make test`. If any errors appear in either stage, consult the trouble shooting section. Otherwise, you are ready to `su` into root and run `make install`. Once this process is completed, you should be able to run `maxima` and `xmaxima`. Emacs mode may take a little more work, and the other interfaces are separate programs - see the Interfaces chapter for details.

### Compiling with CLISP

Clisp has, among other features, the ability to use the readline library. This means that many of the limitations created in the terminal by GCL are not relevant here, and for the beginner especially this is a good place to start. Unfortunately, compiling on a flavor of Lisp other than GCL is not quite as smooth - here are the steps.

Change into the directory `$HOME/maxima/src` (or `$HOME/maxima-5.6/src` - whatever your setup dictates.)

For newer versions of clisp, you need to change a couple lines in the file `compile-clisp.lisp`:

```

< (lisp:gc) → (ext:gc)
< (lisp:saveinitmem "maxima-clisp.mem" → (ext:saveinitmem "maxima-clisp.mem"

```

Then run the following commands:

```

make clisp-compile CLISP=clisp
make maxima-clisp.mem CLISP=clisp
make test-clisp CLISP=clisp

```

Once that is complete, you can either leave maxima where you built it, or move the directory to a more suitable place. Once that is done, in order to set up a convenient way to run Maxima, you can create some shell scripts in `/usr/local/bin` (make sure to do `chmod +x` on both files):

```

=====maxima=====
#!/bin/bash
export MAXIMA_DIRECTORY=/PATH_TO_NEW_DIRECTORY/maxima
exec clisp -M ${MAXIMA_DIRECTORY}/src/maxima-clisp.mem
=====

```

```

=====xmaxima=====
#!/bin/bash
export MAXIMA_DIRECTORY=/PATH_TO_NEW_DIRECTORY/maxima
exec ${MAXIMA_DIRECTORY}/bin/xmaxima -lisp clisp
=====

```

### Compiling with CMULISP

#### Other Lisps

## 2.2 Installing on Windows

### 2.2.1 The Easy Way - Installing the Windows Binary

This is the recommended way to use Maxima on Windows. The current binary release is technically a beta, but should serve most needs quite well. Here is how a basic install works:

1. Download the binary from <http://www.ma.utexas.edu/maxima.html> It should be named maxima551-setup.exe
2. Once you have downloaded that file, launch the installer. You should see this screen:



3. Simply follow the menus from that point - the install dialog should be very clear. Do read the README and License when they are presented - the information there is good to know.
4. Simply launch the program from it's icon - everything should work out of box. It is known to run on the following versions of Windows:

Windows 98

### **2.2.2 The Hard Way**

Compiling on Windows (I have no idea how to do this.)

## Chapter 3

# Available Interfaces to Maxima

Maxima is at heart a command line program, and by itself it is not capable of displaying formatted mathematics beyond the ascii text level. However there are other interfaces which may be used. Maxima has the ability to export expressions using the T<sub>E</sub>X syntax, and several programs use this device to help with output formatting. (None at this time allow formatted input.) All have their strengths and weaknesses - the choice will likely depend on the skill of the user and the task at hand. We will discuss here all of the interfaces currently available, and the user can make the choice him/herself which one to use.

### 3.1 The Terminal Interface

The terminal interface is the original interface to Maxima. While in some sense all of the interfaces to Maxima could be termed Terminal interfaces, when we refer to it here we mean the command line, no frills interface you would use in an xterm or a nongraphical terminal. It is the least capable of all the alternatives in many respects, but it is also the least demanding.

How comfortable this interface is depends to a large extent on what Lisp you used to compile Maxima originally. If you used the default, which is gcl, (what all binary packages I am aware of use) you do not have readline support on the terminal. This means you will not be able to use the back arrow key to go to the middle of an expression and change it - you must erase everything as you go back. This is a serious limitation, and if this is the case it is recommended by the author that you either compile against a Lisp flavor such as Clisp, which has readline support, or use one of the other interfaces. Any of the others should solve this problem. If you choose to use this interface, you activate it by simply typing `maxima` on the terminal prompt.

### 3.2 The Emacs Interface

An Emacs mode has been written for Maxima, and this is probably the choice I would recommend instead of the bare terminal. You get to go back in an expression without having to erase your expression, and you get syntax highlighting. (Need to fill in more complete description of capabilities of emacs mode). To install this mode you need to make sure the files `maxima.el`, `maxima-font-lock.el`, `smart-complete.el`, and `sshell.el` are in your Emacs site-lisp directory. (We need to make sure of the details of this process, may be standardize it better.) Then, in your `.emacs` file, add the following line (without the period: `(autoload 'maxima "maxima" "Maxima editing mode" t)`). Then when you wish to run Maxima you start Emacs and type `M-x maxima`.

(Note: In version 5.6 of Maxima if you don't shut down the maxima process with the `quit()` command, you can leave an unwanted process running which will use almost all of the CPU power until killed. If that should happen, use

Figure 3.1: Maxima in an Xterm.

```

maxima
GCL (GNU Common Lisp) Version(2.3.8) Wed Sep  5 08:00:22 CDT 2001
Licensed under GNU Library General Public License
Contains Enhancements by W. Schelter
Maxima 5.5 Wed Sep  5 07:59:43 CDT 2001 (with enhancements by W. Schelter).
Licensed under the GNU Public License (see file COPYING)
(C1) 'diff(x^2+y^x+2*y+t[y],y);

(D1)
      d
      -- (t + y + 2 y + x )
      dy  y

(C2) ev(D1,diff);

(D2)
      x - 1
      x y  + 2
(C3) █

```

the `kill` command to stop the process. This should be fixed in newer versions.)

