Getting Started with Maxima

1 Introduction

1.1 About these Maxima files

These files are intended as electronic supplements to the book Principles of NMR Spectroscopy: An Illustrated Guide, David P. Goldenberg, University Science Books, (c) 2016. Their primary purpose is to aid in carrying out quantum-mechanical calculations of the type presented in the second half of the book, Chapters 11-18. This type of calculation can very quickly become very laborious and error prone if carried out by hand and are greatly aided by computer algebra programs such as Maxima, Mathematica and Maple. Although the relative virtues of these programs can be, and have been, discussed at length, Maxima has been chosen here because it is available as open source software in versions for the current major operating systems, Linux, OS X and Windows. There are even versions for Android and iOS (through the SAGE interface)! Although it lacks some of the polish found in the user interfaces of commercial products, Maxima is a very capable program and well suited for the type of calculations presented in the book.
The Maxima files provided as supplements to Principles of NMR Spectroscopy include:
gettingStarted.wxm: This file, a wxMaxima workbook, with background information and a tutorial on some of the elementary Maxima functions used in the other workbooks.
1spin.mac: A macro file containing special functions for quantum-mechanical calculations for a single spin-1/2 particle
2spin.mac: A macro file containing special functions for quantum-mechanical calculations for a scalar-coupled pair of spin-1/2 particles
chapter11.wxmx: A wxMaxima workbook file with calculations following Chapter 11 of the text, for a single spin-1/2.
chapter12.wxmx: A wxMaxima workbook following Chapter 12, on the time dependence of a single spin-1/2 and the effects of pulses.
chapter13.wxmx: A wxMaxima workbook following Chapter 13, on scalar-coupled spin-1/2 pairs.
chapter14.wxmx: A wxMaxima workbook following Chapter 14, the effects of pulses and time evolution on scalar-coupled spin-1/2 pairs
chapter15.wxmx: A wxMaxima workbook following Chapter 15, on heteronuclear and homonuclear COSY experiments
chapter16.wxmx: A wxMaxima workbook following Chapter 16, on heteronuclear experiments
chapter17_1.wxmx: A wxMaxima workbook following Chapter 17, Section 2, on the density matrix for populations of isolated spin-1/2 particles.
chapter17_2.wxmx: A wxMaxima workbook following Chapter 17, Section 2, on the density matrix for populations of scalar-coupled spin pairs.
chapter18_1.wxmx: A wxMaxima workbook following Chapter 18, Section 1 of the text, on the operator basis representation of the density matrix for a population of single spins.
chapter18_2.wxmx: A wxMaxima workbook following Chapter 18, Section 1 of the text, on the operator basis representation of the density matrix for a population of scalar-coupled spin pairs.
These and other files related to the book are available for download through links at: http://uscibooks.com/goldenberg.htm

Please send comments, including bug reports, to this address:
David P. Goldenberg
Department of Biology
University of Utah
257 South 1400 East
Salt Lake City, UT 84112-0840
goldenberg@biology.utah.edu

1.2 About Maxima and wxMaxima

The program Maxima is an example of a computer algebra system, or CAS, a program that carries out symbolic mathematical manipulations, as well as the more common numerical calculations that are provided in other programs and programming languages. Although not the first CAS, Maxima is probably the longest surviving program of this type, dating back to the late 1960s. It is derived from Macsyma, which was developed by the artificial intelligence group at MIT as part of Project MAC, a project initially supported by the US Defense Advanced Research Project Agency (DARPA). The acronym MAC is said to have originally stood for "Mathematics and Computation", but was later associated with "Machine Aided Cognition", "Multiple Access Computer" or "Man and Computer".
After the initial development work at MIT, Macsyma underwent a rather complicated and controversial history of commercial development and eventual release as an open source project, renamed Maxima (a rather unfortunate choice for the age of web searches). Although the commercialization of Macsyma was unsuccessful, the program was clearly the inspiration for the very successful Maple and Mathematica programs. Much more detailed histories can be found in:
de Souza, P. N., Fateman, R. J., Moses, E. & Yann, C. (2004). The Maxima Book.
http://maxima.sourceforge.net/docs/maximabook/maximabook-19-Sept-2004.pdf
Moses, J. (2012). Macsyma: A personal history. J. Symb. Comp., 47, 123–130.
http://dx.doi.org/10.1016/j.jsc.2010.08.018
Maxima is now maintained as an open-source project: http://maxima.sourceforge.net/ Maxima also serves as a component of the much larger SAGE open-source mathematics system: http://www.sagemath.org/ Between, these two projects, there is considerable active development, which promises continued availability and usefulness of Maxima.
Macsyma was written using the computer language Lisp, which was invented in 1958 by the computer and artificial- intelligence pioneer John McCarthy at MIT. The core functionality of Maxima continues to be coded in Lisp (specifically Common Lisp), and this strongly influences its behavior. The name "Lisp" stands for "List processing", and lists (of numbers, symbols, functions and other objects, including other lists) are the core data structure of the Lisp language and of Maxima's own language. Lists are used to input multiple parameters in Maxima commands, and enclose multiple outputs. The elements of Maxima lists are enclosed by square brackets, [ ], and it is very important not to confuse these symbols with parentheses, as discussed further below.
The basic Maxima program uses a simple command-line interface, in which commands are typed into a terminal window and results are output as simple text. Commands and results can also be read into the program and output as text files. Although functional, this kind of interface now feels rather old fashioned, and many users prefer a graphical interface with windows and menus controlled with a mouse. Fortunately, "front ends" with graphical interfaces to Maxima have been developed as open software. In addition, SAGE offers a graphical interface to Maxima. Of these, wxMaxima appears to be undergoing the most active development currently and has been used for this project. Versions for Windows, OS X and Linux can be downloaded from the wxMaxima project page:
http://andrejv.github.io/wxmaxima/
Installation of wxMaxima requires a version of Maxima as well, along with the graphing program GnuPlot if the plotting functions are to be used. Configuration

requires some care, and the installation instructions should be followed carefully.
The wxMaxima interface provides a notebook type environment, similar to that used in Mathematica, Maple and Matlab, in which individual commands are typed into "cells", and the output appears below each input. The output is nicely formatted in standard mathematical notation (rather than simple text), but the output elements can be copied and pasted into new input cells.
As an alternative to typing, many Maxima commands can be executed from menus or buttons in window panes. (The panes can optionally be displayed using commands in the Maxima menu. These graphical interface elements generate the text form of the commands, automatically enter them into a cell and execute the commands. In some cases, the graphical commands automatically apply the command to the output of the last command executed. In others, the command opens a dialog box into which the the user is prompted to enter parameters for the command. Upon closing the dialog box, the command is generated and executed. The user is free to use the text entry method, the menus or the panel buttons interchangeably during a session. With experience, users are likely to choose increasingly to use direct text entry, but the graphical methods provide a very effective way to find and explore new commands.
Another useful feature of the wxMaxima interface is the ability to use special cells to add text (as in this cell) or headers to organize a notebook into sections and subsections. Commands to add these special sections are found in the "Cell" menu.

1.3 File formats

The work a user does in wxMaxima can be saved in three file formats:
Maxima macro files, with the extension .mac
The original wxMaxima file format, with extension .wxm
A newer XTML-based wxMaxima format, with extension .wxmx
All of these are simple text files, ensuring that they can be easily read without the Maxima or wxMaxima programs.
Maxima macro files created with wxMaxima contain only the commands entered in cells and text comments. These files cannot be opened from the graphical interface, but can be opened using a typed load command. Upon loading a .mac file, wxMaxima will not usually display any output, but will execute the commands in the file.
Macro files are typically used to store commands defining variables and functions that can then be used again in another session. Two macro files are provided for spin calculations, 1spinLib.mac and 2spinLib.mac.
When a work session is saved in a .wxm file and then reopened in wxMaxima, the session cells are recreated, including text section and subsection cells. However, the output is not saved in this file format. The output can be recreated using the "Evaluate All Cells" command in the "Cell" menu. Be aware, however, that the cells will be executed from the top of the document to the bottom, which may not have been the order in which the cells were executed during the previous session, which may lead to a different state.
The .wxmx file format has the advantage of saving both the input and output of cells, including images generated by the plotting commands, and is the format used to save these workbooks. When a .wxmx file is opened, everything from the previous session will appear. However, the commands in the file will not be executed, so that variable and functions are not yet defined.
The "File" menu of wxMaxima includes an "Export" command, which provides options to save a session in the form of of a a web page (.html together with image files for all of the output cells) or a LaTeX file. LaTeX is a sophisticated document preparation system, and the files generated in this way can be used to generate a high-quality pdf files representing a wxMaxima workbook, but some tweaking of the LaTeX file is likely to be necessary in order for it to be used.
On a Macintosh computer, a pdf file representing a workbook can also be created using the "Print" command in the "File" menu and the save as pdf option in the print dialog box.

