An Overview of Maxima

Note: You may notice differences between this wxMaxima document and the equivalent Maple worksheet or Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of functional differences between Maxima, Maple, and Mathematica.

1 General Information

OBJECTIVE: Learn the basic features, characteristics, and language structure of Maxima.

The working environment for Maxima is called a wxMaxima document. This Overview is an example of a wxMaxima document. Each document is divided into cells as indicated by the brackets on the left margin of the page. A cell that contains executable commands is called an input cell and has a small triangle at the upper end of the cell bracket. If you start to type in a Maxima document, by default the program understands the typing as an executable command and so will open a new input cell. To type other kinds of cells, like the titles above or the text in this cell, for example, pull down the Cell menu at the top of the page and select the type of cell that you would like to insert.

You can type several commands in a single section by separating them with commas (,). If a semicolon (;) follows a command, the output from the command is displayed. If a dollar sign ($) follows a command, its output is suppressed. One or the other is required at the end of each command. In a section with executable commands, pressing Enter moves the cursor to a new line in the same section without executing the commands in the section. If you wish to start a new section you can press one of the following shortcut keys to create a new cell after the current cell (or select the cell type from the Cell menu):

F5 (inserts a new input cell)
F6 (inserts a new text cell)
F7 (inserts a new subsection cell)
F8 (inserts a new section cell)
F9 (inserts a new title cell)

To execute a Maxima command or group of commands in a section, place the cursor anywhere in the section containing the command(s) to be executed and press Ctrl-Enter or Shift-Enter. All of the commands in the section are executed in sequence from the first to the last. If you wish to evaluate all cells in a document, you can select that option from the "Cell" pull-down menu or press Ctrl-r.

In mathematical formulas, always use the times symbol (*) for multiplication. The (^) symbol is used for exponents (i.e. x squared would be written x^2.

There are several Panes (palettes) available for typing standard mathematical forms in commands and in text cells. These are found in the Maxima (Panes) pull-down menu. There are currently panes available for General Math, Statistics, History, and Insert Cells. Many of these standard mathematical forms can also be found in the various menus.

The "Help" pull-down menu is extremely useful when using Maxima. Formats for Maxima commands are given with explanations of how they function and with examples.

There is a set of Technology Guidelines in a separate subsection at the end of the Introduction to each module. Until

1.1 Technology Guidelines

NOTE: If you have just finished a document, restart Maxima before executing a new document. This can be done by choosing "Restart" from the Maxima menu.

TO OPEN OR CLOSE CELLS,
Click on the arrow at the top of the cell bracket.

TO STOP AN EXECUTION
Click on STOP button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any Maxima Input lines within a given document.

Alternatively, you can execute the entire worksheet by selecting the "Evaluate All Cells" command from the "Cell" drop down menu or simply press Ctrl-r.

SAVING WORKSHEETS
You can save anytime to any directory you choose, and it is wise to save often.

EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down Maxima and start it up again.

2 Built-In Commands

Maxima commands consist of a word or a string of words followed by a series of arguments enclosed in a pair of parentheses ( ).

Note that by default when you type an opening parenthesis '(', Maxima automatically puts in a closing parenthesis ')'. To turn this feature off (or on), choose "Configure" from the "Edit" menu. Then click on the box next to "Match parenthesis in text controls".

The following are examples of some Maxima commands.

When %piargs is set to true, trigonometric functions are simplified to algebraic constants.

```
(%i1) %piargs:true;
(%o1) true
```

Their are several system variables that are predefined in Maxima. These are written with a '%' at the beginning of the variable name. The number PI (3.14159...) is one of these and is thus written in expressions as %pi.

Also note that the following two expressions show that extra white is ignored when evaluating cells. Therefore, the following two expressions are identical even though the second one is written on mutliple lines.

```
(%i2) sin(%pi/4);
```

(%o2) $\frac{1}{\sqrt{2}}$

```
(%i3) sin (
      %pi / 4
      );
```

(%o3) $\frac{1}{\sqrt{2}}$

```
(%i4) sqrt(16);
```

(%o4) 4

```
(%i5) abs(x);
```

(%o5) $|x|$

When expandwrt_denom is set to false, the denominator in a rational expression is not expanded when the expandwrt function is used.

Also observe that Maxima stores and outputs polynomial expressions in ascending order.

```
(%i6) expandwrt_denom :false;
```

(%o6) false

```
(%i7) expandwrt((a-b)^2/(c+d)^3,a,b,c,d);
```

(%o7) $\frac{b^2}{(d+c)^3}-\frac{2\,a\,b}{(d+c)^3}+\frac{a^2}{(d+c)^3}$

```
(%i8) solve([x^2=2], [x]);
```

(%o8) $[x=-\sqrt{2},x=\sqrt{2}]$

Words in a command string are typically in lowercase letters. Commands containing multiple-words are typed with no spaces between the words.

In the first four preceding commands, the list of arguments contains only one item, whereas, in the solve( ) command, the list of arguments contains two items, the equation to solve and the symbol that is to be isolated in the solution. When a command has more than one argument, they are separated with commas.

3 Lists

Another important structure in Maxima is the list. A list is any ordered array of items. In Maxima , lists are enclosed in a pair of square brackets [ ] , and the items in a list are separated by commas. The following are examples.

(%i9) [a,b,c,d];

(%o9) $[a,b,c,d]$

(%i10) [a,a^2,a^3,a^4];

(%o10) $[a,a^2,a^3,a^4]$

(%i11) f: [[1,2],[2,3],[3,4]];

(%o11) [[1,2],[2,3],[3,4]]

The last list consists of three items, and each of these items is a list of two items.

A list of lists can also be thought of as a matrix or an array. The matrix( ) command shows a list of lists in more standard matrix form. Each list inside the list is a row of the matrix.

(%i12) c: matrix([1,2],[2,3],[3,4]);

(%o12) $\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$

(%i13) d: matrix([0,3],[0,0],[-1,2]);

(%o13) $\begin{bmatrix} 0 & 3 \\ 0 & 0 \\ -1 & 2 \end{bmatrix}$

Note that the matrix form of a list tends to be more readable readable. Also a matrix is still considered a mutli-dimensional list, which allows to do calculations that involve both lists and matrices together as illustrated below.

```
(%i14) f+d;
```

(%o14) $\begin{bmatrix} 1 & 5 \\ 2 & 3 \\ 2 & 6 \end{bmatrix}$

The makelist( ) command is very useful for generating a list when you know a formula for the elements in the list, as in the following example.

```
(%i15) makelist(2*i, i, 1, 5);
(%o15) [2,4,6,8,10]
```

```
(%i16) makelist(2, i, 1, 5);
(%o16) [2,2,2,2,2]
```

The first argument of the makelist( ) command is a formula that gives each element of the sequence inside the list as a function of the index variable, i . The second, third, and fourth arguments specify that the elements of the sequence are to be generated by replacing the index i in the formula with the integers from 1 thought 5, each in turn.

