1 Introduction to *Maxima*

Maxima is a symbolic-based mathematical software providing a number of functions for algebraic manipulation, calculus operations, matrix and linear algebra, and other mathematical calculations.

Maxima web page

The *Maxima* web page is located at:

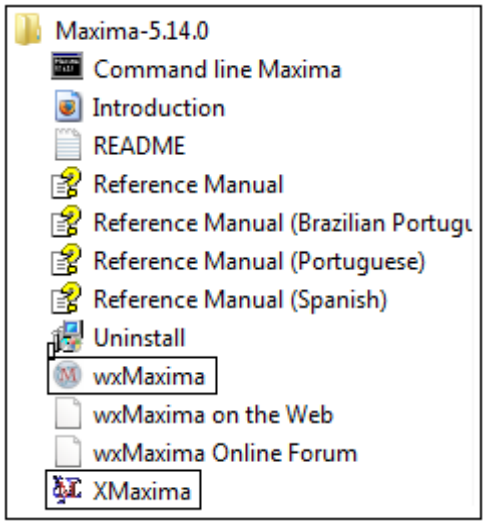
http://maxima.sourceforge.net/

Read the description of *Maxima* shown in this page. The page also includes a number of links including a *Download* link. Download and install *Maxima* in your computer as indicated in the download page.

The *Maxima* web page also includes a *Documentation* link with a number of tutorials on the use of *Maxima*.

xMaxima and *wxMaxima*

The figure below shows the listing of programs and documents available for *Maxima 5.14.0* in a *Windows Vista* installation.



You will notice that there are two possible instances of *Maxima* called *XMaxima* and *wxMaxima*. While both allow the user access to the Maxima commands, the difference is in the graphic user interface (GUI) used to communicate with *Maxima*.

XMaxima

An example of the *XMaxima* interface is shown in Figure 1.1. The top of the GUI is the input window for *Maxima* commands. The lower part is a display of a *Maxima* Primer document providing the user with some information about getting started with *Maxima*. In between the top and lower part of the display you will find buttons labeled *File*, *Back*,

Forward, *Edit*, *Options*, and *Url*: The last button refers to the file specification shown in the field immediately to its right. In this case, the file specification reads:

file:/C:/PROGRA~/MAXIMA~1.0/share/maxima/514~1.0/xmaxima/INTRO~1.HTM

The full reference to this file should be:

file:/C:/Program Files/Maxima-5.14.0/share/maxima/5.14.0/xmaxima/intro.html

The *XMaxima* GUI abbreviates some of the sub-folders in the first file specification producing the reference shown above, which could be a bit confusing. The full file specification shows the location of the file being shown in the bottom window of the *XMaxima* GUI. This *html* file is located in the *Maxima* installation as indicated above.

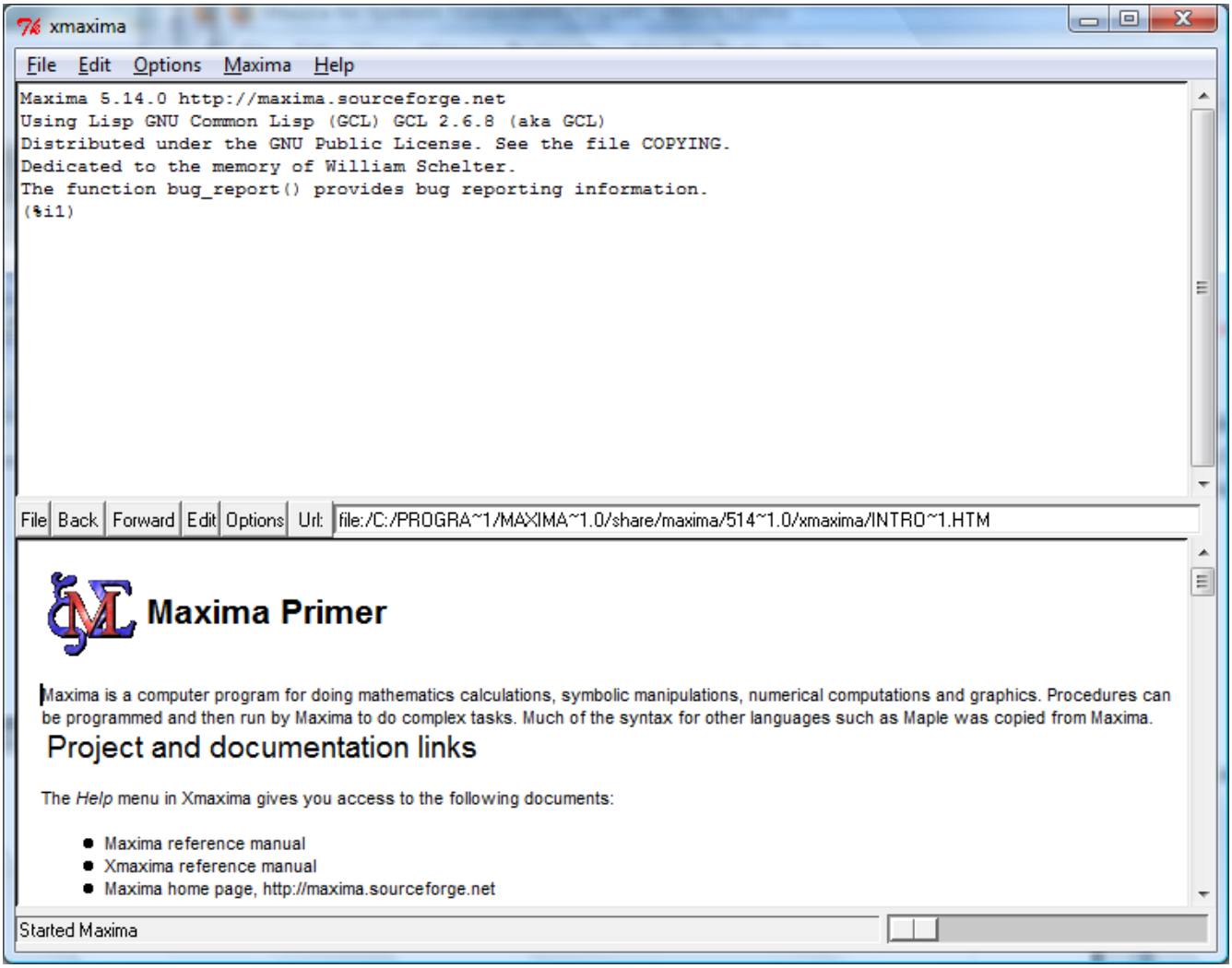
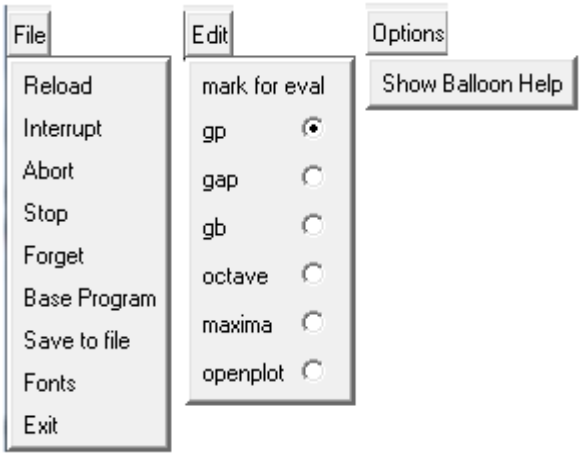


Figure 1.1. *XMaxima* starting GUI

The *Back* and *Forward* buttons allow the user to move about the document, while the other buttons provide the following menu items:



Using the *Maxima Primer*

Scroll down the *Maxima Primer* document to learn about the use of *Maxima*. One of the first applications is presented in the following paragraph (lifted from the document):

To do basic operations, a line is typed, followed by a semicolon, and then entered. This can be done in the window above. Alternately you may edit the blue portions in this buffer, and click on them, to see the result evaluated above and/or inserted in this window, depending on what was specified in the html source for this file. For example clicking below

- `integrate(1/(1+x^3),x)`

You may double click the above formula, and the integral will be substituted into the Maxima evaluation in the other window. There are examples which you may also look at 3d plotting If you wish to have your plots appear in a separate window, go to the preferences button under file, and select separate. You may also go to the netmath page to see some more capabilities.

Double-click on the *integrate* command shown in the *Maxima Primer* to see *Maxima* in action in the *XMaxima* window. The top window will now show the following operation:

```
Maxima 5.14.0 http://maxima.sourceforge.net
Using Lisp GNU Common Lisp (GCL) GCL 2.6.8 (aka GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1)
                                                2 x - 1
                                    2         atan(-------)
                                 log(x  - x + 1)    sqrt(3)     log(x + 1)
(%o1)                          - --------------- + ------------- + ----------
                                        6            sqrt(3)          3
(%i2)
```

Notice that there are two *input* locations labeled *(%i1)*, or input 1, and *(%i2)*, or input 2. Input 1 *(%i1)* is missing any input. This is so, because by double-clicking the *integrate* line

in the *Maxima Primer*, we activated the input without copying it to the top window. The result, however, is available in the top window as output 1 *(%o1)*. Also, notice that *XMaxima* presents the result of the integral as closely as possible as a two-dimensional mathematical expression, i.e.,

```
                                                 2 x - 1
                                      2       atan(-------)
                                   log(x  - x + 1)    sqrt(3)     log(x + 1)
(%o1)                            - --------------- + ------------- + ----------
                                         6             sqrt(3)          3
```

as opposite to a one-dimensional mathematical entry, i.e.,

```
-log(x^2-x+1)/6+ atan((2*x-1)/sqrt(3))/sqrt(3) + log(x+1)/3.
```

The full mathematical operation calculated in this example can be, on paper, written as

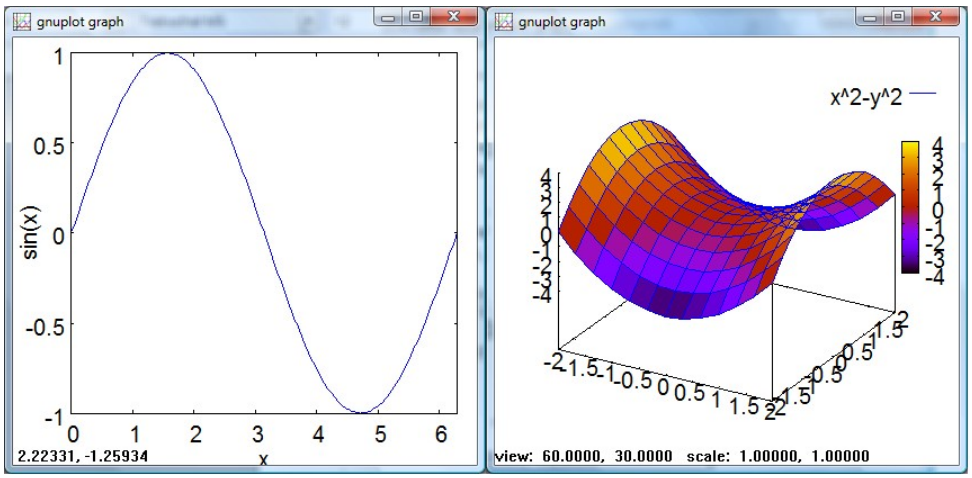
$$\int \frac{dx}{1+x^3} = -\frac{ln(x^2-x+1)}{6} + \frac{tan^{-1}\left(\frac{2*x-1}{\sqrt{3}}\right)}{\sqrt{3}} + \frac{log(x+1)}{3} \quad .$$

The user is invited to continue reading the *Maxima Primer* document and double-click on the different examples listed to learn the basic operation of *Maxima*. Following those exercises, one may notice, for example, that in the *XMaxima* interface, the mathematical constant π (the ratio of the length of a circumference to its diameter) is referred to as `%pi`. Also, infinity (∞) is referred to as *inf*.

The *Maxima Primer* examples include also plots that are produced in their own separate graphics window, e.g., the commands

- `plot2d(sin(x),[x,0,2*%pi])`
- `plot3d(x^2-y^2,[x,-2,2],[y,-2,2],[grid,12,12])`

produce, respectively, the two-dimensional and a three-dimensional graphs shown below.



Click-off the graphical windows before continuing with the other commands in the *Maxima Primer*.

wxMaxima
wxMaxima uses an interface as shown in Figure 1.2, below.

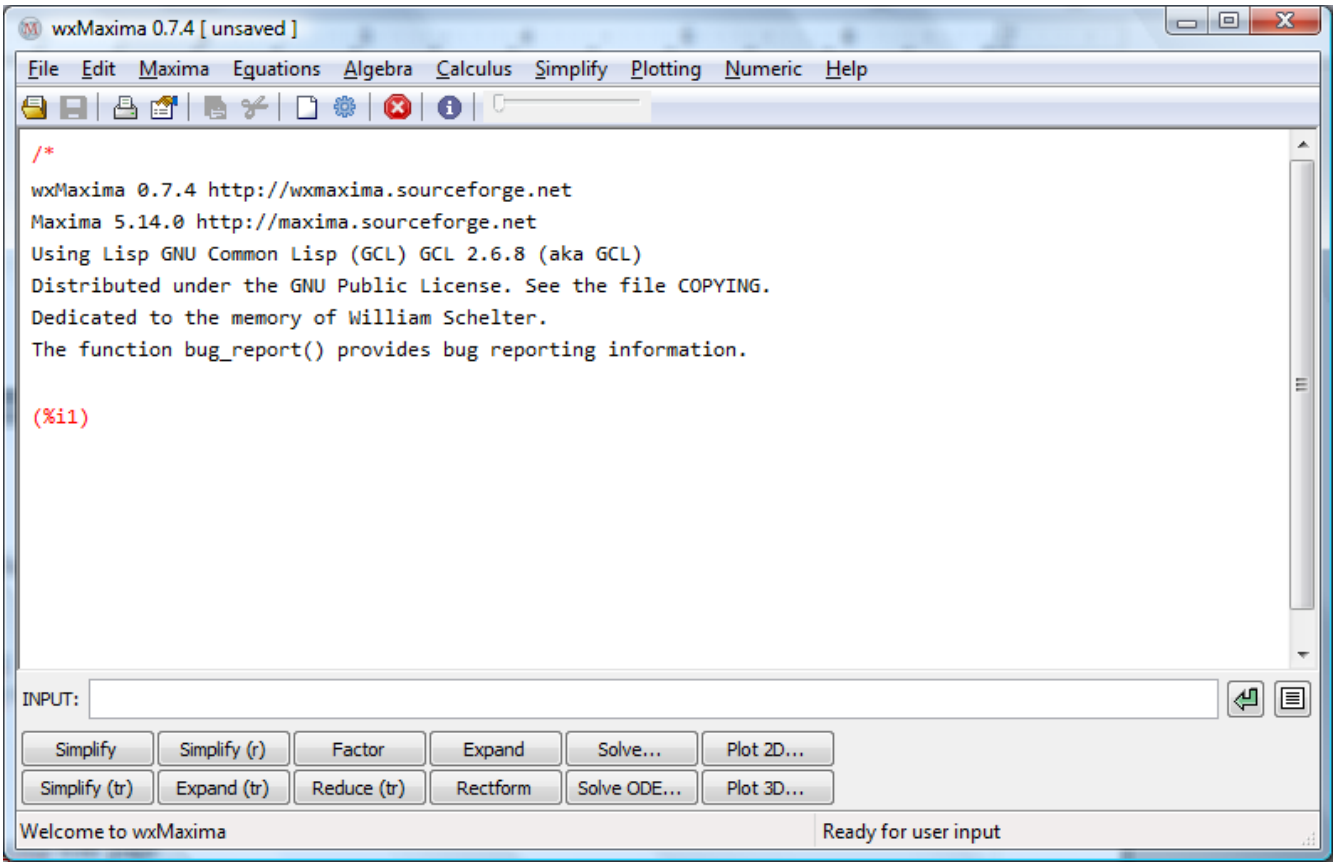


Figure 1.2. The *wxMaxima* GUI.

This interface is more sophisticated than that of *XMaxima* for the following reasons:

- *wxMaxima* produces true two-dimensional mathematical output
- *wxMaxima* provides most *Maxima* commands in menus (e.g., *Equations, Algebra, etc.*)
- Some commands can be activated by using the buttons shown at the bottom of the interface, e.g., *Simplify, Factor,* etc.
- *wxMaxima* provides dialogues to enter parameters of selected commands.
- *wxMaxima* maintains a command line history buffer where previously used commands can be accessed, repeated, or edited.
- *wxMaxima* allows mixing text with mathematical expressions to produce printable documents.
- The current version of *wxMaxima* supports simple animations (to see the current version use the menu item *Help* > *About*).

A web page for *wxMaxima* is available here:

http://wxmaxima.sourceforge.net/wiki/index.php/Main_Page

For hints on the efficient use of *wxMaxima* visit:

http://wxmaxima.sourceforge.net/wiki/index.php/Howto

NOTE: Because of the additional features available in *wxMaxima*, we will use this GUI exclusively to present the examples contained in this and subsequent chapters. We will not be using *XMaxima* anymore in this or subsequent chapters.

***wxMaxima* menus**

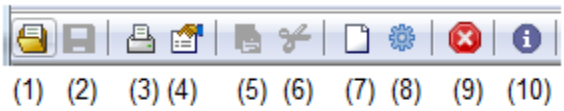
Take some time to explore the different menus in the *wxMaxima* GUI:

- The *File* menu contains items typically found in windows-based applications such as *Open, Read file, Save, Save As..., Export to HTML, Select File, Print,* and *Exit*. Some items in the *File* menu, such as *Load package*, *Batch file*, and *Monitor File*, are proper of *wxMaxima*.
- The *Edit* menu contains typical commands such as *Copy, Cut*, and *Paste*, as well as others that are proper for *wxMaxima*.
- The *Maxima* menu contains items that allow the user to control the operation of *Maxima*.
- The *Equations*, *Algebra*, *Calculus*, *Simplify*, *Plotting*, and *Numeric* menus provide mathematical functions that are entered using dialogues.
- The *Help* menu contains several items of interest such as:
  - *Maxima help*: opens the *Maxima Manual* window with description and examples of *Maxima* commands.
  - *Describe*: produces a dialogue where the user can enter the name of a specific command. Try, for example, *plot3d*, and press OK. The dialogue will access the section of the *Maxima Manual* corresponding to the requested command.
  - *Example*: enters a series of examples of applications of the requested command into the *wxMaxima* interface. Try, for example, *integrate*, and press OK.
  - *Apropos*: use this dialogue to enter a keyword to search for a command that is similar to the keyword. For example, if you were seeking information on integration, you could enter the word *integra*, to get a listing of commands that may be related to *integra*. Then, you can use *Describe* or *Example* with one of the commands listed.
  - *Show tip*: shows tips on the use of *Maxima*.
  - *Build info*: provides information on the current version of *Maxima*.
  - *Bug report*: provides a web site where users can report errors in the operation of *Maxima*, or unexpected results of some operations. These "bugs" are reported to the programming team and solutions to them (if available) get incorporated in the new versions of the software.
  - *About*: provides the current version of *wxMaxima*. Notice that the versions of *Maxima* and *vxMaxima* are not necessarily the same. My installation, at the

moment of typing this book, showed *Maxima* version 5.14.0 and *wxMaxima* version 0.7.4. Remember that *Maxima* is the computer program that performs the mathematical calculations, while *wxMaxima* is the graphics user interface (GUI).

wxMaxima tool bar

The *wxMaxima* GUI provides a tool bar with the following buttons:



(1) Open session
(2) Save session
(3) Print document
(4) Configure wxMaxima
(5) Copy selection
(6) Delete selection
(7) Insert text
(8) Insert input group
(9) Interrupt current computation
(10) Show Maxima help (same as menu item *Help* > *Maxima help*)

Using the *INPUT* line

The *INPUT* line in the *wxMaxima* interface can be used for a variety of purposes such as:

- To perform a calculation, e.g., `sqrt(1+3.5^2)/sin(%pi/6);`
- To define one or more variables, e.g., `a:2; b:2;`
- To define a function, e.g., `f(x):=sqrt(1+x^2);`
- To evaluate a function, e.g., `f(2/3);`
- To produce a plot, e.g., `plot2d(f(x),[x,-2,2]);`
- To enter other type of operations, e.g., a derivative: diff(t^2*sin(t), t);

Here are some observations from the examples shown above:

- To enter the value of a variable use a colon (:)
- To define a function use a colon followed by the equal sign (:=)
- *Maxima* expressions end with a semi-colon. If you forget to enter the semi-colon in the *INPUT* line, *wxMaxima* will enter it for you.

This is additional information useful when entering expressions:

- Variable or function names must start with a letter, and may include letters, numbers, and undersign, e.g.,

```
vx:2; x2:3; y_2:5; Initial_Velocity:-2.5;
```

- The following are reserved words in *Maxima* and cannot be used as variable names:

```
integrate            next          from              diff
in                   at            limit             sum
for                  and           elseif            then
else                 do            or                if
unless               product       while             thru
step
```

Some pre-defined functions: Some of the common pre-defined functions in *Maxima* include:

sqrt	square root	*sin*	sine	*cos*	cosine
tan	tangent	*cot*	cotangent	*sec*	secant
csc	cosecant	*asin*	inverse sine	*acos*	inverse cosine
atan	inverse tangent	*acot*	inverse cotangent	*asec*	inverse secant
acsc	inverse cosecant	*exp*	exponential	*log*	natural logarithm
sinh	hyperbolic sine	*cosh*	hyperbolic cosine	*tanh*	hyperbolic tangent
asinh	hyperbolic asin	*acosh*	hyperbolic acos	*atanh*	hyperbolic atan
floor	integer below	*ceiling*	integer above	*fix*	integer part
float	conver to floating point			*abs*	absolute value

Maxima does not have a logarithm-base-10 function. Instead, use:

$$log_{10}(x)=\frac{log(x)}{log(10)}$$

Here are some examples you can try:

```
sin(2.5*%e);float(sin(2.5*%e));
floor(%pi);ceiling(%pi);
log(5);float(log(5));
k:float(log(3)/log(10));
float(10^k);abs(-2);fix(3.3);fix(-3.2);
```

Notice that *Maxima* will tend to give symbolic results (i.e., results including fractions, square roots, unevaluated trigonometric, exponential, or logarithmic functions) rather than floating-point (or numerical) results. Use function *float*, as in the examples above, to get floating-point solutions.

Automatic parentheses. Whenever you enter an opening parenthesis in the *INPUT* line, a closing parenthesis is added automatically. If you are not used to this feature, you may end up entering more closing parentheses than needed. This situation will result in an error that is easy to spot.

The percentage (%) operator. The percentage (%) symbol represents the most recent result. Try these examples:

```
exp(-2.5)*sin(3*%pi/11);float(%);exp(-3);float(%);log(5);float(%);
```

To access the second-to-last commans use `%2`, the third-to-last, use `%3`, and so on.

Mathematical constants. Some of the common mathematical constants available in *Maxima* are:

`%e`	base of the common logarithms (=`exp(1)`)
`%i`	imaginary unit (=`sqrt(-1)`)
`inf`	real positive infinity
`minf`	real negative infinity
`infinite`	complex infinity
`% phi`	the golden ratio (ϕ)
`% pi`	ratio of length of circumference to its diameter (π)
`%gamma`	Euler's constant (γ)
`false, true`	boolean values (or logical values)

Here are some examples to try (in some examples we use function *is* to check whether comparisons of numbers are true or false):

```
float(%phi);float(%pi);float(%e);%gamma;
is(3>2);is(3<2);is(x<3);
integrate(exp(-x^2/2),x,-inf,inf);integrate(exp(-x^2/2),x,minf,inf);
```

Some examples of complex numbers. The unit imaginary number *i* is entered as `%i` in *Maxima*. Here are some examples of complex number calculations:

```
z1:3+5*%i; z2:-2+6*%i;z1+z2;z1-z2;expand(z1*z2);expand(z1^2);
```

The following functions apply to complex numbers:

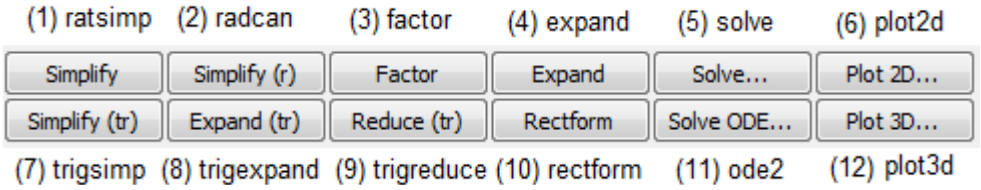
cabs	(*c*omplex *abs*olute value) calculates the modulus
carg	(*c*omplex *arg*ument) calculates the argument
rectform	generate rectangular (Cartesian) form
polarform	generate polar form
realpart	extract the real part
imagpart	extract the imaginary part
conjugate	calculates the complex conjugate

The following examples illustrate some of these functions:

```
cabs(z1);arg(z1);
z2;-z2;conjugate(z2);expand(z2*conjugate(z2));
rectform(z1/z2);rectform(sqrt(z1));polarform(z1);polarform(z2);
```

Using the button panel

The bottom of the *xwMaxima* GUI contains 12 buttons that can be used for common operations. The collection of buttons is shown in the figure below, with the *Maxima* commands associated with them.



Buttons (1) through (4), and (7) through (10) operate on an expression typed in the *INPUT* line before pressing the corresponding button. Buttons (5), (6), (11), and (12) trigger dialogues to performed the associated operations. The operation of the buttons, with appropriate examples, is shown next.

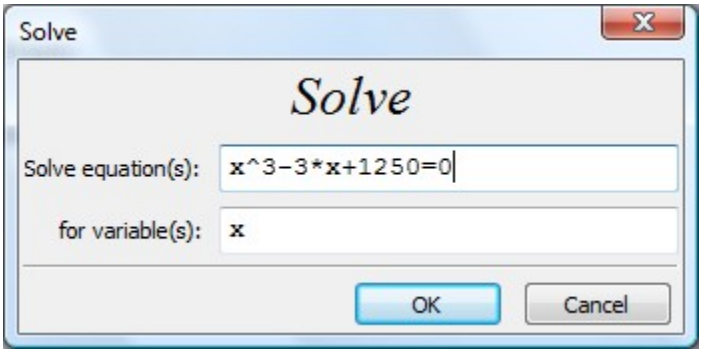
(1) Simplify: simplifies algebraic operations, e.g., `(x+2)*(x-2); [Simplify]`

(2) Simplify(r): simplifies expressions containing logs, exponentials, and radicals, e.g., `(%e^x-1)/(%e^(x/2)+1); [Simplify(r)]`

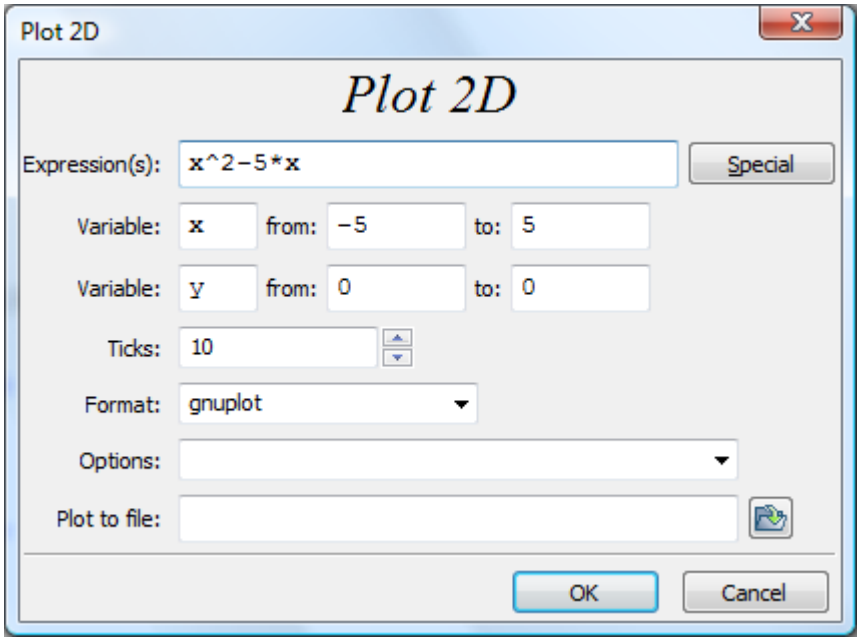
(3) Factor: factors an algebraic expression, e.g., `x^2+y^2-2*x*y; [simplify(r)]`

(4) Expand: expands an algebraic expression, e.g., `(x+1)*(x-1)*(x^2+1); [Expand]`

(5) Solve...: solves an equation, e.g.,



(6) Plot 2D...: produces an *x*-*y* (two dimensional) plot, e.g.,



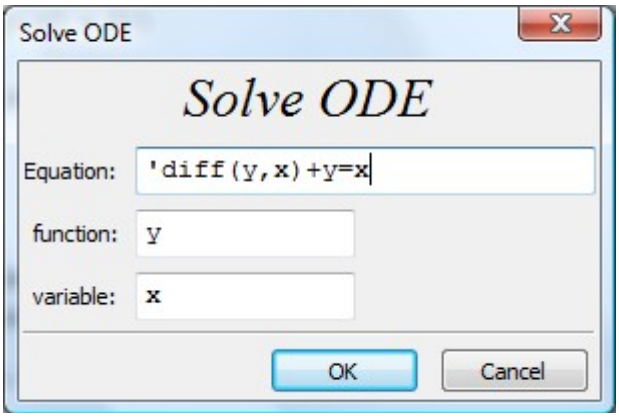
(7) Simplify(tr): trigonometric simplification in terms of *sin* and *cos*,e.g., `tan(x) [Simplify (tr)]`

(8) Expand(tr): expands a trigonometric expression, e.g., `sin(x+y) [Expand(tr)]`

(9) Reduce(tr): convert powers of trigonometric functions to those of multiples of the angle, e.g., `x+3*cos(x)^2-sin(x)^2; [Reduce(tr)]`

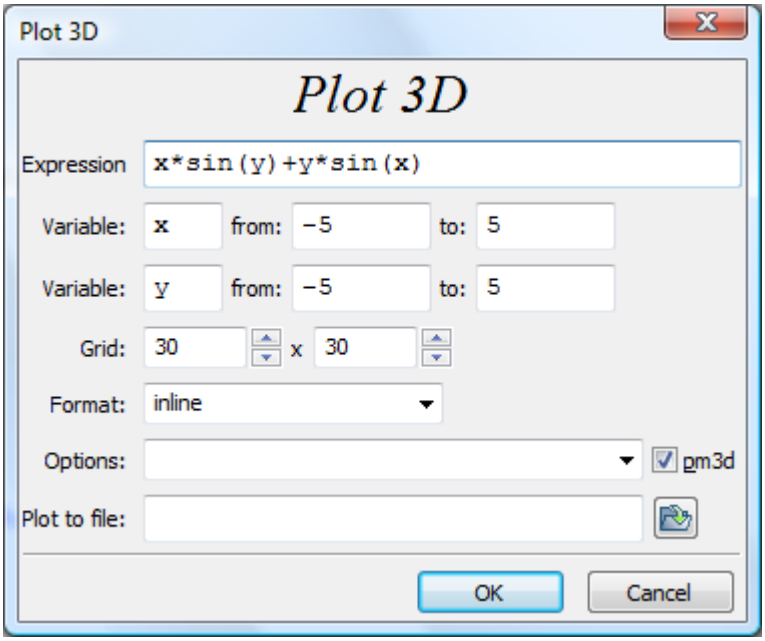
(10)Rectform: produces the rectangular form of a complex number, e.g., `1/(2+3*%i);` [Rectform]

(11)Solve ODE...: solves a 1st order or 2nd order ordinary differential equation, e.g.,



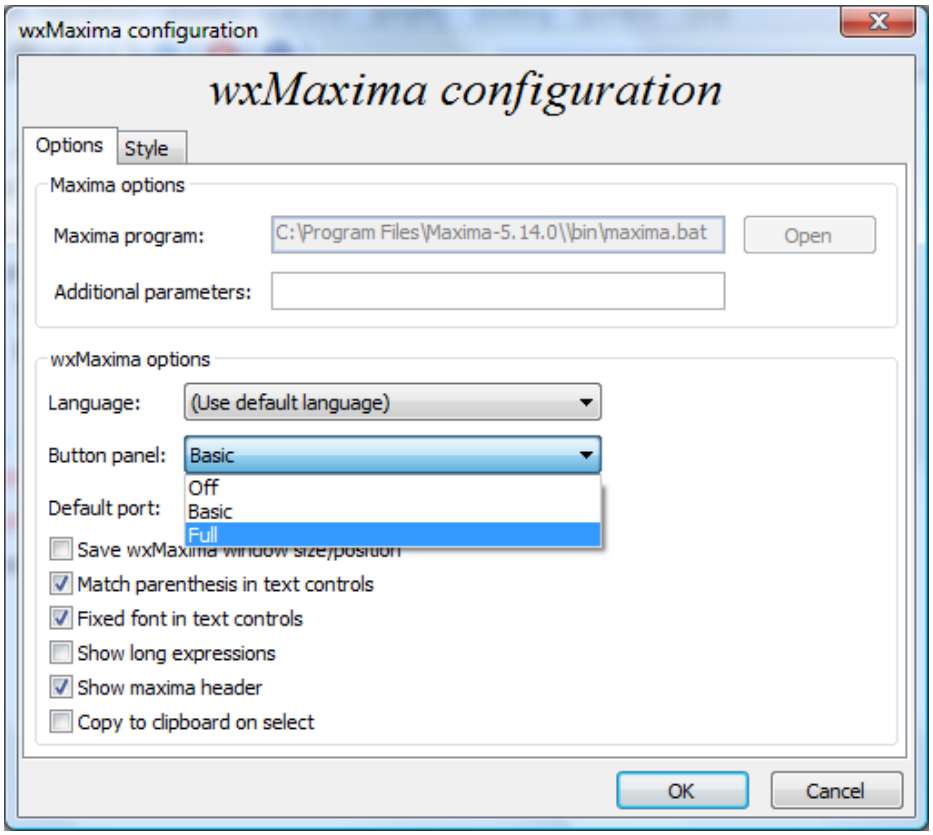
Note: Derivatives are written using `'diff(y,x,n)` to represent $\frac{d^n y}{dx^n}$.

(12) Plot3D: produces a three-dimensional plot, e.g.,

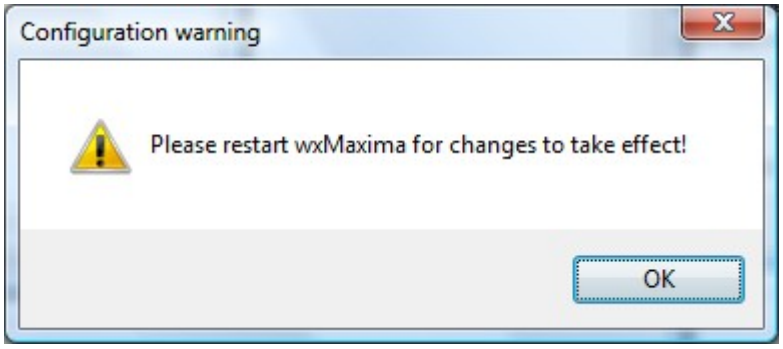


Using the full button panel

The button panel described above is referred to as the *Basic* button panel. It is possible to activate a *Full* button panel by using the menu option *Edit* > *Configure*. This activates a *wxMaxima configuration* window as shown next:

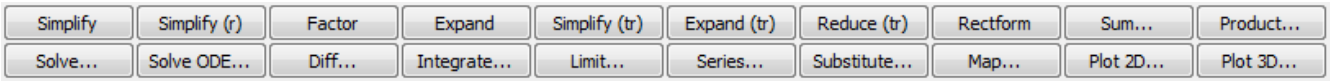


Select the option *Full* in the *Button panel* drop-down menu to activate the *Full* button panel, and press [OK]. *wxMaxima* will respond with the following message:

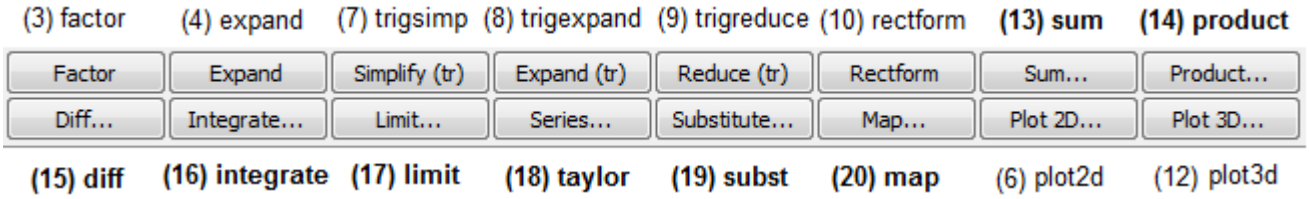


Press [OK] in this message form, and then [OK] again in the *wxMaxima configuration* window, and click off *wxMaxima*. The *Full* button panel will not be active until you re-start *wxMaxima*.

When you re-start *wxMaxima*, the bottom part of the interface will show the *Full* button panel:

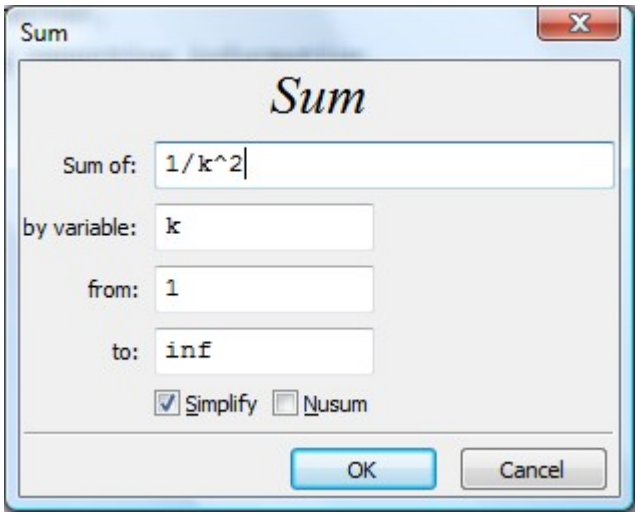


which now includes 20 buttons, instead of the 12 buttons of the *Basic* button panel. The new buttons are shown in the following figure, labeled (13) to (20), with labels shown in boldface letters (no all the buttons are shown):

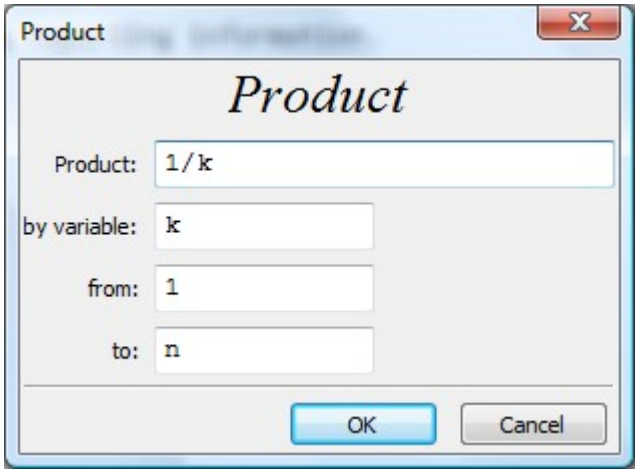


The operation of buttons (13) through (20) is described below:

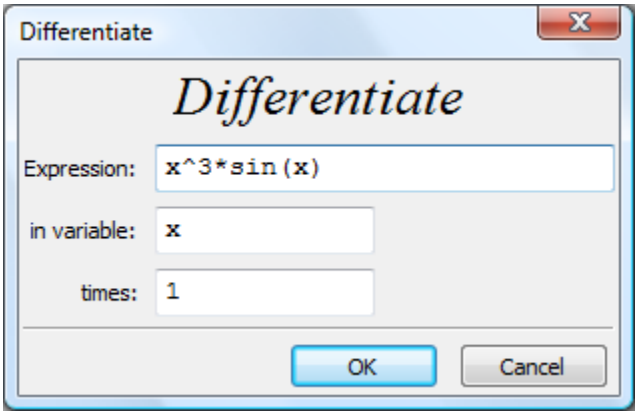
(13) *sum*: allows setting up and calculating a summation, e.g.,



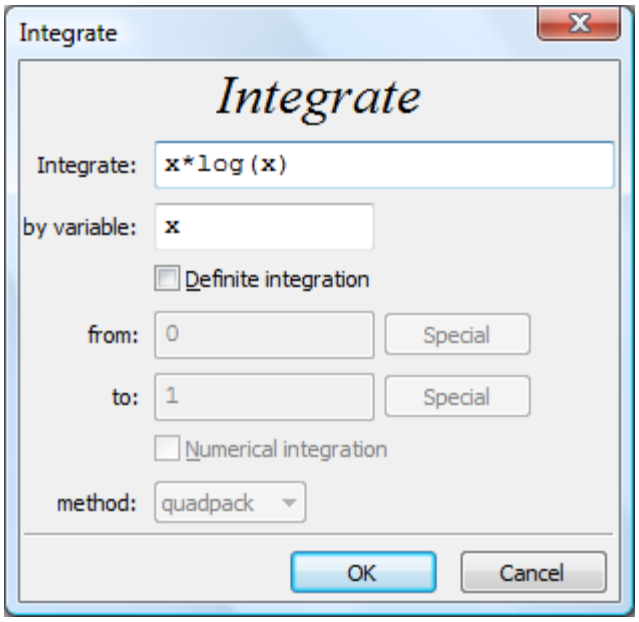
(14) *product*: allows setting up and calculating a product, e.g.,



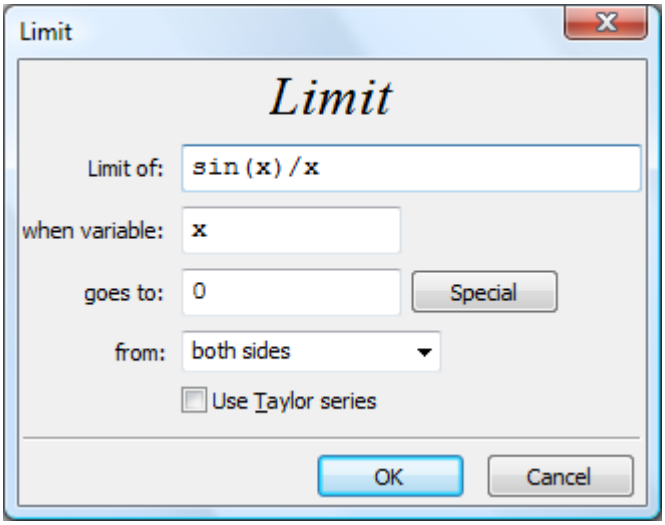
(15) *diff*: calculates a derivative, e.g.,



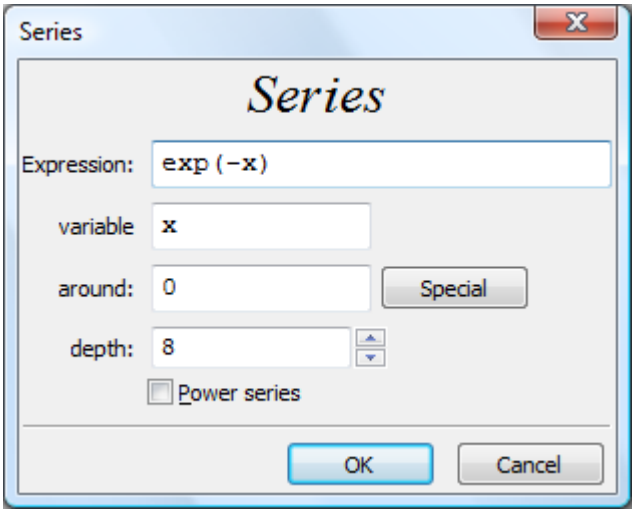
(16) *integrate*: calculates an integral



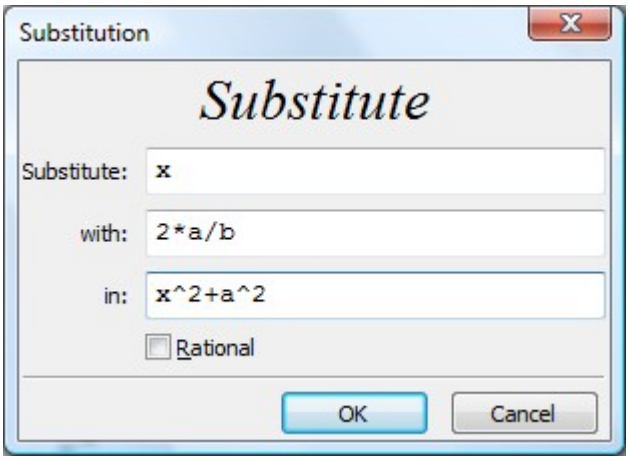
(17) *limit*: calculates a limit



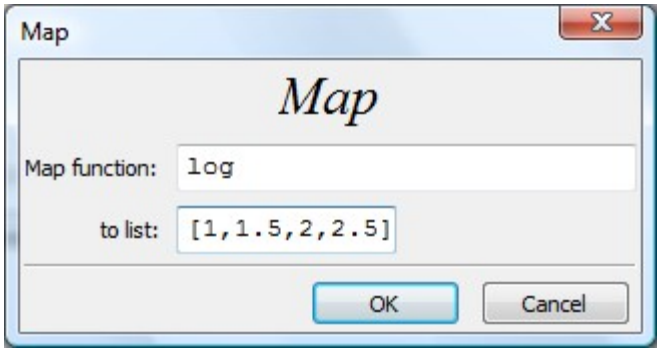
(18) *taylor*: calculates a Taylor series for an expression:



(19) *subst*: substitute an expression into a variable name



(20) *map*: maps a function to a list



Using Greek letters
In order to write Greek letters in *Maxima* you need to have the font *SPIonic* installed in your computer. You can download this font from:

http://www.drshirley.org/fonts/SPIonic.ttf

After installing the font in your computer, you need to select it to show Greek characters in your *wxMaxima* interface. Proceed as follows:

- Select the menu option *Edit>Configure*
- Click on the *Style* tab
- Check-off the *Use greek font* entry, and select *SPIonic*
- Press OK

To enter Greek letters type the English name of the letter in an expression, or precede the name with the percentage symbol (%), e.g.,

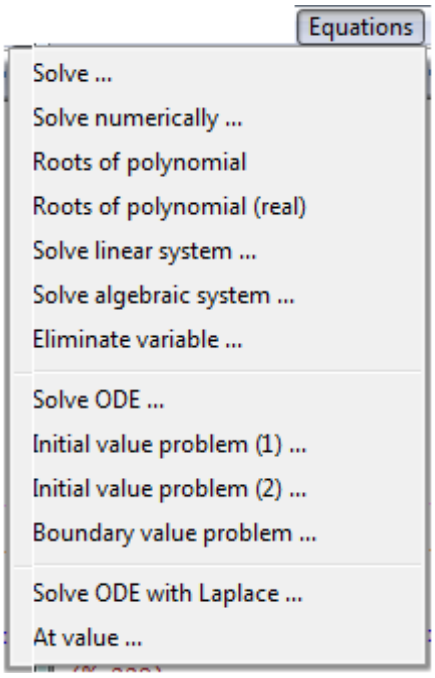
```
factor(beta^2-1);
rectform(1/(%alpha+%beta*%i);
expand((alpa-1)*(beta+gamma));
expand((%alpha-1)*(%beta+%gamma));
```

Notice the difference between typing *gamma* and *%gamma* in the last two examples. Typing *gamma* (without %) produces the upper-case Greek letter Γ which represents the Gamma function from mathematics, whereas, %gamma produces Euler constant γ, defined, as the limit as n , of the quantity

$$\sum_{k=1}^{n} \frac{1}{k} - ln(n) \quad .$$

To illustrate the use of the Gamma function try the following exercises in *wxMaxima*:

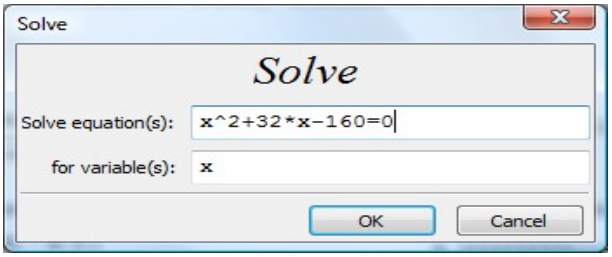
```
gamma(2.5);
plot2d(gamma(x),[x,0.5,3.0]);
```



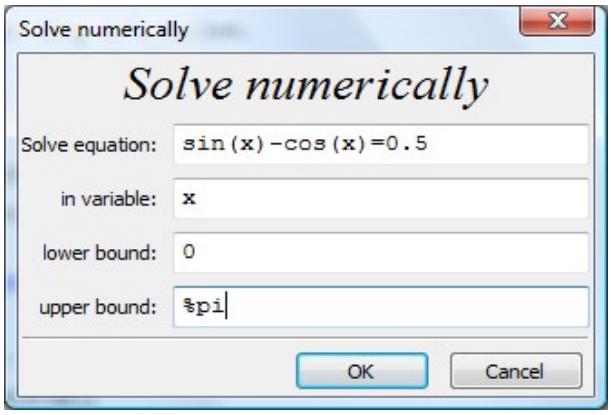
Examples from the *Equations* menu
In this section we present some examples of applications from the *Equations* menu. We use it to illustrate the use of menus such as *Equations*, *Algebra,* Calculus, etc. A listing of the available applications in the *Equations* menu is shown below:

Try the following examples by selecting entries from this menu:

- *Solve* ... same as: `solve([x^2+32*x-160=0], [x]);`



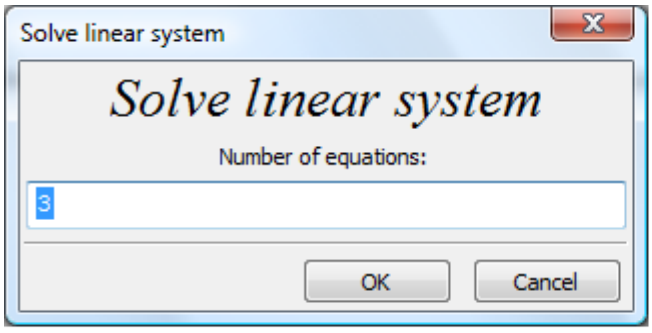
- *Solve numerically ...* equivalent to `find_root(sin(x)-cos(x)=0.5, x, 0, %pi);`

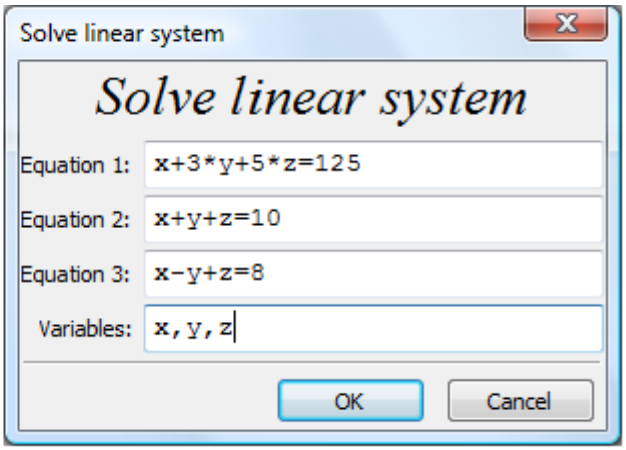


- *Roots of polynomial* -- Try: `x^3+25*x^2-5*x+212=0;` *Equations > Roots of polynomial*. Equivalent to `allroots(%);`

- *Roots of polynomial (real)* - Try: `x^3+25*x^2-5*x+212=0;` *Equations > Roots of polynomial (real)*. Equivalent to `realroots(%);`

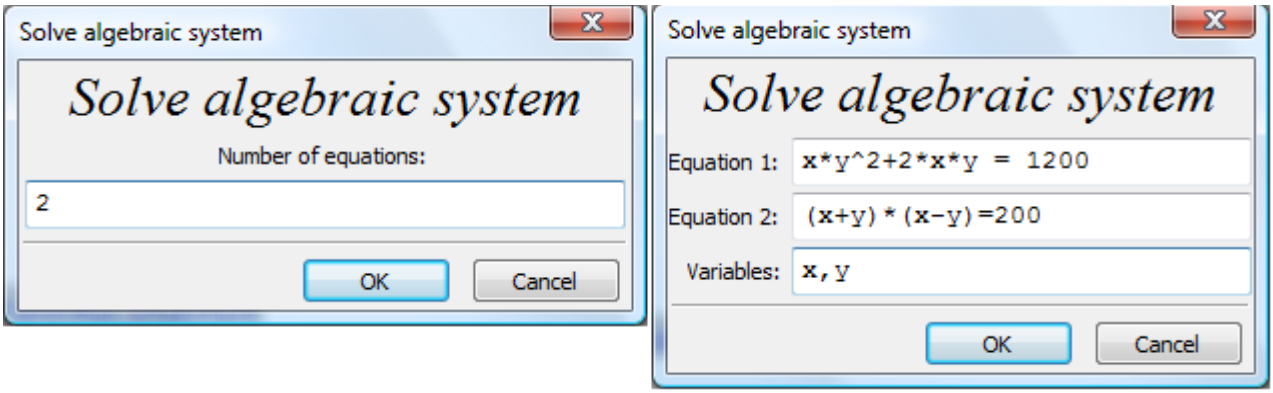
- *Solve linear system ...* equivalent to
  `linsolve([x+3*y+5*z=125, x+y+z=10, x-y+z=8], [x,y,z]);`



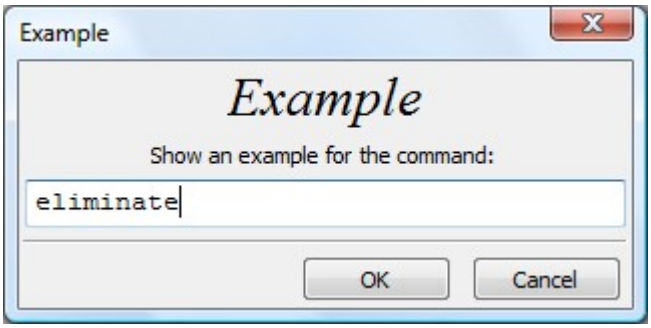


- *Solve algebraic system* ... equivalent to

```
algsys([x*y^2+2*x*y = 1200, (x+y)*(x-y)=200], [x,y]);
```

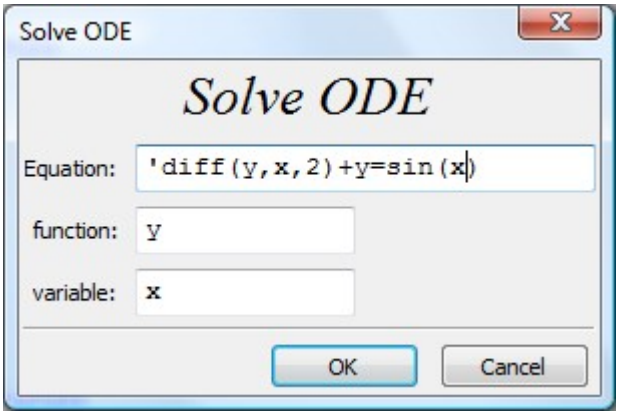


- *Eliminate variable* ... See the example available in the *Maxima Manual* by selecting the menu option *Help* > *Example...*, and type *eliminate*:

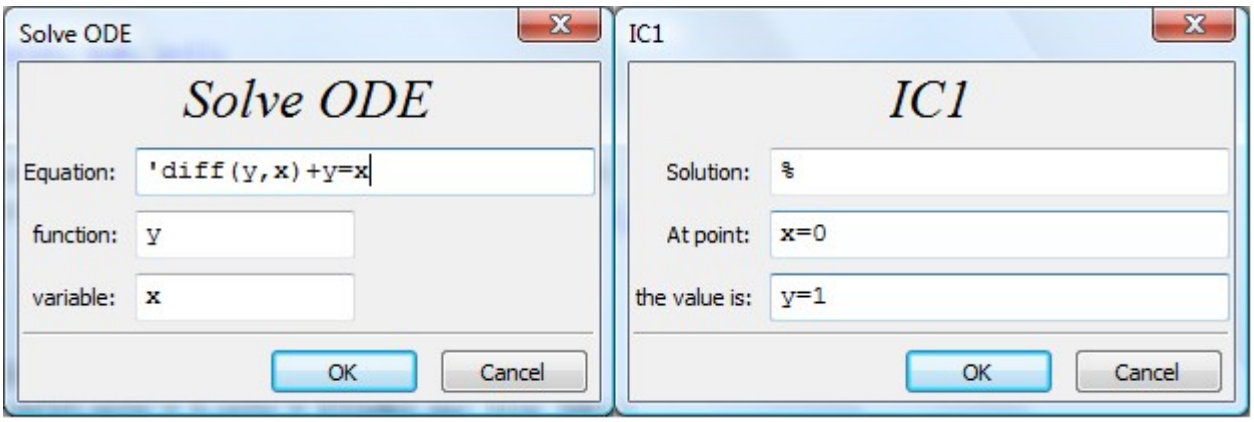


- *Solve ODE*... This is the same as pressing the button [Solve ODE...]. Equivalent to:

```
ode2('diff(y,x,2)+y=sin(x), y, x);
```

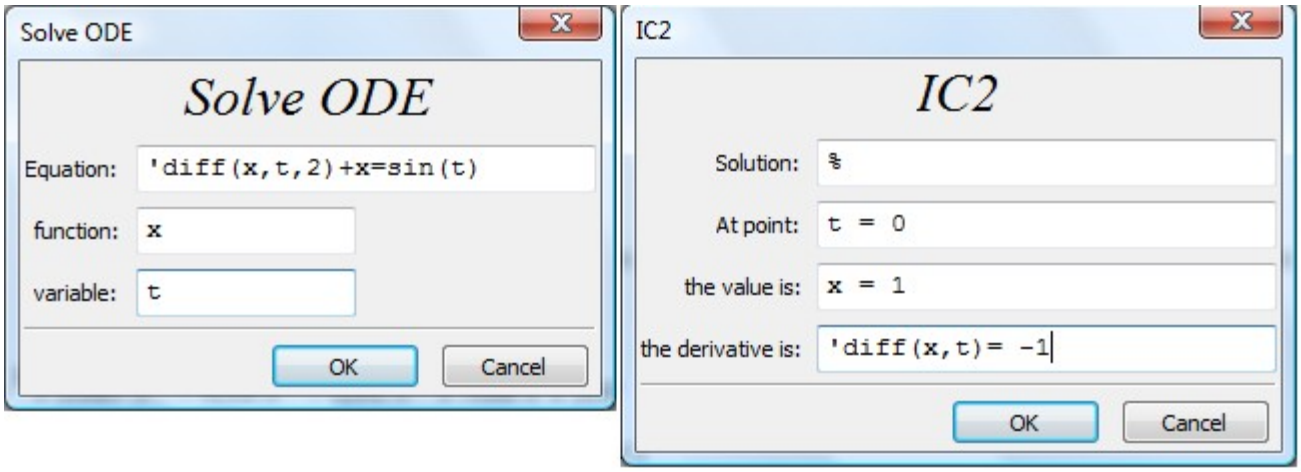


- *Initial value problem (1)* ... Initial value problem for first-order ODE. Uses two steps, first *Solve ODE* ..., then *Initial value problem (1)*. Equivalent to:



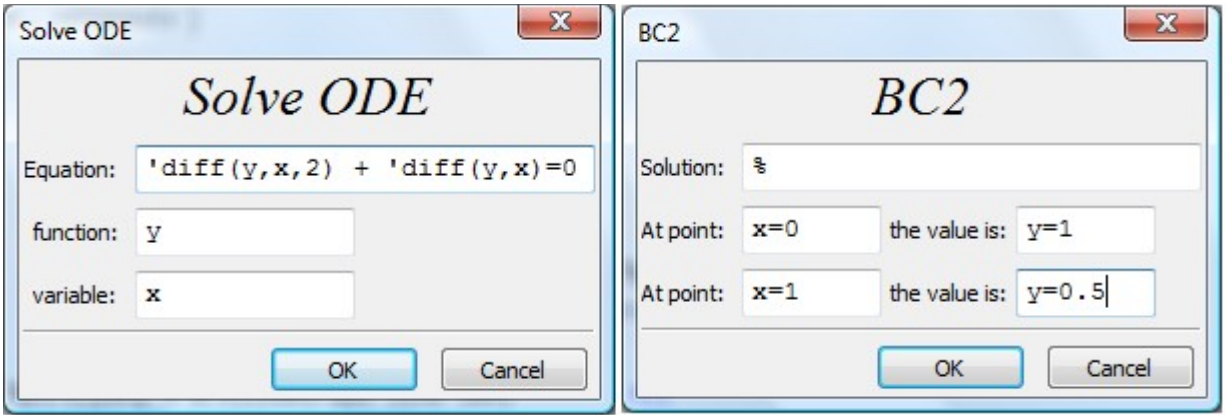
- *Initial value problem (2)* ... Initial value problem for second-order ODE. Uses two steps: first *Solve ODE* ..., then *Initial value problem (2)*. Equivalent to:

```
ode2('diff(y,x)+y=x,y,x); ic1(%,x=0,y=1);
```

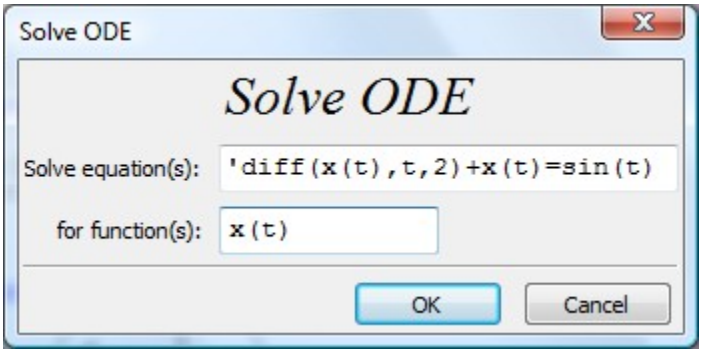


- *Boundary value problem* ... Boundary value problem for second-order ODE. Uses two steps: first *Solve ODE* ..., then *Boundary value problem*. Equivalent to:

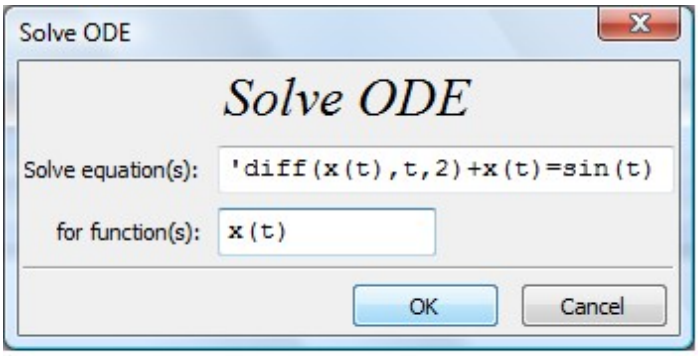
```
ode2('diff(x,t,2)+x=sin(t),x,t); ic2(%,t=0,x=1,'diff(x,t)=-1);
```

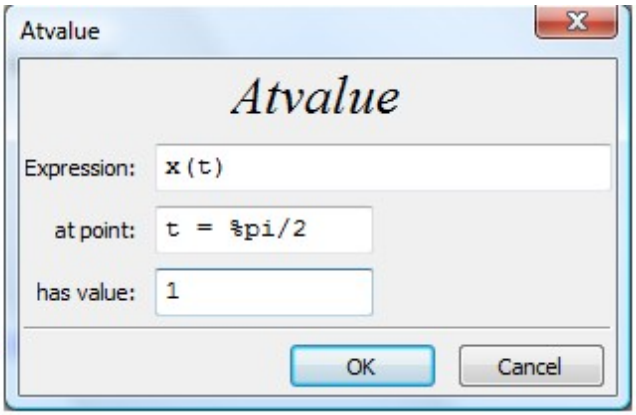


- *Solve ODE with Laplace* – Solve an ordinary differential equation using Laplace transforms. Equivalent to `desolve('diff(x(t),t,2)+x(t) = sin(t),x(t));`



- At value ... - Replace a variable in an expression. In this example the replacement takes place in the solution to an ODE.





Managing a *wxMaxima* session

In this section we illustrate the use of inputs and outputs and of the command history to perform operations on algebraic expressions.

Inputs and outputs

If you have been trying the examples shown above, your *wxMaxima* interface would show a number of inputs and outputs. Inputs are shown by the prompt `(%i...)` with an associated number, e.g.,

```
(%i84) desolve(['diff(x(t),t,2)+x(t)=sin(t)],[x(t)]);
(%i85) atvalue(x(t), t = %pi/2, 1);
```

Outputs are shown by the prompt `(%o...)` with an associated number, e.g.,

$$(\%o84)\quad x(t)=-\frac{\sin(t)}{2}-\frac{t\cos(t)}{2}+\cos(t)$$

$$(\%o85)\quad 1$$

Restarting *Maxima*

Since the inputs and outputs in your *wxMaxima* interface will be different than this example, let's restart *Maxima* by using the menu option *Edit* > *Restart Maxima*, and press [OK] at the prompt. This action will clear *Maxima*'s memory and reset the interface to that shown in Figure 1.2 (see above). At this point, only input `(%i1)` will be available. Let's try the following session. Type the commands as shown next:

```
(%i1)  (x-2)*(x-5)*(x-3)^2;
(%o1)  (x - 5)(x - 3)^2(x - 2)

(%i2)  sin(x)+cos(x+y);
(%o2)  cos(y + x)+ sin(x)

(%i3)  (x+2)/((x-2)*(x+5));
                  x + 2
(%o3)  ---------------------
       (x - 2)(x + 5)
```

Operations on input and output references

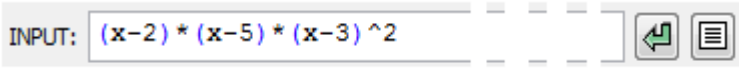
Now, we are going to use the input and output references to perform operations. Try the following commands:

```
(%i4)  expand(%i1);
(%o4)  x^4 - 13 x^3 + 61 x^2 - 123 x + 90

(%i5)  solve(%o1,x);
(%o5)  [x = 2, x = 3, x = 5]

(%i6)  trigexpand(%o2);
(%o6)  - sin(x)sin(y)+ cos(x)cos(y)+ sin(x)

(%i7)  expand(%o3^2);
                    x^2                      4 x                    4
(%o7)  ------------------------- + ------------------------- + -------------------------
       x^4 + 6 x^3 - 11 x^2 - 60 x + 100   x^4 + 6 x^3 - 11 x^2 - 60 x + 100   x^4 + 6 x^3 - 11 x^2 - 60 x + 100
```

Thus, references such as %i1, %o1, etc., act like variable names that can be operated upon as any other variable. An alternative would be to actually assign variable names to the expressions entered. To try this approach we will also illustrate the use of the command history for performing operations.

Using the command history in *wxMaxima*

Every single command that you enter in *wxMaxima* gets stored into a *command history* buffer. This buffer is accessible by clicking on the *INPUT* line and using the up and down arrow keys in your keyboard. As you press the upper arrow key once, the last command will be shown. As you keep pressing that key, the second-to-last command, third-to-last command, etc., will be shown in the *INPUT* line. For example, for the present exercise, click in the *INPUT* line, and press the upper-arrow key until you recover the very first expression entered, namely:



Then, use the left-arrow key to move the cursor to the left of the first parentheses, and type:

```
a:
```

then press [ENTER], or click on the *Enter command* button:  . The result is the following:

```
(%i8)  a:(x-2)*(x-5)*(x-3)^2;
(%o8)  (x - 5)(x - 3)^2(x - 2)
```

Now, we can refer to variable *a* for performing operations on this expression, e.g.,

```
(%i9)  expand(a);
(%o9)  x^4 - 13 x^3 + 61 x^2 - 123 x + 90

(%i10)  solve(a,x);
(%o10)  [x = 2, x = 3, x = 5]
```

The command history can be accessed, therefore, through the use of the up- and down-arrow keys in your keyboard. Once a command is accessed this way, you can either press [ENTER] to repeat it, or edit it in the INPUT line in order to perform a different operation.

End-of-line characters

It was mentioned earlier that every *Maxima* command ends in a semi-colon (;), and that if one fails to enter that end-of-line character, *wxMaxima* will enter it automatically. The fact is that, besides the semi-colon, there is also a *supress-output* character, namely, the dollar sign ($), which can be used as end-of-line character. Using the dollar sign ($) to end a *Maxima* statement suppresses the output of the command. However, the command gets executed in memory. For example, try the following commands:

```
(%i11)  r:25$ s:32$
(%i13)  r^2+s^2;
(%o13)  1649
```

In input (%i11), above, variables *r* and *s* are assigned the values *25* and *32*, respectively, but no output is shown because the statements end in a dollar sign ($), rather than in a semi-colon (;). However, output (%o13) shows that the statement r^2+s^2; was evaluated properly.

The use of the dollar sign ($) as end-of-line character saves space in the *wxMaxima* interface as illustrated in the following example:

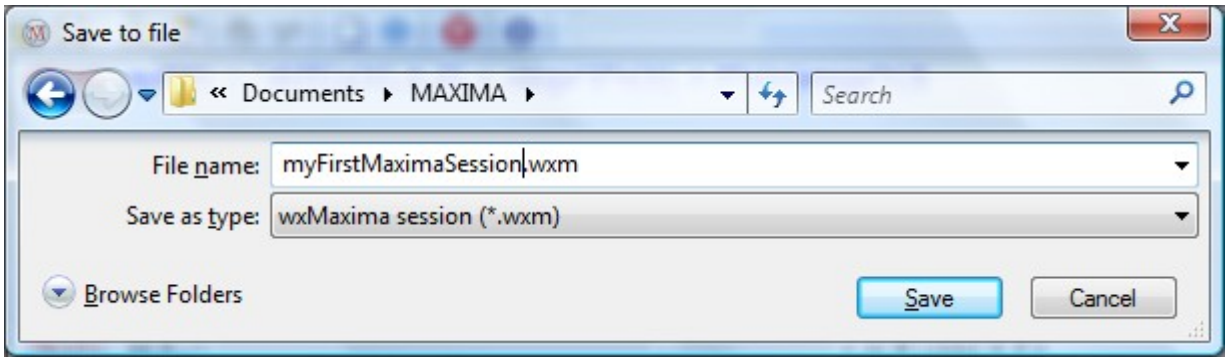
```
(%i14) myODE1 : 'diff(x(t),t,2) + omega^2*x(t) = F*sin(omega*t)$

(%i15) mySOL1 : desolve(myODE1,x(t));
Is  ω  zero or nonzero? nonzero;
```

$$(\%o15)\quad x(t)=\frac{\sin(\omega t)\left(F+2\omega\left(\frac{d}{dt}x(t)\Big|_{t=0}\right)\right)}{2\omega^2}-\frac{t\cos(\omega t)F}{2\omega}+x(0)\cos(\omega t)$$

Notice that the dollar sign ($) in input (%i14) suppresses the output for the differential equation *myODE1*. Also, notice the use of the Greek character *omega* (ω) as a coefficient in the differential equation *myODE1*. Furthermore, notice that, in attempting a solution for *myODE1*, *Maxima* doesn't know a-priori what the value of ω is. So, *Maxima* asks from the user whether ω is zero or nonzero. In this example, the user types *nonzero*, and *Maxima* returns the solution.

Saving your session

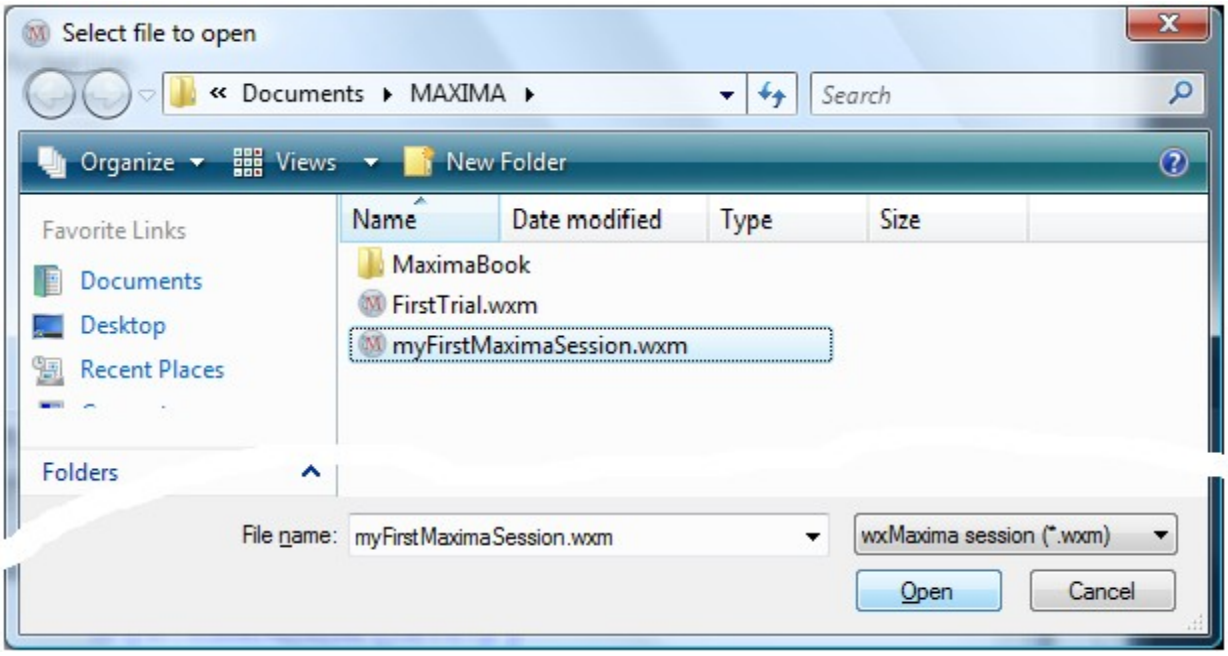
To save your session use the menu option *File* > *Save As* ... and give a name to the file into which you will save your session. The following dialog form was used in a *Windows Vista* environment to save the current session.



The file storing the session will be located in the folder `../Documents/MAXIMA/`, and will be named `myFirstMaximaSession.wxm`.

Reloading your session

Restart *Maxima* (*Edit* > *Restart Maxima*) and use the menu item *File* > *Open* to browse your computer file system. For example, in a *Windows Vista* environment, I located the file I want to load in the following dialog form:



In this case, *Maxima* opens the file and executes every command, stopping at input (%i15) where it asks again about the value of coefficient w in variable *myODE1*. Repeating the response *nonzero* allows *Maxima* to continue evaluating the file to recover the entire session saved.

Printing your session

To produce a hard-copy of your session use the menu item *File* > *Print*.

Loading a session without executing it

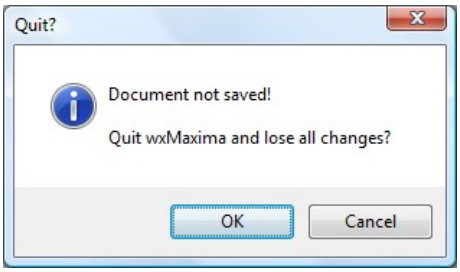
An alternative way to load a saved session is by using the menu item *File* > *Read File*. Using this option will list all the commands in the session without executing it. The commands will be available in the command history, and could be reactivated by using the up- and down-arrow keys, and pressing [ENTER] when the proper command is in the *INPUT* line.

Interrupting a calculation

If, for some reason, *wxMaxima* seems to be hung up in a calculation, you can interrupt the processing by using the menu item *Maxima* > *Interrupt*, or type *Cntl-G*. Alternatively, use the interrupt button in the menu bar:

Ending your session

To end your session use the menu item *File* > *Exit*, or click the [x] in the upper right corner of the *wxMaxima* window. This action produces the dialog form shown below.



Select [OK] if you don't want to save your current session. Otherwise, press [Cancel], save your session as indicated above, and exit *wxMaxima* once more.

Formatting your session

This section includes some examples of the use of text for commenting your session, as well as inserting sections and titles in your session.

Inserting text (comments) in *wxMaxima*

To enter text in *wxMaxima* use the menu item *Edit* > *Insert* > *Text*. The characters /* will be shown above the next input reference. Type one or more lines of text at the current cursor location. This line (or lines) of text can be used to comment your session. An example is shown next:

```
/*
This is an example of an integration:

(%i1)  integrate(x^3*exp(x),x);
(%o1)  (x^3 - 3 x^2 + 6 x - 6)%e^x
```

Text lines contained in saved session files get loaded with the rest of the commands when using *File* > *Open* or *File* > *Read file*.

Inserting a title or a section in *wxMaxima*

To insert a title use the menu item *Edit* > *Insert* > *Title*. This operation is similar to inserting text, except that the text is provided in a larger font.