1.4 A few notes on syntax

There are often many correct ways to write out a given mathematical expression, sometimes using different symbols to represent the same thing. However, computer programs are still not nearly as good as humans at recognizing expressions written in different ways, and they generally demand strict adherence to syntactic rules.
In Maxima, the symbols for the basic arithmetic operations are:
addition: +
subtraction: -
multiplication: *
division: /
exponentiation: ^
Particular care must be taken in using parenthesis, ( ), and square brackets, [ ]. The two major uses of parentheses are: 1. To enclose expressions that should be evaluated before operations outside of the parentheses. For instance (5+2)*3 is evaluated as 7*3=21, rather than as 5+2*3=11 2. To enclose the arguments of a function, such as sin(x) Square brackets are used in very different way, to enclose the elements of lists. As noted earlier, the core functionality of Maxima is written in the computer language Lisp, and lists serve as the basic data structure of both Lisp and Maxima. When commands can accept variable numbers of arguments, the arguments are generally input as lists, enclosed by brackets, and commands that output multiple elements use lists to enclose them.
Two other special symbols are of note, and are discussed further as examples are introduced. These are the assignment operator symbols ":" and ":=". In many other computer languages, the equals sign, "=", is used as an assignment operator, but it is important to distinguish the difference between an assignment operator and the usual meaning of equality in mathematics. In mathematics, the equals sign is used to state a relationship, that the two things on each side of it are equal to one another. In programming languages, an assignment operator *establishes* a relationship between two objects. In languages that use the equals sign as the assignment operator (such as Fortran, C, Perl, Python and many others), statements such as a=5 are interpreted to mean, "Take the value 5 and associate it with a variable called 'a'." When the variable "a" appears later, it is replaced with the stored value.
Because Maxima is often called on to work with expressions that include the equals sign in its mathematical meaning, "=" is not used as an assignment operator. The symbol ":" is used to assign values to symbols. For instance, the expression a:5 is equivalent to "a=5" in the languages mentioned above. But, symbols can be assigned many other kinds of objects than just numerical values. Symbols can represent other symbols, expressions and lists of other objects.
The other assignment operator, ":=", is used for defining mathematical functions, such as the definition: f(x):=x^2
Two small points:
1. When entering expressions in Maxima, the end of the expression must be followed by a semicolon, before pressing the enter key. In wxMaxima, however, the interface does this for you automatically. Having been spoiled in this fashion, it can be bewildering and frustruting when trying to use the standard terminal interface.
2. Usually, we want to see the result of a calculation after entering an expression. In some cases, however, the output may be very long and ugly, and all that we are trying to do is assigning this result to a variable. Adding the dollar sign, "$" at the end of an expression after evaluating it will suppress the output.

1.5 Maxima documentation

One of the significant weaknesses of Maxima as a user-friendly software product, especially in comparison to commercial products such as Mathematica and Maple, is the rather limited and scattered documentation available for it. So far as I am aware, for instance, there are no published books about Maxima (except for a self-published book described below) such as there are for many other commercial and open-source programs.
There is, however, a standard reference manual that is provided in different electronic forms. These include a version embedded in wxMaxima and available using the "Maxima Help" command in the "Help" menu. The manual is also available online, as html and as a pdf document at:
http://maxima.sourceforge.net/docs/manual/maxima.html
In addition, there are a large number of tutorials and other documents that have been written to introduce users to Maxima. Many of these are, like this one, targeted to specific audiences or applications. Links to many of them can be found at: http://maxima.sourceforge.net/documentation.html
Some tutorials specific to wxMaxima can be found at: http://andrejv.github.io/wxmaxima/help.html
Among the various Maxima tutorials available on the internet, a collection of special note is by Gilberto E. Urroz of Utah State University: http://www.neng.usu.edu/cee/faculty/gurro/Maxima.html These include both pdf files and some wxMaxima worksheets. The pdf files are also available as a printed book, through a link on the web page.

Information about a Maxima function or command can be obtained by typing "?" followed by the function.

```
(%i1) ? log;
```

-- Function: log (<x>)
Represents the natural (base e) logarithm of <x>.
Maxima does not have a built-in function for the base 10 logarithm
or other bases. 'log10(x) := log(x) / log(10)' is a useful
definition.
Simplification and evaluation of logarithms is governed by several
global flags:
'logexpand'
causes 'log(a\textasciicircumb)' to become 'b*log(a)'. If it is set to
'all', 'log(a*b)' will also simplify to 'log(a)+log(b)'. If
it is set to 'super', then 'log(a/b)' will also simplify to
'log(a)-log(b)' for rational numbers 'a/b', 'a\#1'.
('log(1/b)', for 'b' integer, always simplifies.) If it is
set to 'false', all of these simplifications will be turned
off.
'logsimp'
if 'false' then no simplification of '\%e' to a power
containing 'log''s is done.
'lognegint'
if 'true' implements the rule 'log(-n)' -> 'log(n)+\%i*\%pi' for
'n' a positive integer.
'\%e_to_numlog'
when 'true', 'r' some rational number, and 'x' some
expression, the expression '\%e\textasciicircum(r*log(x))' will be simplified
into 'x\textasciicircumr'. It should be noted that the 'radcan' command also
does this transformation, and more complicated transformations
of this as well. The 'logcontract' command "contracts"
expressions containing 'log'.
There are also some inexact matches for `log'.
Try `?? log' to see them.

(%o1) true

2 Simple calculations with numbers and symbols

2.1 Arithmetic

Maxima can be used for a wide range of mathematics, from the very simple to the very complicated. At its most basic, the program can serve as a simple calculator. Expressions to be evaluated are entered at a prompt and then evaluated by pressing the Enter key:

```
(%i2) 2+2;
```

(%o2) 4

When using the wxMaxima interface, each entered expression and any output generated is placed within a "cell", as indicated by the braces to the left of the input and output. Cells can also contain text, like this one, or headings to help organize the worksheet.

```
(%i3) 4-3;
```

(%o3) 1

Division is written with a forward slash

```
(%i4) 4/2;
```

(%o4) 2

Multiplication is indicated by an asterisk

```
(%i5) 3*4;
```

(%o5) 12

Exponents are indicated by the ^ character

```
(%i6) 3^2;
```

(%o6) 9

When, as in the examples above, the result is an integer, the expression is evaluated and output. However, if the result is not an integer, integers are retained in the output.

```
(%i7) 2/3;
```

(%o7) $\frac{2}{3}$

Evaluation of a non-integer result can be forced by writing one or more of the terms in decimal form.

```
(%i8) 2/3.0;
```

(%o8) 0.6666666666666666

```
(%i9) 2.0/3;
```

(%o9) 0.6666666666666666

Evaluation of a non-integer result can also be forced by using the float function.

```
(%i10) float(2/3);
```

(%o10) 0.6666666666666666

Often, it is useful to be able to use the previous result in the next calculation. The % character will return the previous result.

```
(%i11) 4+16;
```

(%o11) 20

```
(%i12) %;
```

(%o12) 20

```
(%i13) %/3;
```

(%o13) $\frac{20}{3}$

2.2 Assigning values to symbols

As in other computer languages, numerical values can be assigned to variables represented as symbols. For instance:

```
(%i14) a:5;

(%o14)  5
```

In the expression above, the ":" serves as an assignment operator, assigning the value to its right to the symbol on its left. In many computer languages, the equals sign, "=", is used as the assignment operator. But, in Maxima the equals sign has a different use, more closely related to its traditional mathematical meaning, discussed further below.
After being assigned to a value, the symbol can be used in other expressions:

```
(%i15) a;

(%o15)  5

(%i16) a+10;

(%o16)  15
```

After having one value assigned to it, a symbol can be reassigned:

```
(%i17) a:7;

(%o17)  7

(%i18) a;

(%o18)  7

(%i19) a+10;

(%o19)  17
```

And, assignments can be nullified with the kill command:

```
(%i20) kill(values);

(%o20)  done

(%i21) a;

(%o21)  a
```

Symbols can also be assigned to other symbols

```
(%i22) a:c;

(%o22)  c

(%i23) a;

(%o23)  c

(%i24) b+a;

(%o24)  c + b
```

The wxMaxima interface recognizes Greek letters and will display them in its output lines.

```
(%i25) alpha;

(%o25)  α

(%i26) beta;

(%o26)  β

(%i27) gamma;
```

```
(%i27) gamma;
```

(%o27) γ

```
(%i28) delta;
```

(%o28) δ

```
(%i29) Delta;
```

(%o29) Δ

The Greek letters can be assigned to values

If entered just as "pi", pi has no special value

```
(%i30) float(pi);
```

(%o30) π

However, if preceded with %, pi is recognized as the irrational number approximately equal to 3.141592653589793

```
(%i31) %pi;
```

(%o31) π

```
(%i32) float(%);
```

(%o32) 3.141592653589793

Similarly, the special numbers e and i are entered as %e and %i

```
(%i33) float(%e);
```

(%o33) 2.718281828459045

```
(%i34) %i*%i;
```

(%o34) -1

2.3 Imaginary and complex numbers

Imaginary and complex numbers are specified, using %i to represent the unit imaginary number.

```
(%i35) %i^2;
```

(%o35) -1

```
(%i36) sqrt(-1);
```

(%o36) i

A complex number

```
(%i37) c:3+%i*5;
```

(%o37) $5 \cdot i + 3$

The command conjugate return the conjugate of a complex number

```
(%i38) conjugate(c);
```

(%o38) $3 - 5 \cdot i$

The real and imaginary parts of a complex number are returned by the commands realpart and imagpart

```
(%i39) realpart(c);
```

(%o39) 3

```
(%i40) imagpart(c);
```

$$(\%o40) \quad 5$$

```
(%i41) imagpart(conjugate(c));
```

$$(\%o41) \quad -5$$

By default, variables in Maxima are assumed to be real valued, a fact that can be demonstrated using the conjugate command

```
(%i42) kill(a);
```

$$(\%o42) \quad done$$

```
(%i43) a;
```

$$(\%o43) \quad a$$

```
(%i44) conjugate(a);
```

$$(\%o44) \quad a$$

```
(%i45) realpart(a);
```

$$(\%o45) \quad a$$

```
(%i46) imagpart(a);
```

$$(\%o46) \quad 0$$

However, a variable can be declared to be complex, without actually assigning a value to it.

```
(%i47) declare(a,complex);
```

$$(\%o47) \quad done$$

```
(%i48) conjugate(a);
```

$$(\%o48) \quad \overline{a}$$

```
(%i49) realpart(a);
```

$$(\%o49) \quad \mathrm{realpart}\,(a)$$

```
(%i50) imagpart(a);
```

$$(\%o50) \quad \mathrm{imagpart}\,(a)$$

Even though Maxima does not know the value of a, it does not assume that the imaginary part is zero, as it would for a variable that had not been declared to be complex.

