Menggambar Grafik 2D dengan EMT

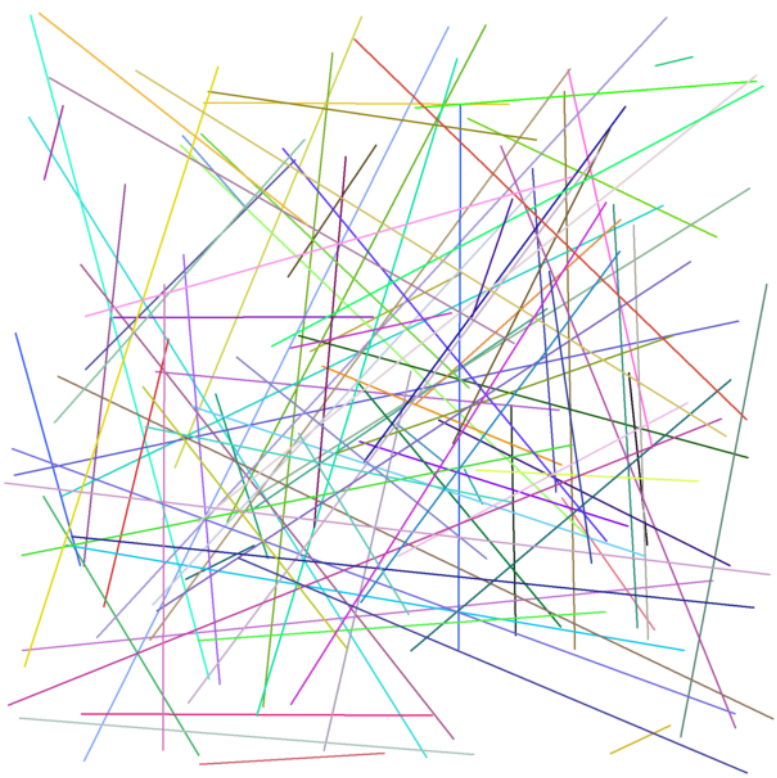
Notebook ini menjelaskan tentang cara menggambar berbagaikurva dan grafik 2D dengan software EMT. EMT menyediakan fungsi plot2d() untuk menggambar berbagai kurva dan grafik dua dimensi (2D).

Basic Plots

There are very basic functions of plots. There are screen coordinates, which always range from 0 to 1024 in each axis, no matter if the screen is square or not. Ant there are plot coordinates, which can be set with setplot(). The mapping between the coordinates depends on the current plot window. E.g., the default shrinkwindow() leaves space for axis labels and a plot title.

In the example, we just draw a few random lines in various colors. For details on these functions, study the core functions of EMT.

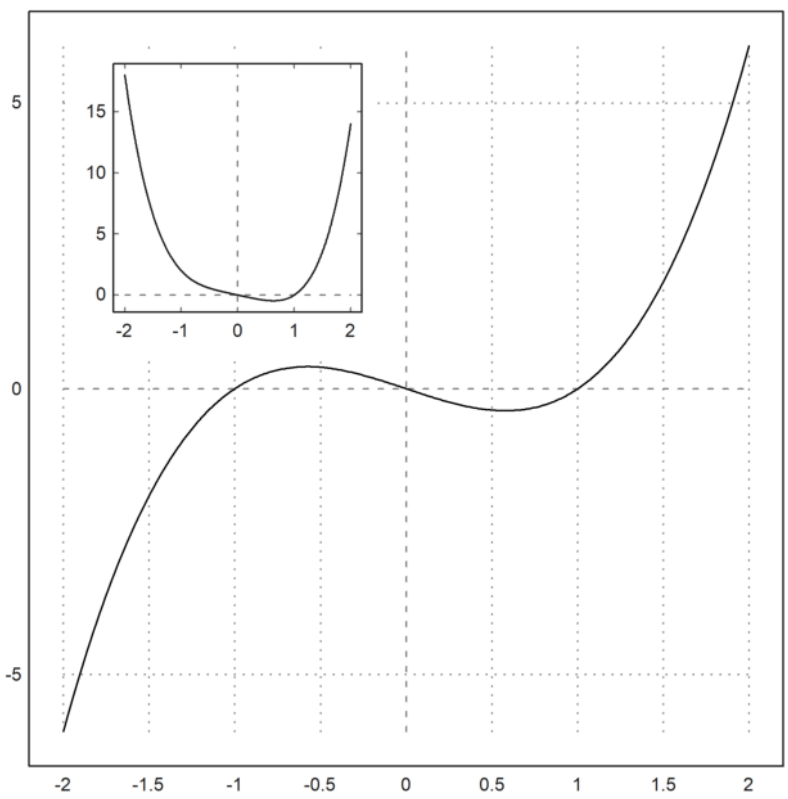
```
>clg; // clear screen
>window(0,0,1024,1024); // use all of the window
>setplot(0,1,0,1); // set plot coordinates
>hold on; // start overwrite mode
>n=100; X=random(n,2); Y=random(n,2);  // get random points
>colors=rgb(random(n),random(n),random(n)); // get random colors
>loop 1 to n; color(colors[#]); plot(X[#],Y[#]); end; // plot
>hold off; // end overwrite mode
>insimg; // insert to notebook
```



```
>reset;
```

It is necessary to hold the graphics, since the plot() command would clear the plot window.
To clear everything we did, we use reset().
Untuk menampilkan gambar hasil plot di layar notebook, perintah plot2d() dapat diakhiri dengan titik dua (:).
Cara lain adalah perintah plot2d() diakhiri dengan titik koma (;), kemudian menggunakan perintah insimg() untuk menampilkan gambar hasil plot.
For another example, we draw a plot as an inset in another plot. This is done by defining a smaller plot window. Note that this window does not provide room for the axis labels outside the plot window. We have to add some margin for this as needed. Note that we save and restore the full window, and hold the current plot while we plot the inset.

```
>plot2d("x^3-x");
>xw=200; yw=100; ww=300; hw=300;
>ow=window();
>window(xw,yw,xw+ww,yw+hw);
>hold on;
>barclear(xw-50,yw-10,ww+60,ww+60);
>plot2d("x^4-x",grid=6):
```



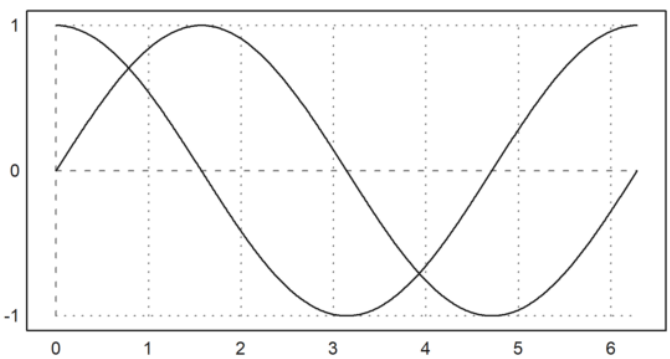

```
>hold off;
>window(ow);
```

A plot with multiple figures is achieved in the same way. There is the utility figure() function for this.

Plot Aspect

The default plot uses a square plot window. You can change this with the function aspect(). Do not forget to reset the aspect later. You can also change this default in the menu with "Set Aspect" to a specific aspect ratio or to the current size of the graphics window.
But you can change it also for one plot. For this, the current size of the plot area is changed, and the window is set so that the labels have enough space.

```
>aspect(2); // rasio panjang dan lebar 2:1
>plot2d(["sin(x)","cos(x)"],0,2pi):
```




```
>aspect();
>reset;
```

The reset() function restores plot defaults including the aspect ratio.

2D Plots in Euler

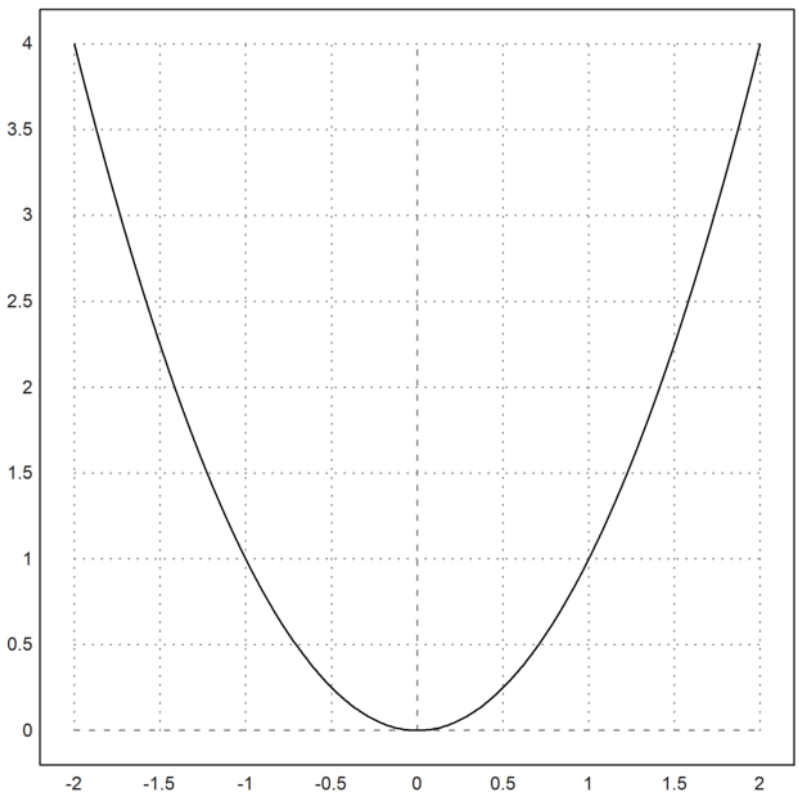
EMT Math Toolbox has plots in 2D, both for data and functions. EMT uses the function plot2d. This function can plot functions and data.
It is possible to plot in Maxima using Gnuplot or in Python using Math Plot Lib.
Euler can plot 2D plots of
- ekspresi
- functions, variables, or parameterized curves,
- vectors of x-y-values,
- clouds of points in the plane,
- implicit curves with levels or level regions.
- Complex function

Plot styles include various styles for lines and points, bar plots and shaded plots.

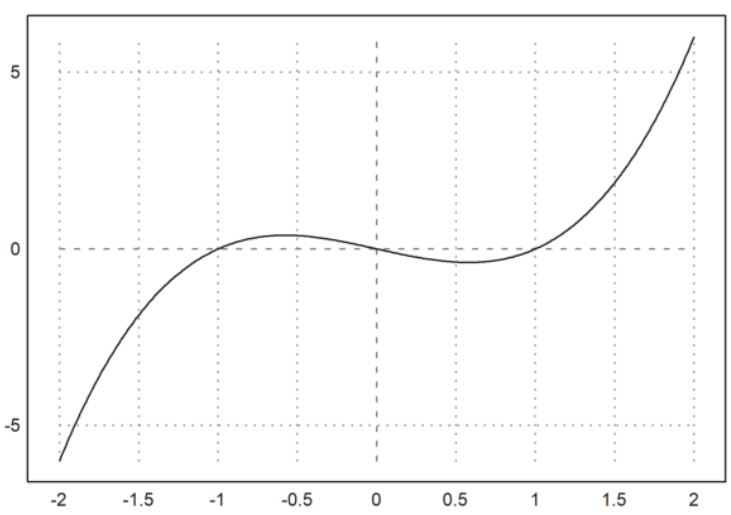
Plot Ekspresi atau Variabel

A single expression in "x" (e.g. "4*x^2") or the name of a function (e.g. "f") produces a graph of the function. Here is the most basic example, which uses the default range and sets a proper y-range to fit the plot of the function.
Note: If you end a command line with a colon ":", the plot will be inserted into the text window. Otherwise, press TAB to see the plot if the plot window is covered.

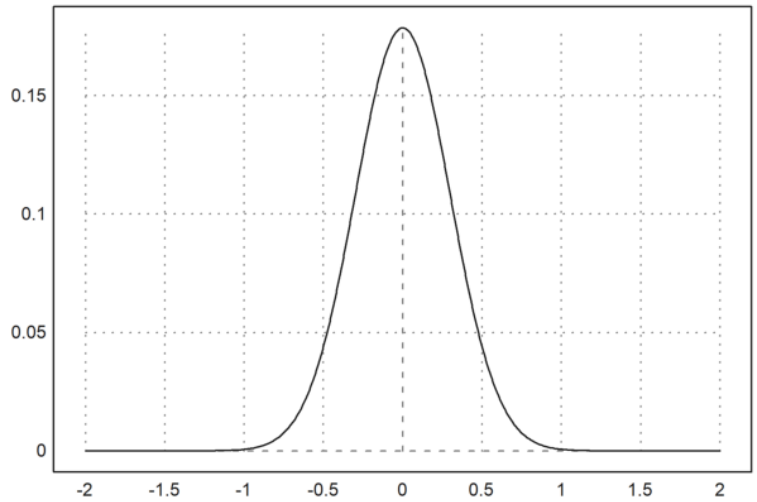
```
>plot2d("x^2"):
```




```
>aspect(1.5); plot2d("x^3-x"):
```

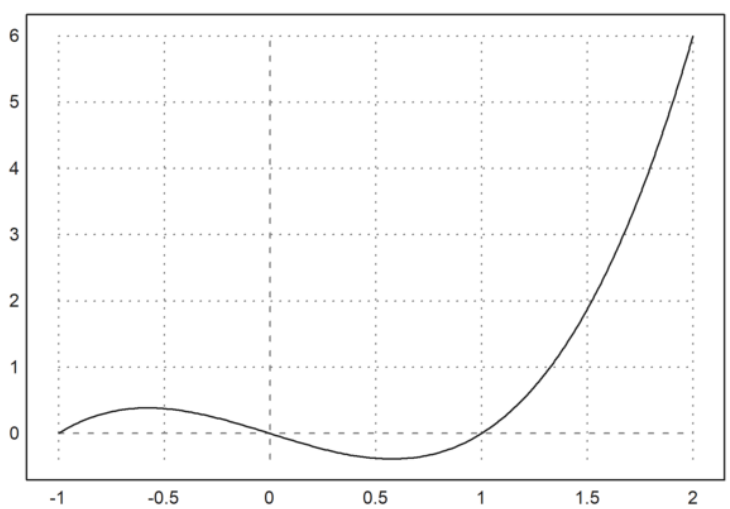


```
>a:=5.6; plot2d("exp(-a*x^2)/a"); insimg(30); // menampilkan gambar hasil plot setinggi 25
```

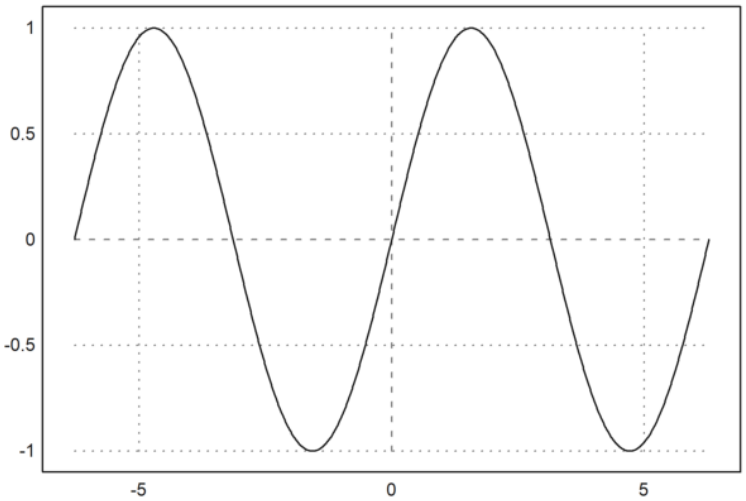


Dari beberapa contoh sebelumnya Anda dapat melihat bahwa aslinya gambar plot menggunakan sumbu X dengan rentang nilai dari -2 sampai dengan 2. Untuk mengubah rentang nilai X dan Y, Anda dapat menambahkan nilai-nilai batas X (dan Y) di belakang ekspresi yang digambar.
The plot range is set with the following assigned parameters
- a,b: x-range (default -2,2)
- c,d: y-range (default: scale with values)
- r: alternatively a radius around the plot center
- cx,cy: the coordinates of the plot center (default 0,0)

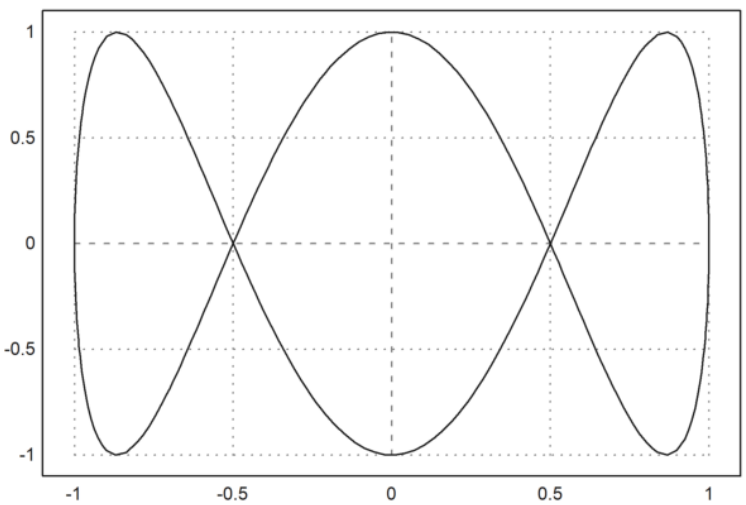
```
>plot2d("x^3-x",-1,2):
```




```
>plot2d("sin(x)",-2*pi,2*pi): // plot sin(x) pada interval [-2pi, 2pi]
```

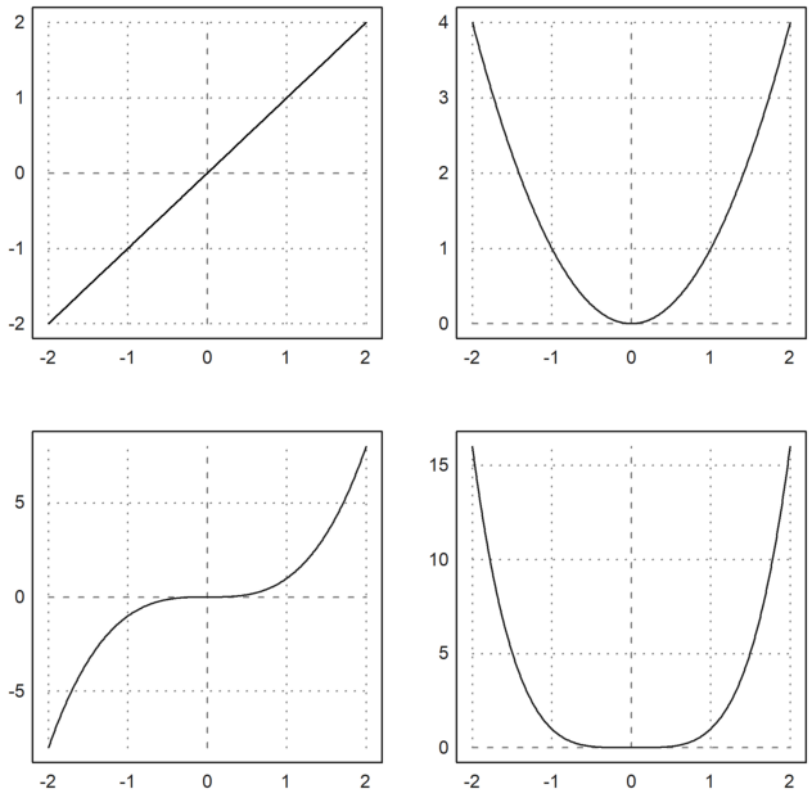



```
>plot2d("cos(x)","sin(3*x)",xmin=0,xmax=2pi):
```



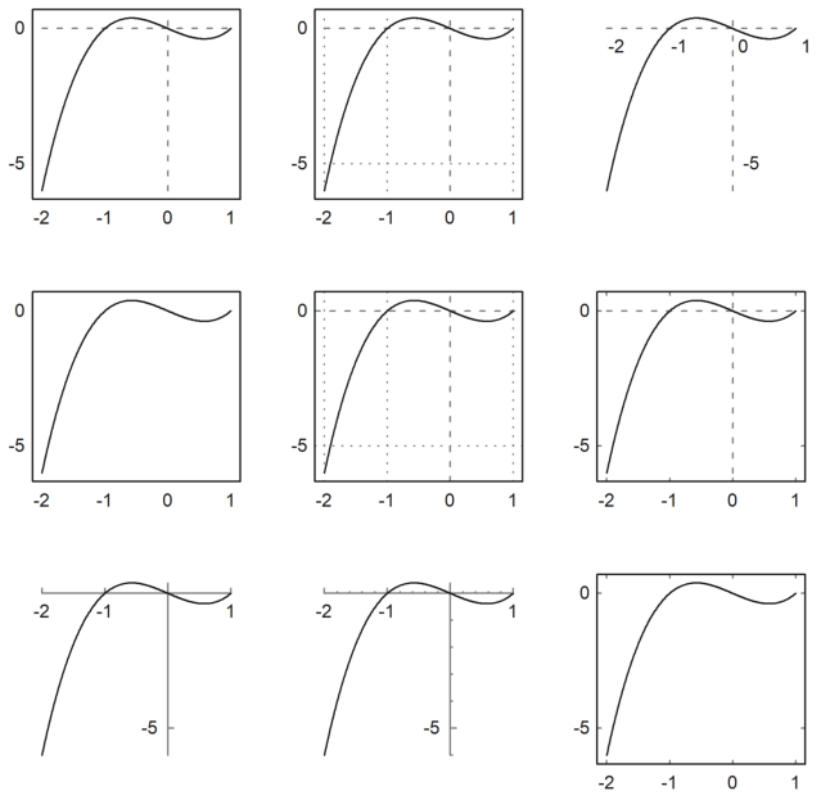
An alternative to the colon is the command insimg(lines), which inserts the plot occupying a specified number of text lines.
In the options, the plots can be set to appear
- in a separate resizable window,
- in the notebook window.
More styles can be achieved with specific plot commands.
In any case, press the tabulator key to see the plot, if it is hidden.
To split the window into several plots, use the figure() command. In the example, we plot x^1 to x^4 into 4 parts of the window. figure(0) resets the default window.

```
>reset;
>figure(2,2); ...
>for n=1 to 4; figure(n); plot2d("x^"+n); end; ...
>figure(0):
```



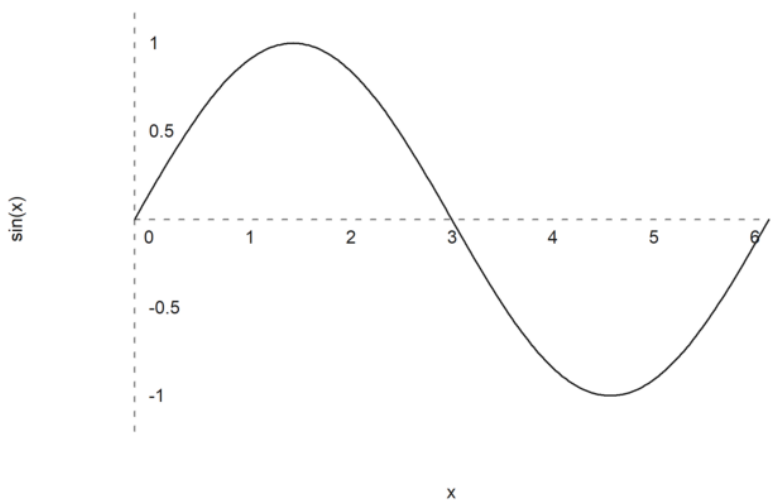
In plot2d(), there are alternative styles available with grid=x. For an overview, we show the various grid styles in one figure (see below for the figure() command). The style grid=0 is not included. It shows no grid and no frame.

```
>figure(3,3); ...
>for k=1:9; figure(k); plot2d("x^3-x",-2,1,grid=k); end; ...
>figure(0):
```

