

Maxima by Example:

Ch. 2, Two Dimensional Plots and Least Squares Fits *

Edwin L. Woollett

July 13, 2009

Contents

2.1	Introduction to plot2d	3
2.1.1	First Steps with plot2d	3
2.1.2	Parametric Plots	6
2.1.3	Line Width and Color Controls	9
2.1.4	Discrete Data Plots: Point Size, Color, and Type Control	12
2.1.5	More gnuplot_preamble Options	15
2.1.6	Using qplot for Quick Plots of One or More Functions	16
2.2	Least Squares Fit to Experimental Data	18
2.2.1	Maxima and Least Squares Fits: lsquares_estimates	18
2.2.2	Syntax of lsquares_estimates	19
2.2.3	Coffee Cooling Model	20
2.2.4	Experiment Data: file_search , printfile , read_nested_list , and makelist	21
2.2.5	Least Squares Fit of Coffee Cooling Data	22

*This version uses **Maxima 5.18.1**. Check <http://www.csulb.edu/~woollett/> for the latest version of these notes. Send comments and suggestions to woollett@charter.net

COPYING AND DISTRIBUTION POLICY

This document is part of a series of notes titled "Maxima by Example" and is made available via the author's webpage <http://www.csulb.edu/~woollett/> to aid new users of the Maxima computer algebra system.

NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.

You may make copies of this document and distribute them to others as long as you charge no more than the costs of printing.

These notes (with some modifications) will be published in book form eventually via Lulu.com in an arrangement which will continue to allow unlimited free download of the pdf files as well as the option of ordering a low cost paperbound version of these notes.

Feedback from readers is the best way for this series of notes to become more helpful to new users of Maxima. *All* comments and suggestions for improvements will be appreciated and carefully considered.

LOADING FILES

The defaults allow you to use the brief version `load(fft)` to load in the Maxima file `fft.lisp`.

To load in your own file, such as `qxxx.mac` using the brief version `load(qxxx)`, you either need to place `qxxx.mac` in one of the folders Maxima searches by default, or else put a line like:

```
file_search_maxima : append(["c:/work2/###.{mac,mc}"],file_search_maxima )$
```

in your personal startup file `maxima-init.mac` (see Ch. 1, Introduction to Maxima for more information about this).

Otherwise you need to provide a complete path in double quotes, as in `load("c:/work2/qxxx.mac")`,

We always use the brief `load` version in our examples, which are generated using the XMaxima graphics interface on a Windows XP computer, and copied into a fancy verbatim environment in a latex file which uses the `fancyvrb` and `color` packages.

Maxima.sourceforge.net. Maxima, a Computer Algebra System. Version 5.18.1
(2009). <http://maxima.sourceforge.net/>

2.1 Introduction to plot2d

You should be able to use any of our examples with either **wxMaxima** or **Xmaxima**. If you substitute the word **wxplot2d** for the word **plot2d** you should get the same plot (using **wxMaxima**), but the plot will be drawn "inline" in your notebook rather than in a separate window.

To save a plot as an image file, using **wxMaxima**, right click on the inline plot, choose a name and a destination folder, and click ok.

To save a plot drawn in a separate **gnuplot** window, right click the small icon in the upper left hand corner of the plot window, choose Options, Copy to Clipboard, and then open any utility which can open a picture file and select Edit, Paste, and then File, Save As. A standard utility which comes with Windows XP is the accessory Paint, which will work fine in this role to save the clipboard image file. The freely available Infanview is a combination picture viewer and editor which can also be used for this purpose. Saving the image via the **gnuplot** window route results in a larger image.

2.1.1 First Steps with plot2d

The syntax of **plot2d** is

```
plot2d( object-list, draw-parameter-list, other-option-lists ).
```

The required object list (the first item) **may** be simply one object (not a list). The object types may be expressions (or functions), all depending on the same draw parameter, discrete data objects, and parametric objects. If at least one of the plot objects involves a draw parameter, say **p**, then a draw parameter range list of the form [**p**, **pmin**, **pmax**] should follow the object list.

We start with the simplest version which only controls how much of the expression to plot, and does not try to control the canvas width or height.

```
(%i1) plot2d ( sin(u), [u, 0, %pi/2] )$
```

which produces the plot:

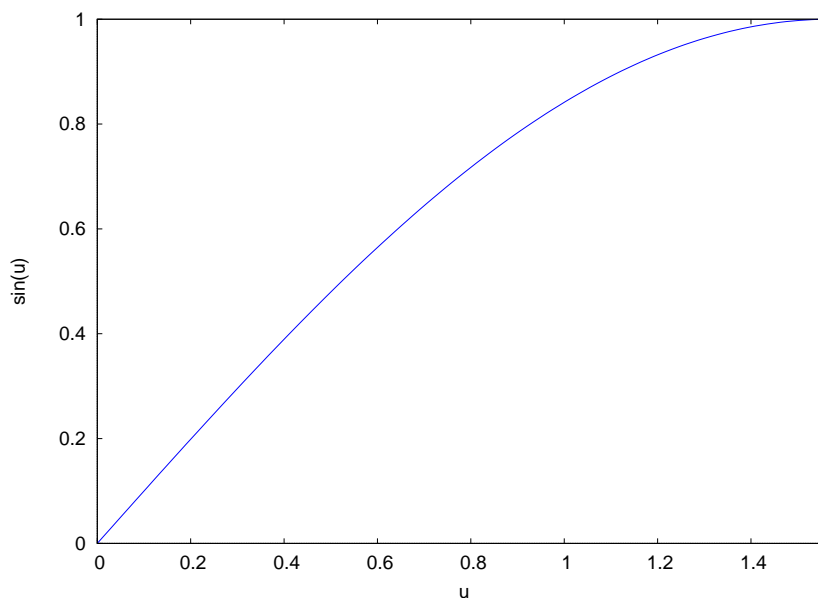


Figure 1: plot2d (sin(u), [u, 0, %pi/2])

We see that **plot2d** has made the canvas width only as wide as the drawing width, and has made the canvas height only as high as the drawing height. Now let's add a horizontal range (canvas width) control list in the form **[x, -0.2, 1.8]**. Notice the special role the symbol **x** plays here in **plot2d**.

```
(%i2) plot2d ( sin(u), [u,0,%pi/2], [x, -0.2, 1.8] )$
```

which produces

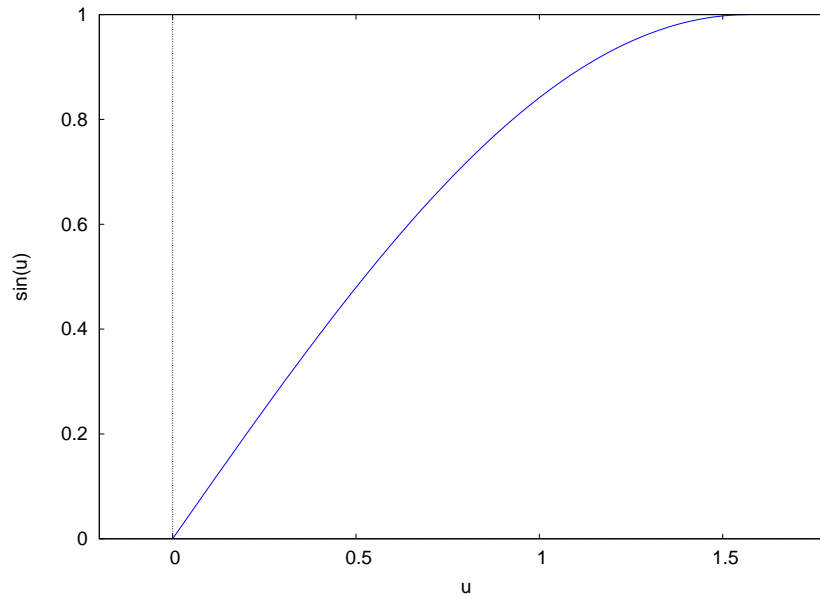


Figure 2: `plot2d (sin(u), [u, 0, %pi/2], [x,-0.2,1.8])`

We see that we now have separate draw width and canvas width controls included. If we try to put the canvas width control list before the draw width control list, we get an error message:

```
(%i3) plot2d(sin(u), [x,-0.2,1.8], [u,0,%pi/2] )$
Unknown plot option specified: u
-- an error. To debug this try debugmode(true);
```

However, if the expression variable **happens** to be **x**, the following command includes both draw width and canvas width using separate **x** symbol control lists.

```
(%i4) plot2d ( sin(x), [x,0,%pi/2], [x,-0.2,1.8] )$
```

in which the first (required) **x** control list determines the drawing range, and the second (optional) **x** control list determines the canvas width.