Before proceeding, we kill all of the previous assignments

```
(%i51) kill(values, functions);
```

$$(\%o51) \quad done$$

This also returns a to is default properties as a real number

```
(%i52) conjugate(a);
```

$$(\%o52) \quad \overline{a}$$

3 Algebra

What makes Maxima and other computer algebra programs, such as Mathematica and Maple, special is their ability to carry out mathematical manipulations with symbols, rather than just numerical values.

In most standard computer languages (like Fortran, C, Perl, Python, etc), the following expressions would result in an error, because the computer doesn't know what to do with an unassigned variable.

```
(%i53) a+b;
```

$$(\%o53) \quad b + a$$

```
(%i54) (a+b)/c;
```

$$(\%o54) \quad \frac{a+b}{c}$$

```
(%i55) (a+b)*(c+d);
```

$$(\%o55) \quad (b+a)\cdot(d+c)$$

The form of an expression can be manipulated by various Maxima commands, many of which are directly available in the wxMaxima menu bar. For instance we can expand the expression above:

```
(%i56) %;
```

$$(\%o56) \quad (b+a)\cdot(d+c)$$

```
(%i57) expand(%);
```

$$(\%o57) \quad b\cdot d + a\cdot d + b\cdot c + a\cdot c$$

This command is found as "Expand Expression" in the "Simplify" menu. When called from the menu bar, commands generally act on the result of the last command, designated %. The previous result can then be factored

```
(%i58) factor(%);
```

$$(\%o58) \quad (b+a)\cdot(d+c)$$

or simplified (according the program's definition of simplification)

```
(%i59) ratsimp(%);
```

$$(\%o59) \quad (a+b)\cdot d + (a+b)\cdot c$$

The "Simplify" menu and its submenus includes a variety of functions for rearranging algebraic and trigonometric functions. Choosing the right command to generate a desired form can sometimes be challenging, and often requires a trial and error approach.

3.1 Solving equations

Maxima can also solve many kinds of equations or systems of equations, using purely symbolic manipulations. First, as a simple example, we define the standard quadratic equation.

```
(%i60) quadEq:(a*x^2+b*x+c=0);
```

$$(\%o60) \quad a\cdot x^2 + b\cdot x + c = 0$$

There are subtle, but important, features to this operation, centered around the symbols ":" and "=". As noted before, in Maxima, ":" is an assignment operator, that is, it is used to assign a value to a symbol.
Thus, the symbol quadEq now represents the equation that it was assigned to:

```
(%i61) quadEq;
```

$$(\%o61) \quad a\cdot x^2 + b\cdot x + c = 0$$

In many computer languages, the equals sign, "=", is used as the assignment operator. But, in Maxima equations are a class of objects that can be manipulated, and the equals sign is used in its more traditional mathematical sense. As a consequence, an expression like a=5, which might look like an assignment, doesn't actually do anything on its own.

```
(%i62) a=5;
```

$$(\%o62) \quad a = 5$$

```
(%i63) a;
```

(%o63) a

Having defined an equation, we can then apply the solve command to it:

```
(%i64) solve(quadEq,x);
```

(%o64) $[x = -\frac{b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}, x = \frac{\sqrt{b^2 - 4 \cdot a \cdot c} - b}{2 \cdot a}]$

Note, that the solve command requires two arguments, the equation and the variable that is to be solved for. In this case, we get the 2 familiar solutions for a quadratic equation.
We can also solve for values of a, b or c that satisfy the equation, in terms of x and the other two variables:

```
(%i65) solve(quadEq,a);
```

(%o65) $[a = -\frac{c + b \cdot x}{x^2}]$

```
(%i66) solve(quadEq,b);
```

(%o66) $[b = -\frac{c + a \cdot x^2}{x}]$

```
(%i67) solve(quadEq,c);
```

(%o67) $[c = -a \cdot x^2 - b \cdot x]$

The solve command, and related commands, can be accessed in the "Equations" menu of wxMaxima. The commands in this menu generally open a dialog box into which the equations and variables are specified.

3.2 Substituions

Another useful technique is substitution. In the example with the quadratic equation, the symbols a, b and c were left unassigned, so that we obtained a general result:

```
(%i68) solve(quadEq,x);
```

(%o68) $[x = -\frac{b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}, x = \frac{\sqrt{b^2 - 4 \cdot a \cdot c} - b}{2 \cdot a}]$

But, we might want to solve the equation with specific values of a, b and c. We could do this by redefining the equation with the specific values, or defining a new equation

```
(%i69) quadEq2:(5*x^2-13*x+2=0);
```

(%o69) $5 \cdot x^2 - 13 \cdot x + 2 = 0$

```
(%i70) solve(quadEq2,x);
```

(%o70) $[x = -\frac{\sqrt{129} - 13}{10}, x = \frac{13 + \sqrt{129}}{10}]$

But, we can also do this with the general form of the equation and substitutions, using the subst function.
The subst function can be used with two syntaxes. In the first, the function takes 3 arguments: The numerical value or symbol to be introduced. The symbol to be replaced. The expression to be altered by the substituion. To replace the symbol a in the quadratic equation defined earlier with the numerical value of 5, we type

```
(%i71) subst(5,a,quadEq);
```

(%o71) $5 \cdot x^2 + b \cdot x + c = 0$

When the other syntax for the subst function is used, the function accepts two arguments, an equation defining the substitution and the expression. Using this syntax, the same substitution shown above is written as:

```
(%i72) subst(a=5,quadEq);
```

(%o72) $5 \cdot x^2 + b \cdot x + c = 0$

Importantly, the subst function does not alter the definition of the expression. Only the output of the subst function is affected. Thus, the expression associated with quadEq is still:

```
(%i73) quadEq;
```

(%o73) $a \cdot x^2 + b \cdot x + c = 0$

Multiple substitutions can be made at once using the syntax with only two arguments. In this case, the individual equations defining the substitutions are enclosed in brackets to designate a list

```
(%i74) subst([a=5,b=-13,c=2],quadEq);
```

(%o74) $5 \cdot x^2 - 13 \cdot x + 2 = 0$

To solve the equation with these substitutions, we can apply the substitution command to the previous result:

```
(%i75) solve(%,x);
```

(%o75) $[x = -\frac{\sqrt{129} - 13}{10}, x = \frac{13 + \sqrt{129}}{10}]$

Or we can include the substitution operation in the first argument of the solve command

```
(%i76) solve(subst([a=5,b=-13,c=2],quadEq),x);
```

(%o76) $[x = -\frac{\sqrt{129} - 13}{10}, x = \frac{13 + \sqrt{129}}{10}]$

3.3 Parts of expressions

Using the subs function is sometimes trickier than it looks. For instance, suppose that we try to substitute a new variable, d, for b*x+c

```
(%i77) subst(d, b*x+c,quadEq);
```

(%o77) $a \cdot x^2 + b \cdot x + c = 0$

This also doesn't work using the alternative syntax:

```
(%i78) subst(b*x+c=d, quadEq);
```

(%o78) $a \cdot x^2 + b \cdot x + c = 0$

The reason for this failure can be found in the fine print of the documentation:
? subst; -- Function: subst (<a>, <b>, <c>) Substitutes <a> for <b> in <c>. <b> must be an atom or a complete subexpression of <c>.
Expressions in Maxima have a distinct structure, made up of parts, parts of parts and the smallest units, called atoms.
As an example, we can define an expression composed of the terms in the quadratic equation

```
(%i79) expr:a*x^2+b*x+c;
```

(%o79) $a \cdot x^2 + b \cdot x + c$

The structure of an expression can be dissected using the part function, which requires at least two arguments, an expression and an integer identifying a part.

```
(%i80) part(expr,0);
```

(%o80) $+$

```
(%i81) part(expr,1);
```

(%o81) $a \cdot x^2$

```
(%i82) part(expr,2);
```

(%o82) $b \cdot x$

```
(%i83) part(expr,3);
```

(%o83) c

part 1 is, in turn made up of parts, and the part function can be applied sequentially

```
(%i84) part(part(expr,1),0);
```

(%o84) $*$

A simpler way to type this is to provide additional arguments to indicate sub parts.

```
(%i85) part(expr,1,1);
```

(%o85) a

```
(%i86) part(expr,1,2);
```

(%o86) x^2

```
(%i87) part(expr,1,3);
```

part: fell off the end. -- an error. To debug this try: debugmode(true);

This means that x^2 cannot be divided into smaller parts.
Similarly, part 2 of the express, can be divided further

```
--> part(expr,2,0);
```

(%o218) $*$

```
--> part(expr,2,1);
```

(%o219) b

```
--> part(expr,2,2);
```

(%o220) x

Any of the parts in an expression, including the operators can be substituted with the subst function

```
--> subst("+","*",expr);
```

