Macsyma
Scientific Graphics Reference Manual

Macsyma Scientific Graphics Reference Manual

This document corresponds to Macsyma version 2.4.

The Documentation Staff of Macsyma Inc. prepared this manual.

A report or thesis that contains results obtained by using Macsyma should acknowledge that the work was done with the aid of Macsyma, a large symbolic manipulation program developed at the MIT Laboratory for Computer Science and supported from 1975 to 1983 by the National Aeronautics and Space Administration under grant NSG 1323, by the Office of Naval Research under grant N00014-77-C-0641, by the U.S. Department of Energy under grant ET-78-C-02-4687, and by the U.S. Air Force under grant F49620-79-C-020, from 1982 to 1992 by Symbolics, Inc. of Concord Mass. and since by Macsyma Inc. of Arlington, Mass.

A copy of the report or thesis should also be sent to Macsyma Inc at the address below.

Printed in the USA.
Printing Number and Year: 98 1

Macsyma Inc.
20 Academy Street
Arlington, MA 02476-6436

(781) 646–4550

E-mail:	info@macsyma.com
	service@macsyma.com
Internet:	URL http://www.macsyma.com

Contents

List of Tables

Chapter 1

Plotting and Graphics

This chapter describes how to produce two- and three-dimensional plots in Macsyma. You can produce five basic kinds of plots with Macsyma:

- Plots of functions of one or two independent variables. See Section 1.1.1, page 2 and Section 1.2.1, page 18.
- Plots of parametric curves and surfaces. See Section 1.1.2, page 5 and Section 1.2.2, page 20 and Section 1.2.5, page 21.
- Contour plots and plots of implicit relations. See Section 1.1.3, page 6 and Section 1.2.3, page 20.
- Plots of discrete point sets. See Section 1.1.4, page 9 and Section 1.2.4, page 21.
- Plots of vector fields. See Section 1.1.5, page 11 and Section 1.2.6, page 23.

Macsyma's plotting commands make many default decisions about plots without bothering you to ask, such as choices of viewpoint, plot line colors, surface coloring models, axis label numbers, and so forth. After you become familiar with the basic plotting functions, you may wish to modify or override Macsyma's graphics defaults to get precisely the effects that you want.

Macsyma offers a wide range of controls so you can obtain precisely the publication-quality scientific graphics you want. You can

- Change basic plot geometry of the plotted object itself, such as the number of plot points, the coordinate system used, and superimposition of plots. See Section 1.3, page 24.
- Change the viewpoint, roll angle, scale and perspective. See Section 1.4.1.2, page 35.
- Change the appearance attributes of the plotted object itself, such as line colors, surface colors, lighting and plot point symbols. See Section 1.4.2.1, page 36.
- Change the secondary elements of the plot which surround the main plotted object, such as the plot axes, bounding box and text labels.
- Animate plots, including animating changes of surface location, shape, color, viewpoint, lighting, and other plot attributes.

Basic plot geometry (such as number of plot points and their coordinates) can be determined only at the time when Macsyma generates a plot. Most other plot characteristics can be changed in two basic ways.

- Plot characteristics can be changed programmatically at the time the plot is generated, by changing option variables and optional arguments to the plotting commands which generate the plot.
- Most attributes of a plot can be changed after the plot is generated, using the graphics editing capabilities in the Macsyma Front End. For Macsyma 2.0 and successors, see *Scientific Notebook Interface Reference Manual* for more information.

Some plot utilities are also described.

- Superimposing plots (See Section 1.3.3, page 31)
- Hardcopying plots (For Macsyma 2.0, see Section 1.6.2, page 51)
- Naming and saving plots (For Macsyma 2.0, see Section 1.6.3, page 52)

1.1 Two-Dimensional Graphics

This section discusses the main two-dimensional plotting commands in Macsyma. The main commands for two-dimensional plotting are

- **plot**, Section 1.1.1, page 2
- **paramplot**, Section 1.1.2, page 5
- **contourplot** and **implicit_plot**, Section 1.1.3, page 6
- **graph**, Section 1.1.4, page 9
- **plot2_vect**, Section 1.1.5, page 11

1.1.1 Plotting Functions of One Variable

plot(*y-exps, x-var,x-range,*{*'arg*$_1$*,...,'arg*$_2$}) *Special Form*

Plots *y-exps* in the y direction, while *x-var*, the x-axis, takes on values specified by *x-range*. The inputs $arg_1, \ldots, arg_n$ are optional. The input arguments can take various forms, each described below.

The input *y-exp* can be any of:

y-exps	Action
exp	**plot** plots a curve of *exp* against *x-var*.
[$exp_1, \ldots, exp_n$]	Plots n curves of exp_i against *x-var*. Each exp_i is evaluated in the context: `float(ev(`exp_i`,` *x-var=value from x-range*`, numer))`. An error is signaled if this does not result in a floating-point number.

The input *x-range* can be any of:

x-range	Meaning
low, *high*	where *low* and *high* evaluate to numbers. The value *low* may be either greater or less than *high*. The input *x-var* takes on **plotnum** values equally spaced between *low* and *high*. See Section 1.3.1, page 24. **Note:** The first argument is evaluated at *low* first. For example, `plot(1/x,x,-1,-3);` calculates $1/(-1.0)$ before $1/(-3.0)$. This matters only if the computation of the first argument changes a variable which in turn changes the value returned by subsequent computation. Whether or not *low* < *high*, **min**(*low*, *high*) is displayed on the left side of the plot. This can be overridden using the optional argument **special** with **reflect** as **xfun**. (see Section 1.3.2.2, page 28).
low, *high*, **integer**	This form has the same effect as the above except that *x-var* takes on only integer values between *low* and *high* inclusive.
[val_1, ..., val_n]	*x-var* takes the values specified by the list of values.
arrayname	where *arrayname* is the name of a declared floating-point one-dimensional array. Such an array can be created by means of a command such as `array(`*arrayname*`, float,` *max-index*`);`. In this case, *x-var* takes the values from *arrayname*[0] through *arrayname*[*max-index*]. (Here, *max-index* is the maximum index of *arrayname*.)

The optional arguments arg_1, ..., arg_n can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs arg_1, ..., arg_n can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

An alternative form for **plot** is `plot(`*y-funs*`,` *x-range*`,` arg_1`, ..., `arg_n`);`. Here, *y-funs* must be a function of one argument or a list of functions of one argument. The functions must be either translated or compiled functions, and they must return a floating-point number when given floating-point or integer arguments. This form of **plot** acts as though you had not only given an argument to the *y-funs*, but also specified that argument as the *variable* in the form above. For example, `plot(f,-2,2);` acts like `plot(f(x), x,-2, 2);`. This provides a quicker evaluation of the first argument and for that reason no checking is done on the result. If the wrong sort of number is returned, the plot is not useful.

Examples

The example below Figure 1.1, page 4, illustrates how you plot $\sin x$ against x as x takes on **plotnum** values between $-\pi$ and π.

```
(c1) plot(sin(x), x, -%pi, %pi)$
```

This example Figure 1.2, page 4, illustrates how you plot a list of the first few Fibonacci numbers versus the integer position in the list.

```
(c1) lst:makelist(fib(i), i, 1, 9);
(d1)                  [1, 1, 2, 3, 5, 8, 13, 21, 34]
(c2) plot(lst[i], i, 1, length(lst), integer)$
```

The example Figure 1.3, page 5, illustrates how you plot $f(x)$ as x takes the values in the specified list.

```
(c1) f(x):=sqrt(x+%pi)$
```

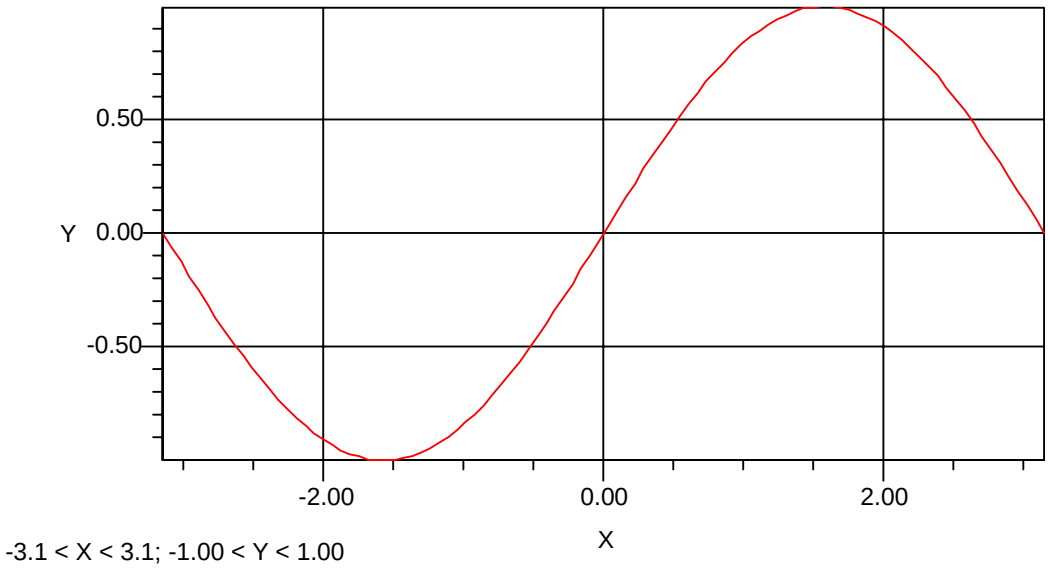

Figure 1.1: Example of the Plot command

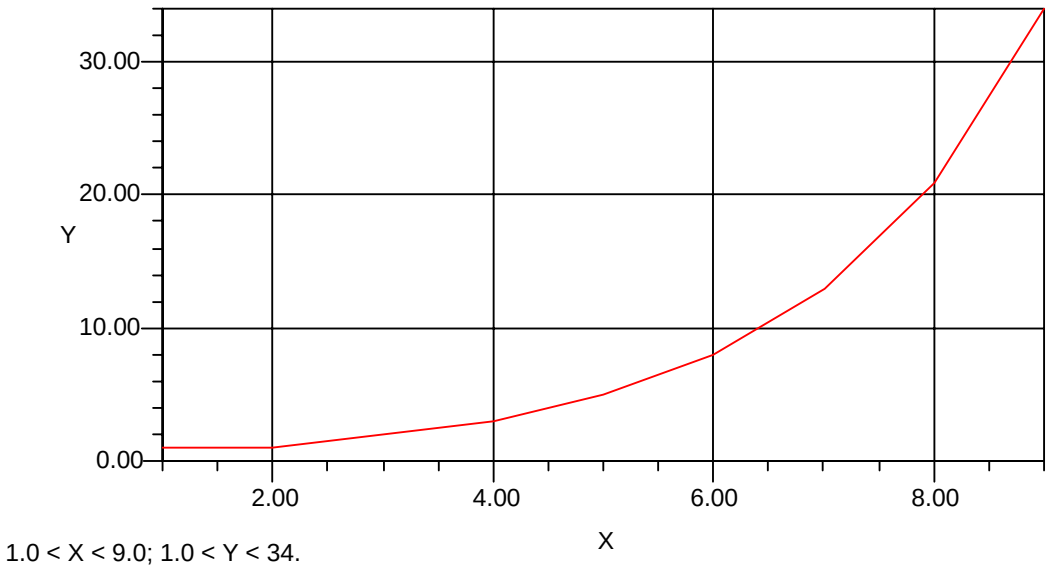

Figure 1.2: Plotting at Integer Values of the Abscissa

```
(c2) translate(f)$
(c3) plot(f, [-2, 3, 100.12],
     "X","F(X)","Plot made from a list of abscissa values")$
```

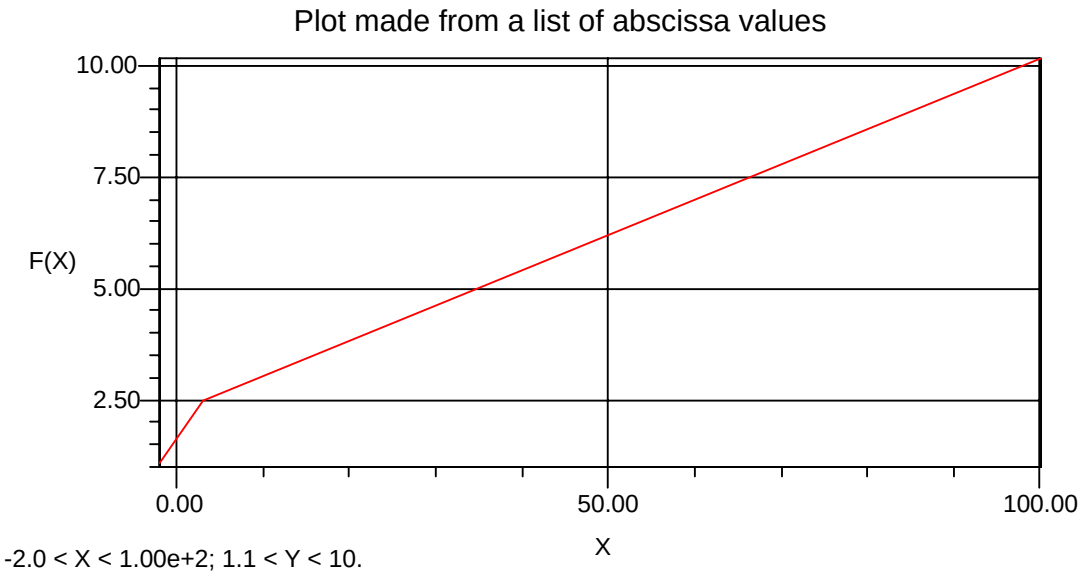

Figure 1.3: Plot of a Macsyma Function with a list of Abscissa Values

1.1.2 Plotting Parametric Curves in Two Dimensions

paramplot(*x-exps, y-exps, t-var, t-range,* {*'arg*$_1$*, ..., 'arg*$_n$}) *Special Form*

Plots *x-exps* as the x coordinate against *y-exps* as the y coordinate. The inputs $arg_1, \ldots, arg_n$ are optional.

The format for the first two arguments is the same as that for the first argument to **plot**. Thus if *x-exps* is $[xexp_1, \ldots, xexp_n]$ and *y-exps* is $[yexp_1, \ldots, yexp_k]$, then **max(n,k)** curves are plotted. Assuming $n > k$, they are: $xexp_1$ vs. $yexp_1$, ..., $xexp_k$ vs. $yexp_k$, $xexp_{k+1}$ vs. $yexp_k$, ..., $xexp_n$ vs. $yexp_k$.

The inputs *x-exps* and *y-exps* can be any of:

x-y-exps	Action
exp	**paramplot** plots a curve of (*x-exp*,*y-exp*) against *t-var*.
$[exp_1, \ldots, exp_n]$	Plots n curves of exp_i against *t-var*. Each exp_i is evaluated in the context: `float(ev(`exp_i`,` *t-var*=*value from x-range*`, numer))`. An error is signaled if this does not result in a floating-point number.

The input *t-range* can be any of:

t-range	Meaning

low, *high*	where *low* and *high* evaluate to numbers. The value *low* may be either greater or less than *high*. The input *t-var* takes on **plotnum** values equally spaced between *low* and *high*. See Section 1.3.1, page 24.
	Note: The dependent variables are evaluated at *low* first.
low, *high*, **integer**	This form has the same effect as the above except that *t-var* takes on only integer values between *low* and *high* inclusive.
[val_1, ..., val_n]	*t-var* takes the values specified by the list of values.
arrayname	where *arrayname* is the name of a declared floating-point one-dimensional array. Such an array can be created by means of a command such as `array(`*arrayname*`, float,` *max-index*`);`. In this case, *t-var* takes the values from *arrayname*[0] through *arrayname*[*max-index*]. (Here, *max-index* is the maximum index of *arrayname*.)

The optional arguments arg_1, ..., arg_n can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs arg_1, ..., arg_n can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

An alternative form for **paramplot** is `paramplot(`*y-funs*`,` *x-range*`,` arg_1`, ..., `arg_n`);`. Here, *y-funs* must be a function of one argument or a list of functions of one argument. The functions must be either translated or compiled functions, and they must return a floating-point number when given floating-point or integer arguments. This form of **paramplot** acts as though you had not only given an argument to the *y-funs*, but also specified that argument as the *variable* in the form above. For example, `paramplot(f,-2,2);` acts like `paramplot(f(x), x,-2, 2);`. This provides a quicker evaluation of the first argument and for that reason no checking is done on the result. If the wrong sort of number is returned, the plot is not useful.

Examples

This example Figure 1.4, page 7, plots $\cos t$ for the x-axis and $\sin t$ for the y-axis as t takes on **plotnum** values between 0 and 2π (see Section 1.3.1, page 24). If **equalscale** is **true** this draws a circle (see Section 1.4.1.1, page 34).

```
(c1) paramplot(cos(t),sin(t), t, 0, 2*%pi), equalscale:true$
```

This example plots $f(x)$ vs. $g(x)$ as x goes from 0 to 2π. The following example plots Macsyma functions, and results in the same plot as the previous example Figure 1.4, page 7.

```
(c1) f(x):=(mode_declare(x,float),cos(x))$
(c2) g(x):=(mode_declare(x,float),sin(x))$
(c3) translate(f,g)$
(c4) paramplot(f, g, x, 0, 2*%pi), equalscale:true$
```

1.1.3 Contour Plots and Implicit Plots in Two Dimensions

contourplot(*z-funs, x-var, x-range, y-var, y-range,* {*'*arg_1*, ..., '*arg_n}) *Special Form*

Plots level contours of the expressions *z-funs*, which are functions of *x-var* and *y-var*. *z-funs* can be a single expression or a list of expressions. *x-range* and *y-range* can take on the same forms as the

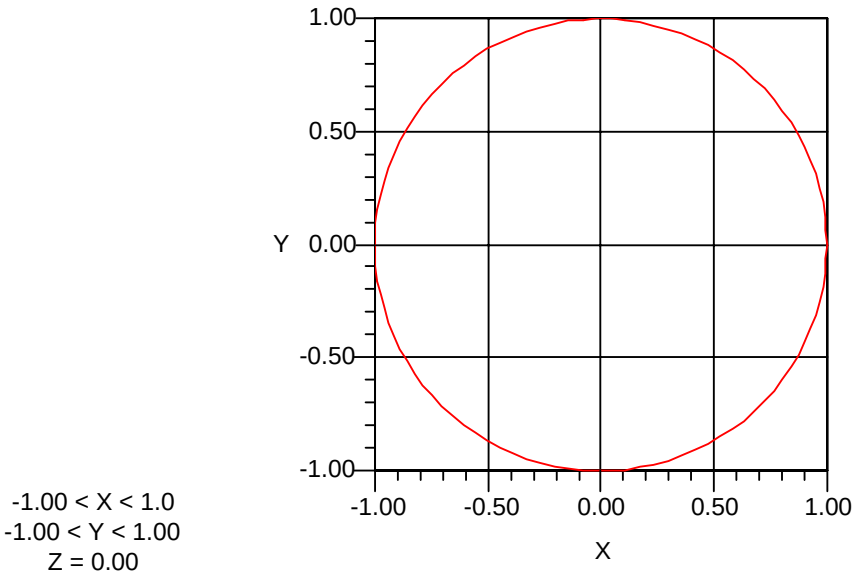

Figure 1.4: Drawing a circle with the Paramplot command

range arguments to the function **plot**. The option variables **contours**, **zmax**, and **zmin** determine what contours are displayed. Changing **contours** does not require any more points to be calculated, so it makes sense to change **contours** in a Macsyma break, after a plot, or before using **replot**. The contours are drawn using linear interpolation, so they tend to look rough, especially near saddle points, unless **plotnum** or **plotnum0** and **plotnum1** are large. Since the contour tracking algorithm is quite smart, all contours are either closed loops, or terminate at the boundaries of the region. The arg_1, ..., arg_n are optional arguments. The optional arguments arg_1, ..., arg_n can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs arg_1, ..., arg_n can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

contourplot is exactly the same as `plot3d(..., contour)`. That is, it calculates the same points as **plot3d** would, but displays the points as a contour plot.

contours *default:* **20** *Option Variable*

If **contours** is a positive integer, then approximately **contours** contours are drawn. The levels between them are "nice" values. "Nice" means in multiples of $a \cdot 10^n$, where a is 1, 2, or 5. The levels are computed by the same function that determines the placement of tick marks on the axes. The contour levels are chosen between the maximum and minimum z values to be plotted. This choice is influenced by the settings of **zmax** and **zmin**, described below.