You can generate lists of lists in a variety of ways. For example, when the formula for the elements of the list is a function of two indices, you can nest makelist( ) commands.

```
(%i17) makelist(makelist(i*j,j,-2,2),i,1,5);
(%o17) [[-2,-1,0,1,2],[-4,-2,0,2,4],[-6,-3,0,3,6],[-8,-4,0,4,8],[-10,-5,0,5,10]]
```

You can generate the same list like this.

```
(%i18) makelist([-2*i,-i,0,i,2*i],i,1,5);
(%o18) [[-2,-1,0,1,2],[-4,-2,0,2,4],[-6,-3,0,3,6],[-8,-4,0,4,8],[-10,-5,0,5,10]]
```

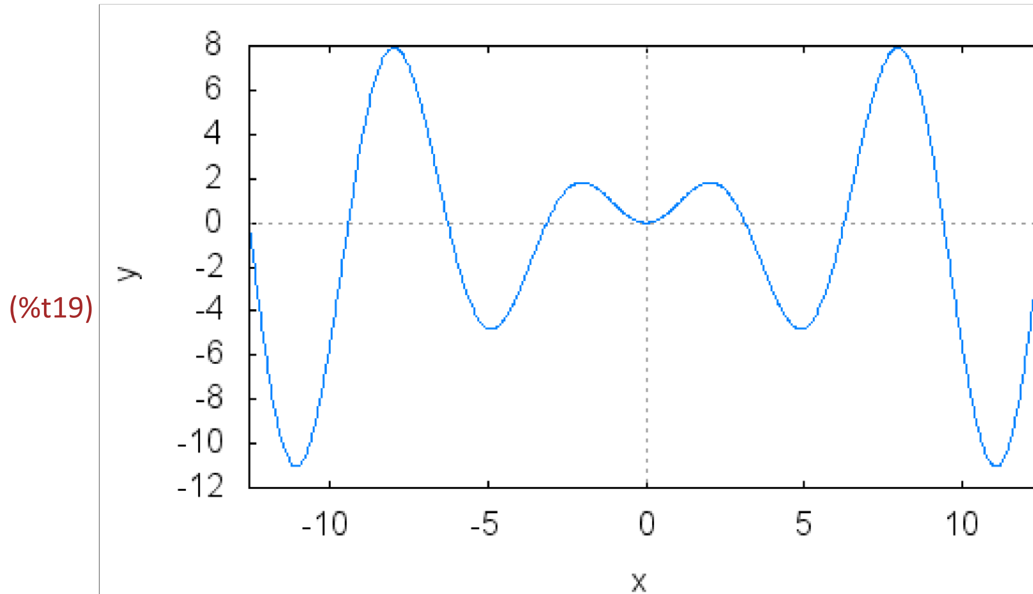
4 2-D Plots

Numerous two-dimensional plot commands in Maxima are used to generate graphical displays of functions and data. The 2-D plot function that is available by default is plot2d( ). This function will display the resulting graph in a popup window. If you want your graph to show up in your document as an inline graph, then use the wxplot2d( ) function.

The plot2d( ) command is used to graph functions of the form y = f (x).

```
(%i19) wxplot2d(x*sin(x),[x,-4*%pi,4*%pi],[xlabel,"x"],[ylabel,"y"]);
```

(%t19)



(%o19)

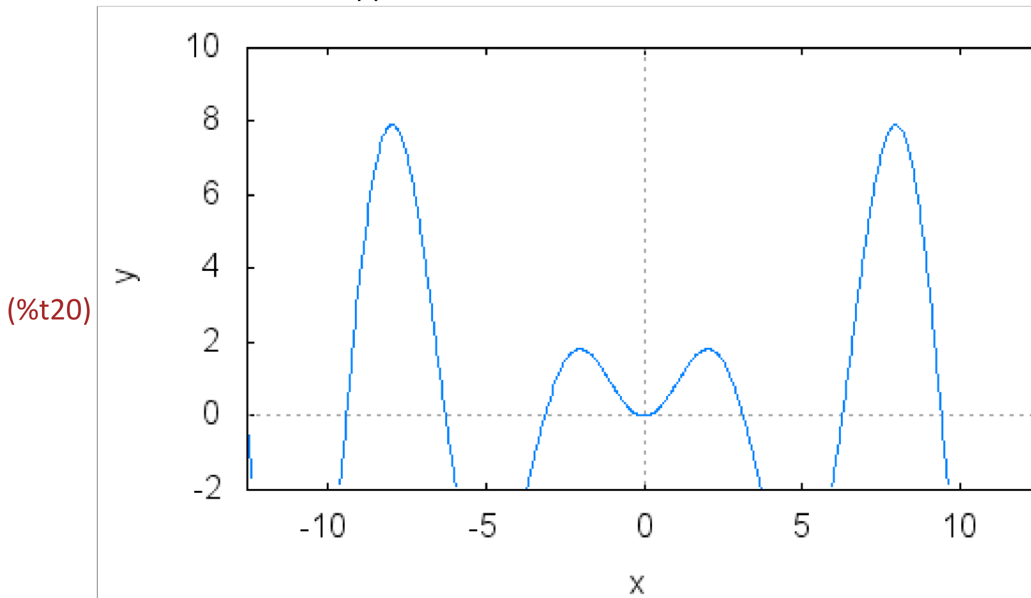
The first argument of the plot2d( ) command is the function to graph, and the second specifies the independent variable, plus the left and right ends of the domain over which you wish to graph the function. The first two arguments are required, but you can add other optional specifications like [xlabel,"x"] and [ylabel,"y"] in the preceding example. Additional options are presented below.

By default, Maxima selects the range (vertical limits) for the graph. However, you can override this by specifying your own vertical limits as follows.

```
(%i20) wxplot2d(x*sin(x),[x,-4*%pi,4*%pi],[y,-2,10],[xlabel,"x"],[ylabel,"y"]);
```

plot2d: some values were clipped.

(%t20)



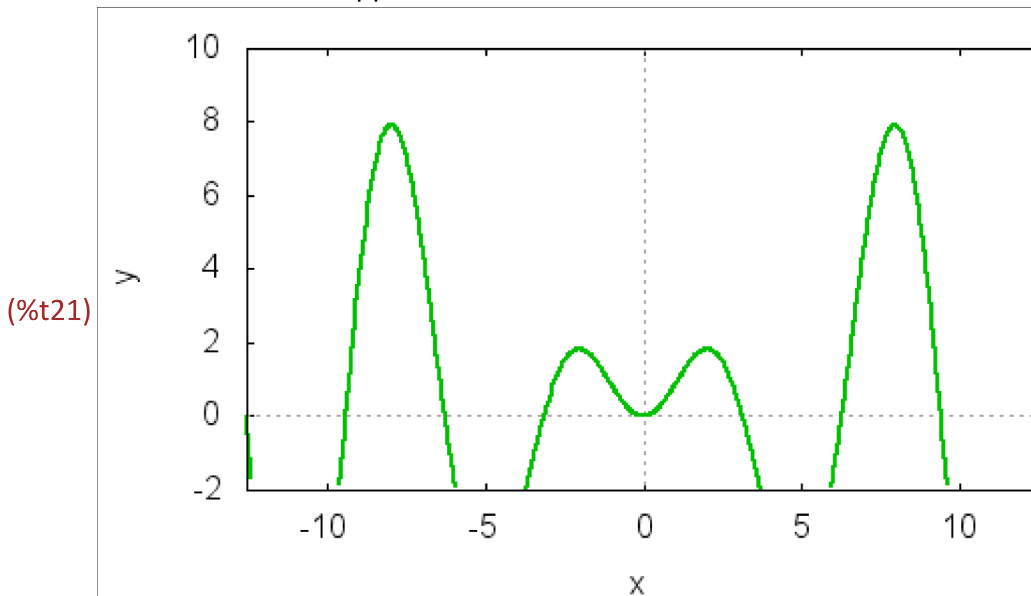
(%o20)

You can format a graph by specifying a wide variety of options after the first two required arguments in the plot2d( ) command.

```
(%i21) wxplot2d(x*sin(x),[x,-4*%pi,4*%pi],[y,-2,10],[style,[lines,2,3]],[xlabel,"x"],[ylabel,"y"]);
```

plot2d: some values were clipped.