(%o221) $x^2 + x + c + b + a$

In this example, each of the multiplication operators, "*", has been substituted with an addition operator, "+".
a, b, and c can all be replaced with substituions

```
--> subst(d,a, expr);
```

(%o222) $d \cdot x^2 + b \cdot x + c$

```
--> subst(d,b,expr);
```

(%o223) $a \cdot x^2 + d \cdot x + c$

```
--> subst(d,c,expr);
```

(%o224) $a \cdot x^2 + b \cdot x + d$

And x can be replaced

```
--> subst(y, x, expr);
```

(%o225) $a \cdot y^2 + b \cdot y + c$

Also, the product b*x can be replaced with a number, with a symbol, or even another expression

```
--> subst(4,b*x,expr);
```

(%o227) $a \cdot x^2 + c + 4$

```
--> subst(d, b*x, expr);
```

(%o229) $a \cdot x^2 + d + c$

```
--> subst(d*5,b*x,expr);
```

(%o228) $a \cdot x^2 + 5 \cdot d + c$

However, because the term b*x + c is not an identifiable part in the expression, it is not recognized by the subst command and cannot be replaced.

```
--> subst(4,b*x+c, expr);
```

(%o230) $a \cdot x^2 + b \cdot x + c$

There is another, related, function called ratsubst. This function knows a bit more about algebra than does subst, and can recognize terms that consist of more than one atom, or part, of an expression.
For instance, the following command with ratsub behaves as we might expect

```
--> ratsubst(a,x+y,x+y+z);
```

(%o246) $z + a$

The same example with subst fails, for the reason explained above.

```
--> subst(a,x+y,x+y+z);
```

(%o244) $z + y + x$

Another example, using the expression defined above

```
--> expr;
```

(%o247) $a \cdot x^2 + b \cdot x + c$

```
--> ratsubst(d,a*x^2+b*x,expr);
```

(%o248) $d + c$

But, the following doesn't work with ratsubst, though it seems it should:

```
--> ratsubst(d,b*x+c,expr);
```

(%o266) $a \cdot x^2 + b \cdot x + c$

Sometimes Maxima can be rather inscrutable!

4 Built-in Functions

Maxima contains a large number of built-in functions. Two functions have already been introduced: float(x) and sqrt(x).

```
--> float(1/3);
```

(%o75) 0.3333333333333333

```
--> sqrt(9);
```

(%o76) 3

Other built-in functions include the trigonometric functions and their inverses, and logarithmic and exponential functions.

```
--> sin(0);
```

(%o77) 0

```
--> sin(%pi/2);
```

(%o78) 1

```
--> sin(%pi/4);
```

(%o79) $\frac{1}{\sqrt{2}}$

```
--> cos(0);
```

(%o80) 1

```
--> cos(%pi/2);
```

(%o81) 0

```
--> cos(%pi/4);
```

(%o82) $\frac{1}{\sqrt{2}}$

```
--> tan(0);
```

(%o83) 0

```
--> tan(%pi/4);
```

(%o84) 1

```
--> tan(%pi/2);
```

tan: $\frac{\pi}{2}$ isn't in the domain of tan. -- an error. To debug this try: debugmode(true);

```
--> acos(1);
```

(%o86) 0

```
--> asin(1);
```

(%o87) $\frac{\pi}{2}$

```
--> atan(1);
```

(%o88) $\frac{\pi}{4}$

The function log(x) is the natural logarithm (base e), not the common logarithm (base 10)

```
--> log(1);
```

(%o89) 0

```
--> log(10);
```

(%o90) $\log(10)$

```
--> log(10.0);
```

(%o91) 2.302585092994046

```
--> log(%e);
```

(%o92) 1

However, the common logarithm is available in a shared library included in the wxMaxima distribution and can be loaded into a session

```
--> load(log10);
```

(%o93) */usr/local/Cellar/maxima/5.36.1/share/maxima/5.36.1/share/contrib/log10.mac*

```
--> log10(10);
```

(%o94) 1

Exponentials of the Euler number, e, can be written either as %e^x or as exp(x)

```
--> %e^1;
```

(%o95) e

```
--> float(%);
```

(%o96) 2.718281828459045

```
--> exp(1);
```

(%o97) e

```
--> exp(0);
```

(%o98) 1

5 Trigonometric identities and simplification

The trigonometric functions arise frequently in NMR, and it is often desirable to convert an expression into another form.
There are several Maxima functions for manipulating expressions that contain trigonometric functions, including the following, which can be accessed in the wxMaxima Simplify>Trigonometric Simplification submenu:
Command Simplify Trigonometric trigreduce() Reduce Trigonometric trigexpand() Expand Trigonometric trigrat() Cannonical form
The commands trigreduce and trigexpand interconvert trigonometric expressions containing sums of angles and products of trig functions. For instance, the cosine of a sum of angles is converted into an expression containing products of trig functions with simplified arguments

```
--> trigexpand(cos(a+b));
```

(%o273) $\cos(a)\cdot\cos(b)-\sin(a)\cdot\sin(b)$

This is reversed with trigreduce

```
--> trigreduce(%);
```

(%o274) $\cos(b+a)$

A product of two trig functions can be converted into a form made up of trig functions of sums.

```
--> trigreduce(cos(a)*cos(b));
```

(%o275) $$\frac{\cos(b+a)}{2}+\frac{\cos(b-a)}{2}$$

This is reversed, with trigexpand, followed by simplification with ratsimp

```
--> trigexpand(%);
```

(%o276) $$\frac{\sin(a)\cdot\sin(b)+\cos(a)\cdot\cos(b)}{2}+\frac{\cos(a)\cdot\cos(b)-\sin(a)\cdot\sin(b)}{2}$$

```
--> ratsimp(%);
```

(%o277) $\cos(a)\cdot\cos(b)$

another example

```
--> trigreduce(cos(a)*sin(b));
```

(%o295) $$\frac{\sin(b+a)}{2}+\frac{\sin(b-a)}{2}$$

```
--> trigexpand(%);
```

(%o296) $$\frac{\cos(a)\cdot\sin(b)+\sin(a)\cdot\cos(b)}{2}+\frac{\cos(a)\cdot\sin(b)-\sin(a)\cdot\cos(b)}{2}$$

```
--> ratsimp(%);
```

(%o297) $\cos(a)\cdot\sin(b)$

expansion of an angular product

```
--> trigexpand(sin(3*b));
```

(%o300) $3\cdot\cos(b)^2\cdot\sin(b)-\sin(b)^3$

```
--> trigreduce(%);
```

(%o301) $$\frac{3\cdot\sin(3\cdot b)+3\cdot\sin(b)}{4}+\frac{\sin(3\cdot b)}{4}-\frac{3\cdot\sin(b)}{4}$$

```
--> ratsimp(%);
```

(%o302) $\sin(3\cdot b)$

Expansion of a trigonometric function of the sum of three angles

```
--> trigexpand(cos(a+b+c));
```

(%o290) $-\cos(a)\cdot\sin(b)\cdot\sin(c)-\sin(a)\cdot\cos(b)\cdot\sin(c)-\sin(a)\cdot\sin(b)\cdot\cos(c)+\cos(a)\cdot\cos(b)\cdot\cos(c)$

```
--> trigrat(%);
```

(%o291) $\cos(c+b+a)$

```
--> trigsimp(%);
```

(%o284) $(-\cos(a)\cdot\sin(b)-\sin(a)\cdot\cos(b))\cdot\sin(c)+(\cos(a)\cdot\cos(b)-\sin(a)\cdot\sin(b))\cdot\cos(c)$

```
--> trigrat(%);
```

(%o285) $\cos(c+b+a)$

Sometimes it is useful to separate out one term in a sum making up the argument of a trigonometric function, while leaving others alone. Here is one way to do that, beginning, again, with the sum of three angles.

```
--> trigexpand(cos(a+b+c));
```

(%o140) $-\cos(a)\cdot\sin(b)\cdot\sin(c)-\sin(a)\cdot\cos(b)\cdot\sin(c)-\sin(a)\cdot\sin(b)\cdot\cos(c)+\cos(a)\cdot\cos(b)\cdot\cos(c)$

Suppose that we want to treat a separately from b and c. We make substitutions for cos(a) and sin(a)

```
--> subst([cos(a)=x,sin(a)=y],%);
```

(%o141) $-\cos(b)\cdot\sin(c)\cdot y-\sin(b)\cdot\cos(c)\cdot y-\sin(b)\cdot\sin(c)\cdot x+\cos(b)\cdot\cos(c)\cdot x$

Then, convert the terms for a and b back into sums of angles

```
--> trigreduce(%);
```

(%o142) $\cos(c+b)\cdot x-\sin(c+b)\cdot y$

Finally, substitute the terms cos(a) and sin(b) back.

```
--> subst([x=cos(a),y=sin(a)],%);
```

(%o143) $\cos(a) \cdot \cos(c+b) - \sin(a) \cdot \sin(c+b)$

More details about the trigsimp, trigexpand and trigrat Maxima functions can be obtained by from the builtin helpl system, by typing "?" followed by the function.

```
--> ? trigsimp;
```

-- Function: trigsimp (<expr>) Employs the identities sin(x)\textasciicircum2 + cos(x)\textasciicircum2 = 1 and cosh(x)\textasciicircum2 - sin

(%o288) true

```
--> ? trigexpand;
```

-- Function: trigexpand (<expr>) Expands trigonometric and hyperbolic functions of sums of angles and of multiple angles occurring in <expr