If **contours** is a negative integer, then `abs(contours);` evenly spaced contours are drawn. In this case the levels are at "nasty" values and you may want to set **labelcontours** to **false**, to suppress the

display of long numbers on the contours. Of course, the range for these contours can be restricted by setting **zmax** and **zmin** appropriately.

If **contours** is a list of numbers such as `contours:[1,0.5,%pi];`, then those numbers are used to determine the heights of the contours. The numbers in the list can be in any order. See also **labelcontours**.

implicit_plot(*expr,x-var,xlo,xhi,y-var,ylo,yhi*) *Function*

Plots the graph of functions which are implicitly specified by *expr* in the (x, y) plane. *expr* can be either an expression in *x-var* and *y-var* or an equation of two expressions in *x-var* and *y-var* or a list of expressions or equations. The inputs *xlo* and *xhi* are the lower and upper limits of the x-coordinate and *ylo* and *yhi* are the lower and upper limits of the y-coordinate. **implicit_plot** uses Macsyma's **contourplot** machinery to generate the plot.

Do `example(implicit_plot);` for an example, and `demo(implicit_plot);` for a longer demonstration.

Examples

This example Figure 1.5, page 8, illustrates how to calculate and display 10 contours of $\frac{1}{2}y^2 + \cos x + \frac{1}{2}x$ where x takes **plotnum0** values between -6 and 6 and as y takes **plotnum1** values between -3 and 3.

```
(c1) contours:10$
(c2) contourplot(y^2/2+cos(x)+x/2,x,-6,6,y,-3,3)$
```

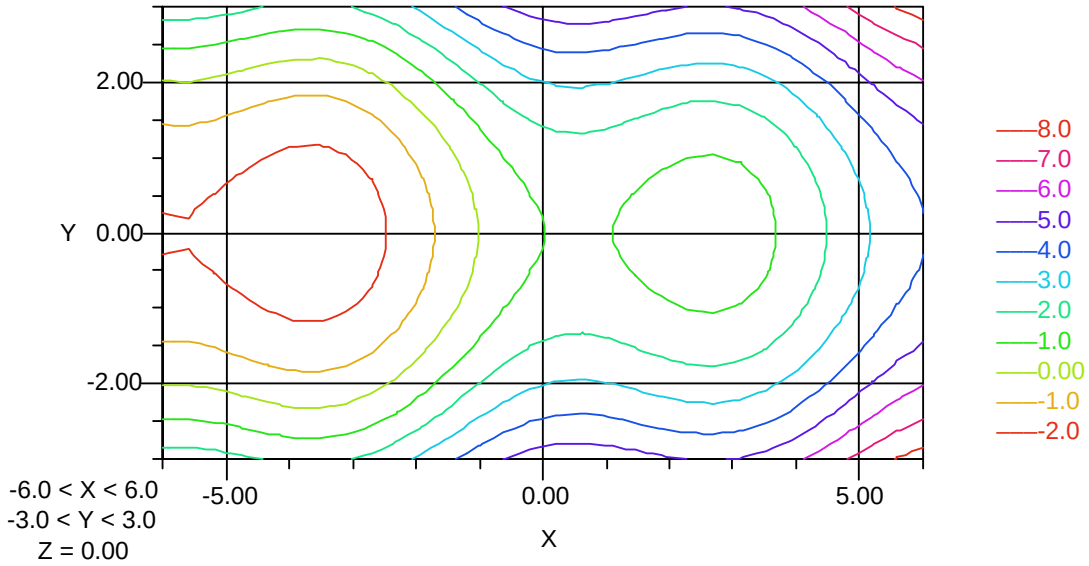

Figure 1.5: Contour plot in two dimensions

The next example Figure 1.6, page 9, illustrates plotting on a 10×10 grid:

```
(c1) contours:plotnum0:plotnum1:10$
(c2) contourplot(random(100), x, 0, 1, y, 0, 1)$
```

To find the zeroes of $z^3 = 1$ in the complex plane, execute either of the following sequences of commands resulting in Figure 1.7, page 10. The odd values for **plotnum0** and **plotnum1** improve the accuracy, which is needed to plot the intersection point at the center of the plot. The three curved lines are the

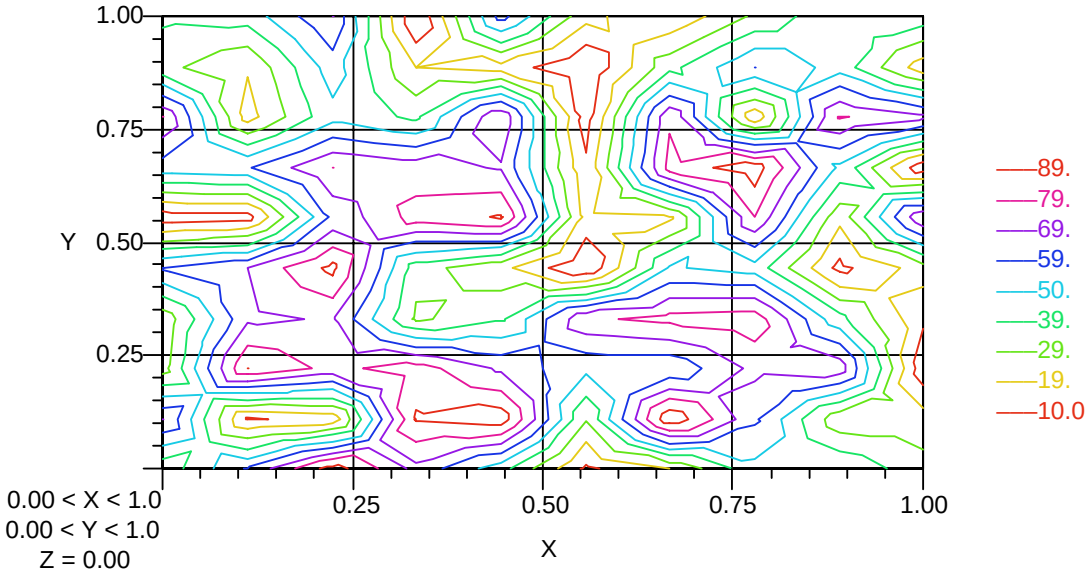

Figure 1.6: Contours Generated by 100 Random Points in Two Dimensions

locus *realpart(expression)=1* and the three straight lines are the locus *imagpart(expression)=1*. The roots of $z^3 = 1$ are the three intersection points of the curved and straight lines.

```
(c1) plotnum0:plotnum1:61$
(c2) implicit_plot([realpart((x+%i*y)^3-1),imagpart((x+%i*y)^3-1)],
     x,-1.5,1.5,y,-1.5,1.5)$
(c1) (plotnum0:plotnum1:61, contours:[0], labelcontours:false)$
(c2) contourplot([realpart((x+%i*y)^3-1),imagpart((x+%i*y)^3-1)],
     x,-1.5,1.5,y,-1.5,1.5)$
```

Macsyma 2.0 and successors enable you to turn on/off and edit contour labels after generating a plot using controls in the Macsyma Front End.

1.1.4 Plotting Discrete Sets of Points in Two Dimensions

graph(*x-lists, y-lists,* {*'arg*$_1$*, ..., 'arg*$_n$}) *Special Form*

Plots points specified by the *x-lists* and *y-lists*, interpreting *x-lists* as a list of abscissas, and *y-lists* as a list of ordinates. The inputs *arg*$_1$, ..., *arg*$_n$ are optional arguments.

The *x-lists* and *y-lists* can be in any of the formats below:

Format	Action
[*xlist*$_1$, ..., *xlist*$_k$]	Each of the *x-list*$_i$ is a list of numbers. If the length of *x-lists* is less than that of *y-lists* then *x-lists* is filled with *x-list*$_k$ to make the lengths the same. Similarly if the length of *y-lists* is less than that of *x-lists*, then *y-lists* is filled with *y-list*$_k$ to make the lengths the same. If the length of *y-lists* is k, then k curves of *y-list*$_1$ vs. *x-list*$_1$, ..., *y-list*$_k$ vs. *x-list*$_k$ are plotted. If one of the *x-list*$_i$ is shorter then the corresponding *y-list*$_i$, then the extra elements of *y-list*$_i$ are ignored.

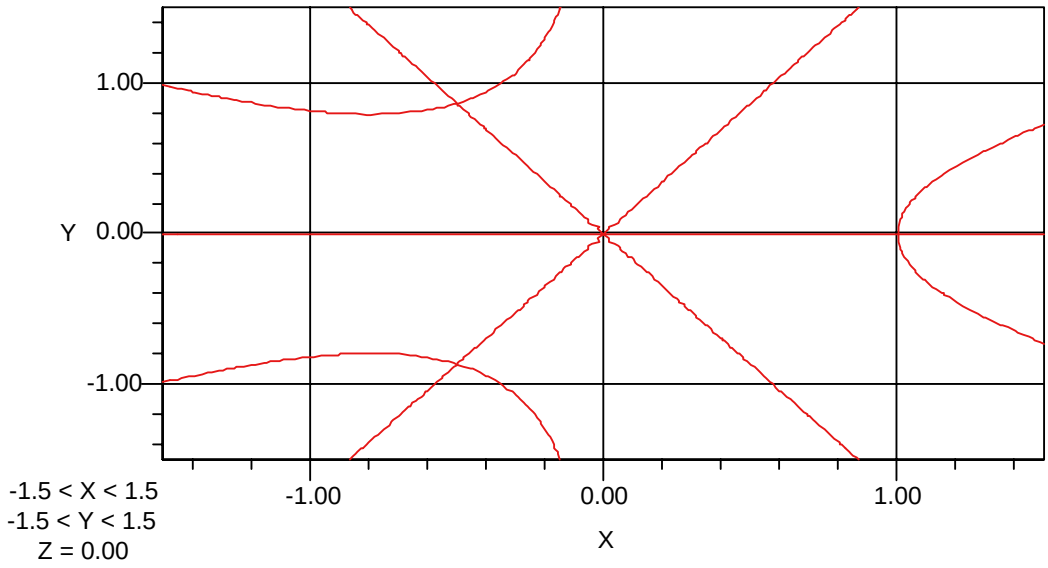

Figure 1.7: A plot of an implicit relation

[*x-ptn*$_1$, ..., *x-ptn*$_n$]	Where *x-ptn*$_i$ evaluates to a number. Lists of numbers are interpreted as lists of x coordinates and y coordinates. They are inserted into a list, which is then in the form of the first case above.
arrayname	Where *arrayname* is the name of a declared one-dimensional array of floating-point numbers. The elements of the array are then inserted into a list, which is inserted into yet another list, thus producing the form of the first case above.
2d-arrayname	Where *2d-arrayname* is the name of a declared two-dimensional array of floating-point numbers. This is interpreted as a list of lists of numbers, which is exactly in the form of the first case above.

The optional arguments *arg*$_1$, ..., *arg*$_n$ can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs *arg*$_1$, ..., *arg*$_n$ can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

Example

This example Figure 1.8, page 11, draws a line connecting $[1, 5]$, $[2, 10]$, and $[3, 6]$.

```
(c1) graph([1,2,3], [5,10,6])$
```

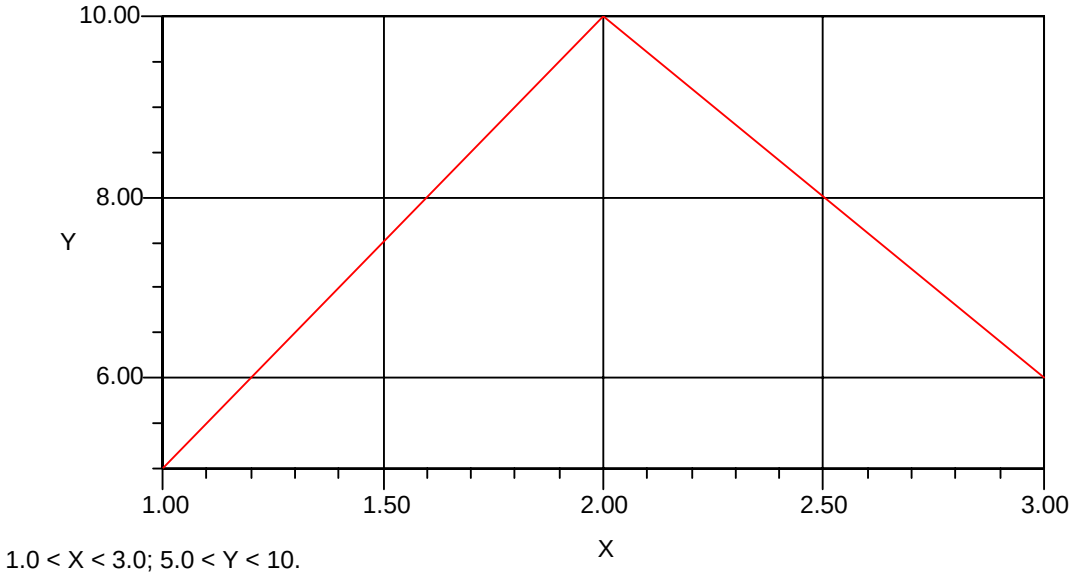

Figure 1.8: Plotting discrete points in two dimensions

1.1.5 Plotting Vector Fields in Two Dimensions

plot2_vect(*vecfield,x-var,x-min,x-max, y-var,y-min,y-max,* {*'arg1,...,'argn*}) *Function*

Plots vector fields and direction fields in two dimensions. *vecfield* is a list of two expressions, for the first and second components of the vector field; or a list of two functions; or one function whose value is a list of two values. *x-var* is the name of the variable which is plotted horizontally, and *y-var* is the name of the variable which is plotted vertically. For the region where vector basepoints are located, *x-min* is the lower limit and *x-max* is the upper limit of *x-var*, and *y-min* is the lower limit and *y-max* is the upper limit of *y-var*. The *'argi* are all the optional plotting variables allowed for the special form **graph2**.

The option variables **plotnum0** and **plotnum1** control the number of plot points in the horizontal and vertical directions respectively.

The *x-y-min-max* variables do not control the plot size directly. **xmin**, **xmax**, **ymin** and **ymax** do this, as for other plotting functions. Usually the plot size should be a little larger than the limits of the *x-var* and *y-var* to allow room to draw the vectors or direction lines.

plot_vect_scale *default:* **false** *Option Variable*

This option variable can be assigned the name of a function which is applied to the length of each vector, before the vector is plotted by **plot2_vect** or **plot3d_vect**. **plot_vect_scale** can be assigned the name of a system function (such as **'log**), or a user-defined function. If the scaling function results in a negative number or negative infinity, *e.g.* `log(veclength)` < 1 or `log(veclength)` $= 0$, then the length of the vector is set to zero. If **plot_vect_scale** is assigned the value `'plot_vect_scale` or **false**, then no scaling occurs.

plot_vect_head *default:* **true** *Option Variable*

If **true** then **plot2_vect** and **plot3d_vect** plots vector fields with arrowheads. When **false** then **plot2_vect** and **plot3d_vect** plot direction fields, in which no arrowheads are drawn, and the shaft of each arrow is centered at the plot point.

plot_vect_head_angle *default:* **0.5236** *Option Variable*

A floating point number which specifies the angle between the shaft of the vector and head edges of the head of the vector.

plot_vect_head_size *default:* **0.100** *Option Variable*

A floating point number which specifies the length of the edges of the vector head, as a fraction of the length of the vector shaft.

Example A sample vector field is plotted in Figure 1.9, page 12.

```
(c1) vect:[y/4.,x/4.]$
(c2) block([plotnum0:10, plotnum1:10],
           plot2_vect(vect,x,-1,1,y,-1,1, false, false,
           "Plot of Saddle Point Vector Field [Y/4,X/4]"))$
```

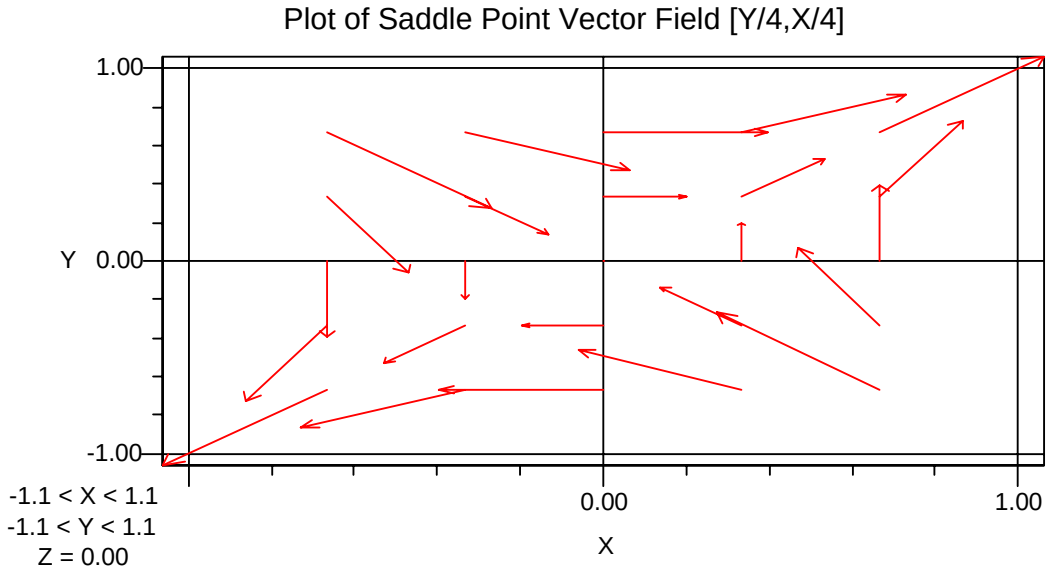

Figure 1.9: Plotting a vector field in two dimensions

1.1.6 Special Capabilities For Two Dimensional Plotting

1.1.6.1 Adaptive Density Plotting

The package **adaplot** does adaptive density plotting of lines in two dimensions. Adaptive density plotting is useful for plotting curves whose curvature varies sharply, and curves for which the defining expressions are expensive to evaluate.

There are two functions, **adaplot2** and **adaparamplot2**:

adaplot2(*yexpr,xvar,xlow,xhigh,ylow,yhigh,*{*'arg1,...,'argn*}) *Function*

yexpr is an expression or function for the dependent variable. *xvar* is the name of the independent variable, and is plotted on the horizontal axis. *xlow* and *xhigh* are the limits of *xvar* in plotting the curve. (Note that these numbers do not specify the limits of the x-axis display. **xmin** and **xmax** perform this function, as for most other Macsyma plotting functions.) *yhigh-ylow* is an estimate of the

range of vertical-values in the plot. It is used by **adaplot2** to determine the angles of various lines as they will appear in the plot. (The limits of the y-axis display are *ymin* and *ymax*.) The *'argi* are the optional plotting variables allowed for the special form **graph2**. (No special provision has been made for log plots, and it is recommended that the logs be applied explicitly.)

adaparamplot2(*xexpr,yexpr,tlow,thigh,xlow,xhigh, ylow,yhigh,*{*'arg1,...,'argn*}) *Function*

xexpr and *yexpr* are expressions or functions for the dependent variables. *tvar* is the name of the independent parameter variable. *tlow* and *thigh* are the limits of *tvar* used in plotting the curve. *xhigh*-*xlow* is an estimate of the range of horizontal values in the plot. *xmin* and *xmax* determine the limits of the display. *yhigh*-*ylow* is an estimate of the range of vertical values in the plot. *ymin* and *ymax* determine the limits of the display. The *'argi* are the optional plotting variables allowed for the special form **graph2**.

Variables which affect the adaptive plot point density are **plotnum** and:

curv_tol *default:* **0.1** *Option Variable*

(A single precision floating point number.) The maximum allowable change in curve angular direction (in radians) at a plot point, for large line segments. The maximum permitted value of `dt` (or `dx`) is `4.0*(`*thigh*`-`*tlow*`)/`*plotnum*. Normally, **curv_tol** is used to control the number of plot points.