Despite the special role the symbol **y** also plays in **plot2d**, the following command produces the same plot as above.

```
(%i5) plot2d ( sin(y), [y,0,%pi/2], [x,-0.2,1.8] )$
```

The optional vertical canvas height control list uses the special symbol **y**, as shown in

```
(%i6) plot2d ( sin(u), [u,0,%pi/2], [x,-0.2,1.8], [y,-0.2, 1.2] )$
```

which produces

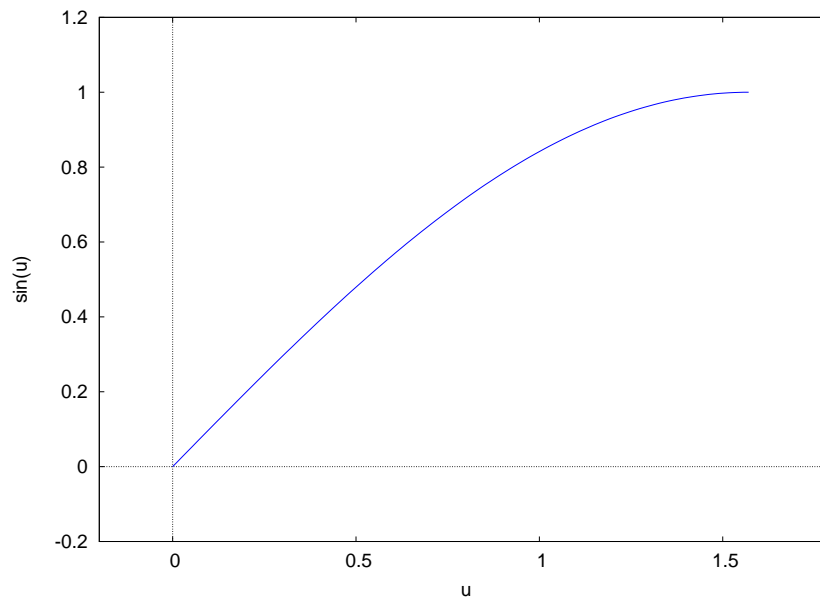


Figure 3: `plot2d (sin(u), [u,0,%pi/2], [x,-0.2,1.8], [y,-0.2, 1.2])`

and the following alternatives produce exactly the same plot.

```
(%i7) plot2d ( sin(u), [u,0,%pi/2], [y,-0.2, 1.2], [x,-0.2,1.8] )$
(%i8) plot2d ( sin(x), [x,0,%pi/2], [x,-0.2,1.8], [y,-0.2, 1.2] )$
(%i9) plot2d ( sin(y), [y,0,%pi/2], [x,-0.2,1.8], [y,-0.2, 1.2] )$
```

2.1.2 Parametric Plots

For orientation, we will draw a sine curve using the parametric plot object syntax and using a parametric parameter t .

```
(%i1) plot2d ( [parametric, t, sin(t), [t, 0, %pi] ] )$
```

which produces

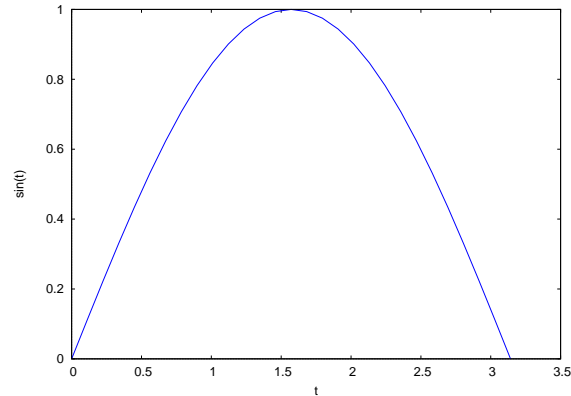


Figure 4: `plot2d ([parametric, t, sin(t), [t, 0, %pi]])`

We see that

```
plot2d ( [ parametric, fx(t), fy(t), [ t, tmin, tmax ] ] )$
```

plots pairs of points $(fx(ta), fy(ta))$ for ta in the interval $[tmin, tmax]$. We have used no canvas width control list $[x, xmin, xmax]$ in this minimal version.

We next use a parametric plot to create a "circle", letting $fx(t) = \cos(t)$ and $fy(t) = \sin(t)$, and again adding no canvas width or height control list.

```
(%i2) plot2d ([parametric, cos(t), sin(t), [t, -%pi, %pi]])$
```

This produces

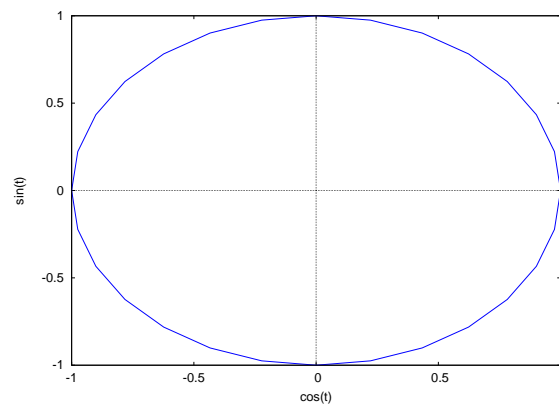


Figure 5: `plot2d ([parametric, cos(t), sin(t), [t, -%pi, %pi]])`

Now let's add a canvas width control list:

```
(%i3) plot2d ([parametric, cos(t), sin(t), [t, -%pi, %pi] ], [x, -4/3, 4/3])$
```

which produces a "rounder" circle:

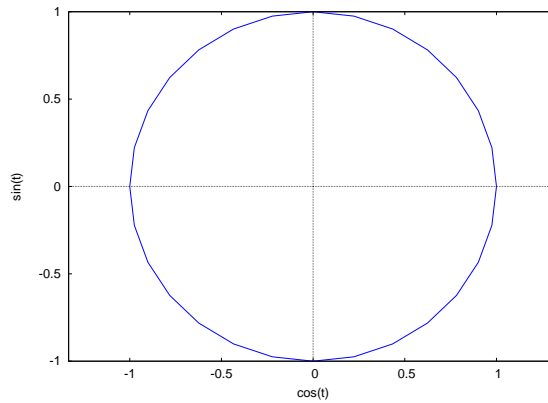


Figure 6: As above, but adding option `[x, -4/3, 4/3]`

The following versions produce precisely the same parametric plot:

```
(%i4) plot2d ([parametric, cos(x), sin(x), [x, -%pi, %pi]],  
             [x, -4/3, 4/3])$  
(%i5) plot2d ([parametric, cos(y), sin(y), [y, -%pi, %pi]],  
             [x, -4/3, 4/3])$
```

An alternative method of producing a "round" circle with a parametric plot is to make use of the **gnuplot_preamble** option in the form:

```
(%i6) plot2d ([parametric, cos(t), sin(t), [t, -%pi, %pi]],  
             [x, -1.2, 1.2], [y, -1.2, 1.2],  
             [gnuplot_preamble, "set size ratio 1;"])$
```

which produces

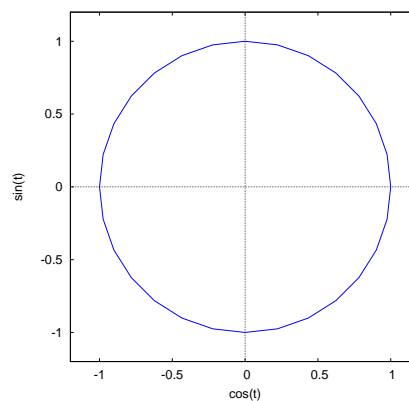


Figure 7: Using "set size ratio 1"

We now make a plot consisting of two plot objects, the first being the explicit expression u^3 , and the second being the parametric plot object used above. We now need the syntax

```
plot2d ([plot-object-1, plot-object-2], possibly-required-draw-range-control,
        other-option-lists )
```