### 3.3 Xmaxima

Xmaxima is a new development in the lifetime of Maxima. It is based on Tcl/Tk, and has the nifty feature of in-line plotting when using the default plotting routine. No version earlier than ? has this interface, so if you aren't using the new versions you won't have this included in your default package. This is probably where most people will start learning Maxima, and it is not a bad place to start. You avoid the earlier mentioned limitations of the vanilla terminal, and avoid having to master the setup for Emacs. In order to start this interface simply type `xmaxima` in a terminal. From there, you should get what looks like the terminal interface in a Tk window, with an introductory html document below it. There is also a pull down menu system. For Windows users, this will more than likely be the default choice. `Symaxx` and `TEXmacs` are Unix only. (What about Emacs in Windows?)

### 3.4 Symaxx

In the words of its creator, "Symaxx provides those 5% of features, that are needed to do 95% of your work." It is more graphical than any of the other Maxima interfaces, and although input is still ascii based the output is formatted. It is based on Perl and Tk, both of which are required to run it. This interface tends, at least in the experience of this author, to be a bit resource intensive, but allows some page formatting possibilities which make it a very interesting program. Symaxx uses a rather different style of interface, and if one wishes to use it seriously the Symaxx manual is a highly recommended read. (Need to decide how much to document about symaxx here - should probably do at least install and basics, and the graphical fix.)

### 3.5 T<sub>E</sub>Xmacs

T<sub>E</sub>Xmacs is a new WYSIWYG scientific editor, and it has the ability to interface with many computer algebra systems, including Maxima. This program takes advantage of the T<sub>E</sub>X output Maxima can produce to format it's output. To launch Maxima inside of T<sub>E</sub>Xmacs you go up to the menu and select Insert -> sessions -> maxima. Then things work like they do in xmaxima or Emacs. This is probably the most visually appealing way to run Maxima.

Figure 3.2: Maxima in Emacs

```

emacs@shemp.physics.smu.edu
Buffers Files Tools Edit Search Mule Complete In/Out Signals Help
GCL (GNU Common Lisp) Version(2.4.0) Wed May 9 12:02:00 CDT 2001
Licensed under GNU Library General Public License
Contains Enhancements by W. Schelter
Maxima 5.6 Wed May 9 12:01:49 CDT 2001 (with enhancements by W. Schelter).
Licensed under the GNU Public License (see file COPYING)
(C1) integrate(sin(x),x);
(D1) -cos(x)
(C2) integrate(1/(x+3),x);
(D2) LOG(x + 3)
(C3) integrate((1)/((x^2)+(x^3)),x);
(D3) LOG(x + 1) - LOG(x) - 1/x
(C4) integrate(sqrt(x^2+x^3),x);
(D4) (2/5)(x + 1)^(5/2) - (2/3)(x + 1)^(3/2)
(C5) integrate(sqrt(x*sqrt(x*sin(x))),x);
(D5) [
      I Sqrt(x) (x sin(x))^(1/4) dx
      ]
(C6)
-1:** *maxima* (Inferior Maxima; run)--L30--All-----