(%o287) true

```
--> ? trigrat;
```

-- Function: trigrat (<expr>) Gives a canonical simplified quasilinear form of a trigonometrical expression; <expr> is a rational fraction of sev

(%o286) true

6 User defined functions

Functions of one or more variables are easily defined. Functions are defined using the assignment operator, :=, which is distinguished from the assignment operator for symbols, : For instance

```
--> f(x):=x^2;
```

(%o99) $\mathrm{f}(x) := x^2$

```
--> f(2);
```

(%o100) 4

```
--> f(9);
```

(%o101) 81

A function of two variables

```
--> g(x,y):=cos(x)+sin(y);
```

(%o102) $\mathrm{g}(x, y) := \cos(x) + \sin(y)$

```
--> g(0,0);
```

(%o103) 1

```
--> g(%pi,%pi);
```

(%o104) -1

```
--> g(0,%pi/2);
```

(%o105) 2

User-defined functions can be used within other functions

```
--> h(x):=f(x)+2*x;
```

(%o106) $\mathrm{h}(x) := \mathrm{f}(x) + 2 \cdot x$

```
--> h(1);
```

(%o107) 3

```
--> h(3);
```

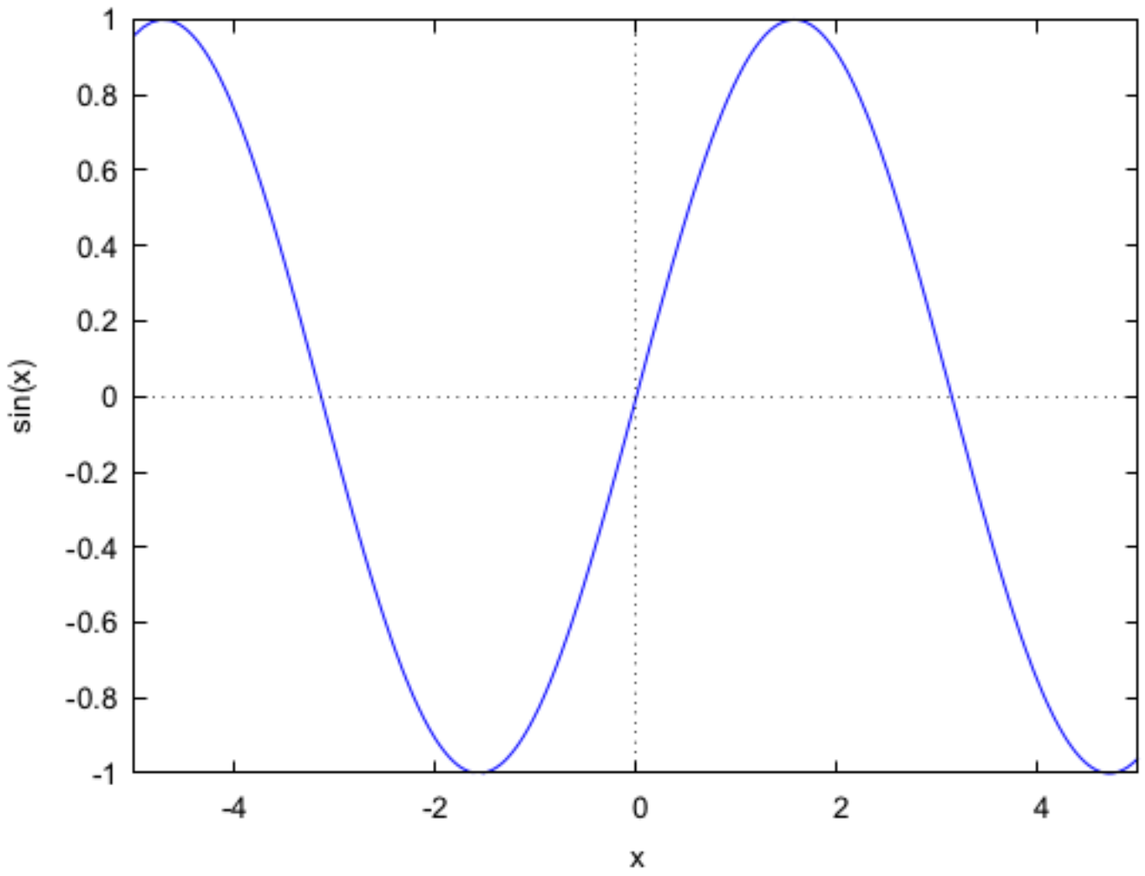
(%o108) 15

7 Plotting functions

wxMaxima provides a simple mechanism to plot functions or data, using the graphing program GnuPlot. Using this feature requires that a special version of GnuPlot be installed alongside wxMaxima and that the programs be properly linked to each other. Details are found in the wxMaxima installation instructions.
The plot functions are most easily accessed using the commands in the wxMaxima Plot menu. These menu commands open dialog boxes in which the function to be plotted is specified, along with various plotting parameters. From these parameters, a plotting command is generated and evaluated.
The example below plots sin(x) versus x for x between -5 and 5.

```
--> wxplot2d([sin(x)], [x,-5,5]);
```

(%t7)

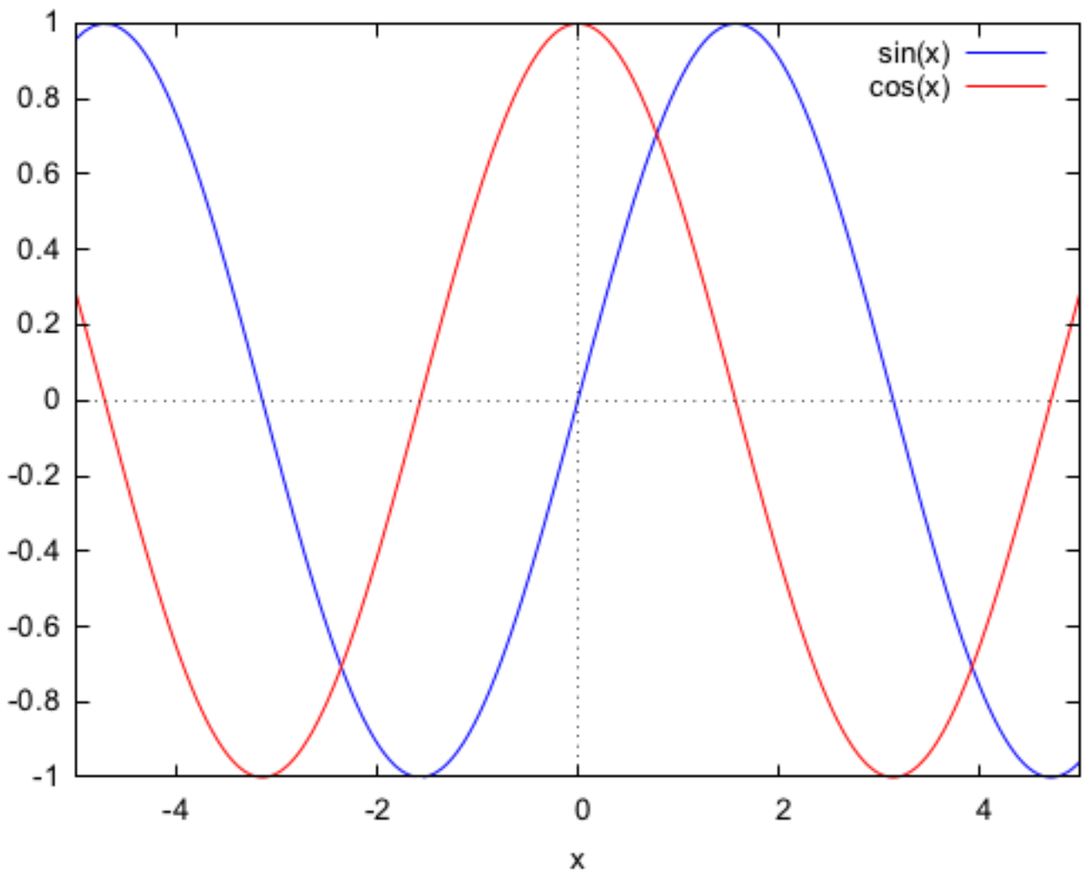


(%o7)

Two functions plotted on the same graph

```
--> wxplot2d([sin(x),cos(x)], [x,-5,5]);
```

(%t110)