Here is an example:

```
(%i7) plot2d (
      [ u^3,
        [parametric, cos(t), sin(t), [t, -%pi, %pi]],
        [u, -0.8, 0.8], [x, -4/3, 4/3] )$
```

in which $[u, -0.8, 0.8]$ is required to determine the drawing range of u^3 , and this produces

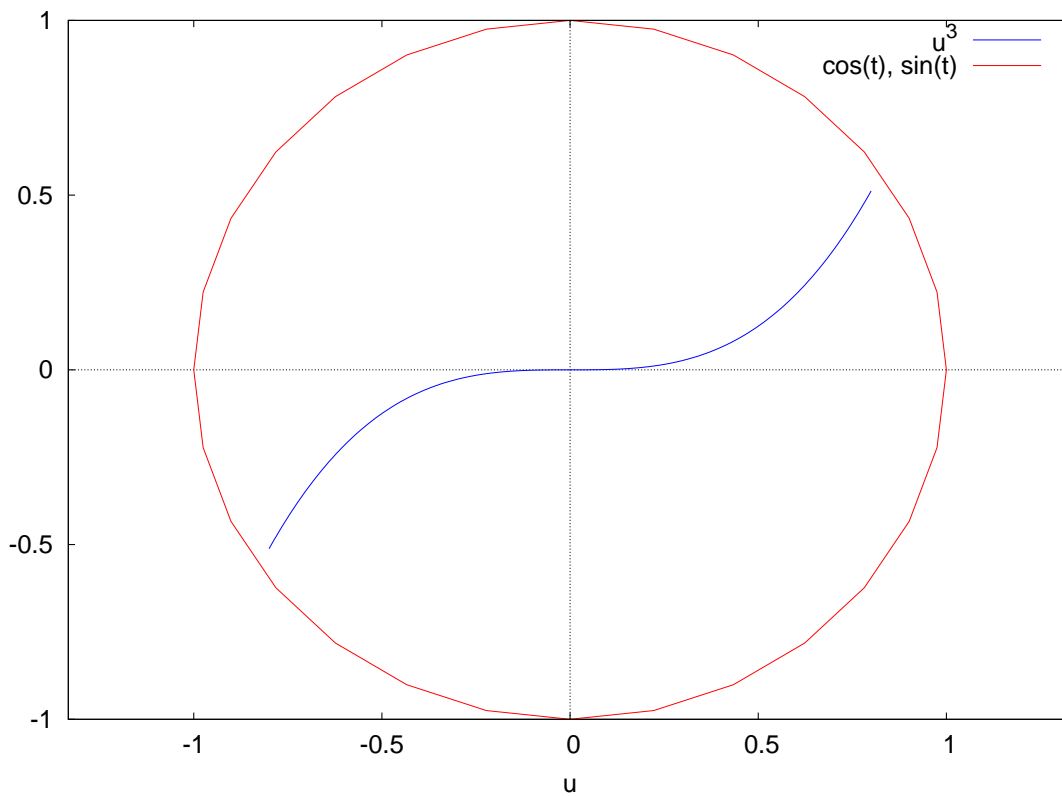


Figure 8: Combining an Explicit Expression with a Parametric Object

in which the horizontal axis label (by default) is u .

We now add a few more options to make this combined plot look a little better (more about some of these later).

```
(%i8) plot2d (
      [ [parametric, cos(t), sin(t), [t,-%pi,%pi]],u^3],
        [u,-1,1], [x,-1.2,1.2], [y,-1.2,1.2], [nticks,200],
        [style, [lines,8]], [xlabel,""], [ylabel,""],
        [box,false], [axes, false],
        [legend,false], [gnuplot_preamble,"set size ratio 1;"])$
```

which produces

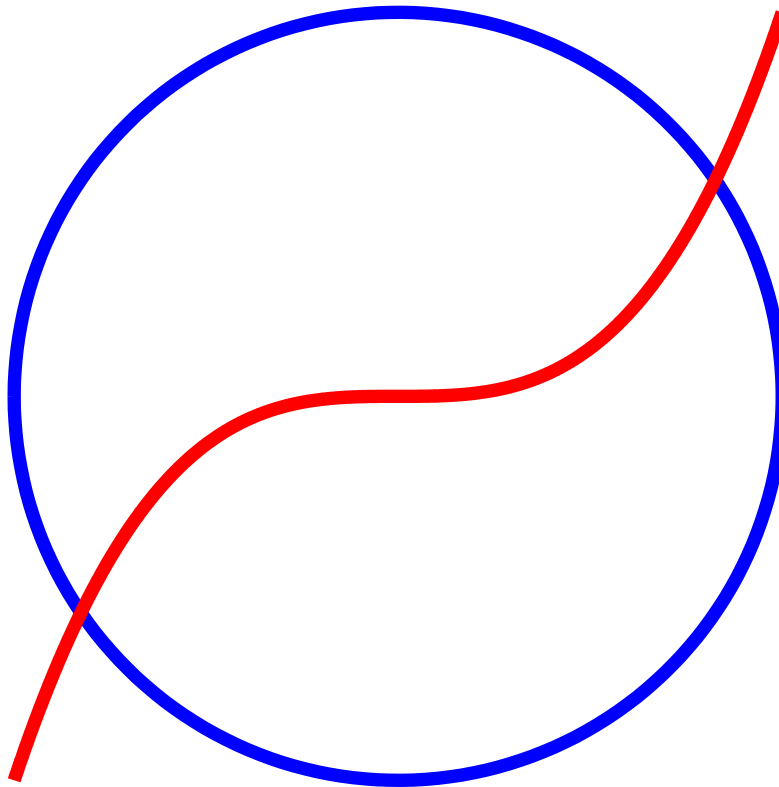


Figure 9: Drawing of u^3 Over a Circle

The default value of **nticks** inside **plot2d** is **29**, and using **[nticks, 200]** yields a much smoother parametric curve.

2.1.3 Line Width and Color Controls

Each element to be included in the plot can have a separate **[lines, nlw, nlc]** entry in the **style** option list, with **nlw** determining the line width and **nlc** determining the line color. The default value of **nlw** is **1**, a very thin weak line. The use of **nlw = 5** creates a strong wider line.

The default values of **nlc** consist of a rotating color scheme which starts with **nlc = 1** (blue) and then progresses through **nlc = 7** (black) and then repeats.

You will see the colors with the associated values of **nlc**, using the following code which draws a set of vertical lines in various colors. This code also shows an example of using **discrete** list objects, and the use of various options available.

```
(%i1) plot2d(
  [ [discrete, [[-4,-5], [-4,5]]], [discrete, [[-3,-5], [-3,5]]],
    [discrete, [[-2,-5], [-2,5]]], [discrete, [[-1,-5], [-1,5]]],
    [discrete, [[0,-5], [0,5]]], [discrete, [[1,-5], [1,5]]],
    [discrete, [[2,-5], [2,5]]], [discrete, [[3,-5], [3,5]]] ],

  [style, [lines,10,0], [lines,10,1], [lines,10,2],
    [lines,10,3], [lines,10,4], [lines,10,5], [lines,10,6],
    [lines,10,7]],

  [x,-5,5], [y,-7,7],
  [legend,"0", "1", "2", "3", "4", "5", "6", "7"],
  [xlabel," "], [ylabel," "],
  [box,false], [axes,false],
  [gnuplot_preamble, "set key bottom"] )$
```

Note that none of the objects being drawn are expressions or functions, so a draw parameter range list is not only not necessary but would make no sense, and that the optional width control list is `[x, -5, 5]`.

The `plot2d` colors are: **0 = black, 1 = blue, 2 = red, 3 = violet, 4 = dark green, 5 = dark brown, 6 = green, 7 = black, 8 = blue, ...**

We cannot use `plot2d` to produce an eps figure which will show the true `plot2d` colors since `plot2d` access to postscript eps file colors are: **0 = light blue, 1 = blue, 2 = red, 3 = violet, 4 = black, 5 = yellow, 6 = green, 7 = light blue.**

Thus the `plot2d eps` color set replaces dark green and dark brown with yellow and light blue (but with different numbers too).

The following color bar plot, which was produced using `qdraw` (see Ch. 5) shows roughly the color bar chart you will see using the usual interactive `plot2d` mode.