```

Figure 3.3: Xmaxima

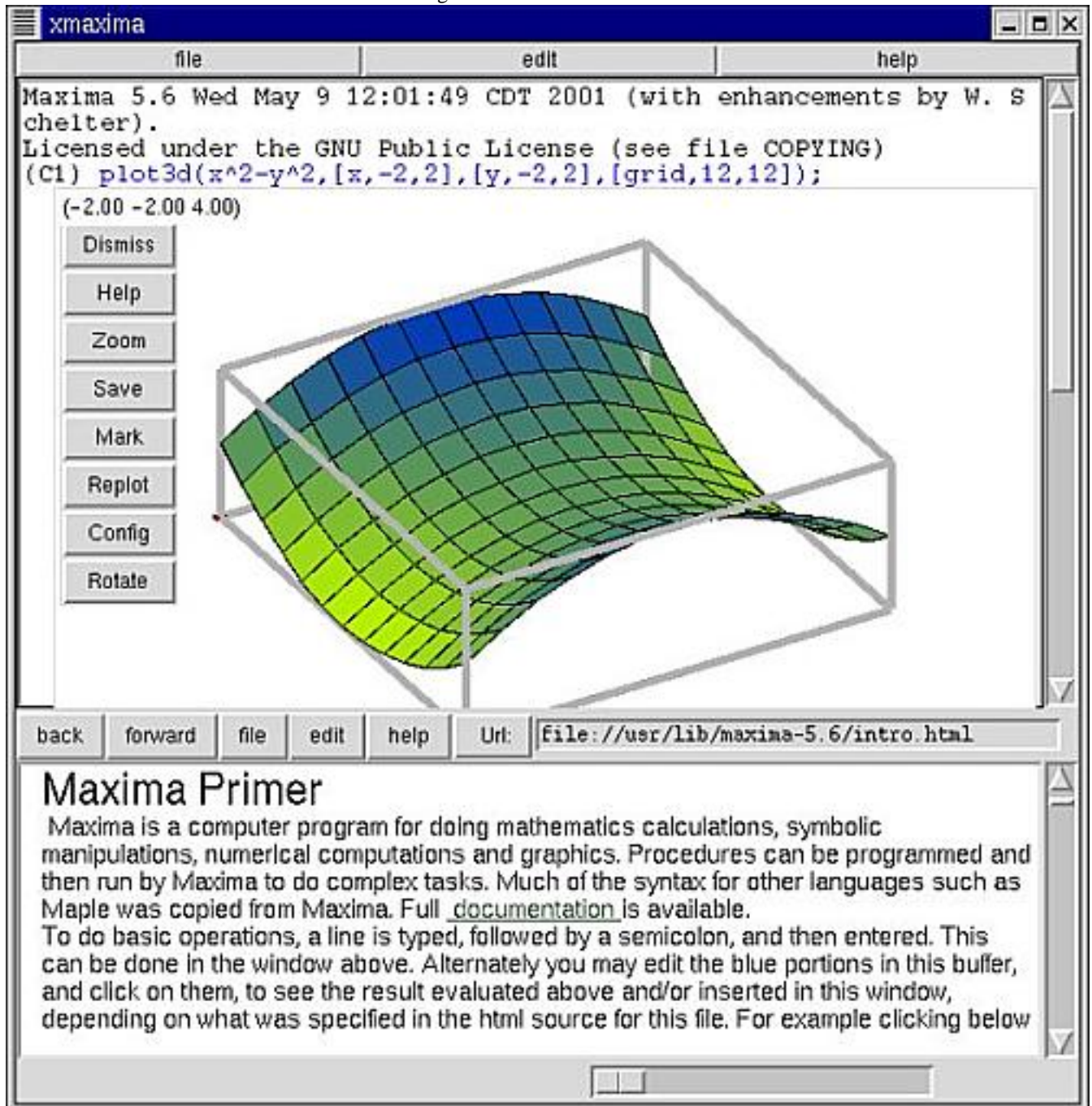


Figure 3.4: Symmax2

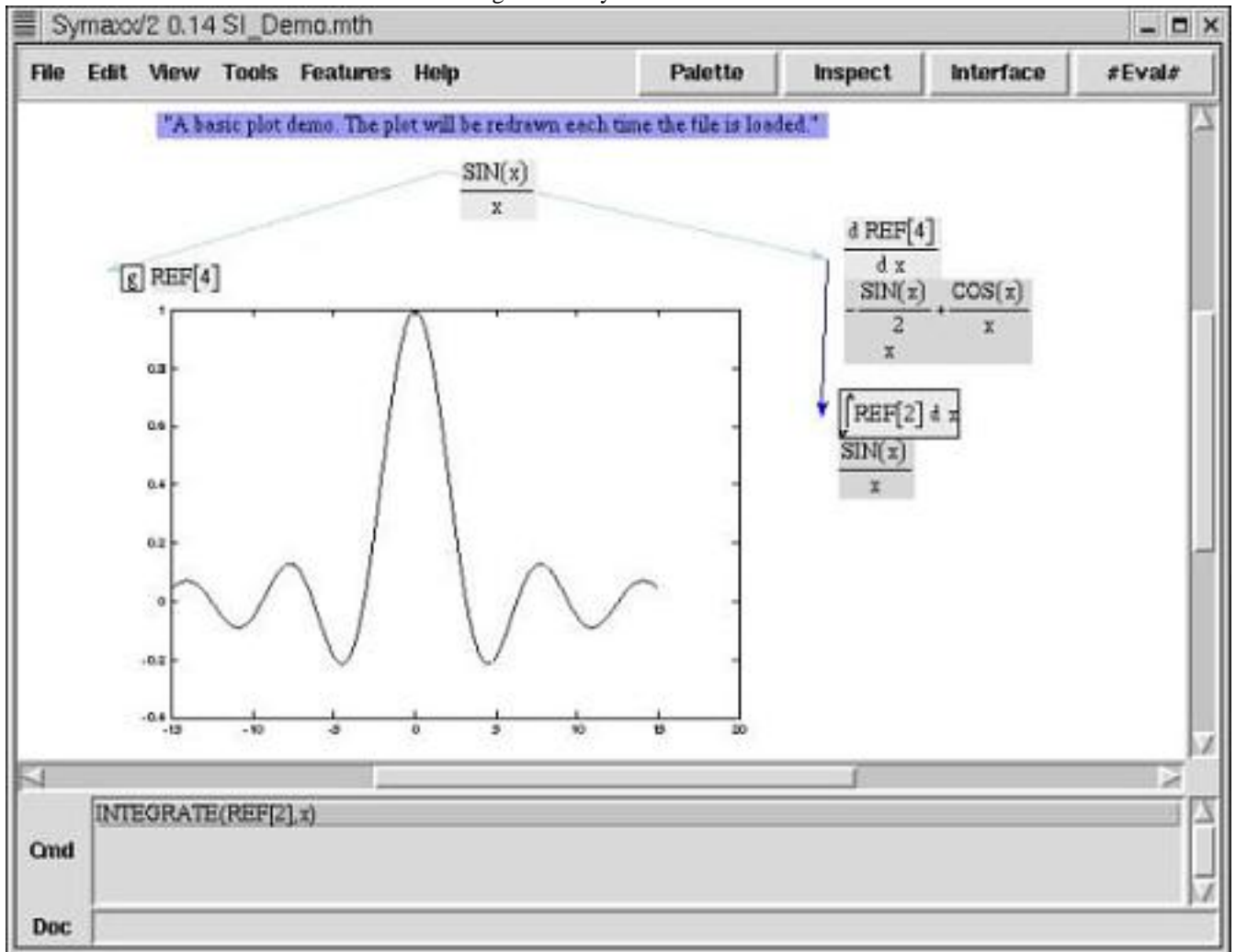
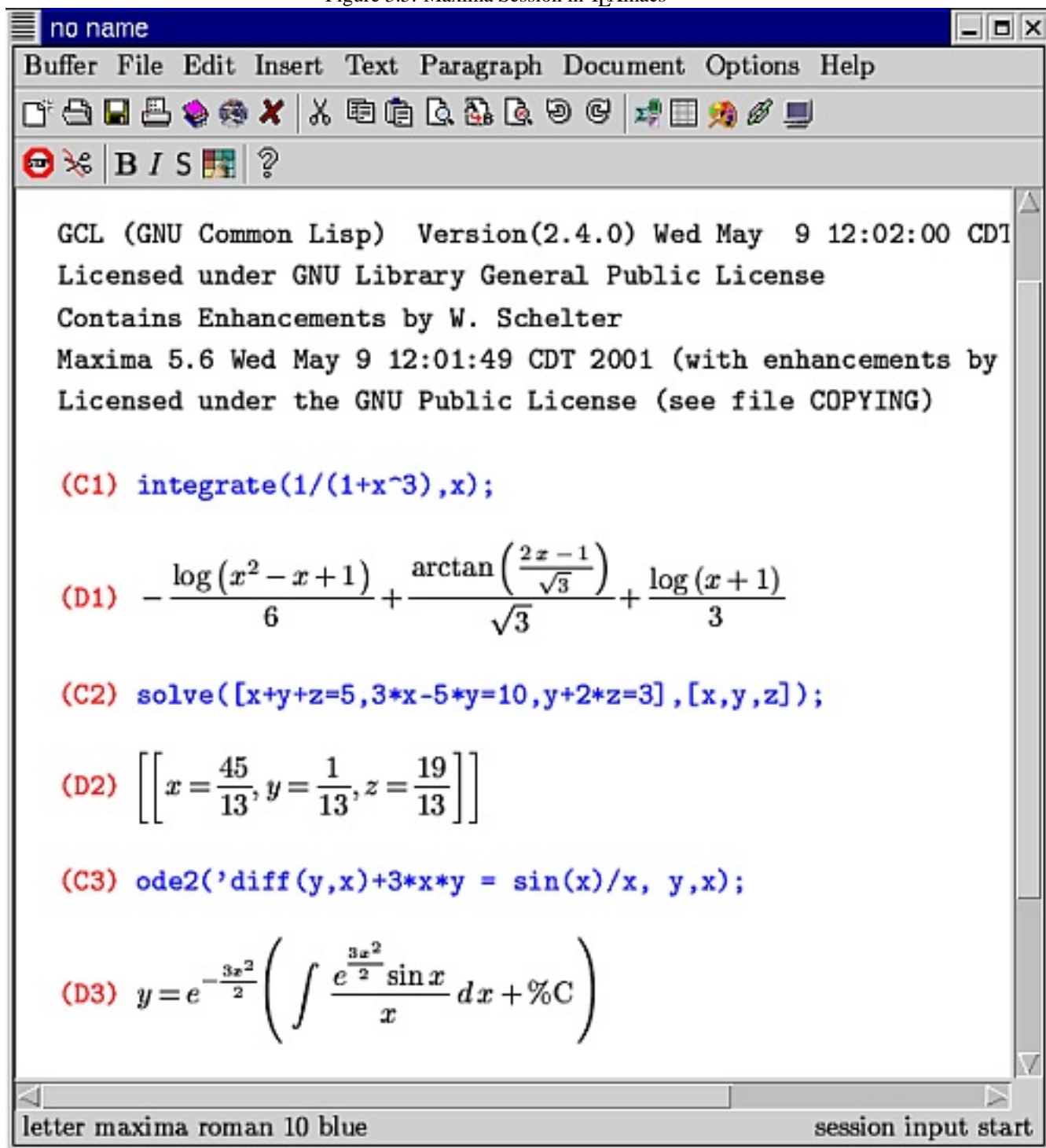


Figure 3.5: Maxima Session in T<sub>E</sub>Xmacs

## Chapter 4

# The Basics - What you need to know to operate in Maxima

This chapter will attempt to address universal concepts which you will need to know when using Maxima for a wide variety of tasks.

### 4.1 The Very Beginning

All computer algebra systems have syntactical rules, i.e. a structured language by which the user communicates his/her commands to the system. Without being able to communicate in this language, it is impossible to accomplish anything in such a system. So we will attempt to describe herein the basics.

#### 4.1.1 Our first Maxima Session

We will start by demonstrating the ultimate basics:  $+$ ,  $-$ ,  $*$ , and  $/$ . These symbols are virtually universal in any mathematical system, and mean exactly what you think they mean. We will demonstrate this, and at the same time introduce you to your first session in Maxima. In the interface of your choice, try the following:

```
GCL (GNU Common Lisp) Version(2.3.8) Wed Sep 5 08:00:22 CDT 2001
Licensed under GNU Library General Public License
Contains Enhancements by W. Schelter
Maxima 5.5 Wed Sep 5 07:59:43 CDT 2001 (with enhancements by W. Schelter).
Licensed under the GNU Public License (see file COPYING)
(C1) 2+2;

(D1)                                     4
(C2) 3-1;

(D2)                                     2
(C3) 3*4;

(D3)                                     12
(C4) 9/3;
```

```

(D4)                                     3
(C5) 9/4;

                                     9
(D5)  -
                                     4
(C6) quit();

```

Above is our first example of a Maxima session. We notice already several characteristics of a Maxima session: The startup message, which gives the version of Maxima being used, (FIXME: What's up with the date? Is that the day it was compiled?) the labels in front of each line, the semicolon at the end of each line, and the way we exit the session. The startup message is not important to the session, but you should take note of what version of Maxima you are using, especially if there is a known problem in an earlier version which might impact what you are trying to do. The other aspects are important, and we will discuss them at some length.