To insert a section use the menu item *Edit* > *Insert* > *Section*. This operation is also similar to inserting text, except that the text is provided in a larger font and with an underline.

The following example shows a title and a section insertion in a *wxMaxima* session.

/*

This is a title

/*

This is a section

Inserting input

The menu item *Edit* > *Insert* > *Input* produces a prompt input as illustrated in the following example:

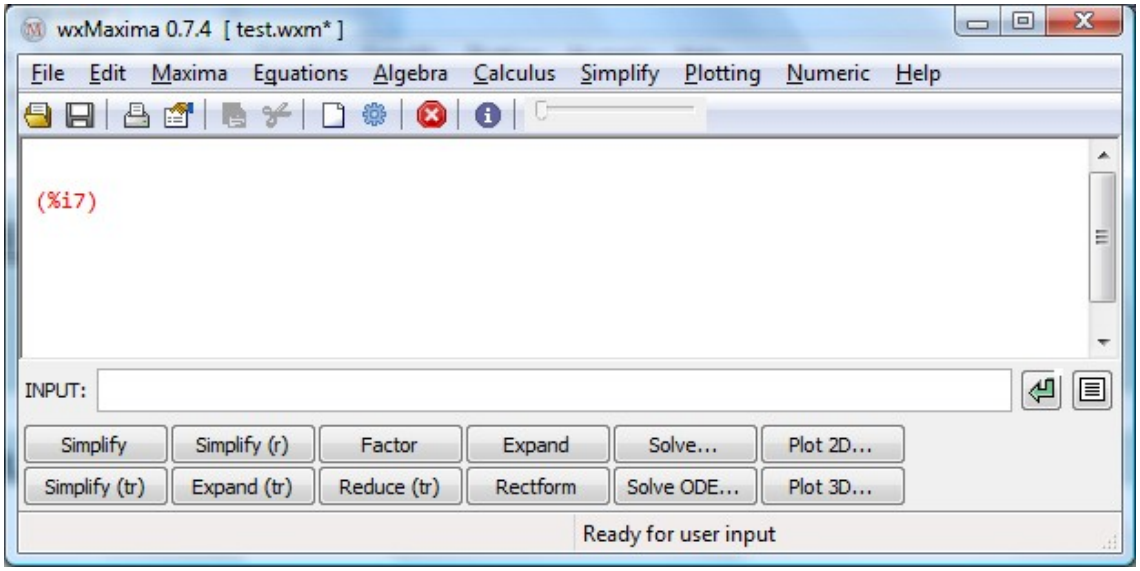
```
>> x:2$  y:x^2

(%i4)
```

If you enter a new command in the INPUT line, then the statement in front of the input prompt remains unevaluated. However, if you click on the input prompt statement, thus selecting it, and do a right-click, you can evaluate the command by selecting the option *Re-evaluate input*. In this case, the input gets evaluated as follows:

```
(%i4)  x:2$  y:x^2;
(%o5)  4
```

Clearing the screen

The option *Edit* > *Clear screen* clears the current *wxMaxima* screen, showing at the top of the screen the current input reference, e.g.,



Additional session management in *wxMaxima*

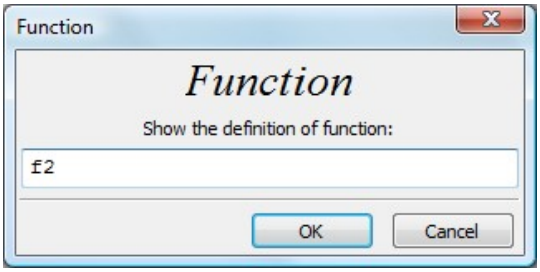
In this session we explore some of the menu items under the *Maxima* menu, namely:

- *Clear memory*: clears all variables user-defined functions - equivalent to `kill(all);`
- *Add to path*: allows user to select folders to add to the search path for *Maxima*
- *Show functions*: lists all user-defined functions in the current session (`functions;`)
- *Show definition*: provides a dialogue form to request function definitions in session
- *Show variables*: lists all variables active in the current session (`values;`)
- *Delete function*: delete selected user-defined function or functions
- *Delete variable*: delete selected variables

The following example shows the definition of variables and functions and the listing of their names:

```
(%i1)  x1 : 2$ x2 : -4$ y1 : -3$ y2 : 3.5$

(%i5)  f1(x) := x^3 + 1 $ f2(x) := 1/(1+x^2) $ g1(y) := sqrt(y+1)$
/*
Show functions:

(%i8)  functions;
(%o8)  [ f1(x), f2(x), g1(y)]
/*
Show variables

(%i9)  values;
(%o9)  [ x1 , x2 , y1 , y2 ]
```

The option *Show definition* is used next to find the definition of function *f2*:

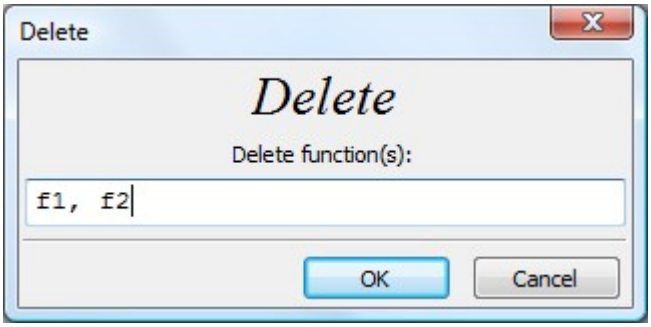


The result is shown below:

```
(%i10)  fundef(f2);
```

$$(\%o10)\quad f2(x) := \frac{1}{1+x^2}$$

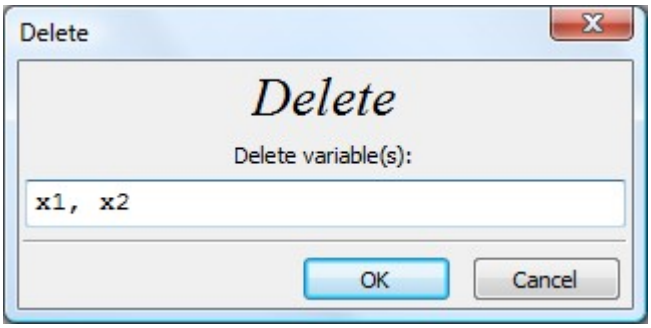
The option *Delete functions* produce the dialogue:



To delete functions *f1* and *f2* we enter those names in the dialogue. The result is shown below:

```
(%i11) remfunction(f1, f2);
(%o11) [ f1 , f2 ]
```

The option *Delete variables* produces the following dialogue:



To delete functions *x1* and *x2* we enter those names in the dialogue. The result is shown below:

```
(%i12) remvalue(x1, x2);
(%o12) [ x1 , x2 ]
```

An alternative way to delete user-defined functions or variables is to use function *kill*. This function basically clears any value or definition associated with a variable or function name. For example, to clear the contents of variable *y1*, use:

```
(%i13) kill(y1);
(%o13) done
```

Check that the value of *y1* is cleared:

```
(%i14) y1;
(%o14) y1
```

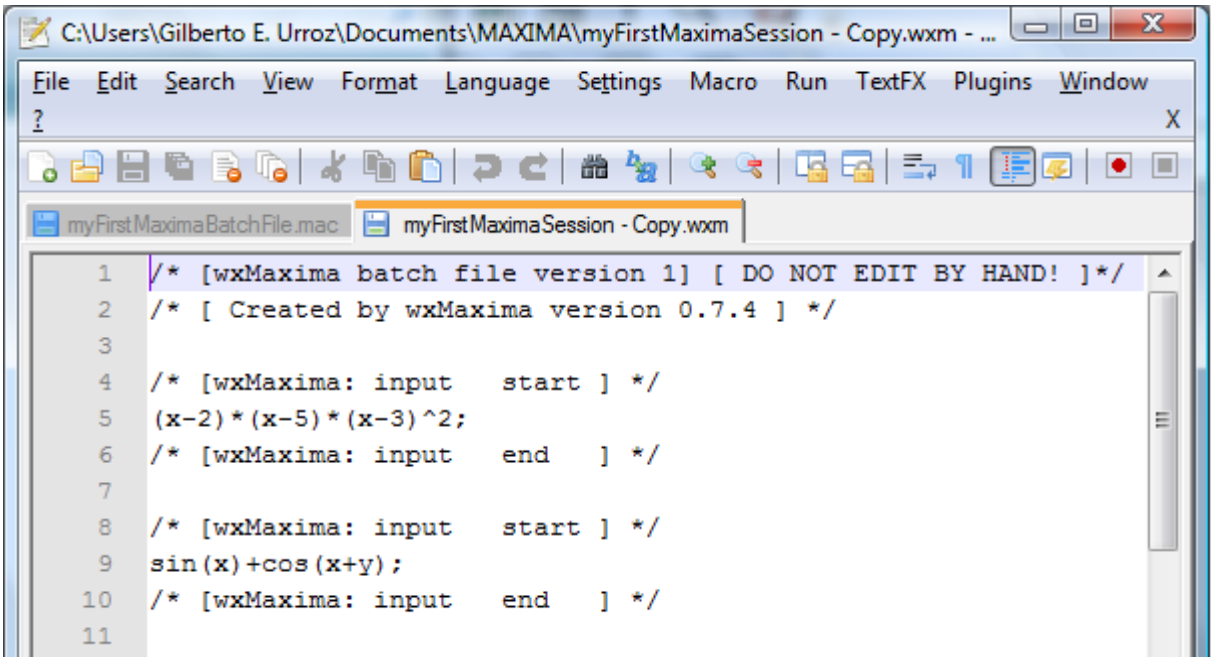
Creating a batch file

In an earlier exercise we saved a file called *myFirstMaximaSession.wxm*. In this section we will show you how to create a *Maxima* batch file out of your saved session.

In order to create a batch file we need to edit the session file using a text editor. In this example I will use the *Notepad++* text editor to open the session file. *Notepad++* is available at

http://notepad-plus.sourceforge.net/uk/site.htm .

When opened with *Notepad ++*, the file *myFirstMaximaSession.wxm* looks as follows:



Notice that you are warned in the very first line of the file to not edit the file by hand. This is for the *wmx* file. If you change anything in the file it may not be readable by *wxMaxima* again. The way to proceed is to save the file as a batch file, with the *.mac* suffix. Save it, for example, as *myFirstMaximaBatchFile.mac*, and edit it to look as shown below. This is the batch file that includes a number of comment lines (text between /* and */), and *Maxima* commands.

```
/* Batch file example */
/* Load a few expressions */
(x-2)*(x-5)*(x-3)^2;
sin(x)+cos(x+y);
(x+2)/((x-2)*(x+5));
/* Apply operations to those expression */
expand(%i1);
solve(%o1,x);
expand(%o3^2);
/* Define an expression as a variable */
a:(x-2)*(x-5)*(x-3)^2;
/* Apply operations to variable a */
expand(a);
solve(a,x);
/* Example of a differential equation solution */
myODE1 : 'diff(x(t),t,2) + omega^2*x(t) = F*sin(omega*t)$
mySOL1 : desolve(myODE1,x(t));
/* End of batch file */
```

To load a batch file use the menu item *File* > *Batch file*, and select the proper file to load. The result of the batch file operation will be shown in your *wxMaxima* window. Notice, however, that the comment lines are not shown in the *wxMaxima* window. If you want to show explanatory text from your batch file, you may want to replace the comments by a string, making sure that the string ends in a dollar sign ($) rather than in a semi-colon (;).

```
/* Batch file example */
"Load a few expressions"$
(x-2)*(x-5)*(x-3)^2 $
sin(x)+cos(x+y) $
(x+2)/((x-2)*(x+5)) $
"Apply operations to those expression"$
expand(%i1);
solve(%o1,x);
expand(%o3^2);
"Define an expression as a variable"$
a:(x-2)*(x-5)*(x-3)^2 $
"Apply operations to variable a "$
expand(a);
solve(a,x);
"Example of a differential equation solution "$
myODE1 : 'diff(x(t),t,2) + omega^2*x(t) = F*sin(omega*t)$
mySOL1 : desolve(myODE1,x(t));
/* End of batch file */
```

With theses changes, the output in the *wxMaxima* is now well documented, although the comment strings are now part of the input (with no output), rather than inserted text. Part of the output from the batch file is shown below:

```
(%i1) batch("C:/Users/Gilberto E. Urroz/Documents/MAXIMA/myFirstMaximaBatchFile.mac")$
batching #pC:/Users/Gilberto E. Urroz/Documents/MAXIMA/myFirstMaximaBatchFile.mac
(%i2) Load a few expressions
(%i3) (x - 2)(x - 5)(x - 3)^2
(%i4) cos(y + x)+ sin(x)
                2 + x
(%i5) ----------------------
      (x - 2)(5 + x)
(%i6) Apply operations to those expression
(%i7) expand(%i1)
(%o7) batch(C:/Users/Gilberto E. Urroz/Documents/MAXIMA/myFirstMaximaBatchFile.mac)
(%i8) solve(%o1, x)
(%o8) []
(%i9) expand(%o3^2)
(%o9) x^8 - 26 x^7 + 291 x^6 - 1832 x^5 + 7099 x^4 - 17346 x^3 + 26109 x^2 - 22140 x + 8100
(%i10) Define an expression as a variable
(%i11) a:(x - 2)(x - 5)(x - 3)^2
```

A batch file can also be created from scratch. Simply type the *Maxima* commands in a text file and save it with the suffix *.mac*. Here is an example of a batch file created from scratch:

```
1  /* Example of a batch file written from scratch */
2  "Examples of integrals:"$
3  "1 - Indefinite integrals:"$
4  integrate(x^2*log(x),x);
5  integrate(sin(x)^2,x);
6  integrate(exp(-x)*sin(x),x);
7  "2 - Definite integrals:"$
8  integrate(x^2*log(x),x,1,5);
9  integrate(sin(x)^2,x,-%pi/6,+%pi/6);
10 integrate(exp(-x)*sin(x),x,0,%pi/8);
11 /* End of batch file */
```

Important basic functions
This section addresses a few basic functions and operators of general application in mathematical functions and that were not addressed in any of the previous sections.

Evaluation or not evaluation of an operation
In many of the examples presented above related to differential equations we use an apostrophe (') in front of the derivative operator, *diff*, in order to avoid its evaluation. To illustrate the difference between the entry 'diff and diff, see the following example:

```
(%i1) myODE1 : 'diff(x,t,2)+'diff(x,t)+x=exp(-t);
```

(%o1) $\frac{d^2}{d\,t^2}x + \frac{d}{d\,t}x + x = \%e^{-t}$

```
(%i2) myODE1 : diff(x,t,2)+diff(x,t)+x=exp(-t);
```

(%o2) $x = \%e^{-t}$

In the first expression, using 'diff(x,t,2) produces as output the derivative thus indicated. However, in the second expression, *Maxima* evaluates the required derivatives. Since function *x(t)* has not been defined, the derivatives in the second expression evaluate to zero, and the result is $x = e^{-t}$.

The following example shows an non-evaluated integral:

```
(%i3) 'integrate(exp(-x)*sin(x),x);
```

(%o3) $\int \%e^{-x}\sin(x)\,d\,x$

An example of a summation is shown next:

```
(%i27) 'sum(1/k^2, k, 1, inf);
```

(%o27) $\sum_{k=1}^{\infty} \frac{1}{k^2}$

Applications of *ev*
Function *ev* evaluates an expression in a given environment determined by a number of arguments. For complete information on function *ev*, use the menu item *Help* > *Describe*, and enter the name *ev* in the dialogue form. In this document we will present only some specific examples of the use of function *ev*.

- Substituting constants in an equation before solving it:

```
(%i35)  ev(solve(a*x^2+b*x+c=0),a=2,b=-5,c=3);
```

$$(\%o35)\quad \left[x=\frac{3}{2},\ x=1\right]$$

- Force floating-point evaluation of rational numbers:

```
(%i41)  ev(sqrt(2),float);
(%o41)  1.414213562373095

(%i42)  ev(3^(1/3),float);
(%o42)  1.442249570307408
```

- Force derivative calculation after result has been suppressed:

```
(%i47)  'diff(x^3+x,x);
```

$$(\%o47)\quad \frac{d}{dx}\left(x^3+x\right)$$

```
(%i48)  ev(%,diff);
```

$$(\%o48)\quad 3x^2+1$$

- Derivative and integral calculations can be forced with the option *nouns*:

```
(%i55)  'integrate(x*log(x),x);ev(%,nouns);
```

$$(\%o55)\quad \int x\log(x)\,dx$$

$$(\%o56)\quad \frac{x^2\log(x)}{2}-\frac{x^2}{4}$$

```
(%i57)  'diff(x*log(x),x);ev(%,nouns);
```

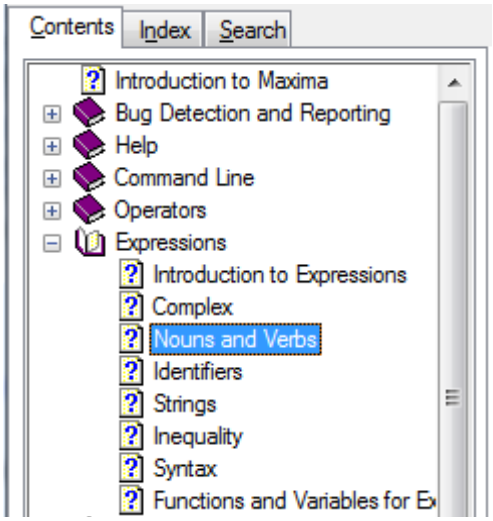
$$(\%o57)\quad \frac{d}{dx}\left(x\log(x)\right)$$

$$(\%o58)\quad \log(x)+1$$

These examples illustrates how to list an expression and their evaluation in the same line. It also introduces the idea of *nouns* in *Maxima* evaluation.

Nouns and verbs

To understand the use of the argument *nouns* in the examples above, please open the *Maxima Manual*, available through the menu item *Help* > *Maxima help*, and find section *6.3 Nouns and verbs* in the *Contents* tab, as shown in the figure below.



Read this section in the *Manual* to understand the idea of verbs and nouns, as well how to convert form one to the other.

Online help

In an earlier section we presented the different options available in the *Help* menu. A quick way to obtain help is by using the *??* operator. For example, if you are interested in finding information about the function *eval*, use:

```
(%i31)  ?? eval;
 0: Assignment operator (evaluates left-hand side)  (General operators)
 1: eval  (General operators)
 2: eval_string  (Functions and Variables for strings)
 3: infeval  (Functions and Variables for Command Line)
 4: lbfgs_nfeval_max  (Functions and Variables for lbfgs)
 5: noeval  (Functions and Variables for Simplification)
 6: timer_devalue  (Functions and Variables for Debugging)
 7: tr_warn_meval  (Functions and Variables for Function Definition)
Enter space-separated numbers, `all' or `none':
```

Maxima reply by listing a number of entries that include the particle *eval*, and requesting additional input from the user. At this point, the user can enter a particular number referring to the 7 options listed, or enter the particles *all* or *none*. Enter *none* to stop the online help process.

The following is another example related to the function *integrate*.

```
(%i33)  ?? integrate;
 0: integrate  (Functions and Variables for Integration)
 1: integrate_use_rootsof  (Functions and Variables for Integration)
Enter space-separated numbers, `all' or `none':  none;
(%o33)  true
```

2 Real numbers, functions, and units in *Maxima*

In this Chapter we introduce calculations with real and complex numbers using *Maxima*. We also introduce the use of functions, conditional statements, and logical particles. This chapter also includes examples of calculations using units of measurements.

Symbolic and floating-point results with real numbers

In this section we present simple calculations with real numbers and introduce conversion from symbolic to floating-point results. Symbolic results represent the results that one would obtain by working by hand, and producing simplification of operations with numbers. By default, *Maxima* produces symbolic results when operating with real (and complex) numbers. To produce floating-point values it is necessary to use function *float* as illustrated in the examples shown below. Notice that the first example shows a simple fraction of integer numbers, while the second example shows a square root calculation.

```
(%i1)  5/3;
(%o1)  5/3

(%i2)  float(%);
(%o2)  1.666666666666667

(%i3)  sqrt(3^2+5);
(%o3)  sqrt(14)

(%i4)  float(%);
(%o4)  3.741657386773941
```

Fractions involving floating-point numbers produce floating-point results, e.g.,

```
(%i5)  3.5/2;
(%o5)  1.75
```

Changing the default format

Use the menu item *Numeric* > *Toggle numeric output* to change the default format of calculations with real numbers. For example, activating this item once, changes the default output format to floating point as illustrated below.

```
(%i7)  if numer#false then numer:false else numer:true;
(%o7)  true
(%i8)  exp(-2);
(%o8)  0.13533528323661
```

A second application of the menu item *Numeric* > *Toggle numeric output* will return the default output format to symbolic:

```
(%i9)  if numer#false then numer:false else numer:true;
(%o9)  false
(%i10)  exp(-2);
                                  - 2
(%o10)  %e
```

In this specific example the output $\%e^{-2}$ is the symbolic representation of the number e^{-2}, while `0.1353...` is the corresponding floating-point result.

Other format changes

The *Numeric* menu includes also the menu *items Numeric* > *To float* and *Numeric* > *To bigfloat* that can be used to convert a symbolic result into either a simple floating-point format, or to a double-precision floating-point format (referred to as *bigfloat* in *Maxima*). These two menu items are equivalent to the functions *float* and *bfloat*, respectively. When applying any of these two menu items, the result refers to the last result available (i.e., to *%*). The following two examples show the application of *To float* and *To bigfloat* to the number e^{-2}.

```
(%i10)  exp(-2);
                                  - 2
(%o10)  %e
(%i11)  float(%), numer;
(%o11)  0.13533528323661
(%i12)  exp(-2);
                                  - 2
(%o12)  %e
(%i13)  bfloat(%);
(%o13)  1.353352832366127b-1
```

Notice that the last result is in *power-of-ten* format with *b* indicating the power of ten. While the output shown uses the default number of digits (16), the value is stored in a double-precision location.

Power-of-ten format

To enter power-of-ten floating-point values use *e* to indicate the power of ten, some examples are shown below:

```
(%i15)  3.5e-2;-2.5e-3;1.2e2;-3.5e4;
(%o15)  0.035
(%o16)  - 0.0025
(%o17)  120.0
(%o18)  - 35000.0
```

If entering a power-of-ten floating-point number using *b* instead of *e*, the number will be stored as a *bigfloat* number. Some examples are shown below:

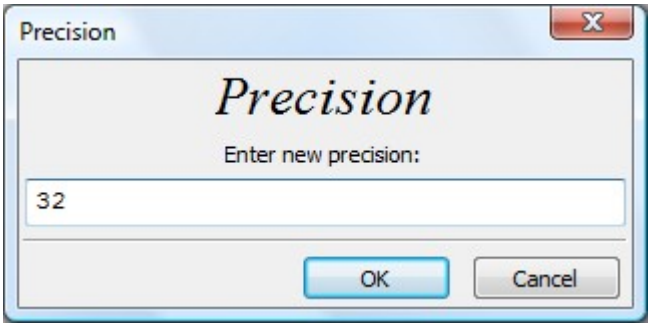
```
(%i19)  3.5b-2;-2.5b-3;1.2b2;-3.5b4;
(%o19)  3.5b-2
(%o20)  - 2.5b-3
(%o21)  1.2b2
(%o22)  - 3.5b4
```

Combining *float* and *bigfloat* numbers results in a *bigfloat* number, e.g.,

```
(%i25)  3.5b-4*2.7e2; 2.2e-2+1.2b-1; -3.2b2/2.0e3;
(%o25)  9.45b-2
(%o26)  1.42b-1
(%o27)  - 1.6b-1
```

Setting floating-point precision

By default, *Maxima* shows 16 digits in a floating-point number. Using the menu item *Numeric* > *Set Precision* ... produces a dialog form where you can enter the precision (number of digits) to show in your floating-point results, e.g.,



A simpler way to change the precision is to redefine parameter *fpprec*. In the following examples we change *fpprec* to values of 16, 32, 64, and 128, and then display the value of p %pi) using function *bfloat*:

```
(%i61)  fpprec:16$ bfloat(%pi);
(%o62)  3.141592653589793b0
(%i63)  fpprec:32$ bfloat(%pi);
(%o64)  3.1415926535897932384626433832795b0
(%i65)  fpprec:64$ bfloat(%pi);
(%o66)  3.1415926535897932384626433832795028841971693993751058209749445920b0
(%i67)  fpprec:128$ bfloat(%pi);
(%o68)  3.1415926535897932384626433832[71 digits]98214808651328230664709384460b0
```

Calculations with real numbers

In this sections we present calculations with real numbers involving not only simple arithmetic operations (+, -, *, /, ^), but also square roots (*sqrt*), other roots, absolute values (*abs*), trigonometric functions and their inverses (*sin, cos, asin, acos*, etc.), hyperbolic functions and their inverses (*sinh, cosh, asinh, acosh*, etc.), exponential (*exp*), natural logarithms (*log* – note: not ln), *ceiling*, *floor*, *fix*, and *float*. By default symbolic results will be provided. Use function *float* to obtain floating-point results. Please notice that the arguments of trigonometric functions are in radians, the natural angular unit. To convert from degrees to radians use the factor `%pi/180`. Try some of the following examples (see the results in your own *Maxima* installation):

```
2+(1/(2+1/(2+1/2)));
4./3.+3./4.+1./6.;
sqrt(1+(3/2)^3);
abs(-2.5+1/2.5);
sin(%pi/3+cos(%pi/3));
sqrt(exp(-2)+log(abs(-2+1/2)));
ceiling(3.25); floor(3.25); fix(3.25);
3.25-fix(3.25); sinh(2.5);
```

Evaluation of formulas

Evaluation of formulas is a common application of real number calculations. I suggest using a three-step approach:

1. Enter formula (remember to use : instead of =)
2. Enter list of values known, separated by $ to avoid taking space in the window
3. Use the command history to repeat the formula expression, which is now evaluated

If need be, use *float*(%) to obtain a floating-point value. Some examples are shown below. Example 2.1:

```
(%i1) hf : f*(L/D)*(V^2/(2*g));
```

$$(\%o1)\quad \frac{f\,L\,V^2}{2\,g\,D}$$

```
(%i2) f:0.017 $ L:1000 $ D:0.2 $ V:2.5 $ g:32.2 $

(%i7) hf : f*(L/D)*(V^2/(2*g));
(%o7) 8.249223602484472
```

Example 2.2:

```
(%i8)  kill(all) $ Q : Cd*A2*sqrt(2*g*Dh/(1+(D2/D1)^2));
```

$$(\%o1)\quad \frac{\sqrt{2}\,Cd\sqrt{Dh\,g}\,A2}{\sqrt{\frac{D2^2}{D1^2}+1}}$$

```
(%i2)  Cd:0.8 $ Dh:2.5 $ g:32.2 $ D1:0.1 $ D2:0.05 $ A2:float(%pi*D2^2/4);
(%o7)  0.0019634954084936

(%i8)  Q : Cd*A2*sqrt(2*g*Dh/(1+(D2/D1)^2));
(%o8)  0.01782698128442
```

Note: in this last example, *kill(all)* was used to clear all existing values. Also, the intermediate equation *A2:float(%pi*D2^2/4)* is calculated before calculating the value of *Q*.

Defining functions

To define a function write the function name and arguments, e.g., *f(x)*, *g(y)*, *h(x,y)*, followed by := and by the function definition. Evaluation of the functions is performed by replacing the unknowns with variable names or numerical values.

```
(%i9)  f(x) := x^3+1 $ f(a); f(2); f(1+3/2);
```

$$(\%o10)\quad a^3+1$$

$$(\%o11)\quad 9$$

$$(\%o12)\quad \frac{133}{8}$$

```
(%i13)  h(x,y) := sqrt(x^2+y^2) $ h(a,b); h(-3,2);
```

$$(\%o14)\quad \sqrt{b^2+a^2}$$

$$(\%o15)\quad \sqrt{13}$$

```
(%i16)  h(a+b,c+d) + f(1/a);
```

$$(\%o16)\quad \sqrt{(d+c)^2+(b+a)^2}+\frac{1}{a^3}+1$$

<u>Defining a function as a sequence of expressions</u>

Suppose that you want to define a function given by the following expression:

$$f(x)=\frac{2}{x^2+4}+\frac{x^3+3x}{x^2+4}+(x^3+3x)^2$$

Notice that the definition of the function includes a couple of groupings of expressions, namely, (x^2+4) and (x^3+3x), that appear twice in the expression for *f(x)*. It could be possible to calculate the function in three steps, namely,

- $a = x^2+4$
- $b = x^3+3x$
- $f(x)=\frac{2}{a}+\frac{b}{a}+b^2$

In *Maxima*, we can define a function such as this by using a sequence of expressions separated by commas. For example,

```
(%i3) f(x) := (a : x^2+4, b : x^3+3*x, 2/a+b/a+b^2);
```

$$(\%o3)\quad f(x):=\left(a:x^2+4,\ b:x^3+3x,\ \frac{2}{a}+\frac{b}{a}+b^2\right)$$

Check the full expression of the function by using, for example, *f(t)*:

```
(%i4) f(t);
```

$$(\%o4)\quad \left(t^3+3t\right)^2+\frac{t^3+3t}{t^2+4}+\frac{2}{t^2+4}$$

Evaluation of the function is straightforward as illustrated in the following examples:

```
(%i32) f(a+1);
```

$$(\%o32)\quad \left((a+1)^3+3(a+1)\right)^2+\frac{(a+1)^3+3(a+1)}{(a+1)^2+4}+\frac{2}{(a+1)^2+4}$$

```
(%i33) f(2);
(%o33) 198
```

Using function ratsimp - Function *ratsimp* (*rat*ional *simp*lification) can be used to simplify rational expressions such as fractions, polynomials, etc. This function will be presented in a later chapter in reference to simplification of algebraic expressions. It is introduced here to show an alternative way to define the function presented above:

```
(%i34) f(x):=(a:x^2+4,b:x^3+3*x,ratsimp(2/a+b/a+b^2));
```

$$(\%o34)\quad f(x):=\left(a:x^2+4,\ b:x^3+3x,\ \text{ratsimp}\left(\frac{2}{a}+\frac{b}{a}+b^2\right)\right)$$

The function definition is now:

```
(%i35)  f(t);
```

$$(\%o35)\quad \frac{t^8+10\,t^6+33\,t^4+t^3+36\,t^2+3\,t+2}{t^2+4}$$

In this case, evaluation of the function with algebraic arguments will produce a fully expanded expression, e.g.,

```
(%i40)  f(a+b);
```

$$(\%o40)\quad \frac{b^8+64\,b^7+1802\,b^6+29152\,b^5+296353\,b^4+1938465\,b^3+7967164\,b^2+18811651\,b+19536666}{b^2+16\,b+68}$$

Defining functions with a *block* statement

Functions that require more than one statements to be defined can use the *block* statement. The *block* statement is used if a *return* statement is to be included. The general form of a *block*-statement function is as follows:

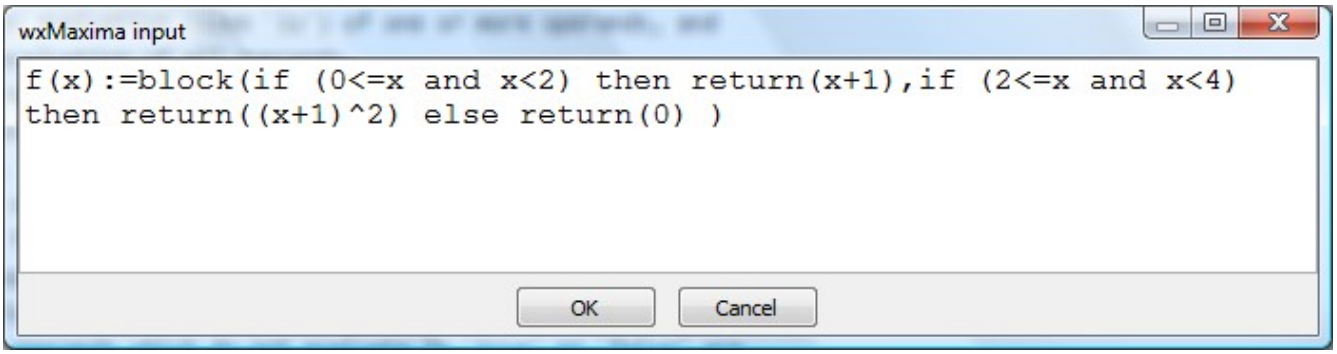
```
block([<variables with assignments>], <expressions>)
```

To illustrate the use of the *block* statement in defining a function, consider the function:

$$f(x)=\begin{cases} x+1, if\ 0\le x<2 \\ (x+1)^2, if\ 2\le x<4 \\ 0, elsewhere \end{cases}\quad .$$

Using the Multiline Input - To enter the command defining the function we will use the *Multiline Input* window, available by clicking the *multiline* icon available at the end of the *INPUT* line in the *wxMaxima* window  .

Clicking this icon produces the following window, in which we enter the definition of function *f(*x) using an *if-then* command, and an *if-then-else* command as shown:



Using vectors - To illustrate the operation of the function we will generate a vector of data by evaluating the function at points *x* = *-2, 0, 1, 2, 3, 4,* and *5*, i.e.,

```
(%i58) [f(-2),f(0),f(1),f(2),f(3),f(4),f(5)];
(%o58) [ 0 , 1 , 2 , 9 , 16 , 0 , 0 ]
```

*The **if-then*** and ***if-then-else*** constructs - In the definition of function f(x) shown above, we used both an *if-then* and an *if-then-else* constructs. The *if-then* construct has the general form:

```
if <condition> then <action>
```

The result from this construct is to perform the <action> if the <condition> is true, or do nothing otherwise.

On the other hand, the *if-then-else* construct has the general form:

```
if <condition> then <action 1> else <action 2>
```

The result from this construct is to perform <action 1> if the <condition> is true, or perform <action 2> if the condition is false.

Conditional statements - Conditional statements are statements that result in a *true* or *false* outcome. Numerical comparisons are common forms of conditional statements. In the definition of function *f(x)* used conditional statements such as:

```
2<=x and x<4
```

This is a combined conditional statement including the simple conditional statements

```
2<=x, x<4.
```

The following comparison operators can be used to produce conditional statements:

- < less than
- <= less than or equal to
- = equal to
- not = not equal to
- > greater than
- >= greater than or equal to

Logical particles - Logical particles, such as *and*, are used to combine simple conditional statements. The following logical particles are available in *Maxima*:

- and combined statement is true only if both composing statements are true
- or combined statement is true if one of the two composing statements is true
- not resulting statement is true if original statement is false, and vice versa

Defining multi-variate functions

Multi-variate functions can be defined in the same fashion of uni-variate functions, namely, by using the function name with a list of arguments and the := symbol. Some examples are shown below:

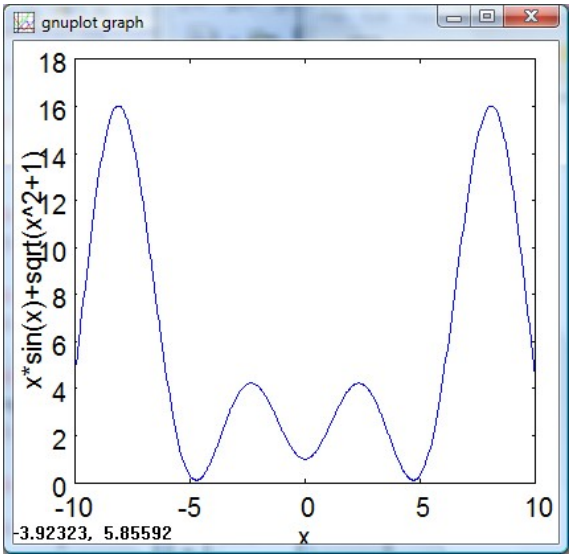
```
(%i1) g(x,y) := x*sin(y) + y*sin(x);
(%o1) g(x, y):=x sin(y)+y sin(x)

(%i2) g(a+b,c+d);
(%o2) (b+a)sin(d+c)+sin(b+a)(d+c)

(%i3) g(%pi/6,%pi/3);
(%o3) sqrt(3) %pi/12+%pi/6

(%i4) h(x,y,z) := (x+y)*exp(-z);
(%o4) h(x, y, z):=(x+y)exp(-z)

(%i5) h(a+1,b+1,c+1);
(%o5) (b+a+2)%e^-c-1

(%i6) h(3,2,4);
(%o6) 5 %e^-4
```

Plotting functions using plot2d and plot3d

Single-variable functions can be plotted using function *plot2d*. Some examples are shown below:

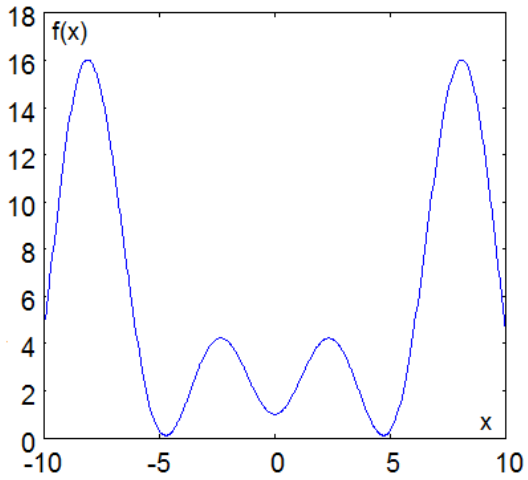
```
(%i7) f(x) := x*sin(x) + sqrt(1+x^2);
(%o7) f(x):=x sin(x)+sqrt(1+x^2)

(%i8) plot2d(f(x),[x,-10,10]);
```

The graph is shown in a *gnuplot* window. To export the graph you may want to do a print screen of the graph and then paste into a word processor for publishing or drawing program for editing. The figure below was copied into this document by using a print screen:



The numbers at the lower left corner are the coordinates of a point in the graph. Thus, the resulting *gnuplot graphs* in *Maxima* are interactive in the sense that you can place the cursor anywhere in the plot and get the coordinates of that particular point.

As mentioned earlier, the graph can be exported to a drawing (or graphics manipulation) software for editing. The following is an edited version of this plot:

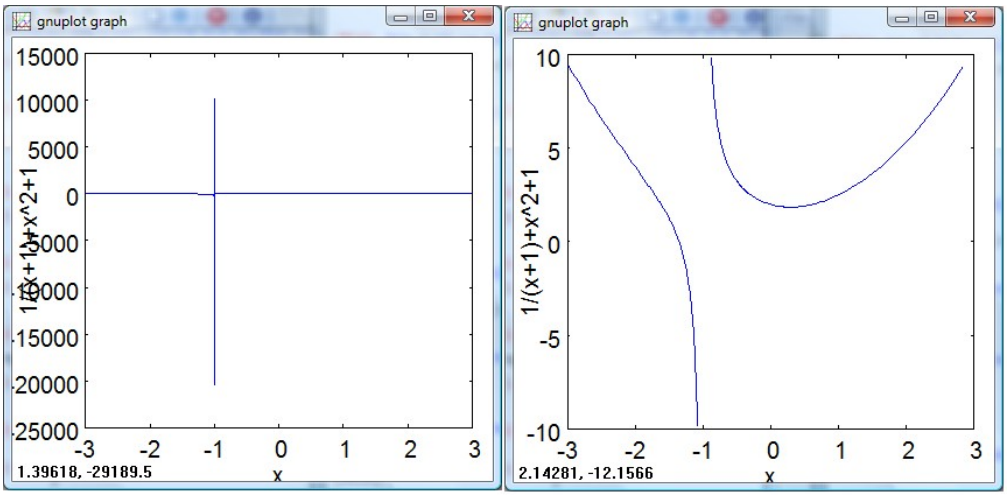


The following example shows a case where the user controls not only the range of values of *x*, but also those of *y*. Make sure to click off the *gnuplot graph* window before entering a new command in the *wxMaxima INPUT* line. In this example we plot the function:

```
f(x) := x^2+1+1/(x+1)$
```

The graph to the left was generated using: `plot2d(f(x),[x,-3,3]);`

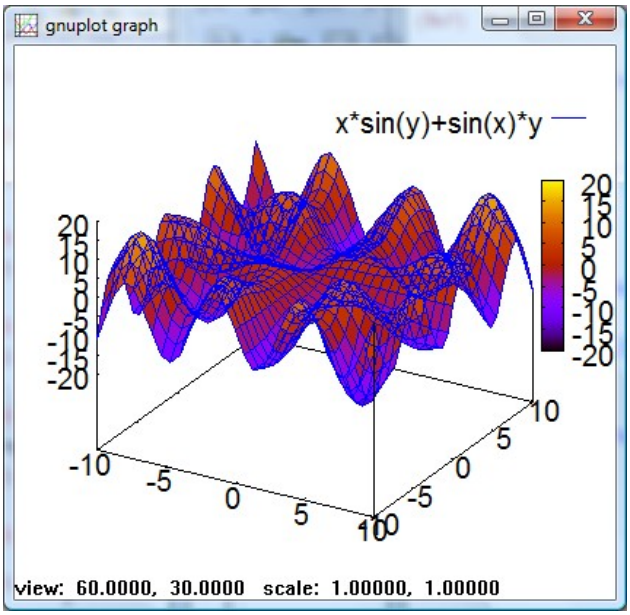
Because the function evaluates to $\pm\infty$ at $x = -1$, the range of values of y is extremely large, and the variation of the function near $y = 0$ is not clearly defined. To produce a more detailed graph, use the command: `plot2d(f(x),[x,-3,3],[y,-10,10]);`



The following example shows a bi-variate function plotted using function *plot3d*:

```
(%i16) g(x,y) := x*sin(y) + y*sin(x)$
(%i17) plot3d(g(x,y),[x,-10,10],[y,-10,10]);
```

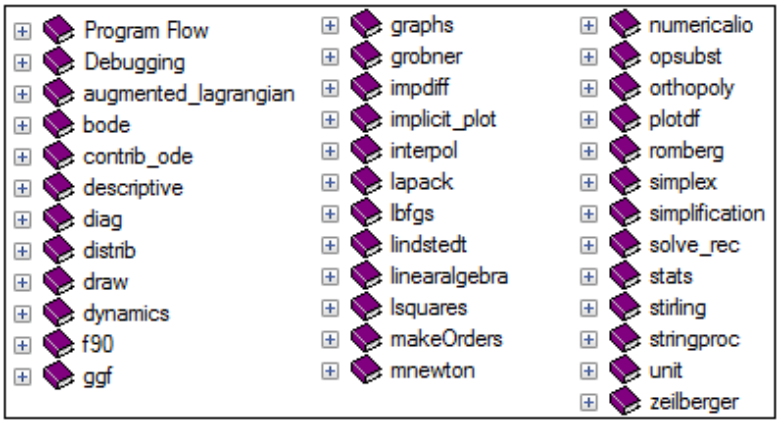
The graph is shown below:



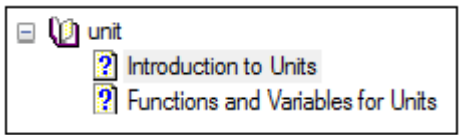
Additional information on the use of graphics functions in *Maxima* will be presented on a subsequent chapter on graphics.

Calculations using units

Operations with units in *Maxima* requires us to load the *unit* package. A package in *Maxima* is a collection of functions related to the package's theme. To see a list of the packages available in *Maxima*, launch the *Maxima Manual* by using the menu item *Help* > *Maxima help*, and click on the *Contents* tab. Then, scroll down until you pass all the chapters that start with an upper-case letter. The *Maxima* packages correspond to those chapters that start with a lower-case letter, as listed below:



If you click on the [+] icon next to the *unit* package, you will find the following items within the book icon:



Next, click on the *Introduction to Units* item to get information on the use of units in *Maxima*. Read the text on *Introduction to Units* in the *Maxima Manual*. As you scroll down you will get a listing of the functions available, namely:

- *setunits* - select preferred units for different dimensions
- *uforget* - clears any preferred units set with *setunits*
- *convert* - convert a unit to a different set of units
- *usersetunits* - overules default units with units selected by the user
- *metricexpandall* - allows selection of metric prefixes according to:

  0 - none. Only base units
  1 - kilo, centi, milli
  2 - giga, mega, kilo, hecto, deka, deci, centi, milli, micro, nano [default value]
  3 - peta, tera, giga, mega, kilo, hecto, deka, deci, centi, milli, micro, nano, pico, femto
  4 - all

- *%unitexpand* - value of the argument for *metricexpandall*

To get started using units, one needs to load the *unit* package:

```
(%i18)  load(unit);
************************************************************************
*                       Units version 0.50                             *
*          Definitions based on the NIST Reference on                  *
*              Constants, Units, and Uncertainty                       *
*       Conversion factors from various sources including              *
*                    NIST and the GNU units package                    *
************************************************************************

Redefining necessary functions...
Initializing unit arrays...
Done.
(%o18)  C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/contrib/unit/unit.mac
```

The International System of units (Systeme International, S.I.)

The basis of the *unit* package is the International System (S.I.) of units. For detailed information on the S.I. please visit the *U.S. National Institute of Standards and Technology* (*NIST*) web page on the S.I.:http://www.physics.nist.gov/cuu/Units/units.html

The S.I. defines seven basic units corresponding to the following base quantities:

- length — meter (m)
- mass — kilogram (kg)
- time — second (s)
- electric current — ampere (A)
- thermodynamic pressure — kelvin (K)
- amount of substance — mole (mol)
- luminous intensity — candela (cd)

Combinations of these units produce S.I. derived units such as area (m^2), velocity (m/s), and so on [see Table 2 in the NIST SI web page]. Some of these derived quantities have special names and symbols, e.g., [see Table 3 in the NIST SI web page]:

- force — newton (N)
- pressure, stress — pascal (Pa)
- energy, work, quantity of heat — joule (J)
- power — watt (W)
- electric charge — coulomb (C)
- electric potential difference — volt (V)
- capacitance — farad (F)
- electric resistance — ohm (Ω)
- magnetic flux — tesla (T)
- inductance — henry (H)
- luminous flux — lumen (lm)
- illuminance — lux (lx)

The S.I. in *Maxima*

Using the *unit* package in *Maxima* we can check the reduction of these derived units to its basic units. Compare the results of the following derived units as shown in *Maxima* with those in Table 3 in the NIST SI web page . To save space, we will create a vector of derived units and check their expressions in terms of the basic units:

```
(%i2) [N,Pa,J,W];
```

$$(\%o2)\quad \left[\frac{kg\ m}{s^2},\frac{kg}{m\ s^2},\frac{kg\ m^2}{s^2},\frac{kg\ m^2}{s^3}\right]$$

```
(%i3) [C,V,F,Ohm,T,H];
```

$$(\%o3)\quad \left[s\ A,\frac{kg\ m^2}{s^3\ A},\frac{s^4\ A^2}{kg\ m^2},\frac{kg\ m^2}{s^3\ A^2},\frac{kg}{s^2\ A},\frac{kg\ m^2}{s^2\ A^2}\right]$$

```
(%i4) [lm,lx];
```

$$(\%o4)\quad \left[cd,\frac{cd}{m^2}\right]$$

We can attach units to values and operate with them as illustrated by this simple example in which we calculate the force required to accelerate a mass of *3 kg* to a constant value of *1.5 m/s²*:

```
(%i8)  mass : 3*kg $  acc : 1.5*m/s^2 $  force : mass*acc;
```

$$(\%o10)\quad \frac{9\ kg\ m}{2\ s^2}$$

Use of function *convert*

Notice that the result will be given in terms of the basic units, thus, instead of showing the result in newtons it is shown in $kg{\cdot}m/s^2$. Using function *convert* it is possible to convert this result to *N*:

```
(%i11)  convert(%,N);
```

$$(\%o11)\quad \frac{9}{2}N$$

Here is another example, in which we calculate the power developed by a particle moving at a velocity of *2.5 m/s* under the effect of a force of *4.5 N*:

```
(%i12)  vel : 2.5*m/s $ force : 4.5*N $ power : force * vel;
```

$$(\%o14)\quad \frac{45\ kg\ m^2}{4\ s^3}$$

```
(%i15)  convert(%,W);
```

$$(\%o15)\quad \frac{45}{4}W$$

The following example shows the calculation of the kinetic energy of a particle of *10 kg* of mass moving at *10 m/s*:

```
(%i5)  mass : 10*kg $ vel : 10*m/s $ KE = 1/2*mass*vel^2;
```

$$(\%o7)\quad KE = 500\frac{kg\ m^2}{s^2}$$

```
(%i8)  convert(%,J);
(%o8)  KE = 500 J
```

Setting default units

Suppose that we are working with units commonly used in fluid mechanics, and that we would like to reduce all unit results to combinations of newtons (N), pascals (Pa), joules (J), and watts (W). In that case, we can force the default units to be:

```
(%i10)  setunits([N,Pa,J,W]);
(%o10)  done
```

Thus, the following calculations will default to these units:

```
(%i11)  [(5*kg)*(32.2*m/s^2), (45*N)/(18*m^2), (1.2*N)*(2*m), (3.5*N)*(1.5*m)/(23*s)];
```

$$(\%o11)\quad \left[\ 161\ N\ ,\ \frac{5}{2}\ Pa\ ,\ \frac{12}{5}\ J\ ,\ \frac{21}{92}\ W\ \right]$$

Resetting the default units

To reset the result to the basic units (*kg*, *m*, *s*, etc.), use function *uforget*:

```
(%i14)  uforget([N,Pa,J,W]);
(%o14)  [ false , false , false , false ]
```

With this command, repeating the calculations shown above will produce the following results:

```
(%i15)  [(5*kg)*(32.2*m/s^2), (45*N)/(18*m^2), (1.2*N)*(2*m), (3.5*N)*(1.5*m)/(23*s)];
```

$$(\%o15)\quad \left[\ 161.0\ \frac{kg\ m}{s^2}\ ,\ \frac{5}{2}\ \frac{kg}{m\ s^2}\ ,\ \frac{12}{5}\ \frac{kg\ m^2}{s^2}\ ,\ \frac{21}{92}\ \frac{kg\ m^2}{s^3}\ \right]$$

Using prefixes in the S.I.

You are probably familiar with some of the most commonly used prefixes in the S.I., e.g.,

- nano (as in *nanometer*, or *nm* = 10^{-9} *m*)
- micro (as in *micrometer*, or $\mu m = 10^{-6}$ *m*)
- milli (as in *millimeter, or mm* 10^{-3} *m*)
- centi (as in *centimeter*, or *cm* = 10^{-2} *m*)
- kilo (as in *kilometer*, or *km* = 10^{3} *m*)
- mega (as in *megawatts*, or *MW* = 10^{6} *W*)
- giga (as in *gigabytes*, or GB $\approx 10^3$ KB[1] $\approx 10^6$ B).

As a matter of fact, the S.I. unit of mass, the *kilogram*, is actually a prefixed unit (*1 kg* = 10^3 *g*, *g* = *gram*). For a complete list of S.I. prefixes visit the NIST S.I. page:

http://www.physics.nist.gov/cuu/Units/prefixes.html

Using the *unit* package in *Maxima* we can use the S.I. prefixes as illustrated in the following examples (notice that μm is written as `%mum`). The prefixes use the following letters:

1 Strictly speaking, a kilobyte is not exactly 1000 bytes, but 2^{10} bytes = 1024 bytes $\approx 10^3$ bytes. Similarly, a gigabyte is 2^{10} kilobytes = 2^{20} bytes = 1048576 bytes $\neq 10^6$ bytes.

Y (yotta = 10^{24})	Z (zetta = 10^{21})	E (exa = 10^{18})	P (peta = 10^{15})
T (tera = 10^{12})	G (giga = 10^{9})	M (mega = 10^{6})	k (kilo = 10^{3})
h (hecto = 10^{2})	da (deka = 10^{1})	d (deci = 10^{-1})	c (centi = 10^{-2})
%%m (milli = 10^{-3})	%mu (micro = 10^{-6})	n (nano = 10^{-9})	p (pico = 10^{-12})
f (femto = 10^{-15})	a (atto = 10^{-18})	z (zepto = 10^{-21})	y (yocto = 10^{-24}).

Notice that the only "strange" characters are `%%m` for *milli* and `%mu` for *micro*. The other letters are the same as written.

Calculations involving S.I. prefixes are shown below:

```
(%i27) (18*%mug)*(23.5*cm/s^2);convert(%,N);
```

$$(\%o27)\quad \frac{423}{100000000000}\frac{kg\,m}{s^2}$$

$$(\%o28)\quad \frac{423}{100000000000}N$$

```
(%i29) (45.6*kN)*(1.2*%%mm);convert(%,J);
```

$$(\%o29)\quad \frac{1368}{25}\frac{kg\,m^2}{s^2}$$

$$(\%o30)\quad \frac{1368}{25}J$$

```
(%i31) (45.6*kN)*(1.2*%mum)/(3.5*s);convert(%,W);
```

$$(\%o31)\quad \frac{342}{21875}\frac{kg\,m^2}{s^3}$$

$$(\%o32)\quad \frac{342}{21875}W$$

Function *convert* can be used to convert results to prefixed units (e.g., *mm* or *kPa*), as illustrated below:

```
(%i34) convert(0.025*m,%%mm);
`rat' replaced 0.025 by 1/40 = 0.025
`rat' replaced 0.025 by 1/40 = 0.025
(%o34) 25 %%mm
(%i37) convert((1250*N)/(150*cm^2),kPa);
```

$$(\%o37)\quad \frac{250}{3}kPa$$

The *MKS* and *cgs* systems of units

Before the S.I. system standardize metric units, there were several "metric" systems used in physics calculations. Two of the best know were the *MKS* (*meter-kilogram-second*) and the *cgs* (*centimeter-gram-second*) given in terms of their basic units of length, mass, and time. The MKS system is basically the same as the S.I. system. However, since the units in the *cgs* system are much smaller in magnitude than the corresponding unit in the *MKS* system, the *cgs* system is still used when dealing with small masses and lengths.

The conversion of the basic units of the cgs to the MKS system is straightforward:

```
(%i7)  [convert(cm,m),convert(g,kg)];
```

$$(\%o7)\quad \left[\frac{1}{100}\,m\,,\frac{1}{1000}\,kg\right]$$

The unit of force in the *cgs* system is the *dyne* (abbreviated *dyn*), such that $1\ dyn = 1\ g \times 1\ cm/s^2$:

```
(%i8)  convert(dyn,N);
```

$$(\%o8)\quad \frac{1}{100000}\,N$$

The unit of work or energy in the *cgs* system is the *erg* (I think this is an abbreviation of the word *energy*), and it is defined as $1\ erg = 1\ dyn \times 1\ cm$. However, the *Maxima unit* package does not contain the *erg* as one of its units. We can still find a conversion from *ergs* to *joules* (*J*) by using:

```
(%i38)  convert((1*dyn)*(1*cm),J);
```

$$(\%o38)\quad \frac{1}{10000000}\,J$$

Alternatively, you can add a variable called *erg* by using:

```
(%i13)  erg:dyn*cm;
```

$$(\%o13)\quad \frac{1}{10000000}\,\frac{kg\,m^2}{s^2}$$

Converting to *joules* (*J*) we get:

```
(%i4)  convert(erg,J);
```

$$(\%o4)\quad \frac{1}{10000000}\,J$$

The *electron-Volt* (*eV*)

Since we are dealing with small quantities, we should mention that the *unit* package does contain the *electron-Volt* (*eV*) as a unit of energy. In electrical fields, a charge *q* (typically given in *coulombs*, or *C*) affected by an electric potential difference *V* (also referred to as *voltage*, and given in *volts* or *V*) has a potential energy equal to the product of the charge and the voltage. The electron-volt is a very small unit of energy equivalent to the charge of 1 electron affected by a voltage of 1 volt. Converted to *joules* (*J*), 1 *eV* is equal to:

```
(%i12) convert(eV,J);
`rat' replaced 1.6021765000000002E-19 by 1/6241509596477041664 = 1.6021765000000002E-19
`rat' replaced 1.6021765000000002E-19 by 1/6241509596477041664 = 1.6021765000000002E-19
```

(%o12) $\frac{1}{6241509596477041664} J$

To see that conversion factor as a floating-point result it is necessary to type the following (this can be accomplished by selecting the denominator, and cutting and pasting into the INPUT line):

```
(%i14) float(1/6241509596477041664);
```

(%o14) $1.6021765000000002\ 10^{-19}$

Thus, *1 eV* = 1.602×10^{-19} *J*.

Other metric units available in the *unit* package

Some of the metric units included in the *units* package that we haven't dealt with before include the following:

- mass — grain (gr)
- time — minute (%min)
- frequency — hertz (Hz)
- pressure — torricelli (torr)
- electric conductance — siemens (S)
- volume — liter (L), basic unit = m^3
- absorbed dose — gray (Gy)
- catalytic activity — katal (kat)

The conversions for these units are shown below:

```
(%i16) [gr,%min,Hz,torr,S,L,Gy,kat];
```

(%o16) $[\frac{53}{817915} kg, 60\ s, \frac{1}{s}, \frac{44663}{335} \frac{kg}{m\ s^2}, \frac{s^3 A^2}{kg\ m^2}, m^3, \frac{m^2}{s^2}, \frac{mol}{s}]$

The *torricelli* (*torr*) can be converted to a more commonly used unit, the *pascal* (*Pa*):

```
(%i39)  convert(torr,Pa);
`rat' replaced 133.32239 by 44663/335 = 133.3223880597015
`rat' replaced 133.32239 by 44663/335 = 133.3223880597015
                                44663
(%o39)                          ----- Pa
                                 335
```

Also, we should point out that electric conductance is the inverse of electric resistance, therefore, the *siemens* (*S*) and the *ohm* (Ω) are inverse units, or $S \cdot \Omega = 1$. Here we use *Maxima* to check that result:

```
(%i41)  S*Ohm;
(%o41)  1
```

Metric units not available in the *unit* package

This list includes other metric units that are not standard S.I. units, but are commonly used in practice. The list also includes units such as the nautical mile and the know, that, although not metric in origim, are accepted for use in the S.I. For conversion factors, see Table 7 in the NIST S.I. web page]

- length: angstrom (Å), astronomical unit (ua), nautical mile, hectare (ha)
- area: are (a), hectare (ha), barn (b)
- time: hour (h), day (d)
- velocity: knot
- mass: metric ton (t), unified atomic mass unit (u)
- pressure: bar

If needed in *Maxima*, you can define these units as done for the *erg*, above.

The English System (E.S.) of units

While the International System (S.I.) has been adopted throughout the world, the English System of units, also known as the British System or the Imperial System, is still very much prevalent in the United States of America[2]. The *unit* package includes only two of the E.S. units, namely,

- length inch (%in)
- mass slug (slug)

The corresponding conversions to S.I. basic units are the following:

2 According to the *Wikipedia* entry on the *International System,* as of May 6, 2008, "(t)hree nations have not officially adopted" the S.I. "as their primary or sole system of measurement: Liberia, Myanmar, and the United States.

```
(%i2)  [%in,slug];
(%o2)  [127/5000 m, 8517/5836 kg]

(%i3)  [float(127/5000),float(8517/5836)];
(%o3)  [0.0254, 1.45938999314599]
```

If you want to do calculations with units of the English System, you could define the following additional unit conversions:

- length foot (ft) = 12 in, mile (mi) = 5280 ft
- force pound (lb) = 1 slug ft/s^2
- energy British thermal unit (BTU) = 1.0545×10^3 J
- power horsepower (hp) = 550 lb·ft/s

Using *Maxima*:

```
ft : 12*%in $ mi: 5280*ft $ lb : slug*ft/s^2 $ BTU : 1.0545*10^3*J $ hp : 550*lb*ft/s $
```

In terms of the basic S.I. units the units defined above are:

```
(%i18)  [ft,mi,lb,BTU,hp];
(%o18)  [381/1250 m, 201168/125 m, 3244977 kg m/(7295000 s^2), 1054 kg m^2/s^2, 13599698607 kg m^2/(182375000 s^3)]
```

Converting to related units (e.g., ft to m, mile to km, lb to N, etc.):

```
(%i18)  [ft,mi,lb,BTU,hp];
(%o18)  [381/1250 m, 201168/125 m, 3244977 kg m/(7295000 s^2), 1054 kg m^2/s^2, 13599698607 kg m^2/(182375000 s^3)]
```

The fractions shown above, written in floating-point format, are:

```
[float(381/1250),float(25146/15625),float(3244977/7295000),float(13599698607/182375000)];
[0.3048, 1.609344, 0.4448220699109, 74.56997179986293]
```

Functions *unitinfo* and *addunit*

The *unit* package is still in development, therefore, it still has some functions that are not yet (May, 2008) implemented such as:

- *unitinfo* provides definition for a specific unit
- *addunits* allows user to add other units.html2

When the *addunits* function finally becomes available, it will be possible to add the units defined above (e.g., *erg, ft, mi, lb, BTU, hp*) as "atom" units. Thus, while a conversion such as:

```
(%i14)  convert(10*BTU,J);
(%o14)  10545 J
```

is possible, the inverse conversion will not be available until *BTU* is added using *addunits*. Currently, an attempt to produce that conversion fails:

```
(%i15)  convert(10*J,BTU);
concat: argument is not an atom: 2109 %energy/2
#0: convert(expression=10*J,desiredunits=[1054.5*J])(unit.mac line 747)
#1: convert(expression=10*J,desiredunits=1054.5*J)(unit.mac line 768)
 -- an error.  To debug this try debugmode(true);
```

The only possibility is to force the conversion using an expression such as:

```
(%i21)  10*J/convert(BTU,J); float(%);
(%o21)  20/2109
(%o22)  0.0094831673779042
```

A result like this shows only the magnitude of the conversion factor. The user will have to interpret this result, for example, as *10 J = 0.00948 BTU*.

The *unit* mac file

In Chapter 1, *Introduction to Maxima*, we learned that a batch file is typically saved using a suffix *.mac*. In the failed conversion attempt shown above, namely, `convert(10*J,BTU)`, the error message provided by *Maxima* shows two references to a file called *unit.mac*:

```
#0: convert(expression=10*J,desiredunits=[1054.5*J])(unit.mac line 747)
#1: convert(expression=10*J,desiredunits=1054.5*J)(unit.mac line 768)
```

This file, *unit.mac*, is the batch file that is loaded when we first invoke the *unit* package (using `load(unit)`). The response from *Maxima*, as shown in page 2-12, above, includes the output:

```
(%o18)  C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/contrib/unit/unit.mac
```

This output represents the path of the *unit.mac* file in a *Windows Vista* environment. The path shown above abbreviates the folders *Program Files* to *PROGRA~1* and *Maxima-5.14.0* to *MAXIMA~1.0*. This is a throw back to the days of *Windows 3.5* – late 1980's to mid 1990's -- when folder names could only use 8 characters. When *Windows 95* came along, longer

folder names where allowed, but many programs still kept the 8-letter restriction in reporting file paths, as in the case above. Taking into account such abbreviations, we will recognize the path to the *unit.mac* file as:

```
c:\Program files\Maxima-5.14.0\share\maxima\5.14.0\share\contrib\unit\unit.mac
```

With this information you can navigate to the corresponding file and open it using a text editor (e.g., *WordPad* in *Windows Vista*). As you scroll down the file you'll find the line:

```
/*====================== User Functions ======================*/
```

Below this line you will see a number of functions including: *setunits*, *convert*, etc., i.e., functions that we have used in the examples above. The *User Functions* segment of the *unit.mac* file also includes functions not yet implemented such as *addunits* and *unitsinfo*, as indicated above. These functions are indicated in a comment segment (starts with /* and ends with */):

```
/*  Not yet implemented

/* Prints out information about a unit */
unitinfo(unit):= block([letrat:true,result,dimension],

/*  Allows the user to create their own units */
addunits([[unitname(s)],definition,0]):=
*/
```

You will also find the following functions:

- showabbr shows abbreviation of a full-named unit
- showfullname shows full name of an abbreviated unit
- dimension shows the dimension corresponding to a given unit

Close the *unit.mac* file without making any changes, and try some examples of functions *showabbr*, *showfullname*, and *dimension*:

```
(%i25) showabbr(kilogram);
(%o25) kg
(%i26) showfullname(J);
(%o26) [ joule , joules ]
(%i27) dimension(W);
                          2
                %length  %mass
(%o27)          ---------------
                        3
                %time
```

Use of function *dimension* in dimensional analysis

Function *dimension* can be useful to determine the dimensions of a given quantity, as in the example shown above for *watts* (*W*). The seven basic dimensions of the S.I. can be found by using function *dimension* with the following basic units:

```
[dimension(m),dimension(kg),dimension(s),dimension(A),dimension(K),dimension(mol),dimension(cd)];
[ %length , %mass , %time , %current , %temperature , %amount_of_substance , %luminous_intensity ]
```

Dimensional analysis is a mathematical technique used, for example, in fluid mechanics, to relate the different variables involved in a physical phenomenon through their dimensions. In fluid mechanics, the basic dimensions typically used are *mass*(M), *length* (*L*), and time (*T*), or *force*(*F*), *length* (*L*), and time (*T*). Thus, you will see references to using the *MLT* or the *FLT* basic dimensions in the solution of a dimensional analysis problem.

Because the S.I. is based on *MLT* units, use of function *dimension* in the *unit* package will, by default, use the *MLT* system of basic dimensions. The basic dimensions of the English System (E.S.) are actually *FLT*, since the unit of force, the *pound* (*lb*), rather than the unit of mass, the *slug*, is the preferred basic unit. However, with the *unit* package in *Maxima* we are forced to use the *MLT* system of basic dimensions, and there is no way to implement the *FLT* system of basic dimensions. This is, however, no obstacle to the determination of the dimensions of any given physical quantity as long as we know what the units of that quantity are in the S.I.

For fluid mechanics problems, where most of the phenomena involves purely mechanical quantities (i.e., no electrical current, no amount of substance, no luminous intensity, and, most of the time, no temperature), the user needs to know the S.I. units of the following quantities:

- mass — kg
- length — m
- time — s
- area — m^2
- volume — m^3
- velocity — length/time = m/s
- acceleration — velocity/time = length/time^2 = m/s^2
- force or weight — mass $\times$ acceleration[3] = $kg{\cdot}m/s^2$
- volumetric discharge[4] — volume/time = m^3/s
- mass discharge[5] — mass/time = kg/s
- weight discharge[6] — weight/time = N/s
- density — mass/volume = kg/m^3
- specific weight — weight/time = N/s

3 This is Newton's second law, F = ma. In the case of weight, W = mg, where g = acceleration of gravity,
4 Also known as the flow rate,
5 Also known as mass flow rate,
6 Also known as weight flow rate,

- surface tension[7] force/length = N/m
- pressure, stress[8] force/area = N/m^2 = Pa
- work, energy, heat force × length = N·m = J
- power work/time or force × velocity = J/s = W
- viscosity[9] shear stress/velocity gradient = Pa/(1/s) = Pa·s
- kinematic viscosity viscosity/density = area/time = m^2/s

Knowing these units, then we can find the dimensions of all these quantities:

- mass, length, time, area, volume, velocity, and acceleration:

```
[dimension(kg),dimension(m),dimension(s),dimension(m^2),dimension(m^3),dimension(m/s),dimension(m/s^2)];
```

$$[\,\%mass\,,\%length\,,\%time\,,\%length^{2}\,,\%length^{3}\,,\frac{\%length}{\%time}\,,\frac{\%length}{\%time^{2}}\,]$$

- force, volumetric discharge, mass discharge, weight discharge, density, specific weight:

```
[dimension(N),dimension(m^3/s),dimension(kg/s),dimension(N/s),dimension(kg/m^3),dimension(N/m^3)];
```

$$[\frac{\%length\ \%mass}{\%time^{2}}\,,\frac{\%length^{3}}{\%time}\,,\frac{\%mass}{\%time}\,,\frac{\%length\ \%mass}{\%time^{3}}\,,\frac{\%mass}{\%length^{3}}\,,\frac{\%mass}{\%length^{2}\ \%time^{2}}\,]$$

- surface tension, pressure or stress, work or energy or heat, power:

```
(%i34)  [dimension(N/m),dimension(Pa),dimension(J),dimension(W)];
```

$$(\%o34)\quad [\frac{\%mass}{\%time^{2}}\,,\frac{\%mass}{\%length\ \%time^{2}}\,,\frac{\%length^{2}\ \%mass}{\%time^{2}}\,,\frac{\%length^{2}\ \%mass}{\%time^{3}}\,]$$

- viscosity, kinematic viscosity:

```
(%i42)  [dimension(Pa*s),dimension(m^2/s)];
```

$$(\%o42)\quad [\frac{\%mass}{\%length\ \%time}\,,\frac{\%length^{2}}{\%time}\,]$$

While these examples illustrate the use of dimensional analysis with the *unit* package, you can find a more detailed set of functions for dimensional analysis in the *physics/dimen* package. This dimensional analysis package is discussed in the next section.

---

7 Surface tension is a property of a liquid free-surface and is defined as the force per unit length that the surface exerts on a body in contact with the surface.

8 Also, the modulus of elasticity of a fluid has units of pressure or stress

9 Based on Newton's law of viscosity for, properly named, Newtonian fluids. The viscosity is the quantity μ in the equation $\tau = \mu \cdot \frac{du}{dy}$, where τ is shear stress and *du/dy* is a velocity gradient (velocity/length).

Using the *physics* package for units, physical constants, and dimensional analysis
The *physics* package is a contributed *Maxima* package available in the following folder or directory in a *Windows Vista* environment (a similar path is available for other environments): `c:\Program files\Maxima-5.14.0\share\maxima\5.14.0\share\physics`

Within that folder you will find the following *.mac* files:

- dimen.mac - dimensional analysis
- physconst.mac - physical constants
- units.mac - extensive list of units
- dimension.mac - dimensional analysis

In the same directory you will find the following files:

- dimen.dem - demo file for *dimen.mac*
- dimension.html - html page with instructions for *dimension.mac*
- dimension.pdf - pdf file with instructions for *dimension.mac*
- dimension.tex - tex file with instructions for *dimension.mac*

First, we will describe the *dimen.dem* file.

Using the *dimen* demo file
A demo file is simply a *Maxima* batch file, typically, ending with the suffix *.dem*. To activate a demo file simply type the command *demo* with the file name (minus *.dem*), in double quotes, as argument. The launching of the *dimen* demo file is shown below:

```
(%i1)  demo("dimen");
batching #pC:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/physics/dimen.dem
 At the _ prompt, type ';' followed by enter to get next demo
(%i2)  load(dimen.mac)
Warning - you are redefining the Maxima function listify
_ ;
```

Notice that the first thing this demo file does is to load the file *dimen.mac*, i.e., load the *dimen* package. The red underline after input (%i2), above, indicates a pause in the execution of the demo file. Press [Enter] to continue its execution. The first two examples in the demo file are listed below.

```
(%i3)  get(β, 'dimension)
(%o3)  false
_ ;
(%i4)  dimension(β = temperature)
(%o4)  done
```

At this point, just continue pressing [Enter] until the end of the demo file is reached. You will recognize the end of the file when the pause prompt no longer appears.

To understand what these examples mean you will have to open the demo file using a text editor. The demo file is typically well documented, but the comment lines are not shown in the *wxWindow* interface. For that reason it is necessary to read the comments in the original demo file. The figure below shows the first ten lines in the *dimen.dem* file using *Notepad++*[10] as the text editor.

```
1  /* First load the necessary file: */
2  load("dimen.mac")$
3  /* It is conjectured that for thermistors there is a physical
4  relationship between the voltage drop, current, ambient temperature,
5  room-temperature resistance, convective heat transfer coefficient, and
6  a constant, BETA, having the dimension of temperature.  First, to see
7  if the dimension of BETA is already known: */
8  get(beta, 'dimension);
9  /* It is not.  To establish it: */
10 dimension(beta=temperature);
```

Comment lines, starting with /* and ending in */, precede each function call, and so it is easy to understand the operation of each function.

Read the *dimen.dem* file and run the demo again to understand the use of functions:

- get - to get dimensions of a variable
- dimension - to define the dimension(s) of a variable
- nondimensionalize - to determine a set of dimensionless variables sufficient to characterize the physical relation
- cons - use if a physical constant is involved so that it is treated as a pure value.

These functions use the dimensions *mass*, *length*, *time*, and *temperature* as basic dimensions. Using a text editor open the file *dimen.mac* and find the command that starts with the words *dimension([*. Below that line, you will find a listing of all the quantities available in the *dimen* package: `acceleration, angle, angularacceleration, angularmomentum, angularvelocity, area, boltzmannsconstant, capacitance, charge, current, currentdensity, density, distance, electricfield, electricpermittivity, electricpermittivityofavacuum, energy, enthalpy, frequency, filmcoefficient, flow, gravityconstant, heat, heatcapacity, heattransfercoefficient, inductance, internalenergy, kinematicviscosity, length, mass, moment, momentum, magneticinduction, magneticpermittivity, planckconstant, poissonsratio, power, pressure, resistance, specificheat, speedoflight, shearmodulus, surfacetension, stefanboltzmannconstant, stress, strain, temperature, thermalconductivity, thermalexpansioncoefficient, thermaldiffusivity, tyme, velocity, volume, voltage, viscosity, work, youngsmodulus.`

10 Notepad++ is available for download at http://notepad-plus.sourceforge.net/uk/site.htm

Using the *units* package

Use a text editor to open the file *units.mac* to check the listing of units provided when loading the *units* package:

```
newton, joule, watt, acre, ampres, ohm,angstrom, are, astronomicalunit, atmosphere,
bar, barreldry, barrel, barreloil, baryl, bolt, btu, bushel, candle, centigram,
centiliter, centimeter, centimeter of mercury, chain, circularmil, cord, coulombs,
cup, cycle, dalton, day, decigram, deciliter, decimeter, degree, degreef, degreec,
degreer, dekagram,dekaliter, dekameter, dyne, ell, empica, er, farads, faraday,
fathom, feet, feetofwater, fluidounce, footcandle, furlong, gallon, gallonimperial,
gill, grade, gram, gramcalorie, hand, hectogarams, hectoliter, hectometer,
hectowatt, henries, hogshead, horsepower, hour, inch, inchofmercury, joules,
kilograms, kilometer, kilowatts, knot, league, lightyear, link, linksurveyor,
liter, lumen, lix, megohm, meters, microfarad, microgram, microhm, microliter,
micron, microsecond, milenautical, mile, millier, millimicron, milligram,
millihenry, milliliter, millimeter, millisecond, mil, minute, minersinch, minims,
myriagram, myriameter, myriawatt, neper, newtons, ohms, parsec, peck, pint,
poundal, poise, pound, lb, lbf, poundmass, lbm, ounce, ounces, ounceavoirdupois,
ouncetroy, quartdry, quart, qt, radian, rad, revolution, rod, slug, seconds,
sphere, stoke, statcoulomb,steradians, tablespoon, teaspoon, volts, watts, week,
yard.
```

Loading the *units* package allows the user to find the corresponding S.I. units for the myriad of units listed above. Some examples are shown below:

```
(%i4) [furlong, yard, inch, ft, mile];
(%o4) [ 201.168 meter , 0.9144 meter , 0.0254 meter , 0.3048 meter , 1609 meter ]
```

```
[milligram, centigram, gram, ounce];
[ 9.9999999999999995 10^-7 kilogram , 1.0000000000000001 10^-5 kilogram , 0.001 kilogram , 0.02835 kilogram ]
```

```
[cup, gallon, liter, hectoliter, microliter];
[ 2.3550000000000001 10^-4 meter^3 , 0.003785 meter^3 , 0.001 meter^3 , 0.1 meter^3 , 1.0000000000000001 10^-9 meter^3 ]
```

```
(%i8) [rad, revolution];
(%o8) [ radian , 6.2831853 radian ]
```

Note: Do not load the *unit* package after loading the *units* package. If you try that *Maxima* will respond with an error and the *unit* package will not be loaded. On the other hand, loading the *unit* package after loading the *units* package does work, but you loose some of the functionality in the *unit* package, such as converting to composite units. In such case all conversions reduce to basic SI units as defined in the *units* package. My advice is to open two *wxMaxima* windows and load the two packages in each of the two windows. By cutting and pasting you can go back and forth between the two windows and take advantage of the full capability of the two packages *unit* and *units*.