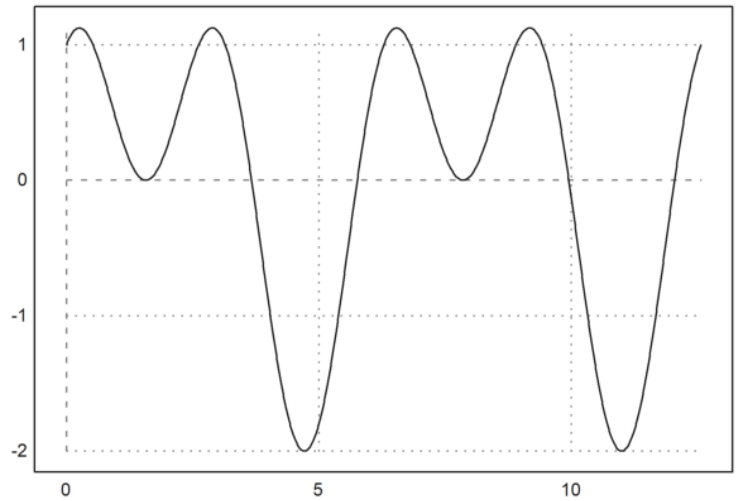


If the the arguments to plot2d() are an expression followed by four numbers, these numbers are the x- and y-ranges for the plot.
Alternatively, a, b, c, d can be specified as assigned parameters as a=... etc.
In the following example, we change the grid style, add labels, and use vertical labels for the y-axis.

```
>aspect(1.5); plot2d("sin(x)",0,2pi,-1.2,1.2,grid=3,xl="x",yl="sin(x)"):
```

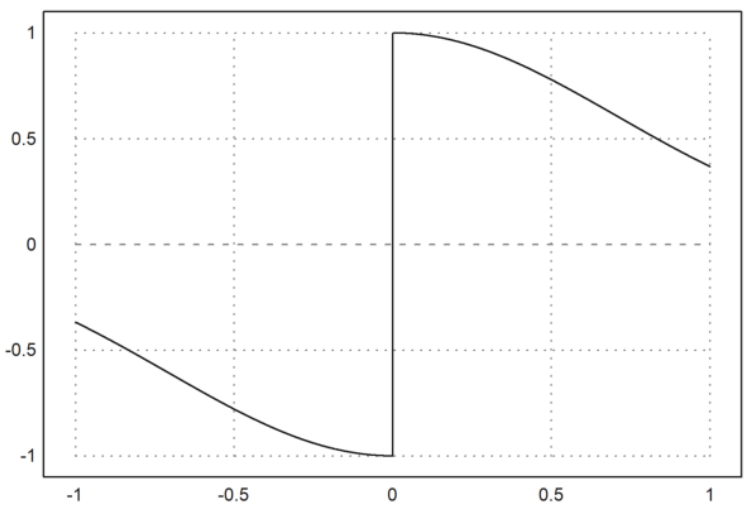



```
>plot2d("sin(x)+cos(2*x)",0,4pi):
```

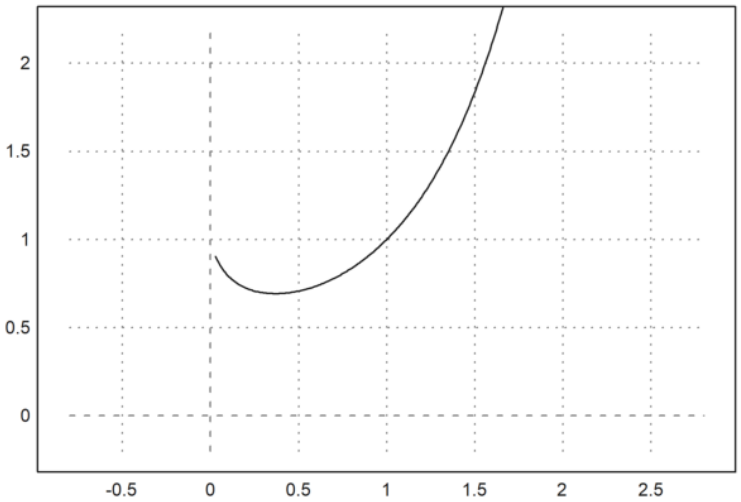


The images generated by inserting the plot into the text window are stored in the same directory as the notebook, by default in a subdirectory named "images". They are also used by the HTML export.
You can simply mark any image and copy it to the clipboard with Ctrl-C. Of course, you can also export the current graphics with the functions in the File menu.
The function or the expression in plot2d is evaluated adaptively. For more speed, switch off adaptive plots with <adaptive and specify the number of subintervals with n=... This should be necessary in rare cases only.

```
>plot2d("sign(x)*exp(-x^2)",-1,1,<adaptive,n=10000):
```

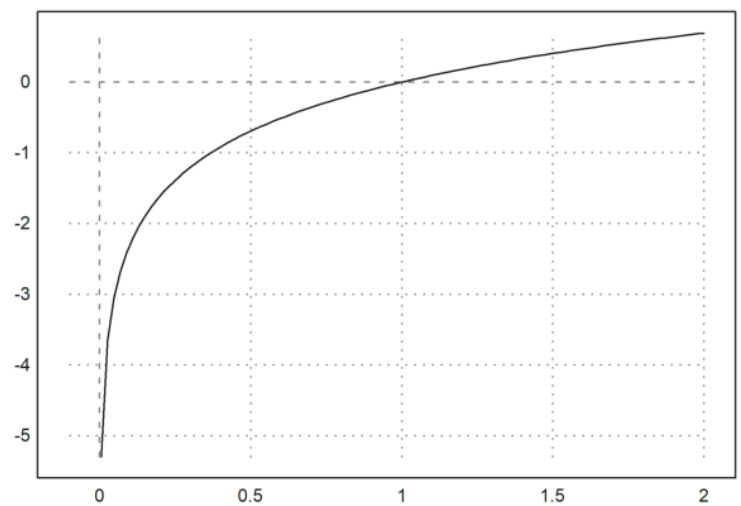



```
>plot2d("x^x",r=1.2,cx=1,cy=1):
```



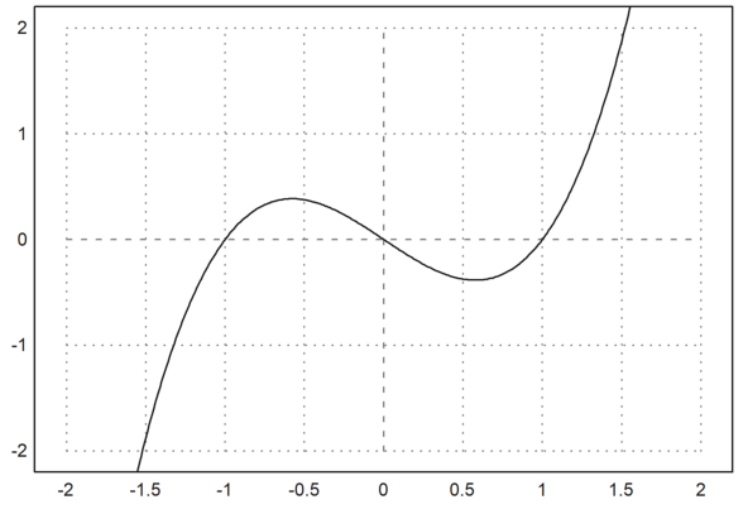
Note that x^x is not defined for x<=0. The plot2d function catches this error, and starts plotting as soon as the function is defined. This works for all functions which return NAN out of their range of definition.

```
>plot2d("log(x)",-0.1,2):
```

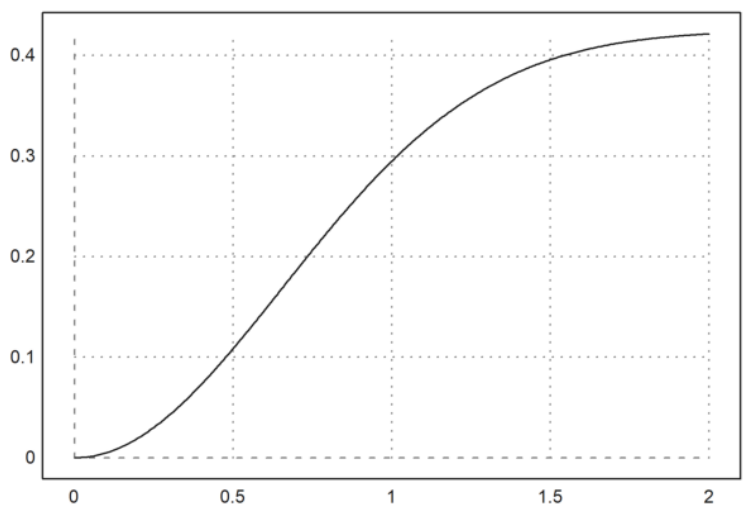


The parameter square=true (or >square) selects the y-range automatically so that the result is a square plot window. Note that by default, Euler uses a square space inside the plot window.

```
>plot2d("x^3-x",>square):
```

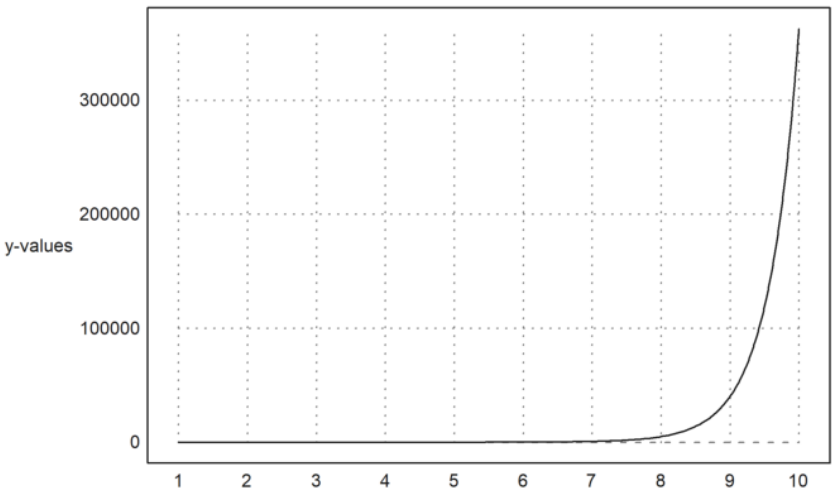



```
>plot2d(''integrate("sin(x)*exp(-x^2)",0,x)'',0,2): // plot integral
```



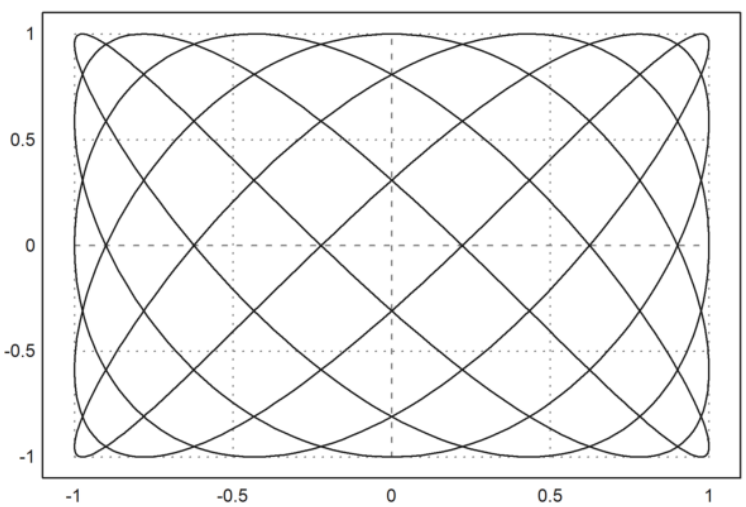
If you need more space for the y-labels, call shrinkwindow() with the smaller parameter, or set a positive value for "smaller" in plot2d().

```
>plot2d("gamma(x)",1,10,yl="y-values",smaller=6,<vertical):
```

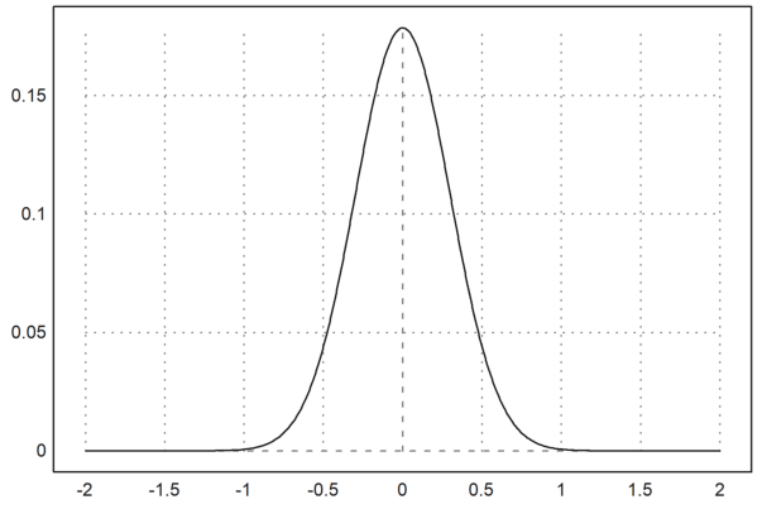


Symbolic expressions can also be used, since they are stored as simple string expressions.

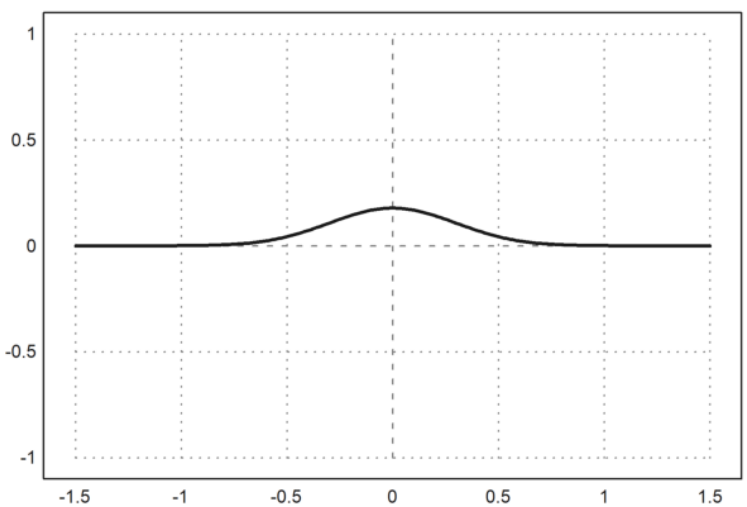
```
>x=linspace(0,2pi,1000); plot2d(sin(5x),cos(7x)):
```



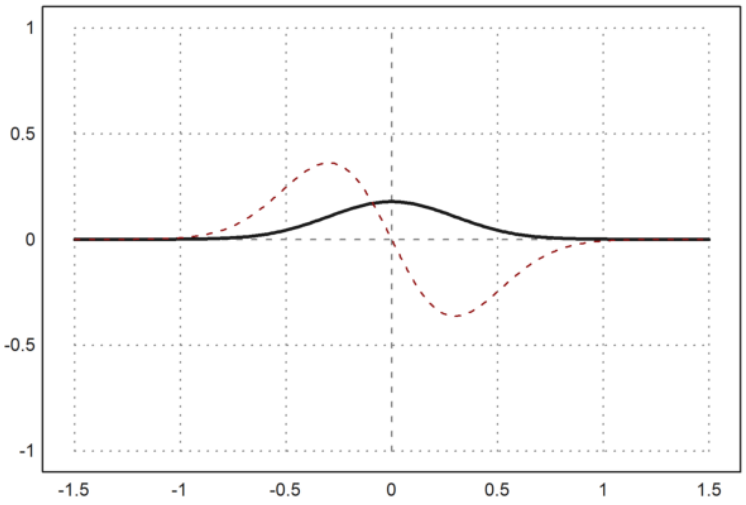

```
>a:=5.6; expr &= exp(-a*x^2)/a; // define expression
>plot2d(expr,-2,2): // plot from -2 to 2
```



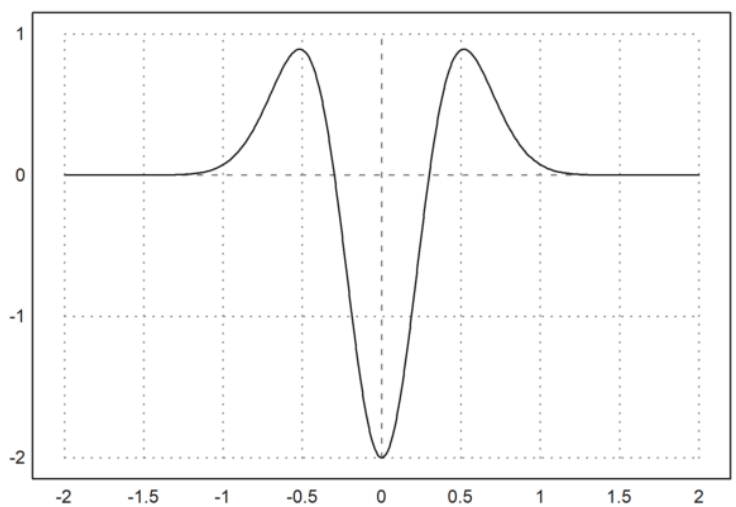

```
>plot2d(expr,r=1,thickness=2): // plot in a square around (0,0)
```



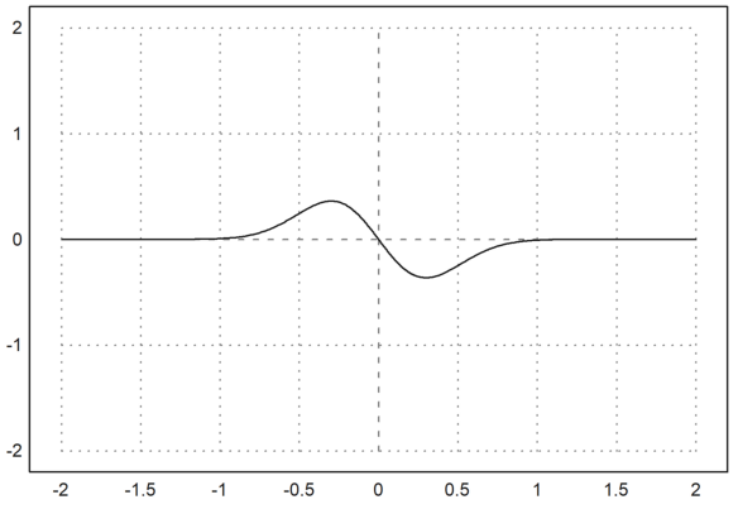

```
>plot2d(&diff(expr,x),>add,style="--",color=red): // add another plot
```



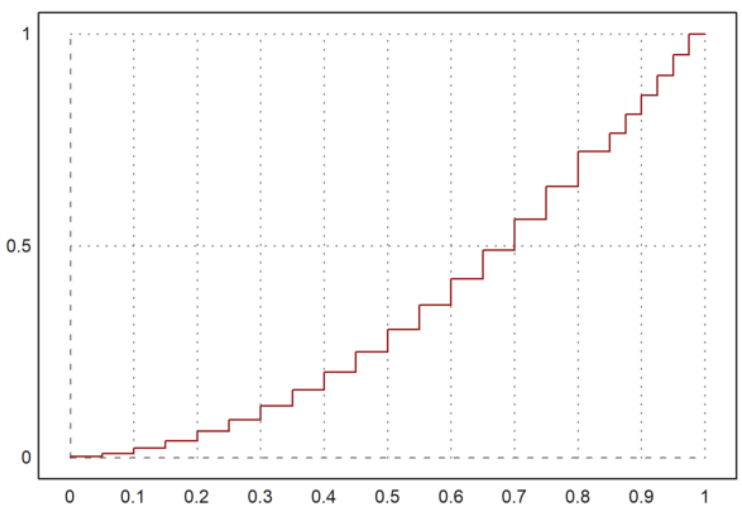

```
>plot2d(&diff(expr,x,2),a=-2,b=2,c=-2,d=1): // plot in rectangle
```



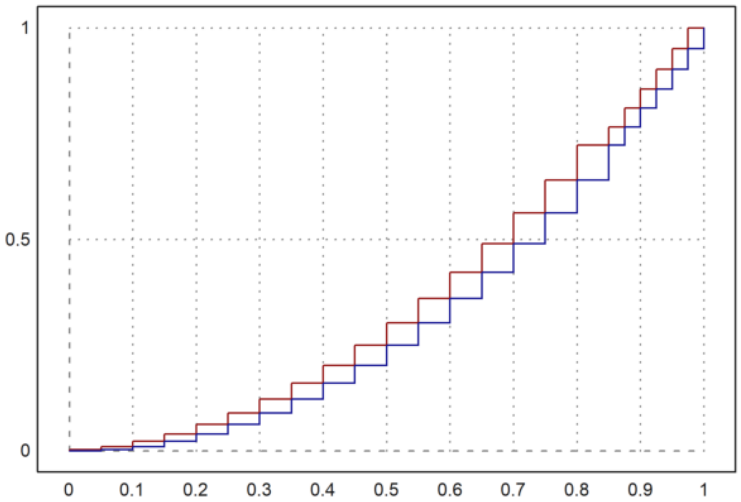

```
>plot2d(&diff(expr,x),a=-2,b=2,>square): // keep plot square
```




```
>plot2d("x^2",0,1,steps=1,color=red,n=10):
```



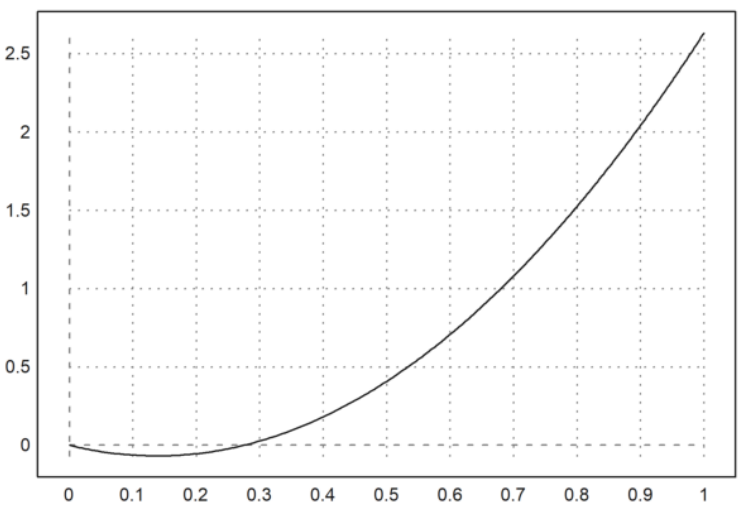

```
>plot2d("x^2",>add,steps=2,color=blue,n=10):
```



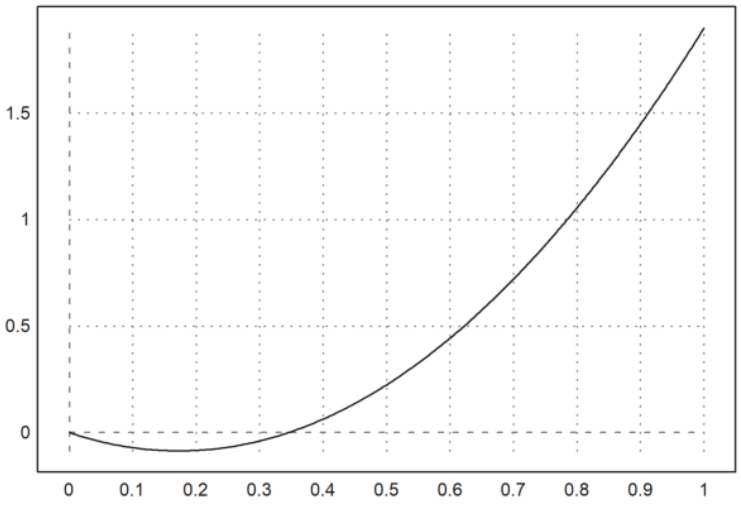
Functions in one Parameter

The most important plotting function for planar plots is plot2d(). The function is implemented in the Euler language in the file "plot.e", which is loaded at the start of the program.
Here are some examples using a function. As usual in EMT, functions that work for other functions or expressions, you can pass additional parameters (besides x) which are not global variables to the function with semicolon parameters or with a call collection.