### Exiting a Maxima Session

You want to be able to get out of what you get into. So the first command we discuss will be the command that gets you out of Maxima, and while we are at it we will discuss how to get out of debugging mode. Debugging mode is quite useful for some things, and the reader is encouraged to look to later chapters for an in-depth look at the debugging mode, but for now we will stick to basics. As you see above, `quit()`; is the command which will exit Maxima. This is a bit confusing for new users, but you must type that full command. Simply typing `quit` or `exit` will not work, nor will pressing `CTRL-C` - if you try the latter you will be dumped into the debugging mode. If that happens, simply type `:q` if you are running GNU Common Lisp, or `:a` if running CLISP. (If in doubt use `:q` - most binary packages use GCL at this time.) Here's an example of what not to do, and how to get out of it if you do:

```

(C1) quit;

(D1)                                     QUIT
(C2) exit;

(D2)                                     EXIT
(C3)
Correctable error: Console interrupt.
Signalled by MACSYMA-TOP-LEVEL.
If continued: Type :r to resume execution, or :q to quit to top level.
Broken at SYSTEM:TERMINAL-INTERRUPT. Type :H for Help.
MAXIMA>>
Correctable error: Console interrupt.
Signalled by SYSTEM:UNIVERSAL-ERROR-HANDLER.
If continued: Type :r to resume execution, or :q to quit to top level.
Broken at SYSTEM:TERMINAL-INTERRUPT.
MAXIMA>>>:q

(C3) quit();

```

The first two lines show what happens if you forget the `()` or type `exit`. Nothing major, but you won't exit Maxima. `CTRL-C` causes a few more problems - if you actually read the message, you will see it tells you how to

handle it. In the above example, CTRL-C was hit twice - that is not a proper way to exit Maxima either. Just remember to use `:q` to exit debugging and `quit()` to exit Maxima, and you should always be able to escape trouble.

### The End of Entry Character

All expressions entered into Maxima must end with either the `;` character or the `$` character. The `;` character is the standard character to use for this purpose. The `$` symbol, while performing the same job of ending the line, suppresses the output of that line. This example illustrates these properties:

(C1)  $x^5+3x^4+2x^3+5x^2+4x+7;$

(D1) 
$$x^5 + 3x^4 + 2x^3 + 5x^2 + 4x + 7$$

(C2)  $x^5+3x^4+2x^3+5x^2+4x+7\$$

(C3) D2;

(D3) 
$$x^5 + 3x^4 + 2x^3 + 5x^2 + 4x + 7$$

(C4)  $x^5+3x^4+2x^3$   
 $+5x^2+4x+7;$

(D4) 
$$x^5 + 3x^4 + 2x^3 + 5x^2 + 4x + 7$$

(C5)  $x^5+3x^4+2x^3$   
 $+5x^2+4x+7;$

(D5) 
$$x^5 + 3x^4 + 2x^3 + 5x^2 + 4x + 7$$

In (C1), we input the expression using the `;` character to end the expression, and on the return line we see that D1 now contains that expression. In (C2), we input an identical expression, except that we use the `$` to end the line. (D2) is assigned the contents of (C2), but does not visually display those contents. Just to verify that (D2) does in fact contain what we think it contains we ask Maxima to display its contents on (C3) and we see that they are in fact present. This is extremely useful if you are working on a problem which has many steps, and some of those steps would produce long outputs you don't need to actually see. In (C4), we input part of the expression, press return, finish the expression, and then use the `;` character. Notice that the input did not end until we used that character and pressed return - return by itself does nothing. You see (D4) contains the same expression as (D1) shown above. To Maxima, the inputs are the same. This can be useful if you are inputting a long expression and wish to keep it straight visually, to avoid errors. You can also input spaces without adversely affecting the formula, as shown in (C5).

### The (C\*) and (D\*) Labels

These labels are more than just line markers - they are actually the names in memory of the contents of the lines. This is quite useful for a number of tasks. Let's say you wish to apply a routine, say a `solve` routine, to an expression for several different values. Rather than retyping the entire expression, we can use the fact that the line numbers act as markers to shorten our task considerably, as in this example:

(C1)  $3x^2+7x+5;$

```

(D1)          2
          3 x  + 7 x + 5
(C2) solve(D1=3,x);

(D2)          1
          [x = - -, x = - 2]
          3
(C3) solve(D1=7,x);

(D3)          Sqrt(73) + 7      Sqrt(73) - 7
          [x = - ----, x = ----]
          6                    6
(C4) solve(D1=a,x);

(D4)          Sqrt(12 a - 11) + 7      Sqrt(12 a - 11) - 7
          [x = - ----, x = ----]
          6                    6

```

In this example, we desire to solve the expression  $3x^2 + 7x + 5$  for  $x$  when  $3x^2 + 7x + 5 = 3$ ,  $3x^2 + 7x + 5 = 7$ , and  $3x^2 + 7x + 5 = a$ . ( $a$  in this case is an arbitrary constant.) Rather than retype the equation many times, we merely enter it once, and then use that label to set up the similar problems more easily.