(%t21)



(%o21)

The option style assigns the type of the curve, line width and line color (and the point type, if desired). In pixel graphics the line width is given in pixels, in vector graphics as multiples of 0.25pt (about 0.088mm). The curve type can obtain the following values:

```
lines       . . . solid line
points      . . . points (with an additional integer number
                  assigning the point type)
linespoints . . . solid line and points
impulses    . . . bars (line widths and colors are ignored)
```

Colors can have the following values:

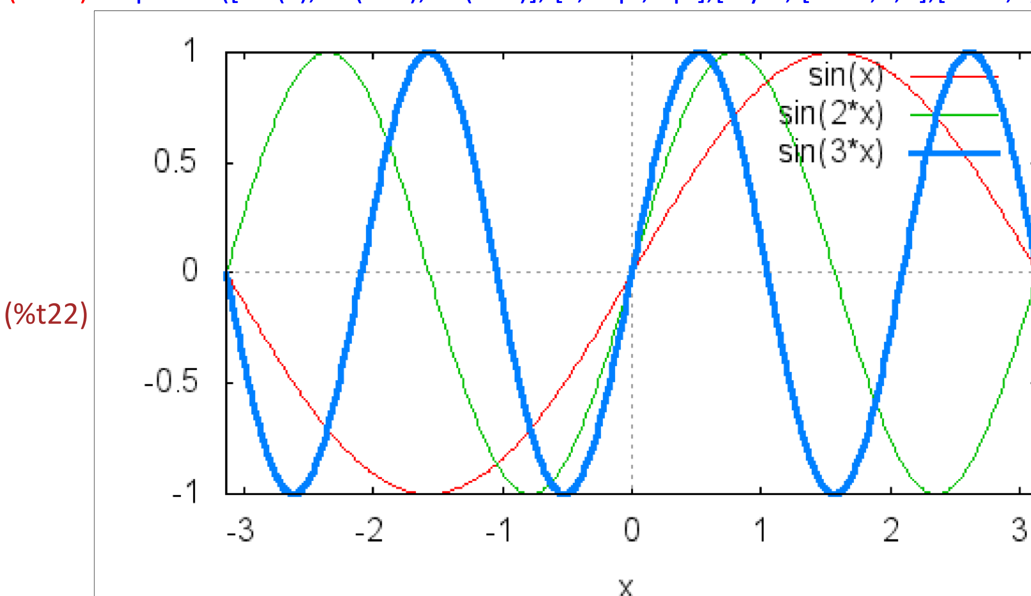
```
1 . . . blue
2 . . . red
3 . . . green
4 . . . magenta
5 . . . black
6 . . . cyan
```

Note that the style option in the command above is [style,[lines,2,3]]. This indicates that the curve will be a solid line, the line thickness is 2 pixels, and the line color is green (3).

Other options can be added to plot2d( ) commands. Commas are used separate whatever options are specified. The following command plots three functions on the same graph; each line is a different color, and the third line is thicker.

```
(%i22) wxplot2d([sin(x),sin(2*x),sin(3*x)], [x,-%pi,%pi],[style, [lines,1,2],[lines,1,3],[lines,3,1]]);
```

(%t22)



(%o22)

Note that inside the style option, the seperate list of style options is given for each function in order from first to last. as they occur in the list of functions to plot. So for sin(x)the style options are [lines,1,2], for sin(2*x) the style options are [lines,1,3], and for sin(3*x) the style options are [lines,3,1].

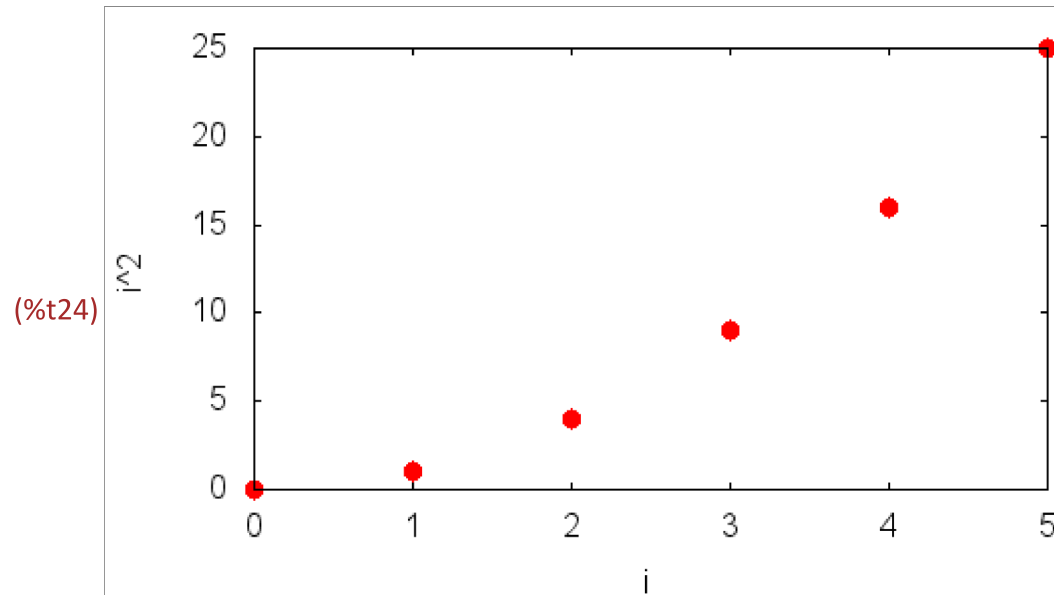
The plot2d( ) function also allows you to plot a discrete set of points. First we will create a list of points.

```
(%i23) plotvalues: makelist([i,i^2], i, 0, 5);
(%o23) [[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
```

Then we plot our list of points using the discrete option for the first argument in the plot2d( ) function.

```
(%i24) wxplot2d([discrete, plotvalues],[style, [points,3,2,1]], [xlabel,"i"],[ylabel,"i^2"]);
```

(%t24)



(%o24)

The style option for the above graph is [style, [points,3,2,1]]. The size of the points is 3, the color is red (2), and the point type is 1 (a closed circle).

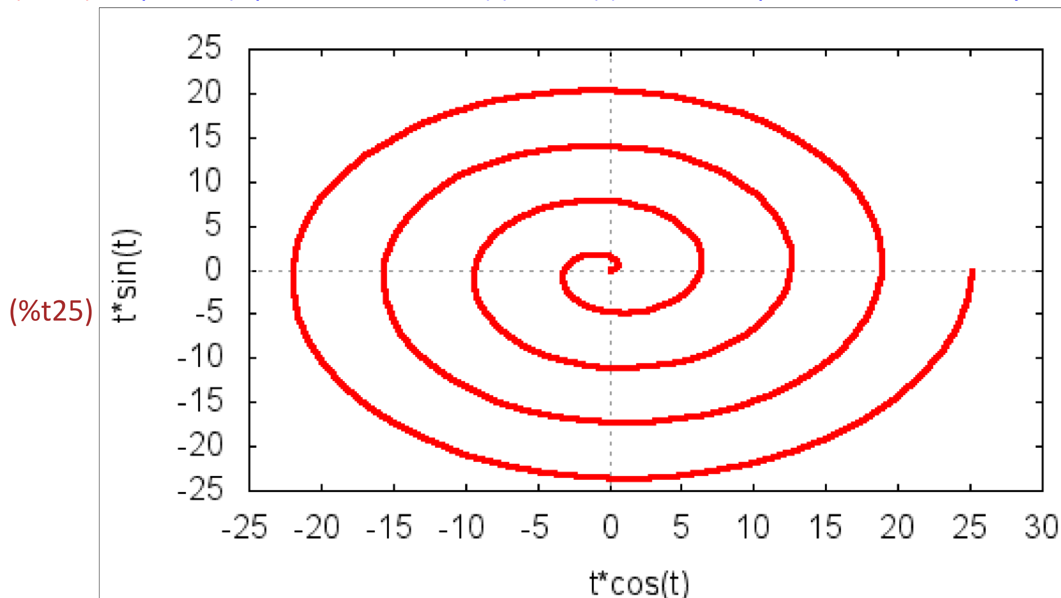
The possible points types are:

1 . . . closed circle
2 . . . open circle
3 . . . +
4 . . . X
5 . . . *
6 . . . closed square
7 . . . open square
8 . . . closed triangle (up)
9 . . . open triangle (up)
10. . . closed triangle (down)
11. . . open triangle (down)
12. . . closed diamond
13. . . open diamond