Using the *physconst* package

The *physconst* package is used to load the definitions of many standard physical constants. Using a text editor, open the file *physconst.mac* to see the definitions and symbols for these constants. The first three universal constants listed in that file are:

```
/*speed of light in vacuum*/
   numerval(%%c, 299792458*m/s)$
   numerval(%c_0, 299792458*m/s)$

/*magnetic constant*/
   numerval(%u_0, 4*%pi*1E-7*N/(A^2))$
   numerval(%mu_0, 4*%pi*1E-7*N/(A^2))$

/*electric constant*/
   numerval(%e_0, 8.854187817E-12*F/m)$
   numerval(%epsilon_0, 8.854187817E-12*F/m)$
```

In order to get the value of a given constant, use function float. The following two examples correspond to the speed of light (`%%c`) and the universal gravitation constant (`%G`):

```
(%i3)  float(%%c);
                                 299792458 m
(%o3)                            -----------
                                      s

(%i4)  float(%G);
                                                    -11  3
                                 6.6729999999999999 10    m
(%o4)                            ----------------------
                                              kg s
```

Using the *dimension* package

Within the folder `c:\Program files\Maxima-5.14.0\share\maxima\5.14.0\share\physics`, you will find the following documentation files that demonstrate the use of package *physics/dimension*:

- dimension.pdf
- dimension.html
- dimension.tex

Open that documentation in your preferred format and follow the examples contained within the document to understand its operation.

Note: I could not get *dimension.mac* to load in *Maxima*.

3 Expressions, complex numbers, polynomials, and fractions in *Maxima*

In this Chapter we introduce *Maxima* functions that allow the manipulation of algebraic, logarithmic, exponential, and trigonometric expressions, among others. The chapter also presents manipulation of factorials and related functions, as well as operations with complex numbers.

The *Simplify* menu

The *Simplify* menu in the *wxMaxima* includes all the options shown in Figure 3.1. In this Chapter we will use these menu options in the simplification of algebraic, trigonometric, factorials, and complex expressions.

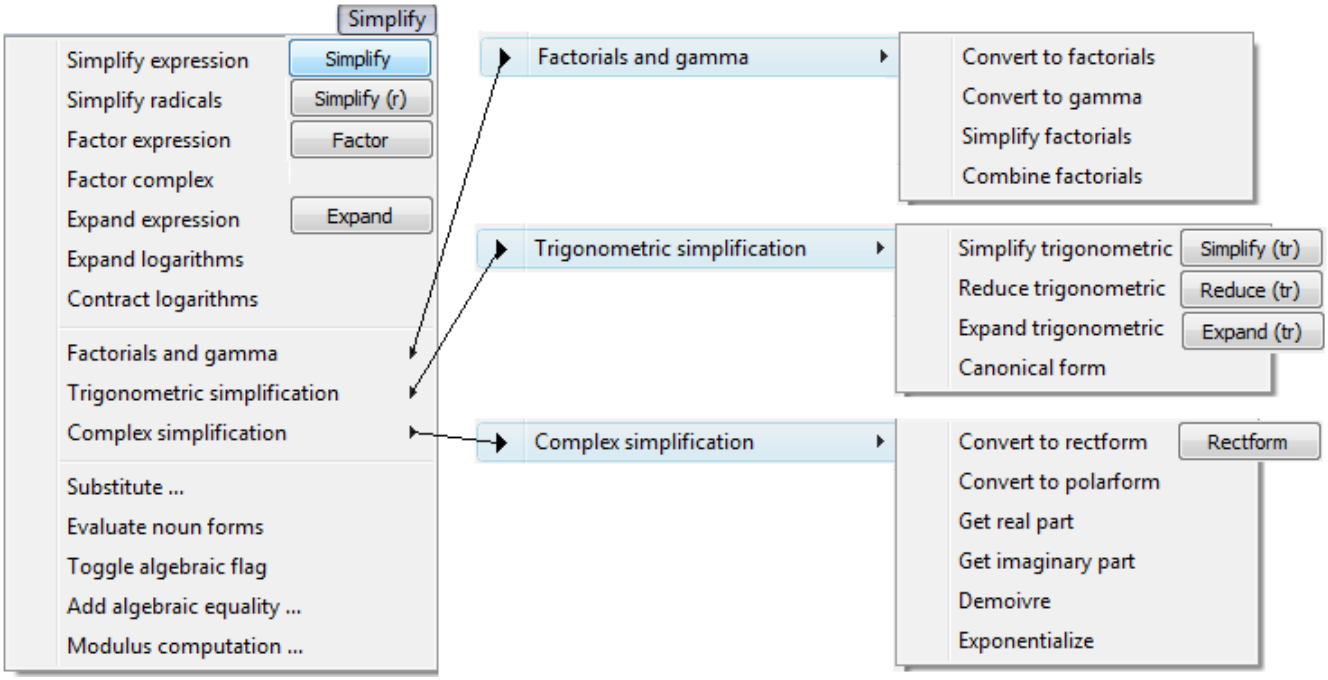


Figure 3.1. Simplify menu and sub-menus

Manipulating algebraic expressions

The following items in the *Simplify* menu can be used to simplify algebraic expressions such as polynomials and fractions:

- Simplify expression - equivalent to `ratsimp()`
- Simplify radicals - equivalent to `radcan()`
- Factor expression - equivalent to `factor()`
- Factor complex - equivalent to `gfactor()`
- Expand expression - equivalent to `expand()`
- Expand logarithms - equivalent to `%,logexpan=super`
- Contract logarithms - equivalent to `logcontract()`

To use these, and other menu items, you should have the expression to be manipulated ready in the *INPUT* line, and then invoke the menu item. The following examples illustrates the use of the *Simplify* menu items listed above.

Simplify expression Simplify

Enter the following expression in the *INPUT* line:

```
INPUT: x^2*y^2*z+a*x*y^2*z-3*x*y^2*z-3*a*y^2*z-2*x^2*y^2-2*a*x*y^2+6*x*y^2+6*a*y^2
```

and select the menu item *Simplify* > *Simplify expression* to get the following output:

```
(%i8)  ratsimp(x^2*y^2*z+a*x*y^2*z-3*x*y^2*z-3*a*y^2*z-2*x^2*y^2-2*a*x*y^2+6*x*y^2+6*a*y^2);
(%o8)  (x^2+(a-3)x-3a)y^2 z+(-2x^2+(6-2a)x+6a)y^2
```

The result of the *Simplify expression* menu item is the command *ratsimp* (*rat*ional *simp*lification), which, in this case, produced a factoring of the expression into two quadratic expressions in *x*, each accompanied by other terms, such as y^2z and y^2, respectively.

In the following example, we apply the *Simplify expression* menu item to a sum of fractions, to produce a single fraction:

```
(%i16)  ratsimp(x + y/(x+2) + z/(x^2+4));
```

$$(\%o16)\quad \frac{(x+2)z+(x^2+4)y+x^4+2x^3+4x^2+8x}{x^3+2x^2+4x+8}$$

The two results above suggest that any simplification in an algebraic expression involving *x* and other variables will expand or collect terms around the *x* variable. In the following two examples *x* is the only variable involved:

```
(%i21)  ratsimp(x*(x+2)+(x-3)*(x+4));
(%o21)  2x^2+3x-12

(%i22)  ratsimp(x + 2*x/(x-1)+3*x^2/(x^2+4));
```

$$(\%o22)\quad \frac{x^4+4x^3+x^2+4x}{x^3-x^2+4x-4}$$

Simplify radicals Simplify (r)

Simplifies expressions involving logarithms, exponentials, and radicals into a canonical form. The following examples illustrates applications of the menu item *Simplify* > *Simplify radicals*:

- Expression involving exponentials:

```
(%i53)  (%e^(-x)+%e^x)/(%e^(-x)-%e^x);
```

$$(\%o53)\quad \frac{\%e^{x}+\%e^{-x}}{\%e^{-x}-\%e^{x}}$$

```
(%i54)  radcan(%);
```

$$(\%o54)\quad -\frac{\%e^{2x}+1}{\%e^{2x}-1}$$

- Expression involving logarithms:

```
(%i49)  (log(x+x^2)-log(x))^n/(log(1+x))^(n/2);
```

$$(\%o49)\quad \frac{\left(\log(x^2+x)-\log(x)\right)^{n}}{\log(x+1)^{n/2}}$$

```
(%i50)  radcan(%);
```

$$(\%o50)\quad \log(x+1)^{n/2}$$

- Expression involving radicals:

```
(%i51)  (x+2*sqrt(x)+1)/(1+sqrt(x));
```

$$(\%o51)\quad \frac{x+2\sqrt{x}+1}{\sqrt{x}+1}$$

```
(%i52)  radcan(%);
```

$$(\%o52)\quad \sqrt{x}+1$$

Factor expression [Factor]

Factors out algebraic expressions, as illustrated below. First, we factor a couple of polynomials:

```
(%i63)  factor(x^4-9*x^3+9*x^2+85*x-150);
```

$$(\%o63)\quad (x-5)^{2}(x-2)(x+3)$$

```
(%i61)  factor(x^2*y^2*z+a*x*y^2*z-3*x*y^2*z-3*a*y^2*z-2*x^2*y^2-2*a*x*y^2+6*x*y^2+6*a*y^2);
```

$$(\%o61)\quad (x-3)(x+a)y^{2}(z-2)$$

The following example shows the factoring of a fraction:

```
(%i67)  factor((x^2+3*x-10)/(x^2+8*x+12));
                              (x - 2)(x + 5)
(%o67)                        --------------
                              (x + 2)(x + 6)
```

Factor can also be applied to integers to produce their factors:

```
(%i68)  factor(238574);
(%o68)  2 7 17041

(%i69)  factor(2342234);
(%o69)  2 1171117

(%i70)  factor(2^15+1);
                          2
(%o70)  3  11 331
```

Factor complex

This menu item is used to force polynomial factoring involving complex numbers. For example, applying *Factor* to the following polynomial produces no factoring:

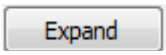
```
(%i71)  factor(x^2+4);
                    2
(%o71)  x  + 4
```

However, with *Factor complex* (*gfactor*) produces the following factors:

```
(%i72)  gfactor(x^2+4);
(%o72)  (x - 2 %i)(x + 2 %i)
```

Notice the difference results for the following two factorings;

```
(%i77)  factor(x^4+64);
                    2            2
(%o77)  (x  - 4 x + 8)(x  + 4 x + 8)

(%i78)  gfactor(x^4+64);
(%o78)  (x - 2 %i - 2)(x - 2 %i + 2)(x + 2 %i - 2)(x + 2 %i + 2)
```

Expand expression 

The menu item *Expand expression* can be applied to algebraic expressions and fractions:

```
(%i80)  expand(x*(x-1)*(x-2)^2);
                    4      3      2
(%o80)  x  - 5 x  + 8 x  - 4 x
```

```
(%i82)  expand((x^2+2*x+2)/(x^2-2*x+2));
```

$$(\%o82)\quad \frac{x^2}{x^2-2x+2}+\frac{2x}{x^2-2x+2}+\frac{2}{x^2-2x+2}$$

To recover the simple fractional form use *ratsimp:*

```
(%i83)  ratsimp(%);
```

$$(\%o83)\quad \frac{x^2+2x+2}{x^2-2x+2}$$

NOTE: To separate numerator and denominator of a fraction use functions *num* and *denom*, e.g.,

```
(%i84)  num((x^2+2*x+2)/(x^2-2*x+2));
```

$$(\%o84)\quad x^2+2x+2$$

```
(%i85)  denom((x^2+2*x+2)/(x^2-2*x+2));
```

$$(\%o85)\quad x^2-2x+2$$

Expand logarithm

The *Expand logarithm* menu item is a post-fix operator of the form *%, logexpan=super.* This command is used to expand a logarithm into sums or differences of logarithms, e.g.,

```
(%i93)  log(x*y), logexpand=super;
```

$$(\%o93)\quad \log(y)+\log(x)$$

```
(%i94)  log(y^k), logexpand=super;
```

$$(\%o94)\quad k\log(y)$$

```
(%i95)  log(y/x^k), logexpand=super;
```

$$(\%o95)\quad \log(y)-k\log(x)$$

Contract logarithm

The *Contract logarithm* menu item performs the inverse of the *Expand logarithm* function, e.g.,

```
(%i96)  logcontract(log(y)+log(x));
```

$$(\%o96)\quad \log(xy)$$

```
(%i98)  logcontract(2*log(y));
```

$$(\%o98)\quad \log(y^2)$$

```
(%i99)  logcontract(log(y)-3*log(x));
(%o99)  log(y/x^3)
```

Factorials, combinations, and permutations

The factorial, *n!*, of a positive integer number *n* is defined as the product:

$$n! = n\cdot(n-1)\cdot(n-2)\ \ldots\ 3\cdot2\cdot1$$

Thus, $2! = 2\times1 = 2$, $3! = 3\times2\times1 = 6$, etc. In Maxima, factorials are calculated by using the post-fix operator *!*. (Post-fix means the operator is placed after the number). Some examples are shown below:

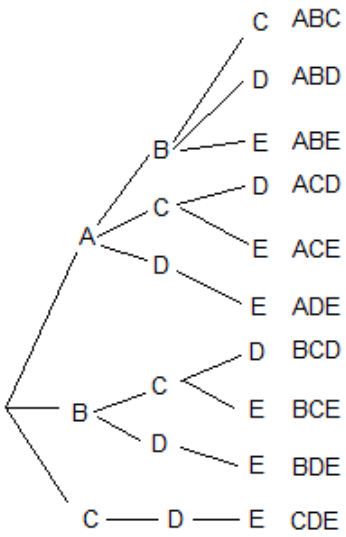
```
(%i1)  [2!,3!,4!,5!,6!,7!,8!,9!,10!];
(%o1)  [ 2 , 6 , 24 , 120 , 720 , 5040 , 40320 , 362880 , 3628800 ]
```

From the definition of factorial, it follows that:

$$n! = n\cdot(n-1)! = n\cdot(n-1)\cdot(n-2)! = n\cdot(n-1)\cdot(n-2)\cdot(n-3)!$$ and so on.

Combinations

Factorials are used, for example, in combinatorial analysis for calculating the number of *combinations* and *permutations* of *n* objects taken *r* at a time, with $n \geq r$. A combination is a selection of objects in which the order in which they are selected is not important. For example, if you have a collection of objects [A,B,C,D,E] and you take three at a time, then selecting, say, [A,B,C], [A,C,B], [B,A,C], etc., corresponds to the same combination of elements since the order is not important. The tree diagram shown below illustrates all 10 combinations of the 5 elements [A,B,C,D,E] taken three at a time.



The number of combinations of *n* elements taken *r* at a time is calculated using[1]:

$$C(n,r) = {}_nC_r = \binom{n}{r} = \frac{n!}{(n-r)!\,r!} \quad .$$

Also,

$$C(n,r) = {}_nC_r = \binom{n}{r} = \frac{n\cdot(n-1)\cdot(n-2)\ldots(n-r+1)\cdot(n-r)!}{(n-r)!\,r!} = \frac{n\cdot(n-1)\cdot(n-2)\ldots(n-r+1)}{r!}$$

Thus, if *n* = *5* and *r* = *3*, as in the case presented above, we find that *C(5,3)* is equal to:

```
(%i4)  5!/((5-3)!*3!);
(%o4)  10
```

Alternatively,

```
(%i10)  5*4*(5-3+1)/3!;
(%o10)  10
```

Maxima includes function *combination*(*n*,*r*) to calculate the number of combinations of *n* elements taken *r* at a time. Using the online *help* command (*??*) we find the following information about function *combination*:

```
(%i7)  ?? combination;
 -- Function: combination (<n>, <r>)
     Returns the number of combinations of <n> objects taken <r> at a
     time.
     To use this function write first `load(functs)'.
(%o7)  true
```

Proceeding according to the information above, we first load the *functs* package and then show some calculations of the number of combinations of 5 elements taken 1, 2, 3, and 4 at a time, respectively:

```
(%i8)  load(functs);
(%o8)  C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/simplification/functs.mac

(%i9)  [combination(5,1), combination(5,2), combination(5,3), combination(5,4)];
(%o9)  [ 5 , 10 , 10 , 5 ]
```

1The notation $\binom{n}{r}$ is also referred to as the *binomial* coefficient, since it represents the *r-th* coefficient in the expansion of the binomial $(x+y)^n$.

Since the order of selection is not important when forming a combination of objects, then the number of combinations of *n* elements taken *n* at a time is *1*. Also, the expression for *C(n,n)* is given by

$$C(n,n)=\binom{n}{n}=\frac{n!}{(n-n)!\,n!}=\frac{n!}{0!\,n!}=1 \quad ,$$

which leds to the interesting conclusion that *0! = 1*.

Permutations

A permutation is a selection of objects such that the order in which they are selected is important. Thus, if you have 5 objects, say, [A,B,C,D,E], and you randomly select three of them, say, [A,C,E], then [A,C,E], [A,E,C], [C,A,E], [E,C,A], etc., are all permutations of those three elements. You can actually produce the permutations of [A,C,E] using the function *permutations* in *Maxima*:

```
(%i17) permutations([A,C,E]);
(%o17) {[A,C,E],[A,E,C],[C,A,E],[C,E,A],[E,A,C],[E,C,A]}
```

The number of permutations of *n* elements taken *r* at a time is calculated using[2]:

$$P(n,r) = {}_nP_r = \frac{n!}{(n-r)!} \quad .$$

Also,

$$P(n,r) = {}_nP_r = \frac{n\cdot(n-1)\cdot(n-2)\ldots(n-r+1)\cdot(n-r)!}{(n-r)!}=n\cdot(n-1)\cdot(n-2)\ldots(n-r+1)$$

Thus, if *n = 5* and *r = 3*, as in the case presented above, we find that *P(5,3)* is equal to:

```
(%i11) 5!/(5-3)!;
(%o11) 60
```

Alternatively,

```
(%i12) 5*4*(5-3+1);
(%o12) 60
```

Maxima includes functions *permutation*(*n*,*r*) to calculate the number of permutations of *n* elements taken *r* at a time. Using the online *help* command (*??*) we find the following information about function *permutation*. Notice that *Maxima* provides three different online help entries related to the word *permutation*, so we have to choose, by entering the proper number, which one of the three definitions we want to explore further. Choose *0* to obtain:

2The notation $\binom{n}{r}$ is also referred to as the *binomial* coefficient, since it represents the *r-th* coefficient in the expansion of the binomial $(x+y)^n$.

```
(%i13)  ?? permutation;
 0: permutation  (Package functs)
 1: permutations  (Functions and Variables for Sets)
 2: random_permutation  (Functions and Variables for Sets)
Enter space-separated numbers, `all' or `none':  0;
 -- Function: permutation (<n>, <r>)
     Returns the number of permutations of <r> objects selected from a
     set of <n> objects.
     To use this function write first `load(functs)'.
```

Since we have already loaded the *functs* package (when dealing with combinations), we proceed to show some calculations of the number of permutations of 5 elements taken 1, 2, 3, and 4 at a time, respectively:

```
(%i14)  [permutation(5,1), permutation(5,2), permutation(5,3), permutation(5,4)];
(%o14)  [ 5 , 20 , 60 , 120 ]
```

The Gamma (Γ) function
The Gamma function is defined by the infinite integral:

$$\Gamma(z)=\int_0^{\infty} t^{z-1}\cdot e^{-t}\,dt$$

In *Maxima*, this function is calculated using *gamma()*, e.g.,

```
(%i27)  [gamma(x), gamma(3), gamma(-2.5), gamma(12.5)];
(%o27)  [ Γ(x) , 2 , - 0.94530872048294 , 1.3684336546556553 10^+8 ]
```

The Gamma function for zero and negative integer numbers is not defined, e.g.,

```
(%i28)  gamma(-2);
gamma(-2) is undefined
```

The Gamma function is related to factorials as follows: $\Gamma(n)=(n-1)!$. The Gamma function allows one to generalize the factorial operator to non-integer numbers, if we use:

$$x!=\Gamma(x+1)$$

Some examples using *Maxima* are shown below:

```
(%i30)  [2.5!, 3.5!, (-1.2)!, 1.2!];
(%o30)  [ 3.323350970447843 , 11.63172839656745 , - 5.821148568626517 , 1.101802490879713 ]
```

The beta (β) function
The beta function is defined in terms of the Gamma function:

$$\beta(x,y)=\frac{\Gamma(x)\cdot\Gamma(y)}{\Gamma(x+y)} \quad .$$

The beta function can also be related to factorials as:

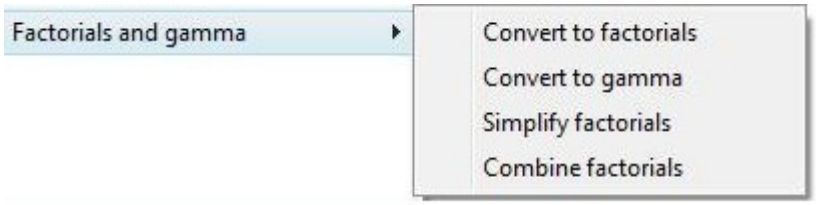
$$\beta(x,y)=\frac{(x-1)!\cdot(y-1)!}{(x+y-1)!}$$

Using *Maxima* the beta function is evaluated as in the following examples:

```
(%i31)  [beta(2,3), beta(3,2), beta(1.2,3.5), beta(-1.2,2.5)];
```

$$(\%o31)\quad \left[\frac{1}{12},\frac{1}{12},0.1977393293741,7.185274432649909\right]$$

Manipulating factorials, Gamma, and beta functions
The *Symplify* menu in the *wxMaxima* interface includes the following items for manipulating factorials and relating them to the Gamma and beta functions:



Convert to factorials
This menu item can be used to convert expressions involving the Gamma and beta functions into factorial expressions, e.g.,

```
(%i45)  makefact(gamma(x));
```

$$(\%o45)\quad (x-1)!$$

```
(%i46)  makefact(gamma(x+y)/x!);
```

$$(\%o46)\quad \frac{(y+x-1)!}{x!}$$

```
(%i47)  makefact(beta(x,y));
```

$$(\%o47)\quad \frac{(x-1)!(y-1)!}{(y+x-1)!}$$

Convert to gamma

This menu item is used to convert factorial expressions into Gamma function expressions, e.g.,

```
(%i48)  makegamma((x+1)!/(x+y)!);
```

$$(\%o48)\quad \frac{\Gamma(x+2)}{\Gamma(y+x+1)}$$

```
(%i49)  makegamma(x!*y!*z!);
```

$$(\%o49)\quad \Gamma(x+1)\Gamma(y+1)\Gamma(z+1)$$

Simplify factorials

This menu item can be used to simplify selected factorial expressions such as:

```
(%i55)  minfactorial((x+5)!/x!);
```

$$(\%o55)\quad (x+1)(x+2)(x+3)(x+4)(x+5)$$

```
(%i56)  minfactorial((x+5)!*(x+2)!/(x!)^2);
```

$$(\%o56)\quad (x+1)^2(x+2)^2(x+3)(x+4)(x+5)$$

```
(%i57)  minfactorial((n+1)!/(n+3)!);
```

$$(\%o57)\quad \frac{1}{(n+2)(n+3)}$$

Combine factorials

This menu item is used to combine factorial expressions such as:

```
(%i78)  (n+1)*n!^2;
```

$$(\%o78)\quad (n+1)n!^2$$

```
(%i79)  factcomb(%);
```

$$(\%o79)\quad nn!^2+n!^2$$

```
(%i80)  (n+2)*n!^3;
```

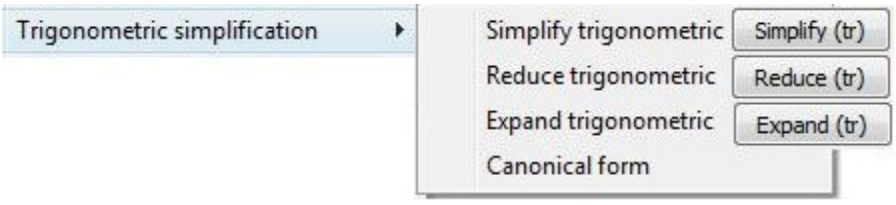
$$(\%o80)\quad (n+2)n!^3$$

```
(%i81)  factcomb(%);
```

$$(\%o81)\quad nn!^3+2n!^3$$

Manipulation of trigonometric expressions

The sub-menu *Simplify* > *Trigonometric simplification* offers the following items:



Simplify trigonometric Simplify (tr)

This menu item utilizes the trigonometric identity $sin^2(x) + cos^2(x) = 1$ and the hyperbolic identity $cosh^2(x) - sinh^2(x) = 1$ to simplify expressions involving *tan*, *sec*, *tanh*, *sech*, etc., to expressions involving only *sin*, *cos*, *sinh*, and *cosh*. To see examples of this command, load the demo file *trgsmp.dem*, i.e.,

```
(%i102) demo("trgsmp.dem");
batching #pC:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/demo/trgsmp.dem
 At the _ prompt, type ';' followed by enter to get next demo
```

$$\text{(\%i103)}\quad \tan(x)\sec(x)^2+\frac{\left(1-\sin(x)^2\right)\cos(x)}{\cos(x)^2}$$

$$\text{(\%o103)}\quad \sec(x)^2\tan(x)+\frac{1-\sin(x)^2}{\cos(x)}$$

```
_ ;
(%i104) trigsimp(% )
```

$$\text{(\%o104)}\quad \frac{\sin(x)+\cos(x)^4}{\cos(x)^3}$$

The way this *demo* file is put together, as illustrated in the example above, is to show a trigonometric expression, and then apply the function *trigsimp* to the given expression to see the corresponding simplified expression. Press [ENTER] to see the rest of the demo examples.

Reduce trigonometric Reduce (tr)

This menu item combines products and powers of trigonometric and hyperbolic sine and cosine into sine and cosine of multiples of the angle, trying to eliminate *sin* and *cos* from denominators in the case of fractions.

```
(%i117) trigreduce(sin(x)^2);
```

$$\text{(\%o117)}\quad \frac{1-\cos(2x)}{2}$$

```
(%i118) trigreduce(sin(x)*cos(x));
```

$$\text{(\%o118)}\quad \frac{\sin(2x)}{2}$$

The following example shows that function *trigreduce* acts term by term:

```
(%i121) trigreduce(sin(theta)^2 + sin(theta)*cos(theta) + cos(theta)^2);
```

$$(\%o121)\quad \frac{\sin(2\,\theta)}{2}+\frac{\cos(2\,\theta)+1}{2}+\frac{1-\cos(2\,\theta)}{2}$$

At this point, we can use the menu item *Simplify* > *Simplify expression* (ratsimp) to simplify the fractional sum to:

```
(%i122) ratsimp(%);
```

$$(\%o122)\quad \frac{\sin(2\,\theta)+2}{2}$$

Other types of reductions achievable with *trigreduce* are illustrated below:

```
(%i123) trigreduce(sin(theta + %pi/2));
(%o123) cos(θ)

(%i124) trigreduce(cos(theta + %pi/2));
(%o124) - sin(θ)
```

Expand trigonometric Expand (tr)

This menu item allows expanding expressions such as *sin(x+y)*, *sin(2*x)*, etc. For example:

```
(%i132) trigexpand(sin(x+y)+cos(x+y));
(%o132) - sin(x)sin(y)+cos(x)sin(y)+sin(x)cos(y)+cos(x)cos(y)

(%i133) trigexpand(sin(2*x)+cos(3*x));
```

$$(\%o133)\quad -3\cos(x)\sin(x)^2+2\cos(x)\sin(x)+\cos(x)^3$$

One type of expansion that requires redefining an option in *Maxima* is the expansion of half-angle expressions. By default, *Maxima* does not expand trigonometric functions of half angles, e.g.,

```
(%i140) trigexpand(sin(theta/2));
```

$$(\%o140)\quad \sin\left(\frac{\theta}{2}\right)$$

This is so because, by default, the *halfangles* option is set to *false*:

```
(%i141) halfangles;
(%o141) false
```

Try setting the option *halfangles* to *true* and repeating the expansion:

```
(%i142) halfangles : true $ trigexpand(sin(theta/2));
```

$$(\%o143)\quad \frac{\sqrt{1-\cos(\theta)}}{\sqrt{2}}$$

Canonical form

This menu item is used to produce a simplification of trigonometric expressions into a quasi-linear form, i.e., avoiding powers of trigonometric functions as much as possible. Some examples are shown below.

```
(%i150) (sin(theta)+cos(theta))^2;
```

$$(\%o150)\quad (\sin(\theta)+\cos(\theta))^2$$ ← **original expression**

```
(%i151) expand(%);
```

$$(\%o151)\quad \sin(\theta)^2+2\cos(\theta)\sin(\theta)+\cos(\theta)^2$$ ← **expand algebraically**

```
(%i152) trigrat(%);
```

$$(\%o152)\quad \sin(2\theta)+1$$ ← **canonical form**

A second example:

```
(%i153) sin(theta)^3+sin(theta)^2+(sin(theta)+1)^2;
```

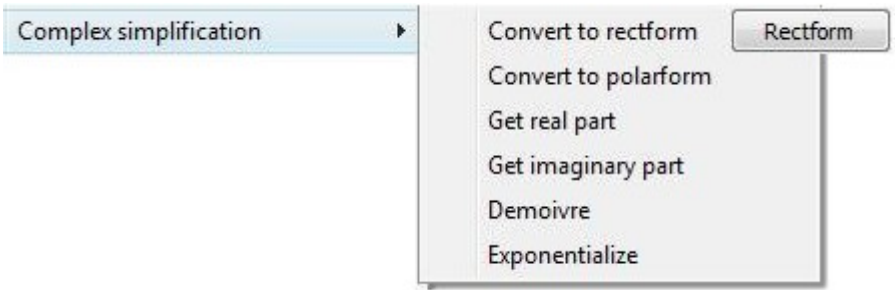
$$(\%o153)\quad (\sin(\theta)+1)^2+\sin(\theta)^3+\sin(\theta)^2$$

```
(%i154) trigrat(%);
```

$$(\%o154)\quad -\frac{\sin(3\theta)+4\cos(2\theta)-11\sin(\theta)-8}{4}$$

Manipulating complex numbers and expressions

The *Simplify* menu offers the following sub-menu for the manipulation of complex numbers and expressions:



To understand the functions listed above, we first provide a few definitions related to complex numbers: A complex variable is a variable of the form

$$z = x + iy,$$

with $i^2 = -1$, where x and y are real numbers.

The *real part* of z is

$$x = Re(z),$$

while the *imaginary part* of z is

$$y = Im(z).$$

Graphical representation - A complex number can be represented as a point in the *Argand's* diagram, a Cartesian coordinate system where the ordinate represents imaginary numbers. This representation is shown in the following figure:

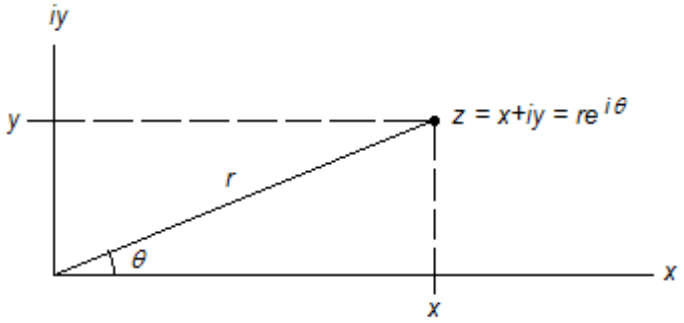


Figure 3.2. Rectangular and polar forms of a complex number.

The figure shows two different representations of the complex variable z, its Cartesian or *rectangular form* ($x+iy$) and its *polar form* ($re^{i\theta}$). The radial distance

$$r = |z| = \sqrt{x^2+y^2}$$

is referred to as the *modulus* of the complex number, while

$$\theta = Arg(z) = \tan^{-1}\left(\frac{y}{x}\right)$$

is referred to as the *argument* of the complex number. The real and imaginary parts, x and y, of the rectangular form of the complex number can be calculated in terms of the modulus and argument, r and θ, by using:

$$x = r\cos(\theta),\ y = r\sin(\theta).$$

The polar form representation uses the De Moivre formula for complex numbers, namely,

$$e^{i\theta} = cos(\theta) + i\, sin(\theta) \quad .$$

As a curious note, if one replaces $\theta = \pi$ into this expression, the result is a combination of some of the most famous numbers in mathematics:

$$e^{i\pi} = -1$$

This expression involves the numbers:

- *e* the base of the natural logarithms
- i the unit imaginary number
- π the ratio of the length of the circumference to its diameter
- -1 the unit negative number

Using the Euler formula, the equivalence of the rectangular and the polar representations of a complex number becomes obvious:

$$z = r\, e^{i\theta} = r(cos(\theta) + i\, sin(\theta)) = r\, cos(\theta) + i\, r\, sin(\theta) = x + i\, y \quad .$$

Next, we present some of the complex variable operations available in *Maxima* using the items in the *Symplify* >*Complex simplification* sub-menu.

Convert to rectform Rectform

This menu item converts a complex expression into its rectangular form. This command can be used to show the results of complex number operations, as illustrated in these examples. First we define two complex numbers z1 and z2 and attempt an addition:

```
(%i1)  z1 : x1 + y1*%i $ z2 : x2 + y2*%i $
(%i3)  z1+z2;
(%o3)  %i y2 + %i y1 + x2 + x1
```

Using the *Convert to rectform* (*rectform*) command we get:

```
(%i4)  rectform(%);
(%o4)  %i (y2 + y1 ) + x2 + x1
```

The following examples show the command *rectform* applied to subtraction, multiplication, division, and powers of complex numbers:

```
(%i7)  rectform(z1*z2);
(%o7)  %i (x1 y2 + x2 y1 ) - y1 y2 + x1 x2
```

```
(%i8) rectform(z1/z2);
```

$$(\%o8)\quad \frac{y1\ y2 + x1\ x2}{y2^2 + x2^2} + \frac{\%i\,(x2\ y1 - x1\ y2)}{y2^2 + x2^2}$$

```
(%i12) rectform(z1^2);
```

$$(\%o12)\quad -y1^2 + 2\,\%i\ x1\ y1 + x1^2$$

Using actual numbers:

```
(%i9) rectform((2-2*%i)*(5-7*%i));
```

$$(\%o9)\quad -24\,\%i - 4$$

```
(%i10) rectform((4+5*%i)/(-2+7*%i));
```

$$(\%o10)\quad \frac{27}{53} - \frac{38\,\%i}{53}$$

```
(%i11) rectform((3.5+2.6*i)^2);
```

$$(\%o11)\quad (2.6\,i + 3.5)^2$$

Convert to polarform

This menu item converts a complex expression into its rectangular form. This command can be used to show the results of complex number operations, as illustrated in these examples. First we define two complex numbers *z1* and *z2* as follows:

```
(%i25) z1:r1*exp(%i*theta[1]) ; z2:r2*exp(%i*theta[2]) ;
```

$$(\%o25)\quad \%e^{\%i\,\theta_1}\,r1$$

$$(\%o26)\quad \%e^{\%i\,\theta_2}\,r2$$

In this case we use sub-indices to define the variables *theta[1]* and *theta[2]*. The sum of the two complex numbers is a long expression in its polar form:

```
(%i30) polarform(z1+z2);
```

$$(\%o30)\quad \sqrt{\left(\sin(\theta_2)\,r2 + \sin(\theta_1)\,r1\right)^2 + \left(\cos(\theta_2)\,r2 + \cos(\theta_1)\,r1\right)^2}\ \%e^{\%i\ \mathrm{atan2}\left(\sin(\theta_2)\,r2 + \sin(\theta_1)\,r1,\ \cos(\theta_2)\,r2 + \cos(\theta_1)\,r1\right)}$$

Multiplications, divisions, and powers will show more manageable expressions, although the user needs to reply to additional requests for information from *Maxima*:

```
(%i31) polarform(z1*z2);
Is  r1  positive, negative, or zero? positive;
Is  r2  positive, negative, or zero? positive;
                          %i(θ2 + θ1)
(%o31)                   %e           r1 r2

(%i32) polarform(z1/z2);
Is  r1  positive, negative, or zero? positive;
Is  r2  positive, negative, or zero? positive;
                          %i(θ1 - θ2)
                         %e           r1
(%o32)                   ------------
                             r2

(%i33) polarform(z1^2);
Is  r1  positive, negative, or zero? positive;
                          2 %i θ1
(%o33)                   %e       r1^2
```

The following examples use actual numbers:

```
(%i35) polarform((2-2*%i)*(5-7*%i));
                                  %i(atan(6) - π)
(%o35)                 4 sqrt(37) %e

(%i36) polarform((5-7*%i)/(5+2*%i));
                                    - %i atan(45/11)
                         sqrt(74) %e
(%o36)                   ------------------------
                              sqrt(29)

(%i37) polarform((5+3*%i));
                                  %i atan(3/5)
(%o37)                 sqrt(34) %e
```

Get real part

This menu item extracts the real part of a complex variable or expression:

```
(%i38) realpart(5+6*%i);
(%o38) 5

(%i39) realpart((5-7*%i)/(5+2*%i));
(%o39) 11/29

(%i40) realpart((x+y*%i)^2);
(%o40) x^2 - y^2
```

Get imaginary part

This menu item extracts the imaginary part of a complex variable or expression

```
(%i41) imagpart(5+6*%i);
(%o41) 6

(%i42) imagpart((5-7*%i)/(5+2*%i));
(%o42) -45/29

(%i43) imagpart((x+y*%i)^2);
(%o43) 2 x y
```

Demoivre

The simplest application of this menu item (*demoivre*) is to implement De Moivre's formula, $e^{i\theta}=cos(\theta)+i\,sin(\theta)$, i.e.,

```
(%i47) demoivre(exp(%i*theta));
(%o47) %i sin(θ)+ cos(θ)
```

Other examples are shown below:

```
(%i48) demoivre(5*exp(-%pi/6*%i));
(%o48) 5 (sqrt(3)/2 - %i/2)

(%i49) demoivre((5+3*%i)*exp(5+%i));
(%o49) %e^5 (3 %i + 5)(%i sin(1)+ cos(1))
```

Exponentialize

This menu item is the inverse of the *Demoivre* menu item, producing the exponential form of complex expressions involving trigonometric and hyperbolic functions, e.g.,

```
(%i68) exponentialize(sin(x+y*%i)); rectform(%);
                                 %i (%e^(%i (%i y + x)) - %e^(- %i (%i y + x)))
(%o68)                         - ----------------------------------------------
                                                    2
                                 - sin(x) %e^y - sin(x) %e^(- y)     %i (cos(x) %e^(- y) - cos(x) %e^y)
(%o69)                           -------------------------------  - ----------------------------------
                                              2                                   2

(%i70) exponentialize(cosh(x+y*%i)); rectform(%);
                                 %e^(%i y + x) + %e^(- %i y - x)
(%o70)                           -------------------------------
                                              2
                                 %i (%e^x sin(y) - %e^(- x) sin(y))     %e^x cos(y) + %e^(- x) cos(y)
(%o71)                           ----------------------------------  + -----------------------------
                                              2                                   2
```

These two expressions, for example, show the definitions of the functions *sin(z)* and *cosh(z)* in term of the real and imaginary parts, *x* and *y*.

More functions for complex numbers

Maxima includes the following functions for manipulation of complex variables or expressions:

- cabs - complex absolute value (modulus)
- carg - complex argument
- conjugate - complex conjugate
- residue - residue in complex plane

While the modulus (cabs) and argument (carg) have been defined before, in this section we include definitions related to the functions *conjugate* and *residue* shown above.

The *complex conjugate* of the complex number $z = x + iy = re^{i\theta}$ is the reflection of z about the x-axis, i.e.,

$$\bar{z} = x - iy = re^{i\theta}.$$

The product of a complex numbers and its conjugate is the square of its modulus:

$$z \cdot \bar{z} = x2 + y2 = |z|^2 = r^2.$$

The following examples cover applications of the functions *cabs*, *carg*, and *conjugate* for complex expressions:

```
(%i1) [cabs(4+3*%i),cabs(5*exp(-%i*%pi/6)),cabs(x+%i*y),cabs(sin(x+%i*y)^2)];
```

$$(\%o1)\quad \left[5,\ 5,\ \sqrt{y^2+x^2},\ \sqrt{\left(\sin(x)^2\cosh(y)^2-\cos(x)^2\sinh(y)^2\right)^2+4\cos(x)^2\sin(x)^2\cosh(y)^2\sinh(y)^2}\right]$$

```
(%i2) [carg(4+3*%i),carg(5*exp(-%i*%pi/6)),carg(x+%i*y),carg(sin(x+%i*y)^2)];
```

$$(\%o2)\quad \left[\operatorname{atan}\left(\frac{3}{4}\right),\ -\frac{\pi}{6},\ \operatorname{atan2}(y,x),\ 2\operatorname{atan2}(\cos(x)\sinh(y),\sin(x)\cosh(y))\right]$$

```
(%i3) [conjugate(4+3*%i),conjugate(5*exp(-%i*%pi/6)),conjugate(x+%i*y),conjugate(sin(x+%i*y)^2)];
```

$$(\%o3)\quad \left[4-3\,\%i,\ 5\left(\frac{\%i}{2}+\frac{\sqrt{3}}{2}\right),\ x-\%i\,y,\ \sin(\%i\,y-x)^2\right]$$

A Laurent series expansion for a complex expression requires a point of expansion z_0. The Laurent series resembles a Taylor series expansion but it includes both positive and negative powers. The *residue* of a complex expression is the coefficient of the power (-1) term in the expansion of the expression in a Laurent series.

For additional information on Laurent series check out the following online links:

- Wikipedia link: http://en.wikipedia.org/wiki/Laurent_series
- Wolfram Mathworld link: http://mathworld.wolfram.com/LaurentSeries.html

Function *residue* requires the complex expression being expanded, the complex variable, and the point of expansion, and returns the residue in the complex plane for the expression. Examples:

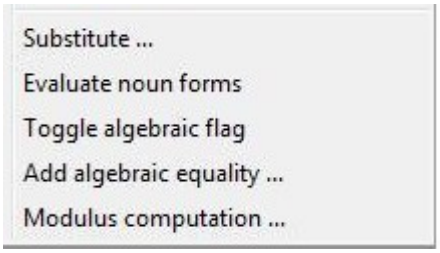
```
(%i30) residue(sin(a*z)/z^4,z,0);
```

$$(\%o30)\quad -\frac{a^3}{6}$$

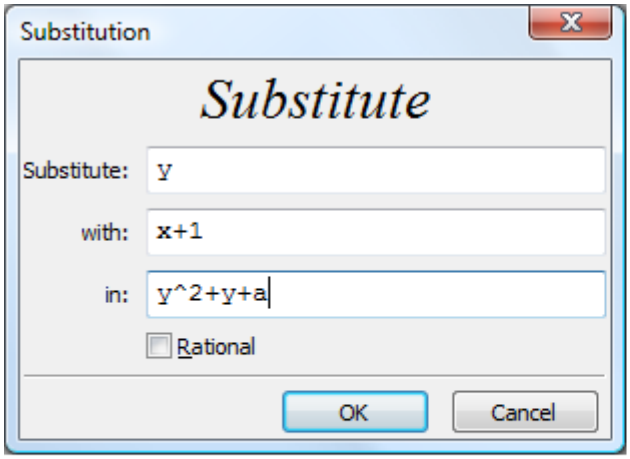
```
(%i31) residue (z/(z**2+alpha**2), z, alpha*%i);
```

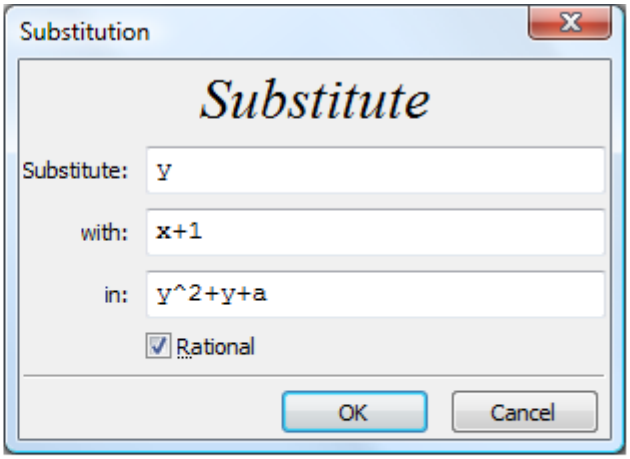
$$(\%o31)\quad \frac{1}{2}$$

Substitution and other menu items for expression manipulation
The last set of functions in the *Simplify* menu include the following items:



Substitute...
This menu item activates a dialogue form that allows the substitution of a variable into an expression. For example, the following two examples show the dialogue form and the resulting entry into the *wxMaxima* window:



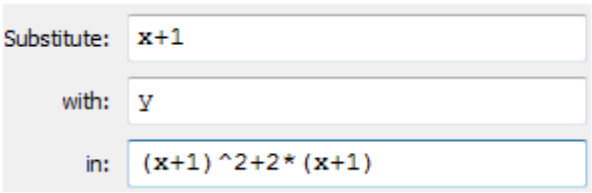



```
(%i35) subst(x+1, y, y^2+y+a);
(%o35) (x+1)^2+x+a+1
```

```
(%i36) ratsubst(x+1, y, y^2+y+a);
(%o36) x^2+3*x+a+2
```

Checking the [ ] *Rational* box in the the dialogue form implements function *ratsubst* (*rat*ional *subst*itution) rather than *subst* alone. The difference, in this example, is that a rational simplification is applied to the resulting expression.

Here is another example:




```
(%i40) ratsubst(y, x+1, (x+1)^2+2*(x+1));
(%o40) y^2+2*y
```

An alternative way to use function *subst* is to create a list of substitutions using equal signs, as illustrated in these examples:

```
(%i20)  subst([x=a,y=b],x^2+y^2);
(%o20)  b^2+a^2
(%i21)  subst([C=1.49,R=0.75,S=0.0001],C*sqrt(R*S));
(%o21)  0.012903778516388
(%i23)  subst([x^2=y,z^2=r],x^2+z^2);
(%o23)  y+r
```

Evaluate noun forms

Noun forms, as opposite to *verb forms*, are executable expressions in *Maxima* that remain unevaluated. The item menu *Evaluate noun forms* allows the evaluation of those unevaluated expressions. To produce an unevaluated expression typically you precede it with an apostrophe. Some examples of unevaluated expressions, and their result after the *Evaluate noun forms* menu item is activated, are shown below:

```
(%i41)  'diff(x^3+3,x);
(%o41)  d/dx (x^3+3)
(%i42)  ev(%, nouns);
(%o42)  3 x^2
```

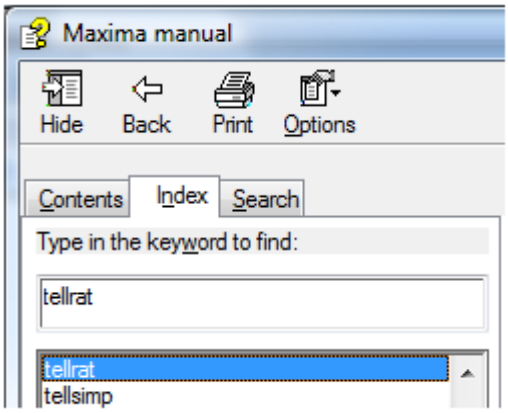
```
(%i43)  'integrate(x/(1+x^2),x);
(%o43)  ∫ x/(x^2+1) dx
(%i44)  ev(%, nouns);
(%o44)  log(x^2+1)/2
```

Toggle algebraic flag

The *Maxima Manual* (select it using *Help* > *Maxima help*) has a simple entry for the *algebraic* flag. The *Manual* indicates that the default value of the *algebraic* flag is *false*, and that it must be set to true (using, *algebraic : true*) "in order for the simplification of algebraic integers to take effect." By using this menu item you can toggle the *algebraic* flag between *true* and *false*. To find out about the current setting use (default set):

```
(%i45)  algebraic;
(%o45)  false
```

Add algebraic equality...
This menu item activates the *tellrat* function to produce algebraic equality between expressions. Refer to the *Maxima Manual* for the operation of this function. Activate the *Maxima Manual* using *Help* > *Maxima help*, click on the *Index* tab, and type *tellrat*:



Modulus computation...
The menu item *Modulus computation...* allows the user to set the modulus for modular arithmetic calculations. The default value is *false*. The user can set the *modulus* value to an integer value, say, 2, 3, etc. Typically the modulus is a positive prime number. The following references address the issue of modulus arithmetic:

- Wikipedia link: http://en.wikipedia.org/wiki/Modular_arithmetic
- Wolfram Mathworld link: http://mathworld.wolfram.com/ModularArithmetic.html

Some examples of modular arithmetic calculations are shown below. First, examples with *modulus = 3*:

```
(%i9)  modulus : 3;
(%o9)  3

(%i10)  [rat(1+1),rat(1+2),rat(1+3),rat(1+4),rat(1+5),rat(1+6),rat(1+7),rat(1+8)];
(%o10)  [ - 1 , 0 , 1 , - 1 , 0 , 1 , - 1 , 0 ]

(%i11)  [rat(1*1),rat(1*2),rat(1*3),rat(2*2),rat(2*3),rat(3*3)];
(%o11)  [ 1 , - 1 , 0 , 1 , 0 , 0 ]
```

The second set of examples correspond to *modulus* = *5*:

```
(%i12) modulus : 5;
(%o12) 5

(%i13) [rat(1+0),rat(1+1),rat(1+2),rat(1+3),rat(1+4),rat(1+5),rat(1+6)];
(%o13) [ 1 , 2 , - 2 , - 1 , 0 , 1 , 2 ]

(%i14) [rat(2*2),rat(2*3),rat(3*3),rat(3*4),rat(3*5),rat(4*5),rat(5*5)];
(%o14) [ - 1 , 1 , - 1 , 2 , 0 , 0 , 0 ]
```

Simple operations with polynomials
In this section we provide examples of functions that apply to polynomials.

coeff
The *coeff* function, `coeff(p,x,n)`, is used to extract the coefficient of the variable *x* of order *n* in the polynomial *p*:

```
(%i36) p : x^4+2*x^2-3*x+5 $

(%i37) [coeff(p,x,4),coeff(p,x,3),coeff(p,x,2),coeff(p,x),coeff(p,x,0)];
(%o37) [ 1 , 0 , 2 , - 3 , 5 ]
```

divide (also *Calculus* > *Divide polynomials...*)
The *divide* function, `divide(p,q)`, produces the quotient and residual of the polynomial division *p*/*q*:

```
(%i59) divide(x^3+2*x^2-8*x,x-2);
                                  2
(%o59) [ x  - x , 0 ]

(%i60) divide(x^3+2*x^2,x-2);
                                  2
(%o60) [ x  - x - 2 , 1 ]

(%i61) divide(y^2+5*y+3,y^2);
(%o61) [ 1 , - 2 ]
```

quotient, and remainder

The *quotient* and *remainder* functions, `quotient(p,q)` and `remainder(p,q)`, produce, respectively, the quotient and residual of the polynomial division *p*/*q*:

```
(%i76) divide(x^3+2*x^2-8,x-2);
(%o76) [x^2-x-2,-2]

(%i77) quotient(x^3+2*x^2-8,x-2);
(%o77) x^2-x-2

(%i78) remainder(x^3+2*x^2-8,x-2);
(%o78) -2
```

ratdiff

The *ratdiff* (*rat*ional *diff*erentiation) function, `ratdiff(p,x)`, produces the derivative of a rational function *p* with respect to variable *x*:

```
(%i80) ratdiff((x+y)^3+(x-2)^2+x+1,x);
(%o80) -2*y^2+x*(y+2)-2*x^2+2

(%i81) ratdiff(x^2+3,x);
(%o81) 2*x
```

allroots (*Equations* > *Roots of polynomial*)

The *allroots* function, `allroots(p)` or `allroots(p,x)`, calculates all the roots *x* of a polynomial *p*:

```
p:x^4+3*x^3+5*x^2-x+5;
x^4+3*x^3+5*x^2-x+5

allroots(%);
[x=0.0,x=0.66545695115282*%i-0.1027847152003,x=-0.66545695115282*%i-0.1027847152003,x=2.205569430400598]
```

realroots (*Equations* > *Roots of polynomial (real)*)

The *allroots* function, `realroots(p)` or `realroots(p,x)`, calculates the real roots *x* of a polynomial *p*:

```
(%i99) p:x^4+3*x^3+5*x^2-x+5;
(%o99) x^4+3*x^3+5*x^2-x+5

(%i100) realroots(%);
(%o100) [x=2.205569416284561,x=0]
```

gcd (*Calculus* > *Greatest common divisor...*)
The *gcd* function calculates the *g*reatest *c*ommon *d*ivisor for two or more polynomials, e.g.,

```
p1 : x^4-4*x^3+5*x^2-2*x;
```
$$x^4 - 4x^3 + 5x^2 - 2x$$

```
p2 : x^3-6*x^2+11*x-6;
```
$$x^3 - 6x^2 + 11x - 6$$

```
gcd(p1, p2);
```
$$x^2 - 3x + 2$$

Let's add one more polynomial to the *gcd* function:

```
p3 : x^2*y^2-3*x*y^2+2*y^2-2*x^3*y+6*x^2*y-4*x*y+x^4-3*x^3+2*x^2;
```
$$x^2y^2 - 3xy^2 + 2y^2 - 2x^3y + 6x^2y - 4xy + x^4 - 3x^3 + 2x^2$$

```
gcd(p1,p2,p3);
```
$$x^2 - 3x + 2$$

Function *gcd* can also be applied to integers:

```
[gcd(204,51),gcd(20,120),gcd(125,25)];
```
$$[51, 20, 25]$$

horner
Function *horner* produces the expression corresponding to the Horner's rule for evaluating polynomials. The following example shows the Horner's rule for a list of two polynomials:

```
[p1,p2];
```
$$[x^4 - 4x^3 + 5x^2 - 2x, x^3 - 6x^2 + 11x - 6]$$

```
[horner(p1),horner(p2)];
```
$$[x(x((x - 4)x + 5) - 2), x((x - 6)x + 11) - 6]$$

lcm (*Calculus* > *Least common multiple* ...)
The *lcm* function calculates the *l*east *c*ommon *m*ultiple for two polynomials, or integers. This function belongs to the *functs* package, which must be loaded before applying the function. Function *lcm* can be invoked from the *Calculus* menu, however, before using this menu item it is necessary to load the *functs* package. Thus, the first command to enter is:

```
load("functs");
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/simplification/functs.mac
```

The following example shows function *lcm* applied to pairs of numbers:

```
[lcm(2,4),lcm(2,3),lcm(5,12,10,6)];
[ 4 , 6 , 60 ]
```

Next, we apply function *lcm* to a pair of polynomials:

```
p1 : x^2-2*x;
```

$$x^2 - 2x$$

```
p2 : x^3+2*x^2-15*x;
```

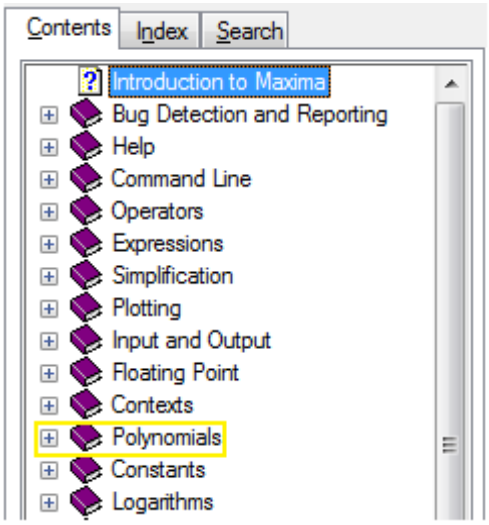
$$x^3 + 2x^2 - 15x$$

```
lcm(p1,p2);
```

$$(x - 3)(x - 2)x(x + 5)$$

NOTE 1: Function *lcm* belongs to package *functs* which contains a number of other useful functions that apply to polynomials and numbers. The contents of package *functs* are presented in a section at the end of this Chapter.

NOTE 2: For additional information on polynomials activate the *Maxima Manual* (*Help* > *Maxima help*) and find the *Polynomial* chapter in the *Contents* tab.



Simple operations with fractions
In this section we provide examples of functions that apply to fractions.

combine
The *combine* function can be used to collect fractions with the same denominator:

```
(%i50)  3/(x+1) + 5/(y-2) -3*x/(x+1) + 2*y/(y-2);
```

$$(\%o50)\quad \frac{2\,y}{y-2}+\frac{5}{y-2}-\frac{3\,x}{x+1}+\frac{3}{x+1}$$

```
(%i51)  combine(%);
```

$$(\%o51)\quad \frac{2\,y+5}{y-2}+\frac{3-3\,x}{x+1}$$

partfrac (*Calculus* > *Partial fractions...*)
The *partfrac* function decomposes a single fraction into its partial fractions:

```
(%i85)  (x^4-2*x^3+x^2+2)/(x^4-2*x^3-x^2+2*x);
```

$$(\%o85)\quad \frac{x^4-2\,x^3+x^2+2}{x^4-2\,x^3-x^2+2\,x}$$

```
(%i86)  partfrac(%,x);
```

$$(\%o86)\quad -\frac{1}{x+1}+\frac{1}{x}-\frac{1}{x-1}+\frac{1}{x-2}+1$$

cfdisrep (*Calculus* > *Continued fraction*)
The *cfdisrep* function is used to produce a continued fraction given the coefficients of those fractions as illustrated in this example:

```
(%i95)  cfdisrep([1,0,1,2,1,3,1,4]);
```

$$(\%o95)\quad 1+\cfrac{1}{0+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+\frac{1}{4}}}}}}}$$

Functions in the *functs* package

This section presents examples of functions in the *functs* package. The descriptions of the functions was taken from the *Maxima* online help facility for *functs*, i.e.,

```
?? functs;
70.4 Package functs
  <output ommited>
```

Interestingly enough, function *lcm*, which was presented in an earlier section, is not included in the help entries for *functs*.

rempart (*expr*, *n*)

Removes part *n* from the expression *expr*. If *n* is a list of the form [*l*,*m*] then parts *l* through *m* are removed.

```
p1 : x^6+x^5+x^4+x^3+x^2+x;
```

$$x^6+x^5+x^4+x^3+x^2+x$$

```
[rempart(p1,2),rempart(p1,3),rempart(p1,4),rempart(p1,5)];
```

$$[x^6,\ x^3+x^2+x+x^6+x^5,\ x^2+x+x^6+x^5+x^4,\ x+x^6+x^5+x^4+x^3]$$

```
[rempart(p1,[2,3]),rempart(p1,[2,4]),rempart(p1,[2,5]),rempart(p1,[3,5])];
```

$$[x^6,\ x^6,\ x^6,\ x+x^6+x^5]$$

wronskian ([f_1, ..., f_n], *x*)

Returns the Wronskian matrix of the functions f_1, ..., f_n in the variable *x*. f_1, ..., f_n may be the names of user-defined functions, or expressions in the variable *x*.

```
f1(x):=x^3+3 $ f2(x):=sqrt(1+x^2) $ f3(x):=x*sin(x)$

wronskian([f1(x),f2(x),f3(x)],x);
```

$$\begin{bmatrix} x^3+3 & \sqrt{x^2+1} & x\sin(x) \\ 3x^2 & \frac{x}{\sqrt{x^2+1}} & \sin(x)+x\cos(x) \\ 6x & \frac{1}{\sqrt{x^2+1}}-\frac{x^2}{(x^2+1)^{3/2}} & 2\cos(x)-x\sin(x) \end{bmatrix}$$

tracematrix(*M*)
Returns the trace (sum of the diagonal elements) of matrix *M*.

```
A : matrix([3,5,-1],[2,5,-2],[0,1,4]);
```

$$\begin{bmatrix} 3 & 5 & -1 \\ 2 & 5 & -2 \\ 0 & 1 & 4 \end{bmatrix}$$

```
tracematrix(A);
```

12

rational(*z*)
Multiplies numerator and denominator of *z* by the complex conjugate of denominator, thus rationalizing the denominator.

```
z1:3+5*%i $ z2:-1+7*%i $

z3 : z1/z2;
```

$$\frac{5\,\%i + 3}{7\,\%i - 1}$$

```
rational(%);
```

$$-\frac{13\,\%i - 16}{25}$$

A similar result is obtained by using function *rectform*:

```
rectform(z3);
```

$$\frac{16}{25} - \frac{13\,\%i}{25}$$

gcdivide(*p*,*q*)
When *takegcd* is *true*, *gcdivide* divides the polynomials *p* and *q* by their greatest common divisor and returns the ratio of the results.

```
p1:x^3-8*x^2+21*x-18 $ p2:x^3-x^2-14*x+24 $

takegcd;
true

gcdivide(p1,p2);
```

$$\frac{x - 3}{x + 4}$$

When *takegcd* is *false*, *gcdivide* returns *p*/*q*.

```
takegcd:false $ gcdivide(p1,p2);
```

$$\frac{x^3 - 8x^2 + 21x - 18}{x^3 - x^2 - 14x + 24}$$

arithmetic (*a*,*d*,*n*)
Returns the *n*-th term of the arithmetic series *a*, *a*+*d*, *a*+2*d*, ..., *a*+(*n*-1)*d*.

```
[arithmetic(a,d,n),arithmetic(3,5,4)];
```

$$[d(n-1)+a,\ 18]$$

geometric (*a*,*r*,*n*)
Returns the *n*-th term of the geometric series *a*, *ar*, ar^2, ..., ar^{n-1}.

```
[geometric(a,r,n),geometric(3,5,4)];
```

$$[a\,r^{n-1},\ 375]$$

harmonic (*a*,*b*,*c*,*n*)
Returns the *n*-th term of the harmonic series *a*/*b*, *a*/(*b*+*c*), *a*/(*b*+2*c*), ..., *a*/(*b*+(*n*-1)c).

```
[harmonic(a,b,c,n),harmonic(2,3,5,4)];
```

$$\left[\frac{a}{c(n-1)+b},\ \frac{1}{9}\right]$$

arithsum (*a*,*d*,*n*)
Returns the sum of the arithmetic series from *1* to *n*.

```
arithsum(3,5,4);
```

42

geosum (*a*,*r*,*n*)
Returns the sum of the geometric series from *1* to *n*. If *n* is infinity (inf) then a sum is finite only if the absolute value of *r* is less than 1.

```
[geosum(3,5,4),geosum(3,5,inf),geosum(3,1/5,inf)];
```

$$\left[468,\ \infty,\ \frac{15}{4}\right]$$

gaussprob (*x*)
Returns the Gaussian probability function `%e^(-*x*^2/2)/sqrt(2*%pi)', i.e., the standard normal probability density function (pdf).

Here is a list of ordinates of the standard normal *pdf*:

```
[gaussprob(-2),gaussprob(-1),gaussprob(0),gaussprob(1),gaussprob(2)];
```

$$\left[\frac{\%e^{-2}}{\sqrt{2}\sqrt{\pi}},\frac{1}{\sqrt{2}\sqrt{\%e}\sqrt{\pi}},\frac{1}{\sqrt{2}\sqrt{\pi}},\frac{1}{\sqrt{2}\sqrt{\%e}\sqrt{\pi}},\frac{\%e^{-2}}{\sqrt{2}\sqrt{\pi}}\right]$$

```
float(%);
[ 0.053990966513188 , 0.24197072451914 , 0.39894228040143 , 0.24197072451914 , 0.053990966513188 ]
```

Here is a list of probabilities of the intervals *-1 < x < 1, -2 < x < 2, and -3 < x < 3*, respectively,

```
[integrate(gaussprob(x),x,-1,1),integrate(gaussprob(x),x,-2,2),
 integrate(gaussprob(x),x,-3,3)];
```

$$\left[\operatorname{erf}\left(\frac{1}{\sqrt{2}}\right),\frac{\frac{\sqrt{2}\sqrt{\pi}\operatorname{erf}(\sqrt{2})}{2}+\frac{\sqrt{2}\sqrt{\pi}\operatorname{erf}\left(\frac{2}{\sqrt{2}}\right)}{2}}{\sqrt{2}\sqrt{\pi}},\operatorname{erf}\left(\frac{3}{\sqrt{2}}\right)\right]$$

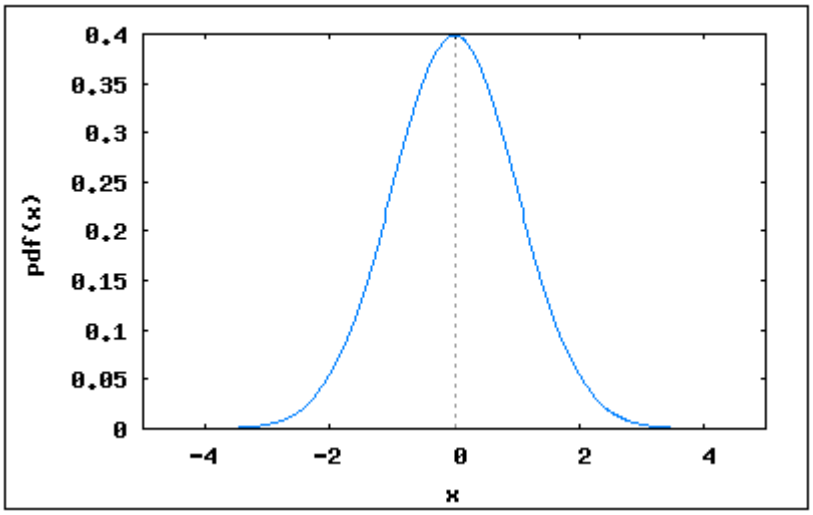
```
float(%);
[ 0.68268949213709 , 0.95449973610364 , 0.99730020393674 ]
```

Notice that integrals of the *gaussprob*(*x*) function are given in terms of the *error function* (*erf*). To find out about the error function check the *Maxima* online help:

```
?? erf;
0: erf  (Functions and Variables for Integration)
1: erfflag  (Functions and Variables for Integration)
2: nextlayerfactor  (Package facexp)
Enter space-separated numbers, `all' or `none':  0;
 -- Function: erf (<x>)
     Represents the error function, whose derivative is:
     `2*exp(-x^2)/sqrt(%pi)'.
```

Finally, a plot of the standard normal *pdf* is shown below:

```
wxplot2d(gaussprob(x),[x,-5,5],[xlabel,"x"],[ylabel,"pdf(x)"]);
```



gd (x)
Returns the Gudermannian function `2 * atan(%e^x - %pi/2)'.

```
[gd(x),gd(-1),gd(0),gd(1)];
```

$$\left[\ 2\ \mathrm{atan}\left(\%e^{x}-\frac{\pi}{2}\right),\ -2\ \mathrm{atan}\left(\frac{\pi}{2}-\%e^{-1}\right),\ -2\ \mathrm{atan}\left(\frac{\pi}{2}-1\right),\ -2\ \mathrm{atan}\left(\frac{\pi}{2}-\%e\right)\right]$$

```
float(%);
```

$$\left[\ 2.0\ \mathrm{atan}\left(2.718281828459045^{x}-1.570796326794897\right),\ -1.754503565018189\ ,\right.$$
$$\left.-1.037338738510033\ ,\ 1.707937438104075\ \right]$$

agd (x)
Returns the inverse Gudermannian function `log (tan (%pi/4 + x/2)))'.

```
[agd(x),agd(-1),agd(0),agd(1)];
```

$$\left[\ \log\left(\tan\left(\frac{x}{2}+\frac{\pi}{4}\right)\right),\ \log\left(\tan\left(\frac{\pi}{4}-\frac{1}{2}\right)\right),\ 0\ ,\ \log\left(\tan\left(\frac{\pi}{4}+\frac{1}{2}\right)\right)\right]$$

```
float(%);
```

$$\left[\ \log\left(\tan\left(0.5\,x+0.78539816339745\right)\right),\ -1.226191170883517\ ,\ 0.0\ ,\ 1.226191170883517\ \right]$$

vers (x)
Returns the versed sine `1 - cos (x)'.

```
[vers(x),vers(%pi/6),vers(%pi/3),vers(%pi/2)];
```

$$\left[\ 1-\cos(x)\ ,\ 1-\frac{\sqrt{3}}{2},\ \frac{1}{2},\ 1\ \right]$$

covers (x)
Returns the coversed sine `1 - sin (x)'.

```
[covers(x),covers(%pi/6),covers(%pi/3),covers(%pi/2)];
```

$$[\ 1 - \sin(x)\ ,\ \frac{1}{2}\ ,\ 1 - \frac{\sqrt{3}}{2}\ ,\ 0\ ]$$

exsec (x)
Returns the exsecant `sec (x) - 1'.

```
[exsec(x),exsec(%pi/6),exsec(%pi/3),exsec(%pi/12)];
```

$$[\ \sec(x) - 1\ ,\ \frac{2}{\sqrt{3}} - 1\ ,\ 1\ ,\ \sec\left(\frac{\pi}{12}\right) - 1\ ]$$

```
float(%);
```

$$[\ \sec(x) - 1.0\ ,\ 0.15470053837925\ ,\ 1.0\ ,\ 0.035276180410083\ ]$$

hav (x)
Returns the haversine `(1 - cos(x))/2'.

```
[hav(x),hav(%pi/6),hav(%pi/3),hav(%pi/12)];
```

$$[\ \frac{1 - \cos(x)}{2}\ ,\ \frac{1 - \frac{\sqrt{3}}{2}}{2}\ ,\ \frac{1}{4}\ ,\ \frac{1 - \cos\left(\frac{\pi}{12}\right)}{2}\ ]$$

```
float(%);
```

$$[\ 0.5(1.0 - 1.0\cos(x))\ ,\ 0.066987298107781\ ,\ 0.25\ ,\ 0.017037086855466\ ]$$

4

Basic plotting commands in *Maxima*

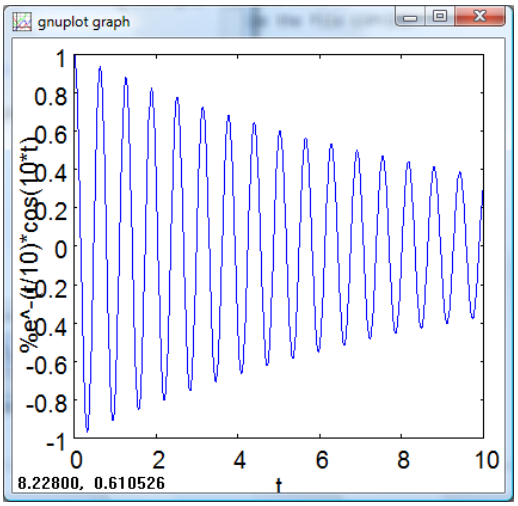
In this Chapter we present examples of plotting commands in *Maxima* useful for the creation of two-dimensional and three-dimensional graphics.

The *plot2d* command

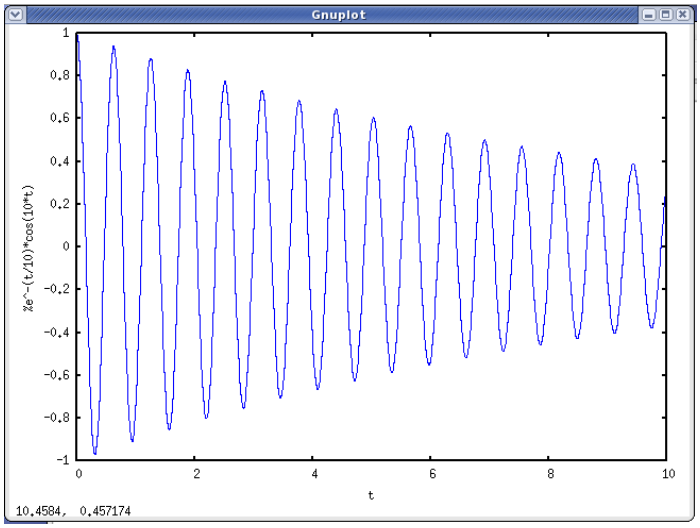
The *plot2d* command, in its simplest form, requires as input an expression or function name, and a range of values of the independent variable. Consider the following example typed in a *wxMaxima* window:

```
(%i1)  plot2d(exp(-t/10)*cos(10*t),[t,0,10]);
```

By default, this command produces a plot in a *gnuplot* window, as shown below for a *Windows Vista* environment:

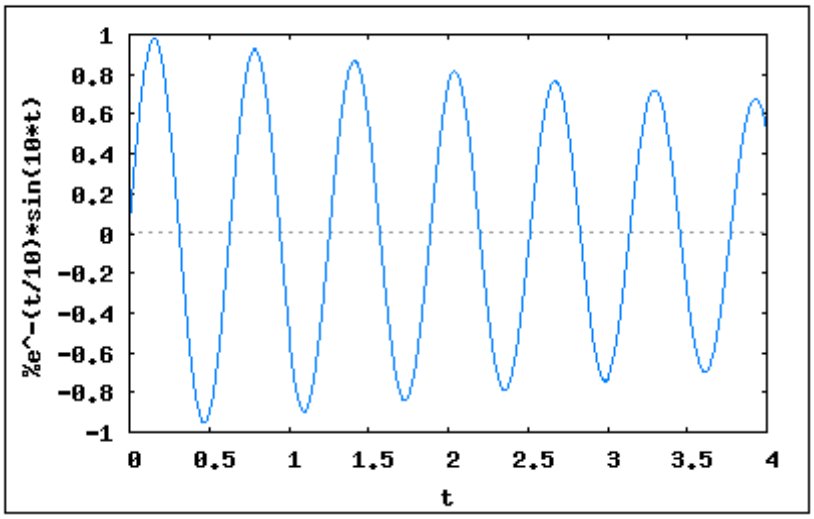


In a *Linux* environment, specifically a *Fedora 7 Linux* environment, the following *Gnuplot* window would be produced:



In the *wxMaxima* window a plot can be produced inline by using the command *wxplot2d*, e.g.,

```
(%i2) wxplot2d(exp(-t/10)*sin(10*t),[t,0,4]);
```



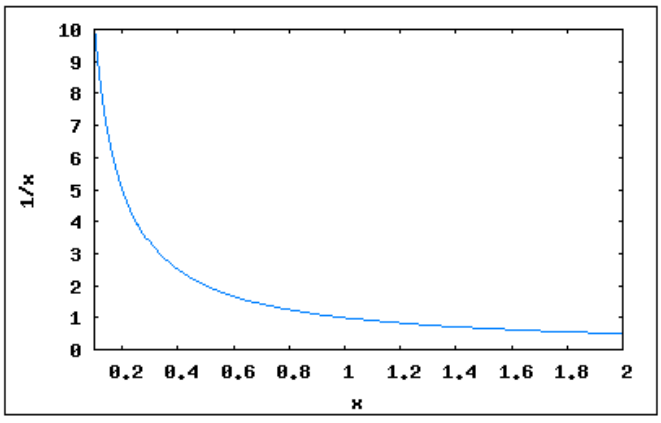
(%t2)

NOTE: It will be more accurate to say that to produce an inline two-dimensional plot you need to use the *wx* "wrapper" with the *plot2d* command, rather than referring to a *wxplot2d* command.

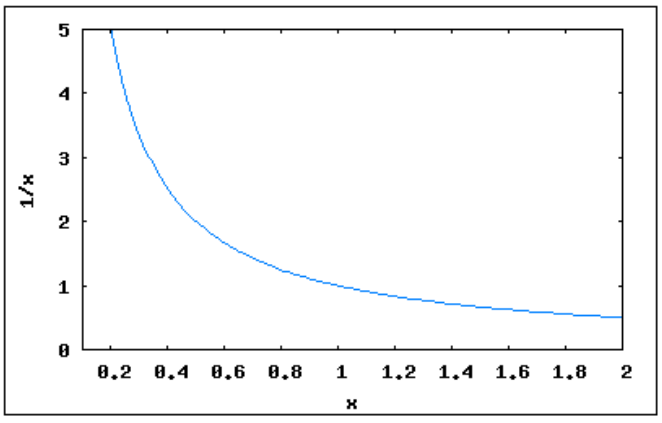
Specifying the vertical range

Besides specifying the abscissa or horizontal range (i.e., the *x* range), the *plot2d* command allows the user to specify the ordinate or vertical range (i.e., the *y* range). Consider the following two examples:

```
wxplot2d(1/x,[x,0.1,2]);
```



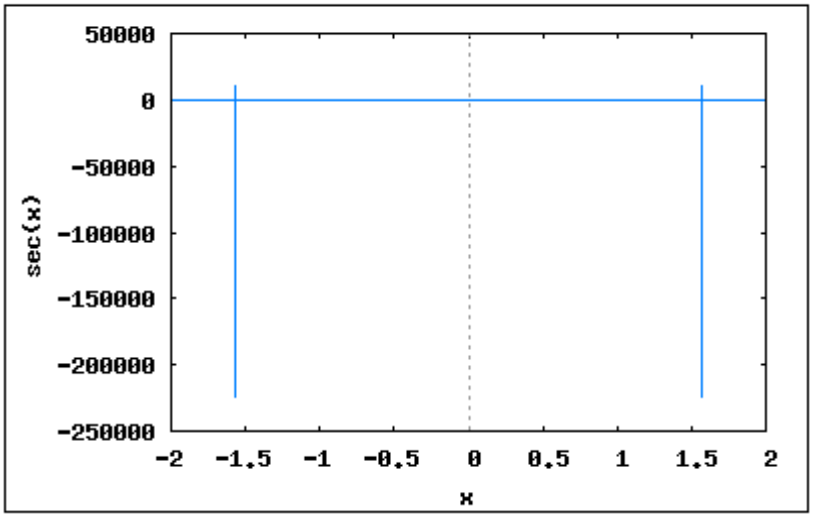

```
wxplot2d(1/x,[x,0.1,2],[y,0,5]);
```



In the first case no vertical range is specified, thus, *Maxima* will tend to include the default vertical range [0,10]. In the second case, the vertical range [y,0,5] is specified, thus reducing the vertical range to half of that in the first case.