You do not need to settle for this method of labeling - you can define your own expressions if you so choose, by using the `:` assign operator. Let us say, for example, that we wish to solve the problem above, but would rather call our equation `FirstEquation` than `D1`. We will show that process here, with one deliberate error for illustration of a property:

```

(C1) FirstEquation:3*x^2+7*x+5;

(D1)          2
          3 x  + 7 x + 5
(C2) solve(FirstEquation=3,x);

(D2)          1
          [x = - -, x = - 2]
          3
(C3) solve(FirstEquation=7,x);

(D3)          Sqrt(73) + 7      Sqrt(73) - 7
          [x = - ----, x = ----]
          6                    6
(C4) solve(FirstEquation=a,x);

(D4)          Sqrt(12 a - 11) + 7      Sqrt(12 a - 11) - 7
          [x = - ----, x = ----]
          6                    6
(C5) solve(firstequation=a,x);

(D5)          []

```

You see that this process works exactly the same as before. On line (C5), you see we entered the name of FirstEquation as lower case, and the calculation failed. These names are case sensitive. This is true of all variables. Later on you will see cases in Maxima such as sin, where SIN and sin are the same, but it is not save to assume this is always true and when in doubt, watch your cases. In general you are likely to find that your sessions will be easier to read if you use consistent cases, so it is not a bad habit to form and keep. You will also make fewer errors of the type above. Here it is obvious what went wrong, but later on it may not be so easy.

### 4.1.2 To Evaluate or Not to Evaluate

Operators in Maxima, such as diff for derivative, are a common feature in many Maxima expressions. The problem is, while you need to include an operator at a given point in your process, you may not want to deal with the output from it at that point in the problem. Therefore, Maxima provides the ' toggle for operators. See the example below for an example of how this works.

(C1) `diff(1/sqrt(1+x^3), x);`

(D1) 
$$-\frac{x^2}{3(x+1)^{3/2}}$$

(C2) `'diff(1/sqrt(1+x^3), x);`

(D2) 
$$\frac{d}{dx} \left( \frac{1}{\text{SQRT}(x^3 + 1)} \right)$$

### 4.1.3 The Concept of Environment - The ev Command

All mathematical operations in Maxima take place in an environment, which is to say the system is assuming it should do some things and not do other things. There will be many times you will want to change this behavior, without doing so on a global scale. Maxima provides a way to define a local environment on a per command basis, using the `ev` command. `ev` is one of the most powerful commands in Maxima, and the user will benefit greatly if they master this command early on while using Maxima.

#### From the top

We will begin with a very simple example:

(C1) `ev(solve(a*x^2+b*x+c=d, x), a=3, b=4, c=5, d=6);`

(D1) 
$$\left[ x = -\frac{\text{SQRT}(7) + 2}{3}, x = \frac{\text{SQRT}(7) - 2}{3} \right]$$

(C2) `a;`

(D2) 
$$a$$

The first line uses the `ev` command to solve for  $x$  without setting variables in the global environment. To make sure that our variables remain undefined, we check that  $a$  is still undefined in line (C2), and it is.

Now let's examine some of the more interesting features of `ev`. The general syntax of the `ev` command is `ev(exp, arg1, ..., argn)`. `exp` is an expression, like the one in the example above. You can also use a D\* entry name or your own name for an expression. `arg*` has many possibilities, and we will try to step through them here.

### EXPAND(m,n)

`Expand` is an argument which allows you to limit how Maxima expands an expression - i.e., how high a power you want it to expand. `m` is the maximum positive power to expand, and `n` is the largest negative power to expand. Here is an example:

```
(C1) ev((x+y)^5+(x+y)^4+(x+y)^3+(x+y)^2+(x+y)+(x+y)^-1+(x+y)^-2+(x+y)^-3+(x+y)^-4+(x+y)^-5,EXPAND(3,3));
```

$$\begin{aligned}
 \text{(D1)} \quad & \frac{1}{y^3 + 3xy^2 + 3x^2y + x^3} + \frac{1}{y^2 + 2xy + x^2} + (y+x)^5 + (y+x)^4 + \frac{1}{y+x} \\
 & + \frac{1}{(y+x)^4} + \frac{1}{(y+x)^5} + y^3 + 3xy^2 + y^2 + 3xy + 2xy + y + x^3 + x^2 + x
 \end{aligned}$$

This may be a little hard to read at first, but if you look closely you will see that every power of  $-3 \leq p \leq 3$  has been expanded, otherwise the subexpression has remained in its original form. This is extremely useful if you want to avoid filling up your screen with large expansions that no one can read or use.

### Numerical Output - FLOAT and NUMER

When one of the arguments of `ev` is `FLOAT`, `ev` will convert non-integer rational numbers to floating point. `NUMER` will do everything that `FLOAT` will, since `FLOAT` is invoked as part of `NUMER`, and `NUMER` also (what?? I can't find any difference.)

```
(C1) a:9/4;
```

```
(D1)                                     9
      -
      4
```

```
(C2) exp(a);
```

```
(D2)                                     9/4
      %E
```

```
(C3) ev(exp(a),FLOAT);
```

```
(D3)                                     9.487735836358526
```

```
(C4) ev(exp(a*x),FLOAT);
```

$$(D4) \quad 2.25 \ x \quad \%E$$

The above example shows the various uses of the FLOAT toggle. The first line defines  $a$  globally to be  $\frac{9}{4}$ . The second shows how  $e^a$  is represented ordinarily. The next lines show how `ev` and `FLOAT` can reduce numbers to their decimal form.