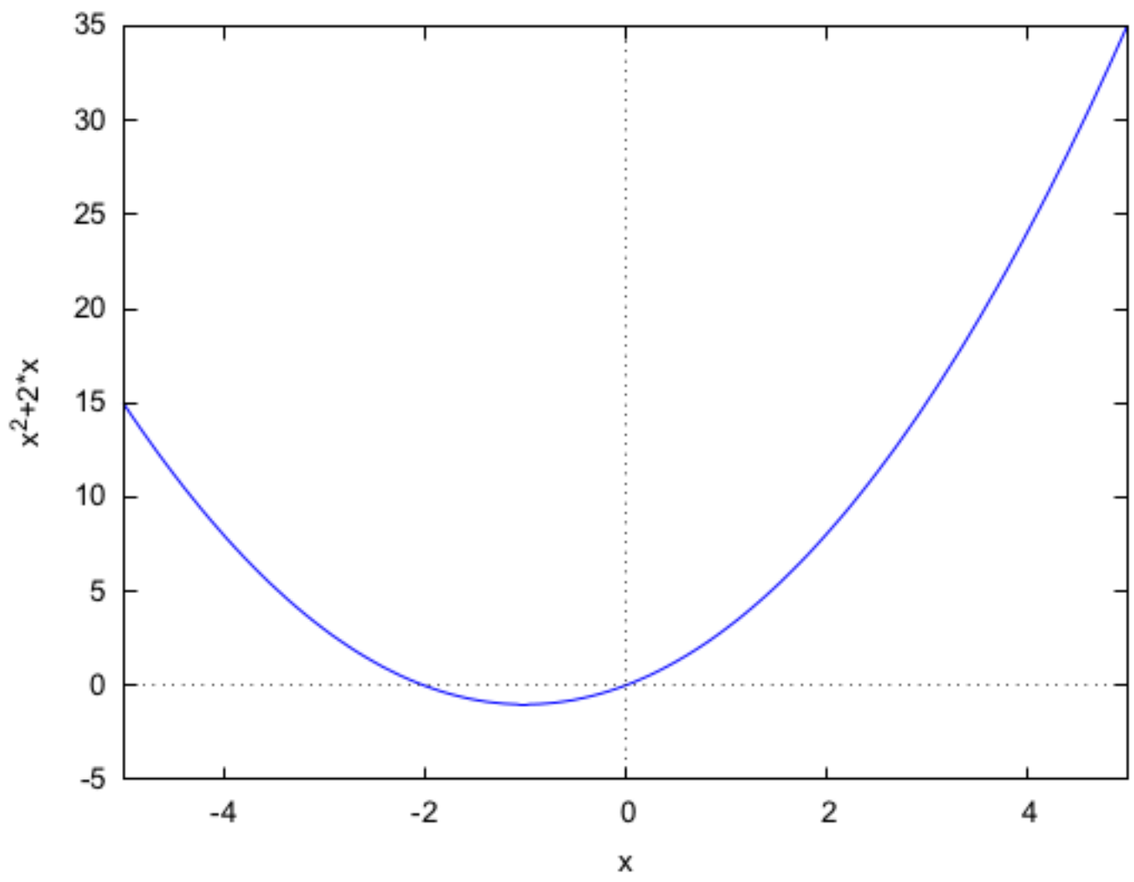


(%o110)

A graph of one of the functions defined above, h(x):=f(x)+2*x; where f(x) was previously defined as f(x):=x^2;

```
--> wxplot2d([h(x)], [x,-5,5]);
```

(%t111)

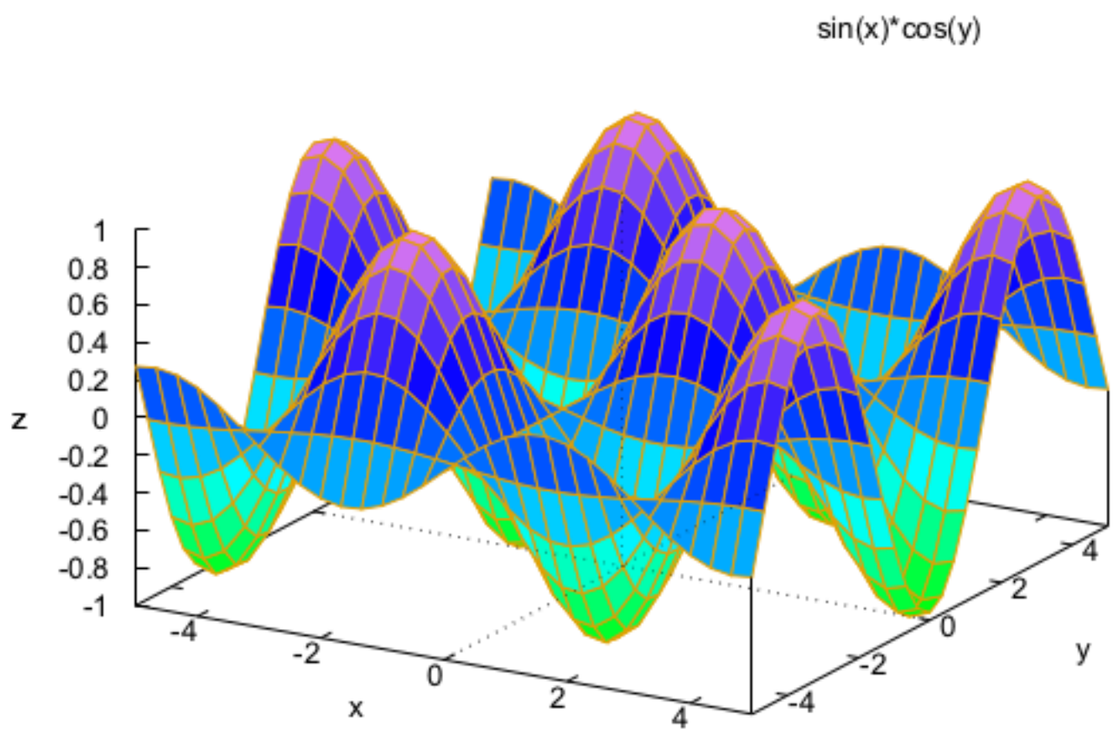


(%o111)

Three-dimensional plots are created with the wxplot3d command

```
--> wxplot3d(sin(x)*cos(y), [x,-5,5], [y,-5,5])$
```

(%t8)



8 Vectors and matrices

8.1 Vectors

Maxima does not actually have a specific representation for vectors, but they can be represented as either lists, enclosed in brackets, or as 1xN or Nx1 matrices. The simplest way is to use lists to represent vectors, though this is less rigorous and may lead to difficulties The preferred alternative, representing vectors as matrices, is discussed in the next subsection.
Here we define two lists of three symbolic elements, which we will treat as vectors,

```
--> A:[A1,A2,A3];
```

(%o263) $[A1, A2, A3]$

```
--> B:[B1,B2,B3];
```

(%o264) $[B1, B2, B3]$

Addition of two vectors represents the addition of the corresponding elements to generate a new vector of the same dimensionality.

```
--> A+B;
```

(%o115) $[B1 + A1, B2 + A2, B3 + A3]$

Vector subtraction is similarly defined

```
--> A-B;
```

(%o116) $[A1 - B1, A2 - B2, A3 - B3]$

Multiplying a vector by a number (scalar multiplication)

```
--> 3.1*A;
```

(%o117) $[3.1 \cdot A1, 3.1 \cdot A2, 3.1 \cdot A3]$

The dot product (or inner product) of two vectors is written using a period (.) between the arguments and results in a scalar

```
--> A.B;
```

(%o118) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

Additional operations on vectors are described below, with the vectors defined as matrices.

The elements of lists, including those representing vectors, and be accessed individually by index. In Maxima, unlike some other programming languages, the indices begin with 1

```
--> A[1];
```

(%o266) $A1$

```
--> A[2];
```

(%o267) $A2$

```
--> A[3];
```

(%o268) $A3$

The elements of a list can be changed individually

```
--> A[3]:0;
```

(%o272) 0

```
--> A;
```

(%o273) $[A1, A2, 0]$

8.2 Matrices

Matrices are generated with the matrix command, for which the argument is a list of lists, separated by commas each representing one row of the matrix.

```
--> C:matrix([C11,C12,C13],[C21,C22,C23],[C31,C32,C33]);
```

(%o270)
$$\begin{pmatrix} C11 & C12 & C13 \\ C21 & C22 & C23 \\ C31 & C32 & C33 \end{pmatrix}$$

```
--> D:matrix([D11,D12,D13],[D21,D22,D23],[D31,D32,D33]);
```

(%o271)
$$\begin{pmatrix} D11 & D12 & D13 \\ D21 & D22 & D23 \\ D31 & D32 & D33 \end{pmatrix}$$

The elements of matrices are accessed by pairs of indices, with the first index specifying the row and the second the column

```
--> C[1,1];
```

(%o274) $C11$

```
--> C[1,2];
```

(%o275) $C12$

```
--> C[2,1];
```

(%o276) $C21$

The dimensions of matrices are specified in the form nxm, where n is the number of rows and m is the number of columns. Both A and B are 3x3 matrices. E is defined below as a 2x3 matrix

```
--> E:matrix([E11,E12],[E21,E22],[E31,E32]);
```

(%o121)
$$\begin{pmatrix} E11 & E12 \\ E21 & E22 \\ E31 & E32 \end{pmatrix}$$

Matrix addition and subtraction are defined only when the two matrices have both the same number of rows and the same number of columns.

```
--> C+D;
```

(%o122)
$$\begin{pmatrix} D11+C11 & D12+C12 & D13+C13 \\ D21+C21 & D22+C22 & D23+C23 \\ D31+C31 & D32+C32 & D33+C33 \end{pmatrix}$$

```
--> C-D;
```

(%o123)
$$\begin{pmatrix} C11-D11 & C12-D12 & C13-D13 \\ C21-D21 & C22-D22 & C23-D23 \\ C31-D31 & C32-D32 & C33-D33 \end{pmatrix}$$

```
--> C+E;
```

fullmap: arguments must have same formal structure. -- an error. To debug this try: debugmode(true);

Multiplication by a scalar

```
--> 2*C;
```

(%o125)
$$\begin{pmatrix} 2\cdot C11 & 2\cdot C12 & 2\cdot C13 \\ 2\cdot C21 & 2\cdot C22 & 2\cdot C23 \\ 2\cdot C31 & 2\cdot C32 & 2\cdot C33 \end{pmatrix}$$

Multiplication of two matrices is represented by a period, as for the dot product of vectors.

```
--> C.D;
```