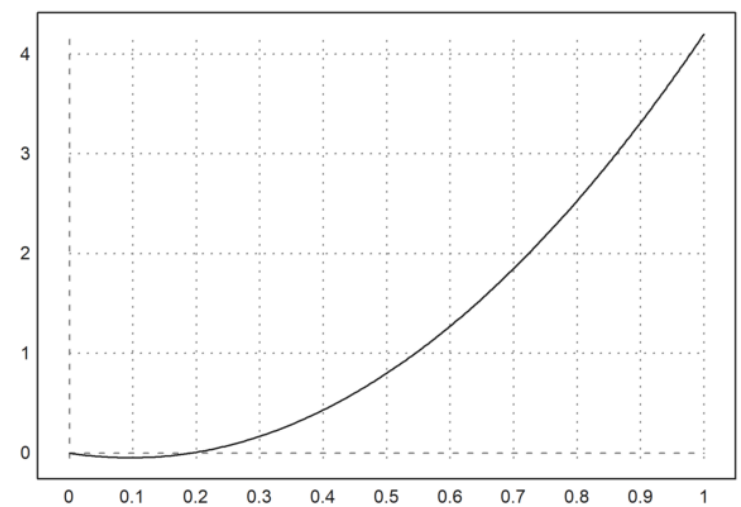
```
>function f(x,a) := x^2/a+a*x^2-x; // define a function
>a=0.3; plot2d("f",0,1;a): // plot with a=0.3
```



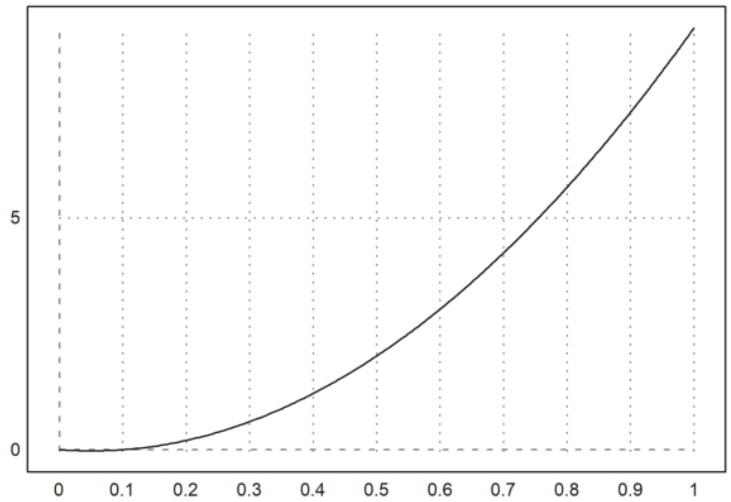

```
>plot2d("f",0,1;0.4): // plot with a=0.4
```



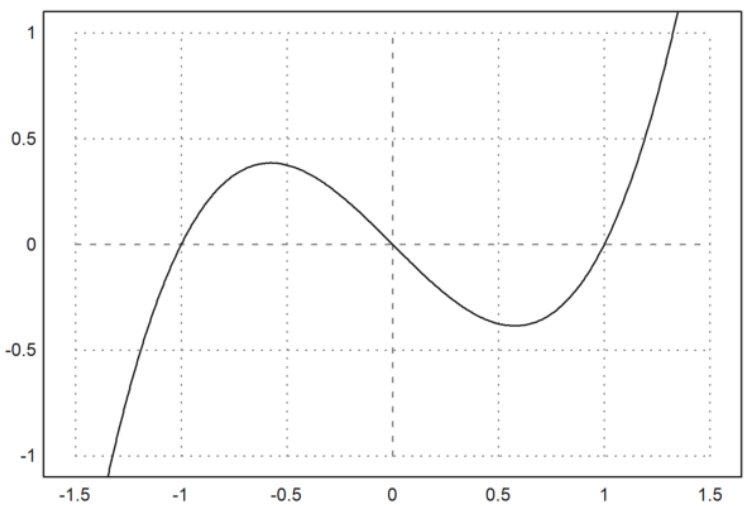

```
>plot2d({{"f",0.2}},0,1): // plot with a=0.2
```




```
>plot2d({{"f(x,b)",b=0.1}},0,1): // plot with 0.1
```




```
>function f(x) := x^3-x; ...
>plot2d("f",r=1):
```

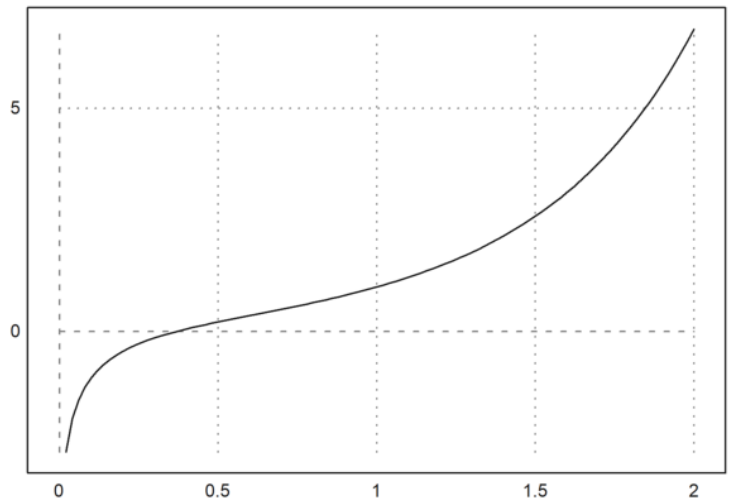


Here is a summary of the accepted functions
- expressions or symbolic expressions in x
- functions or symbolic functions by name as "f"
- symbolic functions just by the name f
The function plot2d() also accepts symbolic functions. For symbolic functions, the name alone works.

```
>function f(x) &= diff(x^x,x)
```

$$x^{x} \, (\log(x) + 1)$$

```
>plot2d(f,0,2):
```

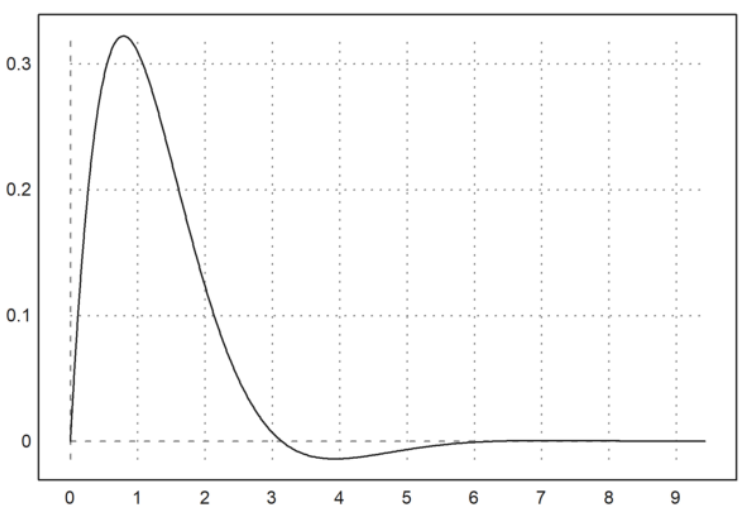


Of course, for expressions or symbolic expressions the name of the variable is enough to plot them.

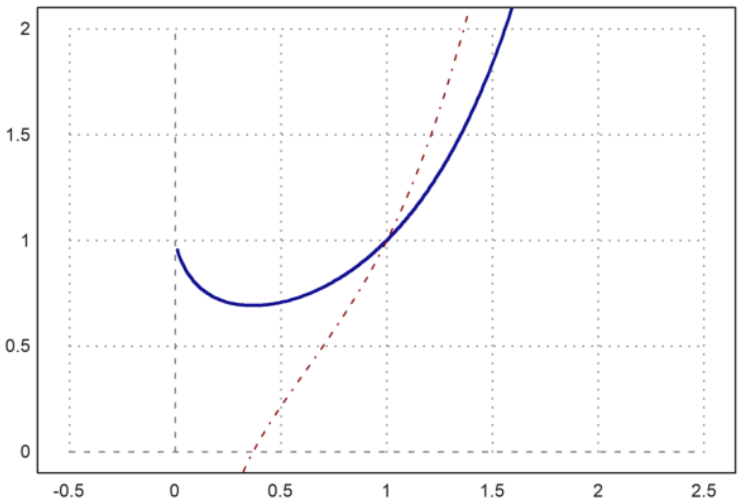
```
>expr &= sin(x)*exp(-x)
```

$$E^{-x}\sin(x)$$

```
>plot2d(expr,0,3pi):
```



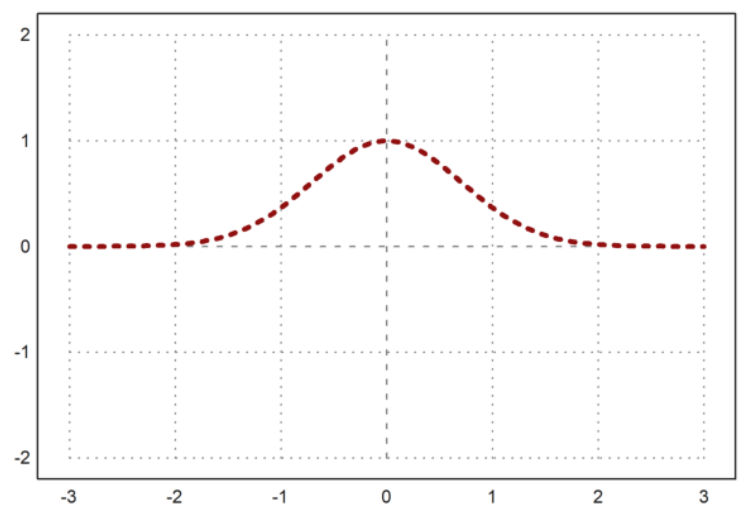
```
>function f(x) &= x^x;
>plot2d(f,r=1,cx=1,cy=1,color=blue,thickness=2);
>plot2d(&diff(f(x),x),>add,color=red,style="-.-"):
```



For the line style there are various options.
- style="...". Select from "-", "–", "-.", ".", ".-.", "-.-".
- color: See below for colors.
- thickness: Default is 1.

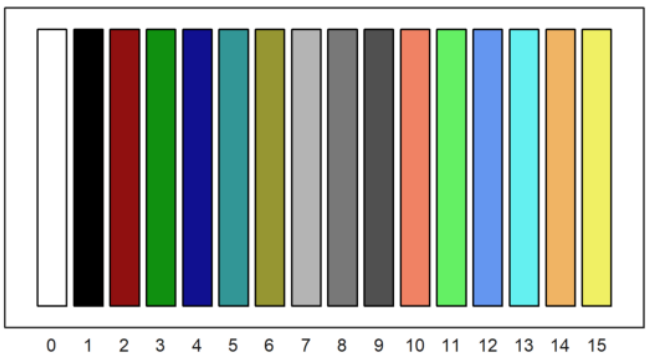
Colors can be selected as one of the default colors, or as an RGB color.
- 0..15: the default color indices.
- color constants: white, black, red, green, blue, cyan, olive, lightgray, gray, darkgray, orange, lightgreen, turquoise, lightblue, lightorange, yellow
- rgb(red,green,blue): parameters are reals in [0,1].

```
>plot2d("exp(-x^2)",r=2,color=red,thickness=3,style="--"):
```



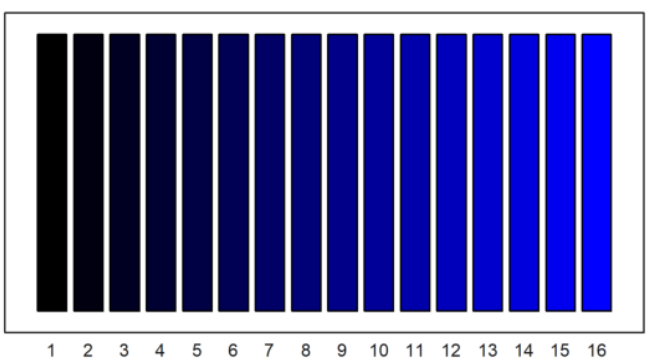
Here is a view of the predefined colors of EMT.

```
>aspect(2); columnsplot(ones(1,16),lab=0:15,grid=0,color=0:15):
```



But you can use any color.

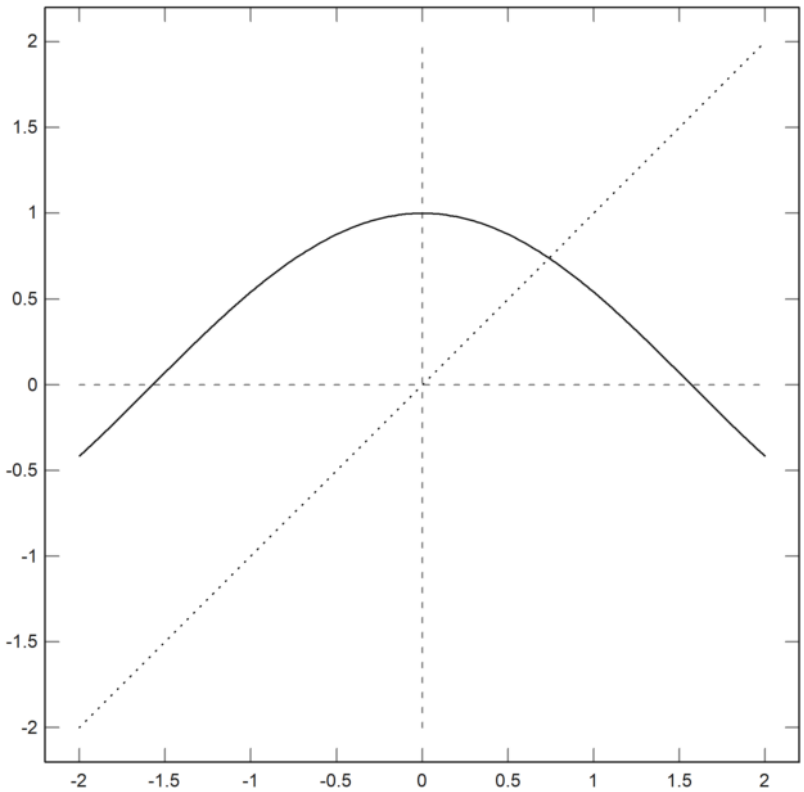
```
>columnsplot(ones(1,16),grid=0,color=rgb(0,0,linspace(0,1,15))):
```



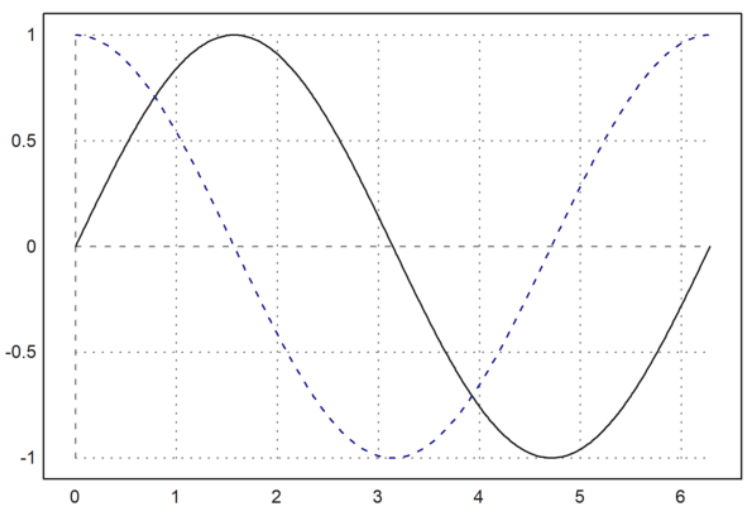
Menggambar Beberapa Kurva pada bidang koordinat yang sama

The plot more than one function (multiple functions) into one window can be done with different ways. One of the methos is using >add for several calls to plot2d in all, but the first call. We have used this feature already in the examples above.

```
>aspect(); plot2d("cos(x)",r=2,grid=6); plot2d("x",style=".",>add):
```

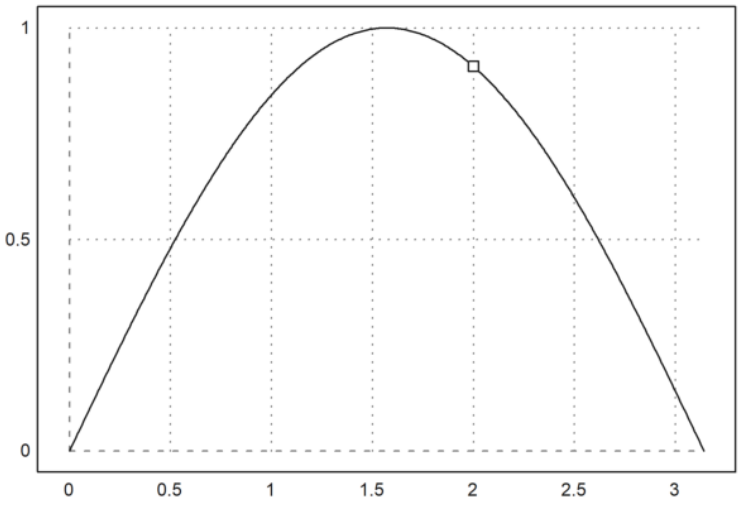



```
>aspect(1.5); plot2d("sin(x)",0,2pi); plot2d("cos(x)",color=blue,style="--",>add):
```



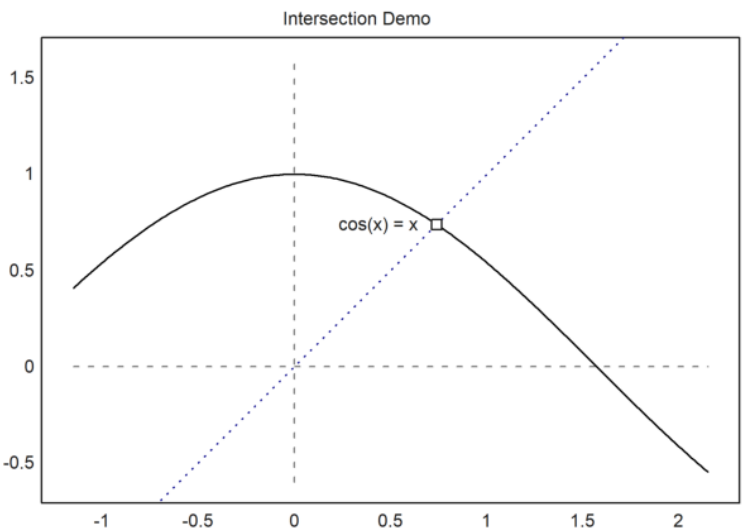
Salah satu kegunaan >add adalah untuk menambahkan titik pada kurva.

```
>plot2d("sin(x)",0,pi); plot2d(2,sin(2),>points,>add):
```



We add the intersection point with a label (at position "cl" for center left), and insert the result into the notebook. We also add a title to the plot.

```
>plot2d(["cos(x)","x"],r=1.1,cx=0.5,cy=0.5, ...
>  color=[black,blue],style=["-","."], ...
>  grid=1);
>x0=solve("cos(x)-x",1);  ...
>  plot2d(x0,x0,>points,>add,title="Intersection Demo");  ...
>  label("cos(x) = x",x0,x0,pos="cl",offset=20):
```

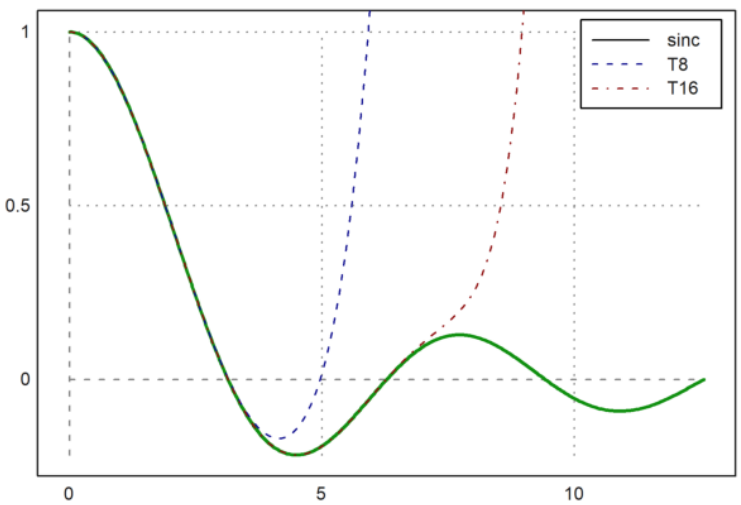


In the following demo, we plot the sinc(x)=sin(x)/x function and its 8-th and 16-th Taylor expansion. We compute this expansion using Maxima via symbolic expressions.
This plot is done in the following multi-line command with three calls to plot2d(). The second and the third have the >add flag set, which makes the plots use the previous ranges.
We add a label box explaining the functions.

```
>$taylor(sin(x)/x,x,0,4)
```

$$\frac{x^4}{120}-\frac{x^2}{6}+1$$

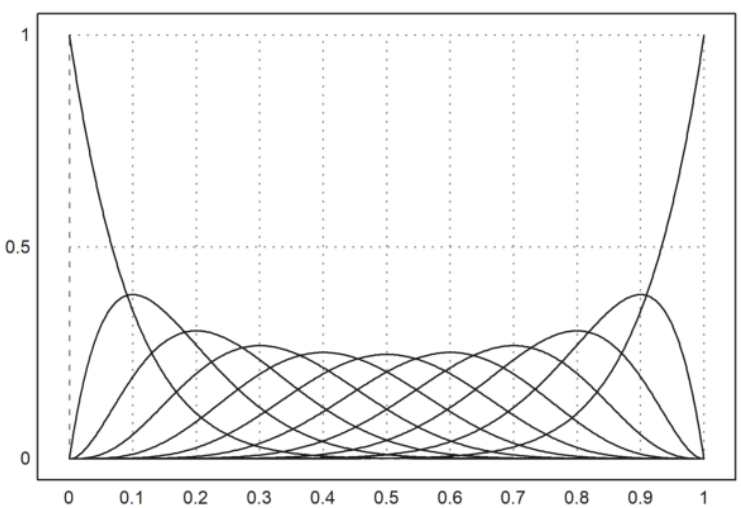
```
>plot2d("sinc(x)",0,4pi,color=green,thickness=2); ...
>  plot2d(&taylor(sin(x)/x,x,0,8),>add,color=blue,style="--"); ...
>  plot2d(&taylor(sin(x)/x,x,0,16),>add,color=red,style="-.-"); ...
>  labelbox(["sinc","T8","T16"],styles=["-","--","-.-"], ...
>    colors=[black,blue,red]):
```



In the following example, we generate the Bernstein-Polynomials.