### Specifying Local Values for Variables, Functions, etc.

One of the best things about the `ev` command is that for one evaluation you may specify in an arg what values are to be used for the evaluation in place of variables, how to define functions, which functions to evaluate, etc. We will work through a series of examples here, probably this will be the best way to illustrate the various possibilities of this aspect of `ev`.

(C1) `eqn1:'diff(x/(x+y)+y/(y+z)+z/(z+x),x);`

$$(D1) \quad \frac{d}{dx} \left( \frac{y}{z+y} + \frac{z}{z+x} + \frac{x}{y+x} \right)$$

(C2) `ev(eqn1,diff);`

$$(D2) \quad -\frac{z}{(z+x)^2} + \frac{1}{y+x} - \frac{x}{(y+x)^2}$$

(C3) `ev(eqn1,y=x+z);`

$$(D3) \quad \frac{d}{dx} \left( \frac{z+x}{2z+x} + \frac{x}{z+2x} + \frac{z}{z+x} \right)$$

(C4) `ev(eqn1,y=x+z,diff);`

$$(D4) \quad \frac{1}{2z+x} - \frac{z+x}{(2z+x)^2} + \frac{1}{z+2x} - \frac{2x}{(z+2x)^2} - \frac{z}{(z+x)^2}$$

In this example, we define `eqn1` to be the derivative of a function, but use the `'` character in front of the `diff` operator to notify Maxima that we don't want it to evaluate that derivative at this time. (More on that in the `??` section.) In the next line, we use the `ev` with the `diff` argument, which instructs `ev` to take all derivatives in this expression. Now, let's say we want to define  $y$  as a function of  $z$  and  $x$ , but again avoid evaluating the derivative. We supply our definition of  $y$  as an argument to `ev`, and in (D3) we see that the substitution has been made. Now, let's evaluate the derivative after the substitution has been made. We work as before, except this time we supply both the new definition of  $y$  and the `diff` argument, telling `ev` to make the substitution and then take the derivative. In this particular case, the order of the arguments does not matter. The case where it will matter is if you are making multiple substitutions - then they are handled in sequence from left to right.

(need example here, one where the difference is noticeable).

We can also substitute for functions:

(C1) eqn4: f(x,y) \*' diff(g(x,y), x);

(D1) 
$$f(x, y) \left( \frac{d}{dx} (g(x, y)) \right)$$

(C2) ev(eqn4, f(x,y)=x+y, g(x,y)=x^2+y^2);

(D2) 
$$(y + x) \left( \frac{d}{dx} (y^2 + x^2) \right)$$

(C3) ev(eqn4, f(x,y)=x+y, g(x,y)=x^2+y^2, DIFF);

(D3) 
$$2 x (y + x)$$

(At the moment, ev seems to take only the first argument in the following example from solve: the manual seems to indicate it should be taking both as a list??)

(C1) eqn1: f(x,y) \*' diff(g(x,y), x);

(D1) 
$$f(x, y) \left( \frac{d}{dx} (g(x, y)) \right)$$

(C2) eqn2: 3\*y^2+5\*y+7;

(D2) 
$$3 y^2 + 5 y + 7$$

(C3) ev(eqn1, g(x,y)=x^2+y^2, f(x,y)=5\*x+y^3, solve(eqn2=5, y));

(D3) 
$$\left( 5 x - \frac{8}{27} \right) \left( \frac{d}{dx} (x^2 + \frac{4}{9}) \right)$$

(C4) ev(eqn1, g(x,y)=x^2+y^2, f(x,y)=5\*x+y^3, solve(eqn2=1, y), diff);

(D4) 
$$2 x \left( 5 x - \frac{(\text{SQRT}(47) \%I + 5)^3}{216} \right)$$

(C5) ev(eqn1, g(x,y)=x^2+y^2, f(x,y)=5\*x+y^3, solve(eqn2=1, y), diff, FLOAT);

(D5) 
$$2 x \left( 5 x - 0.00462962962963 (\text{SQRT}(47) \%I + 5)^3 \right)$$

### Other arguments for ev

INFEVAL - This option leads to an "infinite evaluation" mode, where ev repeatedly evaluates an expression until it stops changing. To prevent a variable, say X, from being evaluated a way in this mode, simply include X='X as

an argument to `ev`. There are dangers with this command - it is quite possible to generate infinite evaluation loops. For example, `ev(X,X=X+1,INFEVAL);` will generate such a loop. Here is an example: (need example where this is useful.)

### How `ev` works

The flow of the `ev` command works like this:

1. The environment is set up by scanning the arguments. During this step, a list is made of non-subscripted variables appearing on the left side of equations in the arguments or in the value of some arguments if the value is an equation. Both subscripted variables which do not have associated array functions and non-subscripted variables in the expression `exp` are replaced by their global values, except for those appearing in the generated list.
2. If any substitutions are indicated, they are carried out.
3. The resulting expression is then re-evaluated, unless one of the arguments was `NO-EVAL`, and simplified according to the arguments. Note that any function calls in `exp` will be carried out AFTER the variables in it are evaluated.
4. If one of the arguments was `EVAL`, the previous two steps are repeated.