Graphs of curves expressed in parametric form are plotted with the plot2d( ) command. For example, if the x-coordinates of the points on a curve are given by x = t cos t , and the y-coordinates are given by y = t sin t , then the curve can be graphed as follows using the parametric option for the first argument of the plot2d( ) function.

```
(%i25) wxplot2d([[parametric, t*cos(t), t*sin(t), [t, 0, 8*%pi],[nticks,200]]],[style,[lines,3,2]])$
```

(%t25)



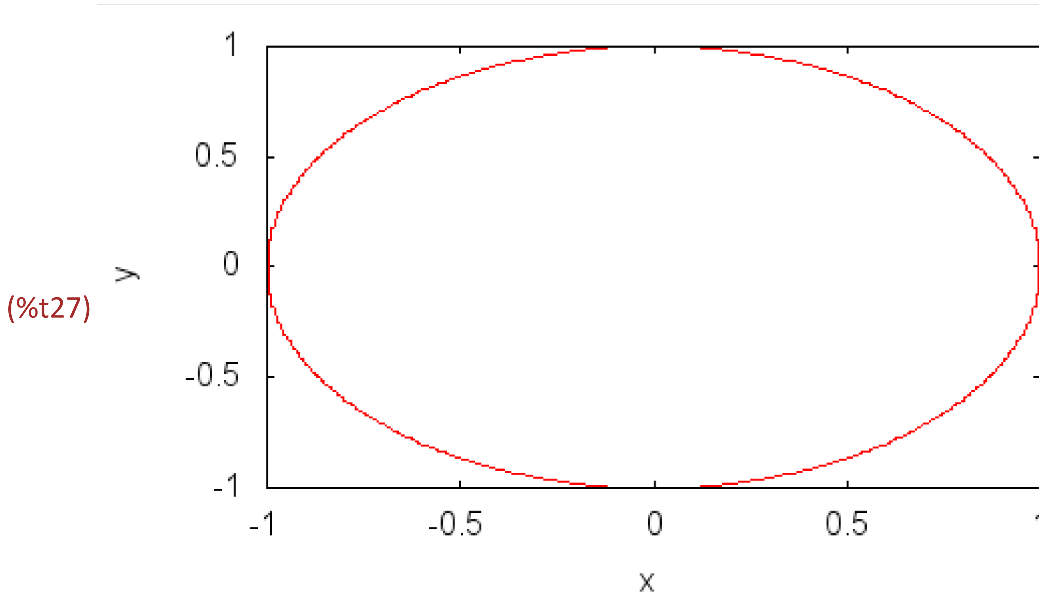
Unfortunately, the plot2d( ) function does not currently allow you to graph a function implicitly. In order to graph implicit functions, we must load the draw package and use the draw2d( ) function contained in this package. Note that all of the 2d plots above can also be drawn using this draw2d( ) function.

(%i26) load(draw);

(%o26) C:/PROGRA~1/MAXIMA~2.0/share/maxima/5.25.0/share/draw/draw.lisp

You can now use the draw2d( ) command to plot curves for mathematical relations that are not functions; that is, they cannot be written in the form y = f (x). For example, you can graph the curve x^2 + y^2 = 1 as follows.

(%i27) wxdraw2d(nticks=200,color=red,implicit(x^2+y^2=1,x,-1,1,y,-1,1),xlabel="x",ylabel="y");

(%t27) 

(%o27)

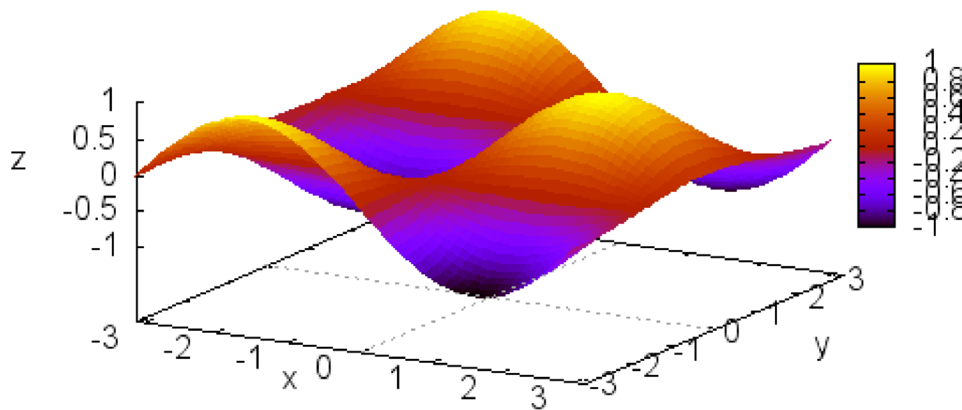
5 3-D Plots

There are numerous Maxima commands for plotting three-dimensional functions and data. In this section we introduce two of them, and in later modules we will introduce more.

The plot3d( ) command can be used to plot functions of two independent variables, that is, functions of the form z=f(x,y).

(%i28) wxplot3d(sin(x)*cos(y), [x,-%pi,%pi], [y,-%pi,%pi],[z,-1,1],
[grid,50,50],[legend,false],[xlabel,"x"],[ylabel,"y"],[zlabel,"z"]);

(%t28)



(%o28)

The first three arguments of the plot3d( ) command are required. They are: 1) the function to graph, 2) the first independent variable and its limits for the graph, and 3) the second independent variable and its limits. As is the case with two-dimensional graphics, you can specify a number of formatting options after the first three arguments in the command.

The inline version of this function (wxplot3d) does not allow you to change the plot's perspective. The following plot3d( ) function will display the 3d plot in a pop-up window which will allow you to set viewpoint options and will allow you to change the perspective of the plot real-time as well. Also note that the azimuth and elevation options are used to change the default perspective of the plot.

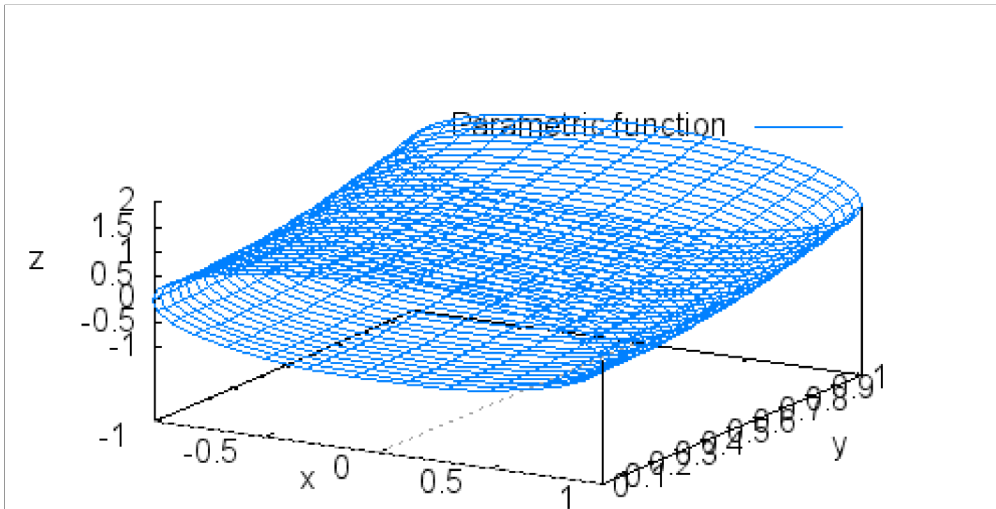
```
plot3d(sin(x)*cos(y), [x,-%pi,%pi], [y,-%pi,%pi],[z,-1,1],
 [grid,50,50],[legend,false],[xlabel,"x"],[ylabel,"y"],
 [zlabel,"z"], [azimuth, -66], [elevation, 18],
 [plot_format, openmath]);
```

You may copy and paste this function into an input cell to observe this functionality.