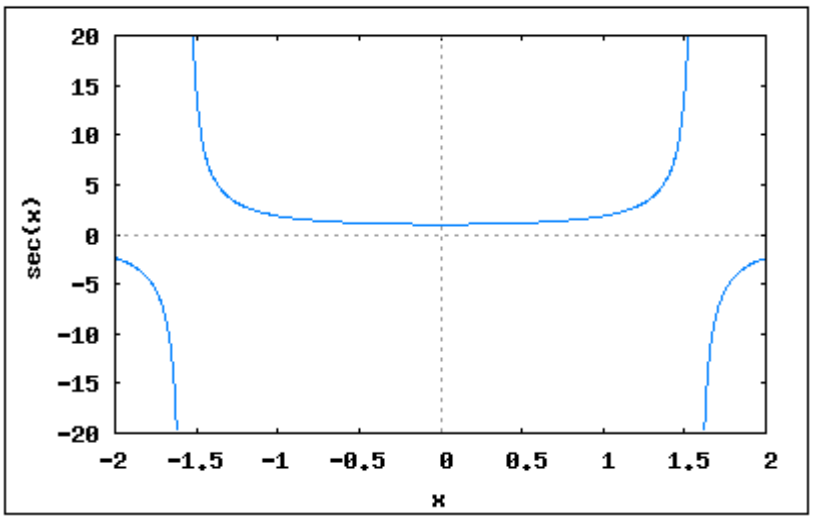
The following example illustrates the case of the function *sec(x)* which takes positive and negative infinite values at certain points. Without specifying the vertical scale the graph does not show much detail of the curve, i.e.,

```
wxplot2d(sec(x),[x,-2,2]);
```



The function diverges at values of -p/2 and p/2 as indicated by the vertical lines. The default vertical range extends from -250,000 to 50,000, a very large range indeed. Specifying a smaller vertical range,say -20 < y < 20, allows the user to see the details of the curve behavior:

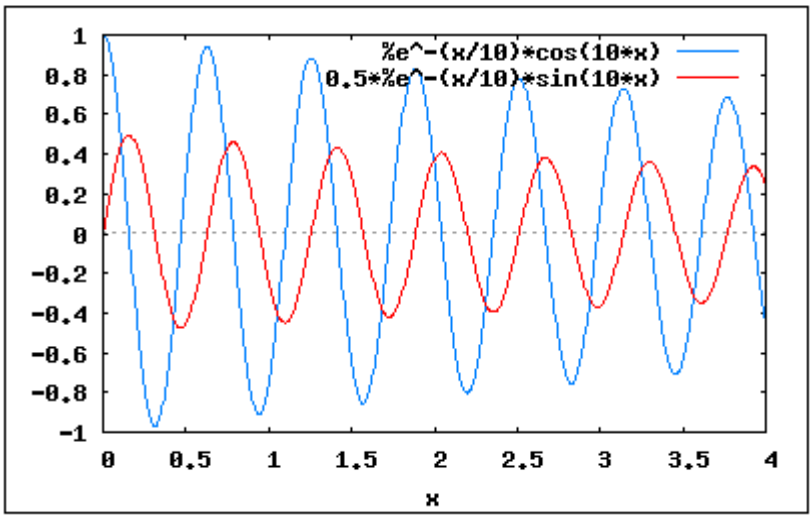
```
wxplot2d(sec(x),[x,-2,2],[y,-20,20]);
```



Plotting more than one curve

Specifying a list of functions or expressions allows the plotting of more than one curve through the use of *plot2d* (or *wxplot2d*), e.g.,

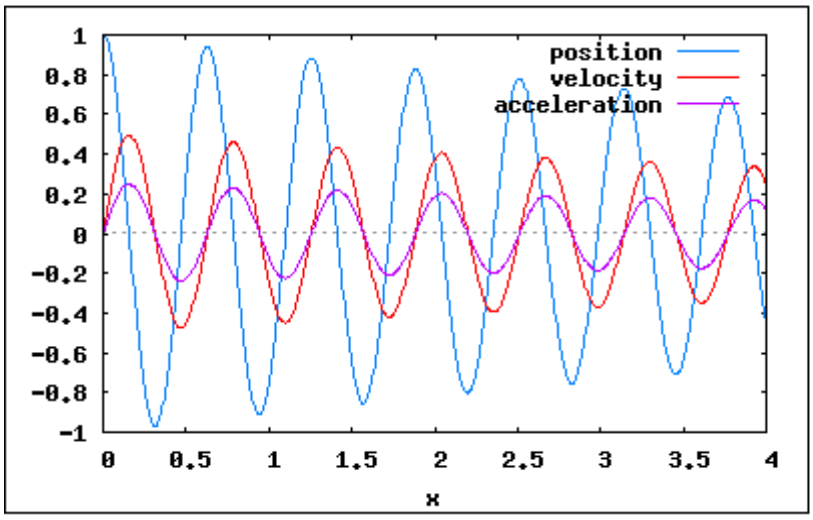
```
f(x):=exp(-x/10)*cos(10*x)  $   g(x):=0.5*exp(-x/10)*sin(10*x) $
wxplot2d([f(x),g(x)],[x,0,4]);
```



Specifying legends for multiple curves

In the example above, the plot shows the function expressions as the legends for the two curves. The user may specify the legends to be included by using the *legend* option as illustrated in the following example:

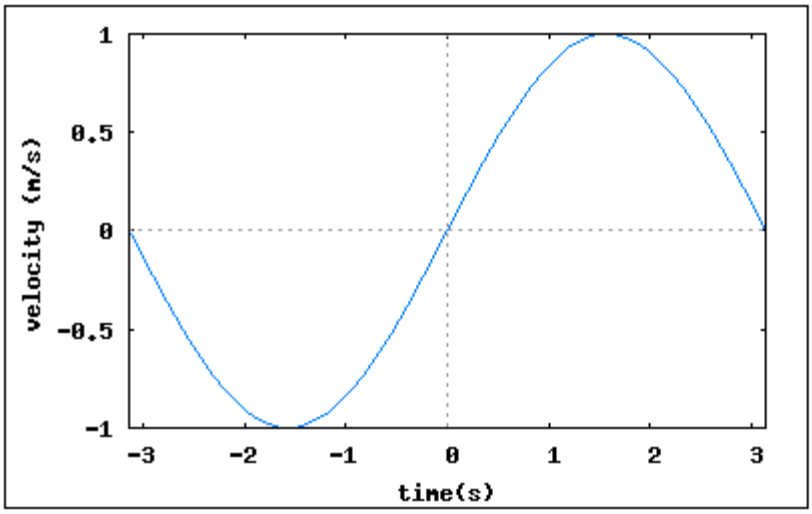
```
f(x):=exp(-x/10)*cos(10*x) $ g(x):=0.5*exp(-x/10)*sin(10*x) $ h(x):= 0.5*g(x) $
wxplot2d([f(x),g(x),h(x)],[x,0,4],[legend,"position","velocity","acceleration"]);
```



Specifying labels for the plot

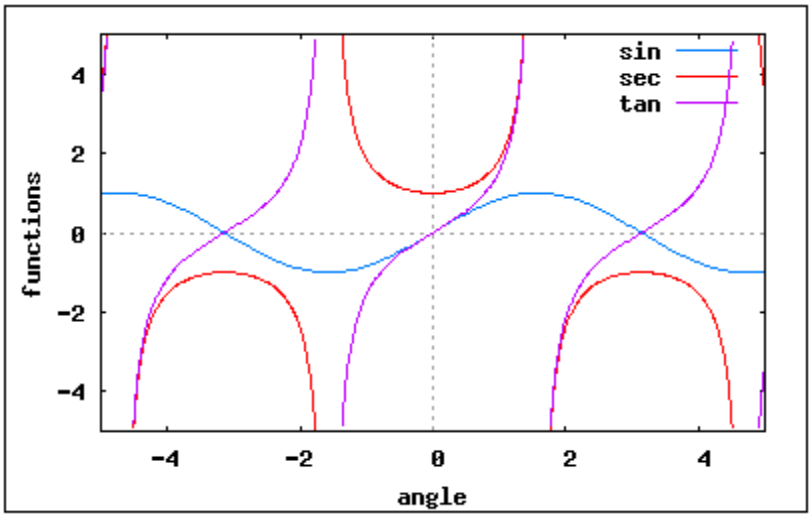
The following example illustrates the use of the options *xlabel* and *ylabel* to specify plot labels:

```
wxplot2d(sin(x),[x,-%pi,%pi],[xlabel,"time(s)"],[ylabel,"velocity (m/s)"]);
```



The following example illustrates the specification of vertical scale, legends, and labels:

```
wxplot2d([sin(x),sec(x),tan(x)],[x,-5,5],[y,-5,5],
[legend,"sin","sec","tan"],[xlabel,"angle"],[ylabel,"functions"]);
```



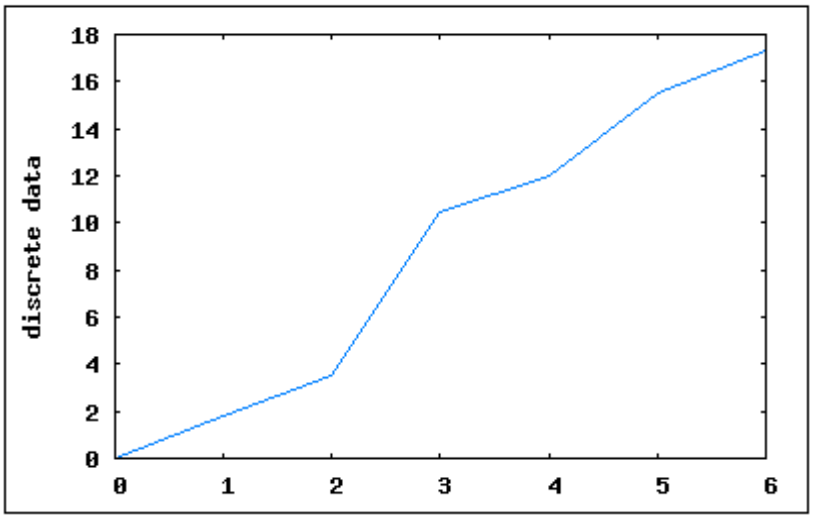
Plotting discrete data

The examples shown above, where the curves plotted are based on expressions or functions, produce, by default, smooth, continuous lines. If the data to be plotted consists of discrete data points, say,

```
x : [0,1,2,3,4,5,6] $ y : [0,1.8,3.5,10.5,12.0,15.5,17.3] $
```

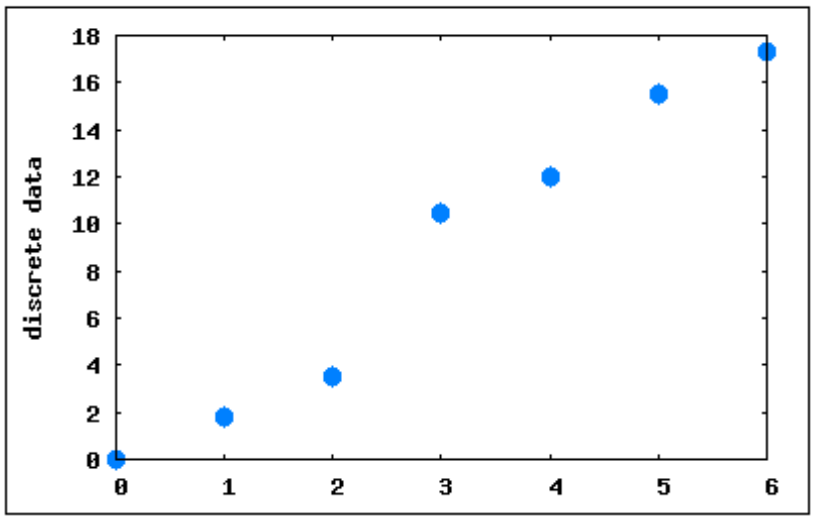
one can use the *discrete* option, altogether with the list of data points *x* and *y*, to produce a plot, e.g.,

```
wxplot2d([discrete,x,y]);
```



Style: points - The resulting plot shows a series of segments joining the individual points corresponding to the data in lists *x* and *y*. The continuous line is the default plot *style*. In order to produce discrete points use the option [*style*, [*points*]], e.g.,

```
wxplot2d([discrete,x,y],[style,[points]]);
```



The *points* option for *style* can have one, two, or three additional options of the form:

[*points*,`radius`,`color`,`object`].

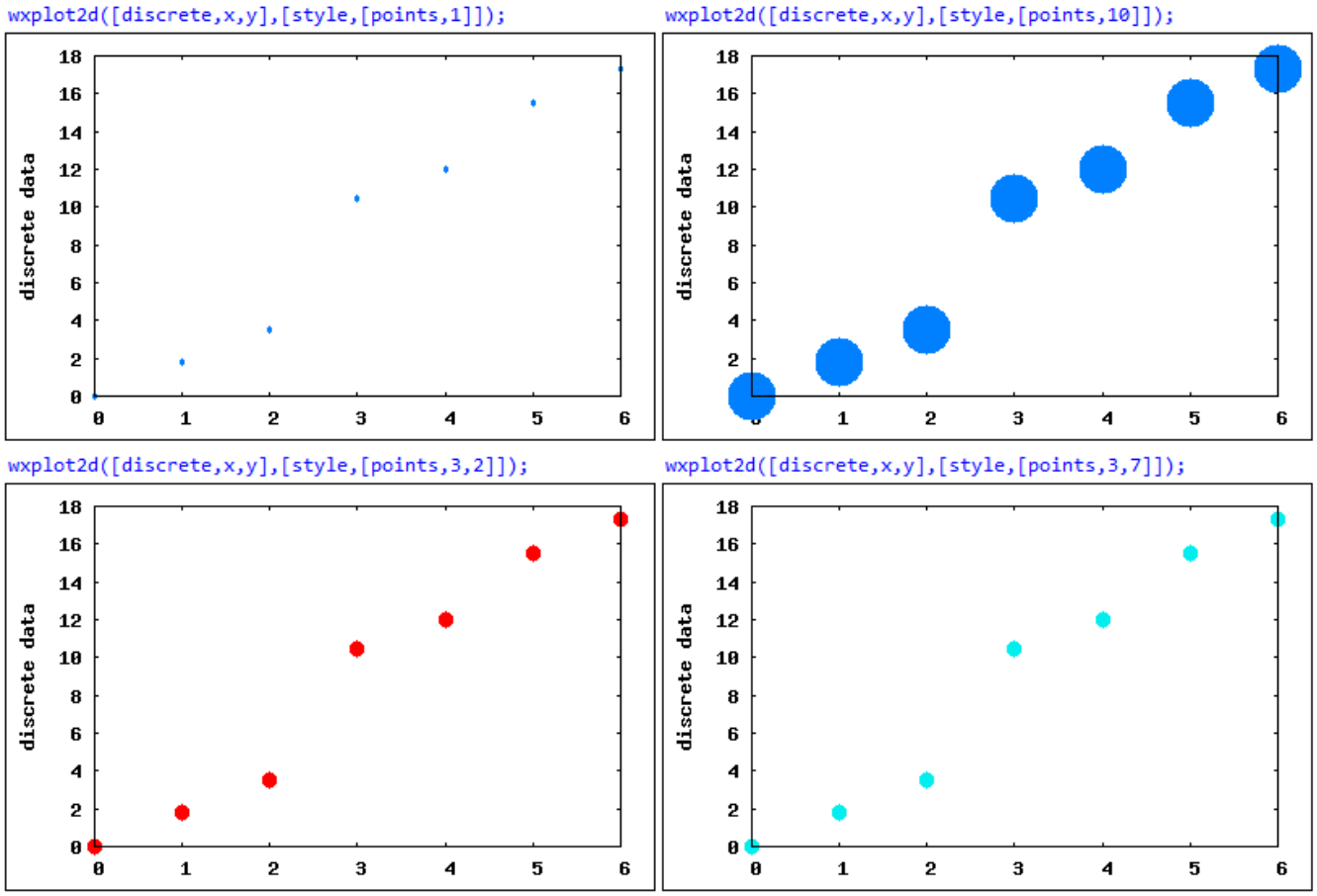
The first option, `radius`, represents the radius of the points to be plotted. The larger the value of this first parameter the larger the individual point symbols. The second option, `color`, represents the color of the points with default values:

1-blue 2-red 3-magenta 4-orange 5-brown 6-lime 7-aqua

The sequence of colors repeat after the number 7, i.e., 8 will be *blue*, 9 *red*, etc. The last option, `object`, in the *points* specification represents the type of symbol, or object, that will be plotted, according to the following codes:

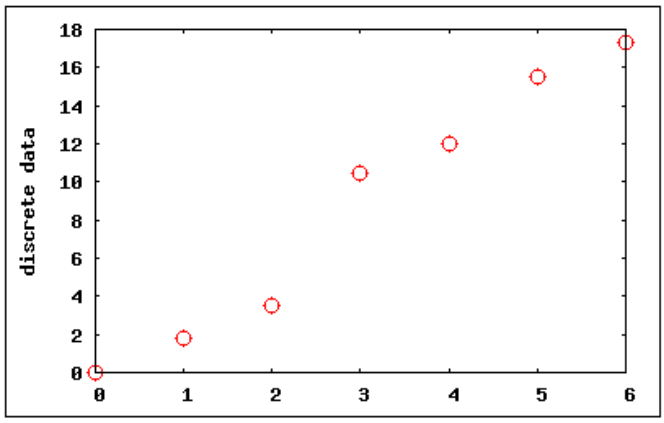
1: filled circles
2: open circles
3: plus signs (+)
4: times sign (x)
5: asterisk (*)
6: filled squares
7: open squares
8: filled triangles
9: open triangles
10:filled inverted triangles
11: open inverted triangles
12: filled lozenges
13: open lozenges.

The following figure illustrates three different point sizes (1, 10, 3) and two different colors (2 – red, 7-acqua):

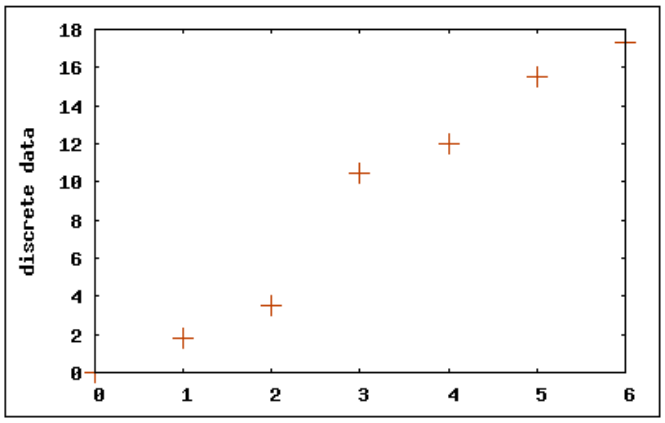


The following examples show different combinations of the three options for *points*:

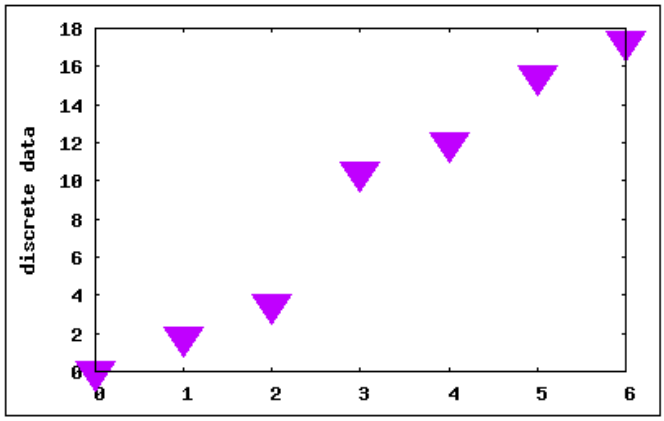
```
wxplot2d([discrete,x,y],[style,[points,3,2,2]]);
```



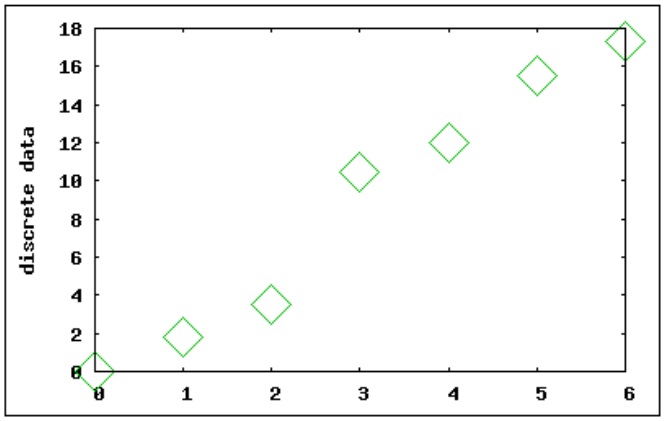

```
wxplot2d([discrete,x,y],[style,[points,4,5,3]]);
```




```
wxplot2d([discrete,x,y],[style,[points,6,3,10]]);
```

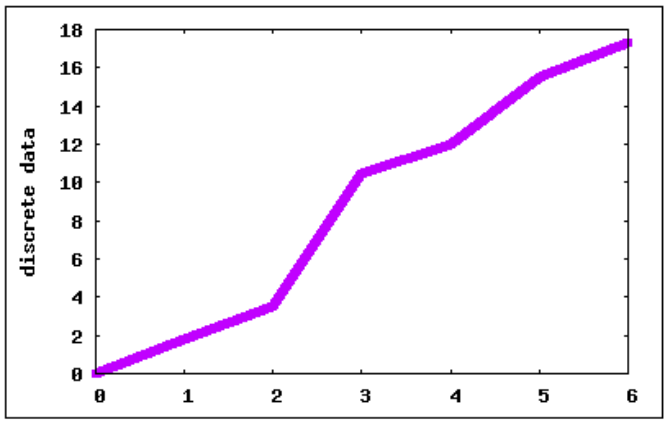



```
wxplot2d([discrete,x,y],[style,[points,8,6,13]]);
```

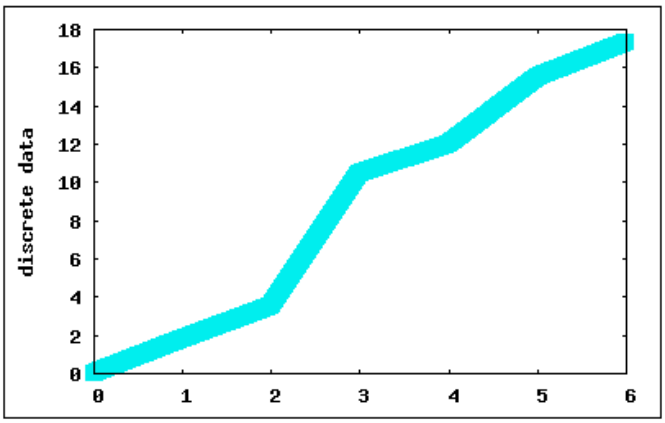


Style: lines – As with the case of *points*, the style option *lines* can alter the appearance of continuous lines by using the form [*lines*,[`thickness,color`]]. The option *thickness* operates similar to *radius* for *points*, while the option *color* is the same as in *points*. Some examples are shown below:

```
wxplot2d([discrete,x,y],[style,[lines,5,3]]);
```

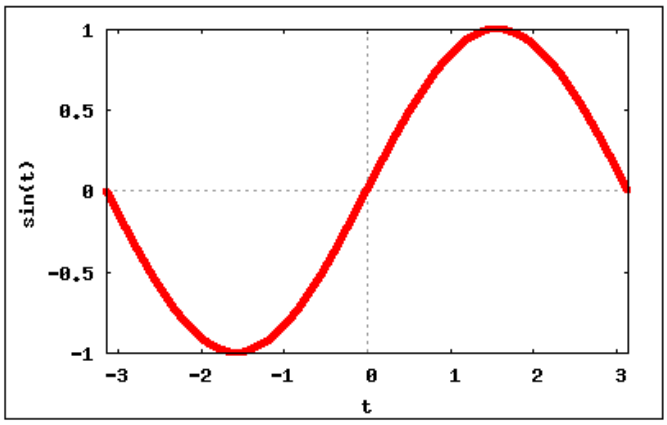



```
wxplot2d([discrete,x,y],[style,[lines,10,7]]);
```

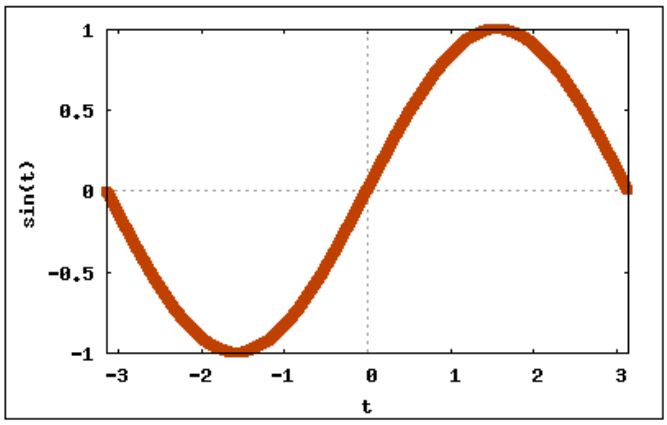


The *lines* option can be used with plots of functions or expressions, e.g.,

```
wxplot2d(sin(t),[t,-%pi,%pi],[style,[lines,4,2]]);
```

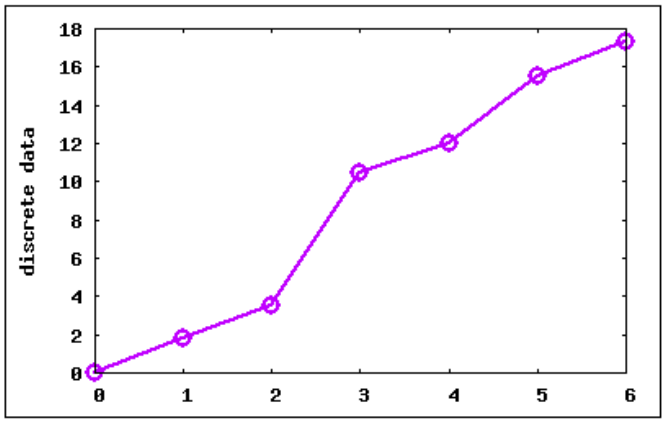



```
wxplot2d(sin(t),[t,-%pi,%pi],[style,[lines,6,5]]);
```

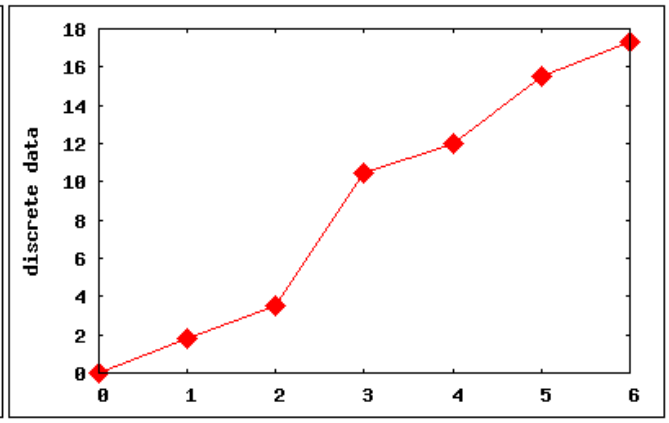


Styles: linespoints - This style combines lines with points and can use up to four options: [*linespoints*,[*line thickness, point radius, color, object*]]. Both lines and points will have the same color. Some examples are shown below:

```
wxplot2d([discrete,x,y],[style,[linespoints,2,3,3,2]]);
```

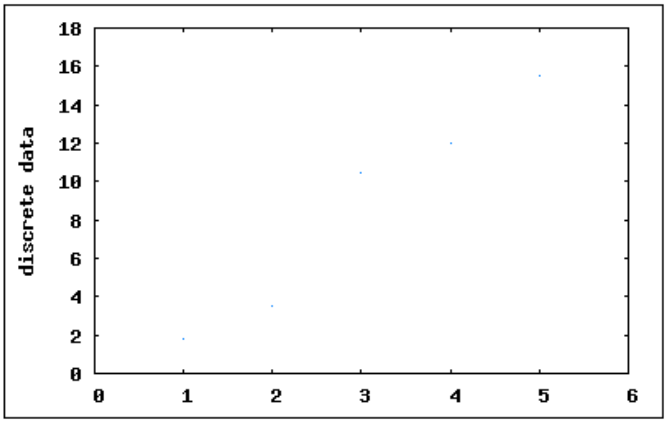



```
wxplot2d([discrete,x,y],[style,[linespoints,1,4,2,12]]);
```



Style: dots - This option shows discrete points a individual dots. These dots are a single-pixel size, therefore, they're hard to see in the plot, e.g.,

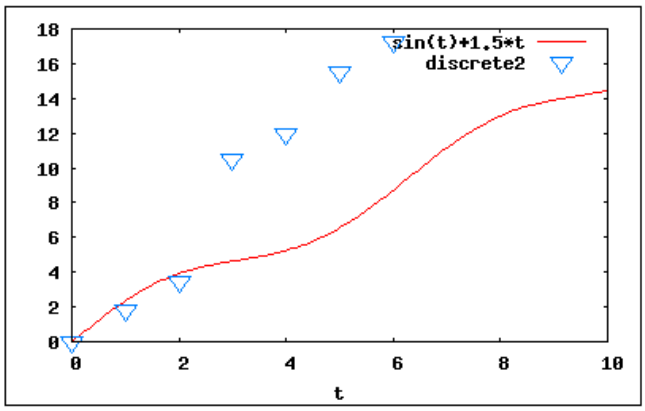
```
wxplot2d([discrete,x,y],[style,[dots]]);
```



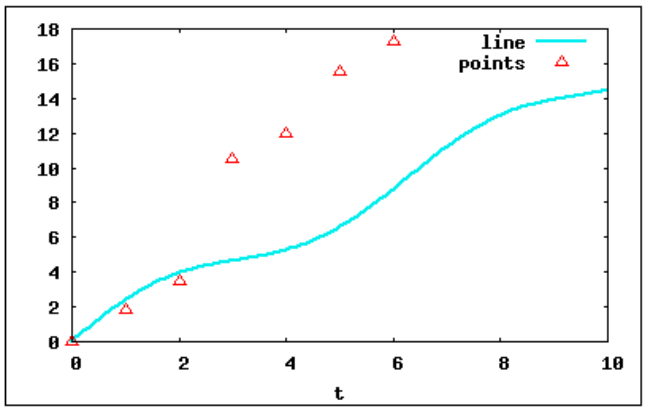
Plotting multiple types of data

The following examples illustrate the plotting of different types of data. We start by showing a plot of a continuous function together with discrete data. The first case shown illustrates the use of two different styles, one for the continuous line, and one for the discrete data points. The second case illustrated shows the use of two different styles, plus a legend specification.

```
wxplot2d([1.5*t+sin(t),[discrete,x,y]],[t,0,10],
        [style,[lines,1,2],[points,3,1,11]]);
```

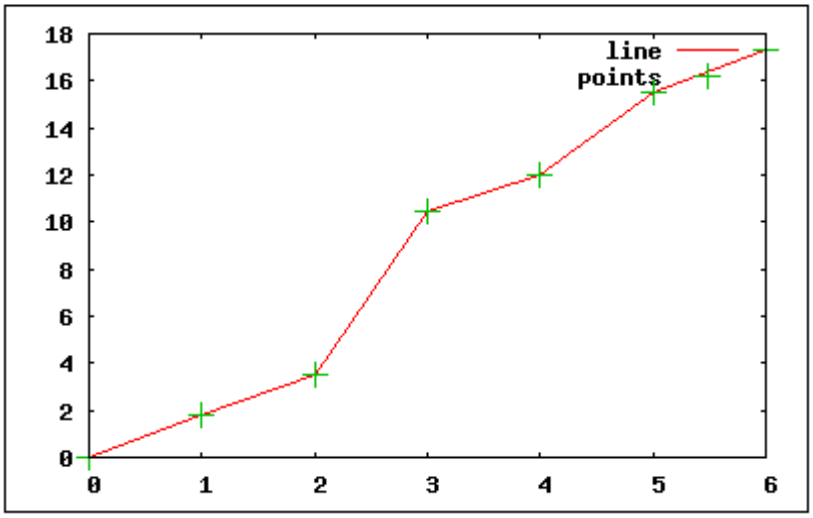



```
wxplot2d([1.5*t+sin(t),[discrete,x,y]],[t,0,10],
[style,[lines,2,7],[points,2,2,9]],[legend,"line","points"]);
```



The next example shows how the same set of discrete data can be plotted simultaneously as a line and points. This is similar to the plot of the single discrete data set with the single style *linespoints*, except that the latter forces both lines and points to be the same color.

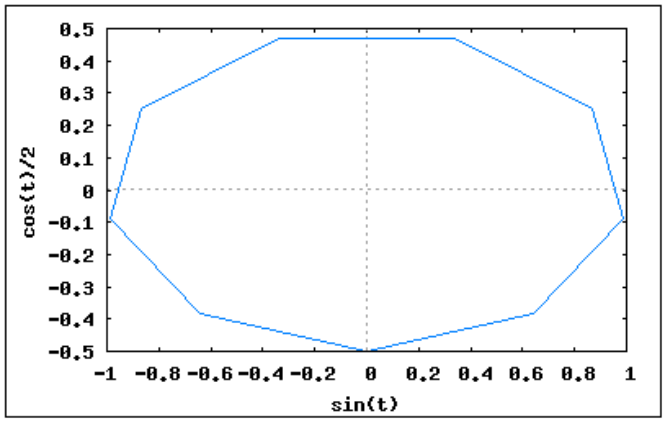
```
wxplot2d([[discrete,x,y],[discrete,x,y]],
[style,[lines,1,2],[points,4,6,3]],[legend,"line","points"]);
```



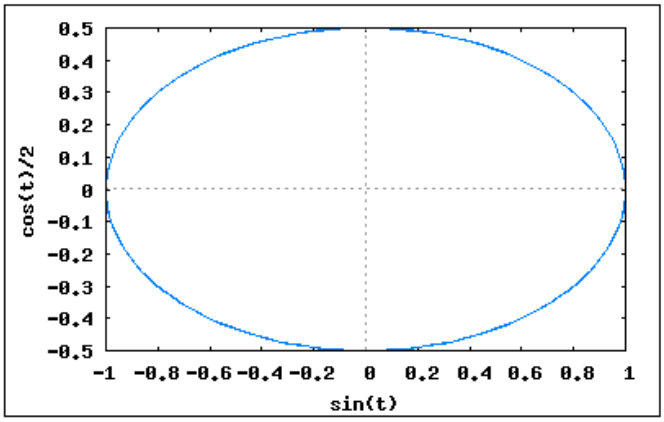
Plotting parametric plots

Function *plot2d* can be used to plot parametric plots as illustrated in the following example:

```
wxplot2d([parametric,sin(t),cos(t)/2],[t,-%pi,%pi]);
```



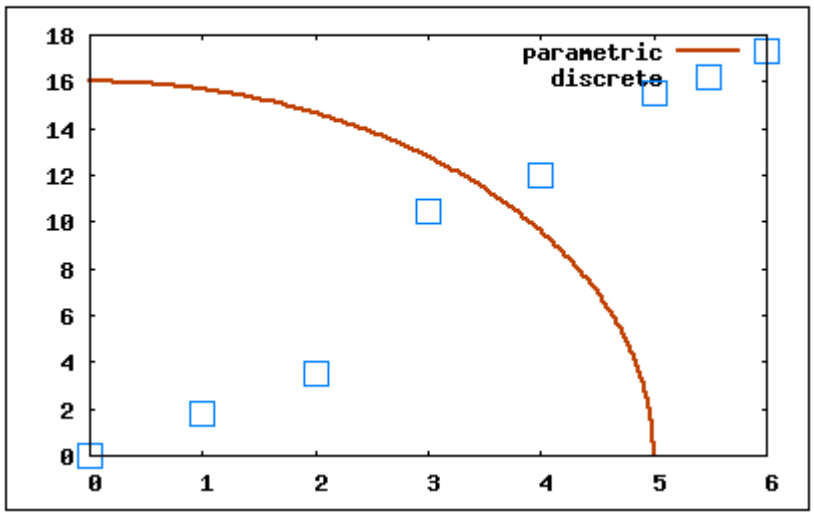

```
wxplot2d([parametric,sin(t),cos(t)/2],
[t,%pi,%pi],[nticks,200]);
```



Notice that the specification of a parametric plot requires the word *parametric*, and the expressions for the *x* and *y* components for the curve. The range of the independent variable is also required. Notice that, by default, the parametric plot used only 10 points to draw the curve. This situation can be improved using the plot option *nticks*. By making *nticks* to be *200* a smooth curve is produced for the parametric equations *x = sin(t), y =cos(t)/2*.

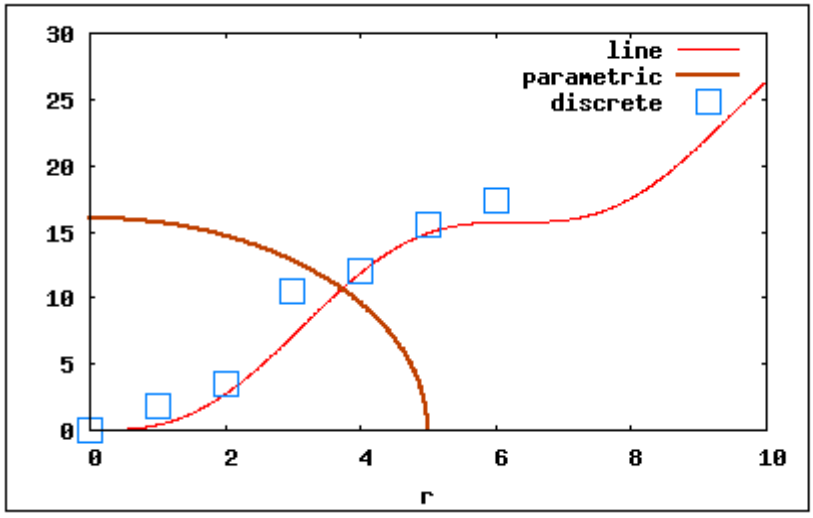
The next example shows a parametric curve and a discrete data set plotted in the same set of axes, including different styles and a legend option.

```
wxplot2d([[parametric,5*sin(t),16*cos(t)],[discrete,x,y]],
[t,0,%pi/2],[nticks,200],[style,[lines,2,5],[points,4,1,7]],
[legend,"parametric","discrete"]);
```



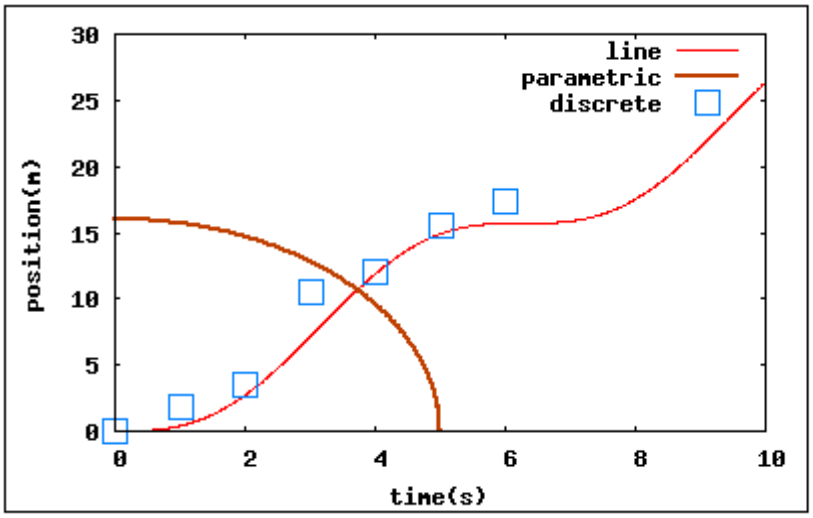
The next example shows a plot combining a function plot, a parametric plot, and a discrete data set:

```
wxplot2d([2.5*(r-sin(r)),[parametric,5*sin(t),16*cos(t)],[discrete,x,y]],
[r,0,10],[t,0,%pi/2],[nticks,200],[style,[lines,1,2],[lines,2,5],[points,4,1,7]],
[legend,"line","parametric","discrete"]);
```



The next example is the same as above, in terms of the plots, but including labels for the axes:

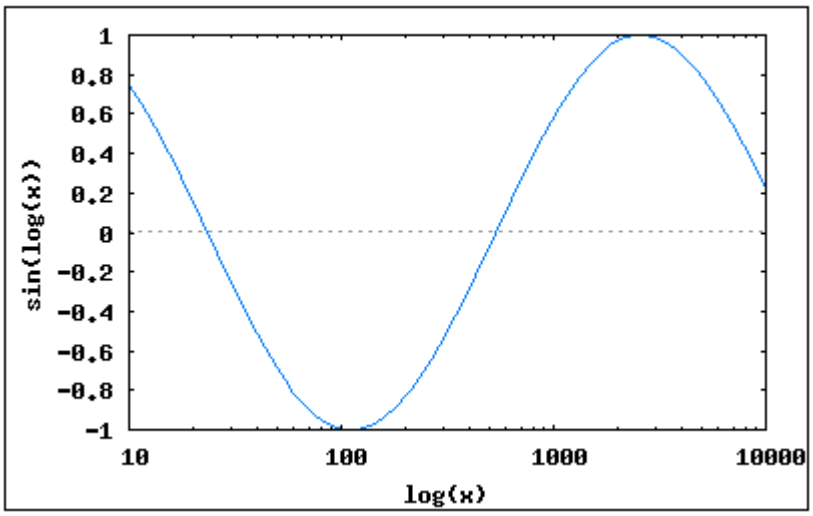
```
wxplot2d([2.5*(r-sin(r)),[parametric,5*sin(t),16*cos(t)],[discrete,x,y]],
[r,0,10],[t,0,%pi/2],[nticks,200],[style,[lines,1,2],[lines,2,5],[points,4,1,7]],
[legend,"line","parametric","discrete"],[xlabel,"time(s)"],[ylabel,"position(m)"]);
```



Logarithmic scales

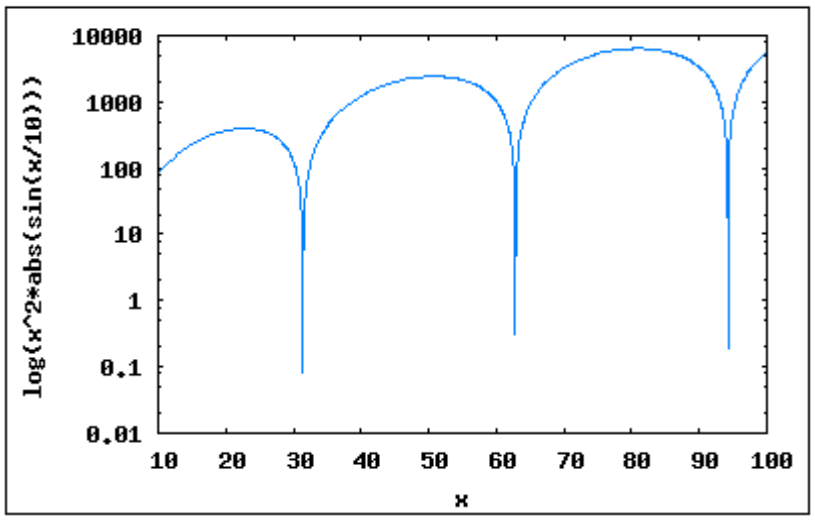
Adding the option [*logx,true*] will force the *x* axis scale to be logarithmic, whereas the option [*logy,true*] will force the *y* axis scale to be logarithmic. Some examples are shown below. First, an example of a semi-logarithmic plot with the *x* scale being logarithmic. Notice that the option [*logx,true*] automatically produces the label *log(x)* in the *x* axis.

```
wxplot2d(sin(log(x)),[x,10,10000],[logx,true]);
```



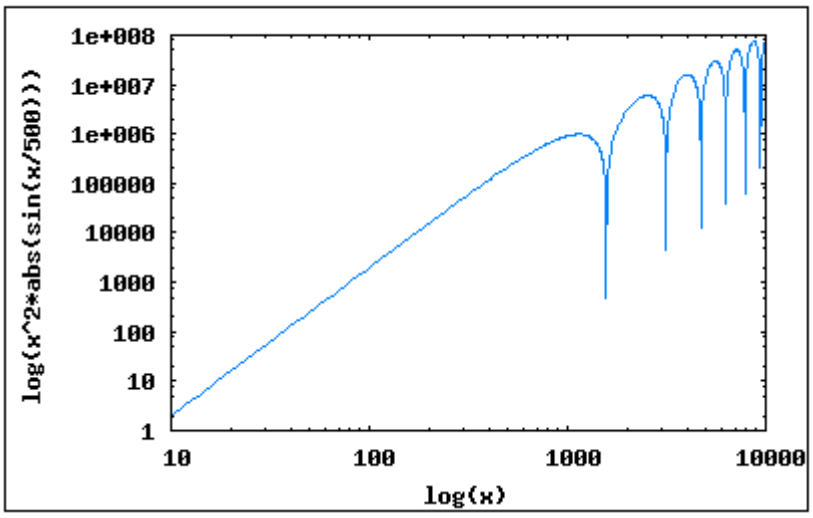
The following example illustrates the use of the [*logy,true*] option to produce a semi-logarithmic plot where the vertical scale is logarithmic:

```
wxplot2d(abs(x^2*sin(x/10)),[x,10,100],[logy,true]);
```



A double-logarithmic, or log-log, plot is shown next:

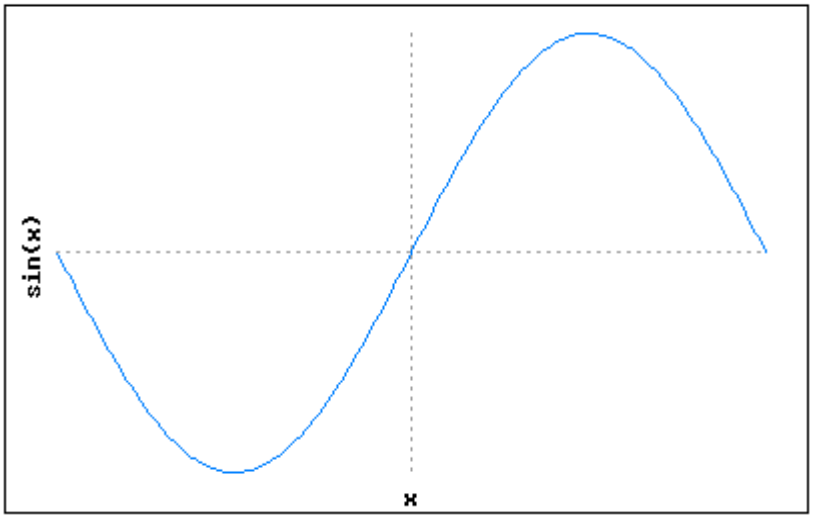
```
wxplot2d(abs(x^2*sin(x/500)),[x,10,10000],[logx,true],[logy,true]);
```



The box option

The box option is set to *true* by default. Changing it to *false* removes the frame from the plot, e.g.,

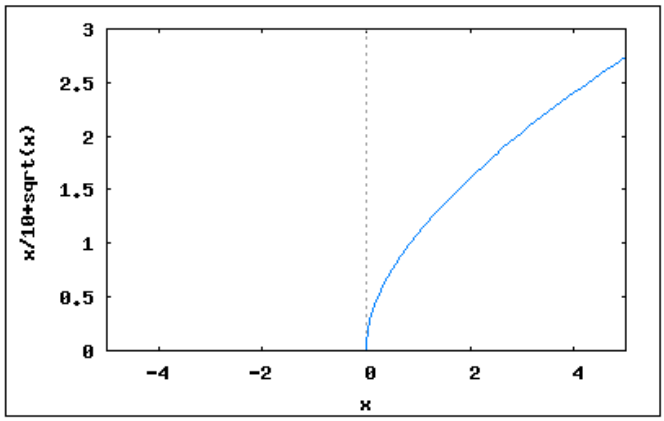
```
wxplot2d(sin(x),[x,-%pi,%pi],[box,false]);
```



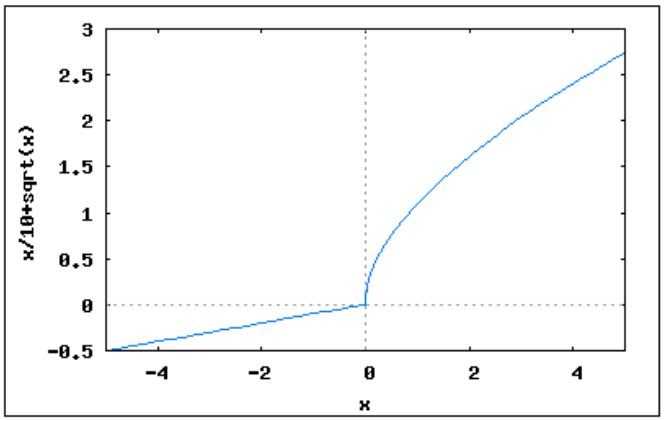
The *plot_realpart* option for complex numbers

Using the option [*plot_realpart,true*] with the *plot2d* command allows plotting the real part of complex numbers. The result is equivalent to plotting *realpart*(*x*) where *x* may contain complex numbers. The default setting for *plot_realpart* is *false*, in which case, complex numbers are ignored in the plot. To illustrate the use of the *plot_realpart* option consider the following plots:

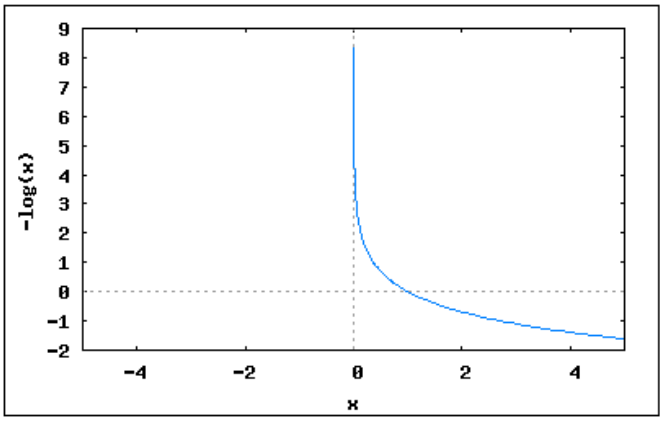
```
wxplot2d(x/10+sqrt(x),[x,-5,5]);
```



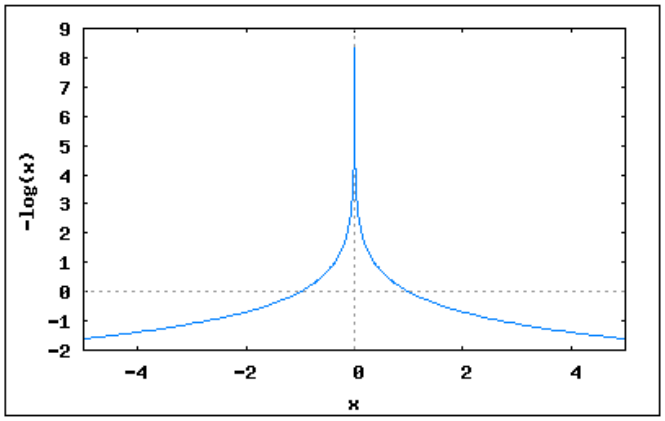

```
wxplot2d(x/10+sqrt(x),[x,-5,5],[plot_realpart,true]);
```




```
wxplot2d(log(1/x),[x,-5,5]);
```




```
wxplot2d(log(1/x),[x,-5,5],[plot_realpart,true]);
```



Gnuplot options

As illustrated in the examples shown above, the output of function *plot2d* is a *gnuplot* window (or an inline *gnuplot* window if the *wxMaxima* wrapper is used, i.e., *wxplot2d*). In this section we present some plot options related the *gnuplot* window.

Selecting the *gnuplot* terminal (*gnuplot_term* and gnuplot_out_file options)

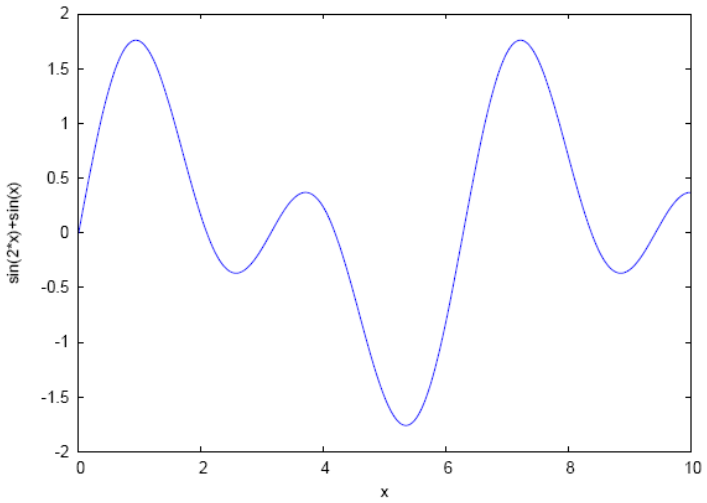
To change the *gnuplot* terminal use the option [*gnuplot_term*, *terminal_type*], where *terminal_type* can take the values:

- *default* output is displayed in a *gnuplot* window or inline
- *dumb* output is displayed in an old-fashioned dumb terminal
- *ps* generates a default PostScript file *maxplot.ps*, unless a filename is given using the option *gnuplot_out_file*
- other *png, jpeg, svg*

The following examples illustrate the use of this option. First, we show the result of a *ps* option using the default PostScript file name:

```
plot2d(sin(x)+sin(2*x),[x,0,10],[gnuplot_term,ps]);
C:/Users/Gilberto E. Urroz/maxplot.ps
```

I used *Adobe Acrobat Distiller* to convert this *ps* file into a *pdf* file, from which I extracted the following plot:



To produce a PostScript file with a specific name, one could use, for example:

```
plot2d(sin(x)+sin(2*x),[x,0,10],[gnuplot_term,ps],[gnuplot_out_file,"C:/Users/Gilberto
  E. Urroz/plot1.ps"]);
C:/Users/Gilberto E. Urroz/plot1.ps
```

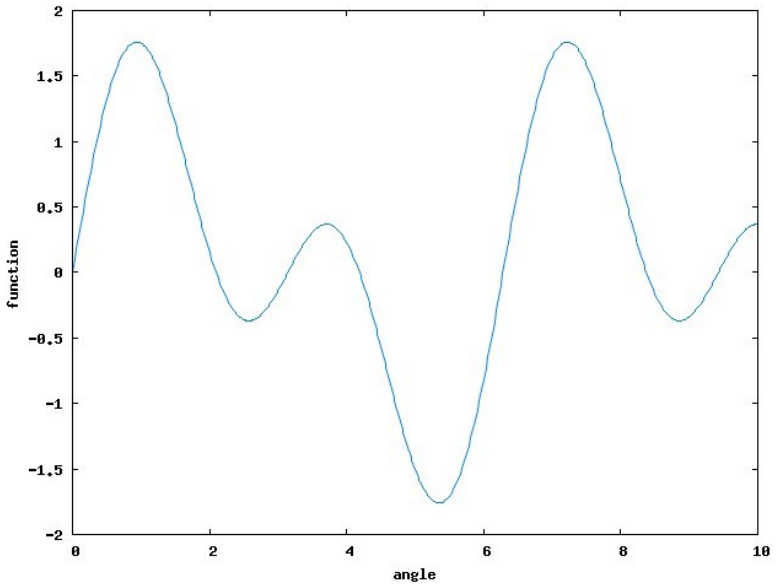
An example of a dumb terminal output, including an output file, is shown below:

```
(%i70)  plot2d(sin(x)+sin(2*x),[x,0,10],[gnuplot_term,dumb],[xlabel,"angle"],[ylabel,"function"]);
9
    2 ++------------+-------------+-------------+-------------+------------++
      +    $$$$     +             +             +      $$$$   +             +
  1.5 ++  $$  $$                                        $$  $$               ++
      |  $$     $                                      $$    $$               |
    1 ++$$      $$                                     $$      $$             ++
      | $        $$                                    $        $$            |
  0.5 +$$         $$                                   $$         $$          ++
      |$           $$         $$$$$                    $           $$         $$$
    0 $+            $$      $$$   $$                  $            $$      $$$++
      |              $$   $$$      $$                 $             $$   $$$   |
      |               $$$$$         $$               $               $$$$$     |
 -0.5 ++                             $$           $$                          ++
      |                              $$           $                           |
   -1 ++                             $$         $$                           ++
      |                               $$      $$                             |
 -1.5 ++                              $$    $$                               ++
      +             +              +       $$$$  +             +             +
   -2 ++------------+-------------+-------------+-------------+------------++
      0             2             4             6             8             10
                                      angle

(%o70)  C:/Users/Gilberto E. Urroz/maxplot.dumb
```

The following example shows a plot which is send to a *jpeg* file (*myPlot1.jpg*):

```
plot2d(sin(x)+sin(2*x),[x,0,10],[gnuplot_term,jpeg],[gnuplot_out_file,"C:/Users/Gilberto
  E. Urroz/myPlot1.jpg"],[xlabel,"angle"],[ylabel,"function"]);
C:/Users/Gilberto E. Urroz/myPlot1.jpg
```

The resulting file can be opened with any graphics software (e.g., *Paint* in *Windows Vista*). The result was copied into this document as shown below:



The *gnuplot_preamble* option

The *gnuplot_preamble* option is set to an empty string "" as default. The string can be replaced by a string containing a number of *gnuplot* commands to set up a number of plot format options. These options may include logarithmic scales, location of legend key, placing zero axes, and location of *x* and *y* ticks, as illustrated in the following examples.

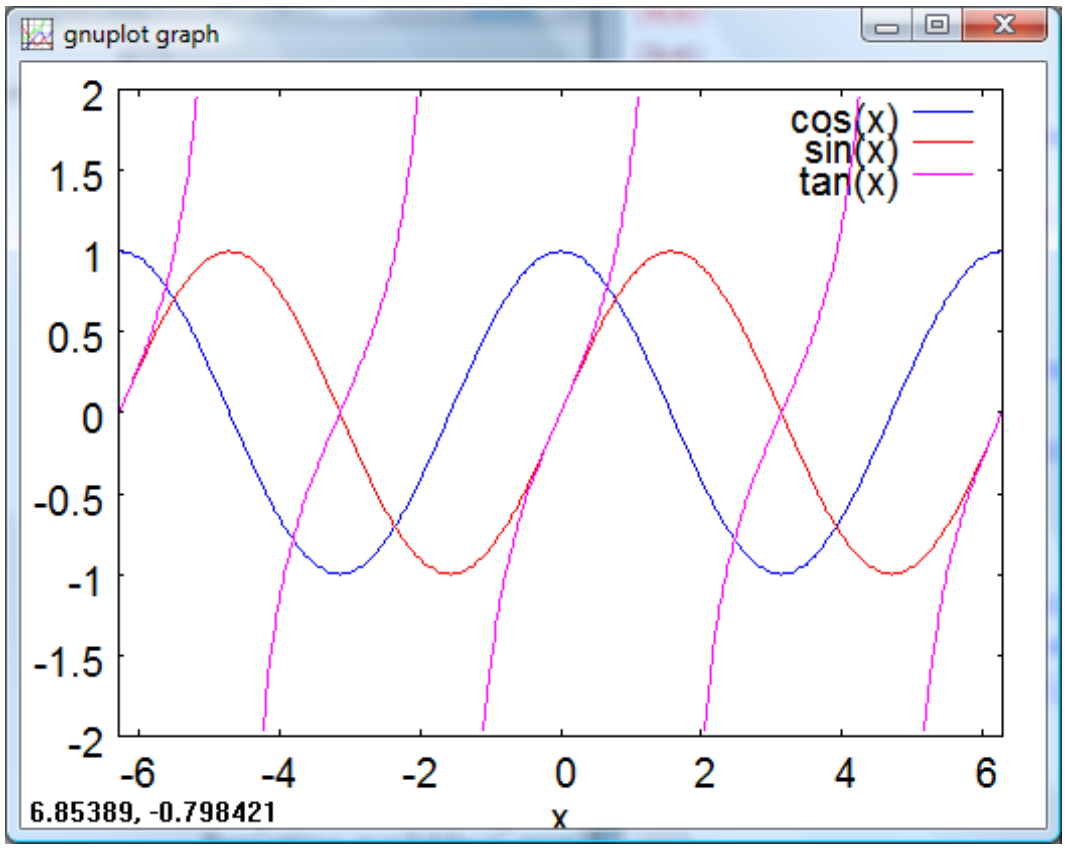
Zero axes – The first case presented shows the setting of zero axes in the plot

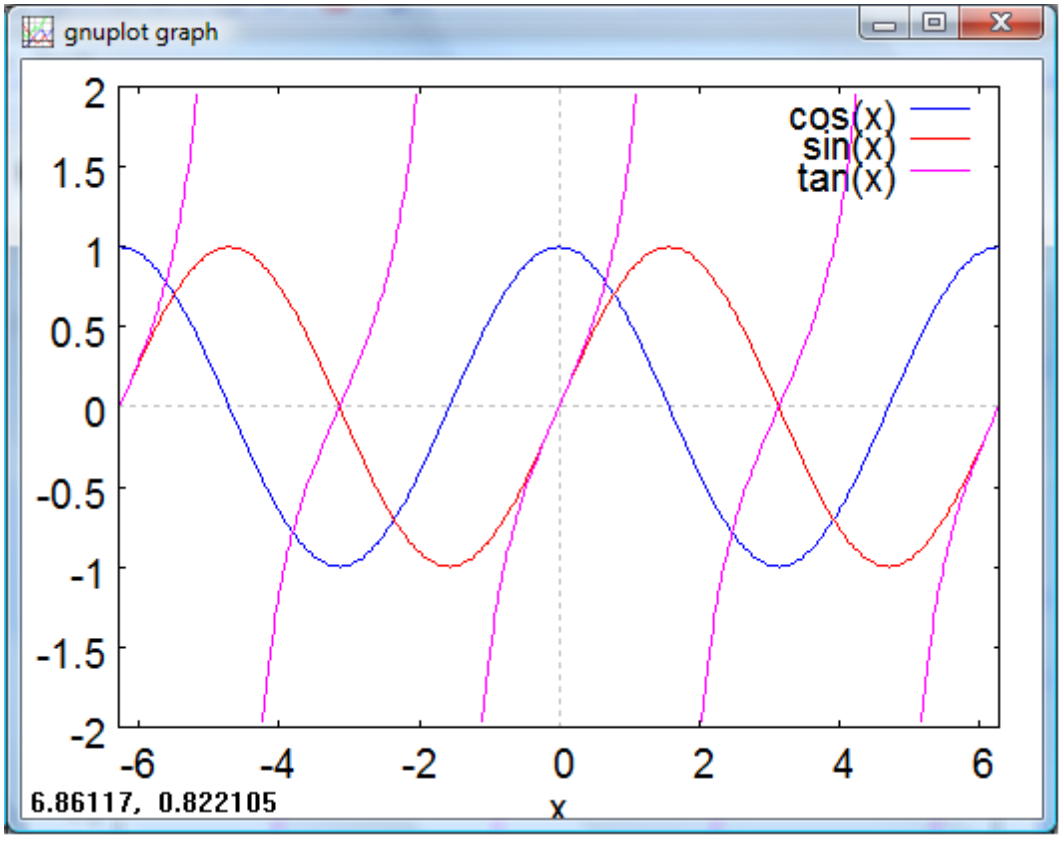
- Default case:

```
plot2d ([cos(x),sin(x),tan(x)],[x,-2*%pi,2*%pi],[y,-2,2])$
```

- Preamble set for zero axis in both x and y:

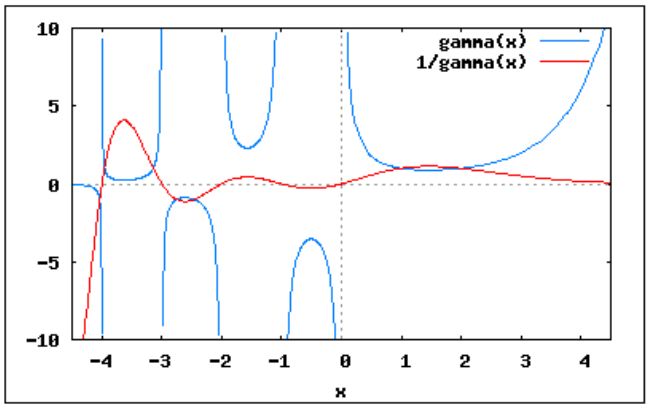
```
my_preamble: "set xzeroaxis; set yzeroaxis" $

plot2d ([cos(x),sin(x),tan(x)],[x,-2*%pi,2*%pi],[y,-2,2],
[gnuplot_preamble,my_preamble])$
```



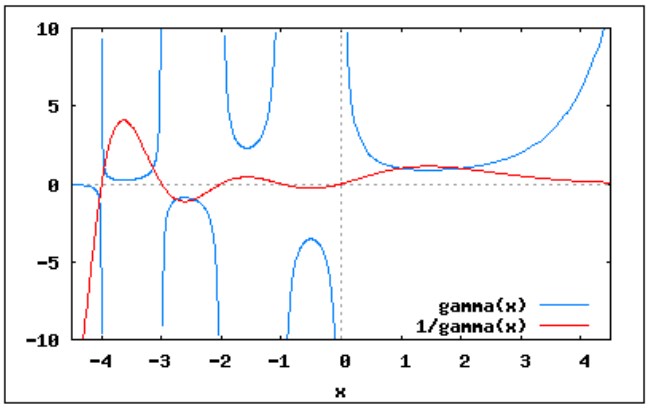


Location of legend key – Using the *gnuplot_preamble* option with value "set key bottom", "set key left", or "set key left bottom" allows changing the location of the legend key. The default location is the upper right corner. The following plots illustrate the four possible corner locations.

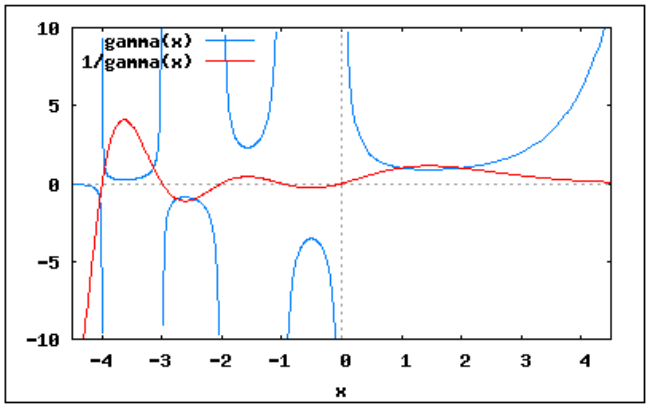
```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10]);
```



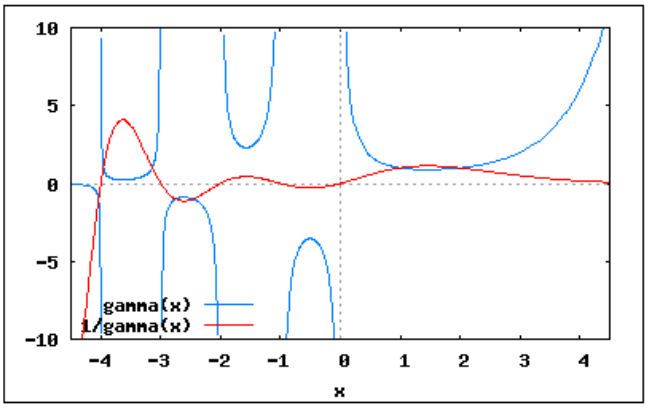

```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,"set key bottom"]);
```




```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,"set key left"]);
```

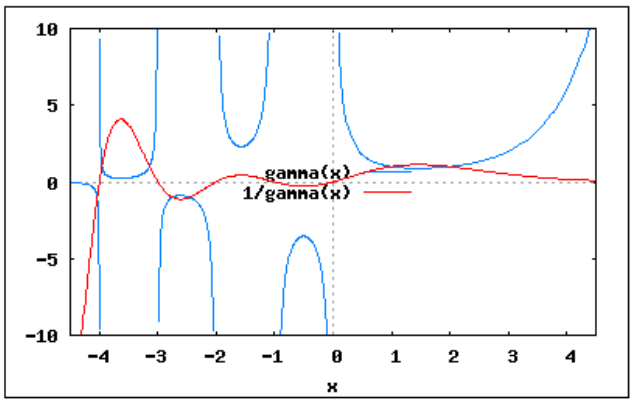



```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,"set key left bottom"]);
```

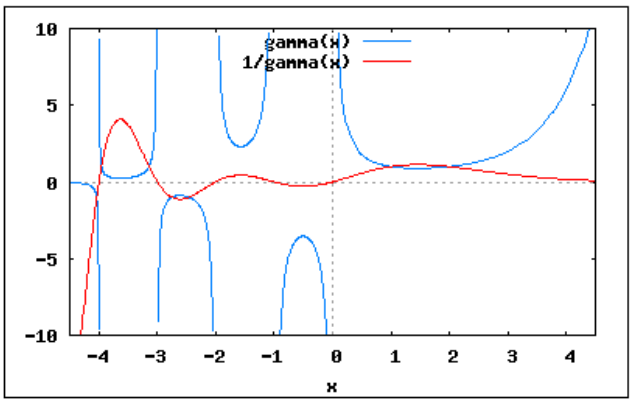


Other options for the location of the legend key are "set key center", "set key top center", "set key bottom center", "set key left center", and "set key right center". Two of these cases are illustrated below.

```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,"set key center"]);
```

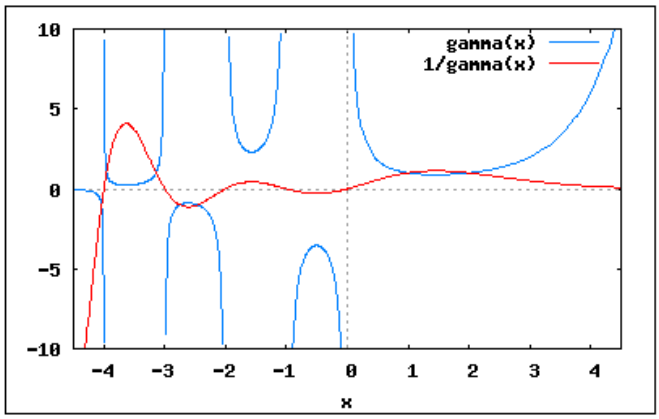



```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,"set key top center"]);
```

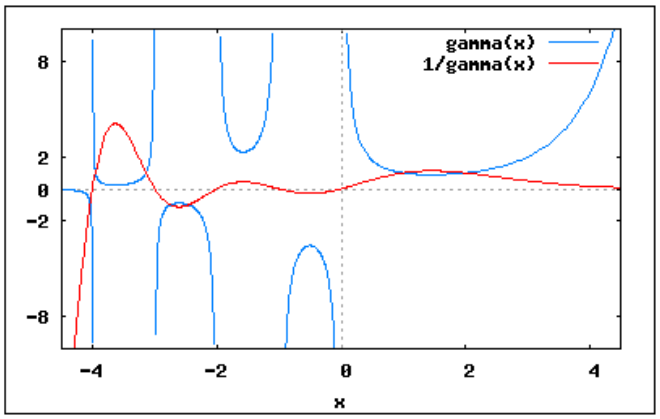


Setting tics on axes – Use the options "*set xtics(...)*" and "*set ytics(...)*" to set the tick marks in the *x* and *y* axes. The following example illustrates the use of this option. The figure to the left is the default setting for ticks, while the figure to the right shows user-defined settings for those tics.

```
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10]);
```



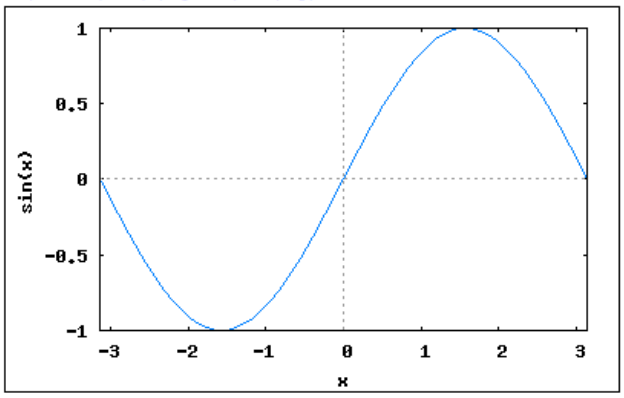

```
preamble:"set xtics(-4,-2,0,2,4);set ytics(-8,-2,0,2,8)"$
wxplot2d([gamma(x),1/gamma(x)],[x,-4.5,4.5],[y,-10,10],
[gnuplot_preamble,preamble]);
```



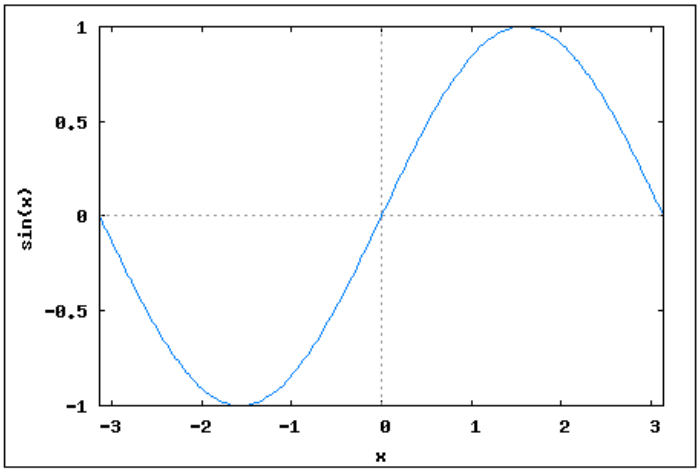
Controlling the *wxplot2d* inline size

The size of an inline plot is controlled by the variable *wxplot_size*. By default, this value is the list [400,250], representing the horizontal and vertical sizes of the inline plot window in pixels. To change the size of the inline plot, therefore, redefine the variable *wxplot_size* accordingly. Try the following examples:

```
wxplot_size;
[ 400 , 250 ]
wxplot2d(sin(x),[x,-%pi,%pi]);
```




```
wxplot_size : [450,300];
wxplot2d(sin(x),[x,-%pi,%pi]);
```



An example of a two-dimensional plot

The specific energy in an open channel is defined as the energy per unit weight measured from the channel bottom. The equation that defines specific energy is:

```
EnerEq : E = y + V^2/(2*g);
```

$$E = \frac{V^2}{2g} + y$$

where E = specific energy, V = flow velocity, g = acceleration of gravity, and y = flow depth.