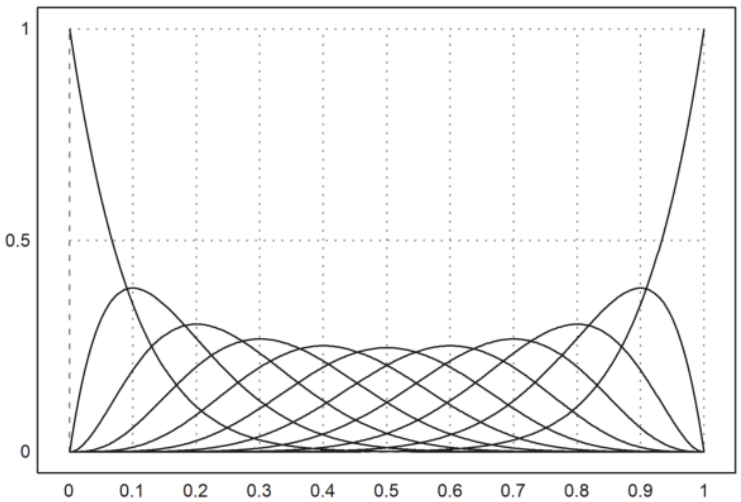
$$B_i(x)=\binom{n}{i}x^i(1-x)^{n-i}$$

```
>plot2d("(1-x)^10",0,1); // plot first function
>for i=1 to 10; plot2d("bin(10,i)*x^i*(1-x)^(10-i)",>add); end;
>insimg;
```



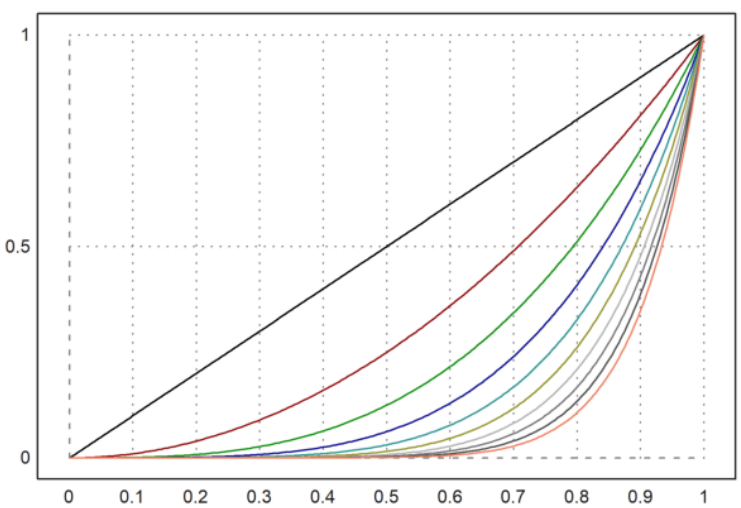
The second method is using a pair of a matrix of x-values and a matrix of y-values of the same size.
We generate a matrix of values with one Bernstein-Polynomial in each row. For this, we simply use a column vector of i. Have a look into the introduction about the matrix language to learn more details.

```
>x=linspace(0,1,500);
>n=10; k=(0:n)'; // n is row vector, k is column vector
>y=bin(n,k)*x^k*(1-x)^(n-k); // y is a matrix then
>plot2d(x,y):
```



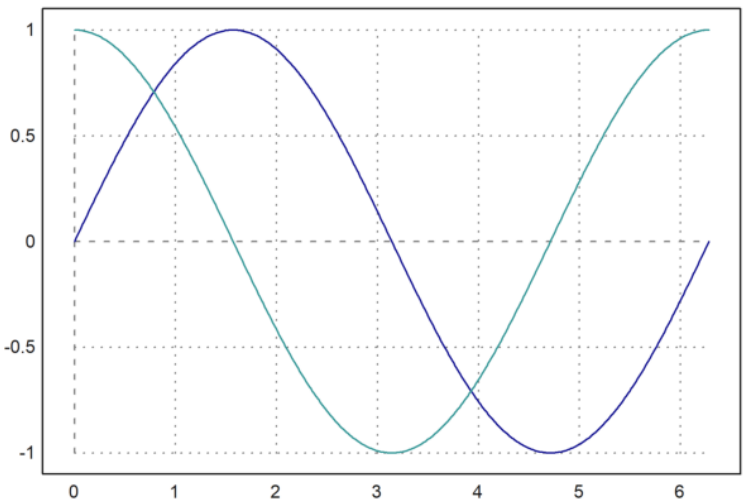
Note that the color parameter can be a vector. Then each color is used for each row of the matrix.

```
>x=linspace(0,1,200); y=x^(1:10)'; plot2d(x,y,color=1:10):
```

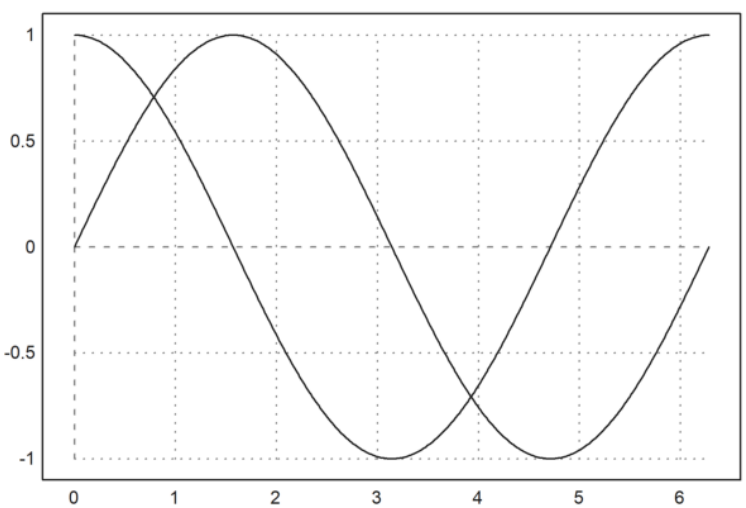


Another method is using a vector of expressions (strings). You can then use a color array, an array of styles, and an array of thicknesses of the same length.

```
>plot2d(["sin(x)","cos(x)"],0,2pi,color=4:5):
```




```
>plot2d(["sin(x)","cos(x)"],0,2pi): // plot vector of expressions
```



We can get such a vector from Maxima using makelist() and mxm2str().

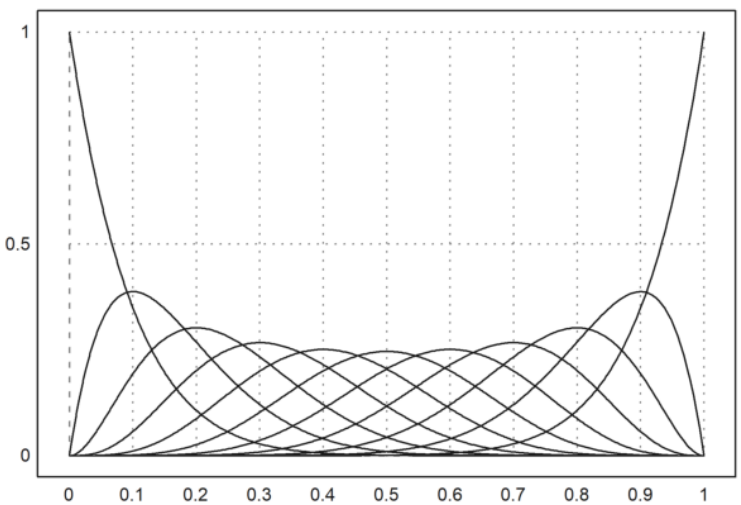
```
>v &= makelist(binomial(10,i)*x^i*(1-x)^(10-i),i,0,10) // make list
```

$$\left[(1-x)^{10},\ 10\,(1-x)^{9}\,x,\ 45\,(1-x)^{8}\,x^{2},\ 120\,(1-x)^{7}\,x^{3},\ 210\,(1-x)^{6}\,x^{4},\ 252\,(1-x)^{5}\,x^{5},\ 210\,(1-x)^{4}\,x^{6},\ 120\,(1-x)^{3}\,x^{7},\ 45\,(1-x)^{2}\,x^{8},\ 10\,(1-x)\,x^{9},\ x^{10}\right]$$

```
>mxm2str(v) // get a vector of strings from the symbolic vector
```

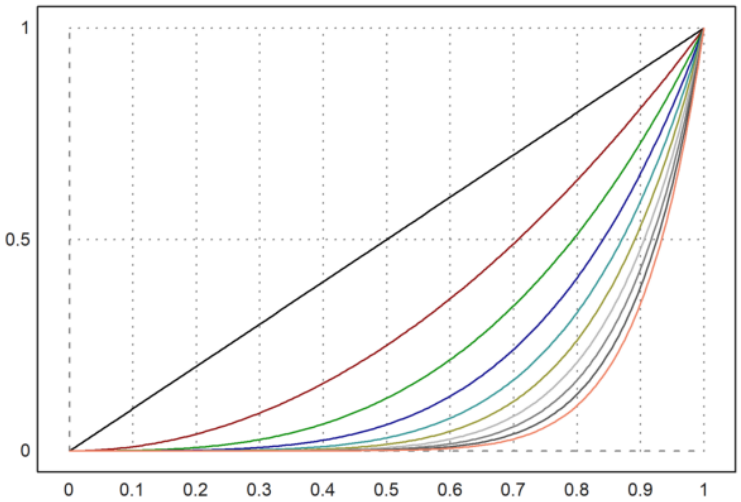
```
(1-x)^10
10*(1-x)^9*x
45*(1-x)^8*x^2
120*(1-x)^7*x^3
210*(1-x)^6*x^4
252*(1-x)^5*x^5
210*(1-x)^4*x^6
120*(1-x)^3*x^7
45*(1-x)^2*x^8
10*(1-x)*x^9
x^10
```

```
>plot2d(mxm2str(v),0,1): // plot functions
```



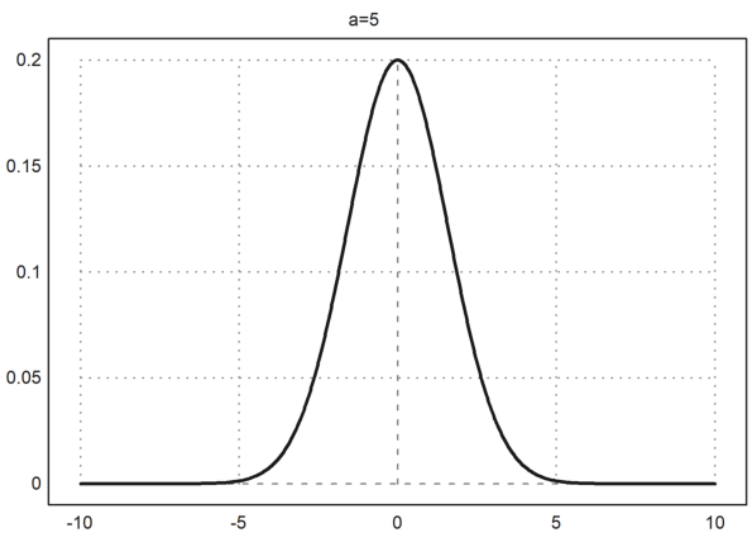
Another alternative is to use the matrix language of Euler.
If an expression produces a matrix of functions, with one function in each row, all these functions will be plotted into one plot.
For this, use a parameter vector in form of a column vector. If a color array is added it will be used for each row of the plot.

```
>n=(1:10)'; plot2d("x^n",0,1,color=1:10):
```



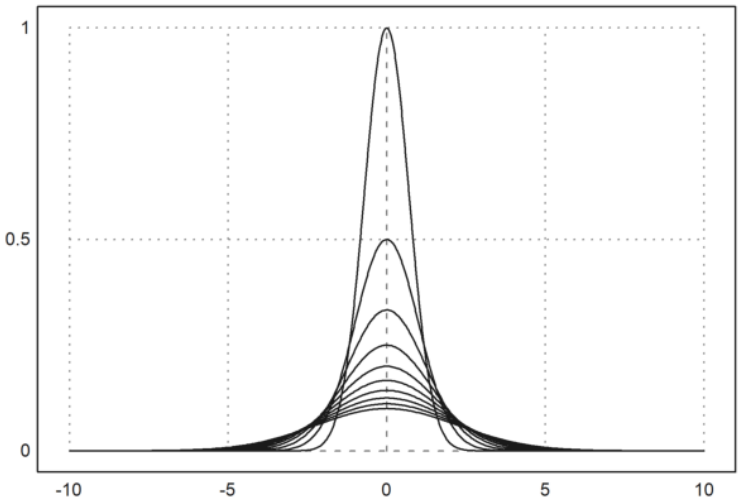
Expressions and one-line functions can see global variables.
If you cannot use a global variable, you need to use a function with an extra parameter, and pass this parameter as a semicolon parameter.
Take care, to put all assigned parameters to the end of the plot2d command. In the example we pass a=5 to the function f, which we plot from -10 to 10.

```
>function f(x,a) := 1/a*exp(-x^2/a); ...
>plot2d("f",-10,10;5,thickness=2,title="a=5"):
```



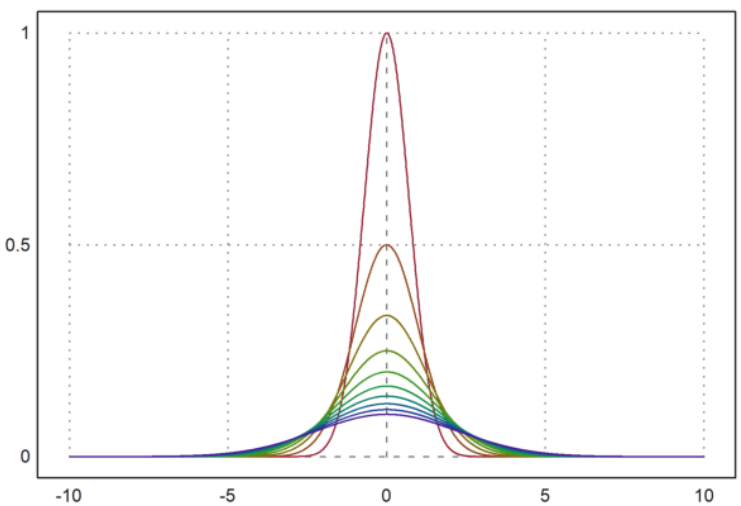
Alternatively, use a collection with the function name and all extra parameters. These special lists are called call collections, and that is the preferred way to pass arguments to a function which is itself passed as an argument to another function.
In the following example, we use a loop to plot several functions (see the tutorial about programming for loops).

```
>plot2d({{"f",1}},-10,10); ...
>for a=2:10; plot2d({{"f",a}},>add); end:
```



We could achieve the same result in the following way using the matrix language of EMT. Each row of the matrix f(x,a) is one function. Moreover, we can set colors for each row of the matrix. Double click on the function getspectral() for an explanation.

```
>x=-10:0.01:10; a=(1:10)'; plot2d(x,f(x,a),color=getspectral(a/10)):
```

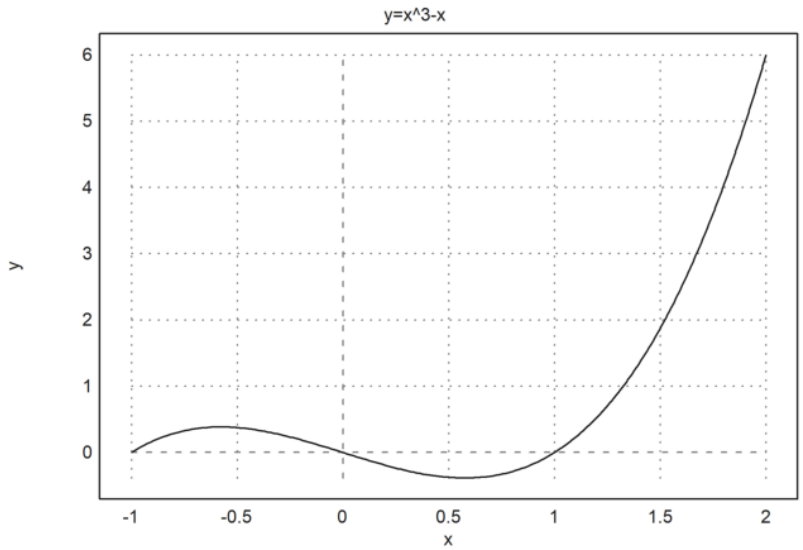


Text Labels

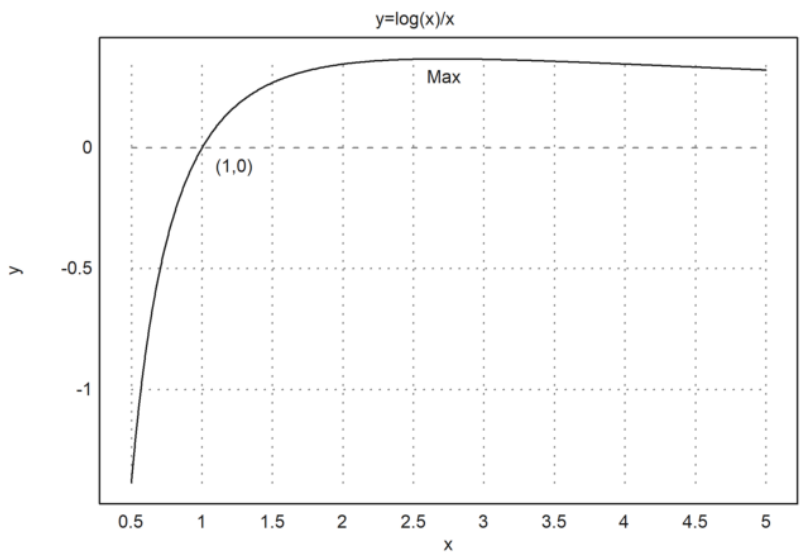
Simple decorations can be
- a title with title="..."
- x- and y-labels with xl="...", yl="..."
- another text label with label("...",x,y)

The label command will plot into the current plot at the plot oordinates (x,y). It can take a positional argument.

```
>plot2d("x^3-x",-1,2,title="y=x^3-x",yl="y",xl="x"):
```

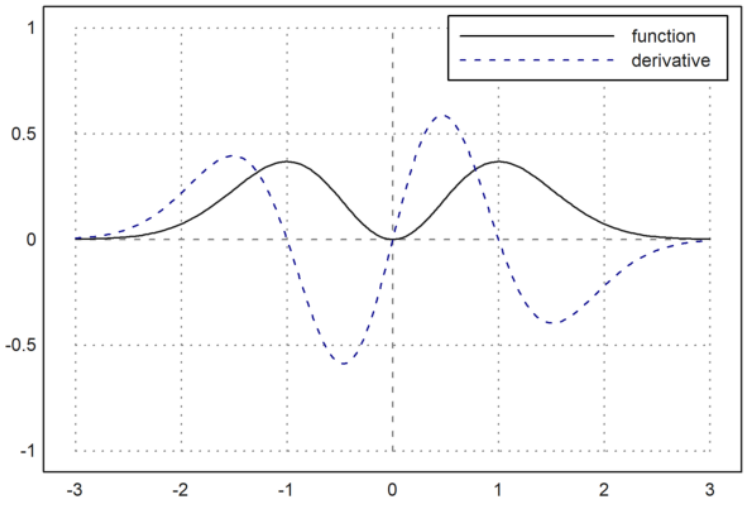



```
>expr := "log(x)/x"; ...
>  plot2d(expr,0.5,5,title="y="+expr,xl="x",yl="y"); ...
>  label("(1,0)",1,0); label("Max",E,expr(E),pos="lc"):
```



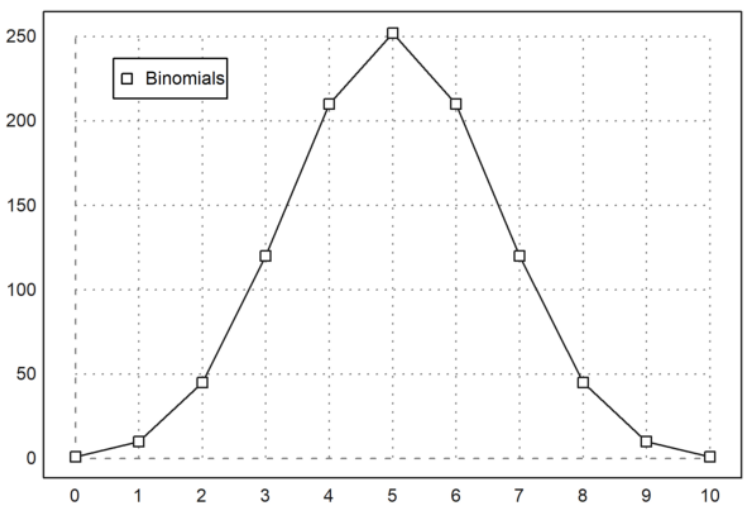
There is also the function labelbox(), which can show the functions and a text. It takes vectors of strings and colors, one item for each function.

```
>function f(x) &= x^2*exp(-x^2);  ...
>plot2d(&f(x),a=-3,b=3,c=-1,d=1);  ...
>plot2d(&diff(f(x),x),>add,color=blue,style="--"); ...
>labelbox(["function","derivative"],styles=["-","--"], ...
>   colors=[black,blue],w=0.4):
```



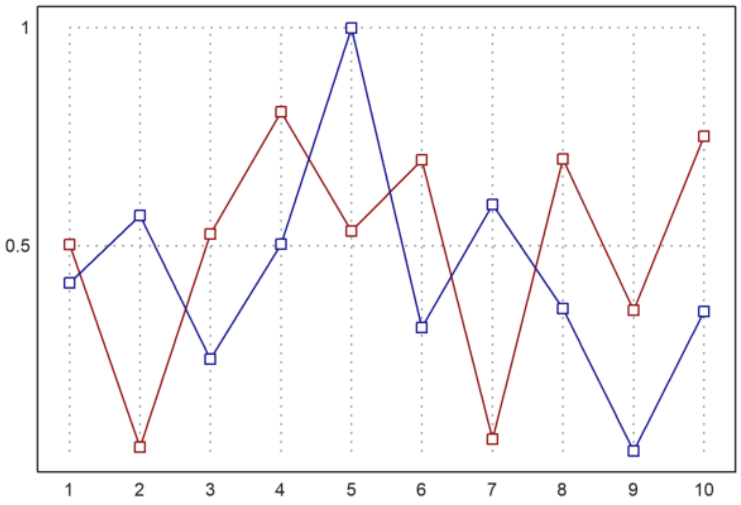
The box is anchored at the top right by default, but >left anchors it at the top left. You can move it to any place you like. The anchor position is the top right corner of the box, and the numbers are fractions of the size of the graphics window. The width is automatic.
For point plots, the label box works too. Add a parameter >points, or a vector of flags, one for each label.
In the following example, there is only one function. So we can use strings instead of vectors of strings. We set the text color to black for this example.

```
>n=10; plot2d(0:n,bin(n,0:n),>addpoints); ...
>labelbox("Binomials",styles="[]",>points,x=0.1,y=0.1, ...
>tcolor=black,>left):
```