You may also plot three-dimensional curves that are specified in parametric form. For example suppose your parametric equations defining your curve are: x=cos(u), y=v, and z=v^2+sin(u). You would plot this as follows.

```
(%i29) wxplot3d([cos(u),v,v^2+sin(u)], [u,0,2*%pi],[v,0,1],[palette,false]);
```

(%t29)



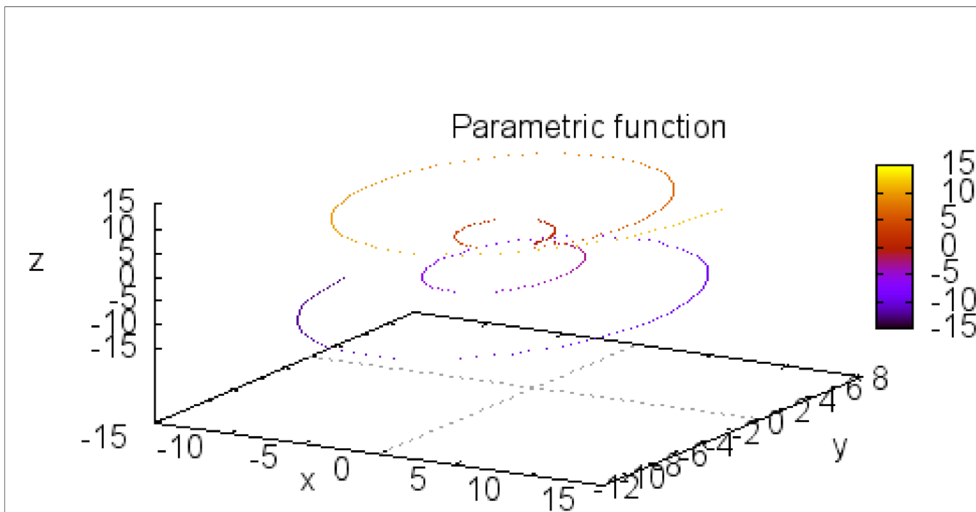
(%o29)

The palette options in the previous function turns off the color palette.

Now suppose that your parametric equations defining your curve are only in terms of one variable. For example, if the x-coordinates of the points on a curve are given by t cos t , the y-coordinates by t sin t , and the z-coordinates by t , then the curve can be graphed as follows.

```
(%i30) wxplot3d([t*cos(t), t*sin(t), t+(1e-230*x)], [t, -4*%pi, 4*%pi], [x, -1, 1], [grid,300,2]);
```

(%t30)



(%o30)

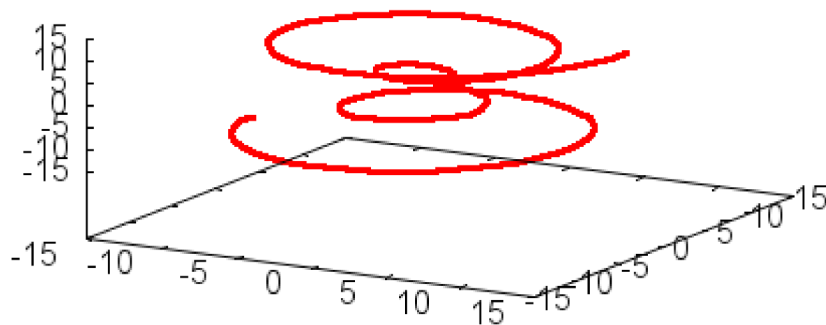
Since the plot3d( ) function currently requires the use of two variables in the parametric equations, we used a workaround by changing the last parametric equation from z=t to z=t+(1e-230*x). 1e-230*x is Maxima version of scientific notation (1 x 10^(-230*x)).

Using the draw3d( ) function in the draw package provides a much more elegant way to plot this particular set of parametric equations.

```
(%i31) load(draw);
(%o31) C:/PROGRA~1/MAXIMA~2.0/share/maxima/5.25.0/share/draw/draw.lisp
```

```
(%i32) wxdraw3d(
        surface_hide=true,
        nticks=200,
        line_width=3,
        color=red,
        xrange=[-15,15],
        yrange=[-15,15],
        zrange=[-15,15],
        parametric(t*cos(t),t*sin(t),t,t,-4*%pi,4*%pi));
```

(%t32)



(%o32)

Note that using draw3d( ) instead of wxdraw3d( ) will display the graph in a pop-up window which will allow you to dynamically change the perspective of the plot.

```
draw3d(
    surface_hide=true,
    nticks=200,
    line_width=3,
    color=red,
    parametric(t*cos(t),t*sin(t),t,t,-4*%pi,4*%pi));
```

Copy and paste the above command into an input cell to observe this functionality.

6 Assignment Commands

In Maxima the colon (:) is used to assign items (values, lists, graphs, etc.) to variable names. Whenever a colon assignment command is executed, Maxima evaluates the expression on the right side of the colon and assigns the result to the variable name on the left side. If the expression on the right side includes variables that have not been assigned values, then Maxima evaluates the right-side expression to the extent possible with values that are available, and then assigns the resulting expression to the variable on the left side of the colon. Here are some examples.

```
(%i33) x: 2;
(%o33) 2
```

```
(%i34) y: x+2;
(%o34) 4
```

```
(%i35) z: y^3+w;
```
(%o35) $w+64$

```
(%i36) s: 70+w;
```
(%o36) $w+70$

In the third assignment command above, the variable w has no assigned value, but y does. Therefore, evaluation gives an expression in terms of w, and this expression is assigned to z.

You can always check to see what has been assigned to a variable by typing the name of the variable followed by a semicolon.

```
(%i37) x;
(%o37) 2
```

```
(%i38) y;
(%o38) 4
```

```
(%i39) z;
(%o39) w+64
```

```
(%i40) s;
(%o40) w+70
```

Once a value or expression has been assigned to a variable, the variable name is replaced with the assigned value or expression wherever it is used in subsequent expressions, as shown in the following examples.

```
(%i41) x^2;
(%o41) 4
```

```
(%i42) 3*y^2;
(%o42) 48
```

```
(%i43) sin(z);
(%o43) sin(w+64)
```

The kill( ) command removes assignments that you have made to variables.

```
(%i44) kill(x);
(%o44) done
```

If you check a variable name after it has been unassigned, you see that it is not replaced with a value or expression.

```
(%i45) x;
(%o45) x
```

If you use the variable name in an expression, it will be treated as a simple variable with nothing assigned to replace it.

```
(%i46) x+2;
(%o46) x+2
```

You can kill more than one variable at a time.

```
(%i47) kill(x,y,z);
(%o47) done
```

```
(%i48) x;
(%o48) x
```

```
(%i49) y;
(%o49) y
```

```
(%i50) z;
(%o50) z
```

Redefining a variable automatically clears whatever was previously assigned to it. Therefore, it is unnecessary to kill a variable name when a new assignment is made to it. The following example demonstrates this feature.

```
(%i51) a:2;
(%o51) 2
```

```
(%i52) a;
(%o52) 2
```

```
(%i53) a:12345;
(%o53) 12345
```

```
(%i54) a;
(%o54) 12345
```

It is extremely important to keep close track of the variable names that you use in a Maxima session. If you use a new variable name in an expression or a command and you are assuming that it is clear of assignments, you should first use the kill( ) command to erase all previous assignments to the new variable name. Failure to do this can produce misleading and confusing results. The following commands illustrate what can happen if you forget to kill a variable name before you use it. Suppose we want to use s in an expression and we have forgotten that we previously assigned another expression to s.

```
(%i55) expand((s+2)*(s-3));
```

(%o55) $w^2+139\,w+4824$

Now we fix the problem.

```
(%i56) kill(s);
```
(%o56) *done*

```
(%i57) expand((s+2)*(s-3));
```
(%o57) $s^2 - s - 6$

That's more like it!