The curvature tolerance can be violated for smaller line segments (as controlled by **dt_factor**) or if *dx* = *dxmin* and the angular change still exceeds **curv_tol**.

dt_rate *default:* **2.0** *Option Variable*

(A single precision floating point number which must be greater than 1.0.) The maximum factor by which step size can be increased or decreased in one iteration at a plot point.

dt_ratio *default:* **40.0** *Option Variable*

(A single precision floating point number.) The ratio of largest to smallest permissible step size between plot points. This is used to control the minimum size limit for line segments.

dt_factor *default:* **4.0** *Option Variable*

(A single precision floating point number.) The factor by which curvature tolerance increases for shorter curve segments. Raising **dt_factor** makes the curvature tolerance larger for short line segments, while leaving the tolerance unchanged for the longest line permissible segments.

1.1.6.2 Character Plotting (2D Only)

Even in a modern user interface environment, character plots can be useful; for example, for sending graphical information in text electronic mail messages.

The functions **char_plot**, **char_paramplot**, **char_graph**, and **char_multigraph** produce plots of specified functions and sets of data points. These commands are described below.

The plots produced by these commands are character plots with coordinate axes located at minimum x and y values of the plot. The x and y coordinates are independently scaled to optimally use the specified graphing area.

Note: This can distort the shape of the graph. Thus a circle could become an ellipse. The origin of the graph, the lower left-hand corner, is given on the graph by the values of **xorg** and **yorg**; the computed increments are given by the values of **xdelta** and **ydelta** and the maximum x and y values are given by **xmax** and **ymax**. The axes are labeled with the number sequence 0, 2, 4, 6, 8, 0, 2, 4, ... as an aid in counting the number of increments from the origin.

In Macsyma 2.0 and successors, you may need to change the font in the notebook section containing a character plot to a fixed width font (such as Courier).

char_plot(*f(x), x, low, high,* { arg_1, ..., arg_n}) *Function*

Plots the expression *f(x)* in the domain $low < x < high$. The inputs arg_1, ..., arg_n are optional arguments that control the precise form of the display.

The first argument to **char_plot** can also be a list of functions rather than just a single function. This allows you to plot several functions on the same set of axes.

The optional arguments can be any of the following:

arg_n	Description
integer	Macsyma computes only integer values of the abscissa. If you specify **integer**, you cannot specify any other optional arguments.
list	Macsyma interprets this list as a list of plotting characters to be used to display the function. If you do not specify *list*, Macsyma uses an asterisk (*). If you want to specify a special symbol such as `;` or `space`, you must precede that symbol with a backslash (\).
xlabel	*xlabel* is a string that labels the x axis of the plot. *xlabel* is not evaluated by Macsyma. If *xlabel* is **false**, the axis is displayed without a label.
ylabel	*ylabel* is a string that labels the y axis of the plot. *ylabel* is not evaluated by Macsyma. If *ylabel* is **false**, the axis is displayed without a label.

The syntax `char_plot(f(x),x,[`x_1`,`x_2`, ...,`x_n`]);` can also be used. This syntax causes **char_plot** to display a plot of $f(x)$ for each of the x_i specified. When **char_plot** is used in this way, the keyword **integer** described above is ignored even if given. The other possible optional arguments behave as described above.

char_paramplot(*f1(t), f2(t), t, low, high,* {*list*}) *Function*

Plots the plane curve $f(t) = (f1(t), f2(t))$ for $low < t < high$. The input *list* is an optional argument for specifying the plotting characters. If you do not specify *list*, Macsyma uses an asterisk (*). For example, `char_paramplot(cos(t), sin(t), t, 0, 2*%pi);` plots a circle.

The first two arguments themselves can be lists if you want to plot more than one curve at one time. The syntax would be: `char_paramplot([f1(t),g1(t),...,h1(t)], [f2(t),g2(t),...,h2(t)], t, low, high);`. This syntax plots the plane curves `f(t) = (f1,f2)`, `g(t) = (g1,g2)`, ..., `h(t) = (h1,h2)` using the default plotting character (*). For example, `char_paramplot([cos(t), cos(t)+7], [sin(t), sin(t)], t, 0, 2*%pi, ["@"]);` plots two circles using the character "@" as the plotting character.

char_graph(*[*x_1, ..., x_n*], [*y_1, ..., y_n*],*{ arg_1, ..., arg_n}) *Function*

Graphs the two sets of points created by (x_1,y_1), ..., (x_n,y_n). The inputs arg_1, ..., arg_n are optional arguments and are described below.

There are some variations of the **char_graph** function. You can specify the points you want to graph as lists of coordinate pairs using the following syntax:

`char_graph([[`x_1`, `y_1`], ..., [`x_n`, `y_n`]]);`

You can also graph one x-domain over several y-ranges using the following syntax:

`char_graph(`*xset*`,[`$yset_1$`, ...,`$yset_n$`]);`

The optional arguments can be any of the following:

arg_n	Description
integer	Macsyma computes only integer values of the abscissa. If you specify **integer**, you cannot specify any other optional arguments.

list	Macsyma interprets this list as a list of plotting characters to be used to display the function. If you do not specify *list*, Macsyma uses an asterisk (*). If you want to specify a special symbol such as `;` or `space`, you must precede that symbol with a backslash (\).
xlabel	*xlabel* is a string that labels the x axis of the plot. *xlabel* is not evaluated by Macsyma. If *xlabel* is **false**, the axis is displayed without a label.
ylabel	*ylabel* is a string that labels the y axis of the plot. *ylabel* is not evaluated by Macsyma. If *ylabel* is **false**, the axis is displayed without a label.

char_multigraph(*[[xset$_1$, yset$_1$], ..., [xset$_n$, yset$_n$]], arg$_1$, ..., arg$_n$*) *Function*

Produces a scatter graph in the Macsyma Listener involving several *x*-domains each with a single *y*-range.

The inputs *arg*$_1$, ..., *arg*$_n$ are optional arguments and are described below.

The optional arguments can be any of the following:

arg$_n$	Description
integer	Macsyma computes only integer values of the abscissa. If you specify **integer**, you cannot specify any other optional arguments.
list	Macsyma interprets this list as a list of plotting characters to be used to display the function. If you do not specify *list*, Macsyma uses an asterisk (*). If you want to specify a special symbol such as `;` or `space`, you must precede that symbol with a backslash (\).
xlabel	*xlabel* is a string that labels the x axis of the plot. *xlabel* is not evaluated by Macsyma. If *xlabel* is **false**, the axis is displayed without a label.
ylabel	*ylabel* is a string that labels the y axis of the plot. *ylabel* is not evaluated by Macsyma. If *ylabel* is **false**, the axis is displayed without a label.

plotheight *default:* **24** *Option Variable*

The height of the area used for character plotting. The plotting is done on a grid that is **plotheight** characters high and **linel** wide.

1.1.6.3 Line Plotting (2D Only)

These commands normally are not needed since the main plotting commands provide the same functionality and adjust to the type of display on your computer.

plot2(*y-exps, variable, x-range, 'arg*$_1$*, ..., 'arg*$_n$) *Special Form*

This function is called by the function **plot** when the terminal type is appropriate. It is not recommended that users call this function directly.

Plots *y-exps* in the *y* direction, while *variable*, the *x* axis, takes on values specified by *x-range*. The arguments can take various forms, each described in the tables below.

The input *y-exp* can be any of:

y-exps	Action
exp	**plot2** plots a curve of *exp* against *variable*.

[exp_1, ..., exp_n]	Plots n curves of exp_i against *variable*. Each exp_i is evaluated in the context: `float(ev(`*exp$_i$*`,` *variable=value from x-range*`, numer))`. An error is signaled if this does not result in a floating-point number.

The input *x-range* can be any of:

x-range	Meaning
low, *high*	where *low* and *high* evaluate to numbers. The value *low* may be either greater or less than *high*. The input *variable* takes on **plotnum** values equally spaced between *low* and *high*. See Section 1.3.1, page 24. **Note:** The first argument is evaluated at *low* first. For example, `plot(1/x,x,-1,-3);` calculates $1/(-1.0)$ before $1/(-3.0)$. This matters only if the computation of the first argument changes a variable which in turn changes the value returned by subsequent computation. Whether or not *low* < *high*, **min**(*low*, *high*) is displayed on the left side of the plot. This can be overridden using the optional argument **special** with **reflect** as **xfun**. (see Section 1.3.2.2, page 28).
low, *high*, **integer**	This form has the same effect as the above except that *variable* takes on only integer values between *low* and *high* inclusive.
[val_1, ..., val_n]	*variable* takes the values specified by the list of values.
arrayname	where *arrayname* is the name of a declared floating-point one-dimensional array. Such an array can be created by means of a command such as `array(`*arrayname*`, float,` *max-index*`);`. In this case, *variable* takes the values from *arrayname*[0] through *arrayname*[*max-index*]. (Here, *max-index* is the maximum index of *arrayname*.)

The optional arguments arg_1, ..., arg_n can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs arg_1, ..., arg_n can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

An alternative form for **plot2** is `plot2(`*y-funs*`,` *x-range*`,` *arg$_1$*`, ...,` *arg$_n$*`);`. Here, *y-funs* must be a function of one argument or a list of functions of one argument. The functions must be either translated or compiled functions, and they must return a floating-point number when given floating-point or integer arguments. This form of **plot2** acts as though you had not only given an argument to the *y-funs*, but also specified that argument as the *variable* in the form above. For example, `plot2(f,-2,2);` acts like `plot2(f(x), x,-2, 2);`. This provides a quicker evaluation of the first argument and for that reason no checking is done on the result. If the wrong sort of number is returned, the plot is not useful.

graph2(*x-lists, y-lists,* {*'arg$_1$, ..., 'arg$_n$*}) *Special Form*

This function is called by the function **graph** when the terminal type is appropriate. It is not recommended that users call this function directly.

This function points specified by the *x-lists* and *y-lists*, interpreting *x-lists* as a list of abscissas, and *y-lists* as a list of ordinates. The arg_1, ..., arg_n are optional arguments.

The *x-lists* and *y-lists* can be in any of the formats below:

Format	Action
[$xlist_1$, ..., $xlist_k$]	Each of the *x-list*$_i$ is a list of numbers. If the length of *x-lists* is less than that of *y-lists* then *x-lists* is filled with *x-list*$_k$ to make the lengths the same. Similarly if the length of *y-lists* is less than that of *x-lists*, then *y-lists* is filled with *y-list*$_k$ to make the lengths the same. If the length of *y-lists* is k, then k curves of *y-list*$_1$ vs. *x-list*$_1$, ..., *y-list*$_k$ vs. *x-list*$_k$ are plotted. If one of the *x-list*$_i$ is shorter then the corresponding *y-list*$_i$, then the extra elements of *y-list*$_i$ are ignored.
[*x-ptn*$_1$, ..., *x-ptn*$_n$]	Where *x-ptn*$_i$ evaluates to a number. Lists of numbers are interpreted as lists of x coordinates and y coordinates. They are inserted into a list, which is then in the form of the first case above.
arrayname	Where *arrayname* is the name of a declared one-dimensional array of floating-point numbers. The elements of the array are then inserted into a list, which is inserted into yet another list, thus producing the form of the first case above.
2d-arrayname	Where *2d-arrayname* is the name of a declared two-dimensional array of floating-point numbers. This is interpreted as a list of lists of numbers, which is exactly in the form of the first case above.

The optional arguments *arg*$_1$, ..., *arg*$_n$ can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs *arg*$_1$, ..., *arg*$_n$ can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

1.2 Three-Dimensional Graphics

Macsyma provides plotting of functions, parametric lines and surfaces, contours, data points, and vector fields in three dimensions. You can combine several plot commands to create a composite plot which combines several plot elements of the same or different types.

The main commands for three-dimensional plotting are

- **plot3d**, Section 1.2.1, page 18
- **paramplot3d**, Section 1.2.2, page 20
- **contourplot3d**, Section 1.2.3, page 20
- **graph3d**, Section 1.2.4, page 21
- **plotsurf**, Section 1.2.5, page 21
- **plot3d_vect**, Section 1.2.6, page 23
- **complex_plot3d**, Section 1.2.1, page 20

1.2.1 Plotting Functions of Two Variables

The command **plot3d** has two different calling sequences, both of which are described below.

plot3d(*z-exprs, x-var, x-range, y-var, y-range,* {*'arg*$_1$*, ..., 'arg*$_n$}) *Special Form*

Plots *z-exprs* against *x-var* and *y-var*. The plot consists of curves of *z-exprs* against *x-var* (the x coordinate*y-var* (the y coordinate) held fixed. The $arg_1, \ldots, arg_n$ are optional. Perspective is used and curves further away from the viewer have those parts of them which are hidden by the closer curves removed.

The context of evaluation is `float(ev(`*exp*$_i$`,` *x-var*=*value gotten from x-range*`,` *y-var*=*value gotten from y-range*`, numer));`

The format for *x-range* and *y-range* is the same as for **plot** except that if *y-range* is of the form *low*, *high* then *y-var* takes on **plotnum1** instead of **plotnum** values.

The format of optional arguments is the same as for **plot** except that additional option variables are available. See **3d** and **contour**, Section 1.7.3, page 53.

plot3d(*z-funs, x-range, y-range,* {*'arg*$_1$*, ..., 'arg*$_n$}) *Special Form*

This is analogous to the alternative form for **plot**. *z-funs* must be a function or list of functions of two arguments, which must return a floating-point argument when given floating-point, or integer, if the argument **integer** is used for either *x-range* or *y-range* arguments. The $arg_1, \ldots, arg_n$ are optional arguments. The functions must be translated or compiled. If you expect to make several three dimensional plots this form is recommended.

The optional arguments $arg_1, \ldots, arg_n$ can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs $arg_1, \ldots, arg_n$ can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

plot_data(*mat*) *Special Form*

Plots a matrix of data, using the matrix indices as integer values of x and y coordinates.

Do `example(plot_data);` for an example.

Examples

In the example below Figure 1.10, page 19, the expression $xe^{-x^2-y^2}$ is plotted as x takes on **plotnum** values between -2 and 2 and y takes on **plotnum1** values between -1.5 and 2.5.

```
(c1) plot3d(exp(-x^2-y^2)*x,x,-2,2,y,-1.5,2.5)$
```

In this example Figure 1.11, page 19, first arrange for automatic translation, then define and plot a function g.

```
(c2) translate:true$
(c3) g(x,y):=(mode_declare([x,y], float), exp(-x*x-y*y))$
(c4) plot3d(g,-2,2,-2,2)$
```

not3d *Keyword*

Keyword for: **plot3d**

The additional argument **not3d** to **plot3d** causes exactly the same points to be calculated as in the bare **plot3d**. Instead of plotting a three-dimensional representation of the data, the data is plotted

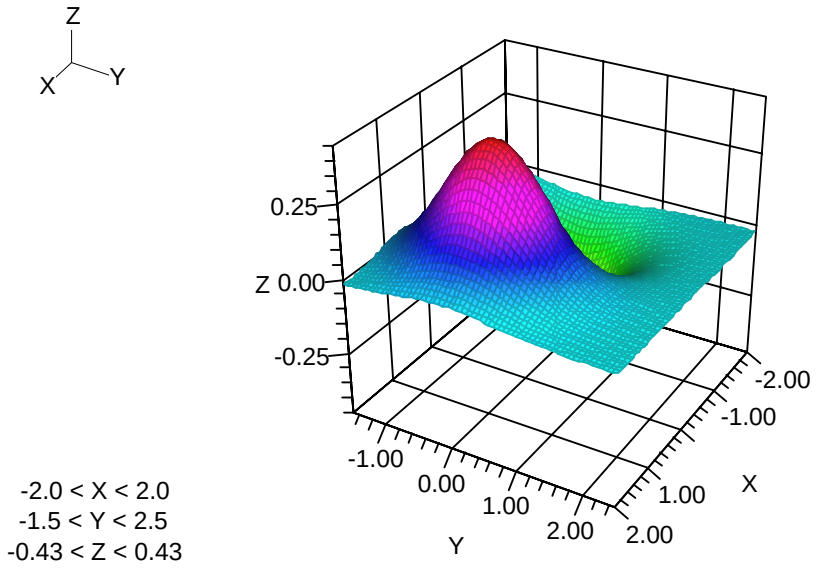

Figure 1.10: Plotting a function of two variables in three dimensions

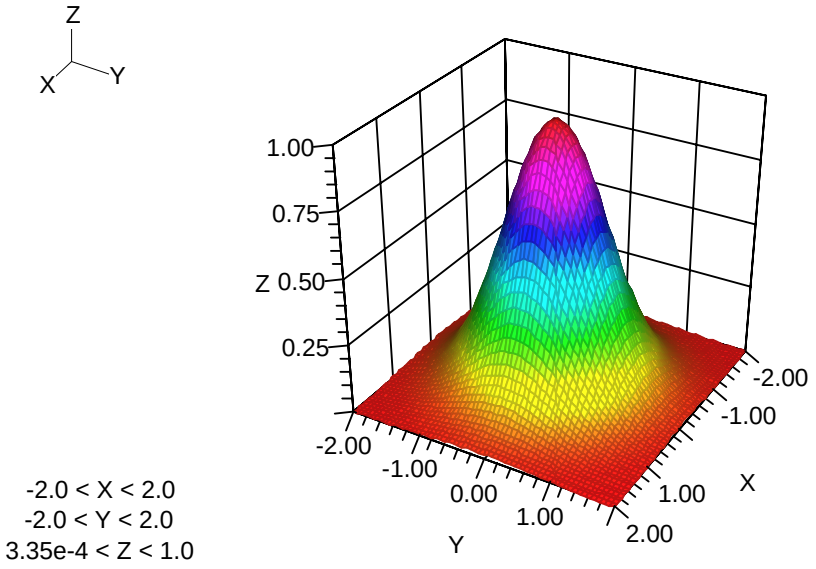

Figure 1.11: Plotting a function which is expressed as a Macsyma function

in two dimensions. Specifically one curve of z versus x for each value of y is plotted and so it is a convenient way to plot several curves on the same plot. (**not3d** is not available in Macsyma 2.0 and successors or Macsyma 419 and successors. See Section 1.3.3.3, page 31 for methods for superimposing plots.)

complex_plot3d(*(z-expr, xvar, xlow, xhigh, yvar, ylow, yhigh {, view})*) *Function*

Plots the possibly complex expression z-expr as a function of the variables xvar with range [xlow, xhigh] and yvar with range [ylow,yhigh].

The optional variable view (default:**'rectform**) can be one or more of:

- **'rectform** - plots the real and imaginary parts of z-expr.
- **'realpart** - plots real part of z-expr.
- **'imagpart** - plots imaginary part of z-expr.
- **'polarform** - plots complex absolute value and argument of z-expr.
- **'cabs** - plots complex absolute value of z-expr.
- **'carg** - plots complex argument of z-expr.

If more than one of these symbols are included in *view*, then one plot is generated for each symbol.

Do `example(complex_plot3d);` for an example.

1.2.2 Plotting Parametric Curves in Three Dimensions

paramplot3d(*[[xexpr,yexpr,zexpr]], var,varlow,varhigh*) *Function*

Generates parametric plots of curves embedded in three dimensional space. One curve is specified by the expressions *xexpr*, *yexpr*, and *zexpr*, and parameterized by the variable *var*. The limits on the curve parameter *var* are *varlow* and *varhigh*. Multiple curves can be plotted with one call to **paramplot3d** by giving a list-of-lists such as `[xexpr1,yexpr1,zexpr1], [xexpr2,yexpr2,zexpr2]` as the first argument to **paramplot3d**.

Alternatively, each curve can also be expressed as a function which returns a list of three floating point numbers. If only one curve is plotted, no list brackets are needed. Multiple curves can be represented as a list of such functions.

1.2.3 Contour Plots in Three Dimensions

contourplot3d(*z-funs, x-var, x-range, y-var, y-range, {'arg$_1$, ..., 'arg$_n$}*) *Function*

Creates a contour plot in the same way as **contourplot**, except that the plot is represented in 3 dimensions instead of being compressed into a plane. Each level contour appears at the appropriate elevation. Options for **contourplot3d** are identical for those of **contourplot**. See also **contours**, page 7 and **labelcontours**, page 47.

Macsyma 2.0 and its successors enable you to turn on/off and edit contour labels after generating a plot using controls in the Macsyma Front End.