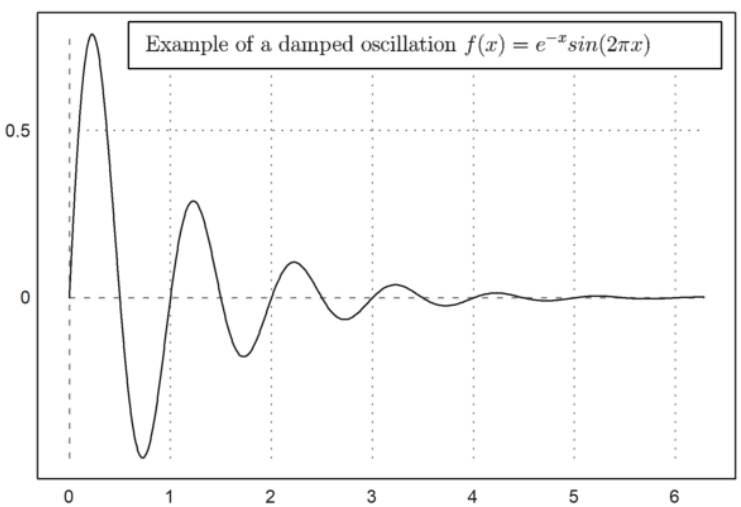
This style of plot is also available in statplot(). Like in plot2d() colors can be set for each row of the plot. There are more special plots for statistical purposes (see the tutorial about statistics).

```
>statplot(1:10,random(2,10),color=[red,blue]):
```



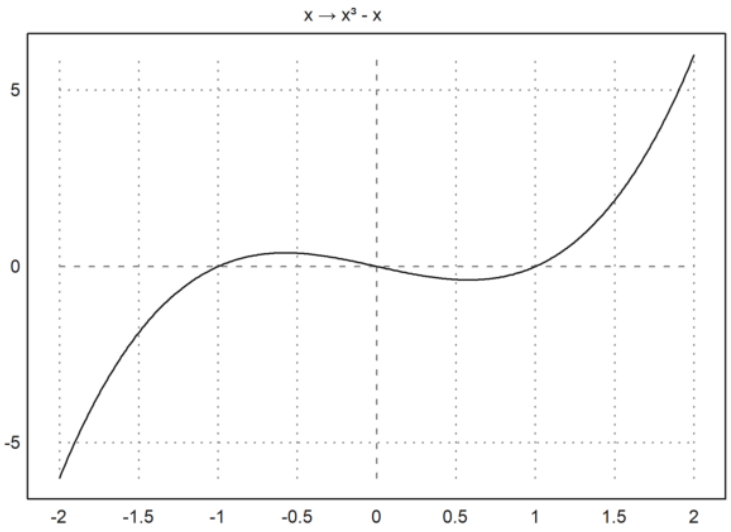
A similar feature is the function textbox().
The width is by default the maximal widths of the text lines. But it can be set by the user too.

```
>function f(x) &= exp(-x)*sin(2*pi*x); ...
>plot2d("f(x)",0,2pi); ...
>textbox(latex("\text{Example of a damped oscillation}\ f(x)=e^{-x}sin(2\pi x)"),w=0.85):
```



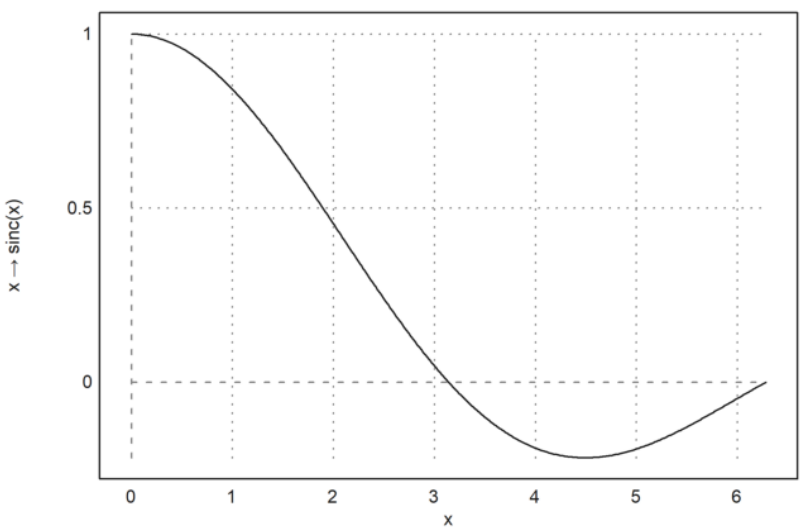
Text labels, titles, label boxes and other text can contain Unicode strings (see the syntax of EMT for more about Unicode strings).

```
>plot2d("x^3-x",title=u"x &rarr; x&sup3; - x"):
```



The labels on the x- and y-axis can be vertical, as well as the axis.

```
>plot2d("sinc(x)",0,2pi,xl="x",yl=u"x &rarr; sinc(x)",>vertical):
```

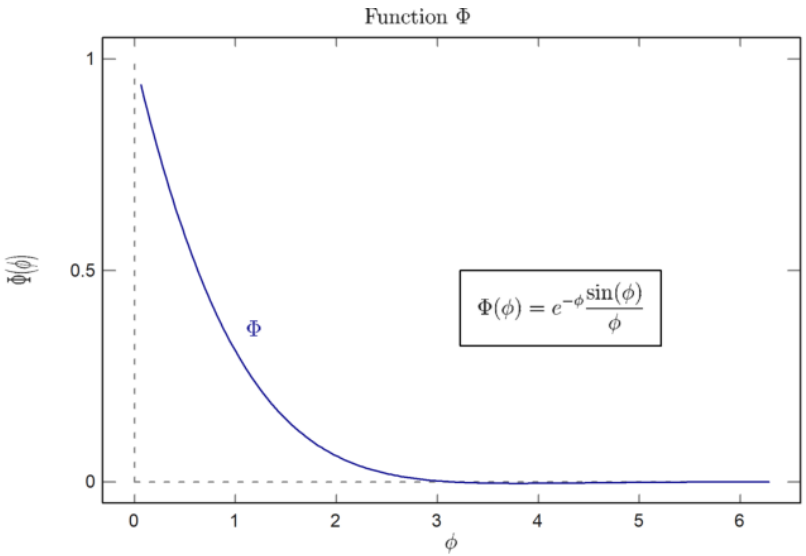


LaTeX

You can also plot LaTeX formulas if you have installed the LaTeX system. I recommend MiKTeX. The path to the binaries "latex" and "dvipng" should be in the system path, or you have to setup LaTeX in the options menu.
Note, that parsing LaTeX is slow. If you want to use LaTeX in animated plots, you should call latex() before the loop once and use the result (an image in a RGB matrix).
In the following plot, we use LaTeX for x- and y-labels, a label, a label box and the title of the plot.

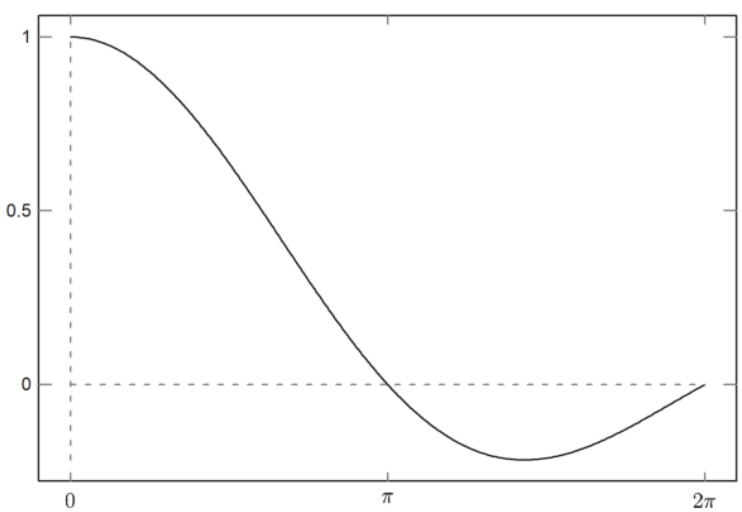
```
>plot2d("exp(-x)*sin(x)/x",a=0,b=2pi,c=0,d=1,grid=6,color=blue, ...
>  title=latex("\text{Function $\Phi$}"), ...
>  xl=latex("\phi"),yl=latex("\Phi(\phi)")); ...
```

```
>textbox( ...
>  latex("\Phi(\phi) = e^{-\phi} \frac{\sin(\phi)}{\phi}"),x=0.8,y=0.5); ...
>label(latex("\Phi",color=blue),1,0.4):
```



Often, we wish a non-conformal spacing and text labels on the x-axis. We can use xaxis() and yaxis() as we will show later.
The easiest way is to do a blank plot with a frame using grid=4, and then to add the grids with ygrid() and xgrid(). In the following example, we use three LaTeX strings for the labels on the x-axis with xtick().

```
>plot2d("sinc(x)",0,2pi,grid=4,<ticks); ...
>ygrid(-2:0.5:2,grid=6); ...
>xgrid([0:2]*pi,<ticks,grid=6);  ...
>xtick([0,pi,2pi],["0","\pi","2\pi"],>latex):
```



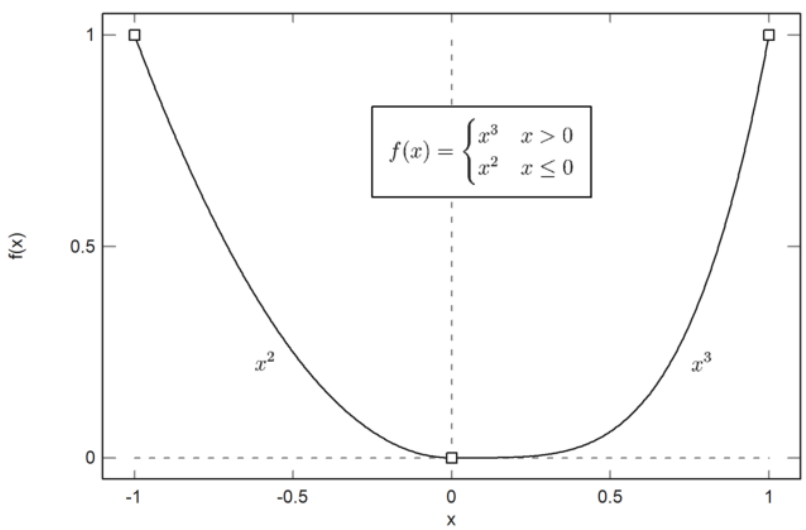
Of course, functions can also be used.

```
>function map f(x) ...
```

```
  if x>0 then return x^4
  else return x^2
  endif
  endfunction
```

The "map" parameter helps to use the function for vectors. For the plot, it would not be necessary. But to demonstrate that vectorization is useful, we add some key points to the plot at x=-1, x=0 and x=1.
In the following plot, we also enter some LaTeX code. We use it for two labels and a text box. Of course, you will only be able to use LaTeX if you have LaTeX installed properly.

```
>plot2d("f",-1,1,xl="x",yl="f(x)",grid=6);  ...
>plot2d([-1,0,1],f([-1,0,1]),>points,>add); ...
>label(latex("x^3"),0.72,f(0.72)); ...
>label(latex("x^2"),-0.52,f(-0.52),pos="ll"); ...
>textbox( ...
>  latex("f(x)=\begin{cases} x^3 & x>0 \\ x^2 & x \le 0\end{cases}"), ...
>  x=0.7,y=0.2):
```

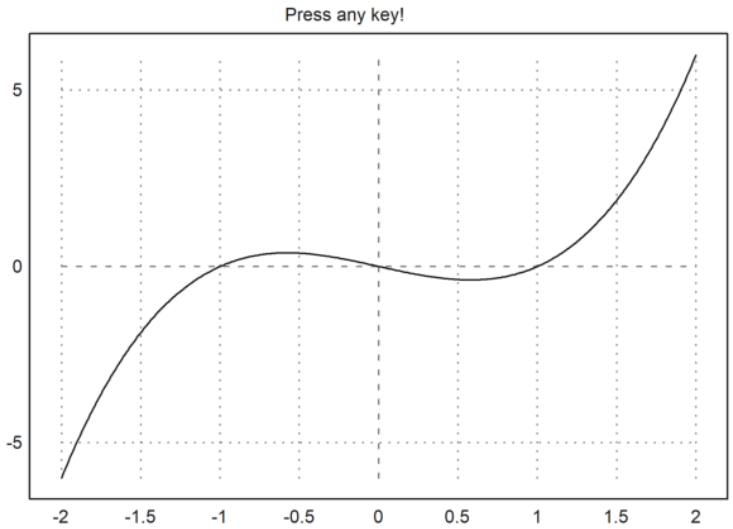


User Interaction

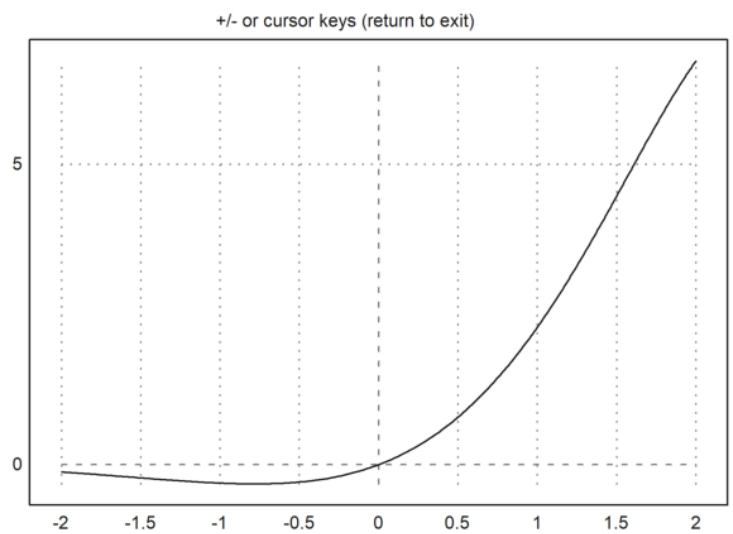
When plotting a function or an expression, the parameter >user allows the user to zoom and shift the plot with the cursor keys or the mouse. The user can
- zoom with + or -
- move the plot with the cursor keys
- select a plot window with the mouse
- reset the view with space
- exit with return

The space key will reset the plot to the original plot window.
When plotting a data, the >user flag will simply wait for key stroke.

```
>plot2d({{"x^3-a*x",a=1}},>user,title="Press any key!"):
```




```
>plot2d("exp(x)*sin(x)",user=true, ...
>  title="+/- or cursor keys (return to exit)"):
```



The following demonstrates an advanced way of user interaction (see the tutorial about programming for details).

The built-in function mousedrag() waits for mouse or keyboard events. It reports mouse down, mouse moved or mouse up, and key presses. The function dragpoints() makes use of this, and lets the user drag any point in a plot.

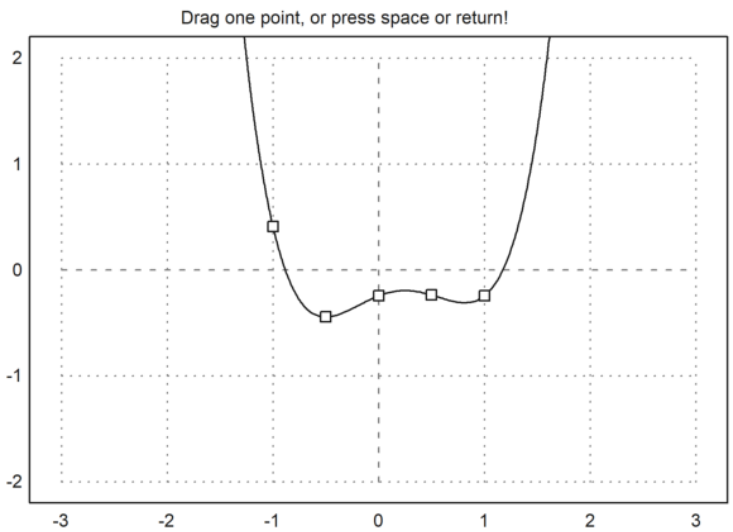
We need a plot function first. For an example, we interpolate in 5 points with a polynomial. The function should plot into a fixed plot area.

```
>function plotf(xp,yp,select) ...
```

```
    d=interp(xp,yp);
    plot2d("interpval(xp,d,x)";d,xp,r=2);
    plot2d(xp,yp,>points,>add);
    if select>0 then
      plot2d(xp[select],yp[select],color=red,>points,>add);
    endif;
    title("Drag one point, or press space or return!");
  endfunction
```

Note the semicolon parameters in plot2d (d and xp), which are passed to the evaluation of the interp() function. Without this, we must write a function plotinterp() first, accessing the values globally.
Now we generate some random values, and let the user drag the points.

```
>t=-1:0.5:1; dragpoints("plotf",t,random(size(t))-0.5):
```

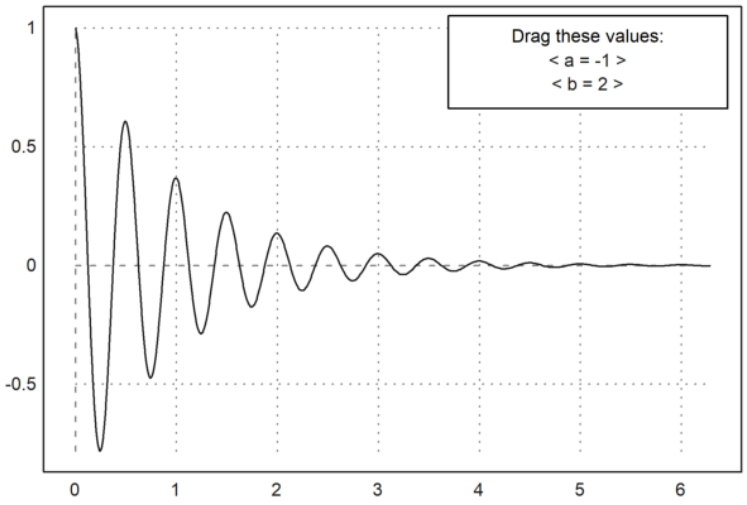


There is also a function, which plots another function depending on a vector of parameters, and lets the user adjust these parameters.
First we need the plot function.

```
>function plotf([a,b]) := plot2d("exp(a*x)*cos(2pi*b*x)",0,2pi;a,b);
```

Then we need names for the parameters, initial values and a nx2 matrix of ranges, optionally a heading line. There are interactive sliders, which can set values by the user. The function dragvalues() provides this.

```
>dragvalues("plotf",["a","b"],[-1,2],[[-2,2];[1,10]], ...
>  heading="Drag these values:",hcolor=black):
```



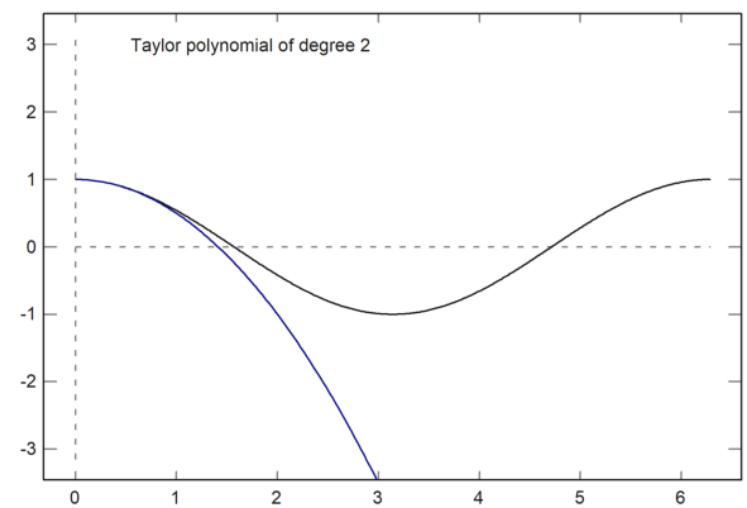
It is possible to restrict the dragged values to integers. For an example, we write a plot function, which plots a Taylor polynomial of degree n to the cosine function.

```
>function plotf(n) ...
```

```
  plot2d("cos(x)",0,2pi,>square,grid=6);
  plot2d(&"taylor(cos(x),x,0,@n)",color=blue,>add);
  textbox("Taylor polynomial of degree "+n,0.1,0.02,style="t",>left);
  endfunction
```

Now we allow the degree n to vary from 0 to 20 in 20 stops. The result of dragvalues() is used to plot the sketch with this n, and to insert the plot into the notebook.

```
>nd=dragvalues("plotf","degree",2,[0,20],20,y=0.8, ...
>   heading="Drag the value:"); ...
>plotf(nd):
```



The following is a simple demonstration of the function. The user can draw over the plot window, leaving a trace of points.

```
>function dragtest ...
```

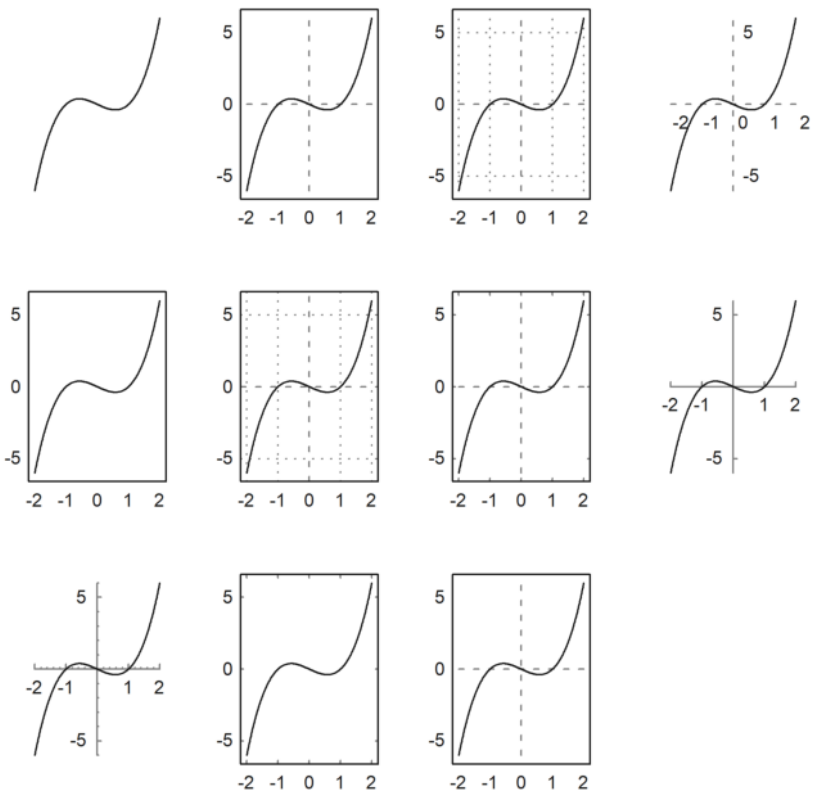
```
    plot2d(none,r=1,title="Drag with the mouse, or press any key!");
    start=0;
    repeat
      {flag,m,time}=mousedrag();
      if flag==0 then return; endif;
      if flag==2 then
        hold on; mark(m[1],m[2]); hold off;
      endif;
    end
  endfunction
```

```
>dragtest // lihat hasilnya dan cobalah lakukan!
```