The flow velocity, in turn, is defined in terms of the unit discharge (or discharge per unit width), q, as $V = q/y$, and replaced into the energy equation as:

```
EnerEqq : subst(V=q/y,EnerEq);
```

$$E = y + \frac{q^2}{2 g y^2}$$

Next, we replace the values $q = 10\ ft2/s$, and $g = 32.2\ ft/s2$, and define a function *E(y)* using the right-hand side (rhs) of the equations *EnerEqQ*:

```
E(y) := subst([q=10, g=32.2],rhs(EnerEqQ));
```

$$E(y) := \text{subst}([q = 10, g = 32.2], \text{rhs}(EnerEqQ))$$

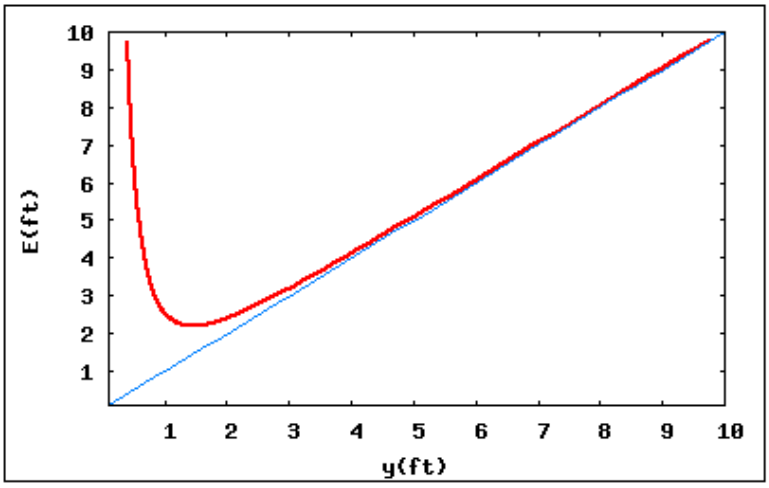
To see the expression for the function *E(y)* use:

```
E(y);
```

$$y + \frac{1.5527950310559}{y^2}$$

The plot of this function is shown together with the line $E = y$.

```
wxplot2d([E(y),y],[y,0.1,10],[y,0.1,10],[style,[lines,2,2],
[lines,1,1]],[xlabel,"y(ft)"],[ylabel,"E(ft)"],[legend,false]);
```



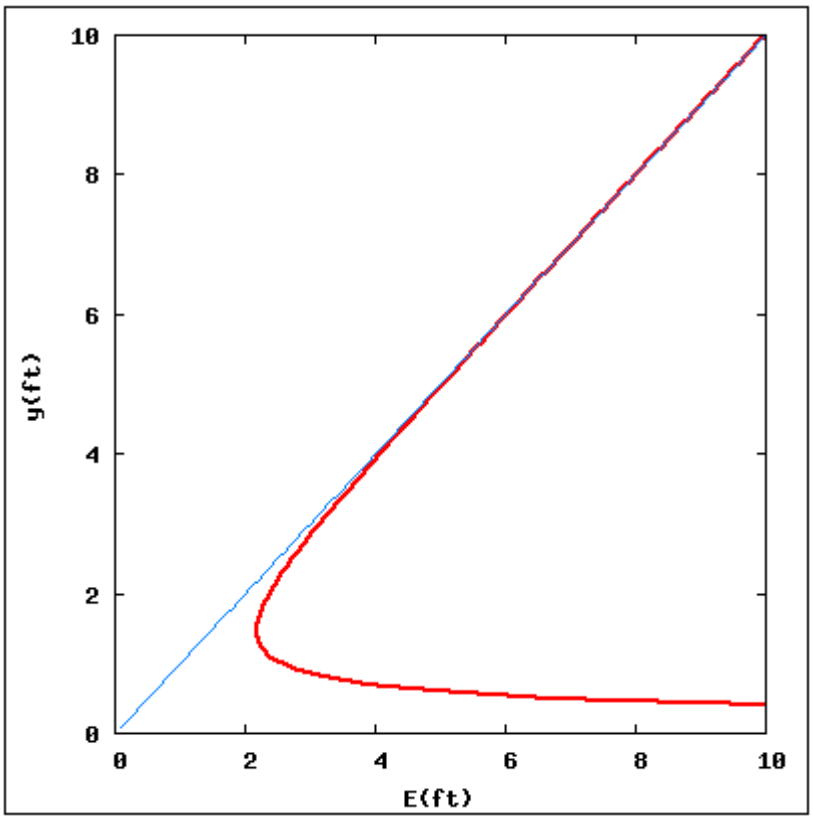
Using lists for producing plots

The graph of specific energy shown above is typically shown with the axes switched. One possible way to produce such a plot is to create a well-populated list of values of *y* and then generate the corresponding list of values of *E(y)*. To create a systematic list of values we use function *makelist*. For example, for the function *E(y)* used above, we can generate lists of values of *y* (*yList*) and *E* (*EList*), as follows:

```
E(y) := y+1.5527950310559/y^2 $
yList : makelist(0.1*k,k,1,100)$
EList : makelist(E(yList[k]),k,1,100) $
```

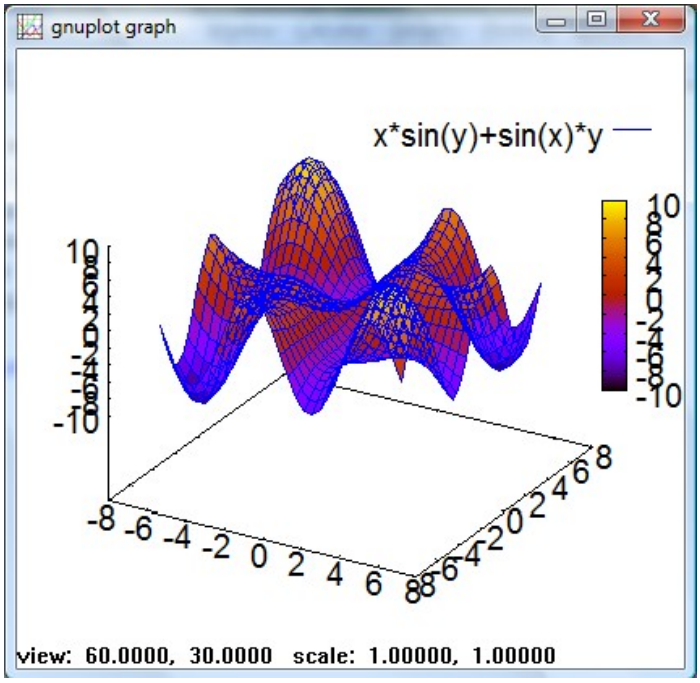
To produce the plot, we first change the inline plot size and then use the following *plot2d* command:

```
wxplot_size : [400,400] $
wxplot2d([[discrete,EList,yList],[discrete,yList,yList]],
[x,0,10],[y,0,10],[style,[lines,2,2],[lines,1,1]],
[xlabel,"E(ft)"],[ylabel,"y(ft)"],[legend,false]);
```

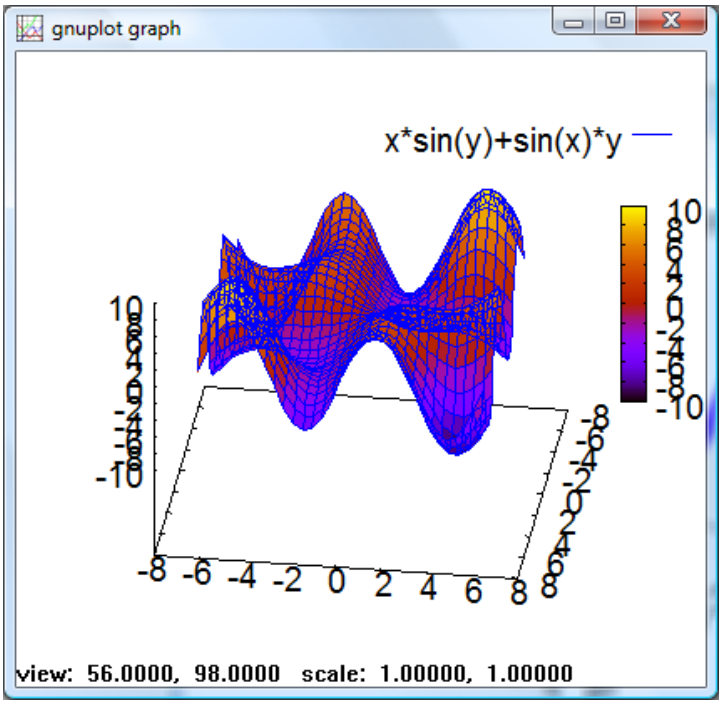


The *plot3d* command

The *plot3d* command can be used to produce a surface plot of a function of the form *z* = *f(x,y)*, e.g., `plot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],[y,-2*%pi,2*%pi]);` produces the plot:



If you click on the *gnuplot graph* window and then hold the left-mouse button while moving the mouse it is possible to rotate the view of the three-dimensional plot, e.g.,

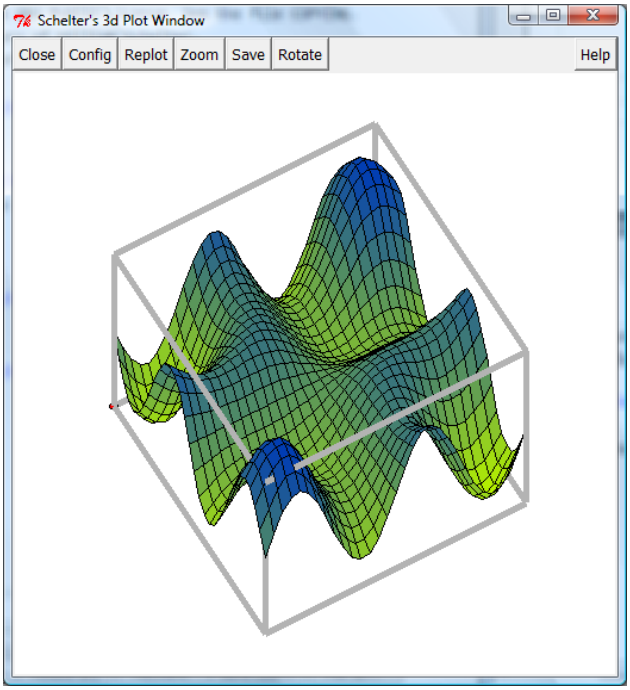


Using the openmath window

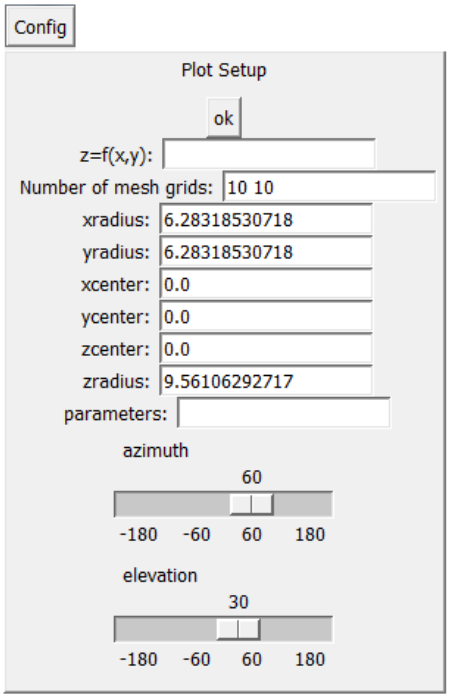
An alternative display for *plot3d* (also available for *plot2d*) is the plot format *openmath*. The following example shows the use of *openmath*:

```
plot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],[y,-2*%pi,2*%pi],[plot_format,openmath]);
```

The resulting graph is shown in a *Tk Schelter's 3d Plot Window* as shown below:



The *openmath* window provides a number of buttons that allows the user to modify the plot format. The options for the *Config* button are shown in the figure to the right.



The *Zoom* button prepares the plot to zoom in or out. The instructions for zooming are as follows:

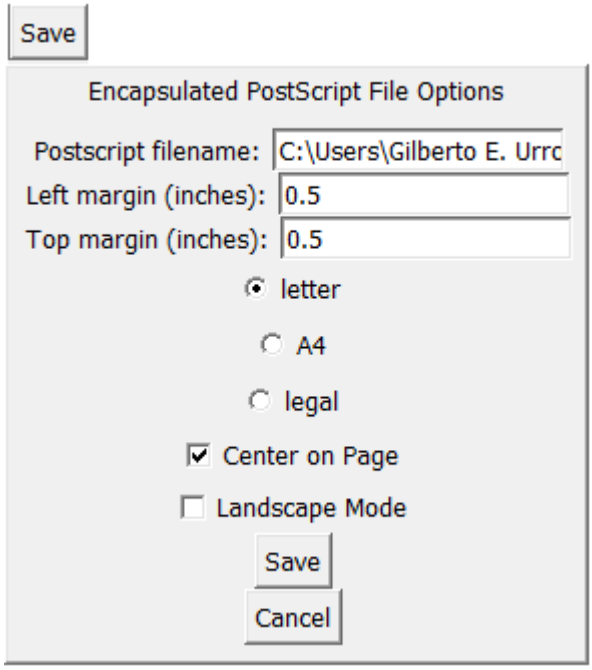
- Click to Zoom
- Shift+Click to Unzoom

The *Save* button allows the user to save the plot in a file (see next page).

The Replot button replots the graph.

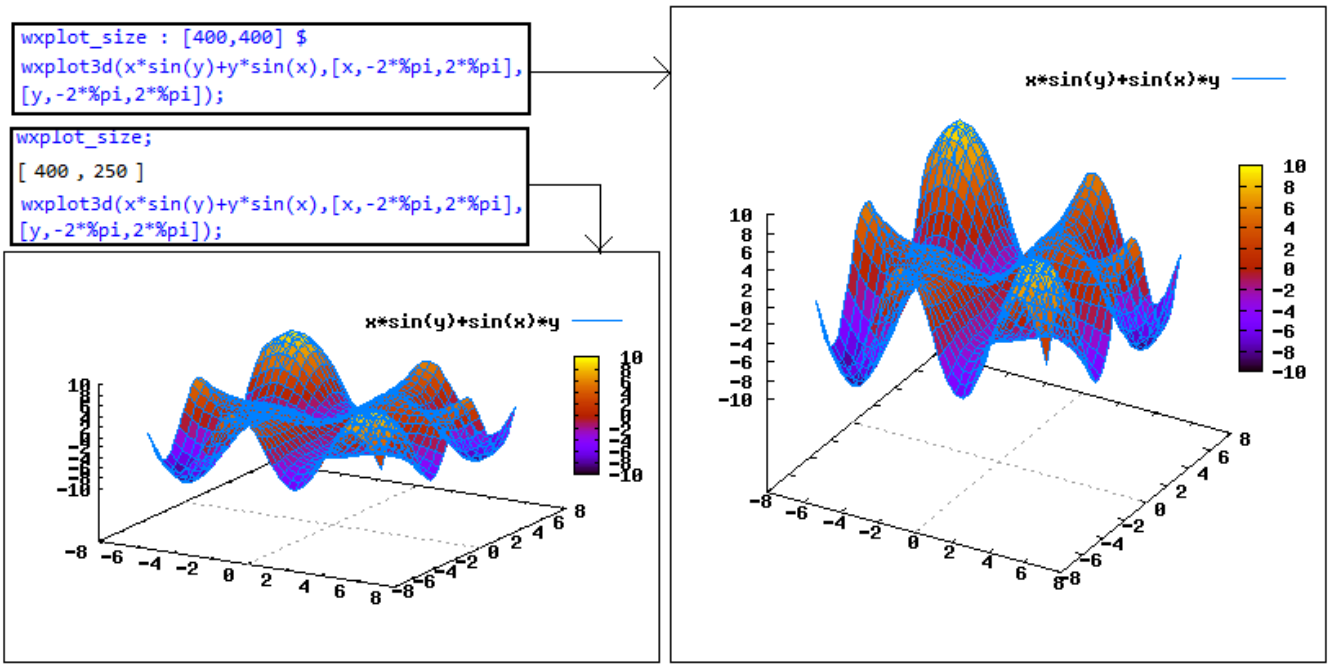
The Rotate button prepares the plot for rotation using the mouse. The Azimut and Elevation angles will be shown.

The Close button closes the figure.



Inline three-dimensional plot with *wxplot3d*

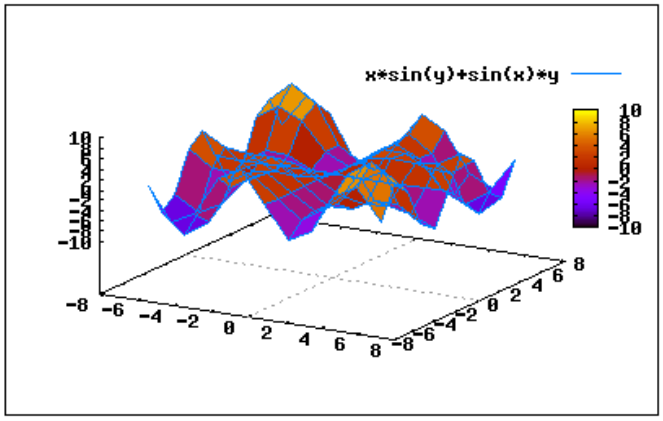
Use function *wxplot3d* to produce an inline three-dimensional plot as illustrated in the figure below. The figure to the left uses the default inline window size of [400,250, while the figure to the right shows a larger inline window of size [400,400].



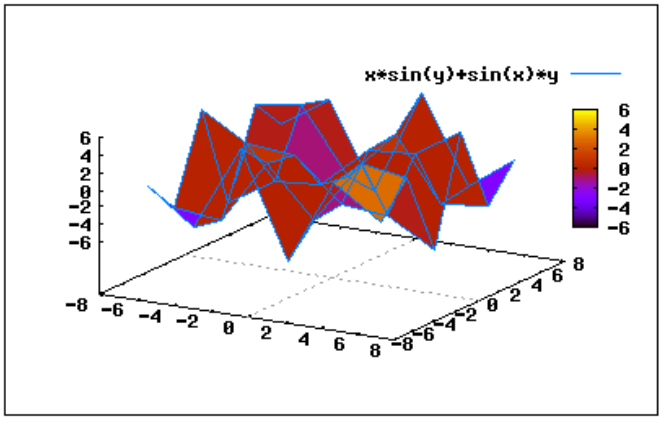
The *grid* option

The *grid* option determines the number of points used in the *x* and *y* variables in the plot. The default value is [grid,30,30].

```
wxplot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],
[y,-2*%pi,2*%pi],[grid,10,10]);
```



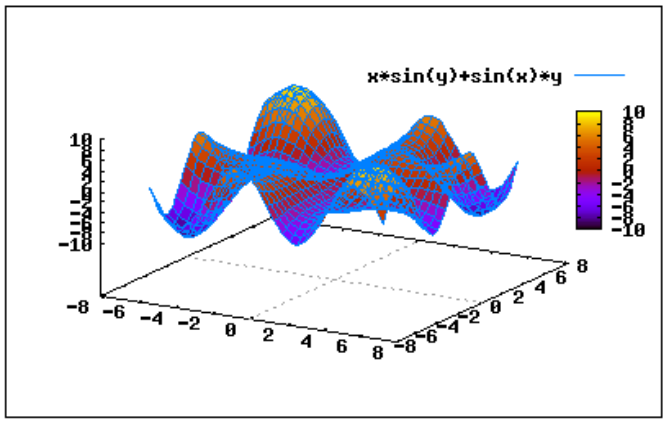

```
wxplot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],
[y,-2*%pi,2*%pi],[grid,5,5]);
```



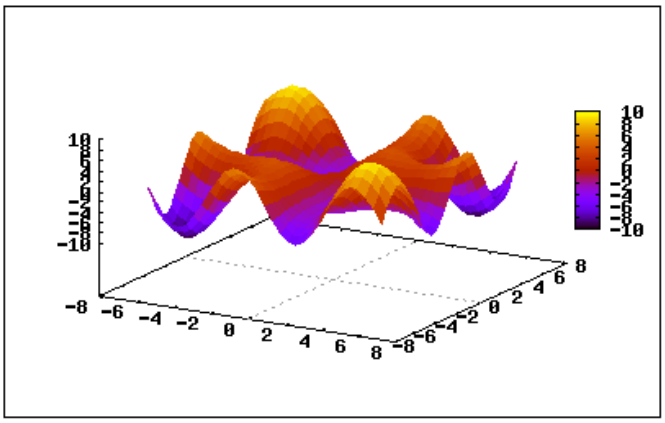
Removing the mesh

To remove the mesh from the surface use the option [gnuplot_preamble, “unset surface”]. The following figure shows the default surface format to the left, and the surface without the mesh to the right.

```
wxplot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],
[y,-2*%pi,2*%pi]);
```



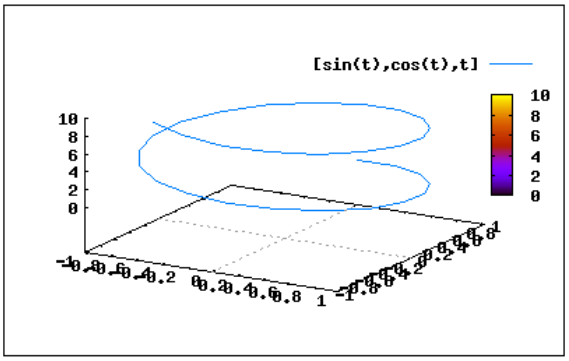

```
wxplot3d(x*sin(y)+y*sin(x),[x,-2*%pi,2*%pi],
[y,-2*%pi,2*%pi],[gnuplot_preamble,"unset surface"]);
```



Plotting a three-dimensional parametric curve

To plot a parametric curve provide a list of three functions [x(t),y(t),z(t)], and two variable ranges, one being the parameter for the curve, in this case, [t, 0, 10], and the second one being a dummy variable, e.g., [s, 0,10]. Only the range [t,0,10] is used in the calculation of the curve, but *plot3d* will not work unless the ranges for two independent variables are given.

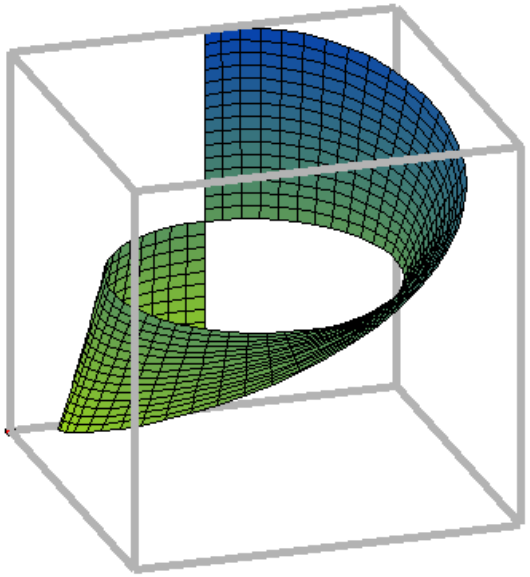
```
wxplot3d([sin(t),cos(t),t],[t,0,10],[s,0,10]);
```



Plotting a parametric surface

The approach followed to produce a parametric surface is the same as in a parametric curve, except that the functions provided are of the form [*x(u,v), y(u,v), z(u,v)*], with ranges for variables *u* and *v*. The following parametric surface is produced using the option [plot_format,openmath]:

```
plot3d([cos(u)*(2+v*cos(u/2)),sin(u)*(2+v*cos(u/2)),
v*sin(u/2)],[u,-%pi,%pi],[v,0,1],[grid,50,15],
[plot_format,openmath]);
```



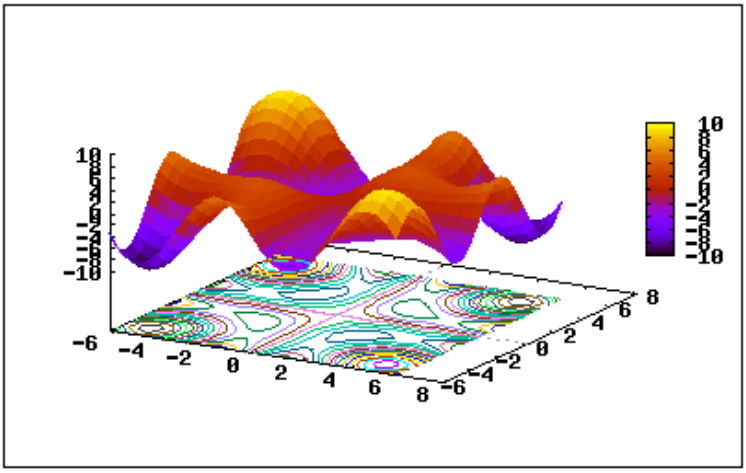
Surface with projected contour plot

Define a string variable:

```
mypreamble : "unset surface; set contour; set cntrparam levels 20; unset key";
```

Then, use the [*gnuplot_preamble,mypreamble*] option.

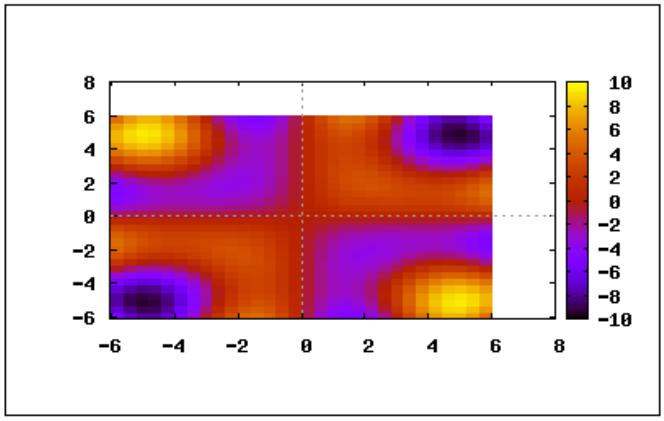
```
wxplot3d(x*sin(y)+y*sin(x),[x,-6,6],[y,-6,6],
[gnuplot_preamble,mypreamble]);
```



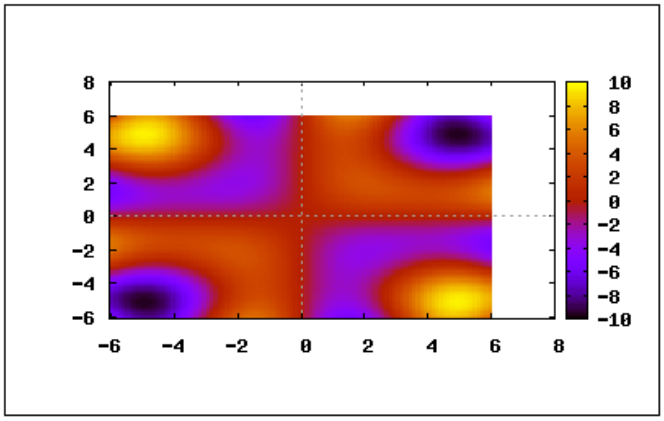
Color map

A color map, somewhat similar to a contour plot, can be generated by using the option [gnuplot_preamble, "set view map; unset surface"]. Increasing the grid size improves the smoothness of the color map.

```
wxplot3d(x*sin(y)+y*sin(x),[x,-6,6],[y,-6,6],
[gnuplot_preamble,"set view map; unset surface"]);
```



```
wxplot3d(x*sin(y)+y*sin(x),[x,-6,6],[y,-6,6],
[gnuplot_preamble,"set view map; unset surface"],
[grid,100,100]);
```



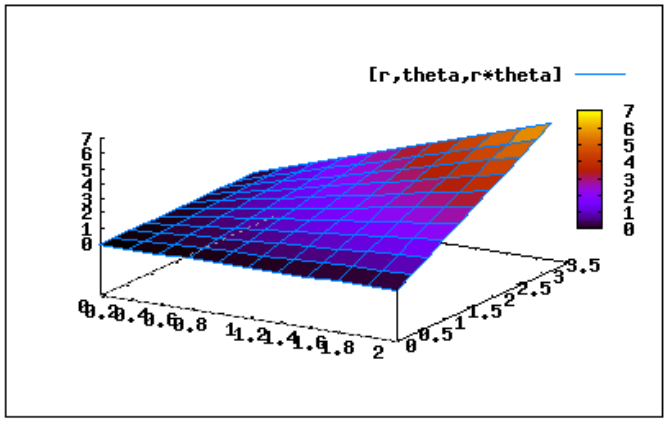
Transformation from polar coordinates

A parametric surface of the form [r,θ,rθ], with parameters [r,0,2] and [θ,0,π], is interpreted as a Cartesian (rectangular) surface, i.e., [$x = r$, $y = \theta$, $z = r\theta$], as in the figure to the left. If the intention is for the functions [r,θ,rθ] to represent the polar coordinates, i.e., [$r = r$, $\theta = \theta$, $z = r\theta$], it is necessary to use the option:

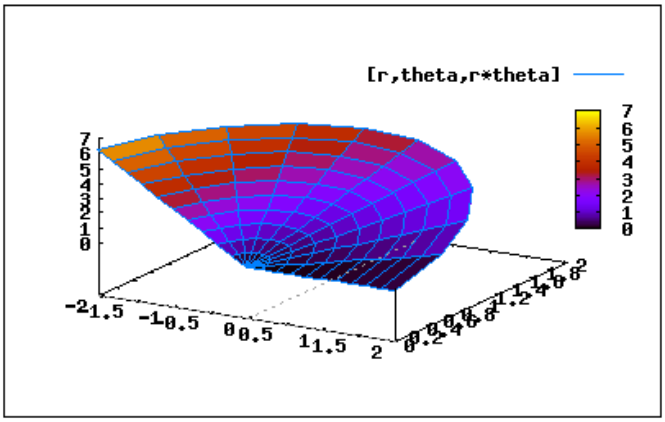
```
[transform_xy, make_transform([r,theta,z],r*cos(theta),r*sin(theta),z).
```

The result is shown in the figure to the right.

```
wxplot3d([r,theta,r*theta],[r,0,2],[theta,0,%pi],
[grid,10,10]);
```

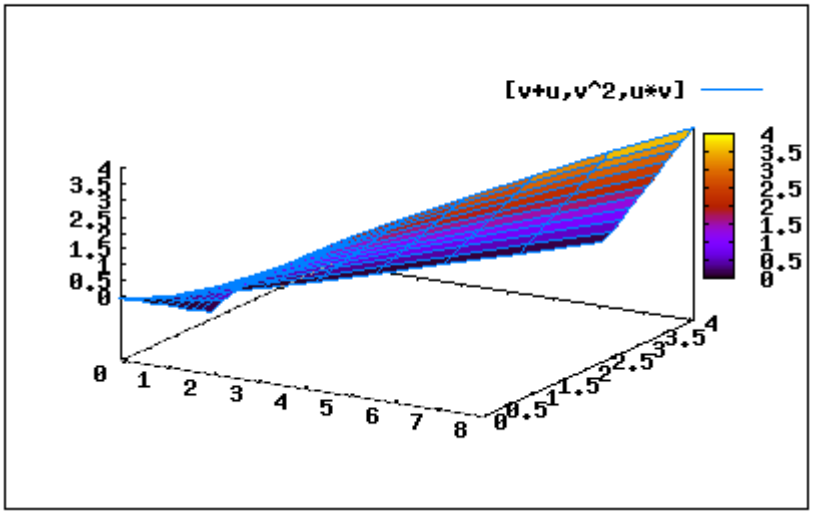



```
wxplot3d([r,theta,r*theta],[r,0,2],[theta,0,%pi],
[grid,10,10],[transform_xy,make_transform([r,theta,z],
r*cos(theta),r*sin(theta),z)]);
```



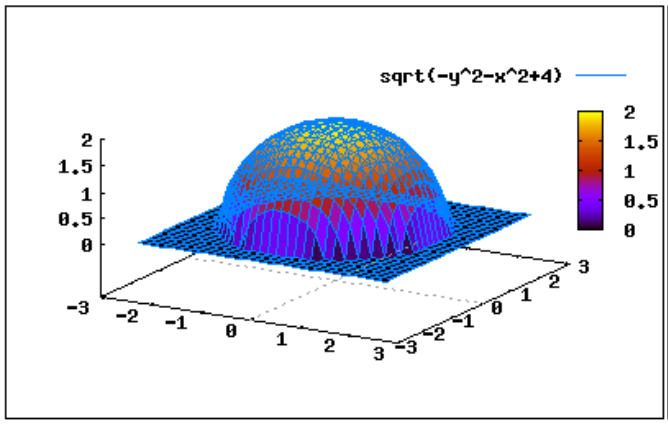
The function *make_transform*, used in the example above, can be used as the value for the option *transform_xy* for other type of transformations, e.g.,

```
wxplot3d([u+v,v^2,u*v],[u,0,2],[v,0,2],[grid,10,10],
[transform_xy,make_transform([u,v,w],u+v,v,w)]);
```

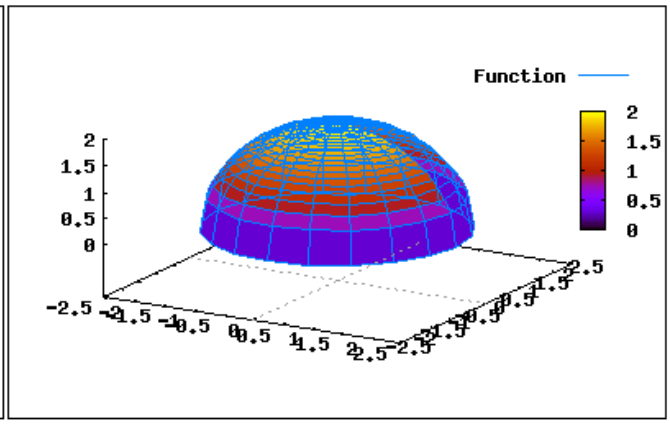


The following figures demonstrate the plotting of a hemisphere using Cartesian coordinates and polar coordinates:

```
wxplot3d(sqrt(4-x^2-y^2),[x,-2.5,2.5],[y,-2.5,2.5]);
```



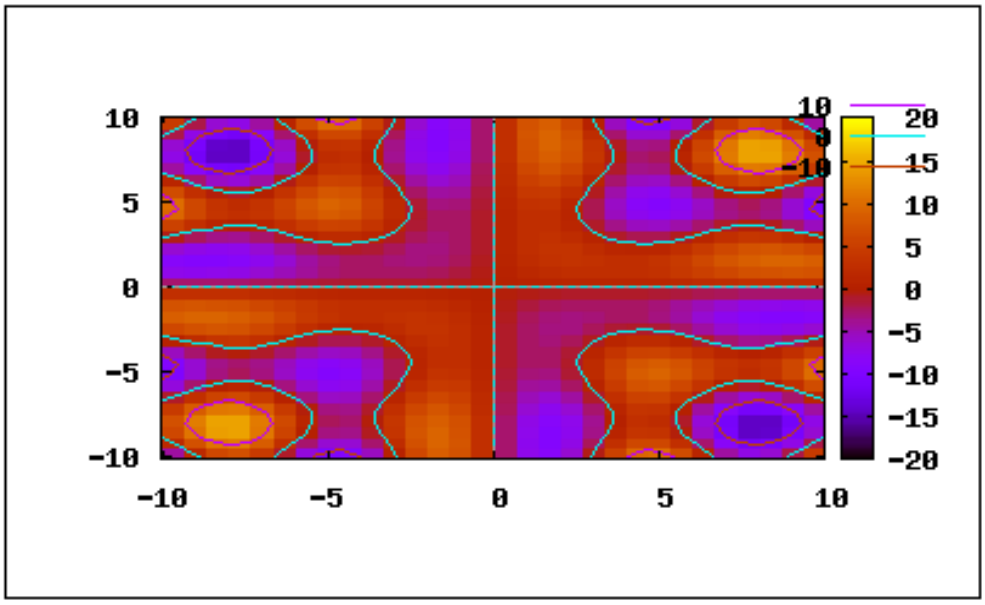

```
wxplot3d([r,theta,sqrt(4-r^2)],[r,0,2],[theta,0,2*%pi],
[grid,20,20],[transform_xy,make_transform([r,theta,z],
r*cos(theta),r*sin(theta),z)]);
```



Contour plots

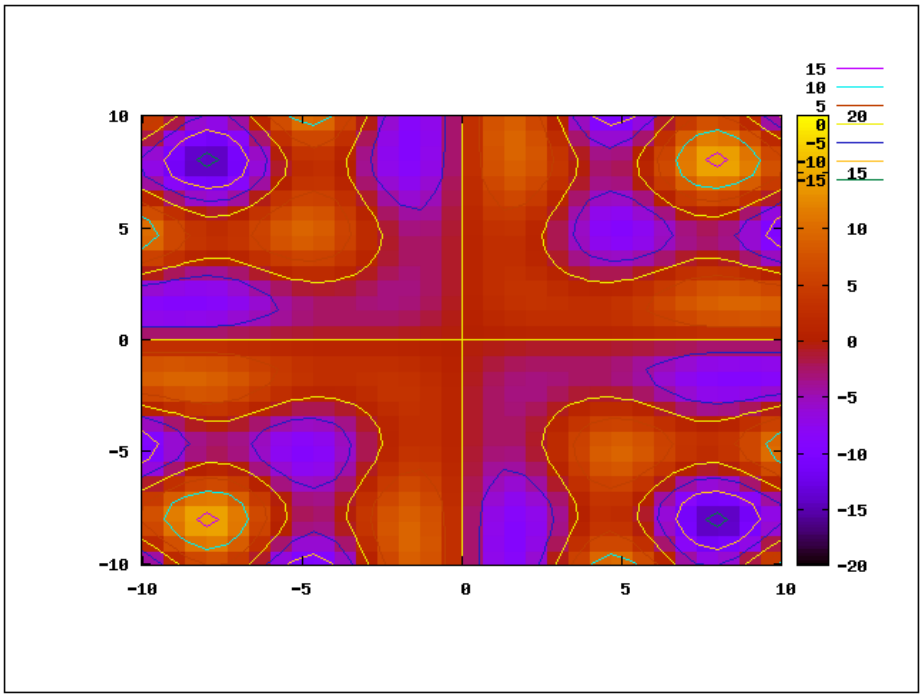
Function *contour_plot*, with similar arguments as *plot3d*, produces contour plots of functions of the form *f(x,y)*. The first example shown uses the default number of parameters:

```
wxcontour_plot(x*sin(y)+y*sin(x),[x,-10,10],[y,-10,10]);
```

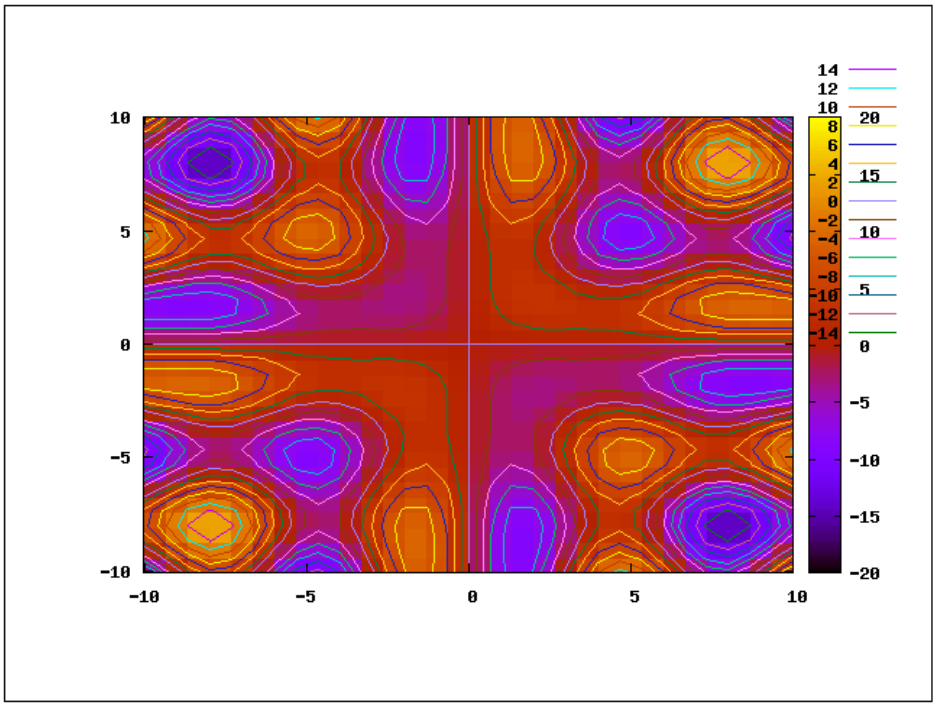


The next two plots show the contourplots corresponding to 10 and 20 contour levels, as specified by the option [*gnuplot_preamble*, *"set cntrparam levels 10"*], etc.

```
wxcontour_plot(x*sin(y)+y*sin(x),[x,-10,10],[y,-10,10],[gnuplot_preamble,"set cntrparam
  levels 10"]);
```

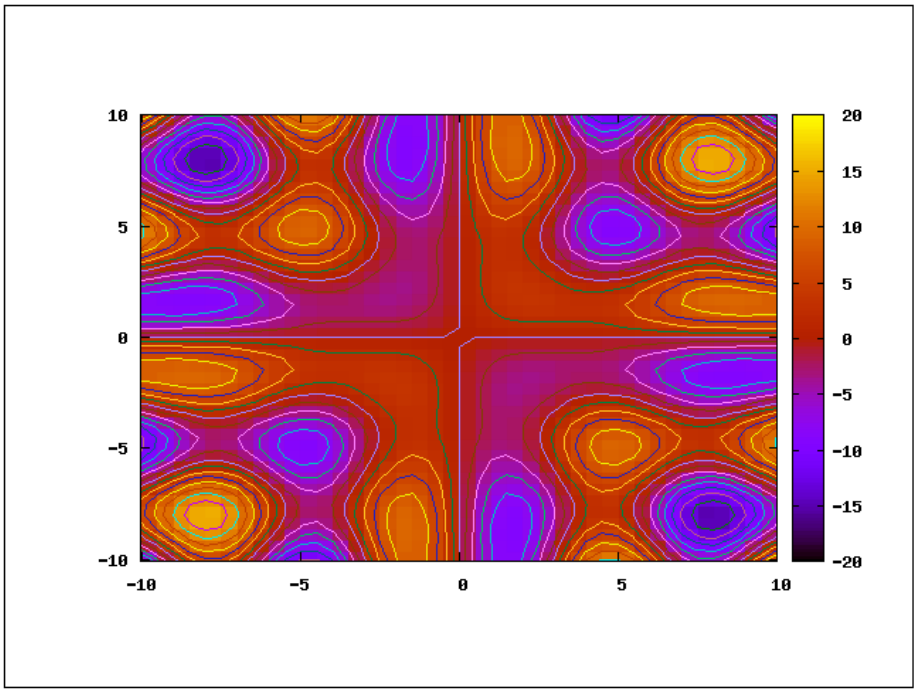


```
wxcontour_plot(x*sin(y)+y*sin(x),[x,-10,10],[y,-10,10],[gnuplot_preamble,"set cntrparam
  levels 20"]);
```



The use of the *grid* option makes for smoother contours, e.g.,

```
wxcontour_plot(x*sin(y)+y*sin(x),[x,-10,10],[y,-10,10],[grid,40,40],[gnuplot_preamble,"set
  cntrparam levels 20;unset key"]);
```

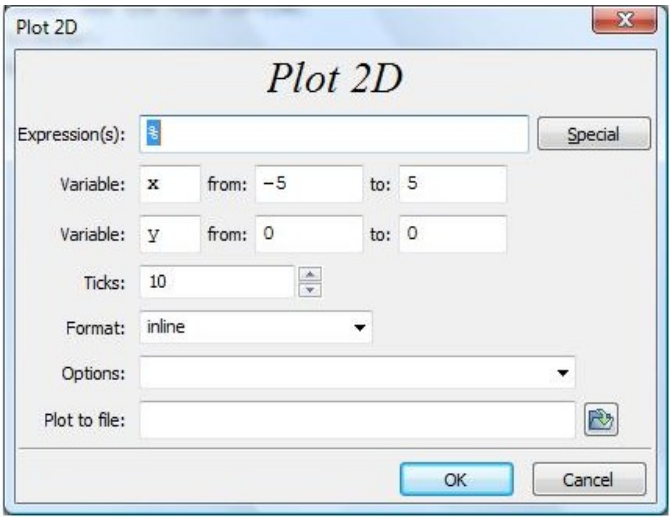


Use of the *Plot 2D*... and *Plot 3D*... menu items/buttons

The *wxMaxima* interface provides quick access to the *Plot2D* and *Plot3D* functions through the menu items *Plotting* > *Plot 2D*... and *Plotting* > *Plot 3D*... , respectively. Alternatively, one can use the *Plot 2D* ... and *Plot 3D*... buttons available in the interface:

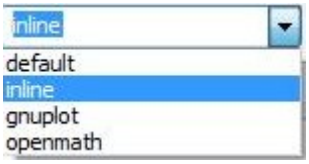
The *Plot 2D...* form

Activating the *Plotting* >*Plot 2D*... menu item produces the following dialogue form:



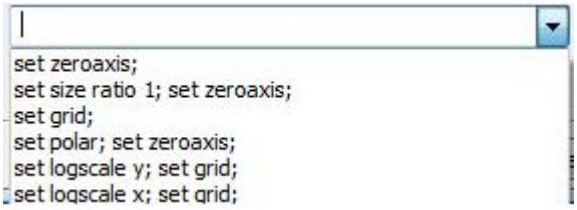
The different entry fields are interpreted as follows:

- The *Expression(s)* field are used to enter an expression in terms of variable *x*, let's refer to it as *f(x)*.
- The suggested range for *x* is *-5<x<5*, however, it can be changed to other values.
- The range of *y*, as in *y = f(x)* can be left unchanged as shown – in which case it is generated automatically --, or it can be selected by the user.
- The *Ticks* option refers to the number of points used to generate the curve to be displayed. Typically this number needs not be changed, except for the case of parametric plots, in which case you may want to choose a large number, say, 50 or larger.
- The *Format* options are the following:



These options are represent the output location for the plot. The *gnuplot* window is the *default* case. The option *inline* means that the graph will be shown in the *wxMaxima* window itselt, and *openmath* is an alternative window.

- The *Options* field include the following choices:



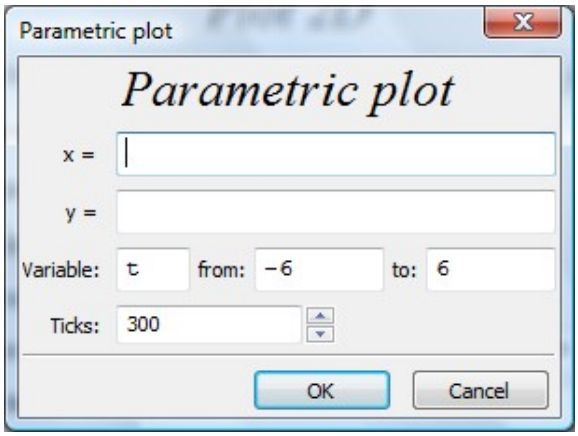
These choices represent the following plot modifications:

○ *set zeroaxis*:	shows *x* and *y* axes intersecting at *(0,0)*
○ *set size ratio 1; set zeroaxis*:	*x* and *y* scales are the same, axes shown
○ *set grid*:	shows a grid in the plot
○ *set polar; set zeroaxis*:	use polar coordinates, axes shown
○ *set logscale y; set grid:*	use logarithmic scale in *y*, show grid
○ *set logscale x; set grid*:	use logarithmic scale in *x*, show grid

- The *Plot to file* field allows the user to enter or select a location where to save the plot as a file. The default format is *.eps*, which represents a *PostScript* file.

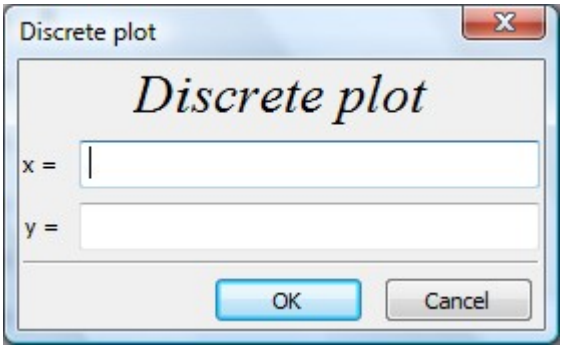
- The *Special* button shown in the dialogue form produces the options:

  ○ parametric plot, which produces the following entry form:



In this form, you enter the parametric functions *x(t)* and *y(t)* into the *x*= and *y*= fields, respectively. You can also select the range of the parameter *t*, or use the default range shown (*-6<t<6*). The *Ticks* field is set to *300* to ensure a smooth curve in the plot.

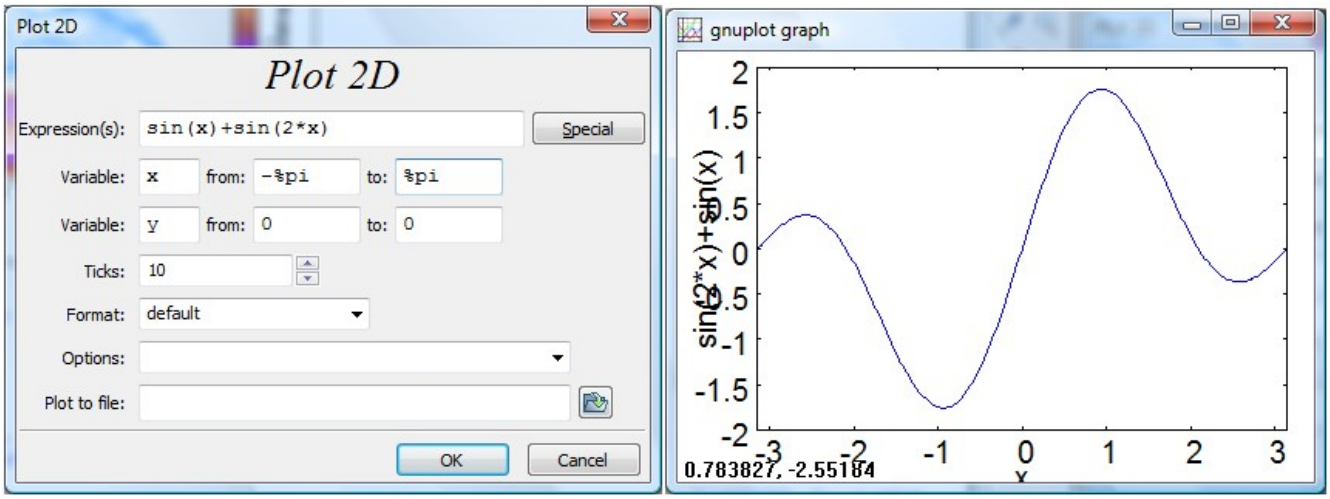
- discrete, which produces the following entry form:



In this form you enter lists of values corresponding to discrete set of points, or you could refer to variables you have already loaded in your *wxMaxima* session which contain the set of values required.

Examples using the *Plot 2D... form*

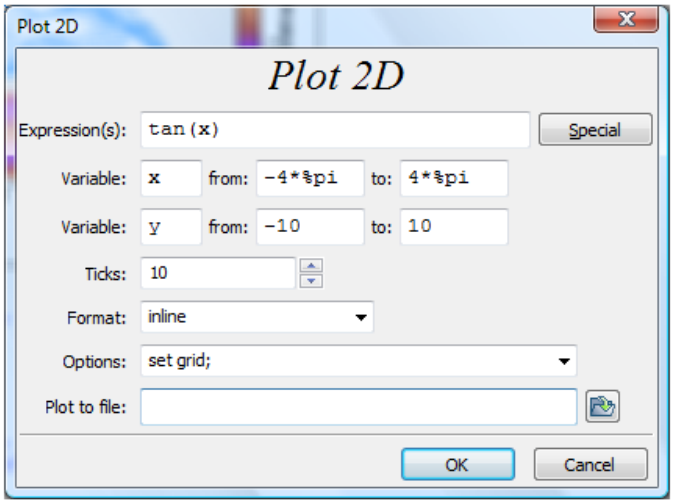
The following examples show uses of the *Plot 2D...* dialogue form. The resulting *Maxima* command is shown after each plot:



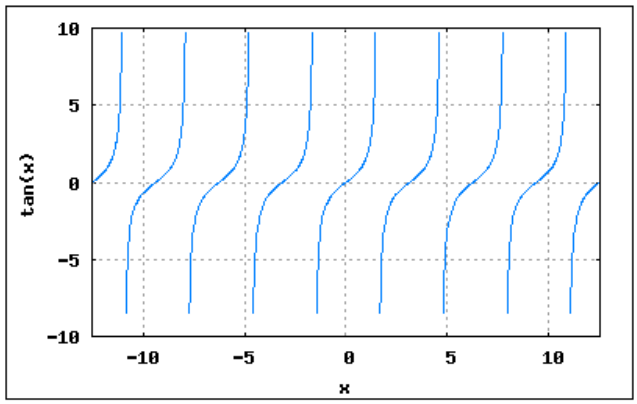

```
plot2d([exp(-x/10)*sin(2*x)], [x,0,5], [gnuplot_term, ps],
 [gnuplot_out_file, "C:/Users/Gilberto E. Urroz/Documents/MAXIMA/plot2d.eps"])$
```

You could enter this command directly in the *wxMaxima* interface and obtain the same plot as shown.

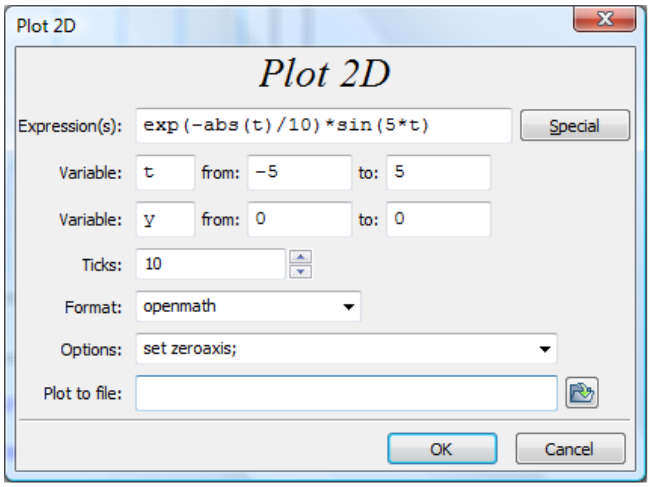
In the following example we plot the function *tan(x)* in an inline plot. Function *tan(x)* diverges to +∞ and -∞ at multiples of $\pi/4$. Therefore, in this case we limit the *y* scale to *-10* < *y* < *10*. A grid is also included.




```
wxplot2d([tan(x)], [x,-4*%pi,4*%pi], [y,-10,10],
 [gnuplot_preamble, "set grid;"])$
```



In the next example we use *openmath* for the output window and show axes intersecting at (*0*,*0*). Notice that the independent variable was changed to *t*:

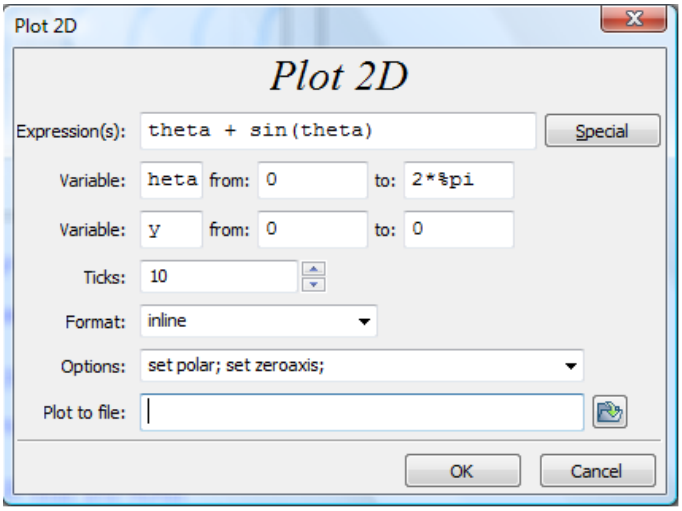


This produces the command:

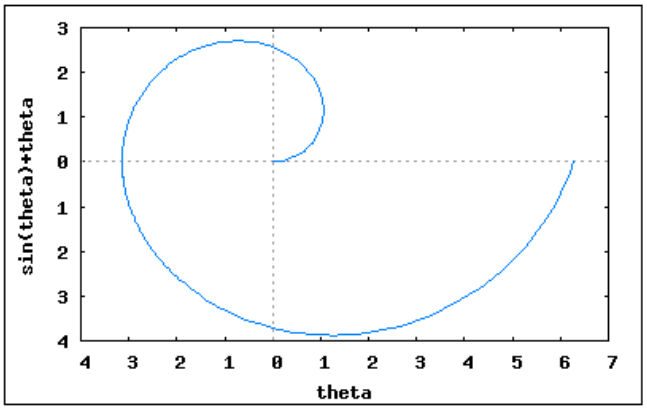
```
plot2d([exp(-abs(t)/10)*sin(5*t)], [t,-5,5],
 [plot_format, openmath],
 [gnuplot_preamble, "set zeroaxis;"])$
```

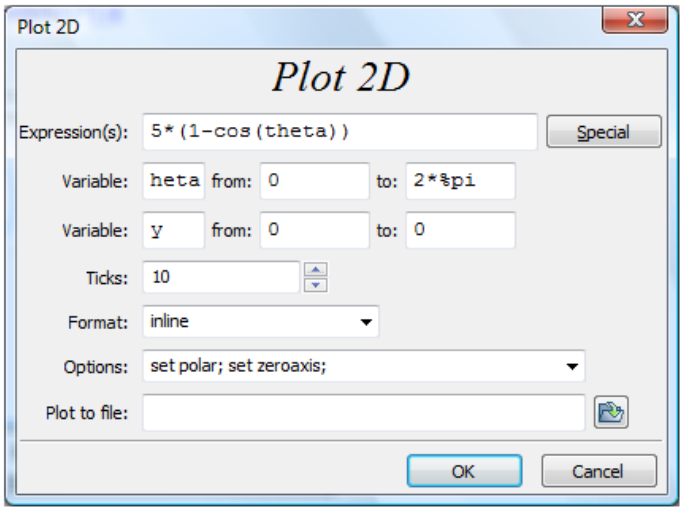
Try this example on your own *wxMaxima* interface to see the *openmath* result.

The following two examples use polar coordinates:

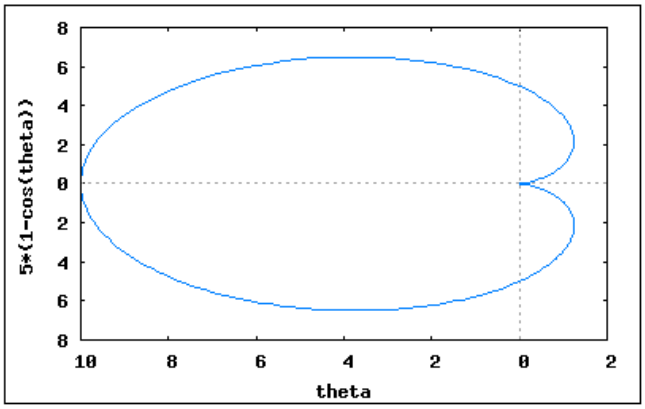



```
wxplot2d([theta + sin(theta)], [theta,0,2*%pi],
 [gnuplot_preamble, "set polar; set zeroaxis;"])$
```

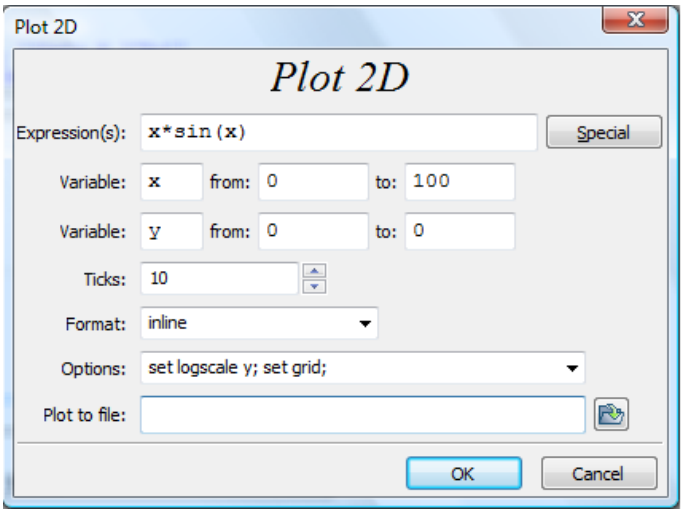




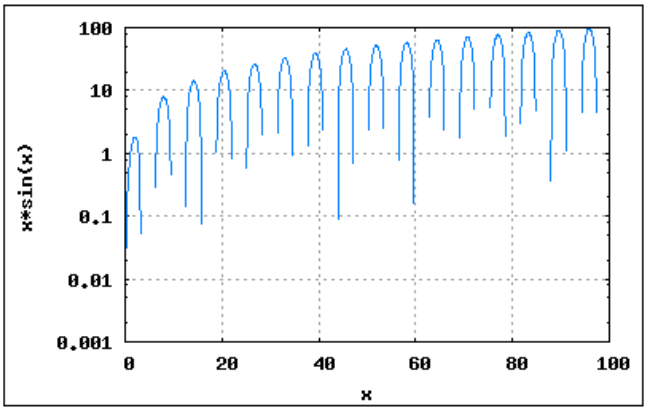

```
wxplot2d([5*(1-cos(theta))], [theta,0,2*%pi],
 [gnuplot_preamble, "set polar; set zeroaxis;"])$
```



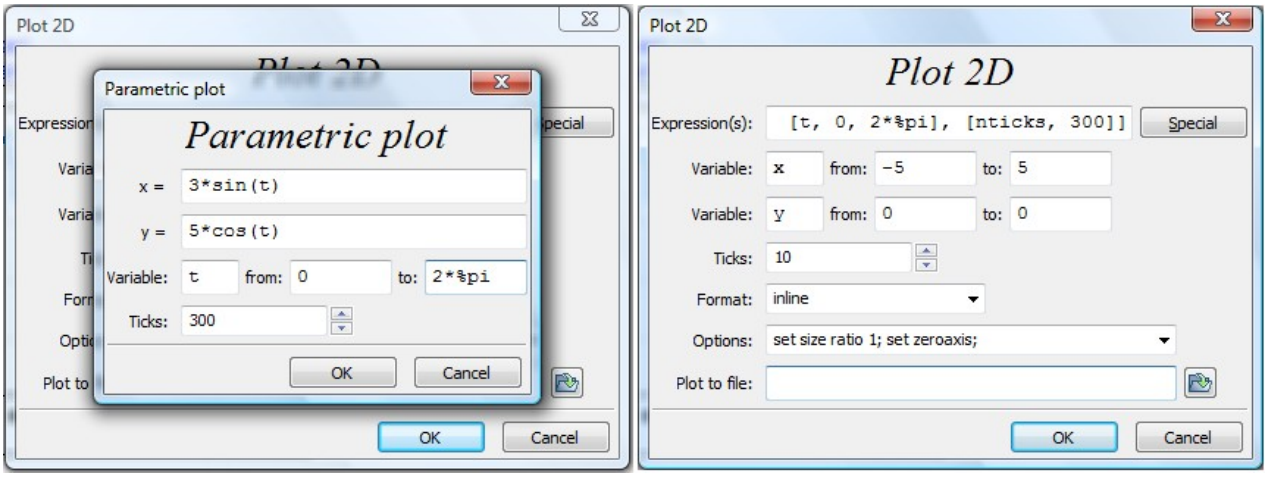
The following example shows a logarithmic *y* scale:



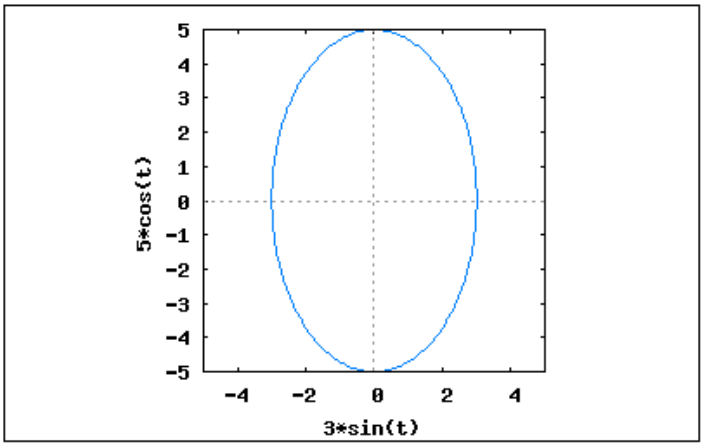

```
wxplot2d([x*sin(x)], [x,0,100],
 [gnuplot_preamble, "set logscale y; set grid;"])$
```



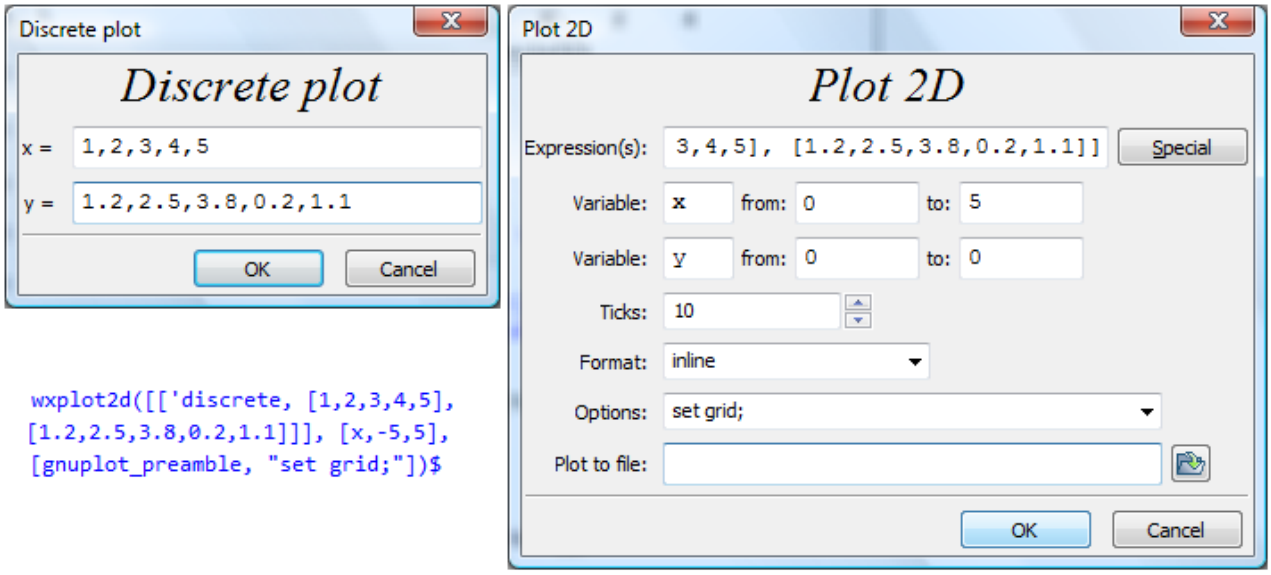
A parametric plot, using the equal scales option, is shown next:




```
wxplot2d([['parametric, 3*sin(t), 5*cos(t), [t, 0, 2*%pi],
[nticks, 300]]], [x,-5,5],
[gnuplot_preamble, "set size ratio 1; set zeroaxis;"])$
```



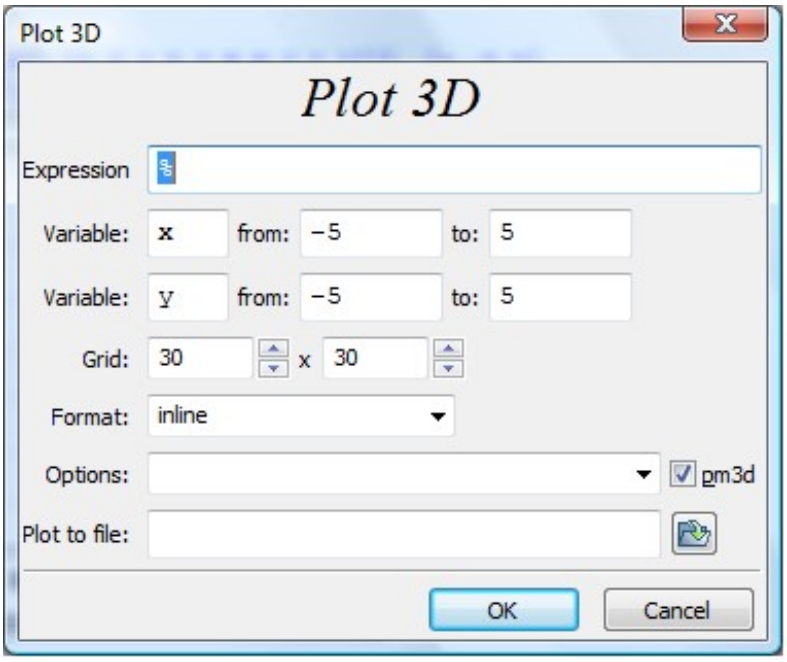
Here's an example of a discrete plot (see the plot in your *wxMaxima* interface):




```
wxplot2d([['discrete, [1,2,3,4,5],
[1.2,2.5,3.8,0.2,1.1]]], [x,-5,5],
[gnuplot_preamble, "set grid;"])$
```

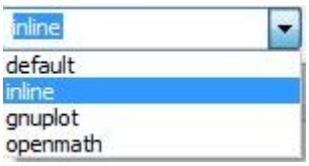
The *Plot 3D...* form

Activating the *Plotting* >*Plot 3D...* menu item produces the following dialogue form:

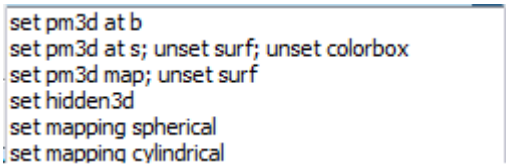


The different entry fields are interpreted as follows:

- The *Expression* field are used to enter an expression in terms of variable *x* and *y,* let's refer to it as *f(x,y)*.
- The suggested ranges for *x* and *y* are *-5<x<5* and *-5<y<5*, however, they can be changed to other values.
- The *Grid* is set, by default, to *30×30*, but it can be changed to other values.
- The *Format* options are the same as in *Plot 2D ...* :



- The *Options* field include the following choices:



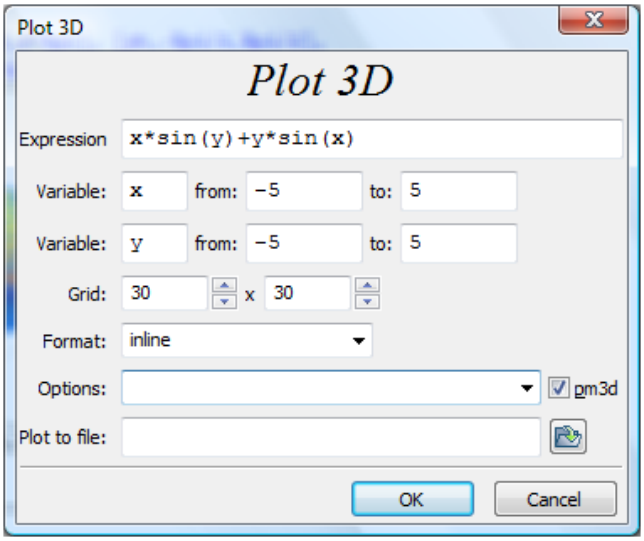
These choices represent the following plot modifications:

- *set pm3d at b*: shows contours at bottom and meshgrid
- *set pm3d at s; unset surf; unset colorbox*: no mesh in surface, no colorbox
- *set pm2d map; unset surf*: produces a contourplot/color map
- *set hidden3d*: no mesh in surface, colorbox shown
- *set mapping spherical:* use spherical coords., $\rho = f(\theta,\phi)$
- *set mapping cylindrical*: use cylindrical coords., $r = f(\theta,z)$

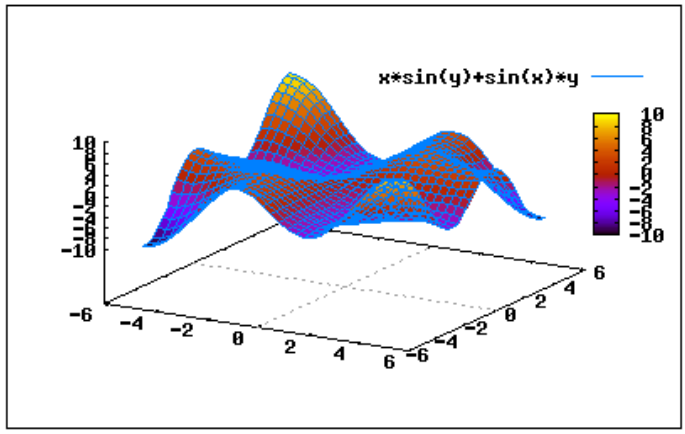
- The *Plot to file* field allows the user to enter or select a location where to save the plot as a file. The default format is *.eps*, which represents a *PostScript* file.

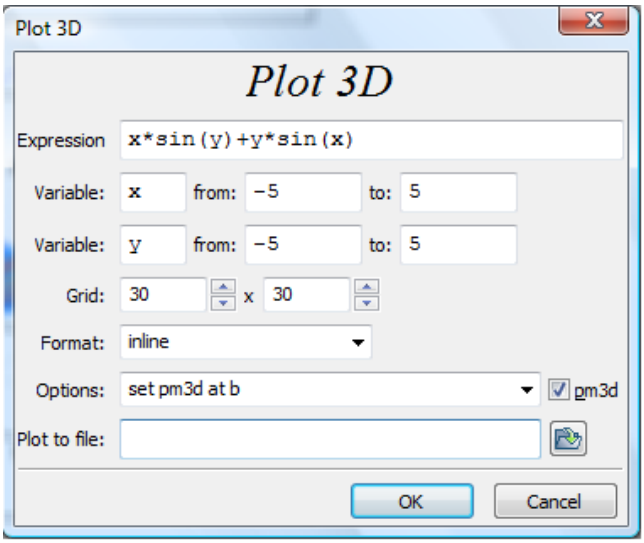
Examples using the *Plot 3D... form*

The following examples show uses of the *Plot 3D*... form. The first four examples show the different options related to the graph format (mesh or no mesh, contour plots, etc.)