Examples

In this example, we display in Figure 1.12, page 21, the same plot as shown in the discussion of **contourplot**, Section 1.1.3, page 6. This time, however, the contour lines are shown in three dimensions. This plot can be obtained with either of the commands shown below.

```
(c1) plot3d(y^2/2+cos(x)+x/2,x,-6,6,y,-3,3,contour)$
(c1) contourplot3d(y^2/2+cos(x)+x/2,x,-6,6,y,-3,3)$
```

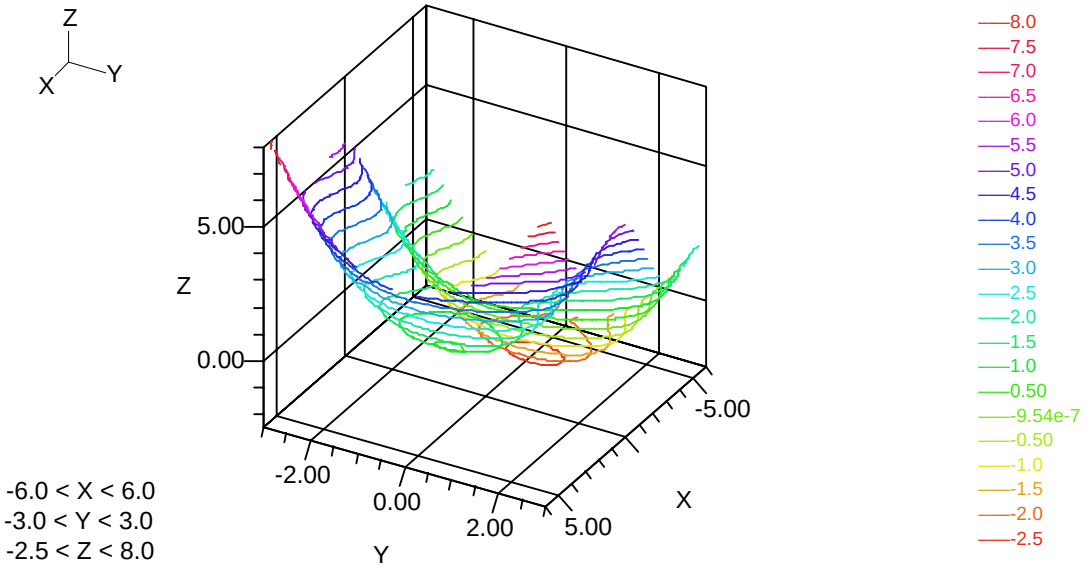

Figure 1.12: Contour plot in three dimensions

1.2.4 Plotting Discrete Sets of Points in Three Dimensions

graph3d(*x-lists, y-lists, z-lists,* {*'arg*$_1$, ..., *'arg*$_n$}) *Special Form*

This special form takes three arguments where **graph** took two and interprets them as lists of x, y and z points. It uses these points to draw lines using the three dimensional transformations. It can be used to add lines or axes to your three dimensional plot (see Section 1.3.3.3, page 32). The hidden line routines are not used.

The *arg*$_1$, ..., *'arg*$_n$ are optional arguments.

The optional arguments *arg*$_1$, ..., *arg*$_n$ can appear in any order. The rule for evaluating optional arguments is as follows: If the argument is atomic, it is evaluated, and the resulting values are used. The inputs *arg*$_1$, ..., *arg*$_n$ can be any of the following:

Optional Argument	Reference
x-label, y-label or title descriptor	See Section 1.4.4, page 45.
Line type descriptor	See Section 1.3, page 24.
first, **last** and **same**	See Section 1.3.3.3, page 31.
polar, **log**, **linlog**, or **loglin**	See Section 1.3.2, page 25.

1.2.5 Plotting Parametric Surfaces in Three Dimensions

plotsurf(*see below*) *Function*

This function plots two-dimensional surfaces embedded in three-dimensional space. Each surface is represented in parametric form as $[x(s,t), y(s,t), z(s,t)]$, where s and t are continuous real parameters, and $x(s,t)$, $y(s,t)$ and $z(s,t)$ are real-valued continuous functions. **plotsurf** plots a grid of **plotnum0** $\times$ **plotnum1** plot points, and interpolates quadrilaterals between the plot points. **plotsurf** permits you to construct a surface

from several sections, each of which is plotted at **plotnum0** × **plotnum1** plot points. (**plotnum0** and **plotnum1** can be lists of integers, so that each surface section can have a nodal grid of a different integer size.)

plotsurf has four calling syntaxes, in which the surface(s) are specified by mathematical expressions, functions, or arrays:

- `plotsurf([[x1,y1,z1],...,[xn,yn,zn]],s,slo,shi,t,tlo,thi)`, where
  - The first argument represents n surfaces with n triples of expressions. Each expression evaluates to a floating-point number and may reference the variables `s` and `t`.
  - `s` and `t` are the parameters used to specify the surface.
  - `slo` and `shi` give the lower and upper limits of the parameter `s`.
  - `tlo` and `thi` give the lower and upper limits of the parameter `t`.
- `plotsurf([f1,f2,...,fn],slo,shi,tlo,thi)`, where
  - The first argument represents n surfaces with n functions, each of which has two arguments and returns a triple of floating-point numbers. Each function must have been defined before the call to **plotsurf**. Note that the arguments to the functions `f1,...,fn` do not appear explicitly in the call to **plotsurf**, either in the list `[f1,...,fn]` or in the other arguments to **plotsurf**.
  - `slo` and `shi` give the lower and upper limits of the first parameter.
  - `tlo` and `thi` give the lower and upper limits of the second parameter.
- `plotsurf(f,slo,shi,tlo,thi)`

  When only one surface is plotted as in calling sequence (2), the list brackets surrounding the function name `f` may be omitted.
- `plotsurf([nodes,polygons])`

  Plots a surface which is specified by node points and polygons. In this case, the surface to be plotted is not specified in parametric form. The two arguments are:
  - `nodes` is a two-index array of nodal coordinates, where the first index is the node number, and the second index is one of $[0, 1, 2]$ to specify the first, second or third coordinate. (Arrays in Macsyma start with index value 0.) The value of each element in the array is a floating point number.

    Note: the arrays must be complete arrays.
  - `polygons` is a one-index array, where the index is the number of a polygon in the image. Each element of the array is a list of three or more (up to 32) integers, which specifies the nodes which form the corners of a polygonal face. Triangles, quadrilaterals and other n-gons can be mixed in the same image. (Note that the amount of storage allocated for each face is determined by the n-gon with the greatest number of sides.)

Do `example(plotsurf);` for an example. For a function to compute the curvature of a two-dimensional plot in three-dimensional space, use `curvsurf`. See the *Macsyma Mathematics and System Reference Manual* for more information.

plot_tessellation *default:* **4** *Option Variable*

A value of 4 means that two-dimensional surfaces are tessellated with quadrilaterals. A value of 3 means that two-dimensional surfaces are tessellated with triangles. Currently **plot_tessellation** affects the behavior of **plotsurf**, and not of **plot3d**.

Do `example(plot_tessellation);` for an example.

Example

The following command plots a sphere which is surrounded by a torus. The result is shown in Figure 1.13, page 23.

```
(c1) block([equalscale:true,plotnum0:17,plotnum1:13,
       title:"A Sphere and a Torus Superimposed"],
     plotsurf([[cos(ph)*sin(th),sin(ph)*sin(th),cos(th)],
       [cos(ph)*(3+sin(2*th)),sin(ph)*(3+sin(2*th)),cos(2*th)]],ph,-%pi,%pi,th,0,%pi))$
```

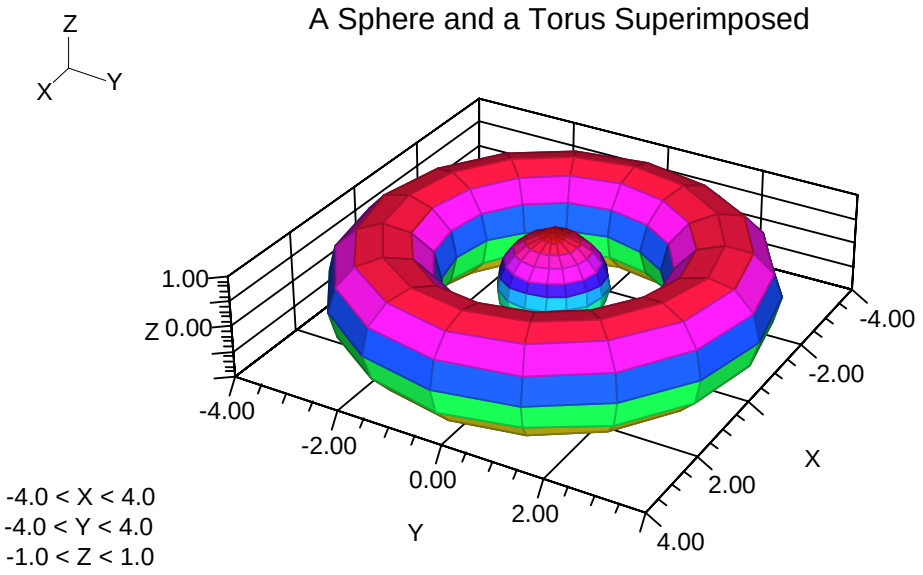

Figure 1.13: Plot of two parametric surfaces with one command

1.2.6 Plotting Vector Fields in Three Dimensions

plot3d_vect(*vecfield, x-var,x-min,x-max, y-var,y-min,y-max, z-var,z-min,z-max, {'arg1,...,'argn}*) *Function*

This function plots vector fields and direction fields in three dimensions. The calling arguments are as follows:

- *vecfield* is a list of three expressions, for the three components of the vector field; or a list of three functions; or one function whose value is a list of three values.
- *x-var*, *x-min*, *x-max* are the name and limits of the first variable.
- *y-var*, *y-min*, *y-max* are the name and limits of the second variable.
- *z-var*, *z-min*, *z-max* are the name and limits of the third variable.
- The *'argi* are all the optional plotting variables allowed for the special form **graph3d**. (However, placing plot symbols at each point will result in plotting the symbol at the head, tail, and arrow tips of each vector drawn.

The option variables **plotnum0**, **plotnum1**, and **plotnum2** control the number of plot points in the two horizontal directions and height directions respectively. If **plotnum2** is not specified, then **plot3d_vect** assigns it the value of **plotnum1**. The arguments *x-min*, *y-min*, *x-max*, *y-max*, *z-min* and *z-max* do not control the plot size directly. **xmin**, **xmax**, **ymin**, **ymax**, **zmin**, and **zmax** do this, as for other plotting functions.

To plot direction fields, see **plot_vect_head** and related option variables, Section 1.1.5, page 11. Direction fields have no vector heads and center plotted lines at each plot point.

Example

Figure 1.14, page 24, shows a sample direction field.

```
(c1) vect:[-y/8,x/8,sqrt(x^2+y^2)/8]$
(c2) block([plotnum0:4, plotnum1:4, plotnum2:9,
       xmin:-1.0,ymin:-1.0,zmin:-1.0,xmax:1.0,ymax:1.0,zmax:1.0,viewpt:[10.,10.,2.]],
         plot3d_vect(vect,x,-1,1,y,-1,1,z,-1,1,false,false,
           "Plot of Spiral Vector Field [-Y/8,X/8,sqrt(x^2+y^2)/8]"))$
```

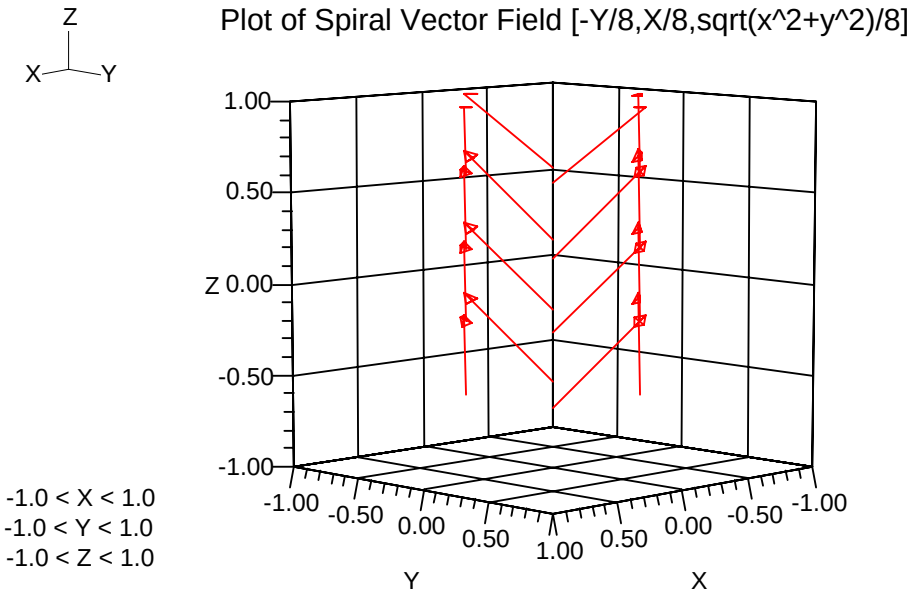

Figure 1.14: Plot of a vector field in three dimensions

1.3 Changing Plot Geometry

1.3.1 Changing the Number of Plot Points

The number of plot points and their coordinate locations are determined when a plot is first generated by Macsyma. While most other plot attributes can be edited in the Macsyma Front End after a plot is generated, the number and locations of plot points can only be modified by regenerating the plot from Macsyma.

plotnum *default:* **100** *Option Variable*

The number of points plotted on each line by **plot** and **paramplot** when given the *low*, *high* type of variable range.

plotnum0 *default:* **20** *Option Variable*

plotnum1 *default:* **20** *Option Variable*

The values of **plotnum0** and **plotnum1** are the numbers of plot points along the first and second coordinate axes respectively in three-dimensional plots. For parametric surfaces generated by **plotsurf**, they specify the number of plot points along the direction of the first and second parameters respectively. **plotnum0** an **plotnum1** affect the number of plot points generated by **plot3d**, **plotsurf**, **contourplot**, **contourplot3d**, **plot2_vect** and **plot3d_vect**.

When plotting several parametric surfaces in one call to **plotsurf**, **plotnum0** and **plotnum1** can have as their values lists of integers, whose length is equal to the number of surfaces in the call to **plotsurf**. **plotsurf** plots the first surface using the first elements of the lists **plotnum0** and **plotnum1**, and so forth.

plotnum2 *default:* **plotnum2** *Option Variable*

The number of plot points in the vertical direction for **plot3d_vect**. If **plotnum2** is not specified, the value of **plotnum1** is used.

1.3.2 Coordinate Transformations

1.3.2.1 Pre-Defined Coordinate Transformations

Six types of coordinate grids are recognized by the two dimensional plotting functions. Their names are **polar**, **log**, **linlog**, **loglin**, **lin**, and **loglog**. If one of **polar**, **log**, **linlog**, **loglin** or **loglog** appear as an optional argument to one of the two-dimensional plotting functions, then the plots appear on the appropriate scale.

Grid Type	Description
polar	Polar coordinate system. The arguments of the plotting functions now are used to denote θ and r instead of x and y respectively.
log or **loglog**	Causes both axes to appear on a $\log_{10}$ scale.
linlog	Causes the y axis to appear on a $\log_{10}$ scale.
loglin	Causes the x axis to appear on a $\log_{10}$ scale.
lin	Both x and y scales are linear.

Examples

Plot a circle in Figure 1.15, page 26.

```
(c1) equalscale:true$
(c2) plot(1, t, 0, 2*%pi, polar)$
```

Replot the object on a linear scale, which gives a horizontal line in Figure 1.16, page 26. (After generating this plot, we thickened the line using the graphics controls in the Macsyma Front End so you can more easily distinguish it from the coordinate grid lines.)

```
(c3) replot(true,lin), equalscale:false$
```

Plot an exponential on a log-linear scale in Figure 1.17, page 27:

```
(c4) plot(exp(x), x, 0, 10, linlog)$
```

This example plots some points on log-log scale in Figure 1.18, page 27. The slope shows that $y \sim \sqrt{x}$.

```
(c5) graph([1,2,5,10,50,100], [1,1.5,2.5,3,7,10], log);
```

When a plot is created the untransformed point values of the plotted object are saved, along with the type of coordinate transformation. The coordinate transformation is reapplied each time you **replot**. You can change the type of transformation either by using the command **replot**. Specifying **lin** as the second argument to **replot** results in no coordinate transformation being performed.