There is another item that is related to the issue of variable-name management. If you finish a Maxima document or one of our modules and wish to start another one, it is best to save your work and then execute the restart command at the beginning of the new module or document. This clears any assignments that have been made to variables. Otherwise, the assignments that were made in the old document are carried over into the new one. Executing the restart command also removes any packages that were previously read into memory. Therefore, if you want to use a package after you restart Maxima , you need to reload it.

7 Special Packages

While there are many built-in commands that are available for use as soon as you open a Maxima document, there are many more specialized commands that are contained in packages. These packages are not automatically available in a Maxima document, and in order to use the specialized commands that they contain, you have to read them into computer memory. You do this by typing load(packagename) and executing the command. Here is an example.

```
(%i58) reset();
```
(%o1) [*tr-unique* , *lispdisp* , *features* , *labels* , _ , % , *multiplicities* , *linenum* , __ , *piece*]

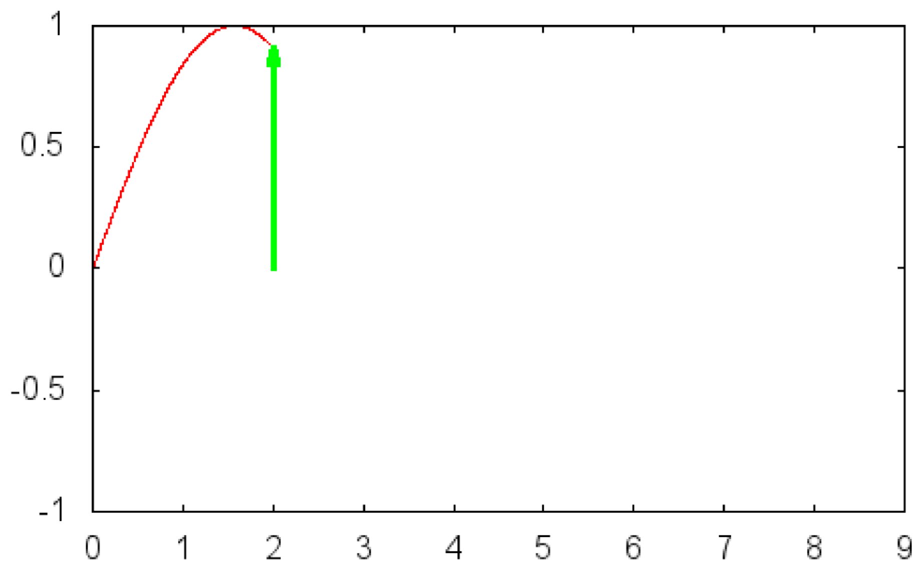
```
(%i2) kill(all);
```
(%o0) *done*

```
(%i1) load(draw);
```
(%o1) *C:/PROGRA~1/MAXIMA~2.0/share/maxima/5.25.0/share/draw/draw.lisp*

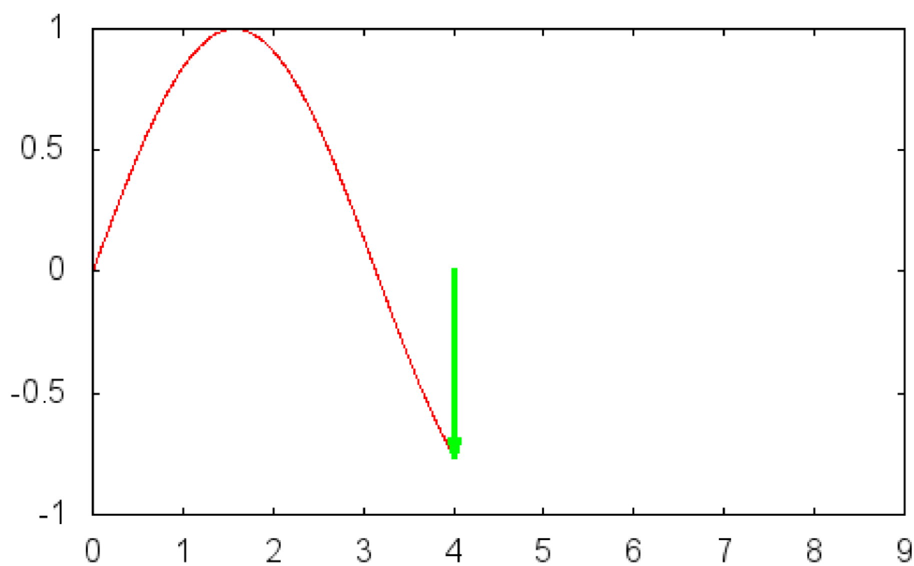
Now we can continue to explore some of the features of the draw package. In the following example, we introduce the vector function which is available after the draw package is read into memory.

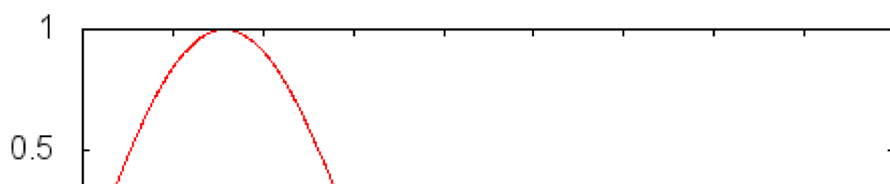
```
(%i2) for a: 2 thru 8 step 2 do (
        arr1: vector([a,0],[0,sin(a)]),
        p1: explicit(sin(x),x,0,a),
        wxdraw2d(
          xrange=[0,9],
          yrange=[-1,1],
          nticks=200,
          color=red,
          p1,
          head_length=0.2,
          head_angle=15,
          line_width=3,
          color=green,
          arr1));
```

(%t2)



(%t3)





The command in the preceding section is a loop that creates the series of graphs. The first command inside the loop (indented) defines the arrow and the second indented command defines the plot of sin x . The third indented command displays the two plot objects, the arrow and the curve, together. Note that each command within the parentheses following "do" have a comma separating them. The specifications in for_do command defines the variable a as the loops counter, to vary from 2 to 8 in steps of 2.

To find out what Maxima packages are available type F1 to open the help files. Chapters 43-84 currently give detailed explanations of the contents of each individual package that has been bundled with Maxima. You may use go to Appendix B "Documentation Categories" where a condensed list of all of the available packages are listed, including the functions available in each one.

Before you use any of the specialized commands available in a package, be sure to load the package first. A package needs to be loaded in only once during a Maxima session unless you restart. If you try to execute a package command before you have loaded the package, the command won't work and you may get an error message. You can fix the problem by loading in the package. Then the command will work.

8 Making Your Own Commands

As a user and programmer in Maxima, you can design and construct your own functions and commands.

Suppose, for example, that you are going to use the function over and over again in a program. You may wish to create a new function, as illustrated by the following definition.

```
(%i6) reset();
```

(%o1) [$lispdisp, labels, _, \%, linenum, __, piece$]

```
(%i2) kill(all);
```

(%o0) *done*

```
(%i1) myfunction (x)  :=  x^2*sin(x);
```

(%o1) $\mathrm{myfunction}(x) := x^2 \sin(x)$

Maxima displays the definition of the new function. Note that the colon-equals (:=) is used when defining functions.

The variable x is a dummy variable that will be replaced by real input in the argument list, whenever myfunction( ) is called. The following examples illustrate how you to use myfunction( ) after you have defined it.

```
(%i2) myfunction(2);
```
(%o2) $4\sin(2)$

```
(%i3) myfunction(2.0);
```
(%o3) 3.637189707302727

```
(%i4) myfunction(r);
```
(%o4) $r^2 \sin(r)$

```
(%i5) myfunction(a+b);
```
(%o5) $(b+a)^2 \sin(b+a)$

The built-in map( ) command can be used to apply a function to each of the values in a list.

```
(%i6) map(myfunction,[2,4]);
```
(%o6) $[4\sin(2), 16\sin(4)]$

The first argument of the map( ) command is the name of the function to apply (without the parentheses), and the second element is the list of values to be used in place of the function's argument. In the preceding example, myfunction is mapped over the list [2, 4].