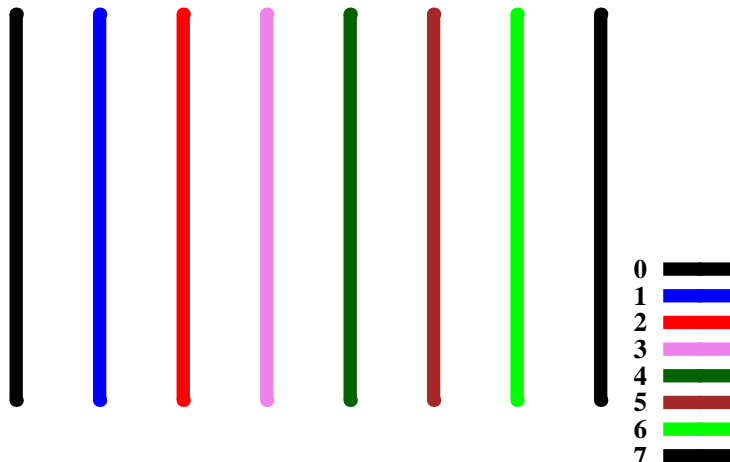


Figure 10: plot2d Color Numbers

For a simple example which uses color and line width controls, we plot the expressions u^2 and u^3 on the same canvas, using lines in black and red colors, and add a height control list, which has the syntax `[y, ymin, ymax]`.

```
(%i2) plot2d( [u^2,u^3],[u,0,2], [x, -.2, 2.5],
             [style, [lines,5,7],[lines,5,2]],
             [y,-1,4] )$
plot2d: expression evaluates to non-numeric value somewhere in plotting range.
```

A bug (v. 5.18.1) in `plot2d` incorrectly treats numbers outside the vertical range which has been set with `[y, -1, 4]` as if they were non-numeric objects.

The width and height control list parameters have been chosen to make it easy to see where the two curves cross for positive u . If you move the cursor over the crossing point, you can read off the coordinates from the cursor position printout in the lower left corner of the plot window.

This produces the plot:

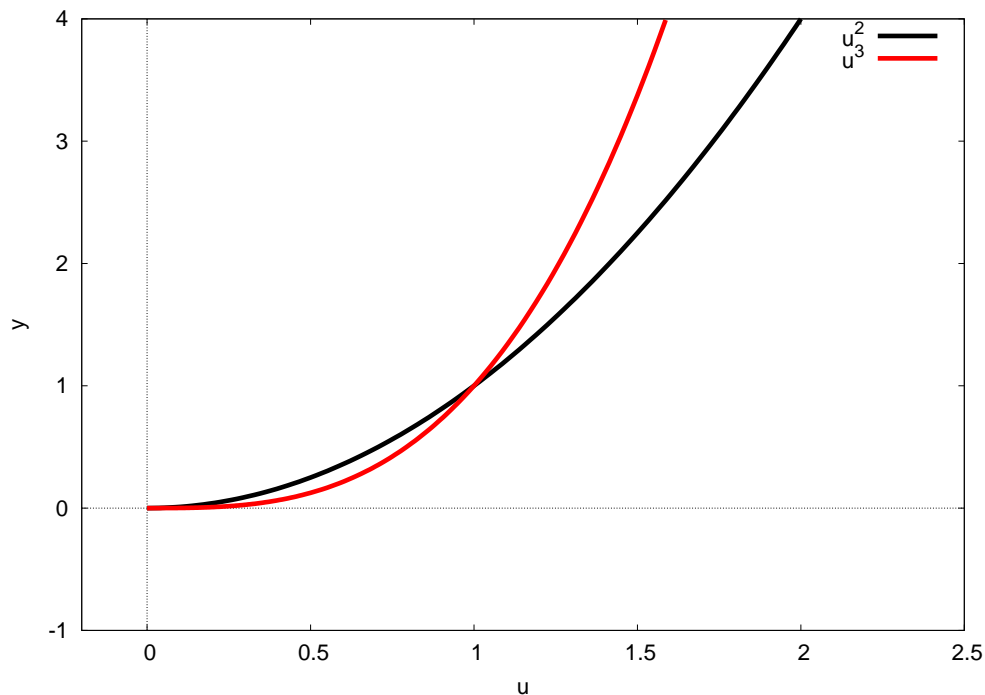


Figure 11: Black and Red Curves

This figure was included in this chapter by creating an eps file version (for inclusion in a latex file) using the code

```
(%i3) plot2d( [u^2, u^3],[u,0,2], [x, -.2, 2.5],
             [style, [lines,10,4],[lines,10,2]],
             [y,-1,4], [psfile, "ch2p11.eps"] )$
```

This eps file version creates the key legend u^2 instead of the `plot2d` console window version which has `u^2` as the legend.

Quoted Symbol Prevents Serious Error

Since **plot2d** has a special role for the symbols **x** and **y**, used in the optional width and height control lists, there is the danger that you might have assigned an expression or value to the symbol **y** (say) in your previous work, and since **plot2d** evaluates its arguments, you will get an error. Here is an example:

```
(%i1) y : x^2$
(%i2) plot2d ( u^3, [u,-1,1], [x,-2,2], [y,-10,10])$

[x^2,-10,10] is not a plot option. Must be [symbol,..data]
-- an error. To debug this try debugmode(true);
(%i3) plot2d ( u^3, [u,-1,1], [x,-2,2], ['y,-10,10])$
(%i4) 'y;
(%o4)                                     y
(%i5) y;
                                     2
(%o5)                                     x
(%i6) ev(y);
                                     2
(%o6)                                     x
```

By using the single quoted symbol '**y**' in the canvas height control list, you are preventing **plot2d** from "evaluating" the symbol and assigning the first slot of that list to the expression x^2 . You can use the single quote ' for the first slot of both control lists:

```
(%i7) plot2d ( u^3, [u,-1,1], ['x,-2,2], ['y,-10,10])$
```

and the plot appears with no problems.

2.1.4 Discrete Data Plots: Point Size, Color, and Type Control

We have seen some simple parametric plot examples above. Here we make a more elaborate plot which includes discrete data points which locate on the curve special places, with information on the key legend about those special points. We force large size points with special color choices, using the maximum amount of control in the `[points, nsize, ncolor, ntype]` style assignments.

```
(%i1) obj_list : [ [parametric, 2*cos(t), t^2, [t,0,2*pi]],
                  [discrete, [[2,0]], [discrete, [[0, (%pi/2)^2]],
                  [discrete, [[-2,%pi^2]], [discrete, [[0, (3*pi/2)^2]]] ]$
(%i2) style_list : [style, [lines,4,7], [points,5,1,1], [points,5,2,1],
                  [points,5,6,1], [points,5,3,1]]$
(%i3) legend_list : [legend, " ", "t = 0", "t = pi/2", "t = pi",
                    " t = 3*pi/2"]$
(%i4) plot2d( obj_list, [x,-3,4], [y,-1,40], style_list,
              [xlabel, "X = 2 cos( t ), Y = t ^2 "],
              [ylabel, " "] , legend_list )$
```

This produces the plot:

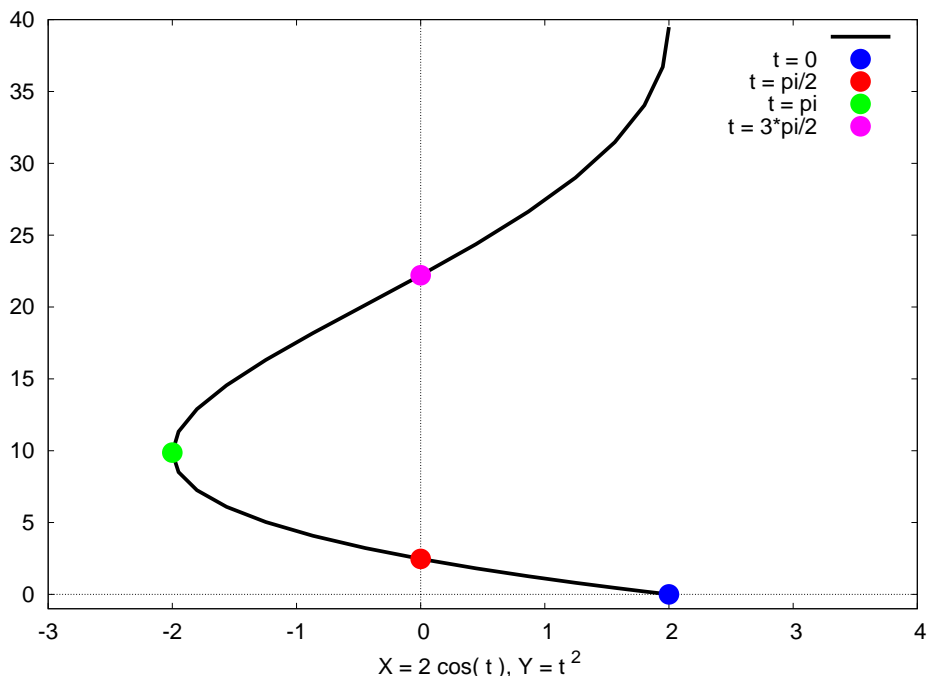


Figure 12: Parametric Plot with Discrete Points

The **points** style option has any of the following forms: `[points]`, or `[points, point_size]`, or `[points, point_size, point_color]`, or `[points, point_size, point_color, point_type]`, in order of increasing control. The default **point size** is given by the integer **1** which is small. The default **point colors** are the same cyclic colors used by the **lines** style. The default **point type** is a cyclic order starting with **2 = open circle**, then continuing **3 = plus sign**, **4 = diagonal crossed lines as capital X**, **5 = star**, **6 = filled square**, **7 = open square**, **8 = filled triangle point up**, **9 = open triangle point up**, **10 = filled triangle point down**, **11 = open triangle point down**, **12 = filled diamond**, **13 = open diamond = 0**, **14 = filled circle**, which is the same as **1**. Thus if you use `[points, 5]` you will get good size points, and both the color and shape will cycle through the default order. If you use `[points, 5, 2]` you will force a red color but the shape will depend on the contents and order of the rest of the objects list.

Next we combine a list of twelve (x,y) pairs of points with the key word **discrete** to form a discrete object type for **plot2d**, and then look at the data points without adding the optional canvas width control.

```
(%i5) data_list : [discrete,
  [ [1.1, -0.9], [1.45, -1], [1.56, 0.3], [1.88, 2],
    [1.98, 3.67], [2.32, 2.6], [2.58, 1.14],
    [2.74, -1.5], [3, -0.8], [3.3, 1.1],
    [3.65, 0.8], [3.72, -2.9] ] ]$
(%i6) plot2d( data_list, [style, [points]])$
```

This produces the plot

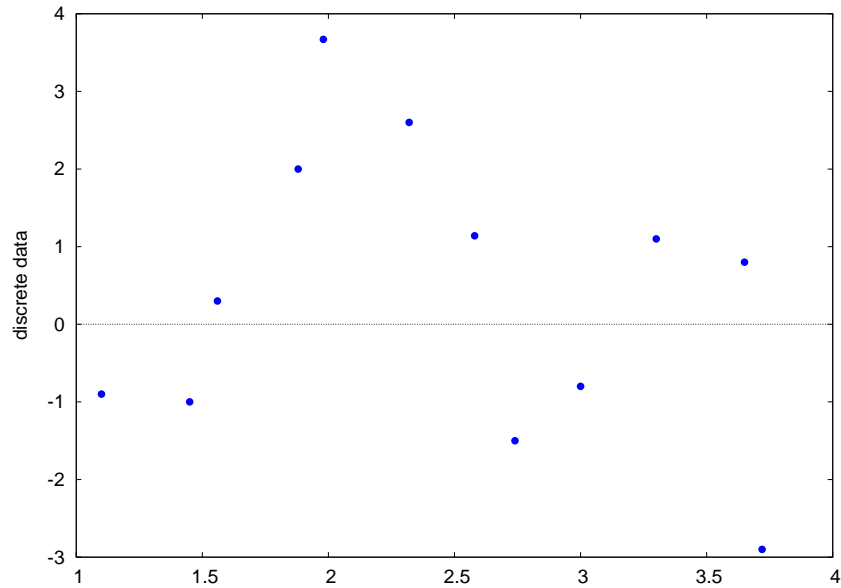


Figure 13: Twelve Data Points

We now combine the data points with a curve which is a possible fit to these data points over the draw parameter range $[u, 1, 4]$.

```
(%i7) plot2d( [sin(u)*cos(3*u)*u^2, data_list],
             [u,1,4], [x,0,5], [y,-10,8],
             [style,[lines,4,1],[points,4,2,1]])$
plot2d: expression evaluates to non-numeric value somewhere in plotting range.
```

which produces the plot

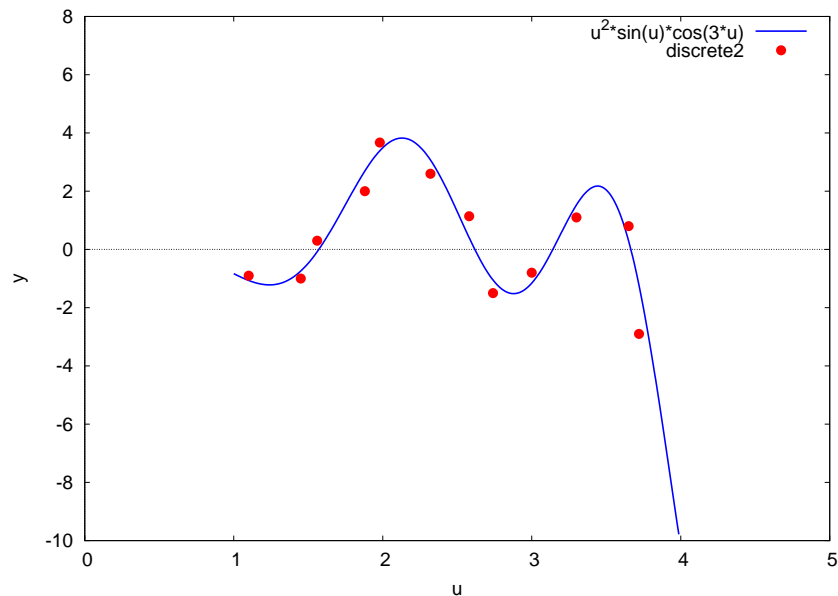


Figure 14: Curve Plus Data

2.1.5 More gnuplot_preamble Options

Here is an example of using the **gnuplot_preamble** options to add a grid, a title, and position the plot key at the bottom center of the canvas. Note the use of a semi-colon between successive gnuplot instructions.

```
(%i1) plot2d([ u*sin(u), cos(u) ], [u,-4,4] , [x,-8,8],
             [style,[lines,5]],
             [gnuplot_preamble,"set grid; set key bottom center;
             set title 'Two Functions';"])$
```