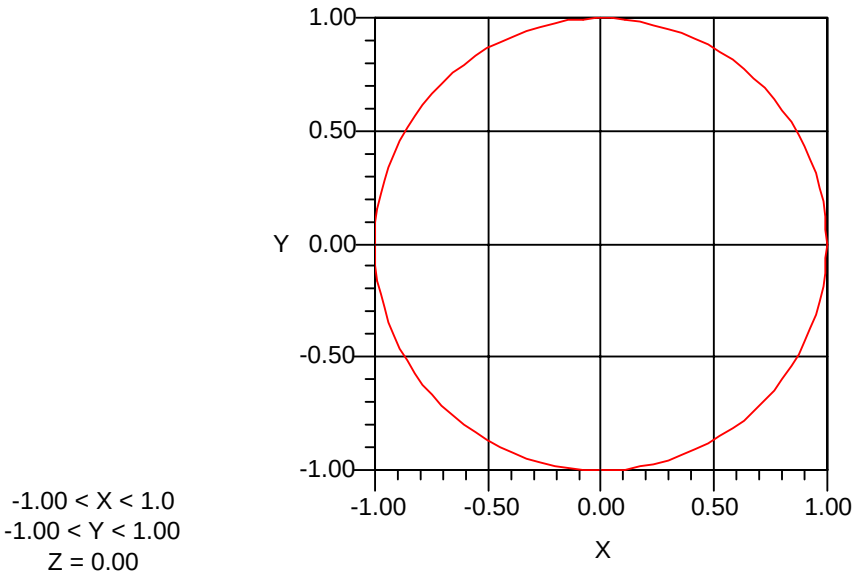

Figure 1.15: Plotting in polar coordinates

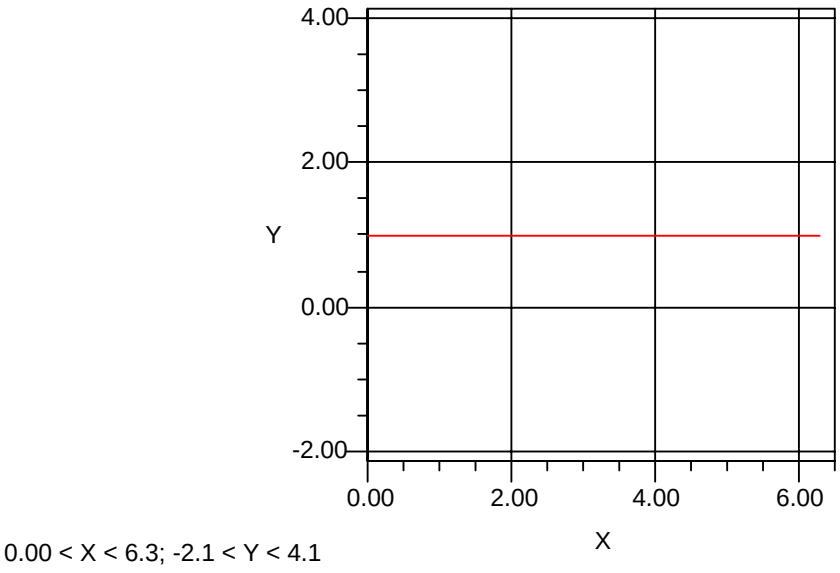

Figure 1.16: Replot the previous plot with a different coordinate transformation

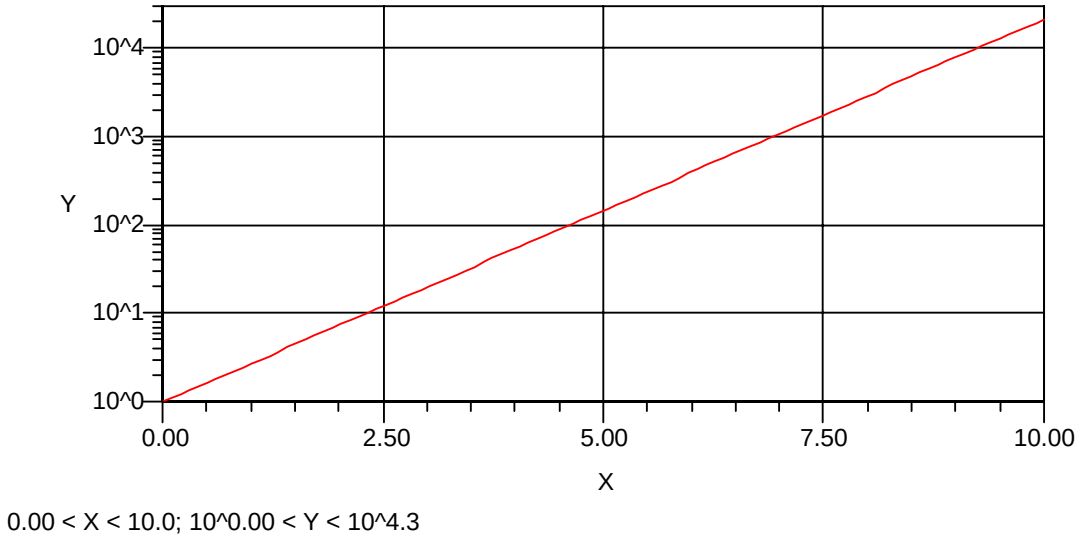

Figure 1.17: Plot using linear and logarithmic scales

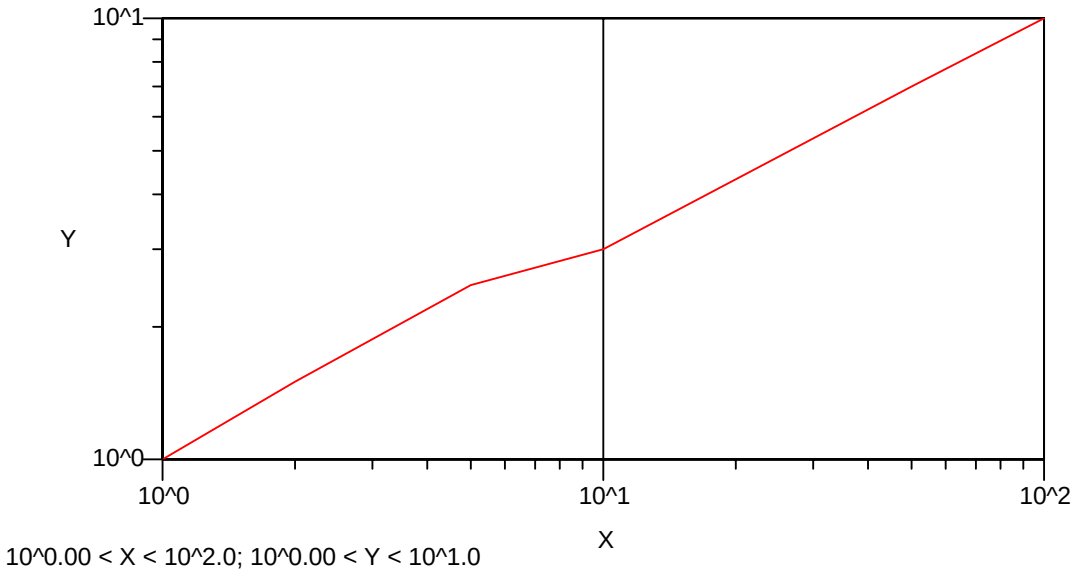

Figure 1.18: Plot using log-log scaling

The coordinate transformation can also be changed from a Macsyma break.

1.3.2.2 User-Defined Coordinate Transformations

Two Dimensional Transformations If the optional argument **special** appears in the call to **plot**, then immediately before displaying your data, **plot** looks at the values of **xfun** (*default:* **false**) and **yfun** (*default:* **false**). If the values are **false** the data is scaled in the normal way and is plotted. If either value is non**false**, then it should be the name of a function of one to three arguments which defines a transformation between the data and the x and y screen coordinates. The standard linear scaling is still applied to the result of this transformation, to make your plot fit on the screen. The functions must be translated or compiled and must return a floating-point result.

The way they work is best illustrated by an example. The predefined functions **polarx** and **polary** are equivalent to:

```
polarx(x,y):=y*cos(x)$
polary(x,y):=y*sin(x)$
```

We can set **xfun** and **yfun** to these two functions respectively, as shown in the example below. Thus x and y become the θ and r coordinates of a polar system. This forces a transformation just prior to plotting in Figure 1.19, page 28:

```
(c1) xfun:polarx$
(c2) yfun:polary$
(c3) plot(8+sin(8*x), x, 0, 2*%pi, special), equalscale:true$
```

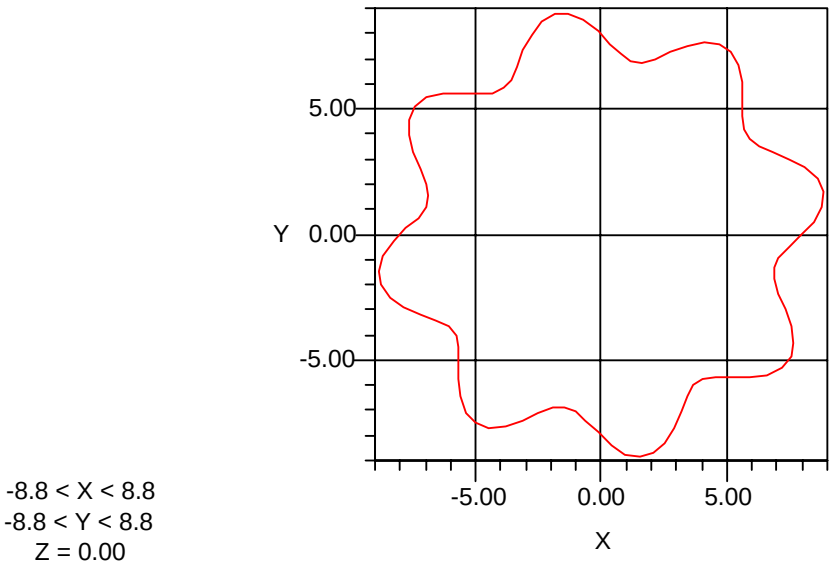

Figure 1.19: Plot which uses built-in transformation to polar coordinates

This command sequence produces **plotnum** x data points, ranging from 0 to 2π and **plotnum** y data points, each having value 1 in this case. These x and y values are given to the **polarx** and **polary** functions as the first and second arguments. The values of these two functions are then supplied to the normal scaling routines. Thus this **plot** command would produce a circle if `equalscale:true`, otherwise an ellipse would be produced. This plot could of course also be generated using the **polar** optional argument to the **plot** command.

The arguments of **xfun** and **yfun** are shown in Table 1.1, page 29.

Function	One Arg	Two Args	Three args
xfun	x	x, y	x, y, z
yfun	y	x, y	x, y, z

Table 1.1: Arguments of **xfun** and **yfun**

The first argument to **plot** is interpreted as a y variable and its second argument as an x variable. For **paramplot** and **graph** the first arguments are x variables and the second arguments are y variables. For all the two dimensional plotting functions the z variable is 0.0. (See Section 1.3.2.2, page 29 for the coordinate conventions for the three dimensional plotting functions).

polarx and **polary** are predefined in the **plot** files. Other predefined **xfun**s and **yfun**s that may be of use are shown below. The **mode_declare**s have been omitted.

```
plot_log10(x) := if x=0.0
           then -90.0
           else log(abs(x))/log(10.)$
reflect(x)     :=-x$
ytox(x,y)      := y$
xtoy(x,y)      := x$
ztoy(x,y,z)    := z$
ztox(x,y,z)    := z$
rotatex(x,y)   := x*cosang - y*sinang$
rotatey(x,y)   := x*sinang + y*cosang$
initrotate(ang):=(cosang:cos(ang),sinang:sin(ang))$)
```

sinang and **cosang** are set up by **initrotate**.

Example

In this example Figure 1.20, page 30, the **xfun** and **yfun** cause x and y to be swapped. Thus the plot of $\sin x$ actually displays $\sin^{-1} x$ vs. x.

```
(c1) xfun:ytox$
(c2) yfun:xtoy$
(c3) plot(sin(x),x,-%pi/2,%pi/2,special)$
```

The **xfun**s and **yfun**s implied by the optional arguments are shown in Table 1.2, page 29.

Argument	**xfun**	**yfun**
polar	**polarx**	**polary**
log	**plot_log10**	**plot_log10**
linlog	**false**	**plot_log10**
loglin	**plot_log10**	**false**
lin	**false**	**false**

Table 1.2: Specifying **xfun** and **yfun** in **plot**

Three Dimensional Transformations The perspective transformations work by the same **xfun** and **yfun** mechanism documented above. If you want to change the transformation, here are the names of the functions used, although it is probably quite easy to confuse the function that figures out the hidden lines. The hidden line routine also makes use of the variable **howclose**. **howclose** should evaluate to the name of a function of three arguments (x, y and z) and should give a measure of how close the point [x, y,z] is

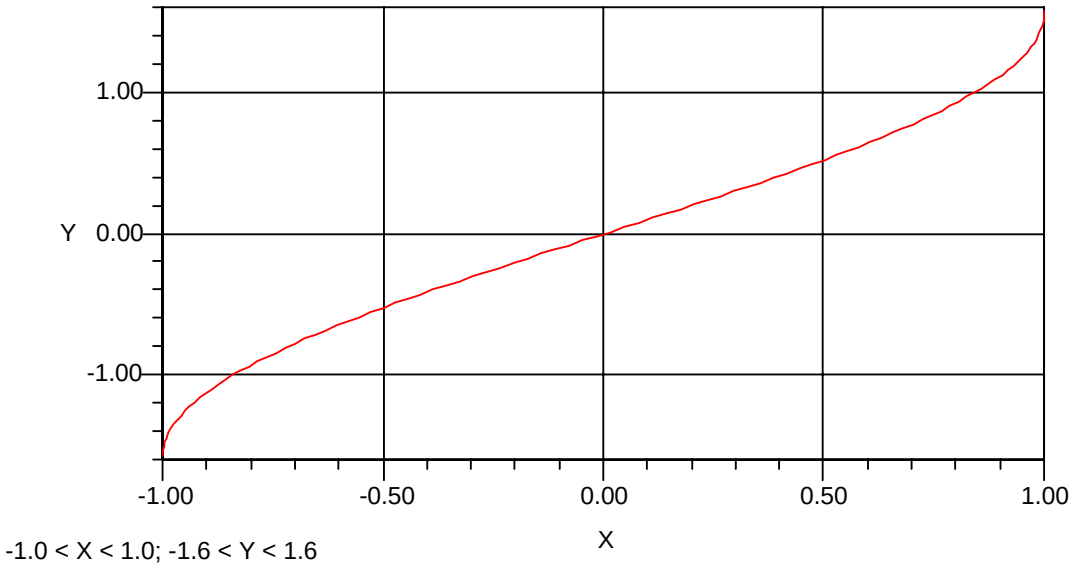

Figure 1.20: Plot using built-in transformation to switch X and Y

to the viewer. It is not necessary that this function return the exact distance to $[x,y,z]$ but it should be a monotonically increasing function of that distance.

The option variables **centerplot**, **perspective** and **reverse** determine which of the functions **xfun**, **yfun** and **howclose** are used. Table 1.3, page 30, shows the names of those functions that are used for particular settings of these option variables.

centerplot	**perspective**	**reverse**	**xfun**	**yfun**	**howclose**
nonfalse	true	false	p3dx	p3dy	howclose3d
nonfalse	true	true	p3dxr	p3dyr	howclose3d
nonfalse	false	false	np3dx	np3dy	howclosenp3d
nonfalse	false	true	np3dxr	np3dyr	howclosenp3d
false	true	false	old3dx	old3dy	howcloseold3d
false	true	true	old3dxr	old3dyr	howcloseold3d
false	false	false	oldnp3dx	oldnp3dy	howcloseoldnp3d
false	false	true	oldnp3dxr	oldnp3dyr	howcloseoldnp3d

Table 1.3: Settings for **centerplot**, **perspective**, and **reverse**

initperspec(*cx, cy, cz, vx, vy, vz*) *Function*

For three dimensional perspective plots, the type of view is specified by doing `initperspec(cx, cy, cz, vx, vy, vz)$` where $[vx, vy, vz]$ is the **viewpt** and $[cx, cy, cz]$ is the **centerplot**. The optional argument **not3d** causes **yfun** to be bound to **ztoy**.

1.3.3 Superimposing Plots

1.3.3.1 Basic Command for Combining Plots

combine_plots is the simplest way to superimpose plots.

combine_plots(*list_of_plots* {, *args*}) *Function*

Combines the plots in the list *list_of_plots* into one plot. The plots must be all two dimensional or all three dimensional. *list_of_plots* can contain plot commands or strings which evaluate to plot commands.

combine_plots is a more convenient way to combine plots than the older graphics switches **clear**, **wait**, **first**, **same** and **last**.

Do `example(combine_plots);` for an example.

1.3.3.2 The "Clear/Wait" Scheme

You can superimpose plots by governing when the plotting commands clear the screen. Two option variables must be reset to accomplish this:

clear *default:* **true** *Option Variable*

If this option variable is set to **true**, the screen is cleared before beginning a new plot. If this option variable is set to **false**, the new plot is superimposed on the most recently generated plot. Clearing occurs just before a plot is done, so setting **clear** to **false** before generating the superimposed plots prevents the previous plot from being cleared.

wait *default:* **true** *Option Variable*

If this option variable is set to **true**, Macsyma waits for you to send a continuation signal (type a space character in older versions of Macsyma) before processing can continue after a plot is produced. This setting is intended to allow you time to view the plot. When **wait** is set to **false**, Macsyma continues to the next command without waiting for a continuation response from the user.

These two option variables should be set to **false** for sequential superposition of a set of plots. The example below illustrates the use of **wait** and **clear** to superimpose four plots. The example begins by enabling clearing and disabling waiting. This assures that the screen starts fresh, and that the superposition can proceed without pausing. After the first plot is displayed, clearing is disabled. Plotting continues, until just before the last plot, when waiting is restored. After the last plot, clearing is restored.

Example

```
(c1) (clear:true, wait:false,
      plot(...),
      clear:false,
      plot(...),
      graph(...),
      wait:true,
      plot(...),
      clear:true)$
```

1.3.3.3 The "First/Same/Last" Scheme

The **first**/**same**/**last** scheme for superimposing plots is more convenient than the **clear**/**wait** scheme for most purposes. All three of these keywords can be used as keyword arguments to nearly all the plotting functions and to **replot**.

The **first**/**same**/**last** scheme cannot be used with the plotting commands **plotsurf** and **paramplot3d**. Use the **clear**/**wait** scheme with these commands.

first *Keyword*

Keyword for: **plot, graph, paramplot, contourplot, replot, plot3d**

When given as a keyword argument to a plotting function, **first** is used to indicate that the plot is to be the first of a series of plots that are to be superimposed or plotted in different parts of the screen. It has the effect of setting **clear** to **true** and **wait** to **false** before plotting and **clear** to **false** after plotting. This option variable is in effect during the plotting process only. It does not become a property of the plot itself.

same *Keyword*

Keyword for: **plot, graph, paramplot, contourplot, replot**

This keyword forces the plotting functions to use the same scale and window as the previous plot. It suppresses display of the axes, the date and the plot bounds. The intention is to make superposition of plots convenient. Specification of a title and labels is allowed, but if the previous plot which defines the scale of this plot did not have them they may appear in the wrong place, wrapped around the screen. If you want to specify a title, then you should probably specify a null title `" "` in the original plot. This option variable is in effect during the plotting process only. It does not become a property of the plot itself.

last *Keyword*

Keyword for: **plot, graph, paramplot, contourplot, replot, plot3d**

When given as a keyword argument to a plotting function, **last** is used to say that this plot is to be the last of a series of plots that are to be superimposed or plotted in different parts of the screen. It has the effect of setting **wait** to **true** before plotting and **clear** to **true** after plotting. This option variable is in effect during the plotting process only. It does not become a property of the plot itself.

Examples

This example illustrates how to superimpose two curves using one plotting command in Figure 1.21, page 32.

```
(c1) plot([x+1,x^2+1], x, -1, 1)$
```

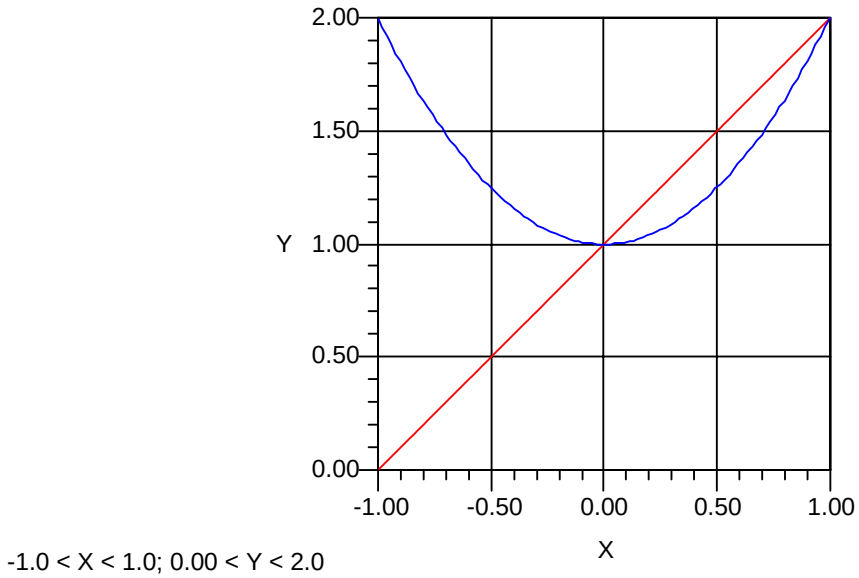

Figure 1.21: Plotting two curves with one plotting command

To superimpose plots generated with separate plotting commands, the normal sequence is:

```
(plot(..., first),
 plot(..., same),
 plot(..., same),
 ...
 plot(..., same, last)
```

This example, Figure 1.22, page 33, superimposes three curves, and uses line colors (which do not show in this book) and plot point symbols to distinguish the curves. The option variables **ymin** and **ymax** are used to clip the plot of the tangent function.

```
(c1) plot(sin(x),x,0,2*%pi,first)$
(c2) plot(cos(x),x,0,2*%pi,same,[1])$
(c3) plot(tan(x),x,0,2*%pi,same,last,[510]), ymax:4, ymin:-4$
```

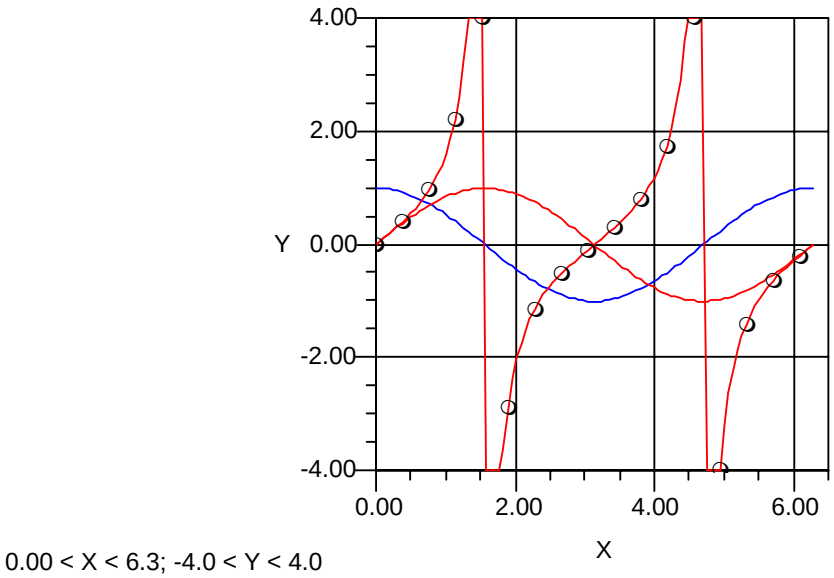

Figure 1.22: Superimposition of plots using the first/same/last scheme

1.3.3.4 Merging Plot Files

merge_plot_files is available only in versions of Macsyma with file-based graphics, which includes Macsyma 419 and successors but not Macsyma 2.0 and successors.

merge_plot_files(*[infile1, infile2]*) *Special Form*

merge_plot_files(*[infile1, infile2, ...], outfile*) *Special Form*

merge_plot_files(*[[infile],[[tx,ty,tz], [sx,sy,sz]]], infile2], outfile*) *Special Form*

This command merges plot files into a new file. The resulting image is an overlay of the input plots using range and perspective information from the aggregate. This function takes 1 required and 1 optional argument.