The last example also illustrates an important feature of most Maxima commands: they are "listable." This means that if you put in a list as an input to a command, Mathematica performs the command operation(s) on each of the elements in the list and returns a list of the results. This even works for simple operations like addition, multiplication, or raising a number to a power.

```
(%i7) [1,2,3]+[4,5,6];
```
(%o7) [5,7,9]

```
(%i8) [1,2,3,4]*[-2,2,3,5];
```
(%o8) [-2,4,9,20]

```
(%i9) [-2,3,5,9]^2;
(%o9) [4,9,25,81]
```

You can design commands that perform more than one operation by using the block( ) command. To create your own command, use a colon-equals (:=) assignment. On the left side of the := , place the name that you wish to use for your command, and on the right side, place the block( ) command. Inside the parentheses of the block command type the series of commands to be executed, separating each command with a comma.

The following example shows how to construct a command called buildplot( ). We want buildplot( ) to generate a list of decaying numbers, plot the list, and return the list as output. First, we load the draw package into memory, because buildplot( ) will use functions command from this package.

```
(%i10) load(draw);
(%o10) C:/PROGRA~1/MAXIMA~2.0/share/maxima/5.25.0/share/draw/draw.lisp
```

```
(%i11) buildplot (n) :=
          block(
            plotlist: makelist([i,0.8^(i-1)],i,1,n),
            g8: points(plotlist),
            wxdraw2d(
              point_size=0.5,
              point_type=1,
              g8),
            return (plotlist));
```

(%o11) buildplot(n) := block$(\,plotlist : \mathrm{makelist}([i, 0.8^{i-1}], i, 1, n), g8 : \mathrm{points}(plotlist),$
$\mathrm{wxdraw2d}(point_size = 0.5, point_type = 1, g8), \mathrm{return}(plotlist)\,)$

Note that in the wxdraw2d( ) function above, the point_type
attribute setting corresponds to the following values:

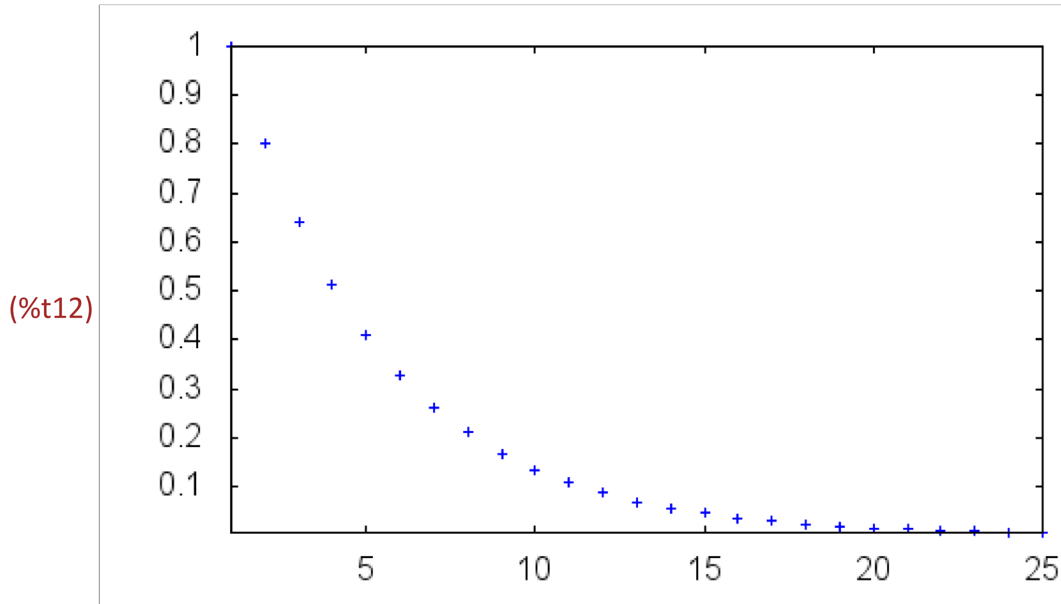
-1. . . none
0 . . . dot
1 . . . +
2 . . . X
3 . . . *
4 . . . open square
5 . . . closed square
6 . . . open circle
7 . . . closed circle
8 . . . open triangle (up)
9 . . . closed triangle (up)
10. . . open triangle (down)
11. . . closed triangle (down)
12. . . open diamond
13. . . closed diamond

In buildplot( ), the variables plotlist and g8 are first
defined. Note that Maxima store these as global variable, which
can still be accessed after the function is terminated.
If you type plotlist; after you execute the buildplot( ) command, you
will find that it still contains the list of points defined in
the previous function call of buildplot( ). We will
check it out below, after we execute the buildplot( ) command.

We execute our new buildplot( ) command the same way we would
execute any other Maxima command.

(%i12) buildplot(25);

(%t12)



(%o12) [[1,1.0],[2,0.8],[3,0.64],[4,0.512],[5,0.4096],[6,0.32768],[7,0.262144],[8,0.2097152],[9,0.16777216],[10,0.134217728],[11,0.1073741824],[12,0.0858993459200001],[13,0.0687194767360001],[14,0.0549755813888001],[15,0.0439804651110401],[16,0.0351843720888321],[17,0.0281474976710656],[18,0.0225179981368525],[19,0.018014398509482],[20,0.0144115188075856],[21,0.0115292150460685],[22,0.0092233720368548],[23,0.00737869762948384],[24,0.00590295810358707],[25,0.00472236648286966]]

Now we check the variable name Plotlist.

(%i13) plotlist;

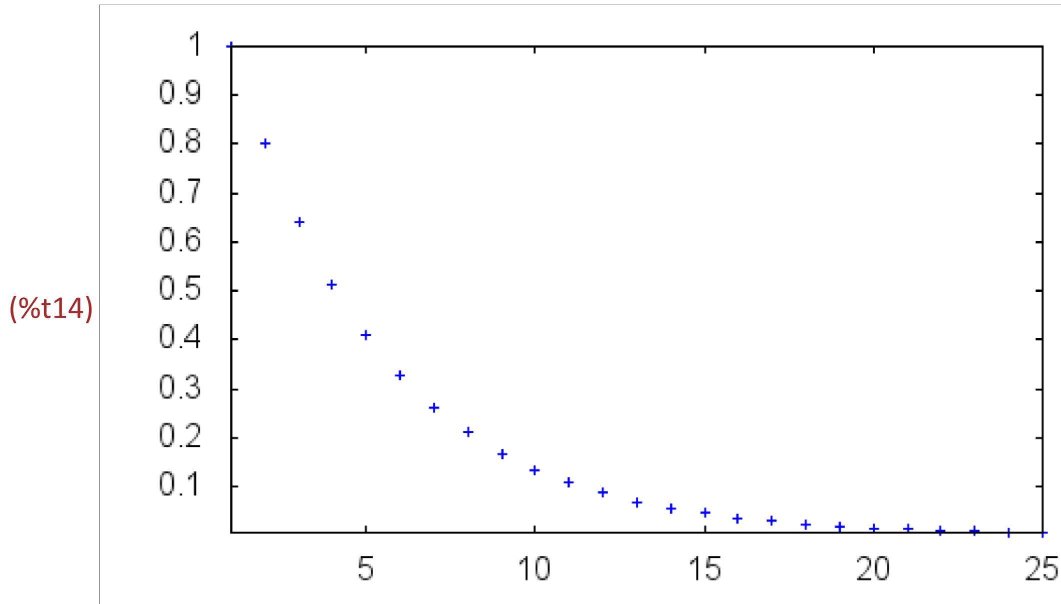
(%o13) [[1,1.0],[2,0.8],[3,0.64],[4,0.512],[5,0.4096],[6,0.32768],[7,0.262144],[8,0.2097152],[9,0.16777216],[10,0.134217728],[11,0.1073741824],[12,0.0858993459200001],[13,0.0687194767360001],[14,0.0549755813888001],[15,0.0439804651110401],[16,0.0351843720888321],[17,0.0281474976710656],[18,0.0225179981368525],[19,0.018014398509482],[20,0.0144115188075856],[21,0.0115292150460685],[22,0.0092233720368548],[23,0.00737869762948384],[24,0.00590295810358707],[25,0.00472236648286966]]

In the global environment outside the buildplot( ) procedure, the variable plotlist is still defined with the list of points created in the previous function call of buildplot( ).