which produces the plot

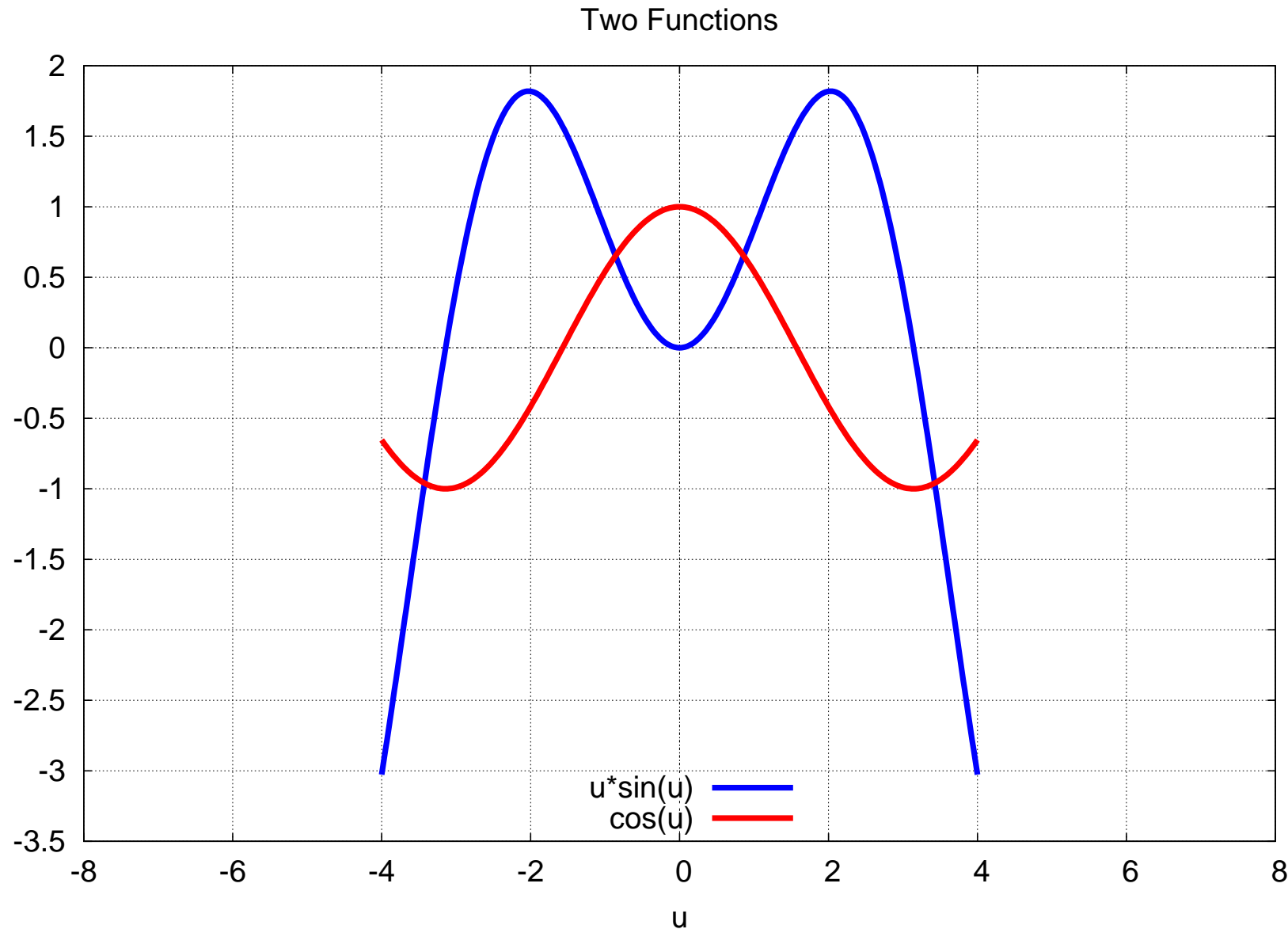


Figure 15: Using the gnuplot_preamble Option

Another option we have previously used is **set zeroaxis lw 2;** to get more prominent **x** and **y** axes. Another example of key location would be **set key top left;**. We have also previously used **set size ratio 1;** to get a "rounder" circle.

2.1.6 Using `qplot` for Quick Plots of One or More Functions

The file `mbe1util.mac` is posted with Ch. 1. This "utility file" contains a function called `qplot` which can be used for quick plotting of functions in place of `plot2d`.

The function `qplot` (`q` for "quick") accepts the default cyclic colors but always uses thicker lines than the `plot2d` default, adds more prominent x and y axes to the plot, and adds a grid. Here are some examples of use. (We include use with `discrete` lists only for completeness, since there is no way to get the `points` style with `qplot`.)

```
(%i1) load(mbe1util);
(%o1)          c:/work2/mbe1util.mac
(%i2) qplot(sin(u), [u, -%pi, %pi])$
(%i3) qplot(sin(u), [u, -%pi, %pi], [x, -4, 4])$
(%i4) qplot(sin(u), [u, -%pi, %pi], [x, -4, 4], [y, -1.2, 1.2])$
(%i5) qplot([sin(u), cos(u)], [u, -%pi, %pi])$
(%i6) qplot([sin(u), cos(u)], [u, -%pi, %pi], [x, -4, 4])$
(%i7) qplot([sin(u), cos(u)], [u, -%pi, %pi], [x, -4, 4], [y, -1.2, 1.2])$
(%i8) qplot([parametric, cos(t), sin(t), [t, -%pi, %pi]],
            [x, -2.1, 2.1], [y, -1.5, 1.5])$
```

The last use involved only a parametric object, and the "prange" list is interpreted as a width control list based on the symbol `x`. The next example includes both an expression depending on the parameter `u` and a parametric object, so we must have a draw parameter control list.

```
(%i9) qplot ([ u^3,
              [parametric, cos(t), sin(t), [t, -%pi, %pi]]],
            [u, -1, 1], [x, -2.1, 2.1], [y, -1.5, 1.5])$
```

We have used `qdraw` (Ch.5) to produce an eps figure we can use here which roughly represents what you get from that last example:

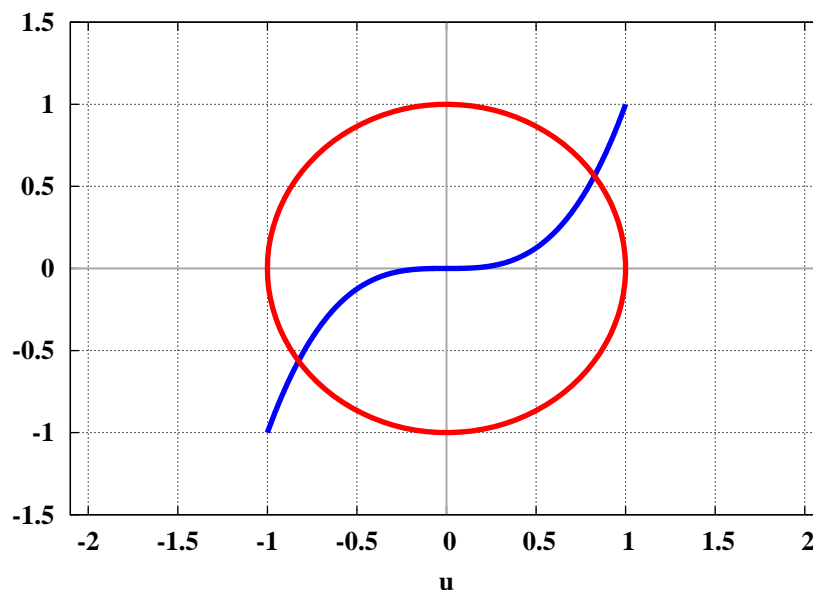


Figure 16: `qplot` example

Here are two **discrete** examples which draw vertical lines.

```
(%i10) qplot([discrete, [[0,-2], [0,2]]], [x,-2,2], [y,-4,4])$
(%i11) qplot( [ [discrete, [[-1,-2], [-1,2]]],
               [discrete, [[1,-2], [1,2]]]], [x,-2,2], [y,-4,4])$
```

Here is the code (in `mbe1util.mac`) which defines the Maxima function **qplot**.

```
qplot ( exprlist, prange, [hvrangle] ) :=
  block([optlist, plist],
    optlist : [ [nticks,100], [legend, false],
               [ylabel, " "], [gnuplot_preamble, "set grid; set zeroaxis lw 2;"] ],
    optlist : cons ( [style,[lines,5]], optlist ),
    if length (hvrangle) = 0 then plist : []
      else plist : hvrangle,
    plist : cons (prange,plist),
    plist : cons (exprlist,plist),
    plist : append ( plist, optlist ),
    apply (plot2d, plist ) )$
```

In this code, the third argument is an optional argument. The local **plist** accumulates the arguments to be passed to **plot2d** by use of **cons** and **append**, and is then passed to **plot2d** by the use of **apply**. The order of using **cons** makes sure that **exprlist** will be the first element, (and **prange** will be the second) seen by **plot2d**. In this example you can see several tools used for programming with lists.

Several choices have been made in the **qplot** code to get quick and uncluttered plots of one or more functions. One choice was to add a grid and stronger **x** and **y** axis lines. Another choice was to eliminate the key legend by using the option **[legend, false]**. If you want a key legend to appear when plotting multiple functions, you should remove that option from the code and reload **mbe1util.mac**.

2.2 Least Squares Fit to Experimental Data

2.2.1 Maxima and Least Squares Fits: lsquares_estimates

Suppose we are given a list of $[x, y]$ pairs which are thought to be roughly described by the relation $y = a \cdot x^b + c$, where the three parameters are all of order 1. We can use the data of $[x, y]$ pairs to find the "best" values of the unknown parameters $[a, b, c]$, such that the data is described by the equation $y = a \cdot x^b + c$ (a three parameter fit to the data).