The required first argument is a list of infile entries. Each entry may be either the name of a file or a list consisting of a filename followed by a transformation list. The transformation list consists of a translation and a scaling, each specified by a list of 3 floating point numbers, one for each axis. The

effect is to transform the infile data first by the translation then by the scaling, before merging it in the outfile. The default translation is $[0, 0, 0]$ and the default scaling is $[1, 1, 1]$.

The optional second argument, *outfile*, is specified as a file name or file pathname in double quotation marks. If not supplied, *outfile* defaults to `plot_file`. When file names are used without pathnames, Macsyma assumes that the files are located in the directory *plot_directory* (which defaults to `macsyma:plots;`) with a filetype of *plot_file_type* (which defaults to `plt`).

1.4 Changing Plot Appearance

There are usually three ways to change the appearance of a plot in Macsyma.

- You can alter the plot under program control (from the keyboard, or in a batch file) at the time it is generated, using the various plotting commands and option variables in Macsyma.
- You can alter the appearance of the plot after it is generated, using the graphics editing controls in the Macsyma Front End.
- You can alter the appearance of the plot after it is generated using graphics styles in the Macsyma Front End. You can specify a graphics style under program control.

This section principally discusses the programmatic methods of altering plot appearance. You can find more information about graphics editing controls and styles in the Macsyma Front End in the *Scientific Notebook Interface Reference Manual*.

1.4.1 Changing the View of the Plot

1.4.1.1 Changing Plot Scale and Perspective

By default, Macsyma scales plots automatically to be as large as possible, while still allowing plots to fit comfortably in the available screen area. Macsyma provides two ways to change the scaling of plots with keyboard commands.

plot_size *default:* **75** *Option Variable*

Determines the size of the plotted object as a percentage of the linear size of the plotting window in the Macsyma Front End. **plot_size** must be an integer between 2 and 200 inclusive. See also `Scale: Initial Height %` and and`Scale: Initial Width %` The Attribute `Notebook: Graphics Height` has a default value of 4 inches.

By default, Macsyma scales plots by different factors in different dimensions, so that plots fill most of the available screen area. While for many purposes this is desirable, sometimes you want the different directions in a plot to be scaled equally (for example, so that a circle appears as a circle and not as an ellipse). The option variable **equalscale** controls the relative scaling of different directions in plots.

equalscale *default:* **false** *Option Variable*

If this option variable is set to **true**, **plot** ensures that the scales are the same in both directions. Thus if the **window** is rectangular, and **equalscale** is **false**, a circle appears as an ellipse, whereas with `equalscale:true;` it appears as a circle.

Macsyma 2.0 and successors enable you to change the scale, perspective, viewpoint and many other attributes of a plot after the plot is generated, using the controls in the `Graphics | Camera View` dialog in the Macsyma Front End. **equalscale** corresponds to the attribute settings in the Macsyma Front End shown in the Table 1.4, page 35.

Dimensions	equalscale	World Scaling	Window Scaling
2D	false	Normalized	Fit to pane
2D	true	None	Isotropic
3D	false	Normalized	Isotropic
3D	true	None	Isotropic

Table 1.4: Settings for **centerplot**, **perspective**, and **reverse**

1.4.1.2 Changing Viewpoint and Orientation

The option variables **viewpt**, **centerplot**, **plot_roll**, **perspective** and **reverse**, allow you to govern the perspective view of a plot.

viewpt *default:* **viewpt** *Option Variable*

viewpt can be set to a list of three numbers which specify the rectangular coordinates of the viewpoint from which the observer views the plot. A projection of the plotted object is made onto a plane perpendicular to the line of sight, which is a line joining the points **viewpt** and **centerplot**. After a plot is generated, the value of **viewpt** which was used to generate the plot is stored in the system variable **viewpt1**.

centerplot *default:* **centerplot** *Option Variable*

centerplot can be set to a list of three numbers which specify the rectangular coordinates of a point on the line of sight. The plotted object is projected with parallel rays onto a plane perpendicular to a line joining **viewpt** and **centerplot**. After a plot is generated, the value of **centerplot** which was used to generate the plot is stored in the system variable **centerplot1**.

When a plot is first generated, the default distance between the viewpoint and the object is determined to give a good perspective view of the object. After the plot is generated, changing **viewpt** will change the viewpoint without changing the effective centerplot, so that the distance between the viewpoint and the object changes. If you change **centerplot**, the default algorithm is again applied to determine the viewing distance.

plot_roll *default:* **0.0** *Option Variable*

This is the angle of rotation (in degrees) of a plotted image about the line of sight. When using the default viewpoint (**viewpt**) in the first octant, a value of zero means that the $x - y$ plane is horizontal.

If **viewpt** and **centerplot** are unbound, the default, then effective values of these variables are chosen as follows.

1. The extreme values of the coordinates are determined. This gives the two points `min:[`*xmin*`,`*ymin*`,` *zmin*`]`, `max:[`*xmax*`,` *ymax*`,` *zmax*`]`.
2. **centerplot** is chosen as $(min + max)/2$
3. **viewpt** is chosen as $max + 3(max - min)$.
4. The view is chosen so that the z axis is vertical, the x axis is increasing towards you to the left and the y axis is increasing towards you to the right.

Example

This plot was shown earlier to illustrate the use of **plot3d**, Section 1.10, page 19.

```
(c1) plot3d(exp(-x^2-y^2)*x,x,-2,2,y,-1.5,2.5)$
```

Move the viewpoint.

```
(c2) viewpt:[-10,10,0.5]$
```

```
(c3) replot();
(c4) reset(viewpt)$
```

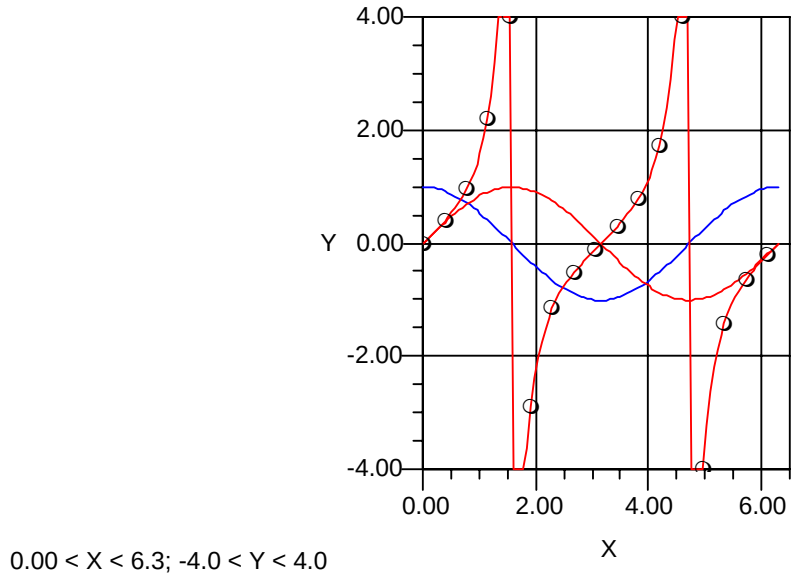

Figure 1.23: Plot with adjusted viewpoint

Before you set either **viewpt** or **centerplot**, you might want to look at the values of **viewpt1** or **centerplot1**, the values that **plot3d** assumed in doing the previous plot. Likewise **xmax3d**, **ymin3d** etc. are the values of **xmax**, **ymin** used by **plot3d** to determine the default **viewpt** and **centerplot**.

perspective *default:* **true** *Option Variable*

If this option variable is set to **false**, it causes **plot3d** to use a nonperspective view. This is equivalent to extending the viewing position out to infinity along a line connecting **viewpt** and **centerplot**.

Macsyma 2.0 and successors enable you to alter the viewpoint, center, scale and many related attributes of a graphic after it has been generated, using graphics controls in the Macsyma Front End.

1.4.2 Appearance of Plotted Points, Lines and Surfaces

1.4.2.1 Line Types, Line Colors, and Plot Symbols

If a list (or an atom that evaluates to a list) appears as an optional argument to a plotting function, then that list is taken as a specification of the line type to be used for the plot. The elements of the list are evaluated once and the elements of the resultant list must be all integers. If this list has the form [$type_1$, $type_2$, ..., $type_i$, ..., $type_k$] then the i^{th} curve is plotted with line of type $type_i$. The $(k+1)^{th}$ curve is plotted with line of type $type_1$ and so on. Omitting this optional argument is equivalent to specifying [0], that is, all the curves are plotted with line of type 0. A line of type 0 is solid line with no symbols.

The curves drawn by the **plot** functions may be drawn as one of nine types of colored and dashed line, with one of nine types of symbol drawn at the data points. In general the line type $type_i$ is a positive integer of the form *abcd* where *a, b, c,* and *d* are digits between 0 and 9, where leading zeroes may be omitted. The four digits have the following meanings:

d Line-type. Line-type 0, the default, is a solid line. Line-type of 9 defaults to no line.

c Symbol-type. Symbol 0 is defined to mean no symbol. Symbol 9 is defined to mean a dot.

b Number of data points with no symbol between the data points with a symbol. If $b = 9$ then b is set to `plotnum-1`. If $b = 98$ then b is set to `plotnum-2`. **plotnum** is taken here to mean the number of point in the plot.

a The first symbol is put at data point number 1 if $a = 0$ and at $b + 2 - a$ otherwise.

Table 1.5, page 37, gives some examples of line and symbol specification.

abcd	Meaning
0	Plain line.
10	Symbol 1 at every point; plain line.
19	Symbol 1 at every point.
99	A dot at every point.
124	Symbol 2 at every other point; line 4.
9874	Line 4 with symbol 7 at end points.
9974	Line 4 with symbol 7 at first point.
19974	Line 4 with symbol 7 at last point.

Table 1.5: Examples of Line and Symbol Specification

Examples

The following commands plot three trigonometric functions with different line types and different plot symbols. *sin(x)* appears with the default line type 0 (solid red), *cos(x)* appears with line type 1 (solid blue), and *tan(x)* appears with line type 0, and plot symbol type 1 at every sixth plot point (skipping five plot points between plot symbols). The plot is shown in Section 1.22, page 33, Figure 1.22, page 33.

```
(c1) plot(sin(x),x,0,2*%pi,first)$
(c2) plot(cos(x),x,0,2*%pi,same,[1])$
(c3) plot(tan(x),x,0,2*%pi,same,last,[510]), ymax:4, ymin:-4$
```

Plot some sinusoidal curves with different colors and different plot point symbols, skipping ten plot points between plot symbols in Figure 1.24, page 38.

```
(c1) plot([cos(3*x)-7,sin(2*x)-4,cos(x),sin(2*x)+4,cos(3*x)+7],
     x,0,2*%pi,[510,521,532,543,554]), ymin:-10, ymax:10$
```

Note: This graph has been enhanced with the Macsyma Front End program to produce better appearing grey scales and symbols rather than color for xerox reproduction. Other graphical attributes have been edited after the plot was generated to change axis labels and locations as well.

Macsyma 2.0 and successors enable you to alter the appearance of plot points and plotted lines after a plot has been generated, using graphics controls in the Macsyma Front End.

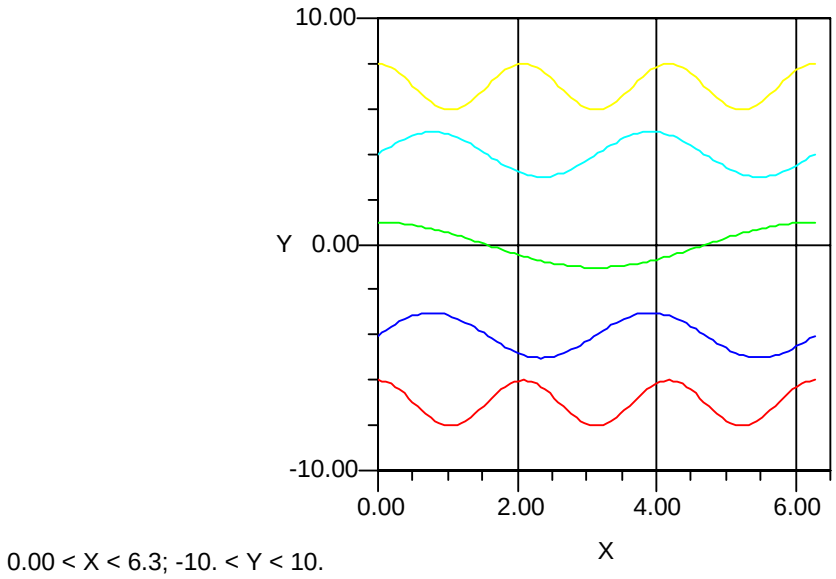

Figure 1.24: Plot of five functions with different line and point styles

1.4.2.2 Surface Colors, Lighting and Mesh

color_function *default:* **color_rainbow_in_z** *Option Variable*

The symbolic name of a function of the coordinates [X1,X2,X3], whose value at each point determines the color to be used at that point. For example,

```
(c1) colorf(X1,X2,X3):= sqrt(x1*X1+X2*X2+X3*X3)$
(c2) color_function:colorf$
```

makes the color hue a function of the radius of each plot point from the coordinate center (0,0,0). As the color hue index increases through the interval [0.,1.], the hue ranges from red through the rainbow back to red. For values greater than 1.0, the fractional part of the color index is used to obtain a value in the range [0.,1.] . In this example, the illumination and intensity of color are set automatically.

In Macsyma 2.0 and successors, **color_function** has these additional features.

- **color_function** is a function of the actual coordinates. If you want the **color_function** to use normalized coordinates, then you should write the function using the variable combinations (*x-xmin1*)/(*xmax1-xmin1*) and so forth. The values of **color_function** are not normalized, but are used as calculated.
- **color_function** can control intensity-hue-saturation (IHS) color coordinates or red-green-blue (RGB) color coordinates. The color function must return a list of three numbers or a list of three numbers plus a symbol.
- In the Macsyma Front End, users can change many aspects of the color distribution of lines and surfaces after a plot has been generated.
- In the Macsyma Front End, users can set the colors and positions of two spot lights and the color of one ambient light.

For example,

- [0.667,0.833,0.50] specifies IHS color coordinates.
- [0.667,0.833,0.5,IHS] specifies the same IHS color coordinates.
- [0.82,0.82,0.918,RGB] specifies the same color in RGB coordinates.

Macsyma 2.0 and successors enable you to change colors, lighting and many other properties of plotted points, lines and surfaces after a plot is generated, using commands in the Macsyma Front End.

1.4.2.3 Colors

Colors of plotted lines are controlled from Macsyma by line type codes of the form "[integer]" as a keyword to the plotting command. See the *Macsyma Reference Manual* for more information.

Colors of plotted surfaces are controlled from Macsyma by the option variable *color_function*.

Macsyma has the following built-in colors.

color_red	color_dark_red	color_orange
color_yellow	color_olive	color_yellow_green
color_green	color_dark_green	color_cyan
color_blue_green	color_blue	color_dark_blue
color_magenta	color_purple	color_white
color_gray	color_black	

Table 1.6: Built-in Colors

color_red *default:* *Option Variable*

color_dark_red *default:* *Option Variable*

color_orange *default:* *Option Variable*

color_yellow *default:* *Option Variable*

color_olive *default:* *Option Variable*

color_yellow_green *default:* *Option Variable*

color_green *default:* *Option Variable*

color_dark_green *default:* *Option Variable*

color_cyan *default:* *Option Variable*

color_blue_green *default:* *Option Variable*

color_blue *default:* *Option Variable*

color_dark_blue *default:* *Option Variable*

color_magenta *default:* *Option Variable*

color_purple *default:* *Option Variable*

color_white *default:* *Option Variable*

color_gray *default:* *Option Variable*

color_black *default:* *Option Variable*

To make a plotted surface all of one color, say magenta, set `color_function: color_magenta` before submitting the plot command.

1.4.3 Changing Bounding Box and Axes

Macsyma 2.0 and successors enable you to alter the appearance of the bounding box and axes in a plot after it has been generated, using graphics controls in the Macsyma Front End.

1.4.3.1 Bounding Box and Clipping Planes

All Macsyma plotting commands accept arguments which specify the ranges of the independent variables which are to used to generate the plotted object. In the case of parametric plotting commands like **paramplot**, **paramplot3d** and **plotsurf**, these arguments define limits of parameters values and do not by themselves define limits to the coordinate ranges.

Macsyma provides option variables for specifying the range of coordinate values for the independent and dependent variables to appear in a plot. Macsyma also reports, as the values of system variables, the limiting values of the coordinates after a plot has been generated.

All of the plotting option variables whose default state is unbound, such as **xmax**, have corresponding system variables whose names are obtained by appending a "1" to the end of the unbound option variable name: **xmax1**. These system variables should be read only, not set, and their values may be loosely defined to be the value that **plot** assumed for the option variable. The value is correct only if **plot** has calculated it.

For example:

Example

```
(c1) plot(x, x, 0, 10.1)$
(c2) ymax;
(d2)                          ymax
(c3) ymax1;
(d3)                          11.0
```

xmax1 *default:* **xmax1** *System Variable*

Contains the maximum value of the x-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **xmax1** has an associated option variable **xmax**. The maximum and minimum values of **xmax** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

xmax *default:* **xmax** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *xmax* of the points it has calculated. However if **xmax** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(xmax);`.

This option variable is especially important when changing the scale of three dimensional plots, because **xmax** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

ymax1 *default:* **ymax1** *System Variable*

Contains the maximum value of the y-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **ymax1** has an associated option variable **ymax**. The maximum and minimum values of **ymax** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

ymax *default:* **ymax** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *ymax* of the points it has calculated. However if **ymax** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(ymax);`.

This option variable is especially important when changing the scale of three dimensional plots, because **ymax** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

zmax1 *default:* **zmax1** *System Variable*

Contains the maximum value of the z-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **zmax1** has an associated option variable **zmax**. The maximum and minimum values of **zmax** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

zmax *default:* **zmax** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *zmax* of the points it has calculated. However if **zmax** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(zmax);`.

This option variable is especially important when changing the scale of three dimensional plots, because **zmax** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

xmin1 *default:* **xmin1** *System Variable*

Contains the maximum value of the x-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **xmin1** has an associated option variable **xmin**. The maximum and minimum values of **xmin** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

xmin *default:* **xmin** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *xmin* of the points it has calculated. However if **xmin** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(xmin);`.

This option variable is especially important when changing the scale of three dimensional plots, because **xmin** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

ymin1 *default:* **ymin1** *System Variable*

Contains the maximum value of the y-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **ymin1** has an associated option variable **ymin**. The maximum and minimum values of **ymin** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

ymin *default:* **ymin** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *ymin* of the points it has calculated. However if **ymin** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(ymin);`.

This option variable is especially important when changing the scale of three dimensional plots, because **ymin** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

zmin1 *default:* **zmin1** *System Variable*

Contains the maximum value of the z-coordinate of the most recently produced plot, computed after the transformation functions were applied, and after rounding to the nearest tickmark. **zmin1** has an associated option variable **zmin**. The maximum and minimum values of **zmin** that **plot** obtains are rounded up and down to the nearest tick mark, unless **ticknum** is negative. See Section 1.4.3.2, page 45.

zmin *default:* **zmin** *Option Variable*

In choosing the scale for a plot, **plot** looks at the maximum and minimum values of *zmin* of the points it has calculated. However if **zmin** has a numeric value then that value is used instead of the one found from the points themselves. To get back to the default value, type `kill(zmin);`.

This option variable is especially important when changing the scale of three dimensional plots, because **zmin** refers to the value after the perspective transformations, and so may be quite unrelated to the data points.

zmax1 *default:* **zmax1** *System Variable*

zmax1 is of use only for contour plots. It is equal to **zmax** if it has a value, or to the maximum value of the first argument to **contourplot**, if **zmax** is unbound. Its value is meaningless if **contours** is a list, because **contourplot** does not need to compute **zmax** in that case.

zmax *default:* **zmax** *Option Variable*

This option variable is useful for controlling contour plotting. It can be set to the maximum value of z for which a contour is desired. Its value is ignored if **contours** is a list, because **contourplot** does not need to compute **zmax** in that case.

zmin1 *default:* **zmin1** *System Variable*

zmin1 is of use only for contour plots. It is equal to **zmin** if it has a value, or to the minimum value of the first argument to **contourplot**, if **zmin** is unbound. Its value is meaningless if **contours** is a list, because **contourplot** does not need to compute **zmin** in that case.

zmin *default:* **zmin** *Option Variable*

This option variable is useful for controlling contour plotting. It can be set to the minimum value of z for which a contour is desired. Its value is ignored if **contours** is a list, because **contourplot** does not need to compute **zmin** in that case.