(%o126)
$$\begin{pmatrix} C13\cdot D31+C12\cdot D21+C11\cdot D11 & C13\cdot D32+C12\cdot D22+C11\cdot D12 & C13\cdot D33+C12\cdot D23+C11\cdot D13 \\ C23\cdot D31+C22\cdot D21+C21\cdot D11 & C23\cdot D32+C22\cdot D22+C21\cdot D12 & C23\cdot D33+C22\cdot D23+C21\cdot D13 \\ C33\cdot D31+C32\cdot D21+C31\cdot D11 & C33\cdot D32+C32\cdot D22+C31\cdot D12 & C33\cdot D33+C32\cdot D23+C31\cdot D13 \end{pmatrix}$$

Matrix multiplication is not commutative

```
--> D.C;
```

(%o127)
$$\begin{pmatrix} C31 \cdot D13 + C21 \cdot D12 + C11 \cdot D11 & C32 \cdot D13 + C22 \cdot D12 + C12 \cdot D11 & C33 \cdot D13 + C23 \cdot D12 + C13 \cdot D11 \\ C31 \cdot D23 + C21 \cdot D22 + C11 \cdot D21 & C32 \cdot D23 + C22 \cdot D22 + C12 \cdot D21 & C33 \cdot D23 + C23 \cdot D22 + C13 \cdot D21 \\ C31 \cdot D33 + C21 \cdot D32 + C11 \cdot D31 & C32 \cdot D33 + C22 \cdot D32 + C12 \cdot D31 & C33 \cdot D33 + C23 \cdot D32 + C13 \cdot D31 \end{pmatrix}$$

When two matrices are multiplied, the number of columns in the first matrix must be equal to the number of rows in the second.

```
--> E;
```

(%o128)
$$\begin{pmatrix} E11 & E12 \\ E21 & E22 \\ E31 & E32 \end{pmatrix}$$

The product D.E is defined:

```
--> D.E;
```

(%o129)
$$\begin{pmatrix} D13 \cdot E31 + D12 \cdot E21 + D11 \cdot E11 & D13 \cdot E32 + D12 \cdot E22 + D11 \cdot E12 \\ D23 \cdot E31 + D22 \cdot E21 + D21 \cdot E11 & D23 \cdot E32 + D22 \cdot E22 + D21 \cdot E12 \\ D33 \cdot E31 + D32 \cdot E21 + D31 \cdot E11 & D33 \cdot E32 + D32 \cdot E22 + D31 \cdot E12 \end{pmatrix}$$

But, E.D is not

```
--> E.D;
```

MULTIPLYMATRICES: attempt to multiply nonconformable matrices. -- an error. To debug this try: debugmode(true);

Matrix transposition is implemented with the transposition function.

```
--> D;
```

(%o131)
$$\begin{pmatrix} D11 & D12 & D13 \\ D21 & D22 & D23 \\ D31 & D32 & D33 \end{pmatrix}$$

```
--> transpose(D);
```

(%o132)
$$\begin{pmatrix} D11 & D21 & D31 \\ D12 & D22 & D32 \\ D13 & D23 & D33 \end{pmatrix}$$

```
--> E;
```

(%o133)
$$\begin{pmatrix} E11 & E12 \\ E21 & E22 \\ E31 & E32 \end{pmatrix}$$

```
--> transpose(E);
```

(%o134)
$$\begin{pmatrix} E11 & E21 & E31 \\ E12 & E22 & E32 \end{pmatrix}$$

Vectors can be thought of as special cases of matrices, with either one row and n columns or n rows and 1 column.
Here, we redefine A and B as 1x3 matrices.

```
--> A:matrix([A1, A2, A3]);
```

(%o277)
$$\begin{pmatrix} A1 & A2 & A3 \end{pmatrix}$$

```
--> B:matrix([B1,B2,B3]);
```

(%o278) $\begin{pmatrix} B1 & B2 & B3 \end{pmatrix}$

```
--> B.A;
```

(%o137) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

```
--> c*A;
```

(%o138) $\begin{pmatrix} c \cdot A1 & c \cdot A2 & c \cdot A3 \end{pmatrix}$

```
--> A.B;
```

(%o139) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

```
--> B.A;
```

(%o140) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

The elements of matrices with only one row or one column must still be accessed with two indices

```
--> A[1,1];
```

(%o279) $A1$

```
--> B[1,3];
```

(%o280) $B3$

When multiplying vectors and matrices, Maxima takes a rather relaxed attitude, whether the vector is represented as a list or a 1xn matrix. For instance, we can multiply the previously defined vector A and the 3x3 matrix C in either order

```
--> A.C;
```

(%o141) $\begin{pmatrix} A3 \cdot C31 + A2 \cdot C21 + A1 \cdot C11 & A3 \cdot C32 + A2 \cdot C22 + A1 \cdot C12 & A3 \cdot C33 + A2 \cdot C23 + A1 \cdot C13 \end{pmatrix}$

```
--> C.A;
```

(%o142)
$$\begin{pmatrix} A3 \cdot C13 + A2 \cdot C12 + A1 \cdot C11 \\ A3 \cdot C23 + A2 \cdot C22 + A1 \cdot C21 \\ A3 \cdot C33 + A2 \cdot C32 + A1 \cdot C31 \end{pmatrix}$$

In the product A.C, A is interpreted as a 1x3 matrix (sometimes called a row vector), and the product represents another row vector. If A were treated as a 3x1 matrix, the multiplication would not be defined.
But, in the product C.A, A is interpreted as a 3x1 matrix (a column vector), and the product is another column vector. Again, the multiplication is only defined with this interpretation.
Note that the elements of the two products are not the same!

We can also specifically define a 3x1 matrix, or column vector

```
--> F:matrix([F1],[F2],[F3]);
```

(%o143)
$$\begin{pmatrix} F1 \\ F2 \\ F3 \end{pmatrix}$$

```
--> G:matrix([G1],[G2],[G3]);
```

(%o144)
$$\begin{pmatrix} G1 \\ G2 \\ G3 \end{pmatrix}$$

When two column vectors are multiplied, in either order, the result is the dot product

```
--> F.G;
```

(%o145) $F3 \cdot G3 + F2 \cdot G2 + F1 \cdot G1$

```
--> G.F;
```

(%o146) $F3 \cdot G3 + F2 \cdot G2 + F1 \cdot G1$

The column vector, F, can also be multiplied by a 1xn matrix in either order, but with different results.

```
--> A.F;
```

(%o147) $A3 \cdot F3 + A2 \cdot F2 + A1 \cdot F1$

```
--> F.A;
```

(%o148)
$$\begin{pmatrix} A1 \cdot F1 & A2 \cdot F1 & A3 \cdot F1 \\ A1 \cdot F2 & A2 \cdot F2 & A3 \cdot F2 \\ A1 \cdot F3 & A2 \cdot F3 & A3 \cdot F3 \end{pmatrix}$$

The behavior here may be a little bit confusing, but the important point is that in both cases A is treated as a row vector, and the order of multiplication determines the result. Formally, the symbol "." is defined in Maxima as representing "non-commutative" multiplication.
In the product A.F, multiplication of a 1x3 matrix by a 3x1 matrix results in a 1x1 matrix. By default, Maxima returns a 1x1 matrix as a scalar value, but this behavior can be modified.
In the product F.A, A is again treated as a row vector (a 3x1 matrix), and the result is a 3x3 matrix. This matrix corresponds to the outer product.
The logic behind this is not super clear, and one has to be careful!

8.3 Additional vector and matrix functions

The basic vector and matrix functions above are provided in the standard packages that are loaded when the Maxima program is loaded. Additional functions can be found in special packages that must be loaded manually.

The function for calculating the trace of a matrix is found in the package nchrpl

```
--> load("nchrpl");
```

(%o149) $/usr/local/Cellar/maxima/5.36.1/share/maxima/5.36.1/share/matrix/nchrpl.mac$

```
--> C;
```

(%o150)
$$\begin{pmatrix} C11 & C12 & C13 \\ C21 & C22 & C23 \\ C31 & C32 & C33 \end{pmatrix}$$

```
--> mattrace(C);
```

(%o151) $C33 + C22 + C11$

Other useful functions are found in the package eigen

```
--> load(eigen);
```

(%o152) $/usr/local/Cellar/maxima/5.36.1/share/maxima/5.36.1/share/matrix/eigen.mac$

One small, but convenient, function in the eigen package is columnvector, which generates a column vector from a list of numbers or symbols.

```
--> columnvector([1,2,3]);
```

(%o153)
$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

This is equivalent to:

```
--> transpose([1,2,3]);
```

(%o154) $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

Another function in the eigen package calculates the inner product of two vectors.

```
--> A:matrix([A1,A2,A3]);
```

(%o155) $\begin{pmatrix} A1 & A2 & A3 \end{pmatrix}$

```
--> B:matrix([B1,B2,B3]);
```

(%o156) $\begin{pmatrix} B1 & B2 & B3 \end{pmatrix}$

```
--> innerproduct(A,B);
```

(%o157) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

This can also be typed in a shorter form:

```
--> inprod(A,B);
```

(%o158) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

This may appear to be the same as the non-commutative multiplication operation, and it is for real valued vectors. (Unless specifically defined otherwise, Maxima assumes that all variables are real numbers).

```
--> A.B;
```

(%o159) $A3 \cdot B3 + A2 \cdot B2 + A1 \cdot B1$

But, the two functions are different when applied to imaginary or complex values.

```
--> C:matrix([c1r+%i*c1i, c2r+%i*c2i, c3r+%i*c3i]);
```