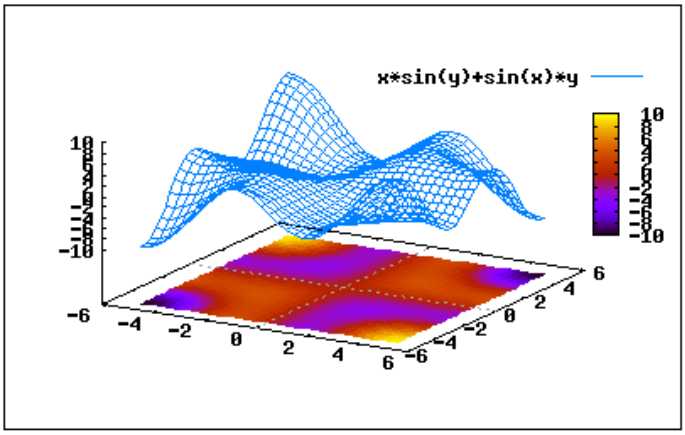


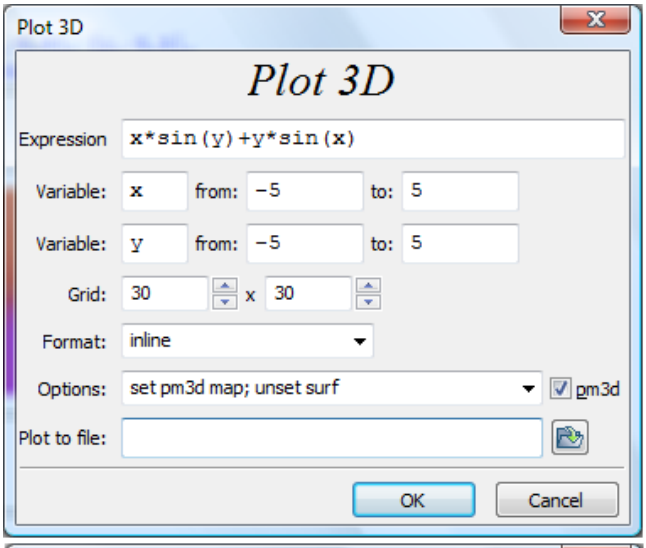

```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5])$
```



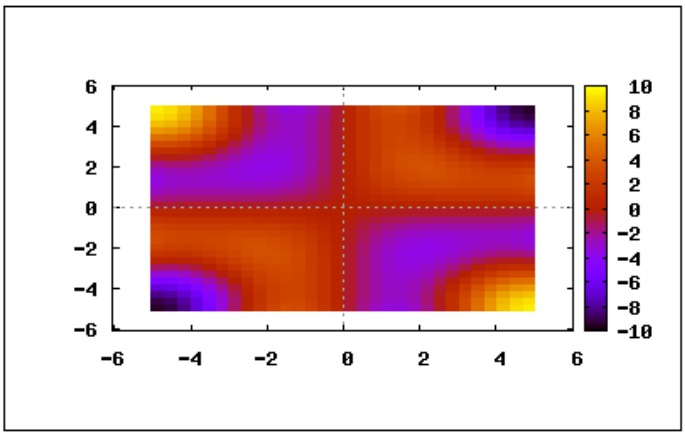


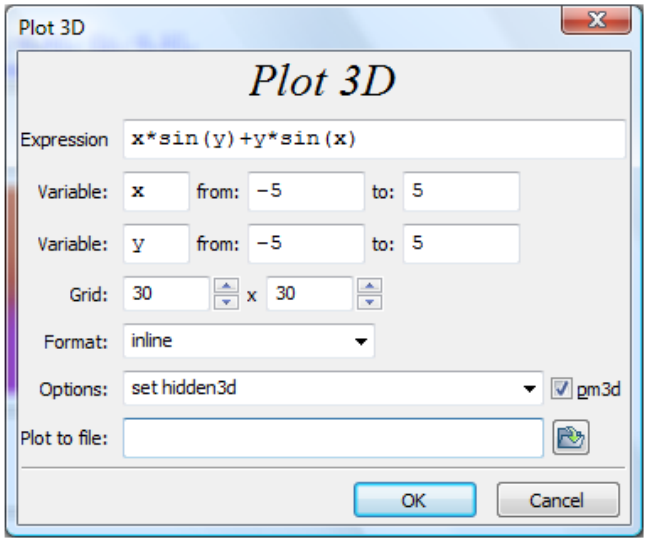

```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5],
 [gnuplot_preamble, "set pm3d at b"])$
```



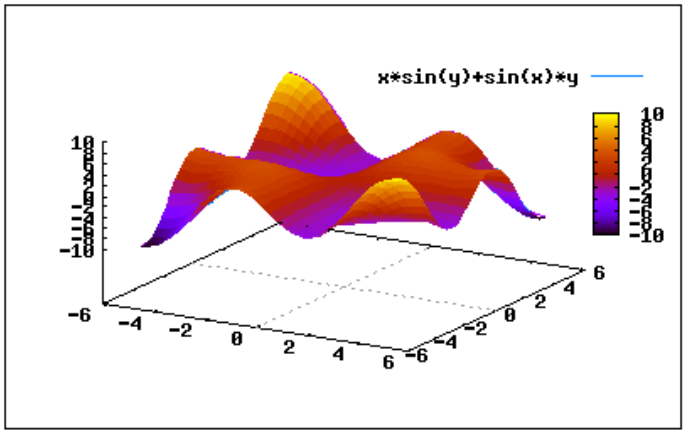



```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5],
 [gnuplot_preamble, "set pm3d map; unset surf"])$
```



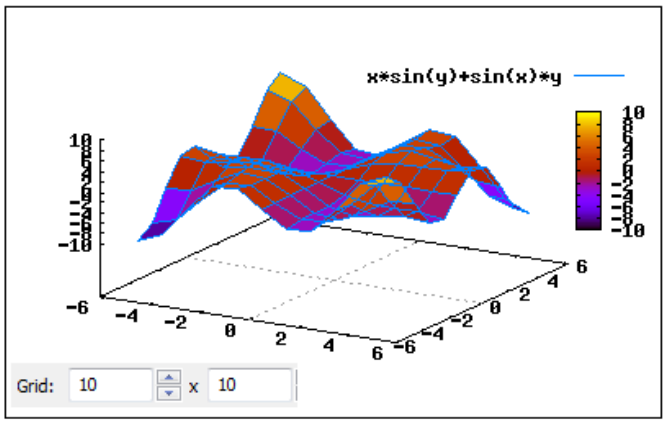



```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5],
 [gnuplot_preamble, "set hidden3d"])$
```

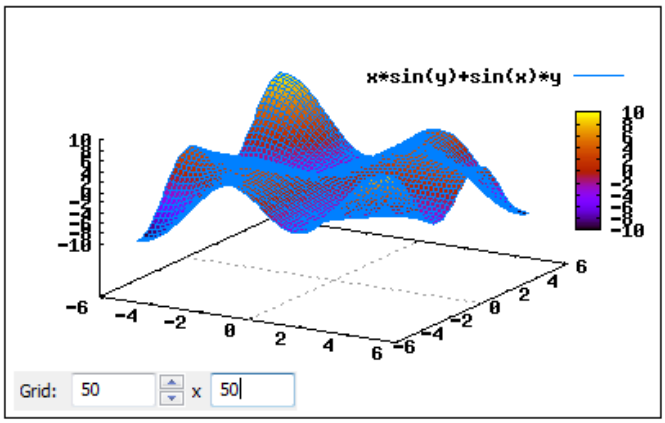


Next, we repeat the first example of *Plot 3D* ..., using grids 10x10 and 50x50:

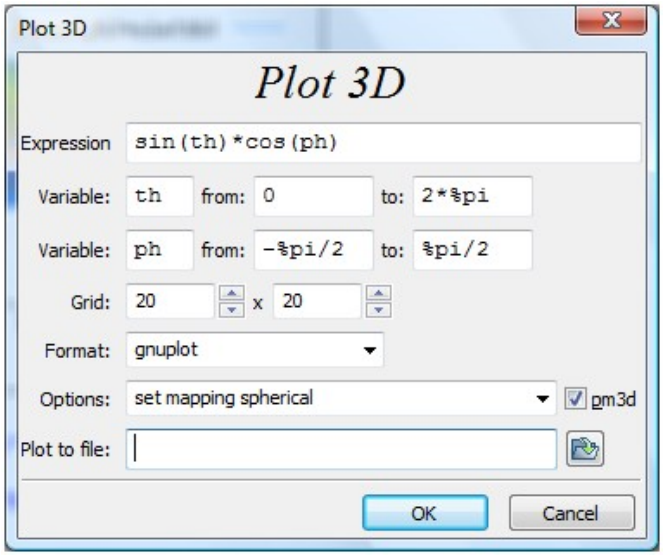
```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5],
 [grid,10,10])$
```



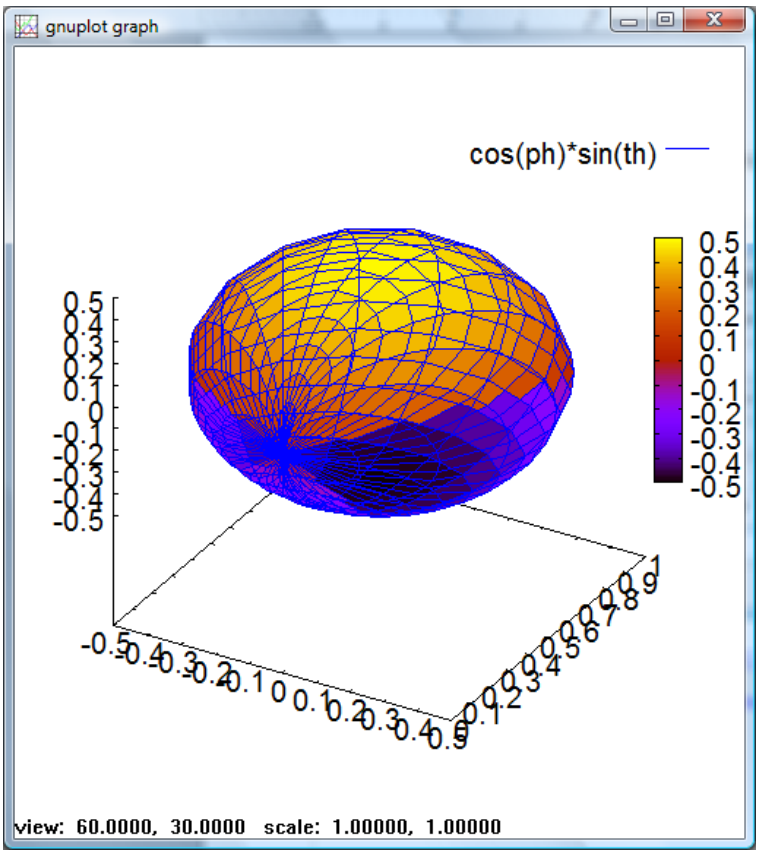

```
wxplot3d(x*sin(y)+y*sin(x), [x,-5,5], [y,-5,5],
 [grid,50,50])$
```



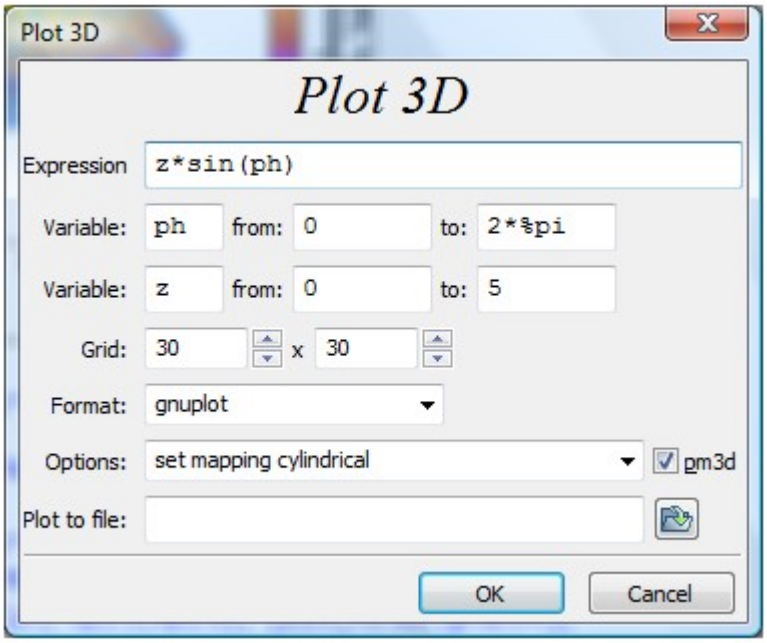
The following example shows the use of spherical coordinates. The output is shown in a *gnuplot* window:



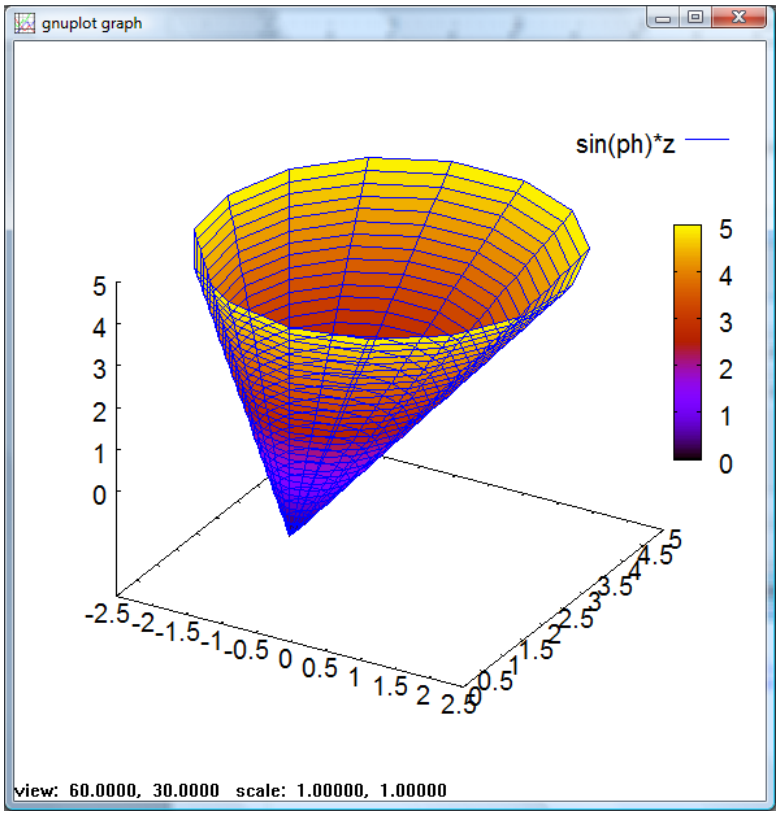

```
plot3d(sin(th)*cos(ph), [th,0,2*%pi], [ph,-%pi/2,%pi/2], [plot_format,gnuplot],
 [grid,20,20],
 [gnuplot_preamble, "set mapping spherical"])$
```



The following example shows the use of cylindrical coordinates. The output is shown in a *gnuplot* window:




```
plot3d(z*sin(ph), [ph,0,2*%pi], [z,0,5], [plot_format,gnuplot],
 [gnuplot_preamble, "set mapping cylindrical"])$
```



The *draw* package

The *draw* package is a contributed package which is described in detail in the following web site:

http://www.telefonica.net/web2/biomates/maxima/gpdraw/

There is a larger variety of graphs available in the *draw* package than the functions presented above. Study the examples in the web site above.

5

Solution to algebraic and non-algebraic equations

In Chapter 1 we introduced the menu items in the *Equations* menu. In this Chapter we re-visit some of those items and present other *Maxima* commands and techniques for the solution of equations. Emphasis will be placed on solving equations from the physical sciences and engineering disciplines.

solve (Equations > Solve ...)

The *solve* function can be used to solve one or more equations. Some of the simplest applications is in the solution of polynomial or fractional equations.

Example 5.1 - The quadratic equation - The classical quadratic equation is solved as follows:

```
(%i1) solve([a*x^2+b*x+c=0], [x]);
```

$$(\%o1)\quad \left[x = -\frac{\sqrt{b^2 - 4ac} + b}{2a}, x = \frac{\sqrt{b^2 - 4ac} - b}{2a}\right]$$

Example 5.2 - The cubic equation - A solution for the cubic equation is also available, however, the output is not included in this Chapter in order to save printing space:

```
(%i2) solve([a*x^3+b*x^2+c*x+d=0],[x]);
```

Example 5.3 - Manipulating solutions - In the following example we solve a specific cubic equation, storing the solutions in a variable called *Sol*:

```
(%i3) Sol : solve([3*x^3+5*x^2+6*x+2=0],[x]);
```

$$(\%o3)\quad \left[x = -\frac{29\left(\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right)}{81\left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right)} + \left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right)\left(-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right) - \frac{5}{9}, x = \left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right)\left(\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right) - \frac{29\left(-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right)}{81\left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right)} - \frac{5}{9}, x = \left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right) - \frac{29}{81\left(3\left(-\frac{7}{2}\sqrt{106} + \frac{37}{729}\right)^{1/3}\right)} - \frac{5}{9}\right]$$

There are three solutions available for this equation. Variable *Sol* is a list of equations. For example, the first term is given by:

```
(%i4)  Sol[1];
```

$$(\%o4)\quad x = -\frac{29\left(\frac{\sqrt{3}\,\%i}{2}-\frac{1}{2}\right)}{81\left(3\left(-\frac{7}{2}\sqrt{106}+\frac{37}{729}\right)^{1/3}\right)}+\left(3\left(-\frac{7}{2}\sqrt{106}+\frac{37}{729}\right)^{1/3}\right)\left(-\frac{\sqrt{3}\,\%i}{2}-\frac{1}{2}\right)-\frac{5}{9}$$

To show a floating-point value for this solution, try the following:

```
(%i5)  float(rhs(Sol[1]));
(%o5)  - 0.55331391615083(0.86602540378444 %i - 0.5)+0.6470552807503(- 0.86602540378444 %i - 0.5)- 0.55555555555556
```

The result still requires collecting the real and imaginary parts of this complex number. One way to accomplish this is to use function *rectform* (*rect*angular *form*) which produces the rectangular or Cartesian form of a complex number:

```
(%i6)  rectform(%);
(%o6)  - 1.039550218436705 %i - 0.60242623785529
```

We could produce the floating-point values of the three solutions as follows:

```
SolF : makelist(rectform(float(rhs(Sol[k]))),k,1,3);
[ - 1.039550218436705 %i - 0.60242623785529 , 1.039550218436705 %i - 0.60242623785529 , - 0.46181419095609 ]
```

NOTE: Function *makelist* has the general form:

```
makelist(F(index),index,start_value,end_value)
```

This function produces a list of values given by `F(index)`. `index` is a variable that serves as the index for the list, and takes integer values from `start_value` to `end_value` in increments of 1.

Example 5.4 - Solution to a fourth-order polynomial equation – We use function *solve* to store the solutions to the fourth-order equation shown below into variable *Sol4*, then use functions *makelist*, *rectform*, and *rhs* to list the floating-point solutions:

```
Sol4 : solve([6*x^4-3*x^3+x^2-x+125=0], [x])$

SolF4 : makelist(rectform(float(rhs(Sol4[k]))),k,1,4);
```

The result, however, is not totally simplified, retaining terms with square roots and producing a very long output. This output is not shown in this Chapter, once again, to save printing space. Interestingly enough, changing the order of functions *rectform* and *float* produces the right result:

```
SolF4R : makelist(float(rectform(rhs(Sol4[k]))),k,1,4);
[ - 1.524062994136233 %i - 1.379306910028965 , 1.524062994136233 %i - 1.379306910028965 , 1.508650793156902 %i +
1.629306910028964 , 1.629306910028964 - 1.508650793156902 %i ]
```

Note on polynomial equations: All the examples used so far correspond to polynomial equations and can be solved using the *Equations* menu options *Roots of polynomial* and *Roots of polynomial (real*), which correspond to the *Maxima* functions *allroots* and *realroots*. Some examples of application of these functions were presented in Chapter 3. These two functions are addressed later in this section using the examples shown above. Next, we use function *solve* to solve non-polynomial equations.

Example 5.5 – The radioactive decay equation. The equation to model radioactive decay is the exponential equation:

```
DEq : q = q0*exp(-t/tau);
```

$$q = q0\ \%e^{-\frac{t}{\tau}}$$

Let's solve this equation for *t*:

```
DSol : solve(DEq,t);
```

$$[\ t = \log\left(\frac{q0}{q}\right)\tau\ ]$$

Suppose we replace the values *q0 = 10 g*, *q = 5 g*, and *t = 2 s*, into this equation, we find that the *half-life* for this material (i.e., the time required for *q0* to be reduced by half) is:

```
subst([q0=10,q=5,tau=2],DSol);
[ t = 2 log( 2 ) ]
```

or, using a floating-point value:

```
float(%);
[ t = 1.386294361119891 ]
```

Example 5.6 – Two-dimensional motion under constant acceleration – Consider the two-dimensional motion of a particle in the *x*-*y* plane with constant acceleration in the *y* direction only as illustrated in Figure 5.1. The equations for the position of the particle in the *x* and *y* directions at any time *t* are given by equations *EqX* and *EqY*:

```
EqX : x=x0+v0*cos(theta[0])*t;
```

$$x = x0 + \cos(\theta_0)\, t\, v0$$

```
EqY : y=y0+v0*sin(theta[0])*t+a*t^2/2;
```

$$y = y0 + \sin(\theta_0)\, t\, v0 + \frac{a\, t^2}{2}$$

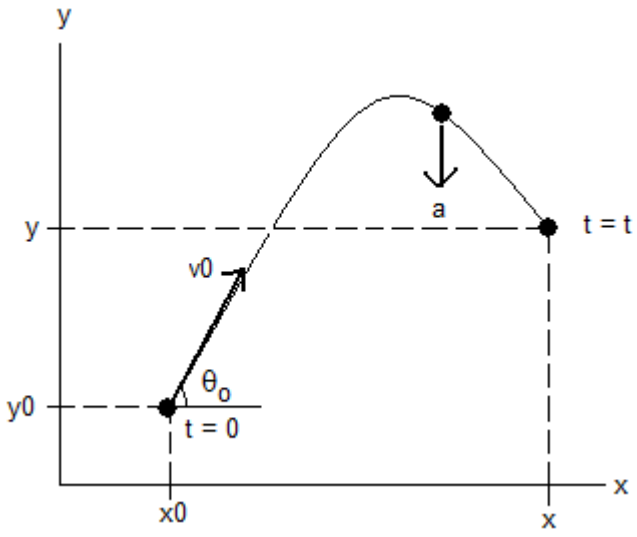


Figure 5.1. Two dimensional particle motion

Let's solve these two equations for the parameters *v0* (the initial velocity) and *a* (the acceleration), given the initial position (*x0*,*y0*), the angle θ_0, and the time *t*:

```
SolXY : solve([EqX,EqY],[v0,a]);
```

$$\left[\left[v0 = -\frac{x0 - x}{\cos(\theta_0)\, t},\ a = \frac{\cos(\theta_0)(2\,y - 2\,y0) + 2\sin(\theta_0)\,x0 - 2\sin(\theta_0)\,x}{\cos(\theta_0)\, t^2} \right]\right]$$

Suppose that we want to substitute the values *x0* = *0*, *x* = *2 m*, *y0* = *2 m*, *y* = *8 m*, $\theta_0 = \pi/6$, and *t* = *2 s*, then we can calculate the values of *v0* (*m*/*s*) and *a* (m/s^2) as follows:

```
SolXYEval : subst([x0=0,x=2,y0=2,y=8,theta[0]=%pi/6,t=2],SolXY);
```

$$\left[\left[v0 = \frac{2}{\sqrt{3}},\ a = \frac{6\sqrt{3} - 2}{2\sqrt{3}} \right]\right]$$

These symbolic results can be converted into floating-point results by using function *float*:

```
float(%);
```

$$[\,[\, v0 = 1.154700538379252\ ,\ a = 2.422649730810375\ ]\,]$$

NOTE 1: Notice that the contents of variables *SolXY* and *SolXYEval* include double brackets [[ ]]. If we were to extract one of the values by using a single sub-index, say, *SolXYEval[1]*, we get:

```
SolXYEval[1];
```

$$[\ v0 = \frac{2}{\sqrt{3}},\ a = \frac{6\sqrt{3} - 2}{2\sqrt{3}}\ ]$$

Basically we get the same result as above, but with only one set of brackets [ ]. The double brackets of variables *SolXY* and *SolXYEval* indicate that these variables are matrices, rather than vectors or lists, but with only one element. Thus, extracting that element by using, as we did above, *SolXYEval[1]*, produces the single element with one set of brackets. To extract the components of this element we need to use an additional subindex, e.g., *SolXYEval[1][1]* or *SolXYEval[1][2]*, as illustrated below:

```
SolXYEval[1][1]; SolXYEval[1][2];
```

$$v0 = \frac{2}{\sqrt{3}}$$

$$a = \frac{6\sqrt{3} - 2}{2\sqrt{3}}$$

NOTE 2: The solution for *v0* and *a* out of equations *EqX* and *EqY* is straightforward because the terms *v0* and *a* are algebraic (and linear) in the equations. Non-algebraic terms, such as θ_0, cannot be isolated using solve as illustrated here:

```
SolXY1 : solve([EqX,EqY],[theta[0],t]);
[ ]
```

In such a case, a numerical solution (using, for example, function *find_root*) is recommended after substituting all the known values in the equations.

NOTE 3: A solution for *x0* and *t* is allowed because both terms are algebraic in the equations, even if *t* is quadratic. The corresponding solution would be:

```
(%i12)  SolXY2 : solve([EqX,EqY],[x0,t]);
```

$$(\%o12)\ [\ [\ x0 = -\frac{\cos(\theta_0)\,v0\sqrt{-2\,a\,y0 + 2\,a\,y + \sin(\theta_0)^2\,v0^2} - a\,x - \cos(\theta_0)\sin(\theta_0)\,v0^2}{a},\ t = \frac{\sqrt{-2\,a\,y0 + 2\,a\,y + \sin(\theta_0)^2\,v0^2} - \sin(\theta_0)\,v0}{a}\ ],\ [\ x0 = \frac{\cos(\theta_0)\,v0\sqrt{-2\,a\,y0 + 2\,a\,y + \sin(\theta_0)^2\,v0^2} + a\,x + \cos(\theta_0)\sin(\theta_0)\,v0^2}{a},\ t = -\frac{\sqrt{-2\,a\,y0 + 2\,a\,y + \sin(\theta_0)^2\,v0^2} + \sin(\theta_0)\,v0}{a}\ ]\ ]$$

The separation of the two possible solutions is shown below:

```
(%i13)  SolXY2[1];
```

$$(\%o13)\quad \left[x0 = -\frac{\cos(\theta_0)\,v0\sqrt{-2\,a\,y0+2\,a\,y+\sin(\theta_0)^2\,v0^2}-a\,x-\cos(\theta_0)\sin(\theta_0)\,v0^2}{a},\ t=\frac{\sqrt{-2\,a\,y0+2\,a\,y+\sin(\theta_0)^2\,v0^2}-\sin(\theta_0)\,v0}{a}\right]$$

```
(%i14)  SolXY2[2];
```

$$(\%o14)\quad \left[x0 = \frac{\cos(\theta_0)\,v0\sqrt{-2\,a\,y0+2\,a\,y+\sin(\theta_0)^2\,v0^2}+a\,x+\cos(\theta_0)\sin(\theta_0)\,v0^2}{a},\ t=-\frac{\sqrt{-2\,a\,y0+2\,a\,y+\sin(\theta_0)^2\,v0^2}+\sin(\theta_0)\,v0}{a}\right]$$

Example 5-7 – Solutions to the specific energy equation in an open channel flow – The specific energy in an open channel flow is the energy per unit weight of the flow measured with respect to the channel bed. Specific *energy, E,* has units of length and is defined as:

```
Eq0 : E = y + V^2/(2*g);
```

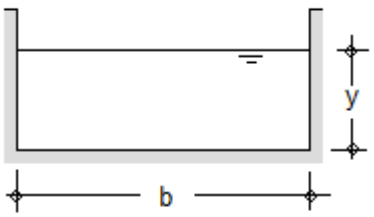
$$E = \frac{V^2}{2g} + y$$

In this equation V is the mean flow velocity, defined as $V = Q/A$, where Q is the flow discharge, and A is the cross-sectional area. With this substitution, the energy equation becomes:

```
Eq0 : subst(V=Q/A,Eq0);
```

$$E = \frac{Q^2}{2gA^2} + y$$

For a rectangular channel, whose cross-section is shown in the figure to the right, the area is $A = by$, where b is the bottom width and y is the flow depth. Substituting the relationship for the area into the specific energy equation results in:




```
Eq0 : subst(A = b*y,Eq0);
```

$$E = \frac{Q^2}{2b^2gy^2} + y$$

Calculating the specific energy – A simple calculation results from replacing all variables in the right-hand side of the equation, for example, $Q = 20\ m^3/s$, $g = 9.81\ m/s^2$, $b = 5\ m$, and $y = 1.2\ m$:

```
Eq1:subst([Q=20,g=9.81,b=5,y=1.2],Eq0);
```

$$E = 1.766315551025031$$

Calculating the bottom width – Replace the values $Q = 20\ m^3/s$, $g = 9.81\ m/s^2$, $y = 1.2\ m$, and $E = 1.77\ m$, into *Eq0*, to produce *Eq2*:

```
Eq2:subst([Q=20,g=9.81,y=1.2,E=1.77],Eq0);
```

$$1.77 = \frac{14.15788877562578}{b^2} + 1.2$$

The solutions for b are stored in variable *bSol*:

```
bSol : solve(Eq2,b);
```

```
`rat' replaced 0.57 by 57/100 = 0.57
`rat' replaced -14.1578887756258 by -124731/8810 = -14.1578887627696
```

$$\left[b = -\frac{\sqrt{415770}}{\sqrt{16739}}, b = \frac{\sqrt{415770}}{\sqrt{16739}} \right]$$

Notice that *Maxima* converted floating-point values to rational values as indicated by the substitutions shown above. To obtain the floating-point values of the solutions for b you may use:

```
float(bSol);
```

$$[b = -4.983813934949041 , b = 4.983813934949041]$$

To check the results thus obtained we can substitute the values of *bSol[1]* and *bSol[2]* into equation *Eq2*:

```
(%i7) subst(bSol[1],Eq2);
(%o7) 1.77 = 1.770000000517594
(%i8) subst(bSol[2],Eq2);
(%o8) 1.77 = 1.770000000517594
```

Calculating the discharge – Substitute the values $g = 9.81\ m/s^2$, $y = 1.2\ m$, $E = 1.77\ m$, and $b = 5\ m$, into *Eq0*, to produce *Eq3*:

```
Eq3:subst([g=9.81,y=1.2,E=1.77,b=5],Eq0);
```

$$1.77 = 0.0014157888775626\ Q^2 + 1.2$$

Solutions to *Eq3*, as symbolic results, are stored into variable *QSol*:

```
QSol : solve(Eq3,Q);
`rat' replaced 0.57 by 57/100 = 0.57
`rat' replaced -0.001415788877563 by -25/17658 = -0.001415788877563
```

$$[\ Q = -\frac{9\sqrt{6213}}{25\sqrt{2}},\ Q = \frac{9\sqrt{6213}}{25\sqrt{2}}\ ]$$

The corresponding floating-point values are:

```
float(QSol);
[ Q = - 20.06495452274936 , Q = 20.06495452274936 ]
```

The solutions check out fine when replaced into equation *Eq3I:*

```
subst(QSol[1],Eq3);
1.77 = 1.77
```

```
subst(QSol[2],Eq3);
1.77 = 1.77
```

Calculating the flow depth - Substitute the values $g = 9.81\ m/s^2$, $Q = 5\ ft^3/s$, $E = 1.77\ m$, and $b = 5\ m$, into *Eq0*, to produce *Eq4*:

```
Eq4:subst([g=9.81,Q=5,E=1.77,b=5],Eq0);
```

$$1.77 = y + \frac{0.050968399592253}{y^2}$$

Symbolic solutions for the flow depth, *y*, are calculated using function *solve*:

```
ySol : solve(Eq4,y);
`rat' replaced 1.77 by 177/100 = 1.77
`rat' replaced -0.05096839959225 by -50/981 = -0.05096839959225
```

< followed by a long symbolic solution >

The three symbolic solutions provided occupy a large amount of output, therefore, they are not shown in this Chapter. To give you an idea, the first solution alone is shown below:

```
ySol[1];
```

$$y = \left(-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right)\left(\frac{2^{-\frac{5}{2}}\sqrt{188976799}\,\%i}{24525} + \frac{176476799}{981000000}\right)^{1/3} + \frac{3481\left(\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right)}{10000\left(\frac{2^{-\frac{5}{2}}\sqrt{188976799}\,\%i}{24525} + \frac{176476799}{981000000}\right)^{1/3}} + \frac{59}{100}$$

This solution includes the unit imaginary number (%i), therefore, the solution is a complex number. To see a floating-point version of this solution we combine functions *float* and *rectform* as follows:

```
y1 : float(rectform(ySol[1]));
y = 7.6327832942979512 10^-17 %i + 0.1789835699857
```

The other two solutions are shown below in their floating-point format:

```
y2 : float(rectform(ySol[2]));
y = 2.0816681711721685 10^-17 %i - 0.16240572720986

y3 : float(rectform(ySol[3]));
y = 1.753422157224155
```

Notice that the imaginary parts of *y* in the solutions *y1* and *y2* are almost zero, thus, the actual solutions in *y1* and *y2* should be:

```
y1 : realpart(y1);
y = 0.1789835699857

y2 : realpart(y2);
y = - 0.16240572720986
```

The following array or list shows the three results for the flow depth:

```
[y1,y2,y3];
[ y = 0.1789835699857 , y = - 0.16240572720986 , y = 1.753422157224155 ]
```

To check if these values satisfy the specific energy solution we substitute the values *y1*, *y2*, and *y3*, into equation *Eq4*:

```
[subst(y1,Eq4),subst(y2,Eq4),subst(y3,Eq4)];
[ 1.77 = 1.770000000000001 , 1.77 = 1.770000000000003 , 1.77 = 1.77 ]
```

NOTE: Equation *Eq4,* created above as:

```
Eq4:subst([g=9.81,Q=5,E=1.77,b=5],Eq0);
```

$$1.77 = y + \frac{0.050968399592253}{y^2}$$

can be converted into a polynomial equation by multiplying both sides by y^2:

```
Eq5:ratsimp(Eq4*y^2);
`rat' replaced 1.77 by 177/100 = 1.77
`rat' replaced 0.050968399592253 by 50/981 = 0.050968399592253
```

$$\frac{177\,y^2}{100} = \frac{981\,y^3 + 50}{981}$$

and solved using function *allroots*:

```
Soly : allroots(Eq5,y);
[ y = 0.1789835699857 , y = - 0.16240572720986 , y = 1.753422157224155 ]
```

Compare with the solutions found above using *solve*:

```
[y1,y2,y3];
[ y = 0.1789835699857 , y = - 0.16240572720986 , y = 1.753422157224155 ]
```

Example 5-8 – Critical depth in a rectangular channel – Critical depth corresponds to the depth of minimum specific energy. This can be calculated using the condition $dE/dy = 0$. Thus, we start with the specific energy equation used in the example above, namely:

```
Eq0;
```

$$E = \frac{Q^2}{2\,b^2\,g\,y^2} + y$$

The equation that results from taking the derivative with respect to *y* for *Eq0* is written as:

```
EqC : dE/dy = diff(rhs(Eq0),y);
```

$$\frac{dE}{dy} = 1 - \frac{Q^2}{b^2\,g\,y^3}$$

Next, we substitute $dE/dy = 0$ into this equation to obtain:

```
EqC : subst(dE/dy=0,EqC);
```

$$0 = 1 - \frac{Q^2}{b^2\,g\,y^3}$$

The solution for this equation (being a cubic equation) produces three values:

```
Solyc : solve(EqC,y);
```

$$\left[y = \frac{(\sqrt{3}\,\%i - 1)\,Q^{2/3}}{2\,b^{2/3}\,g^{1/3}},\ y = -\frac{(\sqrt{3}\,\%i + 1)\,Q^{2/3}}{2\,b^{2/3}\,g^{1/3}},\ y = \frac{Q^{2/3}}{b^{2/3}\,g^{1/3}} \right]$$

The first two solutions shown above are complex values, therefore, only the third one makes sense physically. This is the expression for the critical depth *yc*:

```
EqyC : yc = rhs(Solyc[3]);
```

$$yc = \frac{Q^{2/3}}{b^{2/3}\,g^{1/3}}$$

In rectangular channels the unit discharge (or discharge per unit width), $q = Q/b$, is typically used. Thus, we could write:

```
EqyC : subst(Q = q*b,EqyC);
```

$$yc = \frac{q^{2/3}}{g^{1/3}}$$

This result is typically written as: $y_c = \sqrt[3]{\frac{q^2}{g}}$.

allroots (Equations > Roots of polynomial)
The function *allroots* is used to calculate all roots, both real and complex, in a polynomial equation. Some examples using function *allroots* are presented below.

Example 5-9 – Repeating Examples 5.3 and 5.4 - To illustrate the use of function *allroots* we repeat the solutions to Examples 5.3 and 5.4 using such function:

```
allroots(3*x^3+5*x^2+6*x+2=0);
[ x = - 0.46181419095609 , x = 1.039550218436705 %i - 0.60242623785529 , x = - 1.039550218436705 %i - 0.60242623785529 ]
```

```
allroots(6*x^4-3*x^3+x^2-x+125=0);
[ x = 1.508650793156902 %i + 1.629306910028965 , x = 1.629306910028965 - 1.508650793156902 %i , x = 1.524062994136233 %i -
1.379306910028965 , x = - 1.524062994136233 %i - 1.379306910028965 ]
```

realroots (Equations > Roots of polynomial (real))
Function *realroots* produces the real roots of a polynomial equation. Some examples of the use of *realroots* are presented below.

Example 5-10 – Repeating Examples 5.3 and 5.4 - To illustrate the use of function *realroots* we repeat the solutions to Examples 5.3 and 5.4 using such function:

```
realroots(3*x^3+5*x^2+6*x+2=0);
```

$$[\, x = -\frac{15495913}{33554432} \,]$$

```
realroots(6*x^4-3*x^3+x^2-x+125=0);
```

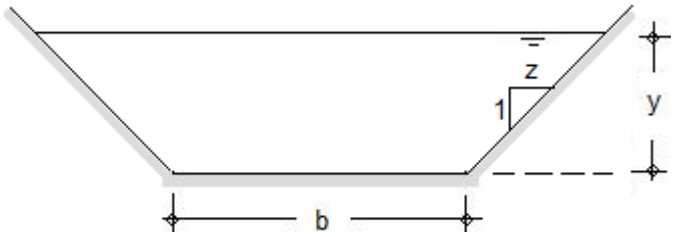
```
[ ]
```

NOTE: Functions *allroots* and *realroots* require a specific polynomial to produce a solution. Thus, attempting a general solution (e.g., as in the quadratic equation in Example 5.1) will produce an error:

```
(%i55)  allroots(a*x^2+b*x+c=0,x);
** error while printing error message **
`allroots': polynomial not univariate: ~M
 -- an error.  To debug this try debugmode(true);
```

The following engineering examples refer to function *allroots* and *realroots*.

Example 5-11 - Critical depth for a trapezoidal channel – The cross-section of a trapezoidal channel, as shown in the figure to the right, is characterized by its bottom width *b*, its depth of flow *y*, and its side slope *z* (i.e., zH:1V).



You can show that the specific energy equation for this cross-sectional shape is calculated as:

```
Eq0 : E = y + Q^2/(2*g*((b+z*y)*y)^2);
```

$$E = \frac{Q^2}{2\,g\,y^2\,(y\,z+b)^2} + y$$

The critical depth can be obtained by taking the derivative of the specific energy with respect to *y*, i.e.,

```
EqC: dE/dy = diff(rhs(Eq0),y);
```

$$\frac{dE}{dy} = -\frac{Q^2}{g\,y^3\,(y\,z+b)^2} - \frac{z\,Q^2}{g\,y^2\,(y\,z+b)^3} + 1$$

and, then, making *dE/dy = 0*:

```
EqC : subst(dE/dy = 0, EqC);
```

$$0 = -\frac{Q^2}{g\,y^3\,(y\,z+b)^2} - \frac{z\,Q^2}{g\,y^2\,(y\,z+b)^3} + 1$$

An attempt to use function *solve* to solve this equation produces a re-arrangement of the previous result into a polynomial, although no solution is given:

```
EqC1 : solve(EqC,y);
```

$$[\ 0 = -2\,y\,z\,Q^2 - b\,Q^2 + g\,y^6\,z^3 + 3\,b\,g\,y^5\,z^2 + 3\,b^2\,g\,y^4\,z + b^3\,g\,y^3\ ]$$

Let's substitute the following parameters into the equation $b = 5\ ft$, $z = 1$, $Q = 200\ ft^3/s$, $g = 32.2\ ft/s^2$:

```
EqC1 : subst([b=5,z=1,Q=200,g=32.2],EqC1);
```

$$[\ 0 = 32.2\,y^6 + 483.0000000000001\,y^5 + 2415.0\,y^4 + 4025.000000000001\,y^3 - 80000\,y - 200000\ ]$$

Trying function *solve* after replacing the parameters still does not produce a solution:

```
SolC1 : solve(EqC1,y);
`rat' replaced -4025.0 by -4025/1 = -4025.0
`rat' replaced -2415.0 by -2415/1 = -2415.0
`rat' replaced -483.0 by -483/1 = -483.0
`rat' replaced -32.2 by -161/5 = -32.2
```

$$[\ 0 = 161\,y^6 + 2415\,y^5 + 12075\,y^4 + 20125\,y^3 - 400000\,y - 1000000\ ]$$

Notice, however, that function *solve* recast the polynomial into one with all integer coefficients. Since the equation is indeed polynomial, we'll use function *allroots* to find the solution:

```
allroots(SolC1);
`allroots': not a polynomial:
```

$$[\ 0 = 161\,y^6 + 2415\,y^5 + 12075\,y^4 + 20125\,y^3 - 400000\,y - 1000000\ ]$$

```
 -- an error.  To debug this try debugmode(true);
```

This attempt to solve the equation using function *allroots* still reports an error. The reason for the error is the fact that the content of variable *SolC1* is a list (as indicated by the brackets [ ] enclosing the polynomial. Thus, to extract the actual polynomial it is necessary to use a subindex. Thus, try:

```
allroots(SolC1[1]);
```

$$[\ y = 2.990293557760706\ ,\ y = -2.597815990480922\ ,\ y = 3.750522392769366\ \%i - 0.6876094381974\ ,\ y = -3.750522392769366\ \%i - 0.6876094381974\ ,\ y = 2.423329437699932\ \%i - 7.00862934544249\ ,\ y = -2.423329437699932\ \%i - 7.00862934544249\ ]$$

Alternatively, one can use *realroots(SolC1[1]);* to find only those real solutions, namely, *y = 2.99 ft ≈ 3 ft*, and *y = -2.59... ft ≈ -2.6 ft*. Only the first result, *y = 3 ft*, makes physical sense.

Example 5-12 - Manning's equation for a trapezoidal channel – Consider an open channel of constant cross-section laid on a bed slope *S0*. The bed slope represents the drop in *ft* per *ft* of channel length. If uniform flow, or flow of constant depth, occurs in this channel, the mean flow velocity is calculated using the Manning's equation, *EqMV*:

```
EqMV : V = Cu/n*R^(2/3)*sqrt(S0);
```

$$V = \frac{Cu\, R^{2/3} \sqrt{S0}}{n}$$

This is an empirical equation developed by an Irish engineer by the name of Manning in the late 1800's. In spite of being totally empirical, it is the most popular equation to calculate uniform flow in open channels. It is widely used around the world, and is known in Europe as the Manning-Stickler equation. In the Manning's equation the parameter *Cu* is a constant that depends on the system of units used. If using units of the International System (S.I.), *Cu = 1*. If using units of the English System (E.S.), *Cu = 1.486*. The parameter *n* is known as the Manning's roughness coefficient and it depends on the type of lining for the channel (e.g., for concrete, *n = 0.012*). Finally, *R* is known as the hydraulic radius, and it's defined as the ratio of the cross-sectional area *A* to the wetted-perimeter *P* of the cross-section (i.e., the length of the cross-section perimeter 'wetted' by the water), thus, *R = A/P*.

The Manning's equation can be combined with the continuity equation (conservation of mass) for a liquid (constant density), *EqQ*, where *Q* is the volumetric discharge through the cross-section:

```
EqQ : Q = V*A;
```

$$Q = A\, V$$

When substituting the Manning's equation (*EqMV*) into the continuity equation (*EqMQ*) results in a Manning's equation based on the discharge, *EqMQ*:

```
EqMQ : subst(EqMV,EqQ);
```

$$Q = \frac{Cu\, A\, R^{2/3} \sqrt{S0}}{n}$$

Replacing the hydraulic radius, *R*, in terms of area *A* and perimeter *P*, *EqMQ* becomes:

```
EqMQ : subst(R=A/P,EqMQ);
```

$$Q = \frac{Cu\, A^{5/3} \sqrt{S0}}{n\, P^{2/3}}$$

For a trapezoidal channel, as shown in Example 5-11, the area and wetted perimeter are define below and replaced into equation *EqMQ:*

```
EqMQ : subst([A=(b+z*y)*y,P=b+2*y*sqrt(1+z^2)],EqMQ);
```

$$Q = \frac{Cu\, y^{5/3} (y\, z + b)^{5/3} \sqrt{S0}}{n \left(2\, y \sqrt{z^2 + 1} + b\right)^{2/3}}$$

We will be solving this equation for the depth of uniform flow, referred to as the *normal depth*. We notice that the equation *EqMQ* has exponents *5/3* and *2/3* for the terms containing *y*. However, since the exponents are fractions with denominator *3*, it is possible to convert the equation into a polynomial equation by first multiplying it out by the denominator of the right-hand side of *EqMQ*:

```
EqMQM : denom(rhs(EqMQ))*EqMQ;
```

$$n \left(2\, y \sqrt{z^2 + 1} + b\right)^{2/3} Q = Cu\, y^{5/3} (y\, z + b)^{5/3} \sqrt{S0}$$

Then, we raise both sides of the equation to the third power:

```
EqMQM : EqMQM^3;
```

$$n^3 \left(2\, y \sqrt{z^2 + 1} + b\right)^2 Q^3 = Cu^3\, y^5 (y\, z + b)^5\, S0^{3/2}$$

Now all the terms involving *y* have integer powers (2 and 5) and could be expanded into a polynomial (although we don't actually expand the polynomial out in this solution)

```
EqMQM1 : subst([n=0.012,z=1.5,b=5,Q=200,Cu=1.486,S0=0.00001],EqMQM);
```

$$13.824 (3.605551275463989\, y + 5)^2 = 1.0376632315788739\, 10^{-7}\, y^5 (1.5\, y + 5)^5$$

Even without expanding the polynomial, we can use function *allroots* to get all *10* roots of the resulting polynomial (the order *10* comes from multiplying out the *y5* term with the (1.5y+5)5 term in the right-hand side of equation *EqMQM1* above):

```
allroots(EqMQM1);
```

```
[ y = 7.9276784870052392 10^-4 %i - 1.386750164711896 , y = - 7.9276784870052392 10^-4 %i - 1.386750164711896 , y =
7.731739150971134 %i - 9.686251014607667 , y = - 7.731739150971134 %i - 9.686251014607667 , y = 10.92855141765184 %i -
1.738661959909446 , y = - 10.92855141765184 %i - 1.738661959909446 , y = 7.727844964430052 %i + 6.213088524164945 , y =
6.213088524164945 - 7.727844964430052 %i , y = 9.504861187206684 , y = - 12.97437862374523 ]
```

If we had used function *realroots* we would find only the values *y = 9.50 ft* and *y = -12.97 ft*. Of these two results, the only one that makes sense physically is *y = 9.50 ft*.

find_root (Equations > Solve numerically ...)
Function *find_root* is used for the numerical solution of an equation *f(x) = 0* in the interval [*a*,*b*]. The general call to the function is `find_root(f(x)=0,x,a,b).`

Example 5-13 – Trajectory of a projectile – A projectile launched in the gravitational field of Earth, with the *x* position along the horizontal direction and the *y* position along the vertical direction, describes a trajectory given by the equation (See Figure 5.1, replace *a* = *g*, and $\theta_0 = \theta$):

```
EqT : y = y0 + (x-x0)*tan(theta) - g/(2*v0^2*cos(theta)^2)*(x-x0)^2;
```

$$y = y0 - \frac{g\,(x - x0)^2}{2\cos(\theta)^2\, v0^2} + \tan(\theta)(x - x0)$$

Here we use θ instead of θ_0 because *Maxima* does not recognized a sub-indexed variable as a variable to solve for.

Now, suppose that you are given the following data values: *x0 = 5 m, y0 = 20 m, x = 20 m, y = 12 m, v0 = 10 m/s,* and *g = 9.81 m/s2*. Replacing those data values into *EqT* produces the following equation *EqT1*:

```
EqT1 : subst([x0=5,y0=20,x=20,y=12,v0=10,g=9.81],EqT);
```

$$12 = 15\tan(\theta) - \frac{11.03625}{\cos(\theta)^2} + 20$$

You can try to solve this equation using *solve*, but you will get an error. Instead, we will attempt a numerical solution using function *find_root*. To get an idea of the interval where the solution is located we will define the following function *f(θ)* using the right-hand side (rhs) and left-hand side (lhs) of equation *EqT1*:

```
f(theta) := rhs(EqT1)-lhs(EqT1);
```

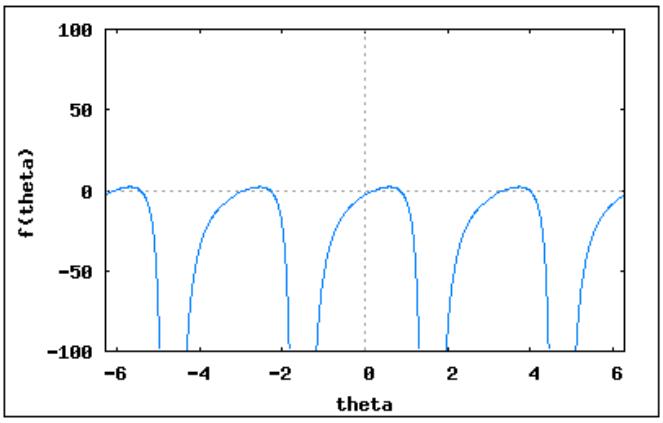
$$f(\theta) := \mathrm{rhs}(EqT1) - \mathrm{lhs}(EqT1)$$

```
f(theta);
```

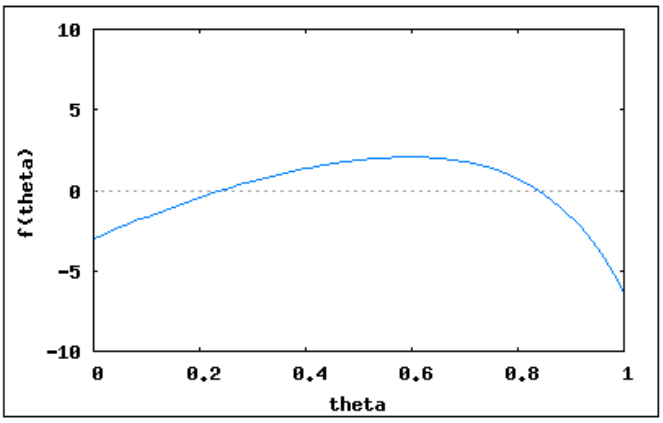
$$15\tan(\theta) - \frac{11.03625}{\cos(\theta)^2} + 8$$

To check the behavior of the function, and the ranges where it becomes zero (roots of the equation), we plot the function *f(θ)* first in the interval $-2\pi < \theta < 2\pi$. This plot corresponds to the left-hand side figure below. We notice that there are two roots for this equation in the range $0 < \theta < \pi$. The figure to the right-hand side, below, shows the behavior of the function in the range $0 < \theta < 1$, showing one root in the interval $0 < \theta < 0.5$ and a second one in the interval $0.5 < \theta < 1$.

```
wxplot2d(f(theta),[theta,-2*%pi,2*%pi],[y,-100,100],
[xlabel,"theta"],[ylabel,"f(theta)"]);
```




```
wxplot2d(f(theta),[theta,0,1],[y,-10,10],
[xlabel,"theta"],[ylabel,"f(theta)"]);
```



Thus, we invoke function *find_root* twice, one for each of the intervals defined above, in order to determine the two smallest solutions indicated by the figures above:

```
theta_1 : find_root(EqT1,theta,0,0.5);
0.24260352318186

theta_2 : find_root(EqT1,theta,0.5,1.0);
0.83823547735931
```

These solutions are given, by default, in the natural units of angular measurement, namely, radians. To convert to degrees, use the conversion factor $\theta^\circ = (180/\pi)\,\theta^r$, thus:

```
theta_1 : float(180/%pi*theta_1);
13.90015797332477

theta_2 : float(180/%pi*theta_2);
48.0273550908223
```

newton

Function *newton*, obtained through the command *load(newton1)*, is used also for the numerical solution of an equation of the form *f(x) = 0*. The solution starts with an initial guess *x0* for the solution, with convergence criteria ε. The general call to the function is `newton(f(x),x,x0,ε)`.

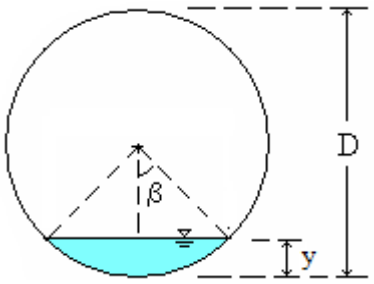
Example 5-14 – Solve Exercise 5-13 using function *newton* - Try the following commands:

```
load(newton1);
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/numeric/newton1.mac
```

```
newton(f(theta),theta,0.1,1/10000);
0.2426034734659
```

```
newton(f(theta),theta,0.9,1/10000);
0.83823636127712
```

Example 5-15 – Manning's equation for a circular open channel – The cross-section of a circular channel is characterized by its diameter *D*, and its depth *y*. These two variables are related by the half-angle β, such that $cos(\beta) = 1-2(y/D)$.



In this example we will use both functions *find_root* and *newton* to determine the depth of flow (normal depth) for a circular open-channel flow. First, we define the Manning's equation as we did in Example 5-12 (*EqM*):

```
EqM: V = Cu/n*R^(2/3)*sqrt(S0);
```

$$V = \frac{Cu\, R^{2/3} \sqrt{S0}}{n}$$

Next, we define the continuity equation, *EqQ*:

```
EqQ : Q = V*A;
```

$$Q = A\, V$$

Next, we combine them into equation *EqMQ*:

```
EqMQ : subst(EqM,EqQ);
```

$$Q = \frac{Cu\, A\, R^{2/3} \sqrt{S0}}{n}$$

Next, we substitute the definition of the hydraulic radius:

```
EqMQ : subst(R=A/P,EqMQ);
```

$$Q = \frac{Cu\, A^{5/3} \sqrt{S0}}{n\, P^{2/3}}$$

Next, we substitute the definitions of the area, *A*, and wetted perimeter, *P*, for a circular cross-section in terms of the half-angle *b* to produce equation *EqMQC*:

```
EqMQC : subst([A=D^2/4*(beta-sin(beta)*cos(beta)),P=beta*D],EqMQ);
```

$$Q = \frac{(\beta - \cos(\beta)\sin(\beta))^{5/3}\, Cu\, D^{8/3} \sqrt{S0}}{4\, 4^{2/3}\, \beta^{2/3}\, n}$$

Next, we replace the half-angle *b* in terms of the depth *y* and diameter *D*, to produce equation *EqMQCy*:

```
EqMQCy : subst(beta=acos(1-2*y/D),EqMQC);
```

$$Q = \frac{Cu\left(-\sqrt{1-\left(\frac{2y}{D}-1\right)^2}\left(1-\frac{2y}{D}\right)-\operatorname{acos}\left(\frac{2y}{D}-1\right)+\pi\right)^{5/3} D^{8/3}\sqrt{S0}}{4\,4^{2/3} n\left(\pi-\operatorname{acos}\left(\frac{2y}{D}-1\right)\right)^{2/3}}$$

Finally, we substitute the parameters of the problem as follows, *Cu = 1.486*, *D = 5 ft*, *Q = 2.5 ft3/s*, *S0 = 0.000023*, and *n = 0.012*, to create equation *EqMQCy1*:

```
EqMQCy1 : subst([Cu=1.486,D=5,Q=2.5,S0=0.000023,n=0.012],EqMQCy);
```

$$2.5 = \frac{3.711773772730574\,5^{2/3}\left(-\sqrt{1-\left(\frac{2y}{5}-1\right)^2}\left(1-\frac{2y}{5}\right)-\operatorname{acos}\left(\frac{2y}{5}-1\right)+\pi\right)^{5/3}}{4^{2/3}\left(\pi-\operatorname{acos}\left(\frac{2y}{5}-1\right)\right)^{2/3}}$$

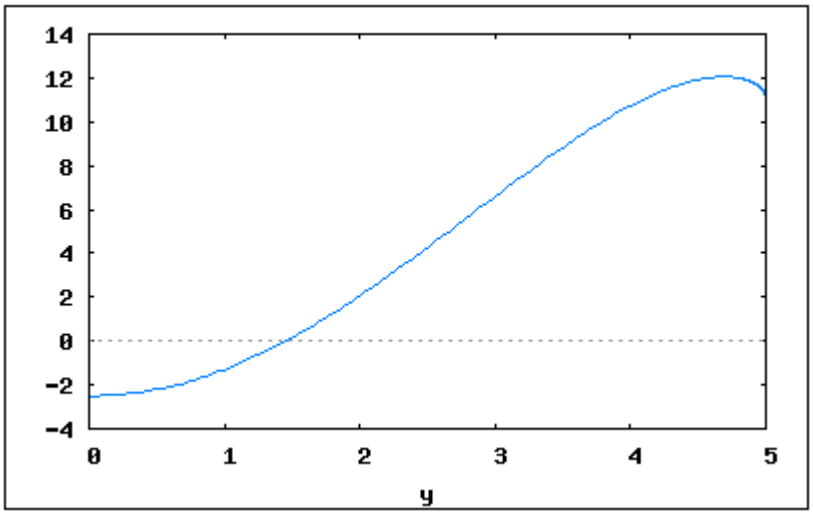
This is the equation we need to solve for *y*. In order to understand the behavior of the equation, we create the variable *fn* representing the difference between the right-hand side (rhs) and the left-hand side (lhs) of equation *EqMQCy1*:

```
fn : rhs(EqMQCy1)-lhs(EqMQCy1);
```

$$\frac{3.711773772730574\,5^{2/3}\left(-\sqrt{1-\left(\frac{2y}{5}-1\right)^2}\left(1-\frac{2y}{5}\right)-\operatorname{acos}\left(\frac{2y}{5}-1\right)+\pi\right)^{5/3}}{4^{2/3}\left(\pi-\operatorname{acos}\left(\frac{2y}{5}-1\right)\right)^{2/3}} - 2.5$$

A plot of the variable *fn* is shown in the figure below. This plot indicates that a solution exists in the interval *1 < y < 2*.

```
wxplot2d(fn,[y,0,5]);
```



The solution can be obtained using, for example, function *find_root*:

```
find_root(fn=0,y,1,2);
1.455668272515256
```

Alternatively, we can use function *newton* (use *load(newton1)* if no yet loaded):

```
newton(fn,y,1.5,1/100);
```

$$1.5 - 0.3392922556908\ 0.34199518933534\ 2.519842099789746\left(\frac{2.283564222881931\ 5^{2/3}}{4^{2/3}} - 2.5\right)$$

To find the value in floating-point format use:

```
float(%);
1.456187349523008
```

NOTE: Function *newton* could be very sensitive to the initial guess used. For example, using an initial value of zero produces an error (most likely due to the solution diverging):

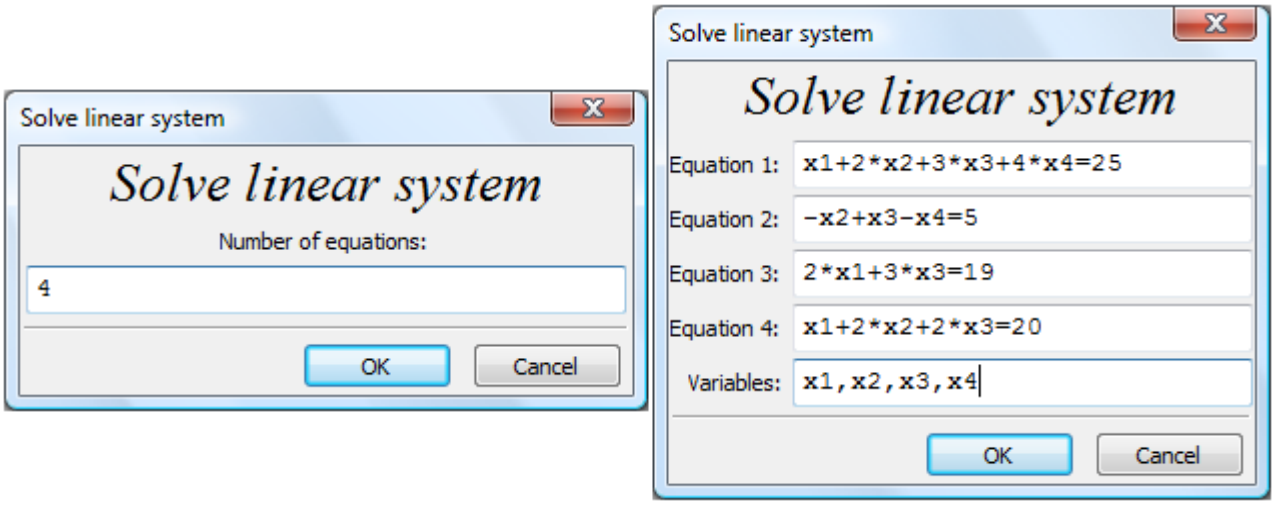
`newton(fn,y,0,1/100);`	`newton(fn,y,1,1/100);`
`Division by 0`	`- ( ( 3.711773772730574 5^(2/3) ... etc`

The second case produces an extremely large expression. To determine the corresponding floating-point value, use *float*:

```
float(%);
1.457822335956577
```

linsolve (Equations > Solve linear system ...) and matrix solutions
Function *linsolve* is used to solve a system of linear equations. The easiest way to set up the solution is using the *Equations* > *Solve linear system*... menu option in the *wxMaxima* interface. Example 5-16 illustrates the use of *linsolve* as well as matrix solutions.

Example 5-16 - System of 4 linear equations - Try the following example using *Equations* > *Solve linear system*...:



The corresponding *wxMaxima* entry line, and solution, is:

```
linsolve([x1+2*x2+3*x3+4*x4=25, -x2+x3-x4=5, 2*x1+3*x3=19, x1+2*x2+2*x3=20], [x1,x2,x3,x4]);
```

$$[\, x1 = -2 \,,\, x2 = \frac{10}{3} \,,\, x3 = \frac{23}{3} \,,\, x4 = -\frac{2}{3} \,]$$

The system of equations can be written as:

```
x1+2*x2+3*x3+4*x4=25
    -x2+  x3-  x4=5
2*x1     +3*x3    =19
 x1+2*x2+2*x3     =20
```

which is equivalent to the matrix equation:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 1 & -1 \\ 2 & 0 & 3 & 0 \\ 1 & 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} \quad \begin{bmatrix} 25 \\ 5 \\ 19 \\ 20 \end{bmatrix}$$

$$\mathbf{A} \;.\; \mathbf{x} = \mathbf{b}$$

Using matrices, the solution is calculated as:

$$\mathbf{x} = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \mathbf{A}^{-1} \cdot \mathbf{b}$$

The coefficients of matrix **A** are the coefficients of the different variables *x1*, *x2*, *x3*, and *x4*, in each of the equations. The right-hand vector **b** is composed of the right-hand side values of the equations. Using *Maxima* matrix **A** can be entered as:

```
A : matrix([1,2,3,4],[0,-1,1,-1],[2,0,3,0],[1,2,2,0]);
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 1 & -1 \\ 2 & 0 & 3 & 0 \\ 1 & 2 & 2 & 0 \end{bmatrix}$$

while vector **b** is entered as:

```
b : matrix([25],[5],[19],[20]);
```

$$\begin{bmatrix} 25 \\ 5 \\ 19 \\ 20 \end{bmatrix}$$

The inverse of matrix **A**, namely $\mathbf{A}^{-1}$, can be calculated in *Maxima* using:

```
A^^-1;
```

$$\begin{bmatrix} -\frac{1}{4} & -1 & \frac{3}{4} & -\frac{1}{4} \\ -\frac{1}{24} & -\frac{1}{6} & -\frac{5}{24} & \frac{11}{24} \\ \frac{1}{6} & \frac{2}{3} & -\frac{1}{6} & \frac{1}{6} \\ \frac{5}{24} & -\frac{1}{6} & \frac{1}{24} & -\frac{7}{24} \end{bmatrix}$$

Matrix multiplication in *Maxima* is indicated by using a dot (.) instead of the scalar multiplication (*). Thus, the solution to the matrix equation is:

```
A^^-1 . b;
```

$$\begin{bmatrix} -2 \\ \frac{10}{3} \\ \frac{23}{3} \\ -\frac{2}{3} \end{bmatrix}$$

This is exactly the same as the solution found earlier with function *linsolve*, i.e., *x1 = -2*, *x2 = 10/3*, *x3 = 23/3*, and *x4 = -2/3*. The corresponding floating-point solution can be obtained by using function *float*, i.e.,

```
float(%);
```

$$\begin{bmatrix} -2.0 \\ 3.333333333333334 \\ 7.666666666666667 \\ -0.66666666666667 \end{bmatrix}$$

This is to say, *x1 = -2.0, x2 = 3.33..., x3 = 7.66...,* and *x4 = -0.66...*

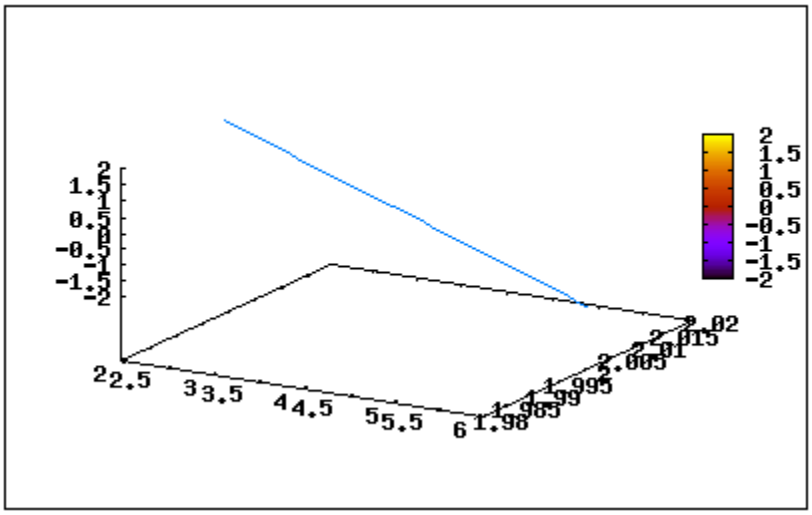
This example represents a system of 4 equations in 4 unknowns. In the following example we consider the case of a system in which there are more unknowns than equations.

Example 5-17 - Underdetermined system - In this case we have a system of two equations and three unknowns:

```
linsolve([x+y+z=6, x+2*y+z=8], [x,y,z]);
[ x = 4 - %r1 , y = 2 , z = %r1 ]
```

The results indicates a variety of solutions since the values of *x* and *z* depend on a parameter, arbitrarily referred to by *Maxima* as *%r1*. This can be interpreted as any variable, say, *r*, thus, you would write for this solution: *x = 4-r, y = 2, z = r*. This solution represent the parametric equations of a straight line in space as illustrated in the following figure. The line is contained in the plane *y = 2*, a plane parallel to the *x-z* axis.

```
wxplot3d([4-r,2,r],[r,-2,2],[s,-2,2],[gnuplot_preamble,"unset key"]);
```



Example 5-18 – Overdetermined system – An overdetermined system has more equations than unknowns, e.g.,

```
linsolve([x+y=5,x+2*y=8,3*x+y=9],[x,y]);
Dependent equations eliminated:  (3)
[ x = 2 , y = 3 ]
```

In this case the solution $x = 2$, $y = 3$ applies to all equations, *Maxima* was able to find the solution and eliminate 1 dependent equation.

If we change the right-hand side value in the third equation, the system becomes inconsistent and *Maxima* is not able to find a solution:

```
linsolve([x+y=5,x+2*y=8,3*x+y=10],[x,y]);
Inconsistent equations:  [ 3 ]
 -- an error.  To debug this try debugmode(true);
```

Example 5-19 – Symbolic system of three linear equations – The following system of three linear equations results from the analysis of a system of two blocks A and B, of masses m_A and m_B, respectively, connected via a pulley, so that block B slides on a inclined plane. The kinetic friction factor between block B and the inclined plane is μ_k. T is the tension in the cord connecting the two blocks, and a_A and a_B are the accelerations of blocks A and B, respectively.

$$(1)\quad T - m_B \cdot a_B = \mu_k \cdot m_B \cdot g$$
$$(2)\quad 2 \cdot T + m_A \cdot a_A = m_A \cdot g$$
$$(3)\quad 2 \cdot a_A - a_B = 0$$

Using function *linsolve* in *Maxima* we obtain the following result:

```
linsolve([T-mB*aB=mu[k]*mB*g,2*T+mA*aA=mA*g,2*aA-aB=0],[T,aA,aB]);
```

$$\left[T = \frac{(g\,\mu_k\,mA + 2\,g\,mA)\,mB}{4\,mB + mA},\ aA = -\frac{2\,g\,\mu_k\,mB - g\,mA}{4\,mB + mA},\ aB = -\frac{4\,g\,\mu_k\,mB - 2\,g\,mA}{4\,mB + mA} \right]$$

The solution can be achieved using matrices by re-writing the equations as follows:

$$\begin{aligned} &(1)\quad T + 0\cdot a_A - m_B\cdot a_B = \mu_k\cdot m_B\cdot g \\ &(2)\quad 2\cdot T + m_A\cdot a_A + 0\cdot a_B = m_A\cdot g \\ &(3)\quad 0\cdot T + 2\cdot a_A - a_B = 0 \end{aligned}$$

The matrices **A**, **x**, and **b** corresponding to the matrix equation **A.x**=**b** are the following:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & -m_B \\ 2 & m_A & 0 \\ 0 & 2 & -1 \end{bmatrix},\ \mathbf{x} = \begin{bmatrix} T \\ a_A \\ a_B \end{bmatrix},\ \text{and}\ \mathbf{b} = \begin{bmatrix} \mu_k\cdot m_B\cdot g \\ m_A\cdot g \\ 0 \end{bmatrix},$$

Using *Maxima*, we define matrix **A** and vector **b** as follows:

```
A : matrix([1,0,-mB],[2,mA,0],[0,2,-1]); b : matrix([mu[k]*mB*g],[mA*g],[0]);
```

$$\begin{bmatrix} 1 & 0 & -mB \\ 2 & mA & 0 \\ 0 & 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} g\,\mu_k\,mB \\ g\,mA \\ 0 \end{bmatrix}$$

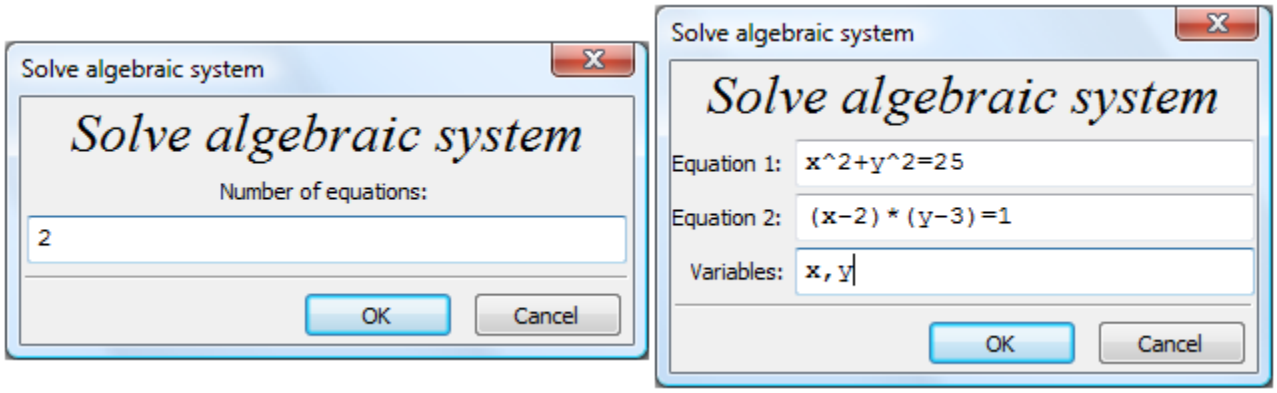
Finally, we solve for the unknown **x** using **x** = $\mathbf{A}^{-1}$ **.** **b** :

```
A^^-1 . b;
```

$$\begin{bmatrix} \frac{g\,\mu_k\,mA\,mB}{4\,mB + mA} + \frac{2\,g\,mA\,mB}{4\,mB + mA} \\ \frac{g\,mA}{4\,mB + mA} - \frac{2\,g\,\mu_k\,mB}{4\,mB + mA} \\ \frac{2\,g\,mA}{4\,mB + mA} - \frac{4\,g\,\mu_k\,mB}{4\,mB + mA} \end{bmatrix}$$

algsys (Equations > Solve algebraic system ...)
Function *algsys* allows the solution of a system of algebraic equations, linear or non-linear.

Example 5-20 – System of two non-linear, algebraic equations – To demonstrate the use of function *algsys*, we use the menu item *Equations* > *Solve algebraic system...* to load the following system of two non-linear, algebraic equations:



The corresponding input line in the *wxMaxima* interface, and the corresponding solutions, are shown below:

```
algsys([x^2+y^2=25, (x-2)*(y-3)=1], [x,y]);
[ [ x = 3 , y = 4 ] , [ x = 1.869067443796836 , y = - 4.63751987281399 ] ,
[ x = 3.248464163822526 , y = 3.80098414434117 ] , [ x = - 4.117531831537709 , y = 2.836535330511234 ] ]
```

Example 5-21 – Pump-pipeline system solution – Consider a pipeline of length *L*, and diameter *D*, connecting two reservoirs such that the free surface in the reservoir downstream is located at an elevation *H* above the free surface in the reservoir upstream. In order to deliver a discharge *Q* of water it is necessary to have a pump providing a hydraulic head (energy per unit weight) h_P. The energy equation written between the free-surfaces of the two reservoirs, after simplification, results in the so-called *system equation*:

$$h_P = H + \frac{8Q^2}{\pi^2 g D^4}\left(f\frac{L}{D} + \Sigma K\right) \quad ,$$

where *f* is a friction factor and ΣK is the sum of coefficients due to local losses in the pipeline (e.g., valves, elbows, entrance from reservoir, discharge into reservoir, etc.).

Centrifugal pumps are characterized by a quadratic pump equation of the form:

$$h_P = a + bQ + cQ^2$$

where the coefficients *a*, *b*, *c* are obtained by testing the pump in the laboratory.

Typically, the values of *H, g, D, f, L, D, ΣK, a, b,* and *c* are given, and the values of *Q* and h_P calculated from the simultaneous solution of the system and pump equations. An example solved using *Maxima*'s function *algsys* is presented next.

First, we define the system equation, *EqS*:

```
EqS : hP = H + 8*Q^2/(%pi^2*g*D^4)*(f*L/D+SK);
```

$$hP = \frac{8\,Q^2\left(SK + \frac{f\,L}{D}\right)}{\pi^2\,g\,D^4} + H$$

and the pump equation, *EqP*:

```
EqP : hP = a+b*Q+c*Q^2;
```

$$hP = c\,Q^2 + b\,Q + a$$

Next, we substitute the values $H = 20\ m$, $g = 9.81\ m/s^2$, $D = 2\ m$, $f = 0.0116$, $L = 100\ m$, $SK = 2.5$, into the system equation, producing equation *EqS1*:

```
EqS1 : subst([H=20,g=9.81,D=2,f=0.0116,L=100,SK=2.5],EqS);
```

$$hP = \frac{0.15698267074414\,Q^2}{\pi^2} + 20$$

Also, we substitute the values $a = 60$, $b = 0$, $c = -0.012$, in the pump equation, producing equation *EqP1*:

```
EqP1 : subst([a=60,b=0,c=-0.012],EqP);
```

$$hP = 60 - 0.012\,Q^2$$

Function *algsys* provides the following solution:

```
algsys([EqS1,EqP1],[hP,Q]);
```

$$\left[\left[hP = \frac{58860\,\pi^2 + 2310000}{2943\,\pi^2 + 38500},\ Q = -\frac{300\sqrt{109}\,\pi}{\sqrt{2943\,\pi^2 + 38500}}\right],\left[hP = \frac{58860\,\pi^2 + 2310000}{2943\,\pi^2 + 38500},\ Q = \frac{300\sqrt{109}\,\pi}{\sqrt{2943\,\pi^2 + 38500}}\right]\right]$$

To see the floating-point value of the solution use function *float*:

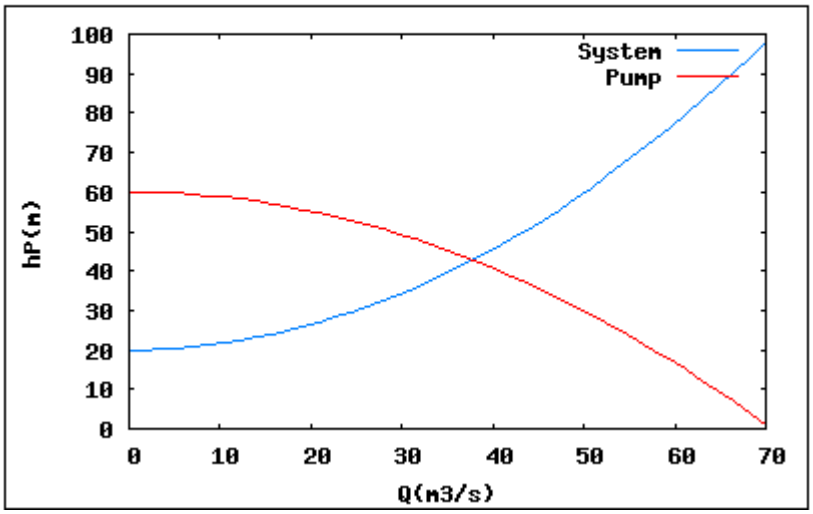
```
float(%);
```

$$[[hP = 42.79919457914721,\ Q = -37.8602753785248],[hP = 42.79919457914721,\ Q = 37.8602753785248]]$$

Only positive values of *Q* and h_P make sense, therefore, the solution is $h_P = 42.80\ m$, $Q = 37.86\ m^3/s$.

Graphical solution – A plot showing both the system and the pump equations can be used to obtain the solution of the pump-pipeline project, as illustrated in the figure below. The point of intersection, known as the *operating point*, is the solution to the problem.