The functions **xmax3d**, **xmin3d**, **ymax3d**, **ymin3d**, **zmax3d**, and **zmin3d** are of use only for three dimensional plots. These are the values of the three dimensional data before transforming it to two dimensions. This information is valid only if **plot3d** calculated it to determine either the default **viewpt** or the default **centerplot**.

The functions **viewpt1** and **centerplot1** are also only of use with three dimensional plots. They are useful to look at before you change **viewpt** or **centerplot**.

Examples

Consider the following plot Figure 1.25, page 43, of a function with a singularity at the origin. Because **plotnum0** and **plotnum1** default to even numbers, there is no plot point at the singularity, and the plot has a finite height.

```
(c1) plot3d(1/(x^2+y^2),x,-3,3,y,-3,3)$
```

The option variable **zmax** can be set to clip this plot at a specified height as shown in Figure 1.26, page 43.

```
(c2) zmax:2;
(d2)                                2
(c3) replot()$
```

Macsyma 2.0 and successors enable you to change many properties of the bounding box and axes in plots after a plot is generated, using controls in the Macsyma Front End.

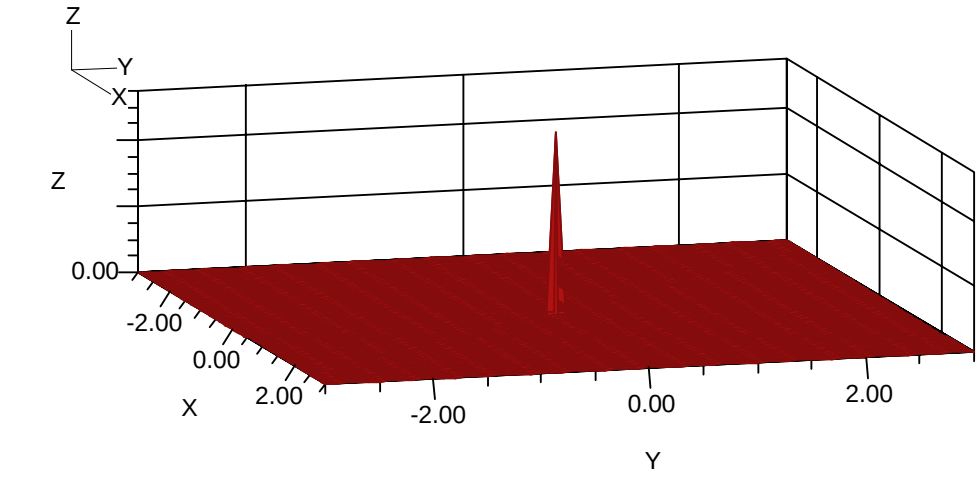

Figure 1.25: Plot of a singular function

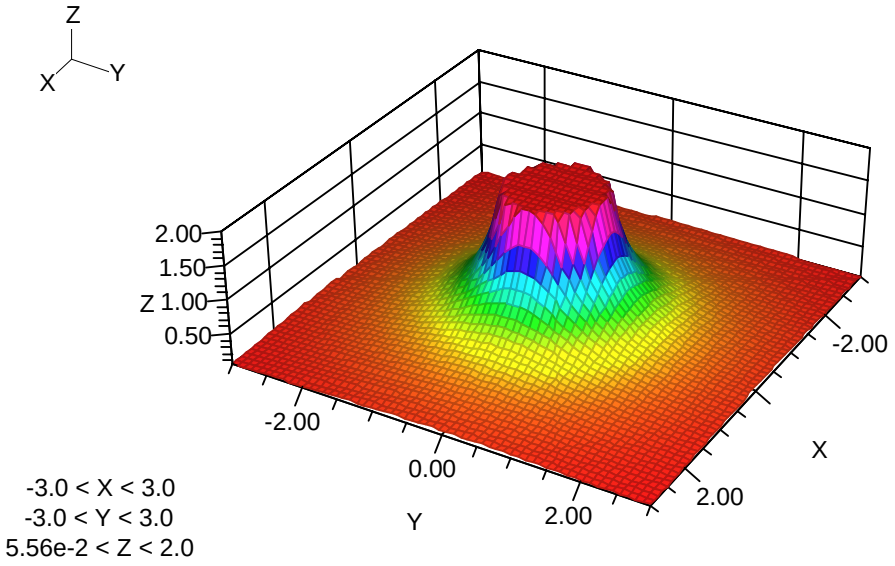

Figure 1.26: Plot of a singular function which is truncated at z=2

Macsyma 2.0 provides a clipping slider to slice through 3D objects. It is located in the **Camera View** dialog in the Macsyma Front End.

1.4.3.2 Plot Axes, Axis Titles and Axis Numbers

xlabel *default:* **xlabel** *Option Variable*

The value of this option variable can be set to a string, which will appear as the label for the X-axis (the axis for the first variable) in any 2D or 3D plot drawn by Macsyma. See also **label**, page 45.

ylabel *default:* **ylabel** *Option Variable*

The value of this option variable can be set to a string, which will appear as the label for the Y-axis (the axis for the second variable) in any 2D or 3D plot drawn by Macsyma. See also **label**, page 45.

plotnumprec *default:* **2** *Option Variable*

This option variable controls the number of digits of precision displayed to the right of the decimal point in axis labels, bounds labels and contour labels.

Example

The following example Figure 1.27, page 44 defines text titles for the X and Y axes and for the plot. It also shows how **plotnumprec** alters the number of significant digits in the axis label numbers and the bounds numbers.

```
(c1) plotnumprec:4$
(c2) plot(exp(x),x,0,3,"x","exp(x)","Plot of the Exponential Function")$
```

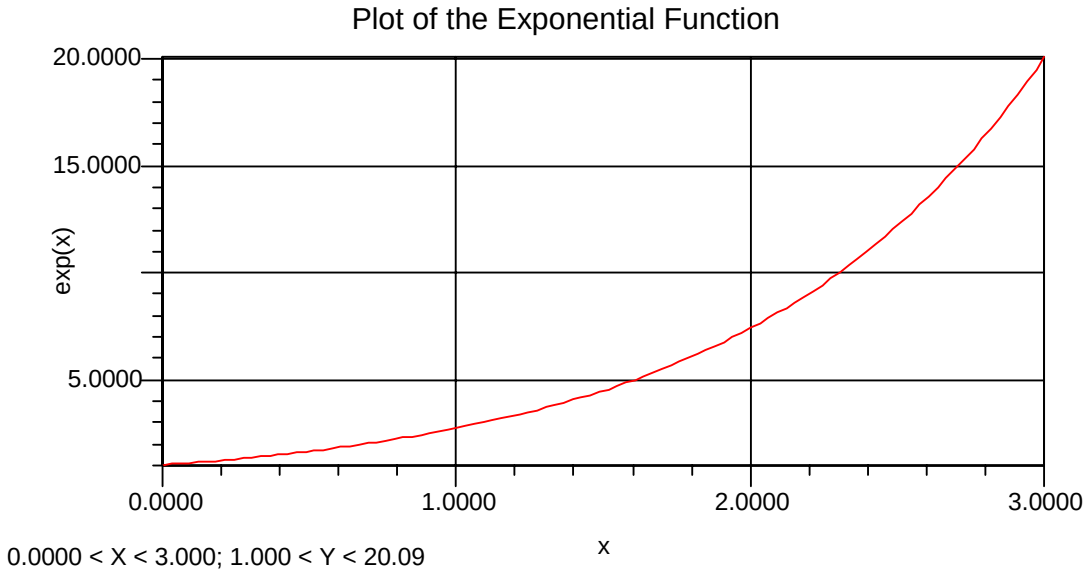

Figure 1.27: Number precision control and titles for X and Y axes

Macsyma 2.0 and successors enable you to alter the number of digits used in each label separately after a plot has been generated, using controls in the Macsyma Front End.

Macsyma 2.0 and successors enable you to edit the axes and axis titles in many ways after generating a plot, using controls in the Macsyma Front End.

ticknum *default:* **10** *Option Variable*

Specifies the number of ticks on each axis. The ticks always come at "nice" values of the x or y variable. "Nice" values are defined to be multiples of $k\,10^n$ where n is an integer and k is 1, 2 or 5. Setting **ticknum** to a negative number causes about `abs(ticknum)` ticks to be drawn, but **plot** will not round the maximum and minimum values on your axes to the nearest tickmark. **ticknum** can be a list of two numbers the first referring to the ticks on the x axis and the second to the ticks on the y axis.

ticknum works in Macsyma 419, but not in Macsyma 2.0, where the Macsyma Front End supplies interactive controls for the number of major and minor tick marks.

1.4.4 Changing Text Labels in Plots

Macsyma enables you to specify a text title for each plot, axis titles, contour labels, and to control the display of bounding coordinate values and other features.

1.4.4.1 Plot Titles and Annotations

If the optional arguments to the plotting functions are not any of the recognized keywords such as **first** or **last**, they can be specifications for the *x-label*, *y-label* and the *title* of the plot. If it is (a) an atom, (b) of the form `'symbol` or `'(...)`, (c) of the form `ev(...);` or `concat(...);` or `label(...);`, then the label is evaluated, otherwise it is used literally.

Note: The evaluation of `'symbl` is `symbl` for all `symbl`, so if you want to get a label of `symbl` and `symbl` has a value, then use `'symbl`. `"symbl"` also works in most instances. See Section 1.4.4.1, page 45.

If one of these is **false**, then nothing is printed. Only as many of the three to be unambiguous need be specified; thus if only a y-label is required then **false**, *y-label* would be sufficient. To avoid confusion with the line-type list, none of these option variables can be a list. See Section 1.3, page 24.

title *default:* **title** *Option Variable*

The value of this option variable can be set to a string, which will appear as the title in any plot drawn by Macsyma. See also **xlabel** and **ylabel**.

label(*'text*$_1$, ..., *'text*$_n$) *Special Form*

This is a special form of **concat** for use with **plot**. It takes any number of arguments and returns an atom containing the arguments concatenated together. It differs from **concat** in the following respects:

- It inserts spaces between the items.
- The arguments need not be atoms.
- If an argument is of the form `'symbl`, then `symbl` appears in the result.
- Otherwise **label** evaluates the argument. If the value differs from the argument, the equation *arg*=*value* appears in the result, otherwise just *value*.

The label is evaluated inside an **errcatch**, so if you have an error in the evaluation of your label, an error message is displayed, but the plot succeeds without a label.

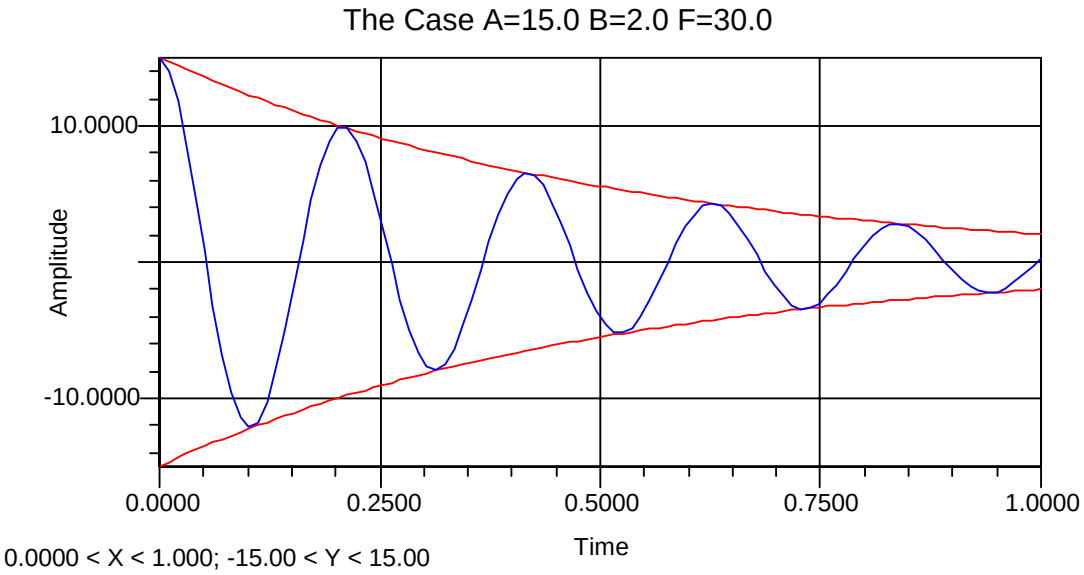

Figure 1.28: Plot with parameter values in the title

Examples

This example Figure 1.28, page 46, produces a plot in which the plot title contains the value of the variable a. The oscillating function and its envelope appear in different colors on a color monitor

```
(c1) (a:15.0, b:2.0, f:30.)$
(c2) plot([a*exp(-b*t),-a*exp(-b*t),a*exp(-b*t)*cos(f*t)],t,0,1,
     "Time","Amplitude",concat("The Case ",label(a,b,f)),[0,0,1])$
```

In the optional arguments to **replot**, use **false** if you do not want to change the corresponding label. If you want to remove a label during **replot**, use **none**. See Section 1.6.1, page 50.

Macsyma 2.0 and successors enable you to edit the plot title text, title font and title locations after generating a plot. It also enables you to add a text caption with a comment about the plot.

1.4.4.2 Contour Labels

labelcontours *default:* **true** *Option Variable*

If this option variable is set to **true**, then **contourplot** and **contourplot** display a legend indicating the value of z for each contour. See also **contours,plotnumprec** and **.**

Macsyma 2.0 and successors enable you to alter the contour labels (but not the number of contours) after the plot is generated, using the controls in the `Graphics | Decorations` dialog.

1.4.4.3 Other Text Labels

plotbounds *default:* **true** *Option Variable*

Plots include a box at the bottom of the plot which displays the maximum and minimum of x and y (and z in 3D plots). Setting **plotbounds** to **false** suppresses display of the plot bounds box.

noprint *default:* **false** *Option Variable*

As a default, the two dimensional plotting functions include a line at the bottom of the plot, called the dataline. The dataline displays the maximum and minimum of x and y. Setting **noprint** to **true** suppresses display of the dataline. In newer versions of Macsyma, this is accomplished by the option variable **plotbounds**.

dateplot *default:* **false** *Option Variable*

If **dateplot** is set to **true**, the date is given at the top right corner of the plot, in a long form if no title is specified, or in a shorter form if a title is specified.

Macsyma 2.0 and successors enable you to alter the bounds box and other text labels after the plot is generated, using the controls in the Macsyma Front End.

1.4.5 Specifying a Graphics Style

plot_style *default:* **normal** *Option Variable*

Name of the default MFE graphics style to apply to a Macsyma graphic in MFE.

See the on-line documentation or the *Scientific Notebook Interface Reference Manual* or the on-line documentation for information about graphics styles. See also your release notes for information about graphics styles in your version of Macsyma.

1.5 Animation

Macsyma 2.0 and successors contain extensive facilities for defining animations of scientific graphics. Animations are of two types:

- Animation of the shape of the plotted object. This can be most easily accomplished using the command **plot_animate**. The **color_function** can also be animated using **plot_animate**.
- Animate many plot attributes which are described by floating point values, such as the viewpoint or color model. These may be animated from the `Graphics | Attribute Editor` dialog in the Macsyma Front End.

In order to start animation, open the `Graphics | Animation` dialog or depress the toolbar button that looks like a rocket ship.

1.5.1 Defining Animations With the Plot_Animate Command

The **plot_animate** command provides a very simple way to animate the movement and change of shape of plotted objects. You simply wrap it around any static Macsyma plotting command to animate it.

plot_animate(*param,lo,hi,'plot_command*) *Special Form*

Animates the *plot_command* with animation parameter *param*, which varies between *lo* and *hi* values during the animation. *plot_command* can any static Macsyma 2D or 3D plot command, where the parameter *param* is included in the plotted expression.

The command **plot_animate** accepts the keyword *cyclic*, which enables animations to piece together smoothly the beginning and end of a periodic animation.

Do `example(plot_animate);` for an example.

plotnum_animate *default:* **10** *Option Variable*

The number of frames in the animation produced by the **plot_animate** command.

Example

The following command produces an animated 3D plot. After executing these commands, you can start the animation by selecting the plot, and clicking on the toolbar button which looks like a rocket ship.

```
(c1) plot_animate(t,0,2*%pi,plot3d(sin(x)*cos(y)*cos(t),x,-,3,3,y,-3,3),cyclic)$
```

1.5.2 Defining Animations in the Macsyma Front End

After a plot has been generated in Macsyma 2.0, it can be animated using plotting controls in the `Graphics | Attribute Editor` dialog in the Macsyma Front End.

Clicking on the `Anim` button exposes the animation editing sections of the dialog. Clicking `Anim` again returns the dialog to its smaller "static" configuration. When expanded only animatable attributes are displayed. Animatable attributes include all Integer, Float, Color, Vertex, and Box (BCube) valued attributes.

Animations are specified as a set of transitions on one or more animatable attributes. All animatable attributes support the three transition functions described below:

Constant Transitions - These consist of a start value and a duration. Constant transitions can be sequenced to created step function like variation in an attribute.

Linear Transitions - These have both start and end values. They transition linearly between start and end values over their duration.

Smooth Transitions - These are similar to linear transitions except the transitions starts out slowly, reaches a maximum rate at the midpoint, and then slows down again as it reaches the endpoint.

To animate an attribute first select it. Then use the Transitions section to select a transition function and add it to the attribute's transition list. After you add the transition select it and use the Duration thumbwheel to set its duration. Add additional transitions to the list as desired. The Transitions list tracks the time interval during which the transition will occur.

When you first add or insert a transition its start and end values are set up based on the start and end values of its temporal "neighbors". You can change these settings by entering new values in the Animation Transition Values section.

On the lower left the Camera Settings group is used to set the total time and frame count for the animation. The time you set in Current Time determines what stage of the animation you see if you press the Preview key. Make sure that the time you set for Total Time is at least as long as the longest transition list you set up. Otherwise all the transitions will not be complete at the end of the Total Time interval.

1.5.3 Playback of Animation Sequences

Once an animation is defined, it can be controlled from the `Graphics | Animation` dialog. This dialog can be opened from the `Graphics` menu which appears when a graphics section is selected, or by clicking on the toolbar button which looks like a movie camera. This dialog has several types of controls.

- A button for building bitmaps. It is generally best to create bitmaps for each frame in an animated sequence. This enables the animation to proceed at higher speed, which becomes independent of the complexity of the image being animated.
- A button to play the animation.
- A button to pause the animation during playback.
- A button to move to the next frame and a button to move to the previous frame in an animation sequence.
- A button to stop an animation.

This `Animation` dialog also contains controls for modifying the behavior of the animation sequence.

- A thumbwheel control for the number of frames in the animation sequence.

  Note: If you wish to change the number of frames in a plot where the shape of the plotted object is animated, then it is usually best to change the number of frames programmatically using the option variable **plot_animate** regenerate the plot using the **plot_animate** command. Altering the number of frames from the `Graphics | Animation` dialog in this situation will generally result in animations which are not smooth, because the controls in the Macsyma Front End cannot generate new data for shapes of the plotted object.
- A thumbwheel control for total time in the animation.
- A thumbwheel control for animation speed. You can alter the number in the control to alter the speed of the animation.
- A checkbox for continuous motion, which makes the animation into an infinitely repeating cycle.
- A checkbox for bouncing off the end of the animation (and returning to time zero by running the animation backwards).
- A checkbox for running the animation in reverse.

You can also run an animation by depressing the toolbar button which looks like a rocket ship. This method of playing back an animation does not give you access to any of the other controls in the `Graphics | Animation` dialog.

1.6 Screen Display, Files and Hardcopy

1.6.1 Screen Display and Redisplay

replot(*plotname, 'arg*$_1$*, ..., 'arg*$_n$) *Special Form*

Replots the plot named *plotname*. Either `replot(true);` or `replot();` will replot the last plot. The *arg*$_1$, ..., *arg*$_n$ are optional arguments. If any of the optional arguments have been changed since the plot was saved, the new values of the optional arguments are used.