The return(plotlist) command inside the procedure returns the list of values stored in the plotlist variable. If you want to save the output list of values for later use, you can assign it to a variable name, as we do next.

(%i14) decay: buildplot(25);

(%t14)



(%o14) [[1,1.0],[2,0.8],[3,0.64],[4,0.512],[5,0.4096],[6,0.32768],[7,0.262144],[8,
0.2097152],[9,0.16777216],[10,0.134217728],[11,0.1073741824],[12,0.0858993459200001],[
13,0.0687194767360001],[14,0.0549755813888001],[15,0.0439804651110401],[16,
0.0351843720888321],[17,0.0281474976710656],[18,0.0225179981368525],[19,
0.018014398509482],[20,0.0144115188075856],[21,0.0115292150460685],[22,
0.0092233720368548],[23,0.00737869762948384],[24,0.00590295810358707],[25,
0.00472236648286966]]

If you use the variable name decay , you will now get the list of numbers that buildplot( ) returned as output.

(%i15) decay;

(%o15) [[1,1.0],[2,0.8],[3,0.64],[4,0.512],[5,0.4096],[6,0.32768],[7,0.262144],[8,
0.2097152],[9,0.16777216],[10,0.134217728],[11,0.1073741824],[12,0.0858993459200001],[
13,0.0687194767360001],[14,0.0549755813888001],[15,0.0439804651110401],[16,
0.0351843720888321],[17,0.0281474976710656],[18,0.0225179981368525],[19,
0.018014398509482],[20,0.0144115188075856],[21,0.0115292150460685],[22,
0.0092233720368548],[23,0.00737869762948384],[24,0.00590295810358707],[25,
0.00472236648286966]]

9 Elements of Lists

So far, we have used square brackets [ ] to enclose the elements of a list, and parentheses ( ) to enclose the arguments in Maxima commands. Parentheses ( ) are also used as grouping symbols in mathematical expressions, as in the epression (x + y)*(x - y) for example.

Square brackets also are used to access specific elements in a list. Consider the following examples.

```
(%i16) odds: [1,3,5,7,9,11,13,15];
(%o16) [1,3,5,7,9,11,13,15]
```

To select the third element of odds, use a selector specification after the name of the list, as follows.

```
(%i17) odds[3];
(%o17) 5
```

For nested lists, that is, a list of lists, use multiple selectors like [3, 2] to retrieve elements. The first number points to an individual element in the outermost list, which may itself be a list. The second number points to an element inside the inner list. The following examples illustrate how this works.

```
(%i18) oddeven: [[1,2],[3,4],[5,6],[7,8],[9,10]];
(%o18) [[1,2],[3,4],[5,6],[7,8],[9,10]]
```

```
(%i19) oddeven[4];
(%o19) [7,8]
```

```
(%i20) oddeven[3][2];
(%o20) 6
```

10 Making Substitutions

Normally, when the name of a command is used, the values enclosed by the parentheses after the command name are automatically substituted for the arguments of the command. These values are then used in the evaluating the command's procedure. The following example uses a function to show how this substitution works.

```
(%i21) kill(x,y,z);
(%o21) done
```

(%i22) f (x,y) := sin(x)^2+cos(y)^2;

(%o22) $f(x,y) := \sin(x)^2 + \cos(y)^2$

(%i23) f(%pi/4,%pi);

(%o23) $\frac{3}{2}$

(%i24) f(theta,theta);

(%o24) $\sin(\theta)^2 + \cos(\theta)^2$

(%i25) theta: %pi/8;

(%o25) $\frac{\pi}{8}$

(%i26) f(theta,theta);

(%o26) $\sin\left(\frac{\pi}{8}\right)^2 + \cos\left(\frac{\pi}{8}\right)^2$

11 Retrieving Output From Commands

If you execute a Maxima command, the output is displayed in
on the next line after the command.

(%i27) solve(x^2=1,x);

(%o27) $[x=-1, x=1]$

Typing the dollar sign ($) after the command line suppress the
output display.

(%i28) solve(x^2=1,x)$

You can assign the output from a command to a variable name.

(%i29) soln: solve(x^2=1,x);

(%o29) $[x=-1, x=1]$

To access an individual solution, use a list-element selector
after the name of the variable to which you assigned the output.

(%i30) soln[1];

(%o30) $x=-1$

```
(%i31) soln[2];
(%o31) x=1
```

Alternatively, you can assign the individual solutions to variable names up front.

```
(%i32) firstsoln: solve(x^2=1,x)[1];
(%o32) x=-1
```

```
(%i33) secondsoln: solve(x^2=1,x)[2];
(%o33) x=1
```

This latter approach is not as efficient as the one preceding it because it requires solving the equation twice, whereas the first approach only solves it once.

Some commands give a function as output. For example, the following command solves the differential equation d[y(x)]/dx=sin(x) with the initial condition y(0)= 0.

```
(%i34) kill(all);
(%o0) done
```

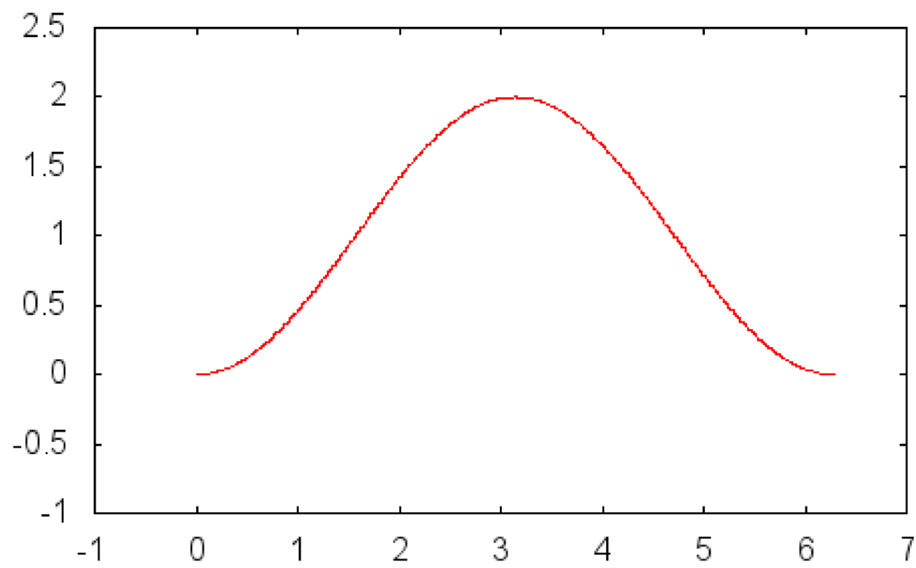
```
(%i1) reset();
(%o1) [lispdisp,labels,_,%,multiplicities,__,piece]
```

```
(%i2) soln: ic1(ode2('diff(y,x)=sin(x), y, x),x=0,y=0);
(%o2) y=1-cos(x)
```

To use the function y in other commands, such as draw2d( ), for example, we form a Maxima function from the output of the ic1( ) command as follows.

```
(%i3) wxdraw2d(
        xrange=[-1,7],
        yrange=[-1,2.5],
        color=red,
        implicit(soln,x,0,2*%pi,y,0,2));
```

(%t3)



(%o3)