We are using one of the Manual examples for `lsquares_estimates`.

```
(%i1) dataL : [[1, 1], [2, 7/4], [3, 11/4], [4, 13/4]];

(%o1)          7      11      13
      [[1, 1], [2, -], [3, --], [4, --]]
          4      4      4

(%i2) dataM : apply ('matrix, dataL);

          [ 1  1 ]
          [      ]
          [  7  ]
          [ 2  - ]
          [   4 ]
          [      ]
(%o2)     [   11 ]
          [ 3  -- ]
          [   4  ]
          [      ]
          [   13 ]
          [ 4  -- ]
          [   4  ]

(%i3) load (lsquares)$
(%i4) fpprintprec:8$
(%i5) lsquares_estimates (dataM, [x,y], y=a*x^b+c,
      [a,b,c], initial=[3,3,3], iprint=[-1,0] );
(%o5)      [a = 1.3873659, b = 0.711096, c = - 0.414271]]
(%i6) fit : a*x^b + c , % ;
          0.711096
(%o6)      1.3873659 x      - 0.414271
```

Note that we must use `load (lsquares)`; to use this method. We can now make a plot of both the discrete data points and the least squares fit to those four data points.

```
(%i7) plot2d ([fit, [discrete, dataL]], [x, 0, 5],
      [style, [lines, 5], [points, 4, 2, 1]],
      [legend, "fit", "data"],
      [gnuplot_preamble, "set key bottom;"])$
```

which produces the plot

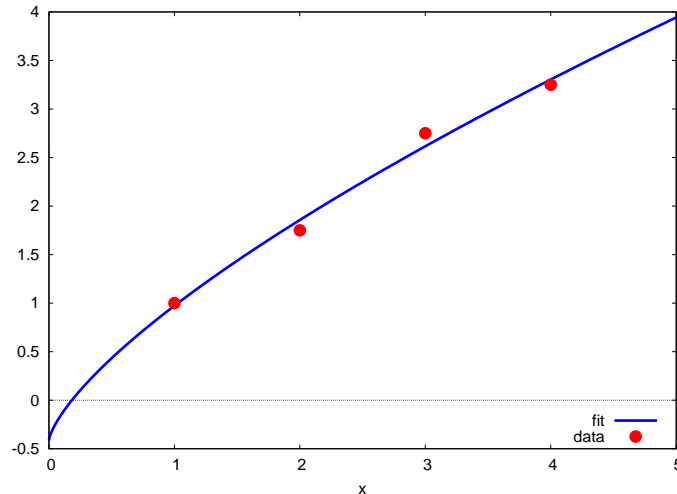


Figure 17: Three Parameter Fit to Four Data Points

2.2.2 Syntax of `lsquares_estimates`

The **minimal** syntax is

```
lsquares_estimates (data-matrix, data-variable-list, fit-eqn, param-list );
```

in which the **data-variable-list** assigns a variable name to the corresponding column of the **data-matrix**, and the **fit-eqn** is an equation which is a relation among the data variable symbols and the equation parameters which appear in **param-list**. The function returns the "best fit" values of the equation parameters in the form `[[p1 = p1val, p2 = p2val, ...]]`.

In the example above, the data variable list was `[x, y]` and the parameter list was `[a, b, c]`.

If an exact solution cannot be found, a numerical approximation is attempted using **lbfgs**, in which case, all the elements of the data matrix should be "numbers" `numberp(x) -> true`. This means that `%pi` and `%e`, for example, should be converted to explicit numbers before use of this method.

```
(%i1) expr : 2*%pi + 3*exp(-4);
(%o1)                - 4
                2 %pi + 3 %e
(%i2) listconstvars:true$
(%i3) listofvars(expr);
(%o3)                [%e, %pi]
(%i4) map('numberp,%);
(%o4)                [false, false]
(%i5) fullmap('numberp,expr);
(%o5)                true
                false      true + false true
(%i6) float(expr);
(%o6)                6.338132223845789
(%i7) numberp(%);
(%o7)                true
```

Optional arguments to `lsquares_estimates` are (in any order)

`initial = [p10, p20, ...], iprint = [n, m], tol = search-tolerance`

The list `[p10, p20, ...]` is the optional list of initial values of the equation parameters, and without including your own guess for starting values this list defaults (in the code) to `[1, 1, ...]`.

The first integer `n` in the `iprint` list controls how often progress messages are printed to the screen. The default is `n = 1` which causes a new progress message printout each iteration of the search method. Using `n = -1` suppresses all progress messages. Using `n = 5` allows one progress message every five iterations.

The second integer `m` in the `iprint` list controls the verbosity, with `m = 0` giving minimum information and `m = 3` giving maximum information.

The option `iprint = [-1, 0]` will hide the details of the search process.

The default value of the `search-tolerance` is `1e-3`, so by using the option `tol = 1e-8` you might find a more accurate solution.

Many examples of the use of the `lsquares` package are found in the file `lsquares.mac`, which is found in the `...share/contrib` folder. You can also see great examples of efficient programming in the Maxima language in that file.

2.2.3 Coffee Cooling Model

"Newton's law of cooling" (only approximate and not a law) assumes the rate of decrease of temperature (celsius degrees per minute) is proportional to the instantaneous difference between the temperature $\mathbf{T}(t)$ of the coffee in the cup and the surrounding ambient temperature \mathbf{T}_s , the latter being treated as a constant. If we use the symbol r for the "rate constant" of proportionality, we then assume the cooling of the coffee obeys the first order differential equation

$$\frac{d\mathbf{T}}{dt} = -r(\mathbf{T}(t) - \mathbf{T}_s) \quad (2.1)$$

Since \mathbf{T} has dimension degrees Celsius, and t has dimension minute, the dimension of the rate constant r must be `1/min`.

(This attempt to employ a rough mathematical model which can be used for the cooling of a cup of coffee avoids a bottom-up approach to the problem, which would require mathematical descriptions of the four distinct physical mechanisms which contribute to the decrease of thermal energy in the system hot coffee plus cup to the surroundings: thermal radiation (net electromagnetic radiation flux, approximately black body) energy transport across the surface of the liquid and cup, collisional heat conduction due to the presence of the surrounding air molecules, convective energy transport due to local air temperature rise, and finally evaporation which is the escape of the fastest coffee molecules which are able to escape the binding surface forces at the liquid surface. If the temperature difference between the coffee and the ambient surroundings is not too large, experiment shows that the simple relation above is roughly true.)

This differential equation is easy to solve "by hand", since we can write

$$\frac{d\mathbf{T}}{dt} = \frac{d(\mathbf{T} - \mathbf{T}_s)}{dt} = \frac{dy}{dt} \quad (2.2)$$

and then divide both sides by $y = (\mathbf{T} - \mathbf{T}_s)$, multiply both sides by dt , and use $dy/y = d\ln(y)$ and finally integrate both sides over corresponding intervals to get $\ln(y) - \ln(y_0) = \ln(y/y_0) = -rt$, where $y_0 = \mathbf{T}(0) - \mathbf{T}_s$ involves the initial temperature at $t = 0$. Since

$$e^{\ln(A)} = A, \quad (2.3)$$

by equating the exponential of the left side to that of the right side, we get

$$T(t) = T_s + (T(0) - T_s) e^{-rt}. \quad (2.4)$$

Using `ode2`, `ic1`, `expand`, and `collectterms`, we can also use Maxima just for fun:

```
(%i1) de : 'diff(T,t) + r*(T - Ts);
      dT
(%o1)  -- + r (T - Ts)
      dt
(%i2) gsoln : ode2(de,T,t);
      - r t      r t
(%o2)  T = %e      (%e      Ts + %c)
(%i3) de, gsoln, diff, ratsimp;
(%o3)  0
(%i4) ic1 (gsoln, t = 0, T = T0);
      - r t      r t
(%o4)  T = %e      (T0 + (%e      - 1) Ts)
(%i5) expand (%);
      - r t      - r t
(%o5)  T = %e      T0 - %e      Ts + Ts
(%i6) Tcup : collectterms ( rhs(%), exp(-r*t) );
      - r t
(%o6)  %e      (T0 - Ts) + Ts
(%i7) Tcup, t = 0;
(%o7)  T0
```

We arrive at the same solution as found "by hand". We have checked the particular solution for the initial condition and checked that our original differential equation is satisfied by the general solution.