```
wxplot2d([rhs(EqS1),rhs(EqP1)],[Q,0,70],[xlabel,"Q(m3/s)"],
[ylabel,"hP(m)"],[legend,"System","Pump"]);
```



An alternative solution – Since both the system and pump equations are expressed in terms of the pump head, h_P, they can be combined to produce a new equation that can be solved for Q. This can be accomplished in *Maxima* by using:

```
(%i12)  EqNEW : rhs(EqS1)=rhs(EqP1);
```

$$(\%o12)\quad \frac{0.15698267074414\,Q^2}{\pi^2} + 20 = 60 - 0.012\,Q^2$$

This equation can be solved using function *solve* (or *allroots* since it is a polynomial equation):

```
SolQ : solve(EqNEW,Q);
`rat' replaced 0.012 by 3/250 = 0.012
`rat' replaced 0.15698267074414 by 154/981 = 0.15698267074414
```

$$\left[\, Q = -\frac{300\sqrt{109}\,\pi}{\sqrt{2943\,\pi^2 + 38500}},\ Q = \frac{300\sqrt{109}\,\pi}{\sqrt{2943\,\pi^2 + 38500}} \,\right]$$

To see the floating-point values of the discharge solutions use:

```
SolQ : float(SolQ);
[ Q = - 37.8602753785248 , Q = 37.8602753785248 ]
```

Substituting the only positive solution for *Q* into the system equation (*EqS1)* produces the following value for the pump head, h_P:

```
SolhP : subst(SolQ[2],EqS1);
```

$$hP = \frac{225.0190311596439}{\pi^2} + 20$$

To see the corresponding floating-point result use:

```
float(%);
```

$$hP = 42.79919457914721$$

Example 5-22 – Entrance from a reservoir into a long open-channel (Subcritical case) – The uniform flow conditions for a long open channel flow, namely, the depth of flow *y* and the discharge *Q*, are determined by the simultaneous solution of the energy equation at the entrance to the channel and the Manning's equation for the channel. The two equations are listed below:

$$H = y + \frac{Q^2}{2gA^2} \qquad \text{Energy}^1$$

$$Q = \frac{Cu}{n}\frac{A^{5/3}}{P^{2/3}}\sqrt{S_0} \qquad \text{Manning's}^2$$

In these equations *H* is the energy head available at the reservoir, *g* is the acceleration of gravity, *A* is the cross-sectional area, *P* is the wetted perimeter, *Cu* is a constant (=1, if using units of the SI, = 1.486 if using units of the English System), *n* is Manning's resistance coefficient, and S_0 is the channel bed slope. These equations can be written as:

```
EqE : H = y + Q^2/(2*g*A^2);
```

$$H = \frac{Q^2}{2\,g\,A^2} + y$$

```
EqQ : Q = Cu/n*A^(5/3)/P^(2/3)*sqrt(S0);
```

$$Q = \frac{Cu\,A^{5/3}\sqrt{S0}}{n\,P^{2/3}}$$

Consider a rectangular cross section[3] for which $A = by$ and $P = b + 2y$, where *b* is the channel width. For this case, the energy and Manning's equation can be written as:

```
EqER : subst(A=b*y,EqE);
```

$$H = \frac{Q^2}{2\,b^2\,g\,y^2} + y$$

```
EqQR : subst([A=b*y,P=b+2*y],EqQ);
```

$$Q = \frac{b^{5/3}\,Cu\,y^{5/3}\sqrt{S0}}{n\,(2y+b)^{2/3}}$$

1 See Example 5.7
2 See Example 5.12
3 See Example 5.7

To ensure that the two equations are algebraic, we modify equation *EqQR* as follows:

```
EqER1 : 2*b^2*g*y^2*(EqER);
```

$$2\,b^2\,g\,y^2\,H = 2\,b^2\,g\,y^2\left(\frac{Q^2}{2\,b^2\,g\,y^2}+y\right)$$

```
EqER1 : ratsimp(EqER1);
```

$$2\,b^2\,g\,y^2\,H = Q^2 + 2\,b^2\,g\,y^3$$

We also modify equation *EqQR* as follows:

```
EqQR1 : denom(rhs(EqQR))*EqQR;
```

$$n\,(2\,y+b)^{2/3}\,Q = b^{5/3}\,Cu\,y^{5/3}\sqrt{S0}$$

```
EqQR1 : EqQR1^3;
```

$$n^3\,(2\,y+b)^2\,Q^3 = b^5\,Cu^3\,y^5\,S0^{3/2}$$

Now the two resulting equations, *EqER1* and *EqQR1*, are algebraic and can be solved simultaneously with function *algsys*, once the values $b = 5\ ft$, $g = 32.2\ ft/s^2$, $H = 6\ ft$, $n = 0.012$, $Cu = 1.486$, $S0 = 0.000037$, have been incorporated into the equations:

```
EqER1A : subst([b=5,g=32.2,H=6],EqER1);
```

$$9660.000000000002\,y^2 = Q^2 + 1610.0\,y^3$$

```
EqQR1A : subst([b=5,n=0.012,Cu=1.486,S0=0.000037],EqQR1);
```

$$1.728\,10^{-6}\,(2\,y+5)^2\,Q^3 = 0.0023078577471422\,y^5$$

A call to function *algsys* produces the following results:

```
(%i25)  algsys([EqER1A,EqQR1A],[Q,y]);
(%o25)  [ [ Q = - 287.6747967479675 , y = - 2.46429879165141 ] , [ Q = 292.3944233606985 - 4.844075550190367 %i , y =
0.03610096901251 %i - 2.499593622445182 ] , [ Q = 4.844075550190369 %i + 292.3944233606985 , y = - 0.03610096901251
%i - 2.499593622445182 ] , [ Q = - 297.3654390934844 , y = - 2.5365141187926 ] , [ Q = 32.87611225188228 , y =
5.981234866828087 ] , [ Q = 28.57829567607114 %i - 16.62344983203852 , y = 0.016291123756947 %i + 6.009382631909332
] , [ Q = - 28.57829567607115 %i - 16.62344983203853 , y = 6.009382631909332 - 0.016291123756947 %i ] , [ Q = 0 , y = 0 ]
]
```

From the eight solutions found many contain complex numbers, therefore, they are not physically feasible, and one is the trivial solution [*Q=0,y=0*]. There are only two solutions that are real, but only one of them has positive values. The latter is the only solution to the problem that makes physical sense, namely, **[*Q = 32.88 ft³/s*, *y = 5.98 ft*]**.

There is one more check to make on the solution and that is to calculate the flow Froude number, which, for a rectangular channel, is defined as:

$$Fr = \frac{V}{\sqrt{gy}} = \frac{Q}{by\sqrt{gy}} \quad .$$

Using Maxima we calculate the Froude number as:

```
FrEq : Fr = Q/(b*y*sqrt(g*y));
```

$$Fr = \frac{Q}{b\,y\sqrt{g\,y}}$$

```
subst([Q=32.88,y=5.98,b=5,g=32.2],FrEq);
Fr = 0.079246865388445
```

If $Fr < 1.0$, the flow is said to be *subcritical*, and the solution as found [$Q = 32.88\ ft^3/s$, $y = 5.98\ ft$] stands. The case $Fr > 1.0$ is presented in the next Example.

Graphical solution – The graphical solution can be found by plotting the discharge as function of *y* for both the energy equation and the Manning's equation. From the energy equation, starting from:

```
EqER;
```

$$H = \frac{Q^2}{2\,b^2\,g\,y^2} + y$$

we can solve for *Q*:

```
solve(EqER,Q);
```

$$[\ Q = -\sqrt{2}\,b\,y\sqrt{g\,H - g\,y}\ ,\ Q = \sqrt{2}\,b\,y\sqrt{g\,H - g\,y}\ ]$$

Out of these two results we keep only the second one (positive values):

```
ExQ : rhs(%[2]);
```

$$\sqrt{2}\,b\,y\sqrt{g\,H - g\,y}$$

and use it to define a function $Q = fE(y)$ as follows:

```
fE(y) := subst([b=5,g=32.2,H=6],ExQ);
fE(y) := subst( [ b = 5 , g = 32.2 , H = 6 ] , ExQ )
```

or,

```
fE(y);
```

$$5\sqrt{2}\sqrt{193.2 - 32.2\,y}\,y$$

From the Manning's equation, starting from

```
EqQR;
```

$$Q = \frac{b^{5/3}\, Cu\, y^{5/3}\sqrt{S0}}{n\,(2y+b)^{2/3}}$$

we can define a second function $Q = fM(y)$ as follows:

```
fM(y):=subst([b=5,Cu=1.486,S0=0.000037,n=0.012],rhs(EqQR));
```

```
fM( y ):= subst( [ b = 5 , Cu = 1.486 , S0 = 3.6999999999999998 10^-5 , n = 0.012 ] , rhs( EqQR ))
```

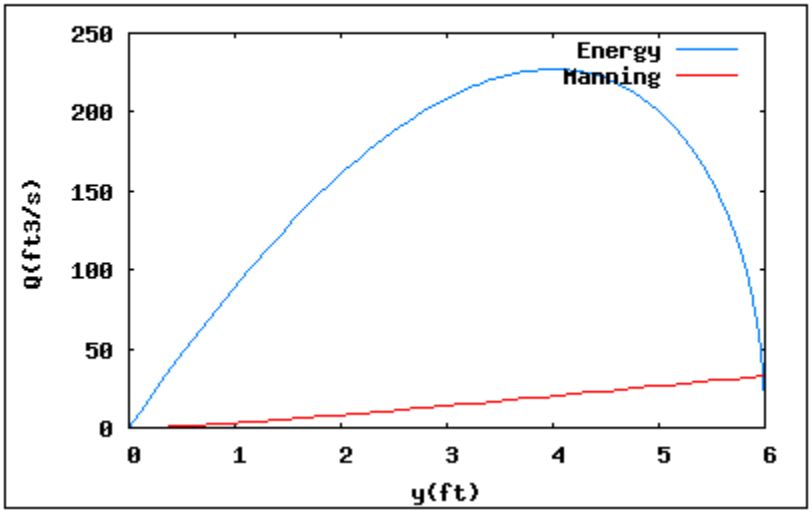
or,

```
fM(y);
```

$$\frac{3.766243800009648\; 5^{2/3}\, y^{5/3}}{(2y+5)^{2/3}}$$

These two functions are to be evaluated in the range $0 < y < H$, with $H = 6\ ft$, for this case, and a plot produced as follows:

```
wxplot2d([fE(y),fM(y)],[y,0,6],[xlabel,"y(ft)"],
[ylabel,"Q(ft3/s)"],[legend,"Energy","Manning"]);
```

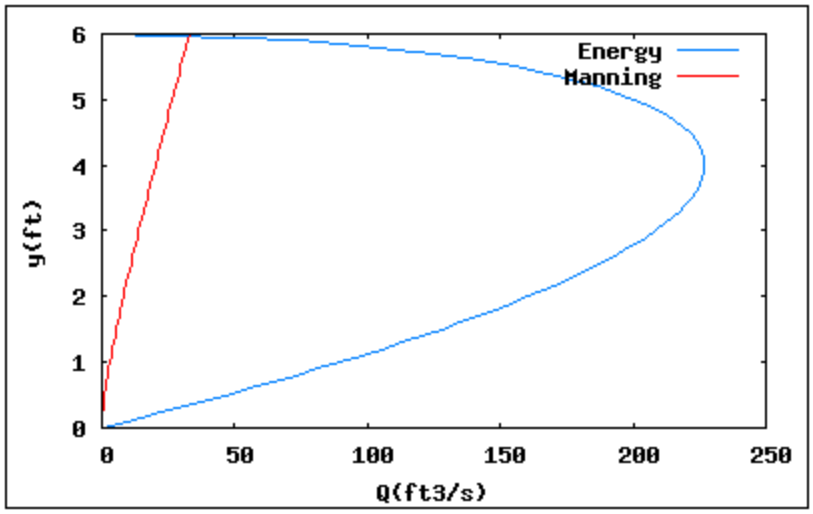


In practice, however, this type of graph is presented with the axes switched. This can be accomplished by creating discrete data sets as follows:

```
yList : makelist(0.1*k,k,0,60) $ nn : length(yList) $
EList : makelist(subst(y=yList[k],fE(y)),k,1,nn) $
MList : makelist(subst(y=yList[k],fM(y)),k,1,nn) $
```

The list *yList* contains values of *y* from *0.1* to *6.0* in increments of *0.1*, while the lists *EList* and *MList* contain the corresponding values of *fE(y)* and *fM(y)*. Then, these lists are plotted using the option *discrete* in command *wxMaxima* as shown below:

```
wxplot2d([[discrete,EList,yList],[discrete,MList,yList]],
[ylabel,"y(ft)"],[xlabel,"Q(ft3/s)"],[legend,"Energy","Manning"]);
```



The *Energy* curve (blue line in the graph) shows a critical depth at the point of maximum discharge *Q*, i.e., close to $y_c \approx 4\ ft$, with $Q = Q_{max} \approx 220\ ft^3/s$. Depths above this value are in the *subcritical* regime, while those below this value are in the *supercritical* regime. The *Manning* curve (red line in the graph) intersects the blue curve a little bit below the value *y* = *6 ft* corresponding to a value of *Q* close to about $30\ ft^3/s$. This intersection point is the solution to the problem, and it clearly shows it to be in the *subcritical* regime.

Example 5-23 – Entrance from a reservoir into a long open-channel (Supercritical case) – The slope used in the previous example, namely, S0 = 0.000037, is small enough that the resulting flow in subcritical. If we repeat the example above, but using S0 = 0.005, we get a supercritical flow solution as shown below. We start from the equations EqER1 and EqQR1:

EqER1;	EqQR1;
$2\,b^2\,g\,y^2\,H = Q^2 + 2\,b^2\,g\,y^3$	$n^3\,(2\,y + b)^2\,Q^3 = b^5\,Cu^3\,y^5\,S0^{3/2}$

Then we replace the following data values, b = 5 ft, g = 32.2 ft/s^2, H = 6 ft, n = 0.012, Cu = 1.486, and S0 = 0.005:

EqER1B : subst([b=5,g=32.2,H=6],EqER1);	EqQR1B : subst([b=5,n=0.012,Cu=1.486,S0=0.005],EqQR1);
$9660.000000000002\,y^2 = Q^2 + 1610.0\,y^3$	$1.728\,10^{-6}\,(2\,y + 5)^2\,Q^3 = 3.625446130566357\,y^5$

The solutions to this system of two equations are the following:

```
algsys([EqER1B,EqQR1B],[Q,y]);
[ [ Q = - 169.327485380117 , y = - 1.53713298791019 ] , [ Q = 205.1063553860142 - 154.2959925848159 %i , y =
1.217694416641699 %i - 1.905879707723448 ] , [ Q = 154.2959925848159 %i + 205.1063553860142 , y = -
1.217694416641699 %i - 1.905879707723448 ] , [ Q = - 631.949860724234 , y = - 4.793823796548592 ] , [ Q =
226.8109452736318 , y = 3.91015854374633 ] , [ Q = 304.9859468358329 %i - 406.5514373406992 , y = 2.570496195196844
%i + 7.116278727916599 ] , [ Q = - 304.985946835831 %i - 406.5514373406992 , y = 7.116278727916599 -
2.570496195196844 %i ] , [ Q = 0 , y = 0 ] ]
```

The proper solution for this case is the pair [Q=*226.81 ft*^{3}/*s*, y = *3.91 ft*]. The corresponding Froude number is calculated as follows:

```
subst([Q=226.81,y=3.91,b=5,g=32.2],FrEq);
Fr = 1.033949011409361
```

Thus, for this case, since Fr > 1.0, the flow is supercritical. The discharge to be used now is that corresponding to critical depth. This requires the simultaneous solution of the the Manning's equation and the critical depth equation, namely,

$$Q=\frac{Cu}{n}\frac{A^{5/3}}{P^{2/3}}\sqrt{S_0}$$ Manning's

and

$$y=\sqrt[3]{\frac{q^2}{g}}=\sqrt[3]{\frac{Q^2}{g\,b^2\,y^2}}$$ Critical depth[4]

The critical depth equation can be manipulated as follows:

```
EqC : y = (q^2/g)^(1/3);
```

$$y=\frac{q^{2/3}}{g^{1/3}}$$

```
EqC : subst(q=Q/b,EqC);
```

$$y=\frac{Q^{2/3}}{b^{2/3}g^{1/3}}$$

Solving for Q requires answering a couple of questions for *Maxima*:

4 See Example 5.8

```
solve(EqC,Q);
Is   b g y   positive, negative, or zero? positive;
Is   b g^(1/3) y   positive, negative, or zero? positive;
```

$$[\,Q = b(g^{1/3}y)^{3/2},\ Q^{1/3} = -b^{1/3}\sqrt{g^{1/3}y}\,]$$

Out of these results we need to retain only the first of the two solutions, i.e.,

```
EqQC : %[1];
```

$$Q = b(g^{1/3}y)^{3/2}$$

When this result gets substituted into equation *EqQR1* we get *EqQR1C*, which is still not algebraic (it has a 9/2 exponent in a *y* term):

```
EqQR1C : subst(EqQC,EqQR1);
```

$$b^3 n^3 (g^{1/3}y)^{9/2} (2y+b)^2 = b^5 Cu^3 y^5 S0^{3/2}$$

If we square this equation we convert it into an algebraic equation, i.e.,

```
EqQR1C : EqQR1C^2;
```

$$b^6 g^3 n^6 y^9 (2y+b)^4 = b^{10} Cu^6 y^{10} S0^3$$

If we now substitute the known values, i.e., *b = 5 ft, g = 32.2 ft/s*^{2}*, H = 6 ft, n = 0.012, Cu = 1.486, and S0 = 0.005*:, we can solve for the critical depth *y*:

```
EqQR1CC : subst([b=5,g=32.2,H=6,n=0.012,Cu=1.486,S0=0.005],EqQR1C);
```

$$0.001557668786688\, y^9 (2y+5)^4 = 13.14385964563857\, y^{10}$$

To avoid excessive output, and since we have a polynomial equation in *y*, we can use function *realroots* to obtain all real solutions:

```
realroots(EqQR1CC);
```

$$[\,y = \frac{2839271}{33554432},\ y = \frac{149878939}{33554432},\ y = 0\,]$$

To see the floating-point values use function *float*:

```
float(%);
[ y = 0.084616869688034 , y = 4.46674045920372 , y = 0.0 ]
```

Of all those values the correct one (corresponding to maximum discharge in the *y-vs-Q* plots shown above) is *y = 4.47 ft*.
The corresponding maximum discharge can be obtained from the *fE(y)* function as indicated below:

```
yc : 4.46674045920372 $ Qmax : subst(y=yc,fE(y));
156.926575130955√2
```

The corresponding floating-point value is:

```
float(%);
221.927690846957
```

Thus, we find that the actual discharge going into the channel is $Q = Q_{max} = 221.93\ ft^3/s$. This value needs to be used in combination with the Manning's equation, namely, *EqQR*,

```
EqQR1;
n^3 (2 y + b)^2 Q^3 = b^5 Cu^3 y^5 S0^3/2
```

to solve for *y*. The equation to use, after replacing the values *b = 5 ft*, *n = 0.012*, *Cu = 1.486*, *S0 = 0.005*, and $Q = 221.93\ ft^3/s$, is the following:

```
EqQR1C : subst([b=5,n=0.012,Cu=1.486,S0=0.005,Q=221.93],EqQR1);
18.8882524046425 (2 y + 5)^2 = 3.625446130566357 y^5
```

Solving this equation, using function *allroots*, produces the result:

```
realroots(EqQR1C);
[ y = 128959699/33554432 ]
```

or,

```
float(%);
[ y = 3.843298524618149 ]
```

Thus, the solution to the present problem is the pair [$Q = 221.93\ ft^3/s$, $y = 3.84\ ft$].

mnewton

Function *mnewton (m*ultiple-equation *newton* solver) allows for the numerical solution of a system of non-algebraic equations. The function needs to be loaded with *load(mnewton)*. The general call to the function is

```
mnewton(<list of equations>,<list of variables>,<initial guesses>)
```

The following examples illustrate the use of function *mnewton* in the numerical solution of systems of equations.

Example 5-24 – Solving a system of two non-algebraic equations – Consider the two non-algebraic equations:

```
Eq1 : x*log(y) + y*log(x) = 4;
```

$$x\log(y)+\log(x)y=4$$

```
Eq2 : x^2*exp(-y) + x*y = 6;
```

$$x^2\,\%e^{-y}+x\,y=6$$

The following calls to function *mnewton* shows different solutions to the system of equations achieved by varying the initial guesses for the variables involved:

```
mnewton([Eq1,Eq2],[x,y],[0.5,0.5]);
```

$$[\,[\,x=1.738322852720863\,,\,y=3.393193023614936\,]\,]$$

```
mnewton([Eq1,Eq2],[x,y],[1.0,0.5]);
```

$$[\,[\,x=2.314394045473058\,,\,y=2.377801246172265\,]\,]$$

```
mnewton([Eq1,Eq2],[x,y],[0.5,1.0]);
```

$$[\,[\,x=2.799515235003295\,10^{-21}\,\%i+1.738322852720863\,,\;y=3.393193023614935-4.573662552564686\,10^{-21}\,\%i\,]\,]$$

```
mnewton([Eq1,Eq2],[x,y],[1.0,1.0]);
```

$$[\,[\,x=2.314394045473057\,,\,y=2.377801246172265\,]\,]$$

```
mnewton([Eq1,Eq2],[x,y],[1.5,1.0]);
```

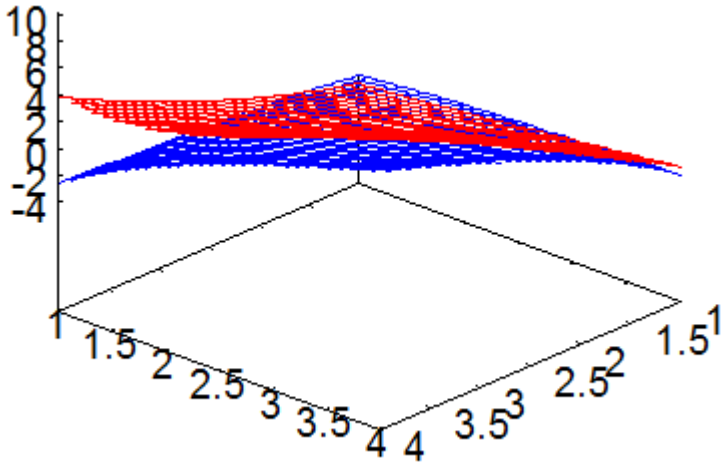
$$[\,[\,x=2.314394045473057\,,\,y=2.377801246172265\,]\,]$$

Using the *draw* package (http://www.telefonica.net/web2/biomates/maxima/gpdraw/), a plot of the two equations can be produced with[5]:

5 See Chapter 3 for more on graphs. Read the *Maxima Manual* (*Help>Maxima help*) to learn more about the *draw* package.

```
load(draw)$
draw3d(key="logs",color=blue,explicit(x*log(y)+log(x)*y-4,x,1,4,y,1,4),
key="exp",color=red,explicit(x^2*%e^(-y)+x*y-6,x,1,4,y,1,4));
```

The graph below shows the two functions as surfaces of two different colors. The figure suggests that the two curves intersect, however, the points of intersection are not clearly defined.



Example 5-25 – Pump-pipeline system solution revisited – In *Example* 5-21 we introduced the equations of the system and of the pump in the solution of a pump-pipeline system. These equations are the pump equation:

$$h_P = a + bQ + cQ^2$$

and the system equation:

$$h_P = H + \frac{8Q^2}{\pi^2 g D^4}\left(f\frac{L}{D} + \Sigma K\right) \quad ,$$

In these equations, coefficients *a, b, c* are obtained empirically from laboratory testing, h_P is the pump head, *Q* is the discharge, *g* is the acceleration of gravity, *L* is the length of the pipeline, *D* is the diameter of the pipeline, *f* is a friction factor, and ΣK is the sum of coefficients due to local losses in the pipeline (e.g., valves, elbows, entrance from reservoir, discharge into reservoir, etc.).

In Example 5-21 we treated the friction factor as a constant, and, as a consequence, we were able to develop two algebraic equations to solve simultaneously. In reality, the friction factor *f* is a function of two parameters: (1) the relative roughness, *e*/*D*, and, (2) the Reynolds number, $R = VD/\nu$, where *e* is the absolute roughness of the pipeline material, *V* is the mean flow velocity in the pipeline, and ν is the kinematic viscosity of the fluid.

In this development we will use the *Swamee-Jain* equation to calculate the friction factor:

$$f = \frac{0.25}{\log_{10}^2\left(\frac{e}{3.7D} + \frac{5.74}{\mathbf{R}^{0.9}}\right)}.$$

In Chapter 1, page 1-8, we defined the *log10* function as

$$log_{10}(x) = \frac{log(x)}{log(10)} \quad ,$$

thus, the Swamee-Jain equation will be re-written as

$$f = \frac{0.25 \cdot log(10)}{log^2\left(\frac{e}{3.7D} + \frac{5.74}{\boldsymbol{R}^{0.9}}\right)} = \frac{0.5756}{log^2\left(\frac{e}{3.7D} + \frac{5.74}{\boldsymbol{R}^{0.9}}\right)} \quad .$$

Thus, the system of equations to solve now includes the pipe equation, the system equation, and the Swamee-Jain equation. We still need to consider the Reynolds number and the equation of continuity:

$$\boldsymbol{R} = \frac{V \cdot D}{\nu} \quad \text{, and} \quad Q = V \cdot \frac{\pi \cdot D^2}{4} \quad .$$

which can be combined into, $\boldsymbol{R} = \frac{4 \cdot Q}{\pi \cdot \nu}$. This result, in turn, can be included into the Swamee-Jain equation:

$$f = \frac{0.5756}{log^2\left(\frac{e}{3.7D} + \frac{5.74}{\left(\frac{4Q}{\pi \nu}\right)^{0.9}}\right)} \quad .$$

The parameters of the problem are: *H = 20 m, g = 9.81 m/s*^{2}*, D = 2 m, L = 100 m, SK = 2.5, a = 60, b = 0, c = -0.012, e = 0.000001 m, ν = 1×10* $^{-6}$ *m*^{2}*/s.* The following *Maxima* commands allows us to set up the equations needed to solve the problem, namely, the pump equation:

```
EqP : hP = a + b*Q + c*Q^2;
```

$$hP = c\,Q^2 + b\,Q + a$$

the system equation:

```
EqS : hP = H + 8*Q^2/(%pi^2*g*D^4)*(f*L/D+SK);
```

$$hP = \frac{8\,Q^2\left(SK + \frac{f\,L}{D}\right)}{\pi^2\,g\,D^4} + H$$

and, the Swamee-Jain equation:

```
EqSJ : f = 0.5756/(log(e/(3.7*D)+5.74/((4*Q)/(%pi*nu))^0.9))^2;
```

$$f = \frac{0.5756}{\log\left(\frac{4.618413198590666}{\left(\frac{Q}{\nu}\right)^{0.9}} + \frac{0.27027027027027\,e}{D}\right)^2}$$

Replacing the known values given above we have the pump equation:

```
EqP1 : subst([a=60,b=0,c=-0.012],EqP);
```

$$hP = 60 - 0.012\,Q^2$$

the system equation:

```
EqS1 : subst([H=20,g=9.81,SK=2.5,D=2,L=100],EqS);
```

$$hP = \frac{0.050968399592253\,(50\,f + 2.5)\,Q^2}{\pi^2} + 20$$

and the Swamee-Jain equation:

```
EqSJ1 : subst([D=2,e=0.000001,nu=1*10^(-6)],EqSJ);
```

$$f = \frac{0.5756}{\log\left(\frac{1.8386234109378571\,10^{-5}}{Q^{0.9}} + 1.3513513513513512\,10^{-7}\right)^2}$$

The following call to function *mnewton* produces a solution to the system of three equations:

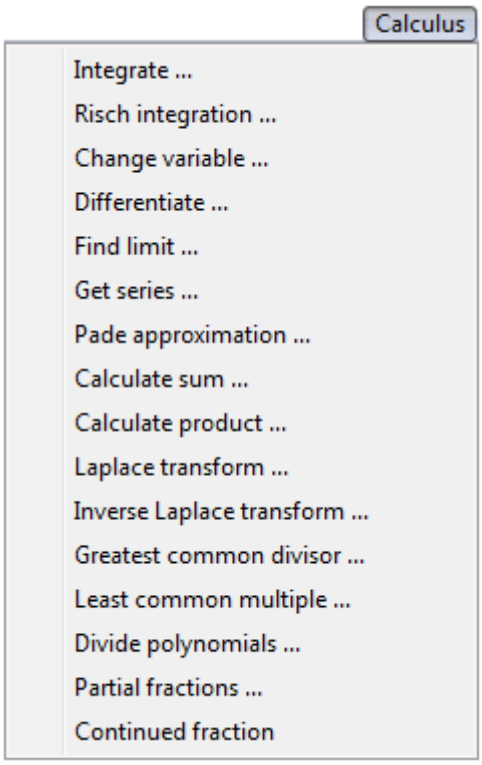
```
mnewton([EqP1,EqS1,EqSJ1],[hP,Q,f],[40,40,0.01]);
[ [ hP = 41.29792838750289 , Q = 39.47791746923959 , f = 0.0029246064096707 ] ]
```

6 Calculus applications in *Maxima*

In this chapter we present examples of functions included in the *Calculus* menu in the *wxMaxima* interface, as well as other calculus applications.

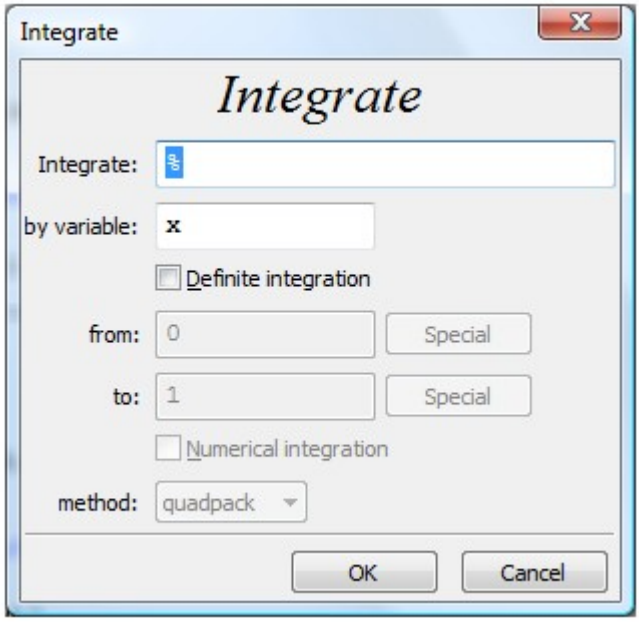
Calculus functions in the *Calculus* menu

The *Calculus* menu in the *wxMaxima* interface is shown in the figure to the right. In this section we presents examples of most of the items in this menu. However, it should be pointed out that the last five items in the menu, namely, from *Greatest common divisor...* to *Continued fraction*, were addressed in Chapter 3 of this book. Therefore, we will only present examples of the remaining items in the *Calculus* menu.



Integrate...

The *Calculus* > *Integrate* ... item produces a dialogue form as shown below:



it to *Indefinite integral* - The integrand, corresponding to the *Integrate:* field, is set, by default, to the last entry (*%*), but can be replaced by any expression. The variable of integration is set, by default, *x*. Next, there is an option to select *Definite integration*, the default being an indefinite integral. As a first example consider the case of a indefinite integral for the expression *1/(1+x^2)*, by entering that expression in the *Integrate:* field, and pressing the [OK] button. The result is the following input and output:

```
integrate(1/(1+x^2), x);
atan(x)
```

This is to say,

$$\int \frac{dx}{1+x^2} = atan(x) \quad .$$

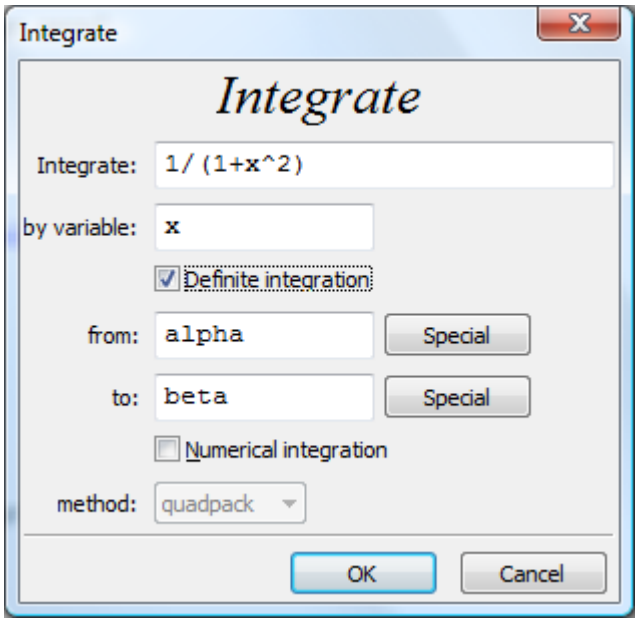
If you want to show the integral before and after evaluation, use the apostrophe (') before the *integrate* command, then use *ev*(*%*,*nouns*), as illustrated in the following example:

```
'integrate(1/(1+x^2),x);ev(%,nouns);
```

$$\int \frac{1}{x^2+1}\,dx$$

```
atan(x)
```

Definite integral - Consider now the definite integral defined in the following input dialogue form for the menu item *Calculus* > *Integrate*...



After pressing the [OK] button the result is the following (after entering *positive* at the question issued by *Maxima*):

```
integrate(1/(1+x^2), x, alpha, beta);
Is  β - α  positive, negative, or zero? positive;
atan(β) - atan(α)
```

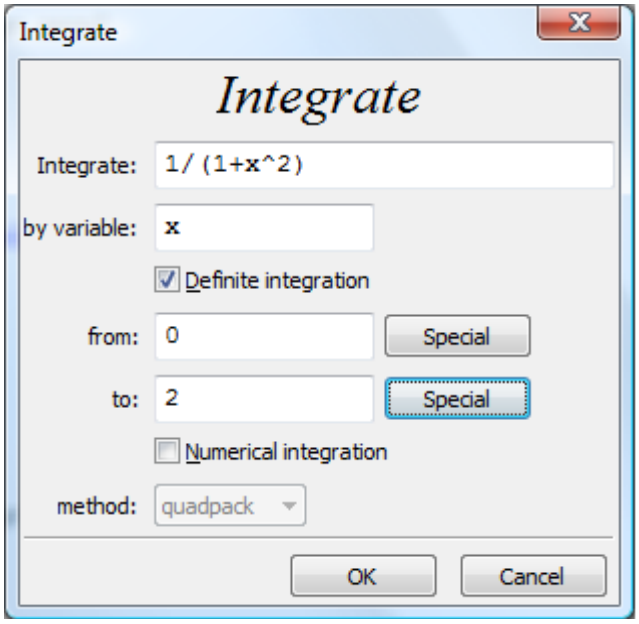
This definite integral can also be entered directly into the *wxMaxima INPUT* line as follows:

```
'integrate(1/(1+x^2),x,alpha,beta);ev(%,nouns);
```

$$\int_{\alpha}^{\beta} \frac{1}{x^2+1}\,dx$$

```
Is  β - α  positive, negative, or zero? positive;

atan(β) - atan(α)
```

The following example shows a definite integral with numeric limits:



This produces the result:

```
integrate(1/(1+x^2), x, 0, 2);

atan(2)
```

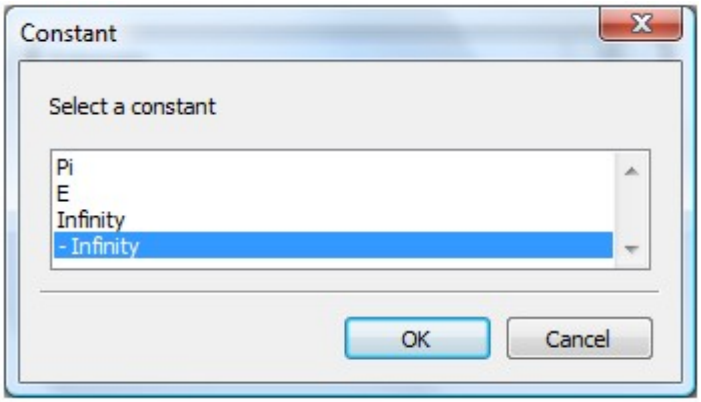
An alternative way to enter this expression directly into the *INPUT* line, so that the integral, and the result are both shown, is the following:

```
'integrate(1/(1+x^2),x,0,2);ev(%,nouns);
```

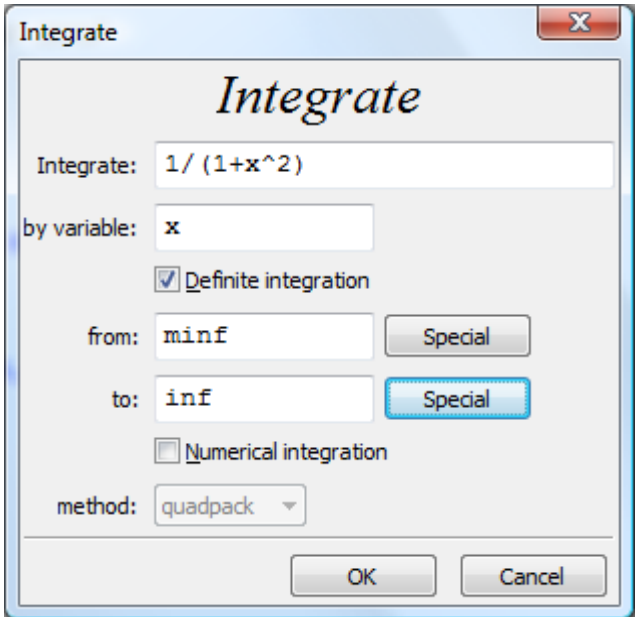
$$\int_{0}^{2} \frac{1}{x^2+1}\,dx$$

```
atan(2)
```

Improper integrals - Notice that in the dialogue form shown above there is a button labeled *Special* attached to each of the limits fields. Clicking on this button provides access to the special values shown below:



These entries can be used, for example, to generate the following dialogue entry form:



which produces the following result:

```
integrate(1/(1+x^2), x, minf, inf);
π
```

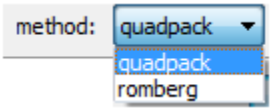
Alternatively, you can enter the improper integral as follows:

```
'integrate(1/(1+x^2),x,minf,inf);ev(%,nouns);
```

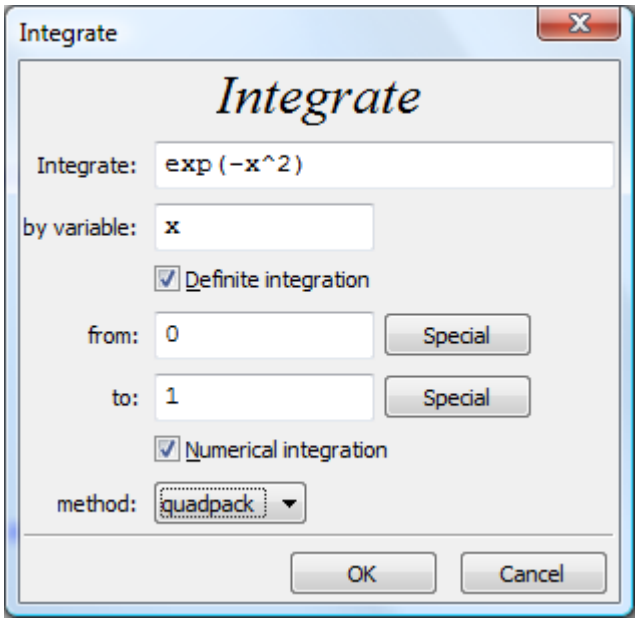
$$\int_{-\infty}^{\infty} \frac{1}{x^2+1}\, dx$$

$$\pi$$

Numerical integration - You may also have noticed that the definite integral dialogue form allows for numerical integration, including one of the following two functions (or methods):



Consider the following numerical integration:



The result is the following call to function *quad_qags*:

```
quad_qags(exp(-x^2), x, 0, 1);
[ 0.74682413281243 , 8.2954620176770253 10^-15 , 21 , 0 ]
```

The outputs from this function call are four numbers representing:

1. The numerical value of the integral
2. The estimated absolute error of the numerical integration
3. The number of integrand evaluations required to produce the numerical value
4. An error code representing the following options:

Error code	Meaning
0	No problems encountered
1	Too many sub-intervals tried
2	Excessive roundoff error detected
3	Extremely bad integrand behavior
6	Input is invalid

The numerical integration shown above can also be accomplished by entering the following:

```
integrate(exp(-x^2), x, 0, 1);float(%);
```

$$\frac{\sqrt{\pi}\,\mathrm{erf}(1)}{2}$$

```
0.74682413281243
```

If you want to show the integral before evaluation use:

```
'integrate(exp(-x^2), x, 0, 1);ev(%,nouns);float(%);
```

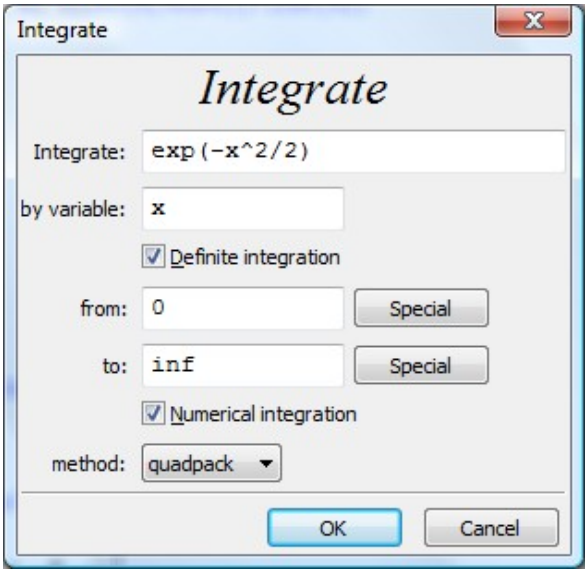
$$\int_0^1 \%e^{-x^2}\,dx$$

$$\frac{\sqrt{\pi}\,\mathrm{erf}(1)}{2}$$

```
0.74682413281243
```

NOTE: *quad functions* - Function *quad_qags*, used in this numerical integration example, belongs to a family of numerical integration functions that includes also functions *quad_qag*, *quad_qagi*, *quad_qawc*, *quad_qawo*, and *quad_qaws*. Details on the operation of these functions can be found in the *Maxima Manual*, available in the menu item *Help* > *Maxima help*. Do a search for *'quad'* in the *Search* tag of the *Maxima Manual* window, to check the individual functions.

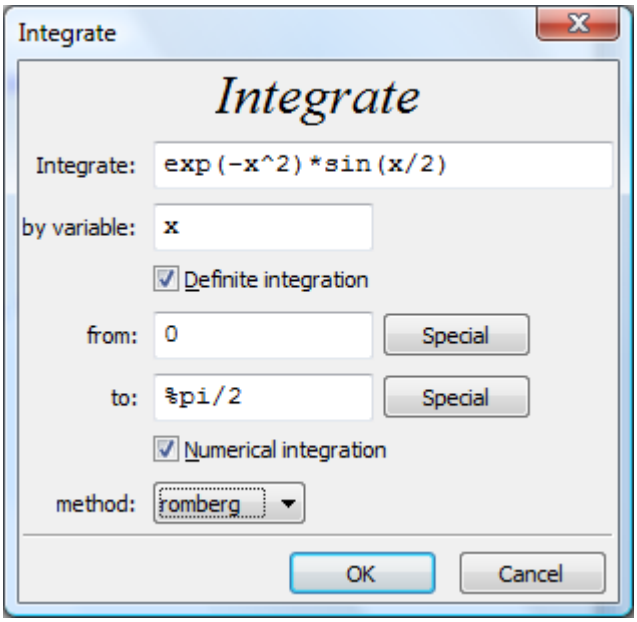
Numerical integration of an improper integral - Selecting, for example, a numerical integration with infinite limits, with the *quad* option selected, results in the activation of function *quad_qagi*:




```
quad_qagi(exp(-x^2/2), x, 0, inf);
```

$$[\ 1.2533141373155\ ,\ 3.4421406678309006\ 10^{-9}\ ,\ 135\ ,\ 0\ ]$$

Romberg integration - Consider the following numerical integration using the Romberg method (reference, e.g., http://en.wikipedia.org/wiki/Romberg's_method).

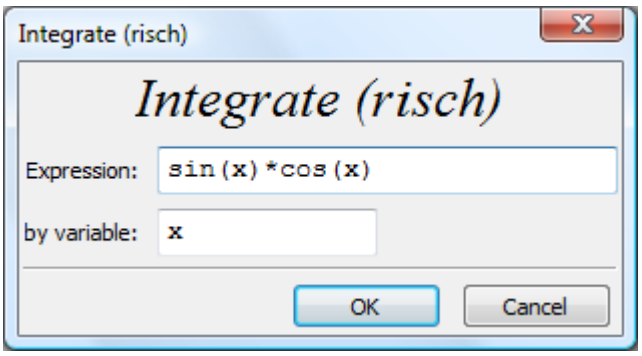


The result is:

```
romberg(exp(-x^2)*sin(x/2), x, 0, %pi/2);
0.22156068518967
```

Risch integration...

The *Risch integration*... menu item activates function *risch* which is used to calculate indefinite integrals by the Risch approach (http://en.wikipedia.org/wiki/Risch_algorithm). The following is an example of this application:



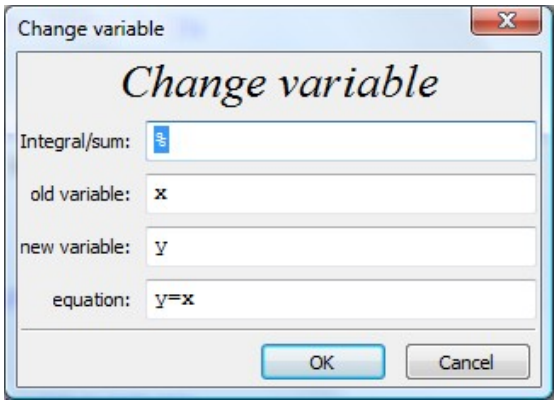
The result is the following:

```
risch(sin(x)*cos(x), x);
```

$$-\frac{\cos(2x)}{4}$$

Change variable...

This menu item from the *Calculus* menu can be used to produce change of variables in a symbolic integral or summation. When this menu item is activated you get the following dialogue form:



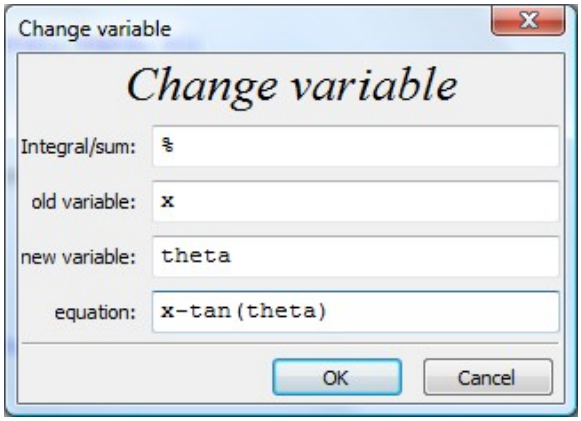
The *Integral/sum* field, which is set by default to the last entry (%), must refer to an integral or summation. The reference to the *old variable* and *new variable* fields is straightforward (the form suggests that old variable is *x* and new variable *y*), however, the suggested value in the *equation* field is misleading. Instead of an equation of the form *y* = *x*, the entry in this field should be of the form *y* - *x*, i.e., a relationship of the form *f(x,y)*, such that *f(x,y)* = *0*.

Before trying the following example, cancel the dialogue form, and enter the following symbolic (non evaluated) integral:

```
'integrate(x^2/(1+x^2),x,0,2);
```

$$\int_0^2 \frac{x^2}{x^2+1}\,dx$$

Then, activate the menu item *Calculus* > *Change variable* ..., and enter the following values in the dialogue form:



This results in the integral:

```
changevar(%, x-tan(theta), theta, x);
```

$$\int_{0}^{\operatorname{atan}(2)} \frac{\sec(\theta)^2 \tan(\theta)^2}{\tan(\theta)^2 + 1}\, d\theta$$

In the resulting integrand you may recognize the trigonometric identity, $sec^2\theta = 1 + tan^2\theta$, which would transform this integral into:

```
'integrate(tan(theta)^2,theta,0,atan(2));
```

$$\int_{0}^{\operatorname{atan}(2)} \tan(\theta)^2\, d\theta$$

An example of a indefinite integral is shown next:

```
'integrate(1/sqrt(1+x),x);
```

$$\int \frac{1}{\sqrt{x+1}}\, dx$$

```
changevar(%,u-x-1,u,x);
```

$$\int \frac{1}{\sqrt{u}}\, du$$

The following change of variable on an indefinite integral uses a trigonometric substitution:

```
'integrate(1/sqrt(1-x^2),x);
```

$$\int \frac{1}{\sqrt{1-x^2}}\,dx$$

```
changevar(%,x=sin(theta),theta,x);
```

$$\int \frac{\cos(\theta)}{\sqrt{1-\sin(\theta)}\sqrt{\sin(\theta)+1}}\,d\theta$$

These change of variables are useful if you are learning how to simplify and calculate integrals by hand. Using *Maxima* there is no need to use these substitutions to calculate the integrals as illustrated by the following examples:

```
'integrate(x^2/(x^2+1),x,0,2);ev(%,nouns);
```

$$\int_0^2 \frac{x^2}{x^2+1}\,dx$$

$$2-\operatorname{atan}(2)$$

```
'integrate(1/sqrt(1+x),x);ev(%,nouns);
```

$$\int \frac{1}{\sqrt{x+1}}\,dx$$

$$2\sqrt{x+1}$$

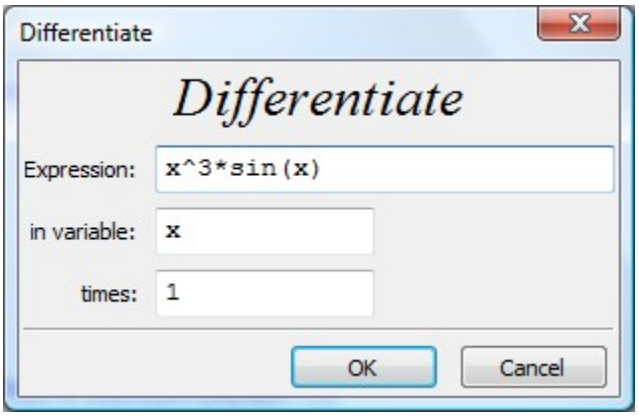
```
'integrate(1/sqrt(1-x^2),x);ev(%,nouns);
```

$$\int \frac{1}{\sqrt{1-x^2}}\,dx$$

$$\operatorname{asin}(x)$$

<u>Differentiate...</u>

The *Calculus* menu option *Differentiate*... produces a dialogue form conducive to calculating derivatives, e.g.,



This action produces the command:

```
diff(x^3*sin(x), x);
```

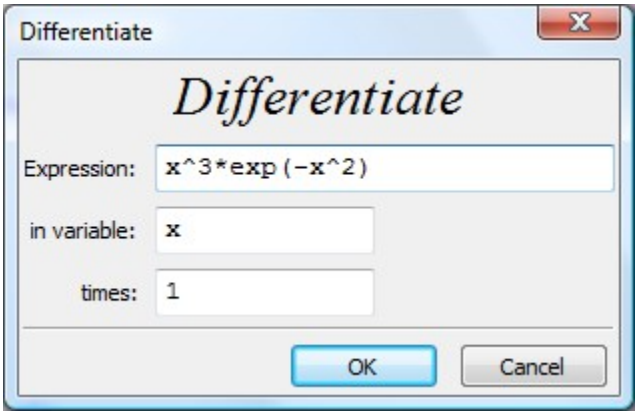
$$3x^2\sin(x)+x^3\cos(x)$$

Thus, the command *diff(f(x),x)* produces the derivative df/dx. For higher-order derivatives, say, derivative of order n, d^nf/dx^n, the corresponding command is *diff(f(x),x,n),* for example,

```
diff(x^3*exp(-x^2), x, 3);
```

$$-8x^6\,\%e^{-x^2}+48x^4\,\%e^{-x^2}-54x^2\,\%e^{-x^2}+6\,\%e^{-x^2}$$

Or you can use the *Calculus* > *Differentiate*... menu item, e.g.,

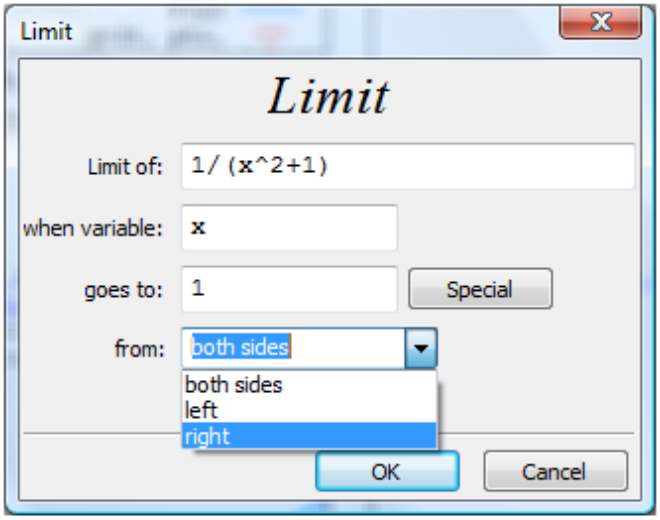


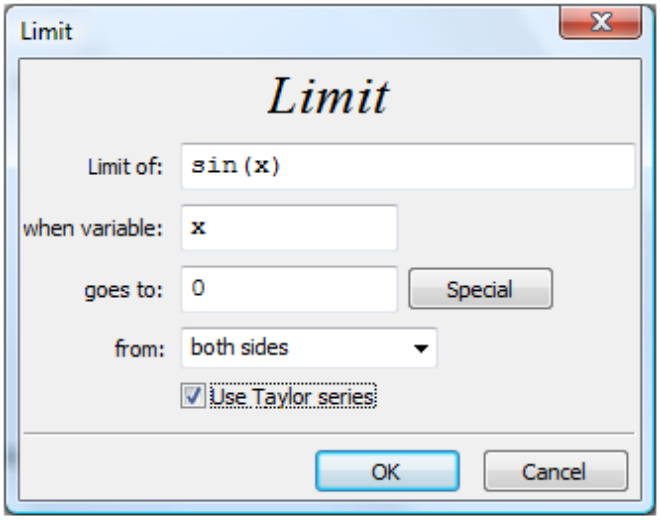
To show the differentiation operation and its result use, for example,

`'diff(x^3*sin(x), x);ev(%,nouns);`	`'diff(x^3*exp(-x^2), x, 3);ev(%,nouns);`
$\frac{d}{dx}(x^3\sin(x))$	$\frac{d^3}{dx^3}\left(x^3\,\%e^{-x^2}\right)$
$3x^2\sin(x)+x^3\cos(x)$	$-8x^6\,\%e^{-x^2}+48x^4\,\%e^{-x^2}-54x^2\,\%e^{-x^2}+6\,\%e^{-x^2}$

Find limit...

Calculus menu item *Find limit*... allows the calculation of limits of functions. The menu item produces a dialogue form that allows the user to define the function of interest, and the point where the limit is calculated. The dialogue also allows to select if the limit is from the left, from the right, or from both sides. An option exists also to use the Taylor series expansion of the function in calculating the limit. The following figure illustrates two cases of limit calculations:






```
limit(1/(x^2+1), x, 1);
```

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

```
tlimit(sin(x), x, 0);
```

$$0$$

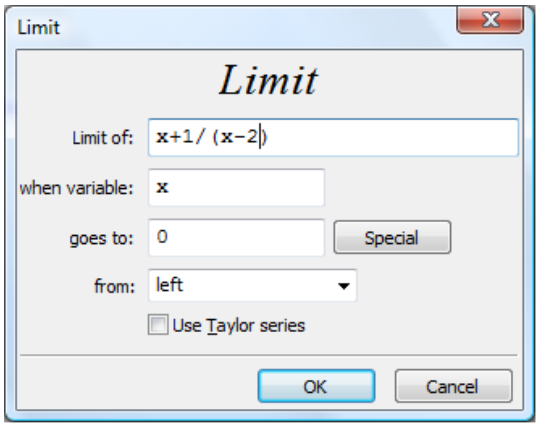
The limit expression can be made visible, for the first case only, if we use an apostrophe (') for the *limit* command. Function *ev* is then used to evaluate the corresponding limit:

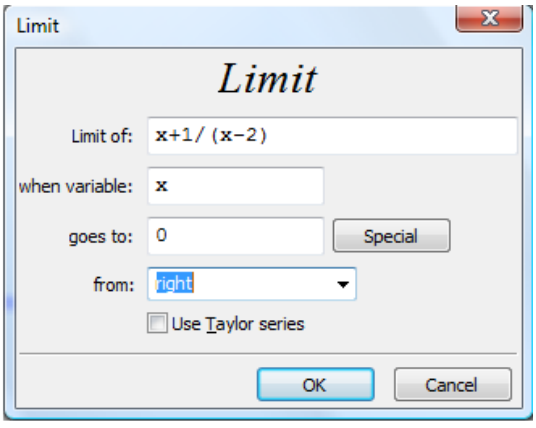
```
'limit(1/(x^2+1), x, 1);ev(%,nouns);
```

$$\lim_{x \to 1} \frac{1}{x^2+1}$$

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

Examples of limits from the left and the right are shown next:






```
limit(x+1/(x-2), x, 0, minus);
```

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

```
limit(x+1/(x-2), x, 0, plus);
```

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

L'Hopital's rule. L'Hopital's rule is automatically incorporated in the calculation of limits, e.g.,

```
'limit((x^2-2*x+1)/(x-1),x,1);ev(%,nouns);
```

$$\lim_{x \to 1} \frac{x^2 - 2x + 1}{x - 1}$$

```
0
```

This is a case in which, without any simplification, both the numerator and denominator limits are zero as *x* goes to *1*. L'Hopital's rule indicates that the following is true:

```
'limit('diff(x^2-2*x+1,x)/'diff(x-1,x),x,1);
```

$$\lim_{x \to 1} \frac{\frac{d}{dx}(x^2 - 2x + 1)}{\frac{d}{dx}(x - 1)}$$

Evaluating the derivatives before the limit we get:

```
'limit(diff(x^2-2*x+1,x)/diff(x-1,x),x,1);
```

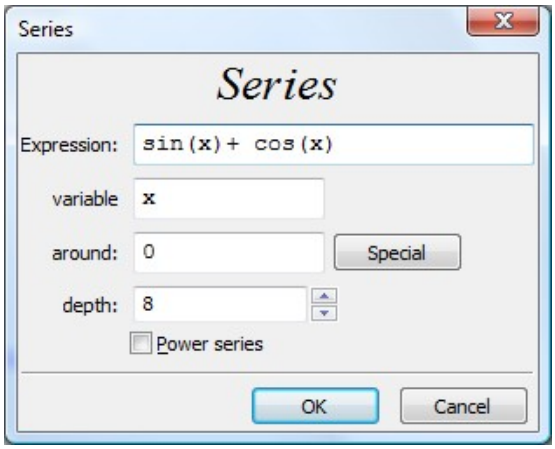
$$\lim_{x \to 1} 2x - 2$$

The limit is calculated as:

```
limit(diff(x^2-2*x+1,x)/diff(x-1,x),x,1);
0
```

Get series...

Calculus menu item *Get series...* allows the user to obtain a Taylor series expansion of an expression. The corresponding dialogue form is shown below in which we seek the expansion of the expression *sin(x)+cos(x)* around *x = 0* up to a power of order *8*.



The resulting *Maxima* input and output is shown below:

```
taylor(sin(x)+ cos(x), x, 0, 8);
```

$$1+x-\frac{x^2}{2}-\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}-\frac{x^6}{720}-\frac{x^7}{5040}+\frac{x^8}{40320}+\ldots$$

If the option *Power series* is activated in the *Series* dialogue form, the result, for the same function as above, is as follows:

```
niceindices(powerseries(%, x, 0));
```

$$\frac{x^8-8x^7-56x^6+336x^5+1680x^4-6720x^3-20160x^2+40320x+40320}{40320}$$

The following example shows the Taylor series expansion for the function log(x) about x = 1 up to a power of order 8:

```
taylor(log(x), x, 1, 8);
```

$$x-1-\frac{(x-1)^2}{2}+\frac{(x-1)^3}{3}-\frac{(x-1)^4}{4}+\frac{(x-1)^5}{5}-\frac{(x-1)^6}{6}+\frac{(x-1)^7}{7}-\frac{(x-1)^8}{8}+\ldots$$

Using the option *Power series* in the dialogue form produces the following infinite series:

```
niceindices(powerseries(log(x), x, 1));
```

$$-\sum_{i=1}^{\infty}\frac{(-1)^i(x-1)^i}{i}$$

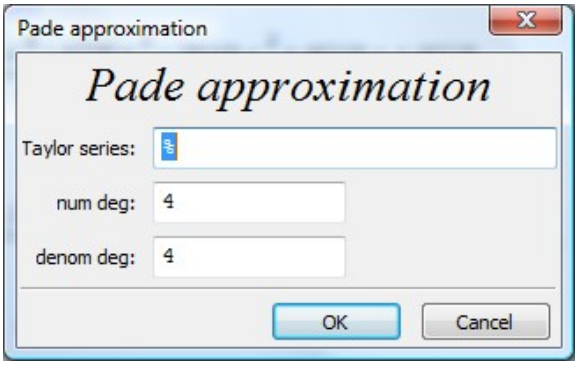
<u>Pade approximation...</u>

A Padé approximation of a Taylor series consists of a fraction in which both the numerator and the denominator are powers of *x*. Since a Taylor series is required, we first set up a Taylor series for the function *sin(x)* about *x = 0* up to a power of order *8*, i.e.,

```
taylor(sin(x),x,0,8);
```

$$x-\frac{x^3}{6}+\frac{x^5}{120}-\frac{x^7}{5040}+\ldots$$

Then, we activate the *Calculus* menu item *Pade approximation...* which produces the following dialogue form:



In this form, we refer to the most recent result (*%*), and request a Padé approximation fraction with the largest degrees of ***x*** in both numerator and denominator being *4*. The result is the following command and fraction:

```
pade(%, 4, 4);
```

$$\left[-\frac{620\,x^3 - 5880\,x}{11\,x^4 + 360\,x^2 + 5880}\right]$$

Another example of a Padé approximation for the Taylor series:

```
taylor(exp(x)*sin(x),x,0,10);
```

$$x + x^2 + \frac{x^3}{3} - \frac{x^5}{30} - \frac{x^6}{90} - \frac{x^7}{630} + \frac{x^9}{22680} + \frac{x^{10}}{113400} + \ldots$$

is the following:

```
pade(%,6,8);
```

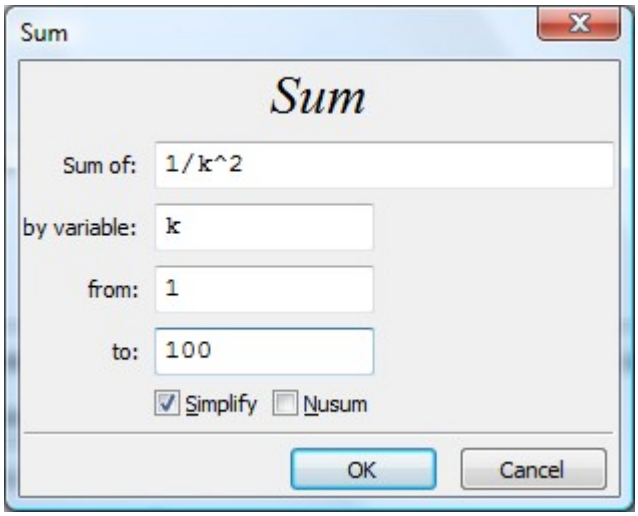
$$\left[\frac{3213000\,x^2 + 9865800\,x}{607\,x^8 - 3260\,x^7 + 19860\,x^6 - 84000\,x^5 + 354060\,x^4 - 1146600\,x^3 + 3364200\,x^2 - 6652800\,x + 9865800},\right.$$
$$\frac{1147230\,x^3 + 6161400\,x^2 + 8391600\,x}{2425\,x^7 - 7869\,x^6 + 41370\,x^5 - 76860\,x^4 + 163170\,x^3 + 580230\,x^2 - 2230200\,x + 8391600},$$
$$-\frac{1527750\,x^4 + 4957470\,x^3 - 23360400\,x^2 - 76242600\,x}{28291\,x^6 - 227430\,x^5 + 1448790\,x^4 - 6410880\,x^3 + 22510530\,x^2 - 52882200\,x + 76242600},$$
$$\left.\frac{1489\,x^5 + 11970\,x^4 + 13440\,x^3 - 113400\,x^2 - 260820\,x}{121\,x^5 - 1395\,x^4 + 9870\,x^3 - 47040\,x^2 + 147420\,x - 260820}, -\frac{2299\,x^6 + 26505\,x^5 + 95340\,x^4 - 69930\,x^3 - 1168020\,x^2 - 1908900\,x}{1845\,x^4 - 22890\,x^3 + 174510\,x^2 - 740880\,x + 1908900}\right]$$

To learn more about Padé approximations visit:

http://en.wikipedia.org/wiki/Pad%C3%A9_approximant

Calculate sum...

The *Calculate sum...* item in the *Calculus* menu produces a dialogue form as follows:



For this example, we select the summation of the term $1/k^2$, from $k = 1$ to $k = 100$. With the *Simplify* option on, to produce:

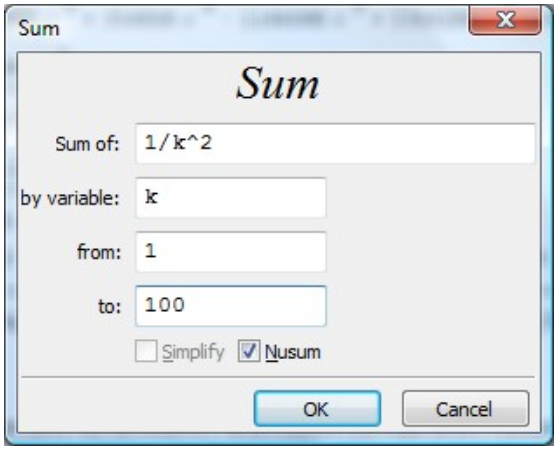
```
sum(1/k^2, k, 1, 100), simpsum;
```

$$\frac{158950869413303787311229792851755385970238349854370985988943283480381813109036990 1}{972186144434381030589657976672623144161975583995746241782720354705517986165248000}$$

The floating-point value of this result is calculated with function *float*:

```
float(%);
1.634983900184893
```

With the *Nusum* option selected:

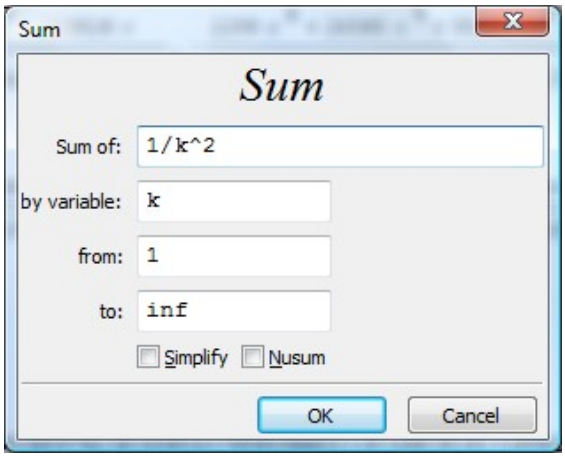


the previous summation produces the same result as above:

```
nusum(1/k^2, k, 1, 100);
```

$$\frac{158950869413303787311229792851755385970238349854370985988943283480381813109036990 1}{972186144434381030589657976672623144161975583995746241782720354705517986165248000}$$

An infinite sum example is shown below:



This sum produces the following result:

```
sum(1/k^2, k, 1, inf);
```

$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$

With the *Simplify* option selected, the value of the summation is calculated as:

```
sum(1/k^2, k, 1, inf), simpsum;
```

$$\frac{\pi^2}{6}$$

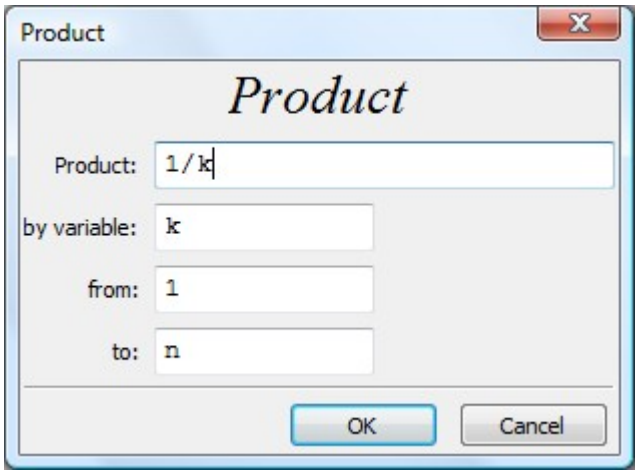
Selecting the option *Nusum* in the *Sum* dialogue form produces an infinite (incorrect) result. This is so because a numerical approximation to an infinite sum is not an appropriate operation.

```
nusum(%, k, 1, inf);
```

$$\frac{\pi^2 \infty}{6}$$

Calculate product...

The dialogue form for the *Calculate product*... item in the *Calculus* menu is similar to that of the summation calculations shown above, except that there are no options to choose, just entering the parameters of the product, e.g.,



This entry represents the product to *n* elements of the quantity *1/k*, i.e.,

```
product(1/k,k,1,n);
```

$$\prod_{k=1}^{n} \frac{1}{k}$$

Replacing the upper limit with a specific value (i.e., *k = 10*) produces:

```
product(1/k,k,1,10);
```

$$\frac{1}{3628800}$$

Here is the product of all integers from 1 to 5:

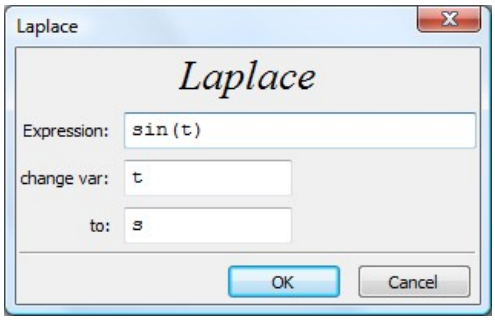
```
product(k,k,1,5);
120
```

which is, by definition, the factorial of 5:

```
5!;
120
```

Laplace transform...

Laplace transforms are a type of integral transforms used in the solution of ordinary differential equations (see http://en.wikipedia.org/wiki/Laplace_transform). The *Calculus* > *Laplace transform*... menu item produces the following dialogue interface.



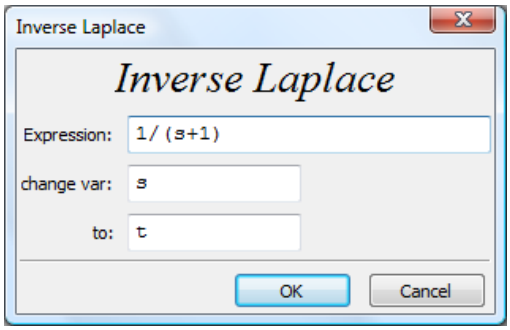
This example represents the Laplace transform of the sine function, i.e.,

```
laplace(sin(t), t, s);
```

$$\frac{1}{s^2+1}$$

Inverse Laplace transform...

As the name indicates, the inverse Laplace transform is the opposite operation to the Laplace transform (http://en.wikipedia.org/wiki/Inverse_Laplace_transform). The *Calculus* > *Inverse Laplace transform*... menu item produces the following dialogue interface:



This example shows that the negative exponential function e^{-t} is the inverse Laplace transform of the fraction *1/(s+1)*:

```
ilt(1/(s+1), s, t);
```

$$\%e^{-t}$$

NOTE: The Laplace transform and its inverse belong in a Chapter on differential equations (ODEs). They were included here for the sake of completeness in describing the functions available in the *Calculus* menu. The *Equations* menu includes an item on solving ordinary differential equations with Laplace transforms (*Equations* > *Solve ODE with Laplace*...). See a simple application in Chapter 1. The solution of ODEs will be addressed in a subsequent chapter.

Examples of applications in calculus
The following examples show specific applications of the different calculus functions presented above.

The limit of *sin(x)/x* as *x* approaches *zero*
Consider the limit of function *f(x)* = *sin(x)/x* as x approaches zero. Direct evaluation of this function at *x* = *0* is an undefined value (0/0), however, the limit is equal to 1:

```
'limit(sin(x)/x,x,0);ev(%,nouns);
```

$$\lim_{x \to 0} \frac{\sin(x)}{x}$$

$$1$$

The limit is, of course, the same whether zero is approached from the left or from the right:

```
'limit(sin(x)/x,x,0,plus);ev(%,nouns);
```

$$\lim_{x \to 0+} \frac{\sin(x)}{x}$$

$$1$$

```
'limit(sin(x)/x,x,0,minus);ev(%,nouns);
```

$$\lim_{x \to 0-} \frac{\sin(x)}{x}$$

$$1$$

Application of the L'Hopital's rule justifies this result:

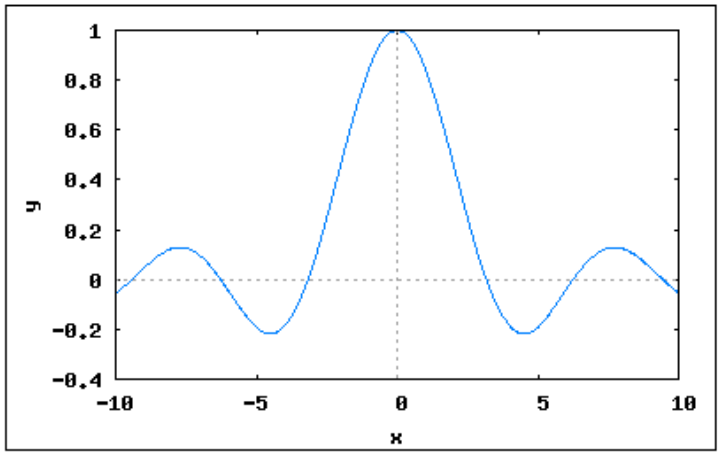
```
'limit('diff(sin(x),x)/'diff(x,x),x,0);ev(%,nouns);
```

$$\lim_{x \to 0} \frac{d}{dx}\sin(x)$$

$$1$$

The plot of the function shows that the value of *f(0)* is indeed 1:

```
wxplot2d(sin(x)/x,[x,-10,10],[xlabel,"x"],[ylabel,"y"]);
```



The derivative as a limit

By definition the derivative of a function *f(x)* is the following limit:

$$\frac{df}{dx} = \lim_{x \to 0} \frac{f(x+h) - f(x)}{h}$$

Let's try some examples:

Example 1 – *f(x)* = *sin(x)*, *f'(x)* = *cos(x)*:

```
'limit((sin(x+h)-sin(x))/h, h, 0);ev(%,nouns);
```

$$\lim_{h \to 0} \frac{\sin(x+h) - \sin(x)}{h}$$

```
Is  sin(x)  positive, negative, or zero? positive;
Is  cos(x)  positive, negative, or zero? positive;
```

$$\cos(x)$$

```
diff(sin(x),x);
```

$$\cos(x)$$

Example 2 – *f(x)* = $\sqrt{(x)}$, *f'(x)* =

```
'limit((sqrt(x+h)-sqrt(x))/h,h,0);ev(%,nouns);
```

$$\lim_{h \to 0} \frac{\sqrt{x+h} - \sqrt{x}}{h}$$

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

```
diff(sqrt(x),x);
```

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

Example 3 - $f(x) = \frac{1}{\sqrt{x}}$, $f'(x) = -\frac{1}{2x^{3/2}}$:

```
'limit((1/sqrt(x+h)-1/sqrt(x))/h,h,0);ev(%,nouns);
```

$$\lim_{h \to 0} \frac{\frac{1}{\sqrt{x+h}} - \frac{1}{\sqrt{x}}}{h}$$

$$-\frac{1}{2\,x^{3/2}}$$

```
diff(1/sqrt(x),x);
```

$$-\frac{1}{2\,x^{3/2}}$$

Implicit differentiation

Function *diff* can be used to produce implicit differentiation in an equation as illustrated in the following example. First, define and equation:

```
Eq : x*u(x)+u(x)^2 = sin(x);
```

$$u(x)^2 + x\,u(x) = \sin(x)$$

Next, apply *diff* to the entire equation to give you the implicit derivatives:

```
EqD : diff(Eq,x);
```

$$2\,u(x)\left(\frac{d}{d\,x}u(x)\right) + x\left(\frac{d}{d\,x}u(x)\right) + u(x) = \cos(x)$$

Finally, solve for the derivative:

```
Sol : solve(EqD,diff(u(x),x));
```

$$\left[\frac{d}{d\,x}u(x) = \frac{\cos(x) - u(x)}{2\,u(x) + x}\right]$$

Finding maxima, minima, and points of inflection

Maxima and minima (critical points) of a function $y = f(x)$ can be found by making $dy/dx = 0$. The points thus found are maxima if $d^2y/dx^2 > 0$, or minima if $d^2y/dx^2 > 0$. Points of inflection are found where $d^2y/dx^2 = 0$.

Consider the following example:

```
f(x) := x^4-5*x^2+6*x+3;
```

$$f(x) := x^4 - 5x^2 + 6x + 3$$

The equation corresponding to $df/dx = 0$ is *EqMM*:

```
EqMM : diff(f(x),x)=0;
```

$$4x^3 - 10x + 6 = 0$$

whose solutions are:

```
SolMM : allroots(EqMM);
```

$$[x = 0.8228756555323, x = 1.0, x = -1.822875655532295]$$

We extract these solutions into variables *x1*, *x2*, and *x3*:

```
x1 : rhs(SolMM[1]) $ x2 : rhs(SolMM[2])  $  x3 : rhs(SolMM[3])  $

[x1,x2,x3];
```

$$[0.8228756555323, 1.0, -1.822875655532295]$$

Then, evaluate the second derivative, *f2*, for each of the points found above.