If the arrays of *plotname* were destroyed either by a `killplots(`*plotname*`);` or by **save**-ing *plotname* and **loadfile**-ing it into another Macsyma, **replot** checks to see if *plotname* was ever saved using the **saveplots** command if it was it finds the *filename* under which it was saved and then attempts a `loadplots(`*filename*`);` (but taking care not to change the current file defaults) and replots *plotname* if it was found in the file.

The optional arguments can be any of the optional arguments to the main plotting functions. These optional arguments are merged in with those supplied to the original plot. The exceptions are **first**, **last** and **same**, **top**, **bottom**, **left** and **right**, and **dont**. See Sections 1.3.3.3, 1.3, and 1.6.1. These optional arguments do not stick around with the plot. If you want to replot the last plot with some optional arguments supplied you must use the fully specified form `replot(`*plotname*`,` *arg*$_1$`, ...,` *arg*$_n$`);`. The form `replot(true,` *arg*$_1$`, ...,` *arg*$_n$`);` will not work because it uses the first optional argument as a *plotname*.

There are two additional optional arguments that you can give to **replot**, but not to the original plotting function. They are **none**, to suppress a label and **lin**, to suppress a transformation. See Sections 1.4.4 and 1.3.2.

Examples Figure 1.29, page 50, shows a sample curve in rectangular coordinates.

```
(c1) plot(t*cos(3.*t),t,0,19/6*%pi), plotnum:250$
```

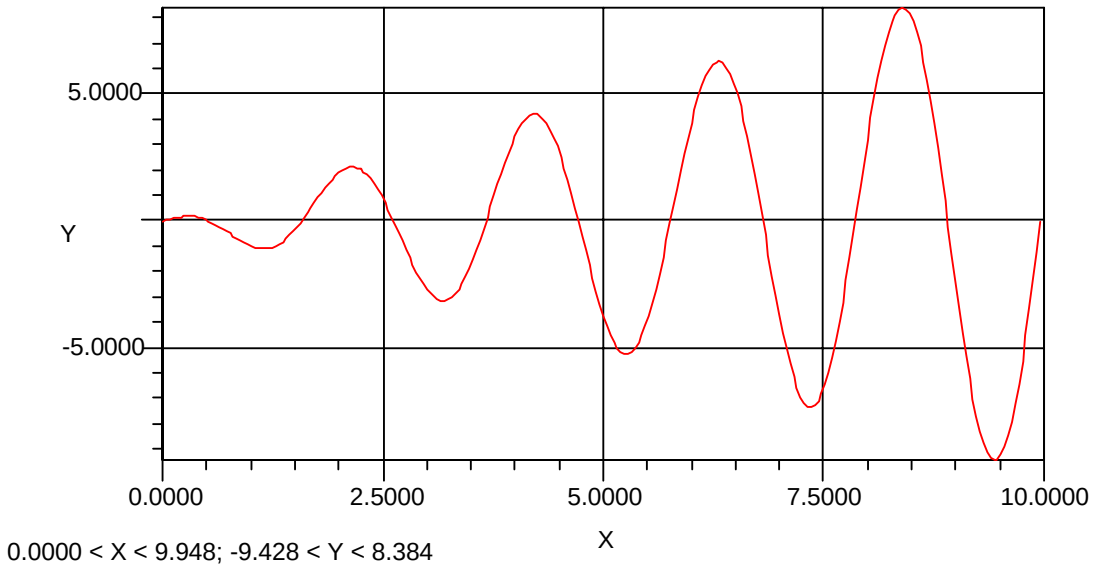

Figure 1.29: Plot of a function in rectangular coordinates

Now replot the Figure 1.29, page 50, in polar coordinates (see Section 1.3.2.1, page 25) and with `equalscale : true;`. The resulting plot is Figure 1.30, page 51.

```
(c2) replot(true,polar), equalscale:true$
```

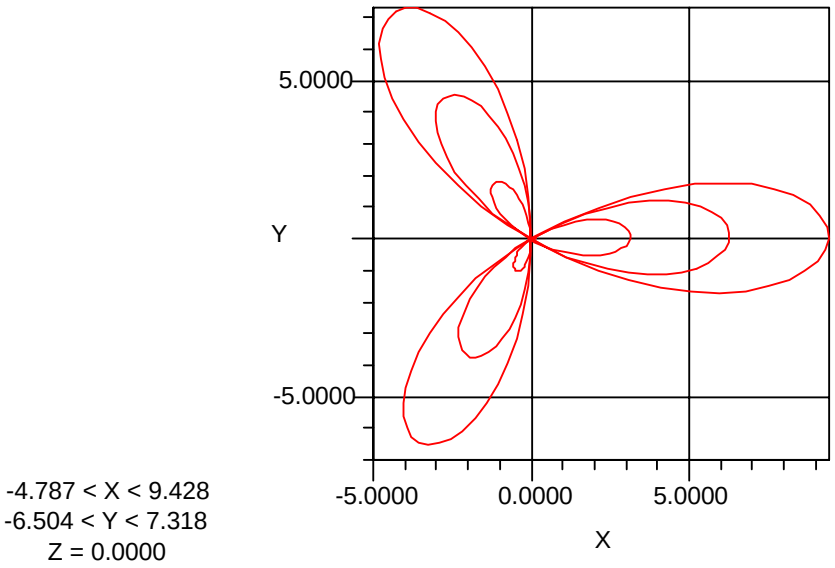

Figure 1.30: Replot of the previous plot using polar coordinates

Note: In certain circumstances, the Macsyma Front End computes a three-dimensional coordinate transformation, and replots its projection in the original coordinate plane, if it is two dimensional.

plot_now *default:* **true** *Option Variable*

If a plotting command is given, Macsyma computes the plot data (and, in versions with file-based graphics, stores it in a file). If **plot_now** is **true**, then Macsyma draws the image in a plot window immediately. If **false**, Macsyma creates the plot file, but does not display the plot. The plot can be displayed using **replot** (or **redraw_plot_file** in versions with file-based graphics).

dont *Keyword*

Keyword for: **plot, graph, paramplot, contourplot, replot**

If **dont** appears as an optional argument to the plotting functions, then the points are calculated, but not plotted. This is primarily of use for creating plots that are to be named and used later. **dont** is a local option variable, like **first** and **last**, and so it is not saved when the plot is named. Thus `plot(exp(x),x,-1,1,dont); replot();` plots something. Setting **plot_now** to **false** has the same effect for all the plotting commands.

1.6.2 Hardcopying Plots

To make a hardcopy of a plot, select the graphics section you wish to plot and

Create a new empty notebook.

Select the notebook section containing the plot you wish to print. Click on the menu command `Edit | Copy Section`.

Select the new empty notebook. Click on the menu command `Edit | Paste Section`.

Click on the menu command `File | Print` and provide the information requested to print the plot.

You may also wish to review Options for Printing Attributes or Options for Graphical Attributes.

1.6.3 Saving Plots in Files

You can save any graphic produced by Macsyma in a Macsyma notebook file with this procedure.

Create a new empty notebook.

Select the notebook section containing the plot you wish to save in a file. Click on the menu command `Edit | Copy Section`.

Select the new empty notebook. Click on the menu command `Edit | Paste Section`.

Save the new notebook in a file by clicking on the menu command `File | Save As`, and assigning the file a name with the filename extension `.mfe`.

You can read this file back into Macsyma by clicking on the menu command `File | Open` and selecting the appropriate file. You can then cut or copy and paste this graphic into any other Macsyma notebook or into any other Windows document.

Note: In order to copy any section of a Macsyma notebook as a Windows metafile, it is necessary to turn on the switch `Clipboard Force Metafile`. This is accomplished by clicking on `File | Option Defaults`, selecting the category called `Notebook`, selecting the notebook attribute `Clipboard Force Metafile`, then turning it on. (If this attribute is not turned on, then Macsyma only copies sections of Macsyma notebooks to other notebooks, and does not create metafiles. Copying sections of Macsyma notebooks is faster if no metafile is created.)

Note: This area of Macsyma is subject to change. Please consult release notes for your version of Macsyma for latest information.

The `Graphics | Export Dialog` enables you to save graphics in four other file formats.

- .BMP files
- .GIF files
- .PCX files
- .RLE files.

1.7 Other Topics in Graphics

1.7.1 Cleaning Up the Plotting Environment

plotreset() *Function*

Resets the option variables affecting plotting back to their default values. The option variables relating to terminal control are not reset.

plot_options *default:* **perspective, ticksize, ticknum, underside, plotnum1, plotnumprec, reverse, equalscale, noprint, plotnum, xaxis, yaxis, contours, xfun, yfun, crosshatch, labelcontours, dateplot, ploterror, plotgap, hardcopy, plotnum0, xmin, xmax, ymin, ymax, zmin, zmax, window, viewpt, centerplot, clear, wait** *System Variable*

This system variable provides a list of the plotting option variables which are re-initialized by the **plotreset** command. The command **ev(plot_options);** will print out a list containing the values of the option variable.

1.7.2 Handling Plot Errors

ploterror *default:* **true** *Option Variable*

Governs what happens when **plot** and most other plotting functions encounter an error in trying to evaluate a function or expression at a plot point.

- If **ploterror** is **true**, the default, then an error occurs.
- If **ploterror** is a floating-point number, then that number is used as the value of the erroneous form.
- If **ploterror** is **false**, then the value of the erroneous form is set to the value of the option variable **plotundefined**. This causes the point to be skipped in the plotting.

1.7.3 Alternate Representations of 3D Plots

Internally, the plots generated by **plot3d** (with or without the **not3d** option variable), **graph3d**, and **contourplot** are the same, and it is possible to choose between any of the four representations. Table 1.7 lists the four representations that are directly accessible.

Representation	Command	Keyword (if any)
hidden-line	**plot3d**	
	graph3d	**hide**
3D, hidden lines not removed	**graph3d**	
	plot3d	**\3d**
contourplot	**contourplot**	
	plot3d	**contour**
not3d	**plot3d**	**not3d**

Table 1.7: Access to Plot Representations

Note: The keyword 3D will also be accepted in all of the following forms: \3d, "3d" or "3D".

The representation can be changed after the plot has been made by using the function **replot**. The following table shows how to specify different representations.

Representation	Command
hidden-Line	`replot(true, hide);`
3D	`replot(true, \3d);`
contourplot	`replot(true, contour);`
not3d	`replot(true, not3d);`

Table 1.8: Controlling Representations From **replot**

Note: There is an interaction between **contourplot** and **not3d**, such that when swapping from a **not3d** representation to a **contourplot**, it is necessary to supply an additional optional argument **lin**. Thus, for example, `replot(true, contour, lin);`. The **true** in the call to **replot** indicates the last plot. See Section 1.6.1, page 50.

1.7.4 Miscellaneous Plotting Commands

plotmode *Option Variable*

An option variable which specifies the format of the plot output. The default and other possible values depend on the machine and (for hardcopy) the output device.

See the *Release Notes* for your Macsyma for the possible values for your version of Macsyma.

plotbell *default:* **true** *Option Variable*

If this option is **true**, the bell on your terminal rings when a plot is finished. To disable the bell, set **plotbell** to **false**.

The bell rings if **wait** is **true** and if you're plotting on the terminal. See Section 1.3.3.2, page 31. The plotting function is now waiting for you to type something before it exits and prints the next C-label on your plot.

This option variable does have any effect in a notebook in Macsyma 2.0 and its successors.

See the *Release Notes* for your Macsyma for any new information.

Chapter 2

The Macsyma Front End Math Engine

MFE itself has the ability to calculate, store, graph and view data associated with a notebook. The MFE math engine is the part of MFE that manages and computes with this data. It is separate and distinct from both the PDEase and Macsyma math engines. Each notebook has its own set of data and names for referencing the data. PDEase uses MFE variables to capture and graph its output. Macsyma can access MFE variables and, amongst other features, can use the MFE math engine to access external numerical libraries.

2.1 Entering Data into the MFE Math Engine

The basic commands to transfer data from the Macsyma math engine to the MFE math engine and MFE variables are:

mfe_put(*mac_var, mfe_var*) *Function*

Places data from Macsyma into an MFE array.

mac_var	Result
scalar	Assigns scalar value to mfe_var
list	Creates 1D MFE array
matrix	Creates 2D MFE array
array	Creates corresponding MFE array
string	Creates 1D MFE array of characters

Macsyma variables must be integers, floating point numbers, or Macsyma strings. They cannot be bigfloats, big integers, complex numbers, rational numbers, block matrices, or any other Macsyma object that does not evaluate to a number or string.

mfe_get(*mfe_var* {, *mac_name*}) *Function*

Places data from the MFE *mfe_var* into the Macsyma environment. The optional argument *mac_name* can be supplied only when getting an MFE array.

mfe_var	mac_name	Result
MFE scalar	(ignored)	returns a scalar
1D MFE array	not given	returns a list filled with values from MFE
2D MFE array	not given	returns a matrix filled with values from MFE
2D MFE array	not given	returns nested list structure
MFE array	given	fills array macs_name with data from and returns the array name *mac_name*

mfe_eval(*command_string* {, *value_flag, mac_name*}) *Function*

Evaluates *command_string* in MFE. If *value_flag* = **true**, then the result of the evaluated MFE command is returned as in **mfe_get**. If *value_flag* = **false**, then no result of the MFE command is returned to Macsyma, and **mfe_eval** returns 'Done.

view_mfe_data(*mfe_name0* {,*mfe_name1, ...*}) *Function*

The MFE object with name *mfe_name0* appears in a DataViewer section in the current notebook. *mfe_namei* can refer to either a 1D or 2D array in the MFE environment. **view_mfe_data** attempts to provide the most suitable display of the data, depending on the dimensions of the MFE arrays.

By default, the DataViewer uses double precision for floating point numbers in MFE math variables. If the *mfe_namei* is single precision, the DataViwer uses *e*-format to display it. You can enter single precision values using *e*-format. The MFE math engine will convert the type of the number as needed.

The DataViewer displays MFE variables or math expressions. You must first move Macsyma Math Engine data into an MFE variable before you can view it. Similarly, you must import data from a file into an MFE variable to view it.

If you try to create an DataViewer section, you may be prompted for the name of an MFE variable to view. You can create an empty DataViewer section by `Edit | Insert Section` and selecting `DataViewer`.

If no MFE variables exist, you must create one first. You can create an MFE variable in at least three ways:

- You can create an MFE variable from a Macsyma Math Engine variable by using the **mfe_put** command. See also **mfe_put**, page 55.
- You can create a new MFE variable by importing data from a file. Use `Data | Import` to select the file, and choose the name of the MFE variable. See also page 57.
- You can create a new MFE variable by assigning the data from an existing graphics section. Use `Graphics | Assign Data to Variable`.

plot_mfe_data(*mfe_name0* {,*mfe_name1, ...*}) *Function*

The MFE object with name *mfe_name0* appears in a graph in the current notebook. *mfe_namei* can refer to either a 1D or 2D array in the MFE environment. **plot_mfe_data** plots all the MFE arrays in one plot, regardless of the dimensions of the MFE arrays.

mfe_update_now() *Function*

Causes MFE to update its math objects before the Macsyma computation continues, after the call to **mfe_update_now** in a Macsyma program.

mfe_kill(*mfe_name*) *Function*

Removes the MFE object named *mfe_name* from the MFE environment.

2.1.1 Import and Export – external data files

You can export an MFE variable to a file by selecting the Menu item **Edit** — **Export**. You can choose CSV format, or fixed field.

Similarly, you can import an external data field in CSV or fixed field format by selecting the Menu item **Edit** — **Import**. You can choose CSV format, or fixed field.

2.2 The mfe_data Package

These commands are preliminary and are subject to change without notice. Refer to the on-line documentation.

See `usage(mfe_data);`.

Do `demo(mfe_data);` for several examples.

2.2.1 Getting Data From the MFE Math Engine

get_data_line(*objname, r_or_c, numb*) *Function*

Returns a Macsyma list whose values are taken from column (or row) number *numb* from the Macsyma matrix, Macsyma array, or MFE array named *objname*. *r_or_c* must be either 'row or 'col.

get_data_line_as_matrix(*objname, r_or_c, numb*) *Function*

Returns a Macsyma matrix whose values are taken from column (or row) number *numb* from the Macsyma matrix, Macsyma array or MFE array named *objname*. *r_or_c* must be either 'row or 'col. This returns a column or row vector.

get_data_as_matrix(*objname*) *Function*

Returns a Macsyma matrix whose values are taken from the Macsyma matrix or Macsyma array or MFE array named *objname*.

get_data_as_array(*objname,arrname*) *Function*

Fills the Macsyma array named *arrname* with the contents of the Macsyma matrix or Macsyma array or MFE array named *objname*. Returns the name *arrname*.

zero_based_arrays *default:* **false** *Option Variable*

For commands in the **mfe_data** package, *zero_based_arrays* determines whether (Macsyma and MFE) arrays are referenced with zero-based indexing or 1-based indexing.

zero_based_arrays	Index for first row/col of array
false	1
true	0

Matrices are always 1-based.

2.2.2 Putting Data into the MFE Math Engine from Macsyma

new_data_object(*mfe_name* {,*numtype, dim1, dim2*}) *Function*

Creates a new MFE object of type args =[type,num1,num2] and displays it in a DataViewer. If no number type or size is given, the default is a double float 10 x 10 array.

put_data_cell(*data, objname, rownum* {,*colnum*}) *Function*

Changes the value of one cell in a 1D or 2D Macsyma matrix or Macsyma array or MFE array.

put_data_1d(*dataobj, mfe_name, r_or_c, index_num*) *Function*

Puts 1D data into a 1D or 2D MFE array with name mfe_name. *mfe_name* must be the name of an existing MFE array. The value of *r_or_c* must be either 'row or 'col, and its value determines the acceptable forms of the 1D data object *dataobj*.

Value of *r_or_c*	Acceptable forms of *dataobj*
row	Macsyma list or row vector
col	Macsyma list or matrix

put_data_1d returns the value 'done.

2.2.3 Viewing Data

Chapter 1 describes how to produce plots from Macsyma variables. This section describes how to produce two- and three-dimensional plots from MFE variables. Please refer to Chapter 1 for more detailed information about plot options.

2.2.3.1 Making 2 and 3 Dimensional Plots of MFE Data

graph2d_data(*objname, r_or_c, col1* {, *col2*}) *Function*

Graphs column (or row) *col1* of data object *objname* on the horizontal axis and columns (or rows) *col2*,... of *objname* on the vertical axis. *objname* must be a Macsyma matrix, a Macsyma array or an MFE array. *r_or_c*, which must be either 'row or 'col, determines whether rows or columns are extracted from *objname*.

If *objname* is a 1-dimensional matrix, Macsyma array or MFE array, then the values are plotted as a function of the integer array index.

See **plot_mfe_data**, Section 2.1, page 56.

graph3d_data(*mfe_name, r_or_c, col1, col2, col3* {, *plotcode*}) *Function*

Makes a scatter plot of the columns (or rows) *col1*, *col2*, *col3* of the MFE array *mfe_name*. *r_or_c* must be either 'row or 'col, and determines whether rows or columns are extracted from *mfe_name*. The optional argument *plotcode* is a list of integers which controls the line type and plot point symbol, as for other Macsyma plotting commands. If *plotcode* is not specified, **graph3d_data** produces a scatter plot with no lines connecting the plot points.

plot3d_data(*objname* {, *whereplot, mfe_name*}) *Function*

Plots the values in the Macsyma matrix, Macsyma 2D array or 2D MFE array *objname* against the integer indices on the two horizontal axes. If the optional argument whereplot has the value 'macs, then the plot is generated in Macsyma. If whereplot has the value 'mfe, then mfe_name must be specified, the plot is generated in MFE, and the values in *objname* are stored in the MFE array *mfe_name*.

contourplot_data(*objname*) *Function*

Draws a contour plot of the matrix, Macsyma array, or MFE array *objname* in a Macsyma graphics section.

2.2.3.2 Smoothing and Graphing MFE Data

graph_smoothed_data(*objname, r_or_c, col1, col2* {,*type*}) *Function*

Fits the data in column (or row) *col2* of the matrix, Macsyma array, or MFE array *objname* to the abscissas in column (row) *col1* with a cubic spline (default), rational function or polynomial. *r_or_c*, which must have one of the values 'row and 'col, determines whether rows or columns of *objname* are graphed. If the value of the optional argument *type* is specified, it must have one of the values 'poly, 'rat, 'spline. The value of *type* determines the type of smoothing performed.

Index