2.2.4 Experiment Data: file_search, printfile, read_nested_list, and makelist

Let's take some "real world" data for this problem (p. 21, An Introduction to Computer Simulation Methods, 2nd ed., Harvey Gould and Jan Tobochnik, Addison-Wesley, 1996) which is in a data file `c:\work2\coffee.dat` on the author's Window's XP computer (data file available with this chapter on the author's webpage).

This file contains three columns of tab separated numbers, column one being the elapsed time in minutes, column two the Celsius temperature of the system glass plus coffee for black coffee, and column three the Celsius temperature for the case glass plus creamed coffee. The glass-coffee temperature was recorded with an estimated accuracy of 0.1°C . The ambient temperature of the surroundings was 17°C .

We need to use double quotes for a file name which includes an extension like `.dat`.

```
(%i1) file_search("coffee.dat");
(%o1)  coffee.dat
```

No path was needed because of the contents of `maxima-init.mac`. We can look at the whole file at one burst with `printfile`. (We will show only the top of the output.)

```
(%i2) printfile("coffee.dat");
0      82.3      68.8
2      78.5      64.8
4      74.3      62.1
6      70.7      59.9
8      67.6      57.7
-----
etc, etc.
-----
```



```
(%i13) lsquares_estimates ( black_matrix, [t,T], black_eqn, [r],
                          iprint = [-1,0] );
(%o13)
      [[r = 0.02592]]
(%i14) black_fit : rhs ( black_eqn ), %;
                          - 0.02592 t
(%o14)
      65.3 %e          + 17
```

Thus **rblack** is roughly 0.026 min^{-1} .

For the white coffee case, **T₀ = 68.8 deg C** and **T_s = 17 deg C**.

```
(%i15) white_eqn : T = 17 + 51.8*exp(-r*t);
                          - r t
(%o15)
      T = 51.8 %e          + 17
(%i16) lsquares_estimates ( white_matrix, [t,T], white_eqn, [r],
                          iprint = [-1,0] );
(%o16)
      [[r = 0.0237]]
(%i17) white_fit : rhs ( white_eqn ), %;
                          - 0.0237 t
(%o17)
      51.8 %e          + 17
```

Thus **rwhite** is roughly 0.024 min^{-1} , a slightly smaller value than for the black coffee (which is reasonable since a black body is a better radiator of thermal energy than a white surface).

A prudent check on mathematical reasonableness can be made by using, say, the two data points for **t = 0** and **t = 24 min** to solve for a rough value of **r**. For this rough check, the author concludes that **rblack** is roughly 0.027 min^{-1} and **rwhite** is roughly 0.024 min^{-1} .

We can now plot the temperature data against the best fit model curve, first for the black coffee case.

```
(%i18) plot2d( [ black_fit , [discrete,black_data] ],
               [t,0,50], [style, [lines,5], [points,2,2,6]],
               [ylabel," "],
               [xlabel," Black Coffee T(deg C) vs. t(min) with r = 0.026/min"],
               [legend,"black fit","black data"] )$
```

which produces the plot

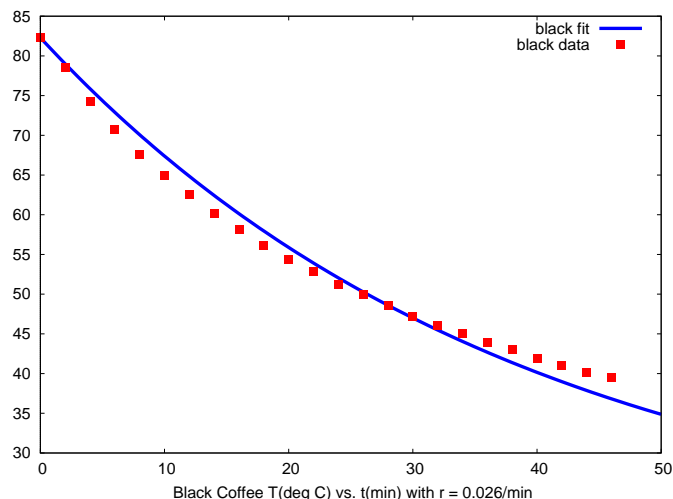


Figure 18: Black Coffee Data and Fit

and next plot the white coffee data and fit:

```
(%i19) plot2d( [ white_fit , [discrete, white_data] ],
               [t,0,50], [style, [lines,5], [points,2,2,6]],
               [ylabel," "],
               [xlabel," White Coffee T(deg C) vs. t(min) with r = 0.024/min"],
               [legend,"white fit","white data"] )$
```

which yields the plot

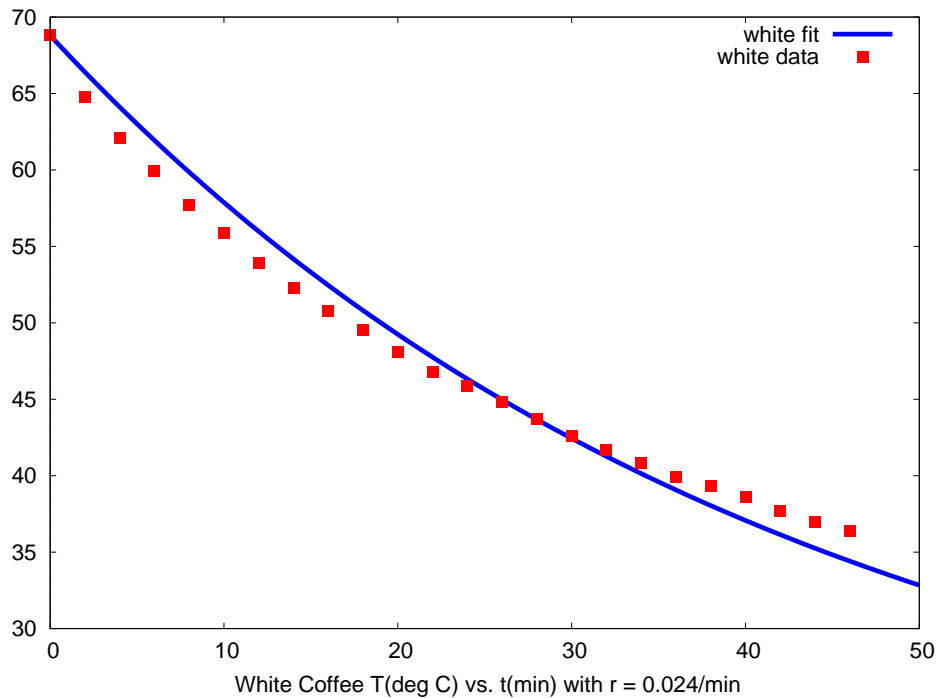


Figure 19: White Coffee Data and Fit

Cream at Start or Later?

Let's use the above approximate values for the cooling rate constants to find the fastest method to use to get the temperature of hot coffee down to a drinkable temperature. Let's assume we start with a glass of very hot coffee, $T_0 = 90^\circ\text{C}$, and want to compare two methods of getting the temperature down to 75°C , which we assume is low enough to start sipping. We will assume that adding cream lowers the temperature of the coffee by 5°C for both options we explore. Option 1 (white option) is to immediately add cream and let the creamed coffee cool down from 85°C to 75°C . We first write down a general expression as a function of T_0 and r , and then substitute values appropriate to the white coffee cooldown.

```
(%i20) T : 17 + (T0 - 17) * exp(-r*t);
               - r t
(%o20)          %e      (T0 - 17) + 17
(%i21) T1 : T, [T0 = 85, r = 0.0237];
               - 0.0237 t
(%o21)          68 %e      + 17
(%i22) t1 : find_root(T1 - 75, t, 2, 10);
(%o22)          6.71159
```

The "white option" requires about **6.7 min** for the coffee to be sippable.

Option 2 (the black option) lets the black coffee cool from 90°C to 80°C , and then adds cream, immediately getting the temperature down from 80°C to 75°C

```
(%i23) T2 : T, [T0 = 90, r = 0.02592];
              - 0.02592 t
(%o23)          73 %e          + 17
(%i24) t2 : find_root(T2 - 80,t,2,10);
(%o24)          5.68382
```

The black option (option 2) is the fastest method to cool the coffee, taking about **5.68 min** which is about **62 sec** less than the white option 1.