## 4.2 Common Operators in Maxima

An operator is simply something that signals a specific operation is to be performed. There are many, many possible operators in Maxima. We will address various operators for specific jobs all throughout this manual - this section is not comprehensive.

### 4.2.1 Assignment Operators

In mathematics, we quite often want to declare functions, assign values to numbers, and do many similarly useful things. Maxima has a variety of operators for this purpose.

`:` The basic assignment operator. We have already seen this operator in action; it is one of the most common in maxima.

```
(C1) A:7;
```

```
(D1)                                     7
```

```
(C2) A;
```

```
(D2)                                     7
```

`::` This is also an assignment operator, but it functions somewhat differently from the one above. If you are familiar with programming, this operator's function seems to be that of creating pointers. The following example will help illustrate how this actually functions: We assign the value of 2 to A, and 3 to B. Now we assign C the value of A, without evaluating A to get the number 2. We see that C knows it is the unevaluated A. Now, we use the `::` operator to assign the value of B, which is 3 to C. So C is now three. But because we use the `::` operator, A, which we declared C to be earlier, is now also 3.

(C1) A:2;	
(D1)	2
(C2) B:3;	
(D2)	3
(C3) C:'A;	
(D3)	A
(C4) C;	
(D4)	A
(C5) C::B;	
(D5)	3
(C6) B;	
(D6)	3
(C7) A;	
(D7)	3

It is quite unfortunate that this concept is so nonintuitive, but it is rooted deep in how computers work. For comparison, we will perform the same operation, but this time use the : operator instead. Observe the difference in how the values are assigned.

(C1) A:2;	
(D1)	2
(C2) B:3;	
(D2)	3
(C3) C:'A;	
(D3)	A
(C4) C;	
(D4)	A
(C5) C:B;	
(D5)	3
(C6) B;	
(D6)	3
(C7) A;	
(D7)	2

Note at the end, A still retains its original value of two. This concept is best learned with lots of practice - after a while you should begin to develop a feel for how it works.

`:=` This is the operator you would use to define functions. This is a common thing to do in computer algebra, so we will illustrate both how to and how not to do this.

```
(C1) y:=x^2;
```

Improper function definition:

```
y
-- an error.  Quitting.  To debug this try DEBUGMODE(TRUE); )
```

```
(C2) y=x^2;
```

```
(D2)          2
          y = x
```

```
(C3) y(2);
```

```
(D3)          y(2)
```

```
(C4) y(x)=x^2;
```

```
(D4)          2
          y(x) = x
```

```
(C5) y(2);
```

```
(D5)          y(2)
```

```
(C6) y[x]=x^2;
```

```
(D6)          2
          y  = x
           x
```

```
(C7) y[2];
```

```
(D7)          y
           2
```

```
(C8) y(x) := x^2;
```

```
(D8)          2
          y(x) := x
```

```
(C9) y(2);
```

```
(D9)          4
```

Look over the above example - the last one is correct, but it pays to know what doesn't work, as well. If you recognize the error or incorrect result you get, it will make for faster debugging.

## **Chapter 5**

# **Trig through Through Calculus**

This chapter and the next will probably split into many more - Trig, Algebra, Calculus, Programming, etc,etc,etc. I just don't know at this point. These chapters will probably largely be example based. Using things such as ratsimp, trigsimp, etc.

## **Chapter 6**

# **Advanced Mathematics - ODEs and Beyond**

Mathematics

## **Chapter 7**

# **Matrix Operations and Vectors**

Maxima has many matrix capabilities. Unfortunately its vector handling at the time of this writing is not terribly robust, and the reader is cautioned to use it with care.

## **Chapter 8**

# **Programming in Maxima**

Real power user stuff here. We will need some good examples.

## **Chapter 9**

# **Graphics and Forms of Output**

1D and 2D ascii display, T<sub>E</sub>X output, plotting options and examples, etc.

## **Chapter 10**

# **Help Systems and Debugging**

Using the debugger and the online help system

# **Chapter 11**

# **Troubleshooting**

Common errors, their cause and solution

# **Appendix 1 - List of Predefined Keywords, Functions and Constants**

## **Appendix 2 - Advanced Examples**

These examples try to draw on everything in the manual to show you what can be done with Maxima in advanced usages. They assume that you are familiar with most or all of the concepts discussed in the manual - this is NOT a good starting point for new users.

## **Part II**

# **External, Additional, and Contributed Packages**

## **Chapter 12**

# **The Concept of Packages - Expanding Maxima's Abilities**

This chapter will try to give some in depth information on the why and how of Maxima packages. It is true that many "standard" abilities of Maxima, such as vectors, are due to packages, but this part of the book will deal with more specialized contributed packages, such as elliptical integrals, special scientific packages, etc. That way the first part of the book can be rewritten less frequently, but we can still have all the information here.

## **Chapter 13**

# **Available Packages for Maxima**

Maybe we can do this in one chapter, maybe each package will warrant a chapter. Not sure. For now, I'll assume one Section per package, not one Chapter.

# Bibliography

[1] Testing

# Index

Compiling  
  CLISP, [9](#)  
  CMULISP, [10](#)  
  GCL, [8](#)

Debugging  
  Exiting, [20](#)

Exiting, [20](#)

Hiding output, [21](#)

Maple, [6](#)  
Mathematica, [6](#)  
Matlab, [6](#)

Octave, [6](#)

Quitting, [20](#)

Reduce, [6](#)