Styles of 2D Plots

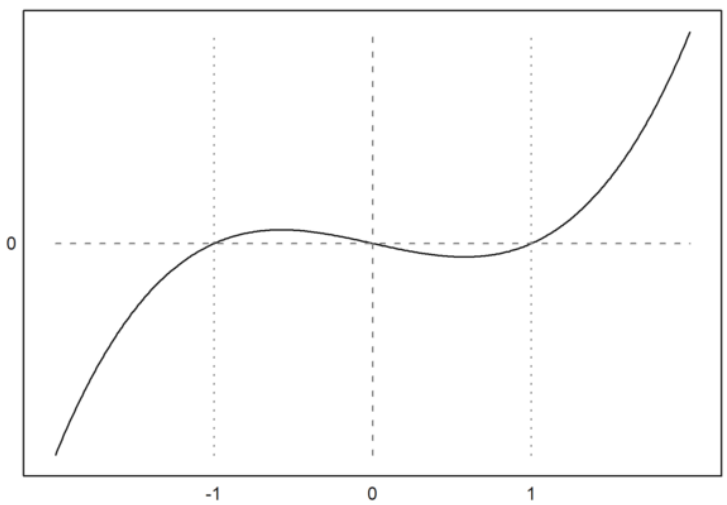
By default, EMT computes automatic axis ticks and adds labels to each tick. This can be changed with the grid parameter. The default style of the axis and the labels can be modified. Additionally, labels and a title can be added manually. To reset to the default styles, use reset().

```
>aspect();
>figure(3,4); ...
> figure(1); plot2d("x^3-x",grid=0); ... // no grid, frame or axis
> figure(2); plot2d("x^3-x",grid=1); ... // x-y-axis
> figure(3); plot2d("x^3-x",grid=2); ... // default ticks
> figure(4); plot2d("x^3-x",grid=3); ... // x-y- axis with labels inside
> figure(5); plot2d("x^3-x",grid=4); ... // no ticks, only labels
> figure(6); plot2d("x^3-x",grid=5); ... // default, but no margin
> figure(7); plot2d("x^3-x",grid=6); ... // axes only
> figure(8); plot2d("x^3-x",grid=7); ... // axes only, ticks at axis
> figure(9); plot2d("x^3-x",grid=8); ... // axes only, finer ticks at axis
> figure(10); plot2d("x^3-x",grid=9); ... // default, small ticks inside
> figure(11); plot2d("x^3-x",grid=10); ...// no ticks, axes only
> figure(0):
```



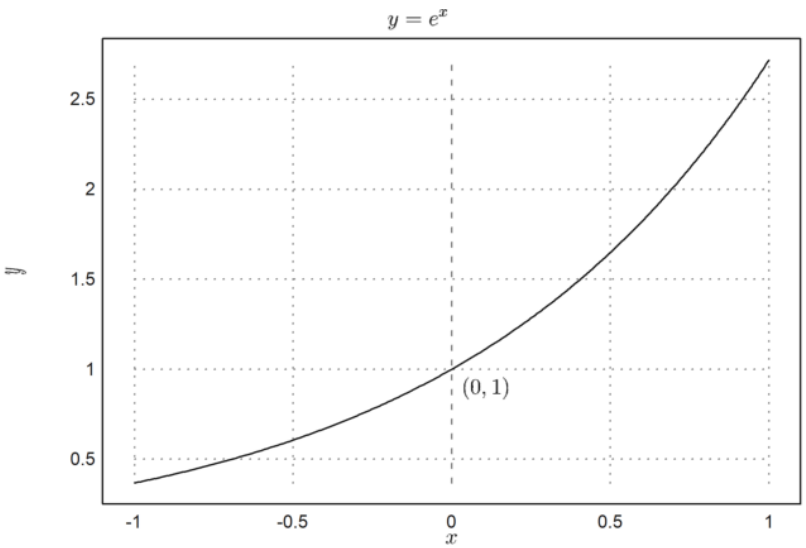
The parameter <frame switches off the frame, and framecolor=blue sets the frame to a blue color.
If you want your own ticks, you can use style=0, and add everything later.

```
>aspect(1.5);
>plot2d("x^3-x",grid=0); // plot
>frame; xgrid([-1,0,1]); ygrid(0): // add frame and grid
```



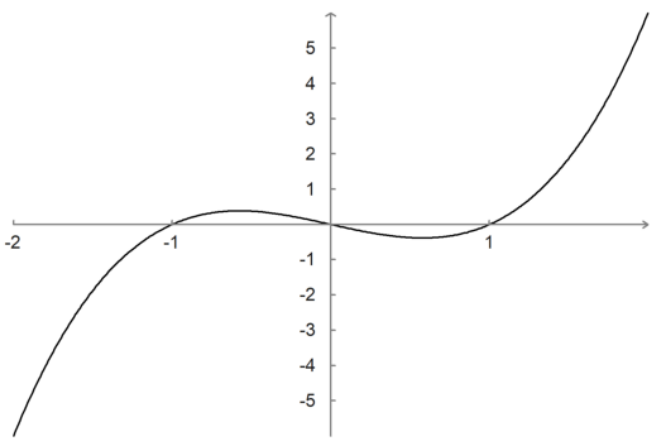
For the plot title and labels of the axes, see the following example.

```
>plot2d("exp(x)",-1,1);
>textcolor(black); // set the text color to black
>title(latex("y=e^x")); // title above the plot
>xlabel(latex("x")); // "x" for x-axis
>ylabel(latex("y"),>vertical); // vertical "y" for y-axis
>label(latex("(0,1)"),0,1,color=blue): // label a point
```



The axis can be drawn separately with xaxis() and yaxis().

```
>plot2d("x^3-x",<grid,<frame);
>xaxis(0,xx=-2:1,style="->"); yaxis(0,yy=-5:5,style="->"):
```

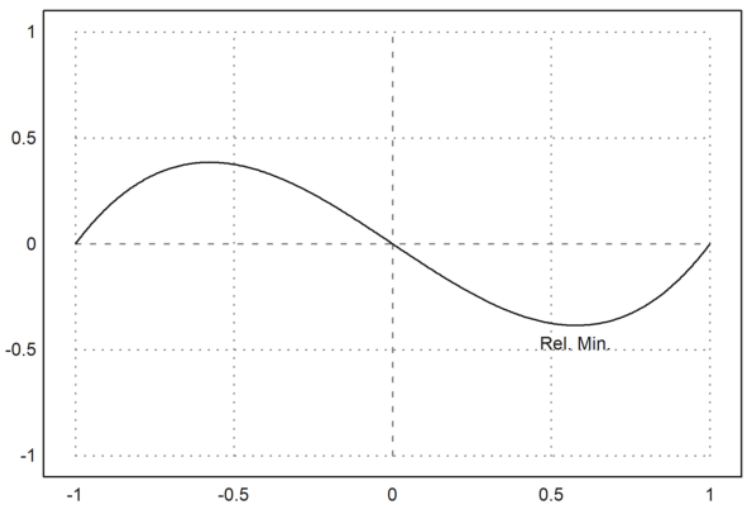


Text on the plot can be set with label(). In the following example, "lc" means lower center. It sets the position of the label relative to the plot coordinates.

```
>function f(x) &= x^3-x
```

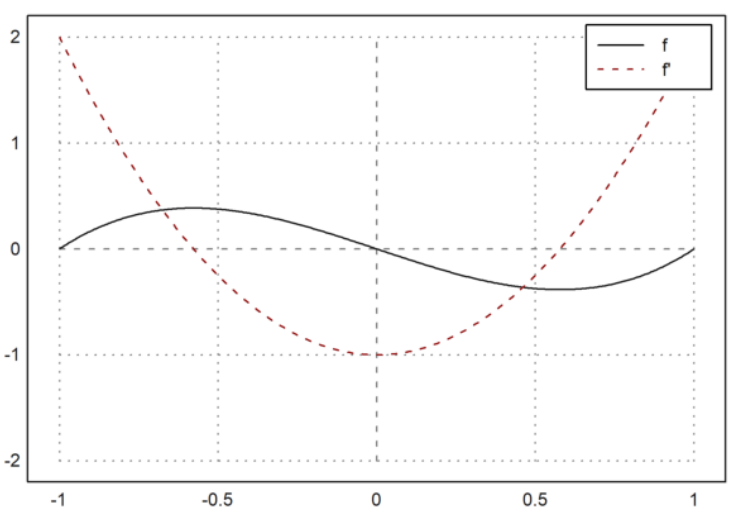
$$x^3 - x$$

```
>plot2d(f,-1,1,>square);
>x0=fmin(f,0,1); // compute point of minimum
>label("Rel. Min.",x0,f(x0),pos="lc"): // add a label there
```

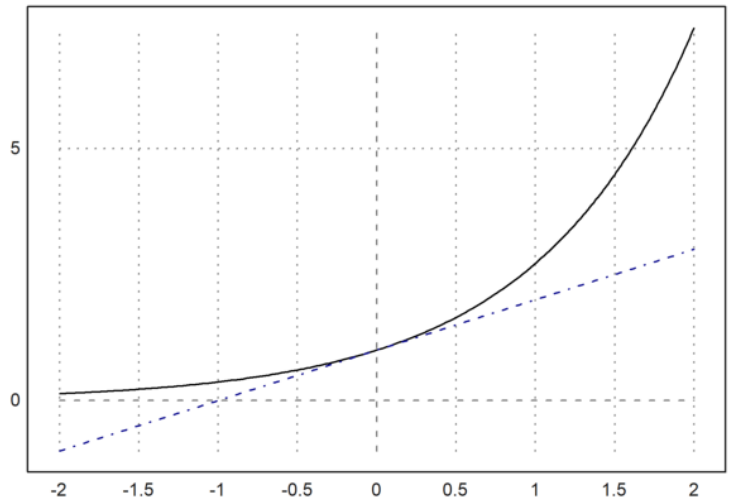


There are also text boxes.

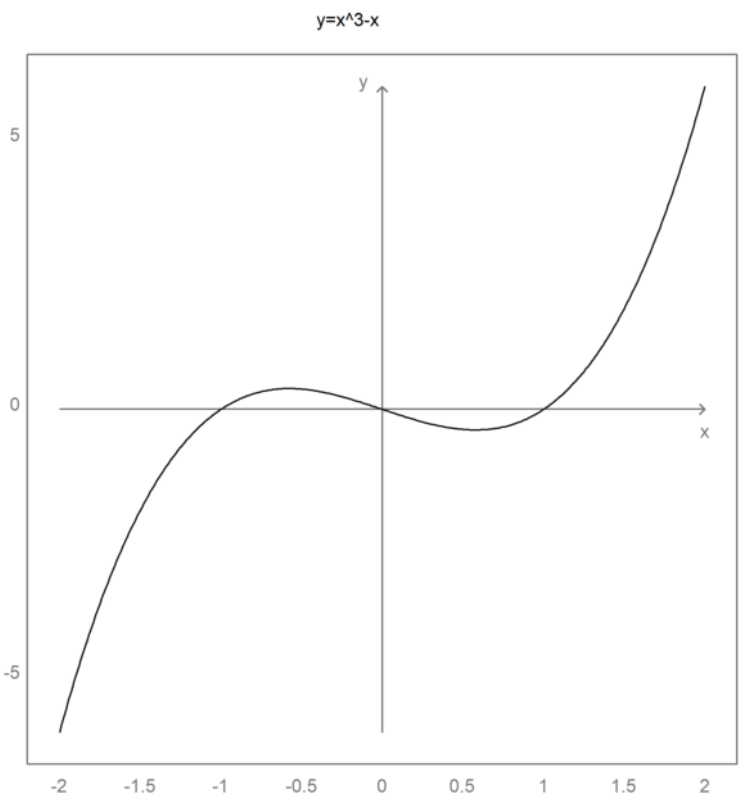
```
>plot2d(&f(x),-1,1,-2,2); // function
>plot2d(&diff(f(x),x),>add,style="--",color=red); // derivative
>labelbox(["f","f'"],["-","--"],[black,red]): // label box
```



```
>plot2d(["exp(x)","1+x"],color=[black,blue],style=["-","-.-"]):
```

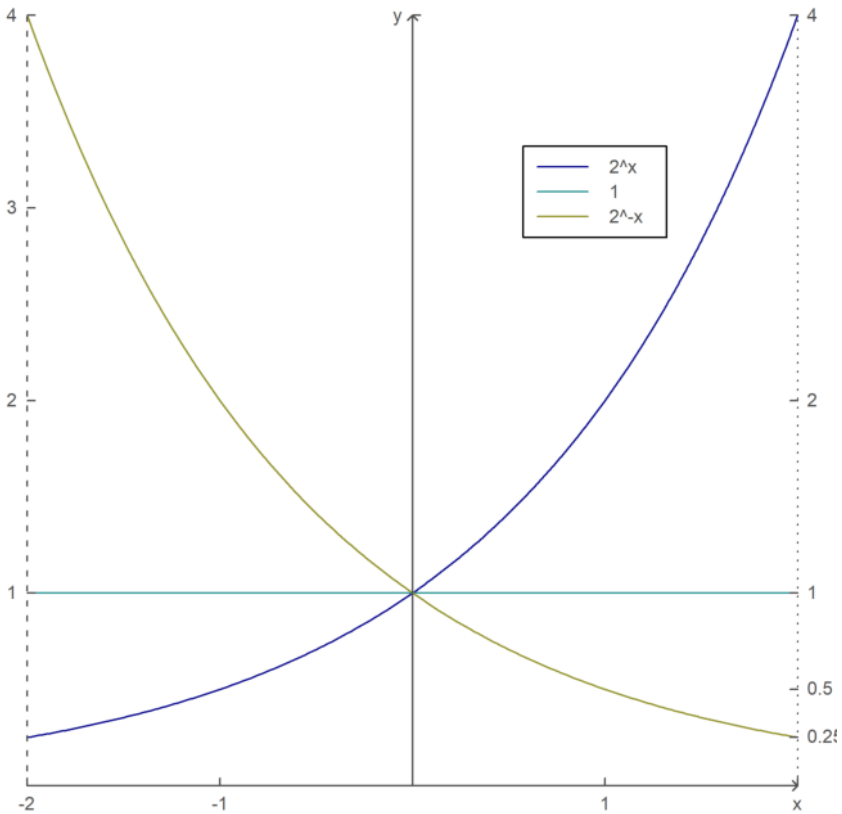



```
>gridstyle("->",color=gray,textcolor=gray,framecolor=gray);  ...
> plot2d("x^3-x",grid=1);   ...
> settitle("y=x^3-x",color=black); ...
> label("x",2,0,pos="bc",color=gray);  ...
> label("y",0,6,pos="cl",color=gray); ...
> reset():
```



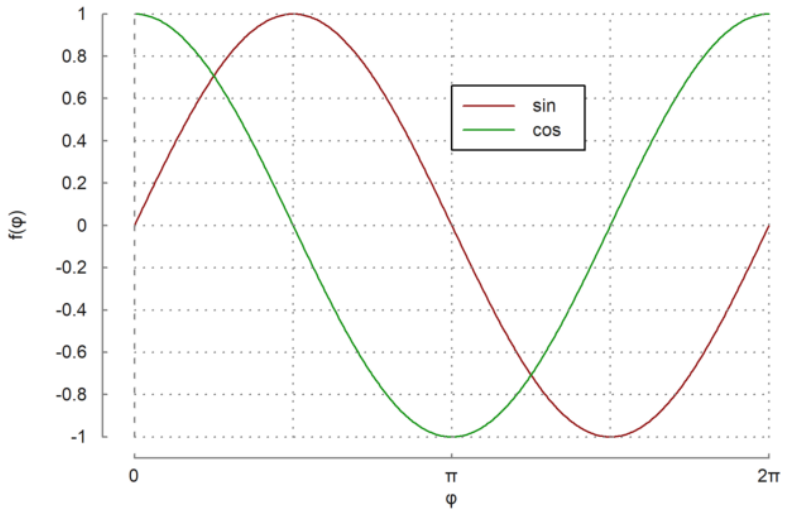
For even more control, the x-axis and the y-axis can be done manually.
The command fullwindow() expands the plot window since we no longer need place for labels outside the plot window. Use shrinkwindow() or reset() to reset to the defaults.

```
>fullwindow; ...
> gridstyle(color=darkgray,textcolor=darkgray); ...
> plot2d(["2^x","1","2^(-x)"],a=-2,b=2,c=0,d=4,<grid,color=4:6,<frame); ...
> xaxis(0,-2:1,style="->"); xaxis(0,2,"x",<axis); ...
> yaxis(0,4,"y",style="->"); ...
> yaxis(-2,1:4,>left); ...
> yaxis(2,2^(-2:2),style=".",<left); ...
> labelbox(["2^x","1","2^-x"],colors=4:6,x=0.8,y=0.2); ...
> reset:
```



Here is another example, where Unicode strings are used and axes outside the plot area.

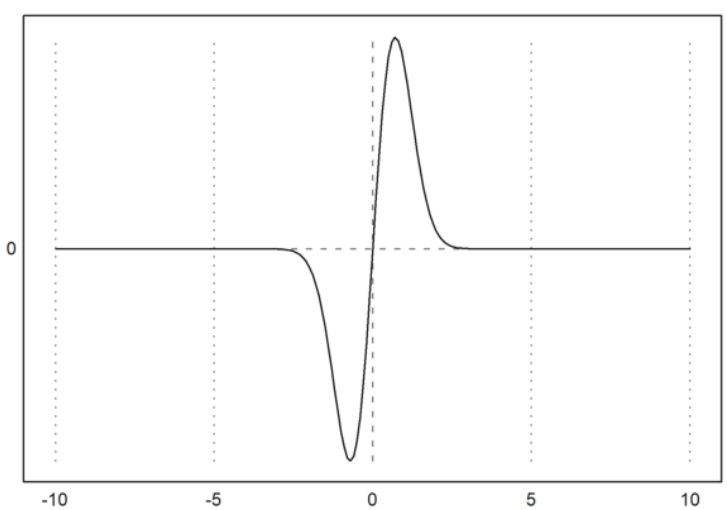
```
>aspect(1.5);
>plot2d(["sin(x)","cos(x)"],0,2pi,color=[red,green],<grid,<frame); ...
> xaxis(-1.1,(0:2)*pi,xt=["0",u"&pi;",u"2&pi;"],style="-",>ticks,>zero);  ...
> xgrid((0:0.5:2)*pi,<ticks); ...
> yaxis(-0.1*pi,-1:0.2:1,style="-",>zero,>grid); ...
> labelbox(["sin","cos"],colors=[red,green],x=0.5,y=0.2,>left); ...
> xlabel(u"&phi;"); ylabel(u"f(&phi;)"):
```



Plotting 2D Data

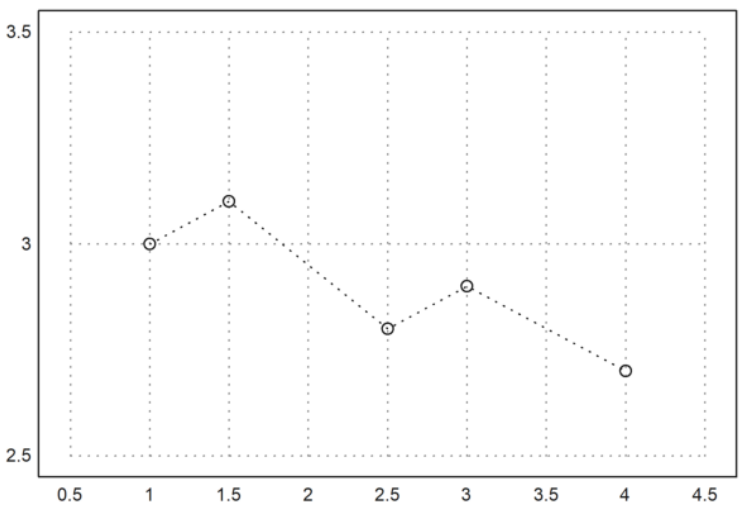
If x and y are data vectors, these data will be used as x- and y-coordinates of a curve. In this case, a, b, c, and d, or a radius r can be specified, or the plot window will adjust automatically to the data. Alternatively, >square can be set to keep a square aspect ratio.
Plotting an expression is only an abbreviation for data plots. For data plots, you need one or more rows of x-values, and one or more rows of y-values. From the range and the x-values the plot2d function will compute the data to plot, by default with adaptive evaluation of the function. For point plots use ">points", for mixed lines and points use ">addpoints".
But you can enter data directly.
- Use row vectors for x and y for one function.
- Matrices for x and y are plotted line by line.
Here is an example with one row for x and y.

```
>x=-10:0.1:10; y=exp(-x^2)*x; plot2d(x,y):
```

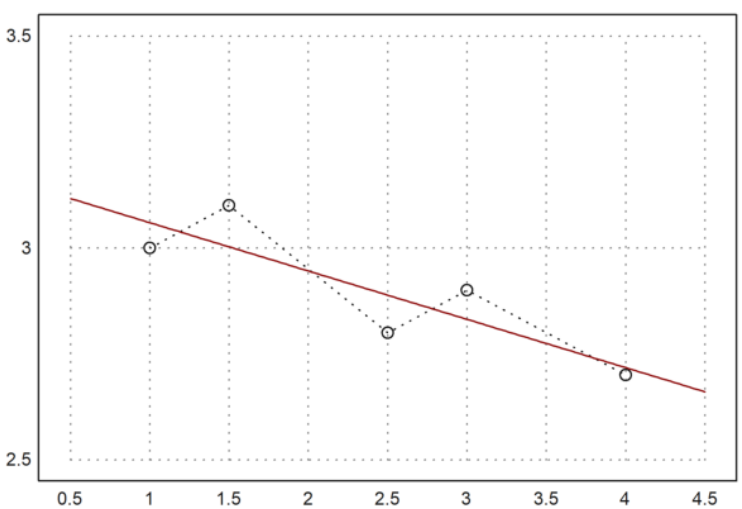


Data can also be plotted as points. Use points=true for this. The plot works like polygons, but draws only the corners.
- style="...": Select from "[]", "<>", "o", ".", "..", "+", "*", "[]", "<>", "o", "..", "", "|".
To plot sets of points use >points. If the color is a vector of colors, each points
gets a different color. For a matrix of coordinates and a column vector, the color applies to the rows of the matrix.
The parameter >addpoints adds points to line segments for plots of data.

```
>xdata=[1,1.5,2.5,3,4]; ydata=[3,3.1,2.8,2.9,2.7]; // data
>plot2d(xdata,ydata,a=0.5,b=4.5,c=2.5,d=3.5,style="."); // lines
>plot2d(xdata,ydata,>points,>add,style="o"): // add points
```



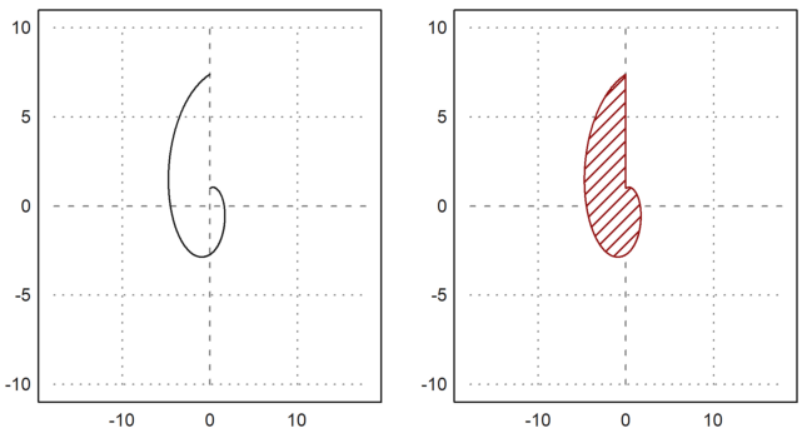

```
>p=polyfit(xdata,ydata,1); // get regression line
>plot2d("polyval(p,x)",>add,color=red): // add plot of line
```



Menggambar Daerah Yang Dibatasi Kurva

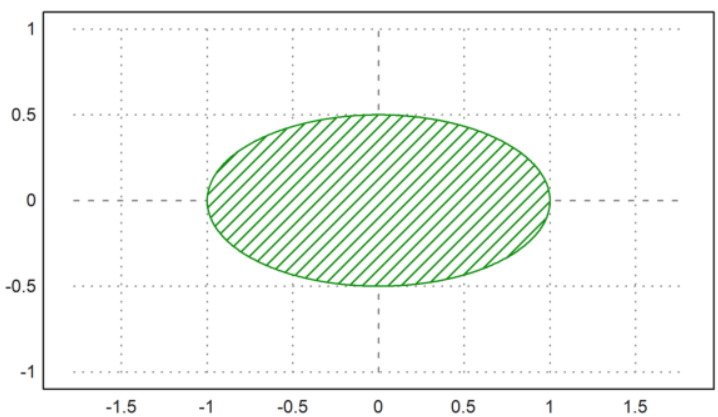
Data plots are really polygons. We can also plot curves or filled curves.
- filled=true fills the plot.
- style="...": Select from "", "/", "\", "\/".
- fillcolor: See above for available colors.
The fill color is determined by the argument "fillcolor", and on optional <outline prevents drawing the boundary for all styles but the default one.

```
>t=linspace(0,2pi,1000); // parameter for curve
>x=sin(t)*exp(t/pi); y=cos(t)*exp(t/pi); // x(t) and y(t)
>figure(1,2); aspect(16/9)
>figure(1); plot2d(x,y,r=10); // plot curve
>figure(2); plot2d(x,y,r=10,>filled,style="/",fillcolor=red); // fill curve
>figure(0):
```

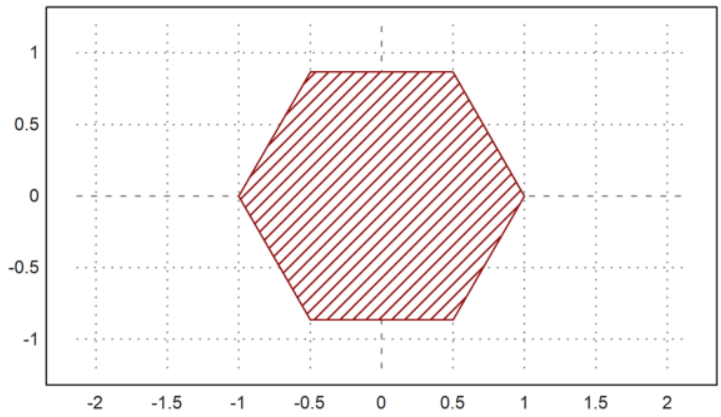


In the following examples we plot a filled ellips and two filled hexagons using a closed curve with 6 points with different fill style.