```
f2 : diff(f(x),x,2);
```

$$12x^2 - 10$$

```
[subst(x=x1,f2),subst(x=x2,f2),subst(x=x3,f2)];
```

$$[-1.874507866387534, 1.999999999999993, 29.87450786638754]$$

The critical points (maxima and minima) are found at the following coordinates, *xL* = list of *x* values, *yL* = list of *y* values:

```
xL;
```

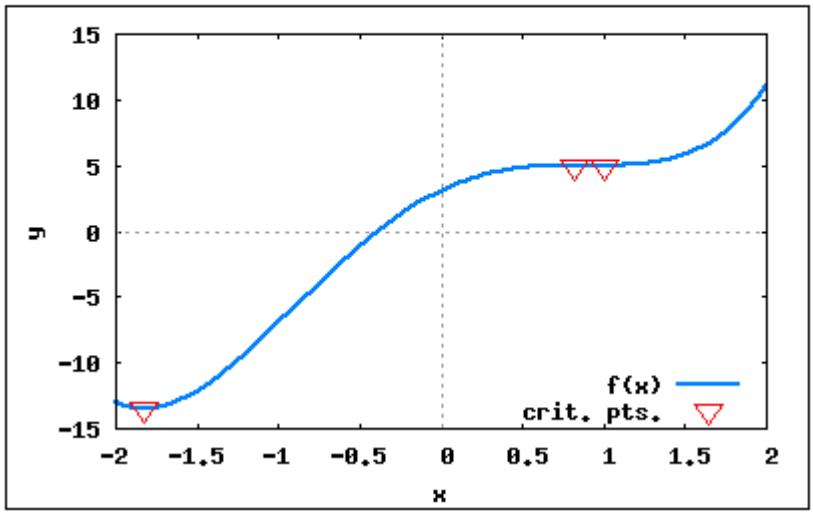
$$[0.8228756555323, 1.0, -1.822875655532295]$$

```
yL;
```

$$[5.010129588726067, 5.0, -13.51012958872607]$$

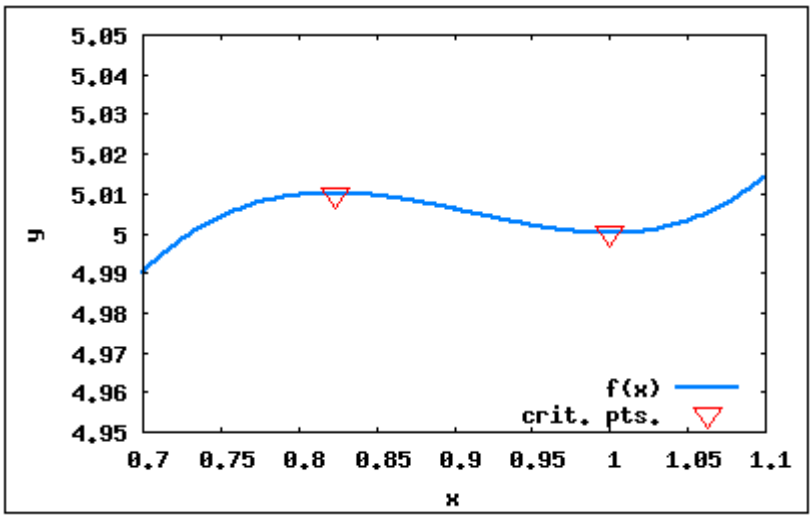
This results suggest that (xL_1,yL_1) = (0.823, 5.010) is a maximum, while the other two points are minima. Let's check these results using a plot of the function.

```
wxplot2d([f(x),[discrete,xL,yL]],[x,-2,2],[style,[lines,2,1],
[points,3,2,11]],[xlabel,"x"],[ylabel,"y"],
[legend,"f(x)","crit. pts."],[gnuplot_preamble,"set key bottom"]);
```



From the resulting figure it is clear that point (xL_3,yL_3) = (*-1.822*,*-13.51*) is indeed a relative minimum, but it's hard to see the relative position of the other two points. Focusing in the area for these two points we produce the following graph:

```
wxplot2d([f(x),[discrete,xL,yL]],[x,0.7,1.1],[y,4.95,5.05],
[style,[lines,2,1],[points,3,2,11]],[xlabel,"x"],[ylabel,"y"],
[legend,"f(x)","crit.pts."],[gnuplot_preamble,"set key bottom"]);
```



From this second graph it is obvious that point (xL_1,yL_1) = (*0.823, 5.010*) is indeed a relative maximum, while point *(xL2,yL2)* = (*1.000,5.000*) is a relative minimum.

To determine the points of inflection we can use:

```
isol : solve(f2=0,x);
```

$$[\, x = -\frac{\sqrt{5}}{\sqrt{6}}, x = \frac{\sqrt{5}}{\sqrt{6}} \,]$$

```
float(%);
```

[x = - 0.91287092917528 , x = 0.91287092917528]

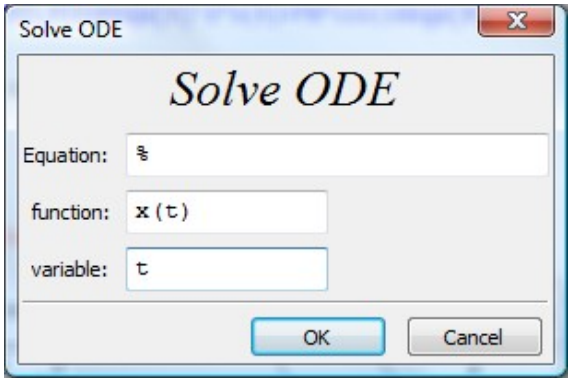
The location of these points is discernible in the previous two graphs.

<u>Differential equations</u>
The apostrophe (') before function *diff* can be used to write differential equations, e.g.,

```
DEq : diff(x(t),t,2)+beta*diff(x(t),t)+omega[n]^2*x(t)=F0*cos(omega[0]*t);
```

$$\frac{d^2}{dt^2}x(t)+\beta\left(\frac{d}{dt}x(t)\right)+\omega_n^2\,x(t)=\cos(\omega_0\,t)\,F0$$

This result could be used, for example, to solve the ordinary differential equation (ODE) using the menu item *Equations* > *Solve ODE*... with the values:



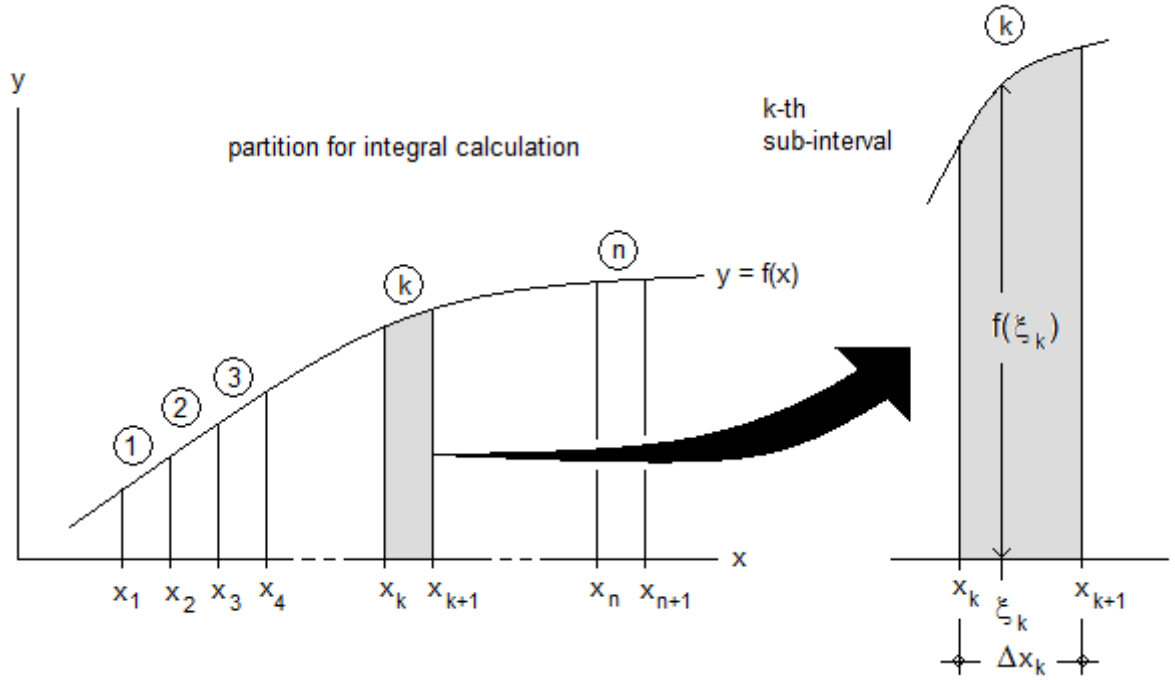
This results in the following *Maxima* input and output:

```
ode2(%, x(t), t);
Is  (2 ω_n - β)(2 ω_n + β)  positive, negative, or zero? positive;
```

$$(\%o305)\quad x(t)=\frac{\left(\omega_0\,\beta\sin(\omega_0\,t)+\left(\omega_n^2-\omega_0^2\right)\cos(\omega_0\,t)\right)F0}{\omega_n^4-2\,\omega_0^2\,\omega_n^2+\omega_0^2\,\beta^2+\omega_0^4}+\%e^{-\frac{\beta\,t}{2}}\left(\%k1\sin\left(\frac{\sqrt{4\,\omega_n^2-\beta^2}\,t}{2}\right)+\%k2\cos\left(\frac{\sqrt{4\,\omega_n^2-\beta^2}\,t}{2}\right)\right)$$

Summations as approximation to integrals

The formal definition of an integral, i.e., a Riemann integral is illustrated in the figure shown below. (For details, see: http://en.wikipedia.org/wiki/Riemann_integral).



The figure shows a partition of the interval $a \le x \le b$, where $a = x_1$, and $b = x_{n+1}$. A partition is the set of values $[a = x_1, x_2, \ldots, x_k, \ldots, x_n, x_{n+1} = b]$, so that $[x_1,x_2]$ limit the 1st sub-interval, $[x_2,x_3]$, the second sub-interval, and so on. For the *k-th* sub-interval, $[x_k,x_{k+1}]$, we identify a value ξ_k , such that, $x_k \le \xi_k \le x_{k+1}$. Identifying similar values for each of the *n* sub-intervals in the partition, the integral of the function $y = f(x)$ in the interval $a \le x \le b$ is defined as:

$$I = \int_a^b f(x)dx = \lim_{n\to\infty} \sum_{k=1}^{n} f(\xi_k)\Delta x_k \quad .$$

While the sub-intervals in the partition need not be of the same size, to make the calculation of an integral systematic, we make the partition be equally-spaced, so that,

$$\Delta x_1 = \Delta x_2 = \ldots = \Delta x_k = \ldots = \Delta x_n = \Delta x$$

and

$$x_1 = a\,,\; x_2 = a + \Delta x\,,\; x_3 = a + 2\Delta x\,,\; \ldots\,,\; x_k = a + (k-1)\Delta x\,,\; \ldots\,,\; b = x_{n+1} = a + n\Delta x \quad .$$

Given *n*, from the value of x_{n+1}, above, it follows that the constant width of sub-intervals can be calculated as

$$\Delta x = \frac{b-a}{n} \quad .$$

The selection of the value ξ_k in [x_k , x_{k+1}] is arbitrary, however, to make it systematic we can select one of the following options:

1. The left limit of the sub-interval, i.e., $\xi_k = x_k = \quad a+(k-1)\Delta x$.

2. The mid-point of the sub-interval, i.e., $\xi_k = \frac{x_k + x_{k+1}}{2} = a+(k-\frac{1}{2})\Delta x$.

3. The right limit of the sub-interval, i.e., $\xi_k = x_{k+1} = \quad a+k\,\Delta x$.

Let's call the value of the integral calculated by these three selections I_L, I_C, and I_R, respectively, thus, we have:

1. A “left” integral calculated as

$$I_L = \lim_{n\to\infty} \sum_{k=1}^{n} f\left(a+(k-1)\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) = \lim_{n\to\infty} \sum_{k=1}^{n} sL_k = \lim_{n\to\infty} SL \quad \text{, with}$$

$$sL_k = f\left(a+(k-1)\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) \quad \text{, and} \quad SL = \sum_{k=1}^{n} sL_k \quad .$$

2. A “center” integral calculated as

$$I_C = \lim_{n\to\infty} \sum_{k=1}^{n} f\left(a+\left(k-\frac{1}{2}\right)\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) = \lim_{n\to\infty} \sum_{k=1}^{n} sC_k = \lim_{n\to\infty} SC \quad \text{, with}$$

$$sC_k = f\left(a+\left(k-\frac{1}{2}\right)\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) \quad \text{, and} \quad SC = \sum_{k=1}^{n} sC_k \quad .$$

3. A “right” integral calculated as

$$I_R = \lim_{n\to\infty} \sum_{k=1}^{n} f\left(a+k\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) = \lim_{n\to\infty} \sum_{k=1}^{n} sR_k = \lim_{n\to\infty} SR \quad \text{, with}$$

$$sR_k = f\left(a+(k-1)\cdot\left(\frac{b-a}{n}\right)\right)\cdot\left(\frac{b-a}{n}\right) \quad \text{, and} \quad SR = \sum_{k=1}^{n} sR_k \quad .$$

These three combinations of summations and limits can be calculated using *Maxima* as illustrated in the example below.

Example - Consider the case $f(x) = x^2$, $a = 1$, $b = 4$. We attempt the calculation of I_L, as follows. First, define $f(x)$ and the summation term sL_k, and expand this term:

```
f(x):=x^2 $

sLk : f(a+(k-1)*(b-a)/n)*(b-a)/n;
```

$$\frac{(b-a)\left(\frac{(b-a)(k-1)}{n}+a\right)^2}{n}$$

```
sLk : expand(sLk);
```

$$\frac{a^2 b}{n}-\frac{a^3}{n}+\frac{2ab^2k}{n^2}-\frac{4a^2bk}{n^2}+\frac{2a^3k}{n^2}-\frac{2ab^2}{n^2}+\frac{4a^2b}{n^2}-\frac{2a^3}{n^2}+\frac{b^3k^2}{n^3}-\frac{3ab^2k^2}{n^3}+$$

$$\frac{3a^2bk^2}{n^3}-\frac{a^3k^2}{n^3}-\frac{2b^3k}{n^3}+\frac{6ab^2k}{n^3}-\frac{6a^2bk}{n^3}+\frac{2a^3k}{n^3}+\frac{b^3}{n^3}-\frac{3ab^2}{n^3}+\frac{3a^2b}{n^3}-\frac{a^3}{n^3}$$

Then, form the summation SL:

```
SL : sum(sLk,k,1,n),simpsum;
```

$$\frac{2b^3n^3+3b^3n^2+b^3n}{6n^3}-\frac{2ab^2n^3+3ab^2n^2+ab^2n}{2n^3}+$$

$$\frac{2a^2bn^3+3a^2bn^2+a^2bn}{2n^3}-\frac{2a^3n^3+3a^3n^2+a^3n}{6n^3}-\frac{b^3n^2+b^3n}{n^3}+\frac{3ab^2n^2+3ab^2n}{n^3}+$$

$$\frac{ab^2n^2+ab^2n}{n^2}-\frac{3a^2bn^2+3a^2bn}{n^3}-\frac{2a^2bn^2+2a^2bn}{n^2}+\frac{a^3n^2+a^3n}{n^2}+\frac{a^3n^2+a^3n}{n^3}-$$

$$\frac{2ab^2}{n}+\frac{4a^2b}{n}-\frac{2a^3}{n}+\frac{b^3}{n^2}-\frac{3ab^2}{n^2}+\frac{3a^2b}{n^2}-\frac{a^3}{n^2}+a^2b-a^3$$

Finally, calculate the limit:

```
IL : limit(SL,n,inf);
```

$$\frac{b^3-a^3}{3}$$

Compare with the integral value calculated using function *integrate*:

```
'integrate(x^2,x,a,b);ev(%,nouns);
```

$$\int_a^b x^2 \, dx$$

$$\frac{b^3}{3} - \frac{a^3}{3}$$

Exercise for the reader: follow a similar approach to the one used above to calculate i_l (*IL*) in order to calculate I_C, and I_R.

Derivatives and integrals

Derivatives and integrals can be combined in the same expression, e.g., the derivative of an integral:

```
'diff('integrate(exp(-x^2),x,0,t),t);ev(%,nouns);
```

$$\frac{d}{dt}\int_0^t \%e^{-x^2} \, dx$$

```
Is  t  positive, negative, or zero?

positive;
```

$$\%e^{-t^2}$$

The integral of a derivative:

```
'integrate('diff(exp(-x)*sin(x),x,2),x,0,t);ev(%,nouns);
```

$$\int_0^t \frac{d^2}{dx^2}(\%e^{-x}\sin(x)) \, dx$$

```
Is  t  positive, negative, or zero?

positive;
```

$$-2\left(\frac{\%e^{-t}(\sin(t)-\cos(t))}{2}+\frac{1}{2}\right)$$

Tables of derivatives and integrals

Functions *diff* and *integrate* can be used as tables of derivatives and integrals, respectively. For example, to remember the formulas for the derivative of a product or a quotient use:

```
'diff(u(x)*v(x),x);ev(%,nouns);
```

$$\frac{d}{dx}(u(x)v(x))$$

$$u(x)\left(\frac{d}{dx}v(x)\right)+v(x)\left(\frac{d}{dx}u(x)\right)$$

```
'diff(u(x)/v(x),x);ev(%,nouns);
```

$$\frac{d\ u(x)}{dx\ v(x)}$$

$$\frac{\frac{d}{dx}u(x)}{v(x)}-\frac{u(x)\left(\frac{d}{dx}v(x)\right)}{v(x)^2}$$

The "chain rule" for derivatives can be illustrated by the following examples:

```
'diff(sin(log(x)),x);ev(%,nouns);
```

$$\frac{d}{dx}\sin(\log(x))$$

$$\frac{\cos(\log(x))}{x}$$

```
'diff(sin(exp(-x^2)),x);ev(%,nouns);
```

$$\frac{d}{dx}\sin\left(\%e^{-x^2}\right)$$

$$-2x\,\%e^{-x^2}\cos\left(\%e^{-x^2}\right)$$

Some examples of integration formulas are presented next:

```
'integrate(x^n*log(x),x);ev(%,nouns);
```

$$\int x^n\log(x)\,dx$$

```
Is  n+1  zero or nonzero?  nonzero;
```

$$1\frac{x^{n+1}}{n+1}\log(x)-\frac{1\,x^{n+1}}{(n+1)^2}$$

```
'integrate(1/sqrt(1+x+x^2),x);ev(%,nouns);
```

$$\int\frac{1}{\sqrt{x^2+x+1}}\,dx$$

$$\operatorname{asinh}\left(\frac{2x+1}{\sqrt{3}}\right)$$

Multiple integrals

The following examples show cases of double and triple integrals.

Double integrals - Use two nested *integrate* commands to produce a double integral, e.g.,

```
'integrate('integrate(x^2+y^2,y,0,x),x,0,1);ev(%,nouns);
```

$$\int_0^1 \int_0^x y^2 + x^2 \, dy \, dx$$

$$\frac{1}{3}$$

Alternatively, you can enter the following command to skip showing the double integral:

```
integrate(integrate(x^2+y^2,y,0,x),x,0,1);
```

$$\frac{1}{3}$$

The following are two more examples of double integrals:

```
'integrate('integrate(exp(x+y),y,0,2),x,0,2);ev(%,nouns);
```

$$\int_0^2 \int_0^2 \%e^{y+x} \, dy \, dx$$

$$\%e^4 - 2\,\%e^2 + 1$$

```
'integrate('integrate(x*sin(y),y,0,2),x,0,2);ev(%,nouns);
```

$$\int_0^2 x \, dx \int_0^2 \sin(y) \, dy$$

$$2(1 - \cos(2))$$

Double integrals may have infinite limits, e.g.,

```
'integrate('integrate(y*exp(-x^2/2),y,0,2),x,minf,inf);ev(%,nouns);
```

$$\int_{-\infty}^{\infty} \%e^{-\frac{x^2}{2}}\,dx \int_{0}^{2} y\,dy$$

$$2^{3/2}\sqrt{\pi}$$

Triple integrals – An example of a triple integral is shown below:

```
'integrate('integrate('integrate(x^2+y^2+z^2,z,-2,2),y,-1,1),x,-1/2,1/2);ev(%,nouns);
```

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} \int_{-1}^{1} \int_{-2}^{2} z^2+y^2+x^2\,dz\,dy\,dx$$

$$14$$

Infinite series prove De Moivre's equation

De Moivre's equation states that $e^{i\theta}=cos(theta)+i\,sin(theta)$. We can check that this statement is true (to order 20) by using Taylor series expansions of the three functions involved:

- E1 = expansion of $e^{i\theta}$:

```
E1 : taylor(exp(%i*theta),theta,0,20);
```

$$1+\%i\,\theta-\frac{\theta^2}{2}-\frac{\%i\,\theta^3}{6}+\frac{\theta^4}{24}+\frac{\%i\,\theta^5}{120}-\frac{\theta^6}{720}-\frac{\%i\,\theta^7}{5040}+\frac{\theta^8}{40320}+\frac{\%i\,\theta^9}{362880}-\frac{\theta^{10}}{3628800}-$$

$$\frac{\%i\,\theta^{11}}{39916800}+\frac{\theta^{12}}{479001600}+\frac{\%i\,\theta^{13}}{6227020800}-\frac{\theta^{14}}{87178291200}-\frac{\%i\,\theta^{15}}{1307674368000}+\frac{\theta^{16}}{20922789888000}+$$

$$\frac{\%i\,\theta^{17}}{355687428096000}-\frac{\theta^{18}}{6402373705728000}-\frac{\%i\,\theta^{19}}{121645100408832000}+\frac{\theta^{20}}{2432902008176640000}+\ldots$$

- Real part of E1:

```
E1R : realpart(E1);
```

$$\frac{\theta^{20}}{2432902008176640000}-\frac{\theta^{18}}{6402373705728000}+\frac{\theta^{16}}{20922789888000}-\frac{\theta^{14}}{87178291200}+\frac{\theta^{12}}{479001600}-\frac{\theta^{10}}{3628800}+\frac{\theta^{8}}{40320}-\frac{\theta^{6}}{720}+\frac{\theta^{4}}{24}-\frac{\theta^{2}}{2}+1$$

- Compare with the expansion of *cos(θ)*:

```
taylor(cos(theta),theta,0,20);
```

$$1-\frac{\theta^{2}}{2}+\frac{\theta^{4}}{24}-\frac{\theta^{6}}{720}+\frac{\theta^{8}}{40320}-\frac{\theta^{10}}{3628800}+\frac{\theta^{12}}{479001600}-\frac{\theta^{14}}{87178291200}+\frac{\theta^{16}}{20922789888000}-\frac{\theta^{18}}{6402373705728000}+\frac{\theta^{20}}{2432902008176640000}+\ldots$$

- Imaginary part of E1:

```
E1I : imagpart(E1);
```

$$-\frac{\theta^{19}}{121645100408832000}+\frac{\theta^{17}}{355687428096000}-\frac{\theta^{15}}{1307674368000}+\frac{\theta^{13}}{6227020800}-\frac{\theta^{11}}{39916800}+\frac{\theta^{9}}{362880}-\frac{\theta^{7}}{5040}+\frac{\theta^{5}}{120}-\frac{\theta^{3}}{6}+\theta$$

- Compare with the expansion of *sin(θ)*:

```
taylor(sin(theta),theta,0,20);
```

$$\theta-\frac{\theta^{3}}{6}+\frac{\theta^{5}}{120}-\frac{\theta^{7}}{5040}+\frac{\theta^{9}}{362880}-\frac{\theta^{11}}{39916800}+\frac{\theta^{13}}{6227020800}-\frac{\theta^{15}}{1307674368000}+\frac{\theta^{17}}{355687428096000}-\frac{\theta^{19}}{121645100408832000}+\ldots$$

Some items of interest related to multivariate calculus
Some items of interest in multivariate calculus include: plots of bivariate functions, multiple integrals, and partial derivatives.

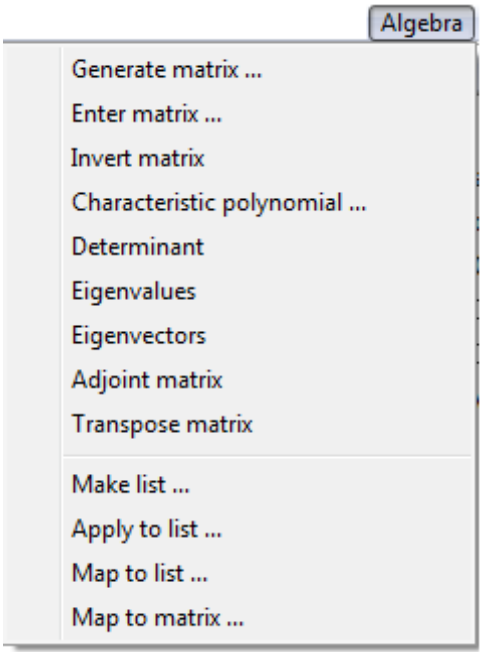
- The issue of plotting multivariate functions is addressed by function *plot3d* (see Chapter 4).
- The issue of multiple integrals was addressed earlier in this Chapter when discussing the *integrate* command.
- Regarding partial derivatives we should point out that function *diff* provides for the calculation of both ordinary and partial derivatives. Thus, the partial derivative $\frac{\partial}{\partial x}\left(x^2+z\,y\,sin(x)\right)$ is calculated in *Maxima* using:

7 Basic matrix and linear algebra functions in *Maxima*

In this chapter we present examples of matrix and linear algebra functions included in the *Algebra* menu in the *wxMaxima* interface.

Functions in the *Algebra* menu

The items in the *Algebra* menu, shown in the figure to the right, are presented in the following sections.



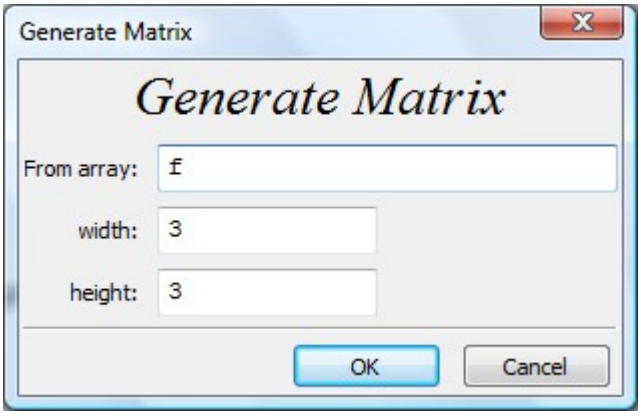
Generate matrix ...

The *Algebra* > *Generate matrix* ... utilizes a function of the matrix sub-indices *i* and *j*, defined previous to invoking the menu item, e.g.,

```
f[i,j]:=1/(i+j);
```

$$f_{i,j} := \frac{1}{i+j}$$

Then, use *f* in the dialogue form that results from the *Algebra* > *Generate matrix* ... menu item, e.g.,



This will result in the command:

```
genmatrix(f, 3, 3);
```

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \end{bmatrix}$$

Other examples of *genmatrix* are shown next:

- Defining specific values of the matrix:

```
genmatrix(a,3,3);
```

$$\begin{bmatrix} -1 & a_{1,2} & a_{1,3} \\ a_{2,1} & 3 & a_{2,3} \\ a_{3,1} & a_{3,2} & 5 \end{bmatrix}$$

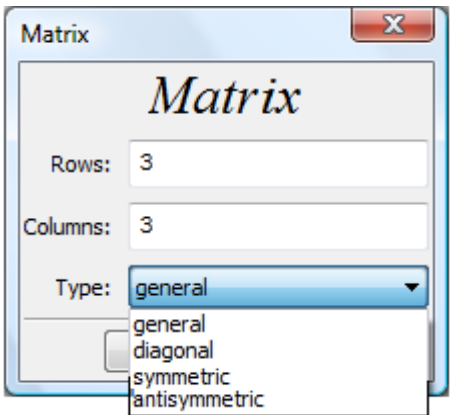
- Using a generic name with sub-indices:

```
genmatrix(c,3,3);
```

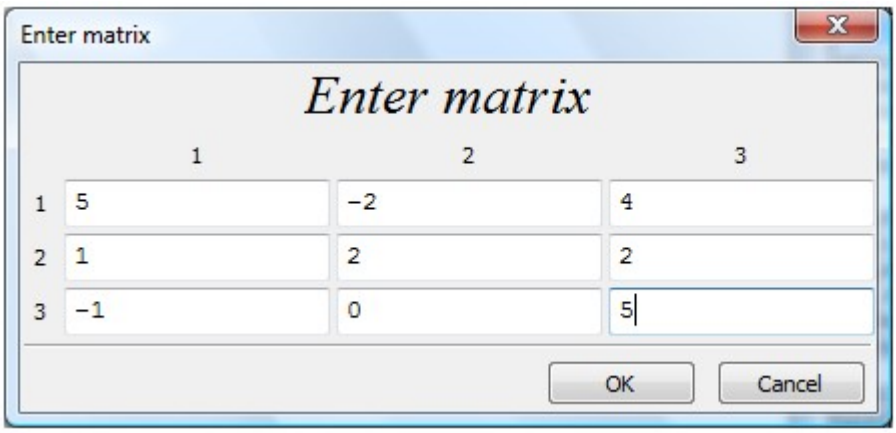
$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$$

Enter matrix ...

The *Algebra* > *Enter matrix* ... menu item is used to enter a matrix of given dimensions. The resulting dialogue form provides the following options:



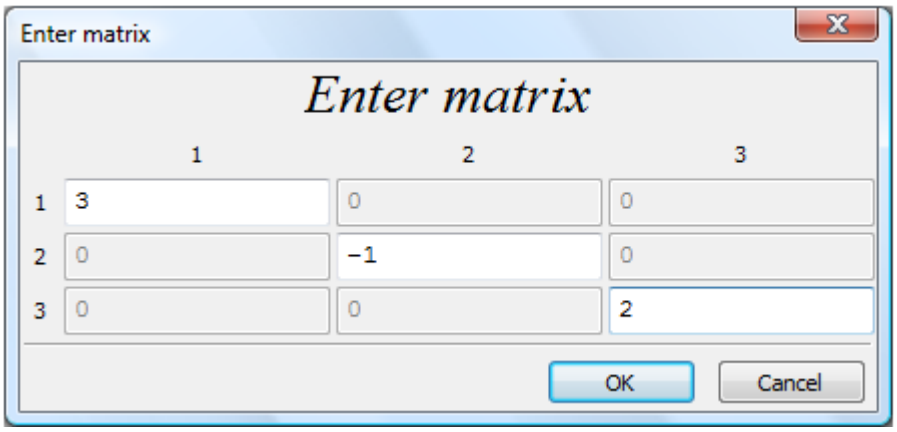
- *general matrix*:




```
matrix(
 [5,-2,4],
 [1,2,2],
 [-1,0,5]
);
```

$$\begin{bmatrix} 5 & -2 & 4 \\ 1 & 2 & 2 \\ -1 & 0 & 5 \end{bmatrix}$$

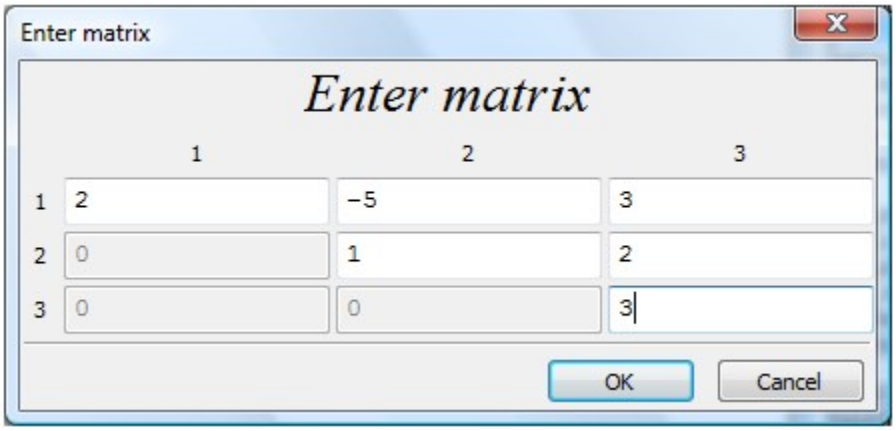
- diagonal matrix:




```
matrix(
 [3,0,0],
 [0,-1,0],
 [0,0,2]
);
```

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

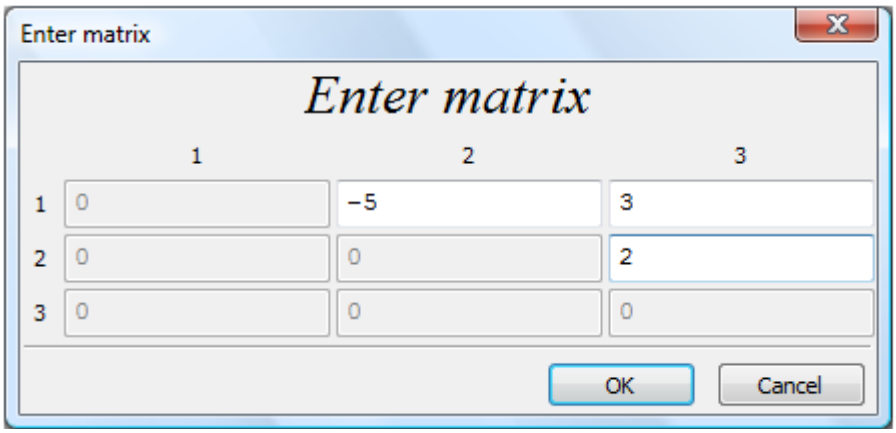
- symmetric matrix:




```
matrix(
 [2,-5,3],
 [-5,1,2],
 [3,2,3]
);
```

$$\begin{bmatrix} 2 & -5 & 3 \\ -5 & 1 & 2 \\ 3 & 2 & 3 \end{bmatrix}$$

- antisymmetric matrix




```
matrix(
 [0,-5,3],
 [-(-5),0,2],
 [-(3),-(2),0]
);
```

$$\begin{bmatrix} 0 & -5 & 3 \\ 5 & 0 & 2 \\ -3 & -2 & 0 \end{bmatrix}$$

Invert matrix

When invoked from the *Algebra* menu, the menu item *Invert matrix* produces the inverse of the matrix referred to with *%*, or of a matrix referred to by name or listen in the *INPUT* line, e.g.,

```
matrix([1,2,-3],[3,2,1],[5,5,3]);
```

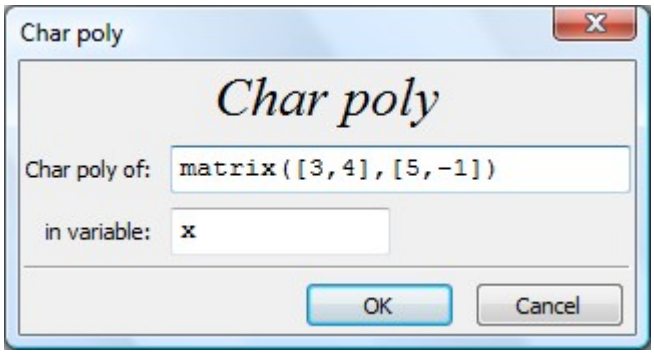
$$\begin{bmatrix} 1 & 2 & -3 \\ 3 & 2 & 1 \\ 5 & 5 & 3 \end{bmatrix}$$

```
invert(%);
```

$$\begin{bmatrix} -\frac{1}{22} & \frac{21}{22} & -\frac{4}{11} \\ \frac{2}{11} & -\frac{9}{11} & \frac{5}{11} \\ -\frac{5}{22} & -\frac{5}{22} & \frac{2}{11} \end{bmatrix}$$

Characteristic polynomial

The characteristic polynomial of a square matrix **A** results from expanding the determinant of the matrix **A**-x**I** where **I** is the identity matrix with the same dimensions of **A**, i.e., *charpoly*(**A**) = det(**A**-x**I**). Consider the following examples:




```
charpoly(matrix([3,4],[5,-1]), x), expand;
```

$$x^2 - 2x - 23$$

```
charpoly(matrix([3,2,-1],[1,2,5],[3,-2,1]),x), expand;
```

$$-x^3 + 6x^2 - 22x + 72$$

Determinant

The *Algebra* > *Determinant* menu item calculates the determinant of a matrix. By default, *determinant* uses the most recent result, as illustrated in the following example:

```
A : matrix([3,-1,2],[-1,4,2],[2,2,1]);
```

$$\begin{bmatrix} 3 & -1 & 2 \\ -1 & 4 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

```
determinant(%);
```

$$-25$$

Eigenvalues

The *Algebra* > *Eigenvalues* menu item calculates the eigenvalues of a matrix, i.e., it finds the roots of the characteristic polynomial of the matrix. Here is an example of this function applied to matrix **A** defined above:

```
eigenvalues(A);
```

$$\left[\left[-\frac{\sqrt{29}-3}{2}, \frac{\sqrt{29}+3}{2}, 5\right], [1, 1, 1]\right]$$

Notice that the output of function *eigenvalues* consists of two lists. The first list is the list of eigenvalues, and the second list is the multiplicity of those values. In this example, there are 3 eigenvalues and none repeats.

You can extract the eigenvalues as indicated in the following example, by assigning the output to a variable:

```
x : eigenvalues(A);
```

$$\left[\left[-\frac{\sqrt{29}-3}{2}, \frac{\sqrt{29}+3}{2}, 5\right], [1, 1, 1]\right]$$

The individual values are extracted as follows:

`x1 : x[1][1];`	`x2 : x[1][2];`	`x3 : x[1][3];`
$-\frac{\sqrt{29}-3}{2}$	$\frac{\sqrt{29}+3}{2}$	5

Eigenvectors

The *Algebra* > *Eigenvectors* menu item calculates the eigenvectors of a matrix, i.e., it solves for the vectors **v** from the eigenvalue equation **Av** =x**v**. For example, for the matrix **A** defined above:

```
eigenvectors(A);
```

$$\left[\left[\left[-\frac{\sqrt{29}-3}{2},\frac{\sqrt{29}+3}{2},5\right],[1,1,1]\right],\left[1,\frac{\sqrt{29}+3}{10},-\frac{\sqrt{29}+3}{5}\right],\left[1,-\frac{\sqrt{29}-3}{10},\frac{\sqrt{29}-3}{5}\right],\left[0,1,\frac{1}{2}\right]\right]$$

The output of this command includes the eigenvalues as the first element consisting of two lists as described above, i.e., eigenvalues and multiplicity. The remaining lists are the eigenvectors of the matrix corresponding to the eigenvalues listed first.

If we assign this output to a variable we can then extract the individual eigenvectors as follows:

```
v : % $
```

```
v1 : v[2];
```

$$\left[1,\frac{\sqrt{29}+3}{10},-\frac{\sqrt{29}+3}{5}\right]$$

```
v2 : v[3];
```

$$\left[1,-\frac{\sqrt{29}-3}{10},\frac{\sqrt{29}-3}{5}\right]$$

```
v3 : v[4];
```

$$\left[0,1,\frac{1}{2}\right]$$

Adjoint matrix

The adjoint matrix produced by the *Algebra* > *Adjoint matrix* menu item corresponds to the definition of the *adjugate* matrix as given in http://en.wikipedia.org/wiki/Adjugate. In the following example we first put together a complex matrix **A** and then calculate the adjoint or adjugate matrix. First, we define a couple of 3x3 real matrices *AR* and *AI*:

```
AR : matrix([2,1,-1],[3,5,6],[9,2,8]);
```

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

```
AI : matrix([3,-1,2],[7,6,2],[1,2,3]);
```

$$\begin{bmatrix} 3 & -1 & 2 \\ 7 & 6 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

Then, we put together matrix *A* = *AI* + i·*AR*:

```
A : AR + AI*%i;
```

$$\begin{bmatrix} 3\,\%i+2 & 1-\%i & 2\,\%i-1 \\ 7\,\%i+3 & 6\,\%i+5 & 2\,\%i+6 \\ \%i+9 & 2\,\%i+2 & 3\,\%i+8 \end{bmatrix}$$

Next, we invoke the *Algebra* > *Adjoint matrix* menu item to produce the adjugate (or adjoint) matrix:

```
adjoint(%);
```

$$\begin{bmatrix} (3\,\%i+8)(6\,\%i+5)-(2\,\%i+2)(2\,\%i+6) & (2\,\%i-1)(2\,\%i+2)-(1-\%i)(3\,\%i+8) & (1-\%i)(2\,\%i+6)-(2\,\%i-1)(6\,\%i+5) \\ (\%i+9)(2\,\%i+6)-(3\,\%i+8)(7\,\%i+3) & (3\,\%i+2)(3\,\%i+8)-(\%i+9)(2\,\%i-1) & (2\,\%i-1)(7\,\%i+3)-(2\,\%i+6)(3\,\%i+2) \\ (2\,\%i+2)(7\,\%i+3)-(\%i+9)(6\,\%i+5) & (1-\%i)(\%i+9)-(2\,\%i+2)(3\,\%i+2) & (3\,\%i+2)(6\,\%i+5)-(1-\%i)(7\,\%i+3) \end{bmatrix}$$

Use function *rectform* to simplify the matrix to:

```
rectform(%);
```

$$\begin{bmatrix} 47\,\%i+14 & 7\,\%i-17 & 25-8\,\%i \\ 49-41\,\%i & 13\,\%i+18 & -23\,\%i-23 \\ -39\,\%i-47 & 12-18\,\%i & 23\,\%i-18 \end{bmatrix}$$

Transpose matrix

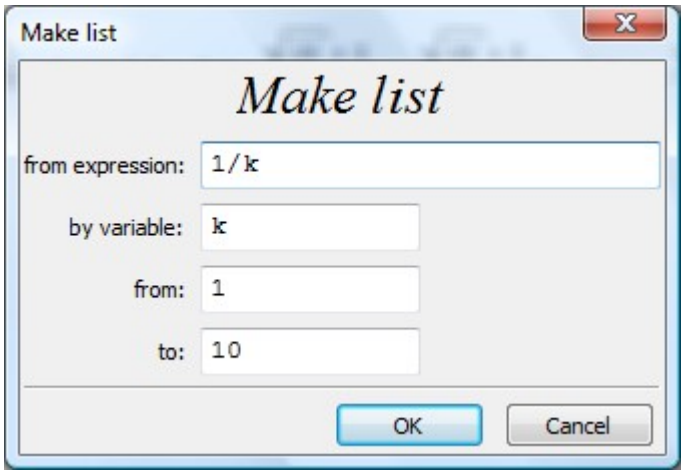
The *Algebra* > *Transpose matrix* menu item produces the transpose of a matrix. See the definition here: http://en.wikipedia.org/wiki/Transpose. For example, for the matrix *A* defined above, we have:

```
transpose(A);
```

$$\begin{bmatrix} 3\,\%i+2 & 7\,\%i+3 & \%i+9 \\ 1-\%i & 6\,\%i+5 & 2\,\%i+2 \\ 2\,\%i-1 & 2\,\%i+6 & 3\,\%i+8 \end{bmatrix}$$

Make list ...

The *Algebra* > *Make list* ... menu item produces a dialogue form that can be used to generate a list. The elements in the list are defined by an expression which is function of an index (say, *k*) for the range of integer values specified for that index. For example, the following dialogue:



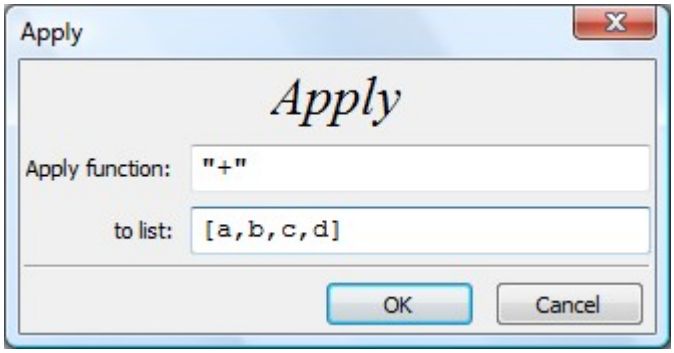
produces the list shown below:

```
makelist(1/k, k, 1, 10);
```

$$\left[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}\right]$$

Apply to list ...

The *Algebra* > *Apply to list* ... menu item produces a dialogue form that can be used to apply an operator (e.g., "+", a sum) to the elements of a list. The application of this menu item is illustrated by the following example:

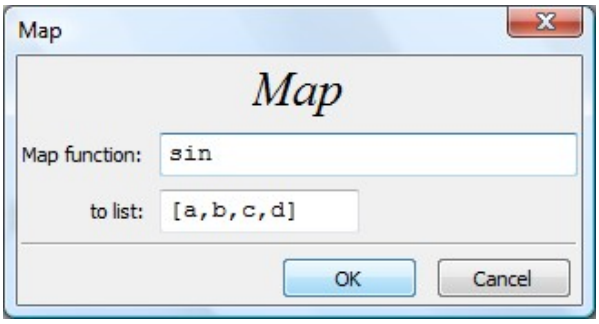



```
apply("+", [a,b,c,d]);
```

$$d + c + b + a$$

Map to list ...

The *Algebra* > *Map to list* ... menu item produces a dialogue form that can be used to "map" (or distribute) a function to the elements of a list. The application of this menu item is illustrated by the following example:




```
map(sin, [a,b,c,d]);
[ sin(a), sin(b), sin(c), sin(d)]
```

A second example of function *map* is show next:

```
map(log,[-2,-1,1,2]);
[ log(-2), log(-1), 0, log(2)]
```

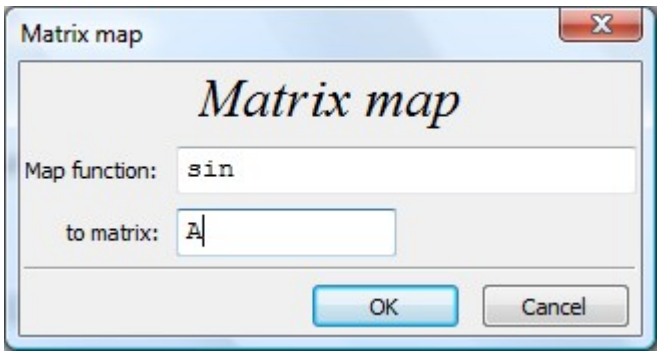
Map to matrix ...

The *Algebra* > *Map to matrix* ... menu item behaves similar to the *Map to list* ... menu item, "mapping" a function to all elements of a matrix. The application of this menu item is illustrated by the following example in which we first define a 3x3 matrix A:

```
A : matrix([3,-1,2],[-1,4,2],[2,2,1]);
```

$$\begin{bmatrix} 3 & -1 & 2 \\ -1 & 4 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

Then, we invoke the *Map to matrix* ... menu item:



The result is the following matrix:

```
matrixmap(sin, A);
```

$$\begin{bmatrix} \sin(3) & -\sin(1) & \sin(2) \\ -\sin(1) & \sin(4) & \sin(2) \\ \sin(2) & \sin(2) & \sin(1) \end{bmatrix}$$

To find floating-point elements in the matrix use function *float*:

```
float(%);
```

$$\begin{bmatrix} 0.14112000805987 & -0.8414709848079 & 0.90929742682568 \\ -0.8414709848079 & -0.75680249530793 & 0.90929742682568 \\ 0.90929742682568 & 0.90929742682568 & 0.8414709848079 \end{bmatrix}$$

Functions for creating matrices

In this section we present examples of *Maxima* functions to generate matrices. Some functions for creating matrices that are available in the *Algebra* menu were introduced above (*genmatrix*, *matrixmap*, *transpose*). The following examples demonstrate the use of additional functions:

copymatrix

Use *copymatrix* to copy a matrix into a variable name. For example, first create matrix *A*:

```
A : matrix([1,2,4],[3,4,2],[2,-1,1]);
```

$$\begin{bmatrix} 1 & 2 & 4 \\ 3 & 4 & 2 \\ 2 & -1 & 1 \end{bmatrix}$$

then, copy matrix *A* into *B* using *copymatrix*:

```
B : copymatrix(A);
```

$$\begin{bmatrix} 1 & 2 & 4 \\ 3 & 4 & 2 \\ 2 & -1 & 1 \end{bmatrix}$$

columnvector

A column vector is a matrix of *n* rows and 1 column. Function *columnvector* lets you build a column vector out of a list of values, e.g.,

```
v : columnvector([-1,0,4,5]);
```

$$\begin{bmatrix} -1 \\ 0 \\ 4 \\ 5 \end{bmatrix}$$

diag

Function *diag*, which needs to be loaded separately, allows you to build a diagonal matrix based on two or more matrices. For example, using matrices *A* and *B*, defined above, we can build the following diagonal matrix:

```
load("diag");
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/contrib/diag.mac
```

```
diag([A,B]);
```

$$\begin{bmatrix} 1 & 2 & 4 & 0 & 0 & 0 \\ 3 & 4 & 2 & 0 & 0 & 0 \\ 2 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 4 \\ 0 & 0 & 0 & 3 & 4 & 2 \\ 0 & 0 & 0 & 2 & -1 & 1 \end{bmatrix}$$

diagmatrix

Function *diagmatrix*(*n*,*a*) creates a diagonal matrix of dimensions $n \times n$ with all its diagonal elements equal to *a*:

```
diagmatrix(4,-5);
```

$$\begin{bmatrix} -5 & 0 & 0 & 0 \\ 0 & -5 & 0 & 0 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & -5 \end{bmatrix}$$

Function *diagmatrix* can be used to generate an identity matrix as illustrated below:

```
diagmatrix(3,1);
```

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

ematrix

Function *ematrix*(*n*,*m*,*a*,*i*,*j*) creates a matrix of dimensions *n*×*m* full of zero elements except for element [*i*,*j*] which is replaced by the value *a*, *e.g.*,

```
ematrix(3,3,-5,2,2);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

entermatrix

Function *entermatrix*(*n*,*m*) provides for an interactive, if long, way to enter a matrix, e.g.,

```
entermatrix(2,2);
Is the matrix  1. Diagonal  2. Symmetric  3. Antisymmetric  4. General
Answer 1, 2, 3 or 4 :  4;
Row 1 Column 1:  2;
Row 1 Column 2:  3;
Row 2 Column 1:  1;
Row 2 Column 2:  5;
Matrix entered.
```

$$\begin{bmatrix} 2 & 3 \\ 1 & 5 \end{bmatrix}$$

ident

Function *ident*(*n*) allows to create an *n*×*n* identity matrix:

```
I3 : ident(3);
```

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

matrix

Function *matrix*, which has been used before in this and other Chapters, allows the user to enter a matrix by defining the matrix rows as lists of the same length. The use of *matrix* is illustrated in the following example:

```
A : matrix([1,2,3],[3,2,1]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}$$

submatrix

Function *submatrix* allows the user to extract a submatrix out of a matrix. To illustrate the use of function *submatrix* consider the *4×4* matrix *A*:

```
A;
```

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

To extract a matrix by eliminating rows from i_1 to i_2 and columns from j_1 to j_2 out of matrix *A*, use the general call *submatrix*(i_1,i_2,*A*,j_1,j_2). In the following example we eliminate rows 2 and 3 and columns 2 to 3 out of matrix *A* and store the resulting matrix into *B*:

```
B : submatrix(2,3,A,2,3);
```

$$\begin{bmatrix} 3 & 4 \\ 3 & 8 \end{bmatrix}$$

To eliminate rows from i_1 to i_2, only, use the modified call *submatrix*(i_1,i_2,*A*), e.g.,

```
C : submatrix(2,3,A);
```

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

To eliminate columns from j_1 to j_2, only, use the modified call *submatrix*(*A*,j_1,j_2), e.g.,

```
D : submatrix(A,2,3);
```

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

zeromatrix

Function *zeromatrix*(*m*,*n*) creates a matrix of dimensions *m*×*n* such that all its elements are zero values, e.g.,

```
Z : zeromatrix(3,5);
```

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

Functions for manipulating matrices

The following functions allows the user to extract or add rows and columns out of matrices. To illustrate the use of these functions we will refer to matrix *A* defined above, and repeated here:

```
A;
```

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

col

Function *col* extracts a column out of a matrix, e.g.,

```
C2 : col(A,2);
```

$$\begin{bmatrix} 2 \\ 2 \\ 2 \\ -5 \end{bmatrix}$$

row

Function *row* extracts a row out of a matrix:

```
R3 : row(A,3);
```

$$\begin{bmatrix} 1 & 2 & 1 & 4 \end{bmatrix}$$

addcol

Function *addcol* is used to append one or more columns to a matrix, e.g.,

```
A5C : addcol(A,[5,4,3,2]);
```

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

```
A6C : addcol(A,[3,2,1,4],[4,3,2,1]);
```

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

addrow

Function *addrow* is used to append one or more rows to a matrix, e.g.,

```
A5R : addrow(A,[5,4,3,2]);
```

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

```
A6R : addrow(A,[3,2,1,4],[4,3,2,1]);
```

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

Matrix operations

Basic matrix operations include addition, subtraction, multiplication, division, and powers. To illustrate those operations we will use the following matrices *A* and *B*:

```
A : matrix([3,2,1],[1,-4,2],[4,5,-2]);
```

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & -4 & 2 \\ 4 & 5 & -2 \end{bmatrix}$$

```
B : matrix([3,-1,2],[5,-3,1],[1,4,2]);
```

$$\begin{bmatrix} 3 & -1 & 2 \\ 5 & -3 & 1 \\ 1 & 4 & 2 \end{bmatrix}$$

Addition and subtraction

Addition and subtration are term-by-term operations on matrices of the same dimensions. The examples below include linear combinations of additions and subtractions:

```
A + B;
```

$$\begin{bmatrix} 6 & 1 & 3 \\ 6 & -7 & 3 \\ 5 & 9 & 0 \end{bmatrix}$$

```
A - B;
```

$$\begin{bmatrix} 0 & 3 & -1 \\ -4 & -1 & 1 \\ 3 & 1 & -4 \end{bmatrix}$$

```
2*A + 5*B;
```

$$\begin{bmatrix} 21 & -1 & 12 \\ 27 & -23 & 9 \\ 13 & 30 & 6 \end{bmatrix}$$

```
A-2*B;
```

$$\begin{bmatrix} -3 & 4 & -3 \\ -9 & 2 & 0 \\ 2 & -3 & -6 \end{bmatrix}$$

Multiplication

Multiplication can be *term-by-term*, in which case we use an asterisk for the multiplication symbol, e.g.,

```
A*B;
```

$$\begin{bmatrix} 9 & -2 & 2 \\ 5 & 12 & 2 \\ 4 & 20 & -4 \end{bmatrix}$$

Traditional, *non-commutative, matrix multiplication* is achieved by using a dot (.) as the multiplication symbol:

```
A.B;
```

$$\begin{bmatrix} 20 & -5 & 10 \\ -15 & 19 & 2 \\ 35 & -27 & 9 \end{bmatrix}$$

```
B.A;
```

$$\begin{bmatrix} 16 & 20 & -3 \\ 16 & 27 & -3 \\ 15 & -4 & 5 \end{bmatrix}$$

Power

A matrix raised to a scalar exponential produces a term-by-term exponentiation, e.g.,

```
A^2;
```

$$\begin{bmatrix} 9 & 4 & 1 \\ 1 & 16 & 4 \\ 16 & 25 & 4 \end{bmatrix}$$

```
B^3;
```

$$\begin{bmatrix} 27 & -1 & 8 \\ 125 & -27 & 1 \\ 1 & 64 & 8 \end{bmatrix}$$

A scalar base raised to a matrix exponent is also a term-by-term operation, e.g.,

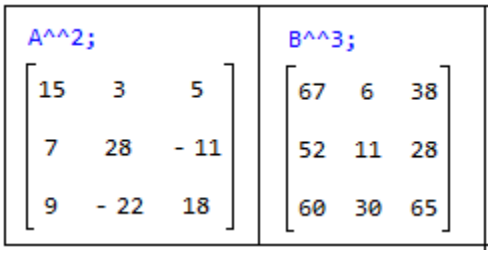
```
exp(A);
```

$$\begin{bmatrix} \%e^{3} & \%e^{2} & \%e \\ \%e & \%e^{-4} & \%e^{2} \\ \%e^{4} & \%e^{5} & \%e^{-2} \end{bmatrix}$$

```
5^B;
```

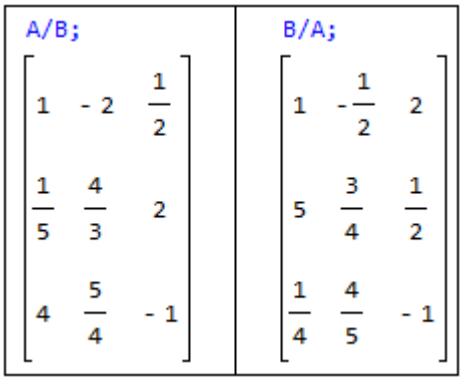
$$\begin{bmatrix} 125 & \frac{1}{5} & 25 \\ 3125 & \frac{1}{125} & 5 \\ 5 & 625 & 25 \end{bmatrix}$$

Matrix exponentiation uses a double caret (^^) and represents the result of repeated matrix multiplication, i.e., A^^2 = A.A, A^^3 = A^^2.A, and so on, e.g.,



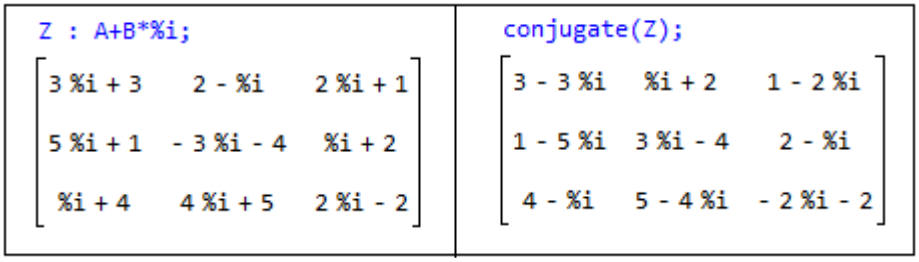
Division

Division of matrices is a term-by-term operation, e.g.,



Conjugate

The *conjugate* function, used to calculate the complex conjugate of a number, can be used to find the conjugate matrix of a matrix of complex numbers, e.g.,



Functions for linear algebra operations

The functions whose operation is illustrated in this section are used in linear algebra applications. Some functions, such as *adjoint*, *charpoly*, *determinant*, *eigen*, and *invert*, were introduced as part of the *Algebra* menu items.

coefmatrix

Function *coefmatrix*([*list of linear equations*],[*list of variables*]) can be used to extract the coefficients from a *list of linear equations* containing the variables in the *list of variables*, e.g.,

```
Eq1: 2*x+5*y+3*z = 25 $ Eq2: 3*x-y+5*z=105 $ Eq3: x+y+z=18 $

A : coefmatrix([Eq1,Eq2,Eq3],[x,y,z]);
```

$$\begin{bmatrix} 2 & 5 & 3 \\ 3 & -1 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

augcoefmatrix

Function *augcoefmatrix* ([*list of linear equations*],[*list of variables*]) produces an augmented matrix of coefficients similar to that produced by *coefmatrix*, except that the last column contains the negatives of the right-hand side elements corresponding to the *list of equations*. For example, for the system of linear equations used in the *coefmatrix* example shown above, the resulting augmented matrix of coefficients is calculated as follows:

```
AA : augcoefmatrix([Eq1,Eq2,Eq3],[x,y,z]);
```

$$\begin{bmatrix} 2 & 5 & 3 & -25 \\ 3 & -1 & 5 & -105 \\ 1 & 1 & 1 & -18 \end{bmatrix}$$

echelon and triangularize

Both functions *echelon*(*A*) and *triangularize*(*A*) produce upper triangular matrices representing row-reduced echelon forms of matrix *A*. The difference between these two functions is that function *echelon* produces a matrix such that its main diagonal elements are reduced to the number *1*. The following examples, using matrices *A* and *AA* created above, illustrate the application of these two functions highlighting their differences:

```
triangularize(A);
```

$$\begin{bmatrix} 2 & 5 & 3 \\ 0 & -17 & 1 \\ 0 & 0 & 10 \end{bmatrix}$$

```
triangularize(AA);
```

$$\begin{bmatrix} 2 & 5 & 3 & -25 \\ 0 & -17 & 1 & -135 \\ 0 & 0 & 10 & -109 \end{bmatrix}$$

```
echelon(A);
```

$$\begin{bmatrix} 1 & \frac{5}{2} & \frac{3}{2} \\ 0 & 1 & -\frac{1}{17} \\ 0 & 0 & 1 \end{bmatrix}$$

```
echelon(AA);
```

$$\begin{bmatrix} 1 & \frac{5}{2} & \frac{3}{2} & -\frac{25}{2} \\ 0 & 1 & -\frac{1}{17} & \frac{135}{17} \\ 0 & 0 & 1 & -\frac{109}{10} \end{bmatrix}$$

mattrace

Function *mattrace*, which is available by loading package "nchrpl", calculates the trace of a matrix (i.e., the sum of its main diagonal elements):

```
load("nchrpl");
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/matrix/nchrpl.mac

mattrace(A);
2
```

minor

The minor matrix (*i*,*j*) of a matrix *A* is the matrix that results from eliminating row *i* and column *j*. Minor matrices are used, for example, in the calculation of determinants.

```
A;
```

$$\begin{bmatrix} 2 & 5 & 3 \\ 3 & -1 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

```
minor(A,2,2);
```

$$\begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix}$$

ncharpoly (alternate to charpoly)

Function *ncharpoly*, loaded with package *nchrpl*, an alternative function to *charpoly*, used to obtain the characteristic polynomial of a matrix *A*, e.g.,

```
load("nchrpl");
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/matrix/nchrpl.mac

ncharpoly(A,%lambda);
λ^3 - 2 λ^2 - 24 λ - 10
```

permanent

Function *permanent* calculates the *permanent* of a matrix. To understand the definition of the permanent of a matrix, see http://en.wikipedia.org/wiki/Permanent:

```
permanent(A);
54
```

rank

Function *rank* calculates the *rank* of a matrix. To understand the definition of the rank of a matrix, see http://en.wikipedia.org/wiki/Rank_(linear_algebra) :

```
rank(A);
3
```

tracematrix

Function *tracematrix*, which needs to be loaded with package *functs*, calculates the trace of a matrix, e.g.,

```
load("functs");
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/simplification/functs.mac

tracematrix(A);
2
```

Functions in the *eigen* package

The functions described in this package are used in the calculation of eigenvalues and eigenvectors of matrices. Functions *eigenvalues* and *eigenvectors*, which belong to this package, were described in the context of the *Algebra* menu at the beginning of this chapter. When functions *eigenvalues* and *eigenvectors* are invoked, the *eigen* package is invoked automatically. To use the other functions make sure to load the package *eigen* beforehand:

```
load(eigen);
C:/PROGRA~1/MAXIMA~1.0/share/maxima/5.14.0/share/matrix/eigen.mac
```

innerproduct

Function *innerproduct* produces the inner product, or scalar product, of two lists of the same length that represent vectors. A three-element list, for example, may represent a three-dimensional physical vector such as velocity, acceleration, force, moment, or momentum. In such cases, the inner product is referred also as a *dot* product because the notation used is **u**• **v**, where **u** and **v** are physical vectors. A dot product is distinguished from a vector, or *cross*, product which is expressed as **u**× **v**. Some examples of function *innerproduct* are shown next:

```
innerproduct([x1,y1,z1],[x2,y2,z2]);
z1 z2 + y1 y2 + x1 x2

innerproduct([2,-3,1,3],[5,5,2,4]);
9
```

For physical vectors, the inner product of a vector with itself is the square of its magnitude, e.g., given vector **v**, as shown below, its magnitude is $|\mathbf{v}|$ = *magv*:

```
v : [4,2,4,5];
[ 4 , 2 , 4 , 5 ]

magv : sqrt(innerproduct(v,v));
```

$$\sqrt{61}$$

unitvector

Function *unitvector* produces the unit vector associated with a vector. For example, if **v** represents a physical vector, the corresponding unit vector is $\mathbf{e}_v = \mathbf{v}/|\mathbf{v}|$. For the vector **v** shown above, the unit vector can be calculated using:

```
unitvector(v);
```

$$\left[\frac{4}{\sqrt{61}}, \frac{2}{\sqrt{61}}, \frac{4}{\sqrt{61}}, \frac{5}{\sqrt{61}}\right]$$

which is the same than

```
v/magv;
```

$$\left[\frac{4}{\sqrt{61}}, \frac{2}{\sqrt{61}}, \frac{4}{\sqrt{61}}, \frac{5}{\sqrt{61}}\right]$$

uniteigenvectors

Consider the symmetric matrix **A**:

```
A : matrix([1,3,2],[3,-5,6],[2,6,4]);
```

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

whose eigenvalues and eigenvectors are:

```
x : eigenvectors(A);
```

$$[[[-\sqrt{70},\sqrt{70},0],[1,1,1]],[1,-\frac{\sqrt{70}+5}{3},2],[1,\frac{\sqrt{70}-5}{3},2],[1,0,-\frac{1}{2}]]$$

The three eigenvectors can be extracted by using:

`x1 : x[2];`	`x2 : x[3];`	`x3 : x[4];`
$[1,-\frac{\sqrt{70}+5}{3},2]$	$[1,\frac{\sqrt{70}-5}{3},2]$	$[1,0,-\frac{1}{2}]$

Function *uniteigenvectors* produces the unit eigenvectors of the matrix, e.g.,

```
ex : uniteigenvectors(A);
```

$$[[[-\sqrt{70},\sqrt{70},0],[1,1,1]],[\frac{3}{\sqrt{10\sqrt{70}+140}},-\frac{\sqrt{70}+5}{\sqrt{10\sqrt{70}+140}},\frac{6}{\sqrt{10\sqrt{70}+140}}],$$
$$[\frac{3}{\sqrt{140-10\sqrt{70}}},\frac{\sqrt{70}-5}{\sqrt{140-10\sqrt{70}}},\frac{6}{\sqrt{140-10\sqrt{70}}}],[\frac{2}{\sqrt{5}},0,-\frac{1}{\sqrt{5}}]]$$

The unit eigenvectors can be extracted using:

```
ex1 : ex[2];
```

$$[\frac{3}{\sqrt{10\sqrt{70}+140}},-\frac{\sqrt{70}+5}{\sqrt{10\sqrt{70}+140}},\frac{6}{\sqrt{10\sqrt{70}+140}}]$$

```
ex2 : ex[3];
```

$$[\frac{3}{\sqrt{140-10\sqrt{70}}},\frac{\sqrt{70}-5}{\sqrt{140-10\sqrt{70}}},\frac{6}{\sqrt{140-10\sqrt{70}}}]$$

```
ex3 : ex[4];
```

$$[\frac{2}{\sqrt{5}},0,-\frac{1}{\sqrt{5}}]$$

We can check, for example, that *ex1* is the unit vector corresponding to *x1* by using:

```
x1/sqrt(innerproduct(x1,x1));
```

$$[\frac{3}{\sqrt{10\sqrt{70}+140}},-\frac{\sqrt{70}+5}{\sqrt{10\sqrt{70}+140}},\frac{6}{\sqrt{10\sqrt{70}+140}}]$$

gramschmidt

Function gramschmidt performs a Gram-Schmidt orthogonalization for the rows of a matrix. This process is described in http://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process. In this example we define a matrix *A*, and apply function *gramschmidt* to that matrix:

```
A : matrix([1,2,3],[9,18,30],[12,48,60]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 9 & 18 & 30 \\ 12 & 48 & 60 \end{bmatrix}$$

```
y : gramschmidt(A);
```

$$[[1,2,3],[-\frac{3^2}{27},-\frac{3^2}{7},\frac{35}{27}],[-\frac{2^4 3}{5},\frac{2^3 3}{5},0]]$$

```
ev(%,nouns);
```

$$[[1,2,3],[-\frac{9}{14},-\frac{9}{7},\frac{15}{14}],[-\frac{48}{5},\frac{24}{5},0]]$$

similaritytransform or simtran

When applied to a real matrix, function *similaritytransform* or *simtran* produces the same output than function *uniteigenvectors*. A complete description of this function is available by using:

```
?? simtran;
-- Function: simtran (<M>)
```

(see details in the wxMaxima window)

Functions for matrix decomposition

Matrix decomposition is useful in linear algebra applications. *Maxima* provides the following functions for matrix decomposition:

cholesky

Function *cholesky* produces the Cholesky decomposition of a symmetric, positive-definite matrix (http://en.wikipedia.org/wiki/Cholesky_decomposition). An example is shown next, in which we first define a matrix *A*:

```
A : matrix([5,4,2],[4,10,6],[2,6,15]);
```

$$\begin{bmatrix} 5 & 4 & 2 \\ 4 & 10 & 6 \\ 2 & 6 & 15 \end{bmatrix}$$

The cholesky decomposition results in

```
cholesky(A);
```

$$\begin{bmatrix} \sqrt{5} & 0 & 0 \\ \frac{4}{\sqrt{5}} & \frac{\sqrt{34}}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & \frac{22\sqrt{5}}{5\sqrt{34}} & \frac{\sqrt{193}}{\sqrt{17}} \end{bmatrix}$$

eigens_by_jacobi

Function *eigens_by_jacobi* calculates the eigenvalues of a symmetric real matrix by the method of Jacobi rotations (http://mathworld.wolfram.com/JacobiRotationMatrix.html). Consider the following example in which we first define a symmetric matrix *A*:

```
A : matrix([10,5,-3],[5,4,2],[-3,2,5]);
```

$$\begin{bmatrix} 10 & 5 & -3 \\ 5 & 4 & 2 \\ -3 & 2 & 5 \end{bmatrix}$$

The eigenvalues and eigenvectors are calculated as follows:

```
eigens_by_jacobi(A);
The largest percent change was 6.812078595512458
The largest percent change was 0.029914382001285
The largest percent change was 3.6035958827271688E-8
The largest percent change was 0.0
number of sweeps: 4
number of rotations: 9
[ [ 13.19130631125922 , - 0.70944455924744 , 6.518138247988224 ] ,
```

$$\begin{bmatrix} 0.87664852672572 & -0.47729976466124 & -0.060599465706809 \\ 0.42986541460449 & 0.72042800127206 & 0.54424187665968 \\ -0.21610896769114 & -0.50315845380702 & 0.83673680715384 \end{bmatrix}]$$

lu_factor

Function *lu_factor* produces the *LU* decomposition of a matrix. To learn about *LU* decomposition see, for example, http://en.wikipedia.org/wiki/LU_decomposition . The example shown below uses matrix *A* defined above:

```
lu_factor(A);
```

$$\left[\begin{bmatrix} 10 & 5 & -3 \\ \frac{1}{2} & \frac{3}{2} & \frac{7}{2} \\ -\frac{3}{10} & \frac{7}{3} & -\frac{61}{15} \end{bmatrix}, [1, 2, 3], generalring\right]$$

Hilbert matrix and Vandermonde matrix

The Hilbert matrix and Vandermonde matrix are specialized matrices used in linear algebra. In this section we describe and present examples of functions that *Maxima* provides with the purpose of generating such matrices.

hilbert_matrix

Function *hilbert_matrix*(n) produces the Hilbert matrix of order $n \times n$. A Hilbert matrix has elements $h_{i,j}=\frac{1}{i+j-1}$. An example of a Hilbert matrix is shown next:

```
hilbert_matrix(3);
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

vandermonde_matrix

A Vandermonde matrix is an $n \times n$ matrix generated from a column vector (or a list) of length *n*. The resulting matrix is such that column *j* results from raising the elements of the originating column vector (or list) to the power (*j-1*). To calculate a Vandermonde matrix in *Maxima* use *vandermonde_matrix*(*list*), where *list* is a list of numbers, e.g.,

```
vandermonde_matrix([1,2,3,4]);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}$$

Constant matrices, random matrices, matrix size, and individual elements
This section addresses a few items not addressed in the previous sections.

Constant matrices
Suppose that you want to produce a matrix with all its elements being the same constant value. Let's refer to it as a *constant matrix*. The following, user-defined, function can be used to produce a matrix of *m* rows and *n* columns with all its elements equal to a value *c*:

```
cmatrix(m,n,c) := sum(sum(ematrix(m,n,c,i,j),j,1,n),i,1,m);
```

$$\text{cmatrix}(m, n, c) := \sum_{i=1}^{m} \sum_{j=1}^{n} \text{ematrix}(m, n, c, i, j)$$

Two examples of application of function *cmatrix* are shown next:

```
cmatrix(3,3,-2);
```

$$\begin{bmatrix} -2 & -2 & -2 \\ -2 & -2 & -2 \\ -2 & -2 & -2 \end{bmatrix}$$

```
cmatrix(4,4,k^2);
```

$$\begin{bmatrix} k^2 & k^2 & k^2 & k^2 \\ k^2 & k^2 & k^2 & k^2 \\ k^2 & k^2 & k^2 & k^2 \\ k^2 & k^2 & k^2 & k^2 \end{bmatrix}$$

Random numbers and random matrices
Function *cmatrix*, in combination with function *random*, can be used to generate a random matrix of integer numbers. Function *random(x)* produces a random number between *0* and *x*. For example, the following command produces a list of *10* random numbers between *0* and *10*:

```
makelist(random(10),k,1,10);
[6, 9, 9, 3, 9, 6, 6, 6, 3, 0]
```

If we change the integer number *10* to the floating point value *10.0* in the *random* function call above, the resulting random numbers are floating point, e.g.,

```
makelist(random(10.0),k,1,10);
[ 1.664698572439665 , 5.342264221033901 , 8.52705109085575 , 8.812042615541047 ,
0.80974462919851 , 9.138913189532065 , 2.750394801412726 , 0.33635484022142 , 6.363346213572809 ,
0.99208527907885 ]
```

If we want to produce a random number *x* between values *a* and *b*, we can generate a random number *r* between *0.0* and *1.0*, and use the relationship $x = a + r\cdot(b-a)$. For example, to generate *10* random values between *-5* and *10*, we can use:

```
makelist(-5.0+random(1.0)*(10-(-5)),k,1,10);
[ 8.248739157702842 , - 4.729435607386618 , 7.591978910199609 , - 3.357518714666188 , -
3.632824555553826 , - 3.546505018155282 , 7.951666725587369 , 3.351700928817859 , - 2.917889816049209 ,
1.452751680964251 ]
```

If we wanted to convert this list to integer values only, we use function *fix* mapped onto the list shown above, i.e,

```
map(fix,%);
[ 1 , 5 , 8 , 8 , 0 , 9 , 2 , 0 , 6 , 0 ]
```

Random matrices

The following functions can be used to generate matrix of random elements. Suppose that the matrix has dimensions $n \times m$ and that the random numbers will be generated between values *a* and *b*, then we can define functions *randmatrix* and *randmatrixfix* to generate matrices of floating-point or integer values, respectively:

```
randmatrix(m,n,a,b):=sum(sum(ematrix(m,n,a+random(1.0)*(b-a),i,j),j,1,n),i,1,m);
```

$$\text{randmatrix}(m, n, a, b) := \sum_{i=1}^{m} \sum_{j=1}^{n} \text{ematrix}(m, n, a + \text{random}(1.0)(b - a), i, j)$$

```
randmatrixfix(m,n,a,b):=matrixmap(fix,sum(sum(ematrix(m,n,a+random(1.0)*(b-a),i,j),j,1,n),i,1,m));
```

$$\text{randmatrixfix}(m, n, a, b) := \text{matrixmap}\left(fix, \sum_{i=1}^{m} \sum_{j=1}^{n} \text{ematrix}(m, n, a + \text{random}(1.0)(b - a), i, j)\right)$$

Examples of random matrices generated with these functions follow:

```
randmatrix(3,2,-10,10);
```

$$\begin{bmatrix} 4.154784323600014 & -8.08326788012749 \\ 8.034136856811855 & 2.481124780811509 \\ 5.017574067991353 & 7.087206400961335 \end{bmatrix}$$

```
randmatrixfix(3,2,-10,10);
```

$$\begin{bmatrix} -5 & -3 \\ 4 & 7 \\ -1 & 4 \end{bmatrix}$$

Matrix size

The size of a matrix can be determined by function *matrix_size*, e.g.,

```
A : matrix([1,2,3],[3,5,-1]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 5 & -1 \end{bmatrix}$$

```
sA : matrix_size(A);
[ 2 , 3 ]
```

The number of rows and columns can be extracted by using:

`nrows : sA[1];`	`ncols : sA[2];`
2	3

Individual elements

Individual elements of a matrix are referred to by using sub-indices as illustrated in the following examples:

`A[1,1];`	`A[2,3];`	`A[2,2]+A[1,2];`
1	-1	7