(%o160) $\begin{pmatrix} c1r + i \cdot c1i & c2r + i \cdot c2i & c3r + i \cdot c3i \end{pmatrix}$

The same results are obtained when A and C are multiplied using the "." operator in either order.

```
--> A.C;
```

(%o161) $(c3r + i \cdot c3i) \cdot A3 + (c2r + i \cdot c2i) \cdot A2 + (c1r + i \cdot c1i) \cdot A1$

```
--> C.A;
```

(%o162) $(c3r + i \cdot c3i) \cdot A3 + (c2r + i \cdot c2i) \cdot A2 + (c1r + i \cdot c1i) \cdot A1$

But, the inner product gives different results when complex numbers are involved

```
--> inprod(A,C);
```

(%o163) $(c3r + i \cdot c3i) \cdot A3 + (c2r + i \cdot c2i) \cdot A2 + (c1r + i \cdot c1i) \cdot A1$

```
--> inprod(C,A);
```

(%o164) $(c3r - i \cdot c3i) \cdot A3 + (c2r - i \cdot c2i) \cdot A2 + (c1r - i \cdot c1i) \cdot A1$

The inner product is defined so that inprod(A,B) = conjugate(A).B This definition ensures that when the inner product is formed between a complex vector and itself, the result is real.

```
--> inprod(C,C);
```

(%o165) $c3r^2 + c3i^2 + c2r^2 + c2i^2 + c1r^2 + c1i^2$

In contrast, the dot product is:

```
--> C.C;
```

(%o166) $(c3r + i \cdot c3i)^2 + (c2r + i \cdot c2i)^2 + (c1r + i \cdot c1i)^2$

```
--> expand(%);
```

(%o167) $c3r^2 + 2 \cdot i \cdot c3i \cdot c3r - c3i^2 + c2r^2 + 2 \cdot i \cdot c2i \cdot c2r - c2i^2 + c1r^2 + 2 \cdot i \cdot c1i \cdot c1r - c1i^2$

But, this means that inprod(A,B) = conjugate(A).B is not equal to inprod(B.A) = conjugate(B).A, unless A and B both contain only real values.

8.4 Eigenvectors and eigenvalues

As suggested by its name, the eigen package has functions for calculating eigenvectors and eigenvalues for a square matrix.
For a square nxn matrix, eigenvectors are nx1 matrices (column vectors) have the the property that when they are multiplied by the matrix, the result is another nx1 matrix that is the original vector multiplied by a constant, the eigenvalue.
M.X = lambda.X where M is the matrix, X is the eigen vector, and lambda is the eigenvalue.

The following examples are from the Maxima documentation.
A matrix which has just one eigenvector per eigenvalue.

```
--> M1 : matrix ([11, -1], [1, 7]);
```

(%o168) $\begin{pmatrix} 11 & -1 \\ 1 & 7 \end{pmatrix}$

```
--> eigenvectors(M1);
```

(%o169) $[[[9 - \sqrt{3}, \sqrt{3} + 9], [1, 1]], [[[1, \sqrt{3} + 2]], [[1, 2 - \sqrt{3}]]]]$

The output is rather difficult to parse, but it consists of two lists:
[[9–sqrt(3),sqrt(3)+9],[1,1]] and [[[1,sqrt(3)+2]],[[1,2–sqrt(3)]]]
The first list is, itself, a list of two lists: A list of the eigenvalues and a list of their multiplicities.
The eigenvalues are: 9–sqrt(3) and sqrt(3)+9 The multiplicities are both one, which means that each eigenvalue is associated with just one eigenvector. (More properly, the multiplicities given by Maxima are the geometric multiplicities. There is also an algebraic multiplicity, with a different meaning.)
The second list contains the eigenvectors: [1,sqrt(3)+2] and [1,2–sqrt(3)]
To confirm that these are eigenvectors, we multiply the matrix by each of them and compare the results to those obtained by multiplying the eigenvectors by the eigenvalues:

The first eigenvector

```
--> M1.[1,sqrt(3)+2];
```

(%o170) $\begin{pmatrix} 9 - \sqrt{3} \\ 7 \cdot (\sqrt{3} + 2) + 1 \end{pmatrix}$

```
--> ratsimp(%);
```

(%o171) $\begin{pmatrix} 9 - \sqrt{3} \\ 7 \cdot \sqrt{3} + 15 \end{pmatrix}$

```
--> (9-sqrt(3))*columnvector([1,sqrt(3)+2]);
```

(%o172) $\begin{pmatrix} 9 - \sqrt{3} \\ (9 - \sqrt{3}) \cdot (\sqrt{3} + 2) \end{pmatrix}$

```
--> ratsimp(%);
```

(%o173) $\begin{pmatrix} 9 - \sqrt{3} \\ 7 \cdot \sqrt{3} + 15 \end{pmatrix}$

The second eigenvector

```
--> M1.[1,2-sqrt(3)];
```

(%o174) $\begin{pmatrix} \sqrt{3}+9 \\ 7\cdot(2-\sqrt{3})+1 \end{pmatrix}$

```
--> ratsimp(%);
```

(%o175) $\begin{pmatrix} \sqrt{3}+9 \\ 15-7\cdot\sqrt{3} \end{pmatrix}$

```
--> (sqrt(3)+9)*columnvector([1,2-sqrt(3)]);
```

(%o176) $\begin{pmatrix} \sqrt{3}+9 \\ (2-\sqrt{3})\cdot(\sqrt{3}+9) \end{pmatrix}$

```
--> ratsimp(%);
```

(%o177) $\begin{pmatrix} \sqrt{3}+9 \\ 15-7\cdot\sqrt{3} \end{pmatrix}$

The eigenvalues of the matrix can also be obtained, without the eigenvectors, with the eigenvalues command.

```
--> eigenvalues(M1);
```

(%o178) $[[9-\sqrt{3},\sqrt{3}+9],[1,1]]$

Again, the first list contains the eigenvalues and the second list contains the (geometric) multiplicities.

The eigenvectors function does not generally return normalized eigenvectors, that is vectors with unit length. However, the normalized eigenvectors can be obtained with the uniteigenvectors function.

```
--> uniteigenvectors(M1);
```

(%o179) $[[[9-\sqrt{3},\sqrt{3}+9],[1,1]],[[\frac{1}{\sqrt{4\cdot\sqrt{3}+8}},\frac{\sqrt{3}+2}{\sqrt{4\cdot\sqrt{3}+8}}]],[[\frac{1}{\sqrt{8-4\cdot\sqrt{3}}},-\frac{\sqrt{3}-2}{\sqrt{8-4\cdot\sqrt{3}}}]]]$

The output has the same form as that of the eigenvectors function, and the normalized eigenvectors are: [1/sqrt(4*sqrt(3)+8),(sqrt(3)+2)/sqrt(4*sqrt(3)+8)] and [1/sqrt(8–4*sqrt(3)),-(sqrt(3)–2)/sqrt(8–4*sqrt(3))]
We can confirm that they are normalized by calculating the inner product of each of the vectors with itself

```
--> inprod([1/sqrt(4*sqrt(3)+8),(sqrt(3)+2)/sqrt(4*sqrt(3)+8)],[1/sqrt(4*sqrt(3)+8),(sqrt(3)+2)/sqrt(4*sqrt(3)+8)]);
```

(%o180) 1

```
--> inprod([1/sqrt(8-4*sqrt(3)),-(sqrt(3)-2)/sqrt(8-4*sqrt(3))],[1/sqrt(8-4*sqrt(3)),-
    (sqrt(3)-2)/sqrt(8-4*sqrt(3))]);
```

(%o181) 1

The eigen package also includes a function to normalize any vector

```
--> A;
```

(%o182) $\begin{pmatrix} A1 & A2 & A3 \end{pmatrix}$

```
--> unitvector(A);
```

(%o183) $\begin{pmatrix} \frac{A1}{\sqrt{A3^2+A2^2+A1^2}} & \frac{A2}{\sqrt{A3^2+A2^2+A1^2}} & \frac{A3}{\sqrt{A3^2+A2^2+A1^2}} \end{pmatrix}$

```
--> %.%;
```

(%o184) $\frac{A3^2}{A3^2+A2^2+A1^2}+\frac{A2^2}{A3^2+A2^2+A1^2}+\frac{A1^2}{A3^2+A2^2+A1^2}$

```
--> ratsimp(%);
```

(%o185) 1

Here is another example from the documentation, a matrix with two eigenvectors and one eigenvalue

```
--> M2:matrix ([0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 2, 0], [0, 0, 0, 2]);
```

(%o186) $$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

```
--> eigenvectors(M2);
```

(%o187) [[[0, 2], [2, 2]], [[[1, 0, 0, 0]], [[0, 0, 1, 0], [0, 0, 0, 1]]]]

The eigenvalues are reported are 0 and 2, but an eigenvalue of zero is generally considered a trivial result. The eigenvalue 2 has a geometric multiplicity of 2 The two eigenvectors associated with the eigenvalue 2 are [0,0,1,0] and [0,0,0,1]

```
--> M2.matrix([0,0,1,0]);
```

(%o188) $$\begin{pmatrix} 0 \\ 0 \\ 2 \\ 0 \end{pmatrix}$$

```
--> M2.matrix([0,0,0,1]);
```

(%o189) $$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix}$$

The trivial eigenvector

```
--> M2.matrix([1,0,0,0]);
```

(%o190) $$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