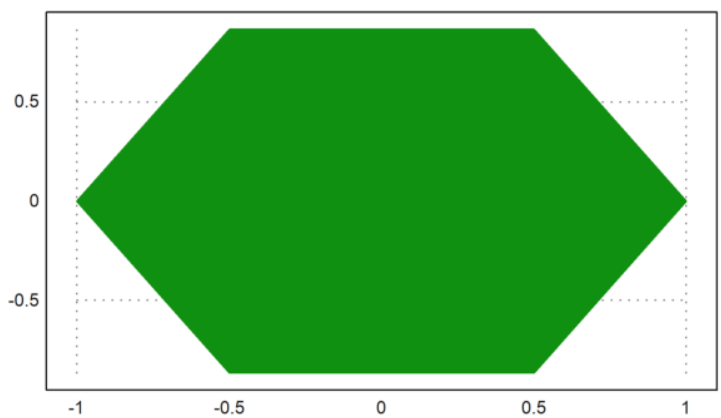
```
>x=linspace(0,2pi,1000); plot2d(sin(x),cos(x)*0.5,r=1,>filled,style="/"):
```



```
>t=linspace(0,2pi,6); ...
>plot2d(cos(t),sin(t),>filled,style="/",fillcolor=red,r=1.2):
```

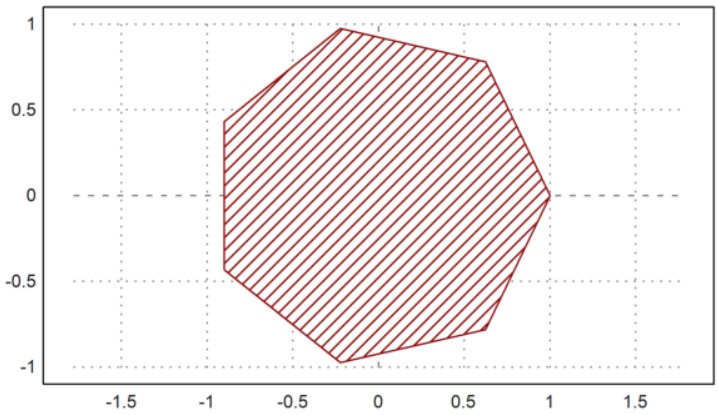



```
>t=linspace(0,2pi,6); plot2d(cos(t),sin(t),>filled,style="#"):
```



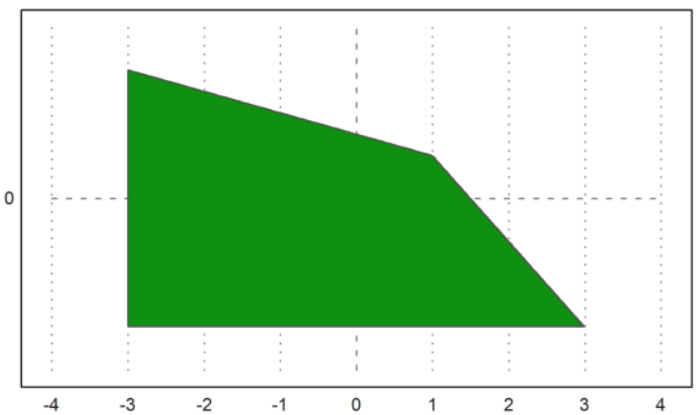
Another example is a septagon, which we create with 7 points on the unit circle.

```
>t=linspace(0,2pi,7);  ...
> plot2d(cos(t),sin(t),r=1,>filled,style="/",fillcolor=red):
```



The following is the set of the maximal value of four linear conditions less than or equal 3. This is A[k].v<=3 for all rows of A. To get nice corners, we use n relatively large.

```
>A=[2,1;1,2;-1,0;0,-1];
>function f(x,y) := max([x,y].A');
>plot2d("f",r=4,level=[0;3],color=green,n=111):
```

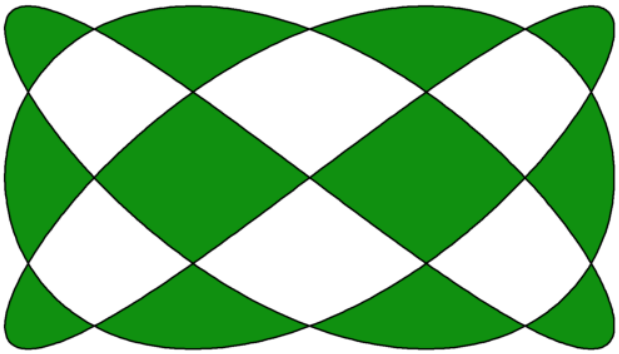


The main point of the matrix language is that it allows to generate tables of functions easily.

```
>t=linspace(0,2pi,1000); x=cos(3*t); y=sin(4*t);
```

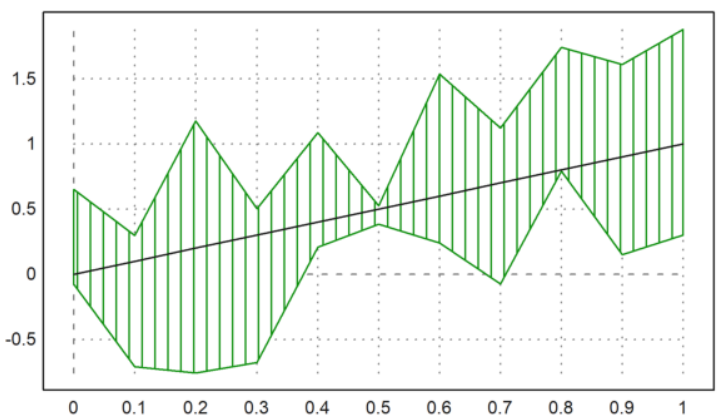
We now have vectors x and y of values. plot2d() can plot these values as a curve connecting the points. The plot can be filled. In this case this yields a nice result due to the winding rule, which is used for the fill.

```
>plot2d(x,y,<grid,<frame,>filled):
```



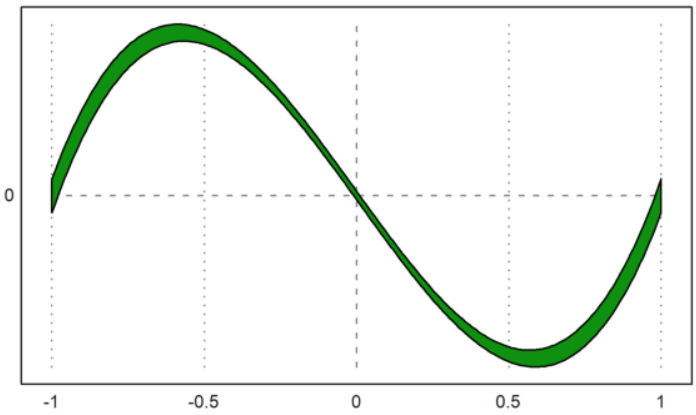
A vector of intervals is plotted against x values as a filled region between lower and upper values of the intervals.
This is can be useful to plot errors of the computation. But it can also be used to plot statistical errors.

```
>t=0:0.1:1; ...
> plot2d(t,interval(t-random(size(t)),t+random(size(t))),style="|");  ...
> plot2d(t,t,add=true):
```



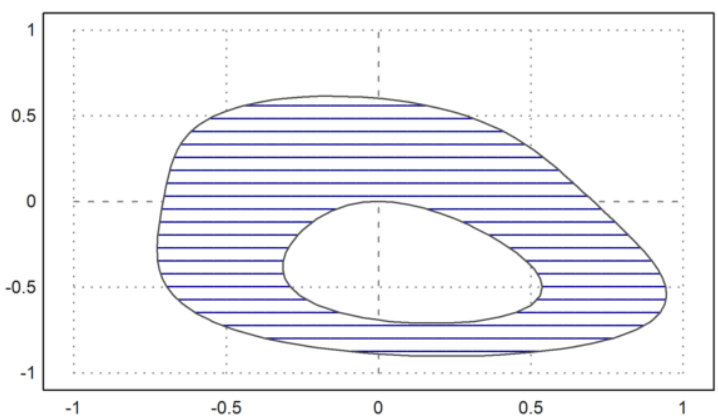
If x is a sorted vector, and y is a vector of intervals, then plot2d will plot the filled ranges of the intervals in the plane.The fill styles are the same as the styles of polygons.

```
>t=-1:0.01:1; x=~t-0.01,t+0.01~; y=x^3-x;
>plot2d(t,y):
```



It is possible to fill regions of values for a specific function. For this, level must be a 2xn matrix. The first row are the lower bounds and the second row contains the upper bounds.

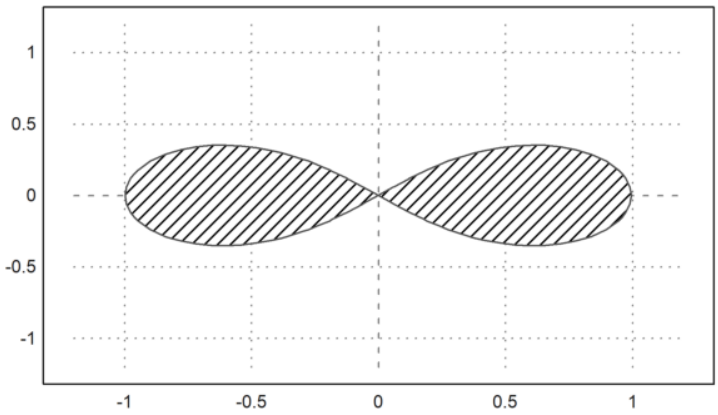
```
>expr := "2*x^2+x*y+3*y^4+y"; // define an expression f(x,y)
>plot2d(expr,level=[0;1],style="-",color=blue): // 0 <= f(x,y) <= 1
```



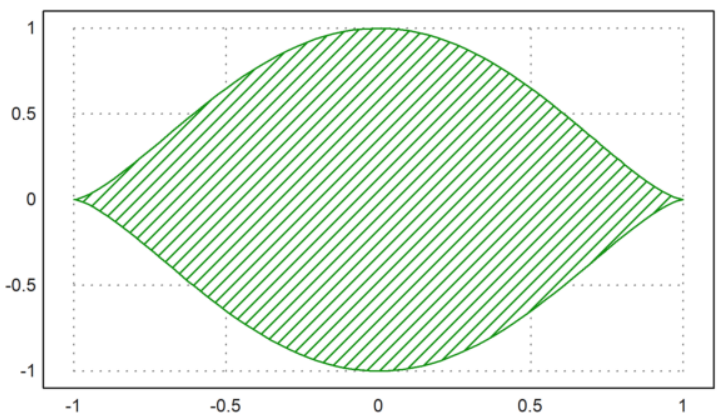
We can also fill ranges of values like

$$-1 \le (x^2+y^2)^2 - x^2 + y^2 \le 0.$$

```
>plot2d("(x^2+y^2)^2-x^2+y^2",r=1.2,level=[-1;0],style="/"):
```




```
>plot2d("cos(x)","sin(x)^3",xmin=0,xmax=2pi,>filled,style="/"):
```



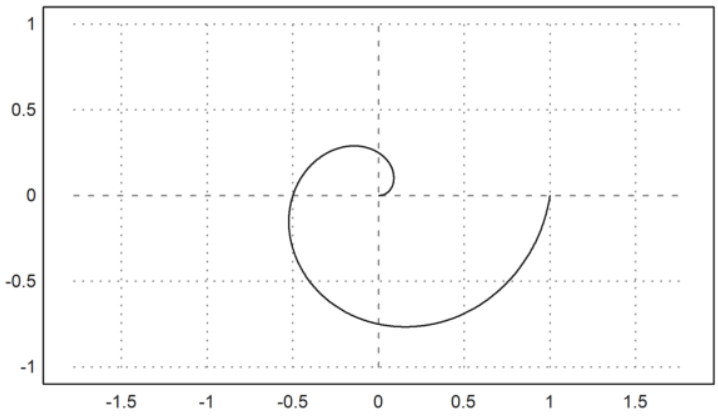
Grafik Fungsi Parametrik

The x-values need not be sorted. (x,y) simply describes a curve. If x is sorted, the curve is a graph of a function. In the following example, we plot the spiral

$$\gamma(t) = t \cdot (\cos(2\pi t), \sin(2\pi t))$$

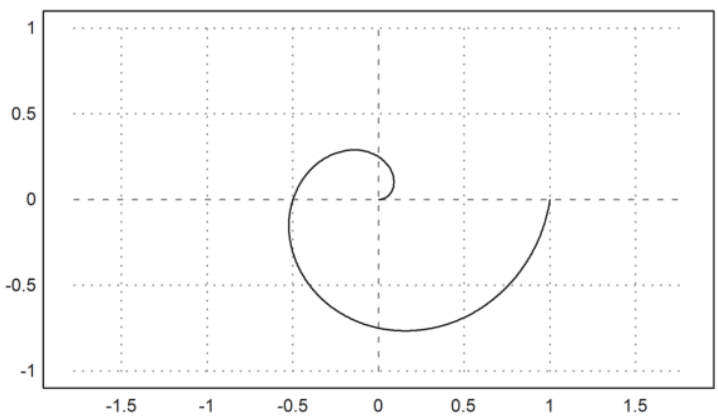
We either need to use very many points for a smooth look or the function adaptive() to evaluate the expressions (see the function adaptive() for more details).

```
>t=linspace(0,1,1000); ...
>plot2d(t*cos(2*pi*t),t*sin(2*pi*t),r=1):
```

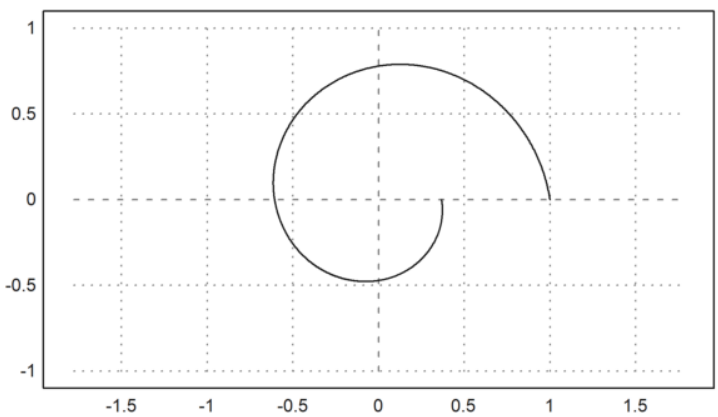


Alternatively, it is possible to use two expressions for curves. The following plots the same curve as above.

```
>plot2d("x*cos(2*pi*x)","x*sin(2*pi*x)",xmin=0,xmax=1,r=1):
```




```
>t=linspace(0,1,1000); r=exp(-t); x=r*cos(2pi*t); y=r*sin(2pi*t);
>plot2d(x,y,r=1):
```



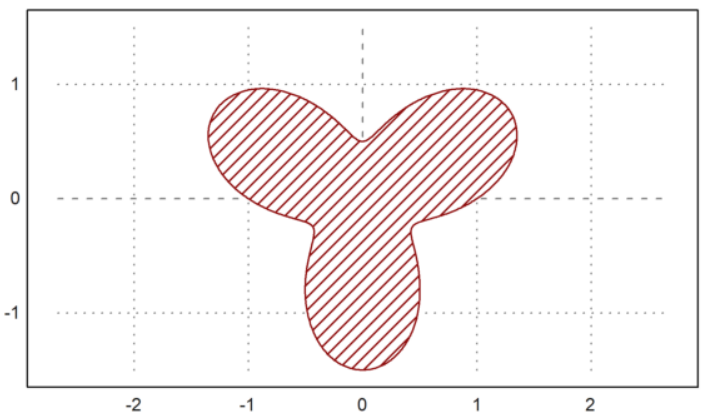
In the next example, we plot the curve

$$\gamma(t) = (r(t)\cos(t), r(t)\sin(t))$$

with

$$r(t) = 1 + \frac{\sin(3t)}{2}.$$

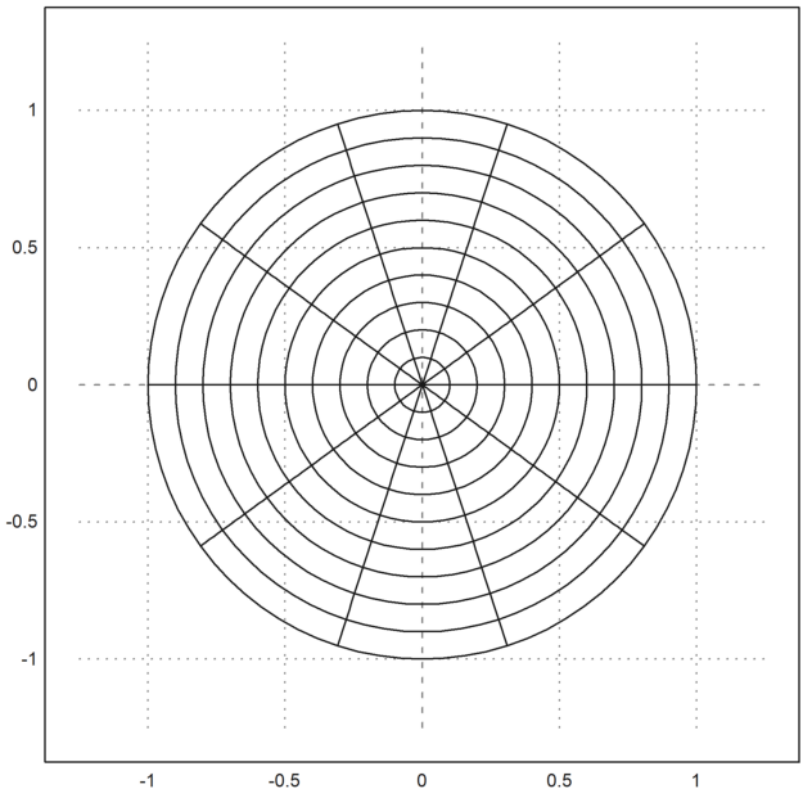
```
>t=linspace(0,2pi,1000); r=1+sin(3*t)/2; x=r*cos(t); y=r*sin(t); ...
>plot2d(x,y,>filled,fillcolor=red,style="/",r=1.5):
```



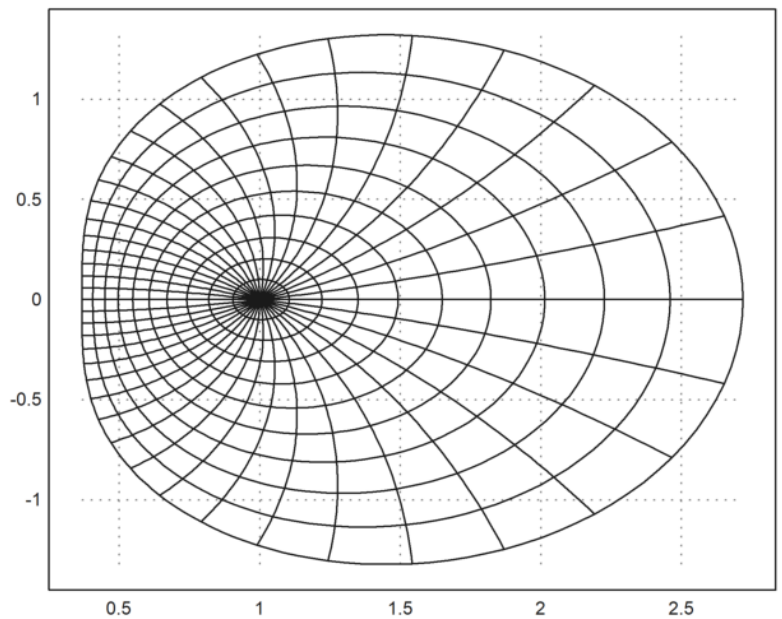
Menggambar Grafik Bilangan Kompleks

An array of complex numbers can also be plotted. Then the grid points will be connected. If a number of grid lines is specified (or a 1x2 vector of grid lines) in the argument cgrid only those grid lines are visible.
A matrix of complex numbers will automatically plot as a grid in the complex plane.
In the following example, we plot the image of the unit circle under the exponential function. The cgrid parameter hides some of the grid curves.

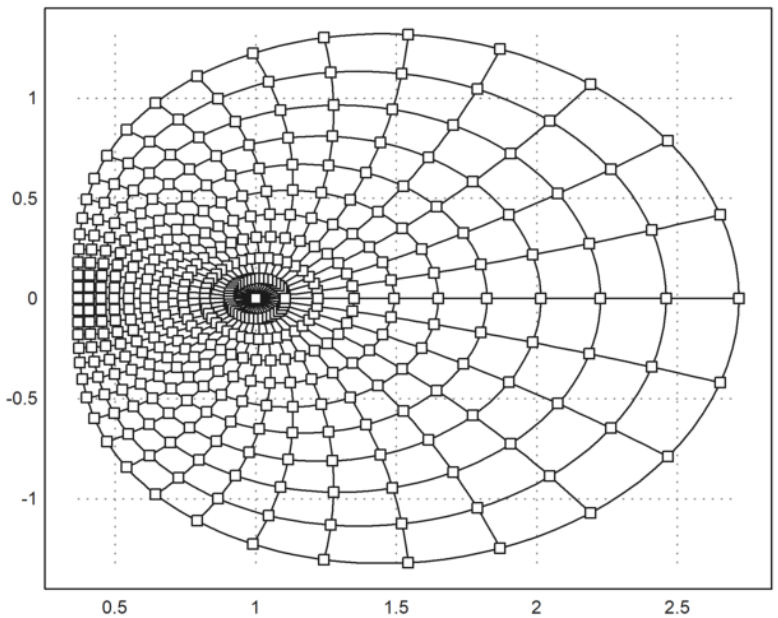
```
>aspect(); r=linspace(0,1,50); a=linspace(0,2pi,80)'; z=r*exp(I*a);...
>plot2d(z,a=-1.25,b=1.25,c=-1.25,d=1.25,cgrid=10):
```




```
>aspect(1.25); r=linspace(0,1,50); a=linspace(0,2pi,200)'; z=r*exp(I*a);
>plot2d(exp(z),cgrid=[40,10]):
```



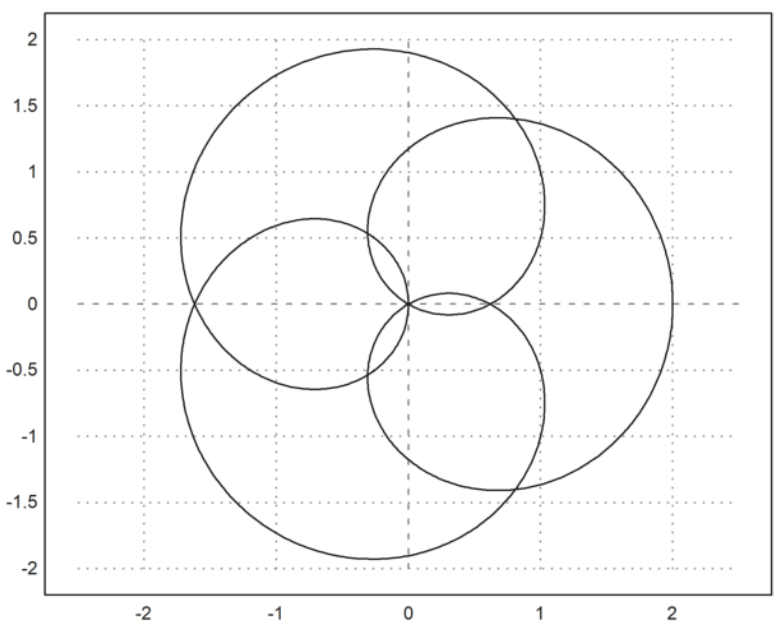
```
>r=linspace(0,1,10); a=linspace(0,2pi,40)'; z=r*exp(I*a);
>plot2d(exp(z),>points,>add):
```



A vector of complex numbers is automatically plotted as a curve in the complex plane with real part and imaginary part.
In the example, we plot the unit circle with

$$\gamma(t) = e^{it}$$

```
>t=linspace(0,2pi,1000); ...
>plot2d(exp(I*t)+exp(4*I*t),r=2):
```



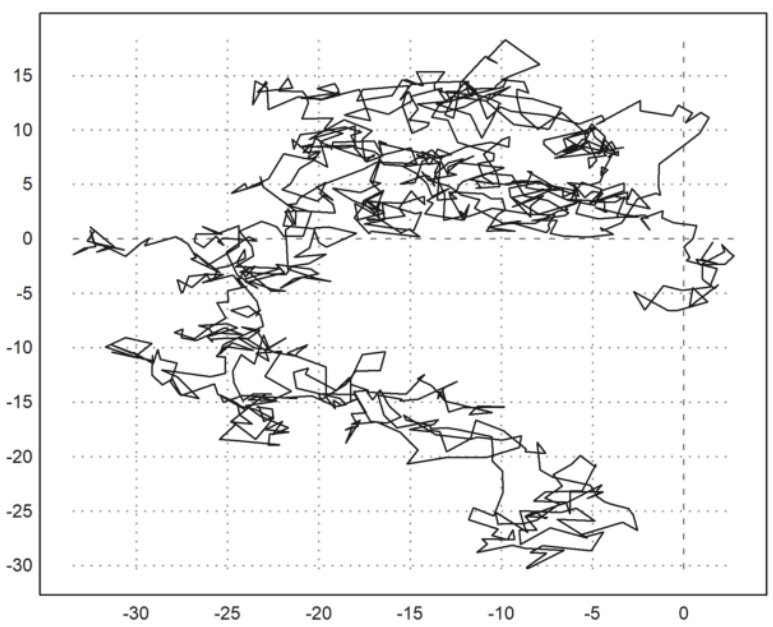
Statistical Plots

There are many functions which are specialized on statistical plots. One of the often used plots is a column plot.
A cumulative sum of a 0-1-normal distributed values produces a random walk.

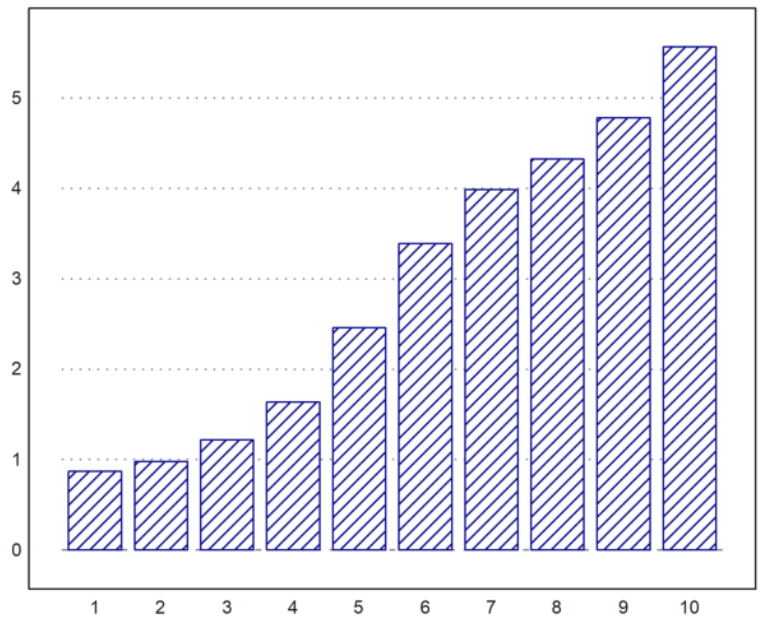
```
>plot2d(cumsum(randnormal(1,1000))):
```

Using two rows shows a walk in two dimensions.

```
>X=cumsum(randnormal(2,1000)); plot2d(X[1],X[2]):
```

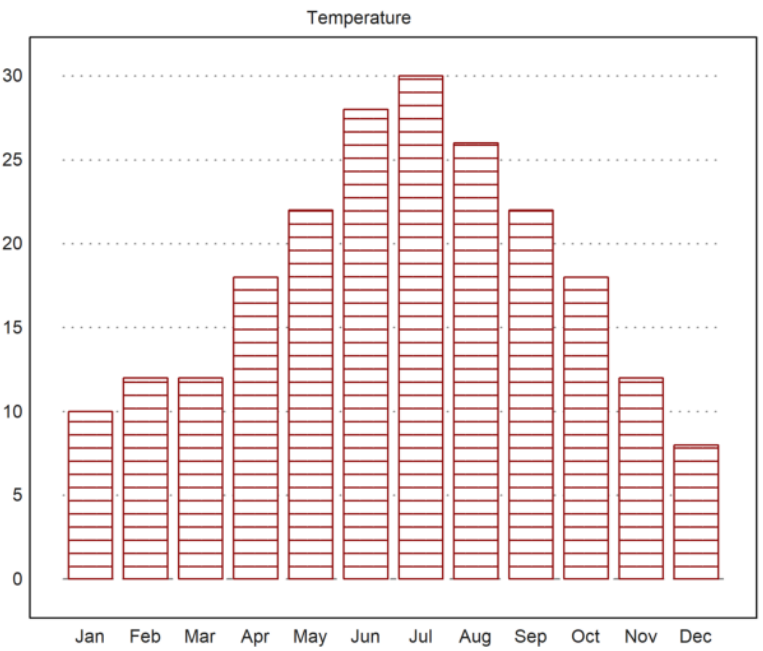


```
>columnsplot(cumsum(random(10)),style="/",color=blue):
```

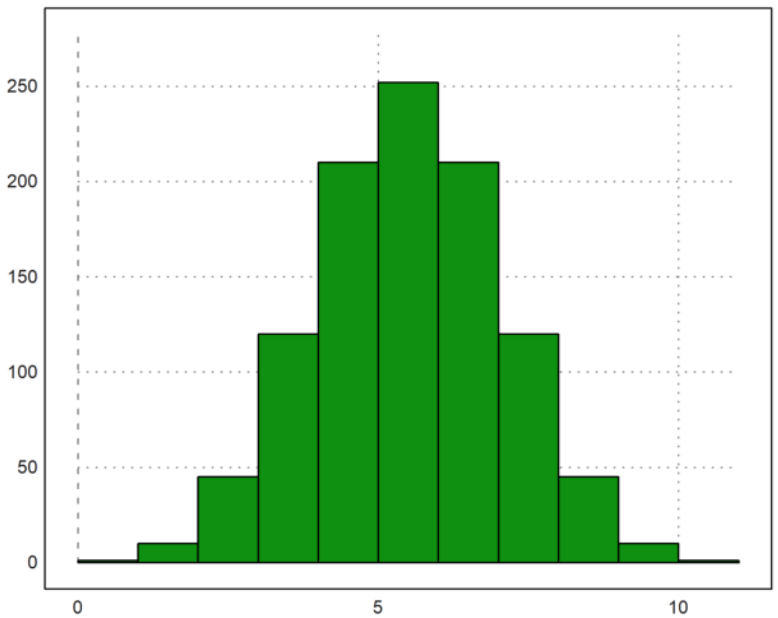


It can also show strings as labels.

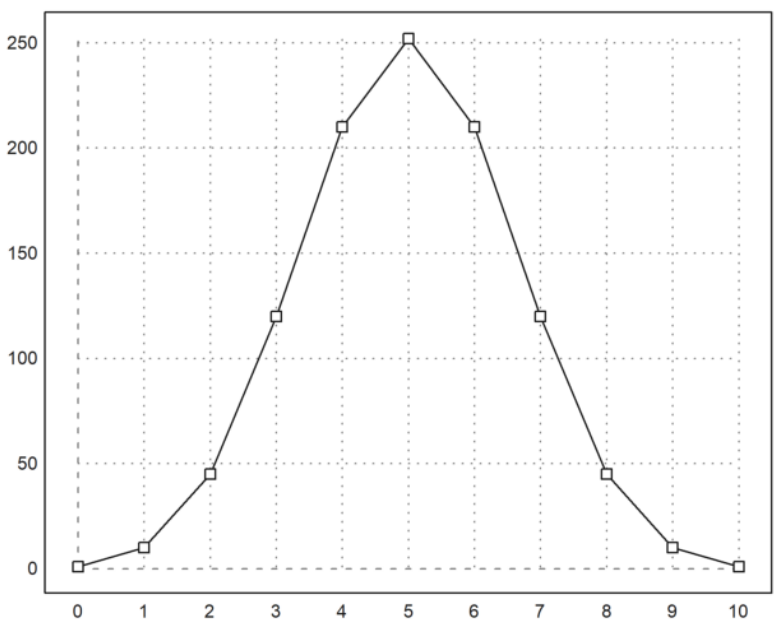
```
>months=["Jan","Feb","Mar","Apr","May","Jun", ...
>  "Jul","Aug","Sep","Oct","Nov","Dec"];
>values=[10,12,12,18,22,28,30,26,22,18,12,8];
>columnsplot(values,lab=months,color=red,style="-");
>title("Temperature"):
```



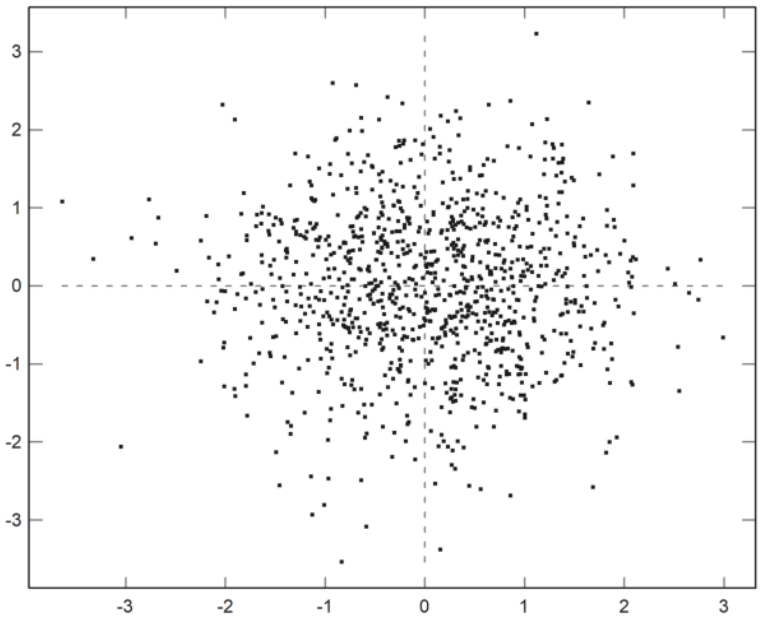
```
>k=0:10;
>plot2d(k,bin(10,k),>bar):
```



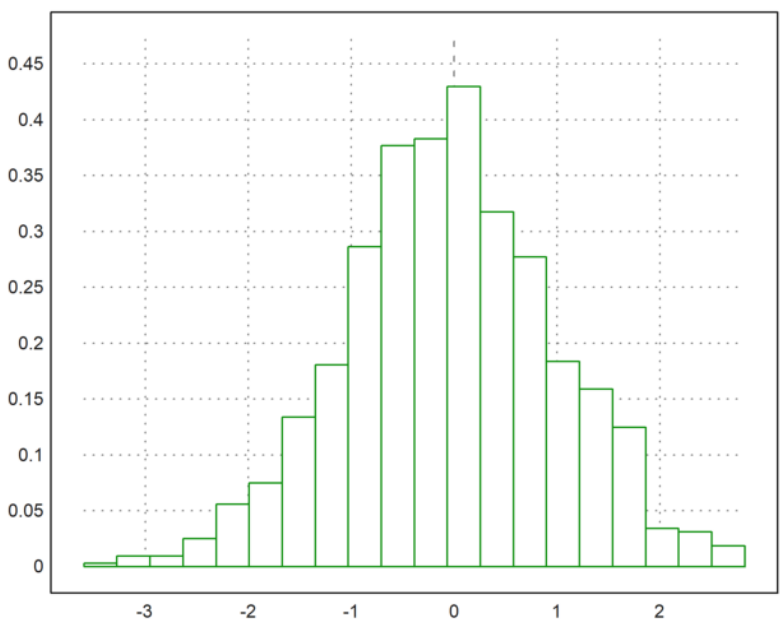

```
>plot2d(k,bin(10,k)); plot2d(k,bin(10,k),>points,>add):
```



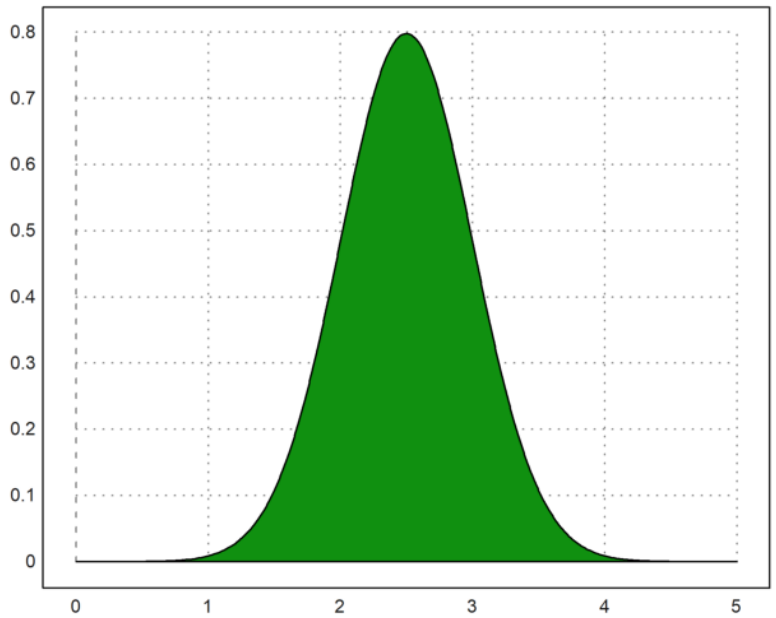

```
>plot2d(normal(1000),normal(1000),>points,grid=6,style=".."):
```




```
>plot2d(normal(1,1000),>distribution,style="O"):
```

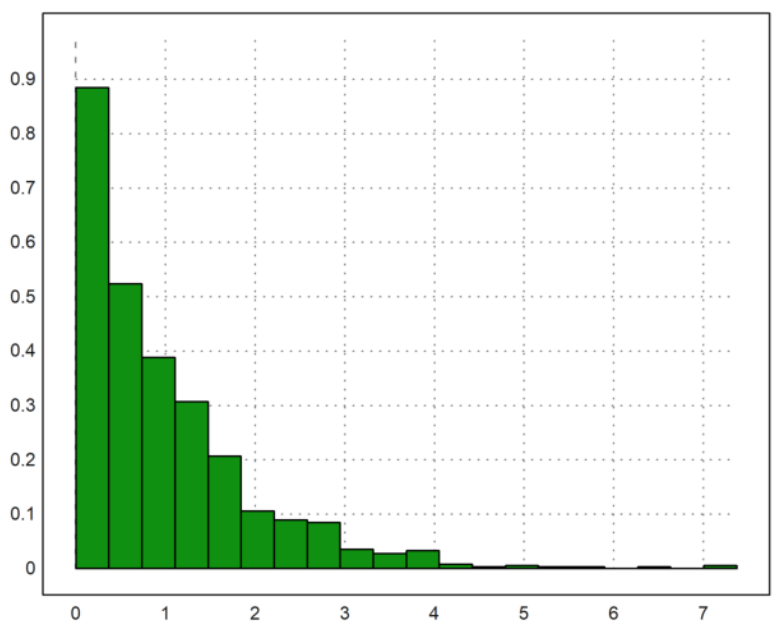



```
>plot2d("qnormal",0,5;2.5,0.5,>filled):
```



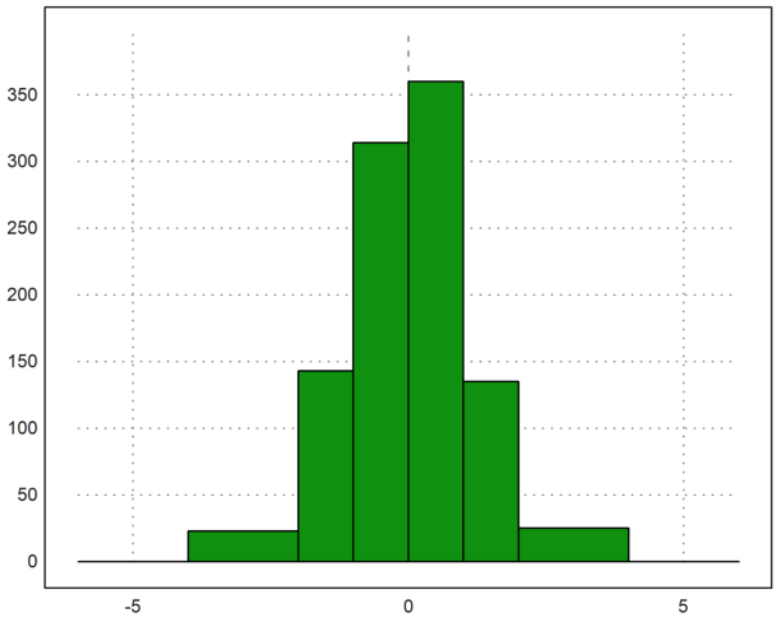
To plot an experimental statistical distribution, you can use distribution=n with plot2d.

```
>w=randexponential(1,1000); // exponential distribution
>plot2d(w,>distribution): // or distribution=n with n intervals
```



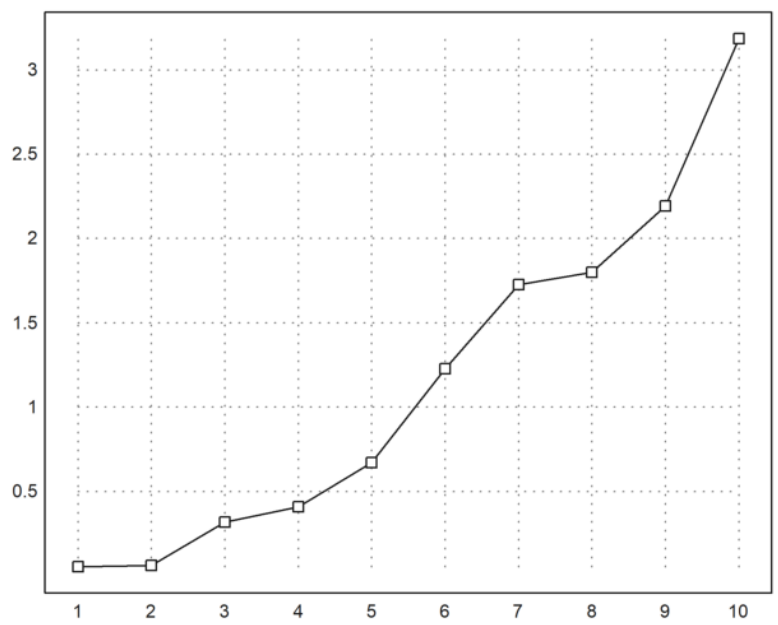
Or you can compute the distribution from the data and plot the result with >bar in plot3d, or with a column plot.

```
>w=normal(1000); // 0-1-normal distribution
>{x,y}=histo(w,10,v=[-6,-4,-2,-1,0,1,2,4,6]); // interval bounds v
>plot2d(x,y,>bar):
```

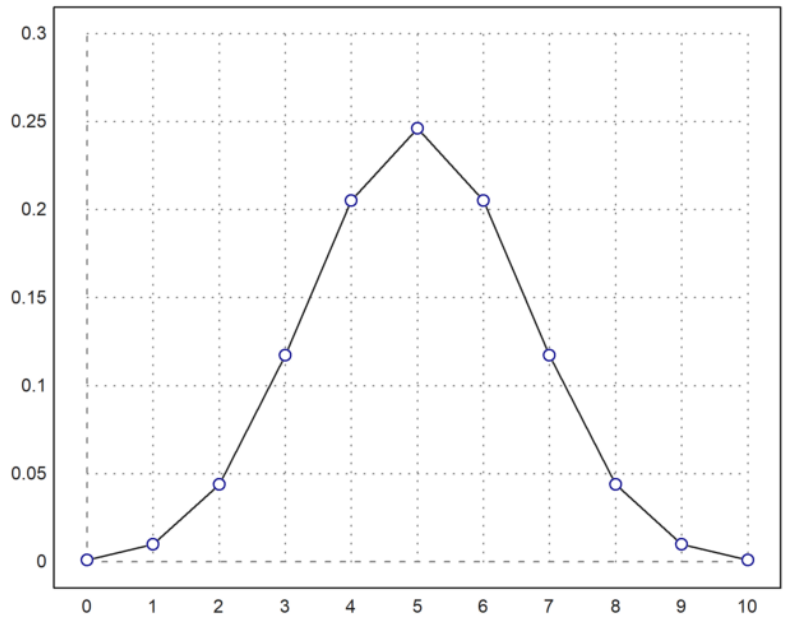


The statplot() function sets the style with a simple string.

```
>statplot(1:10,cumsum(random(10)),"b"):
```

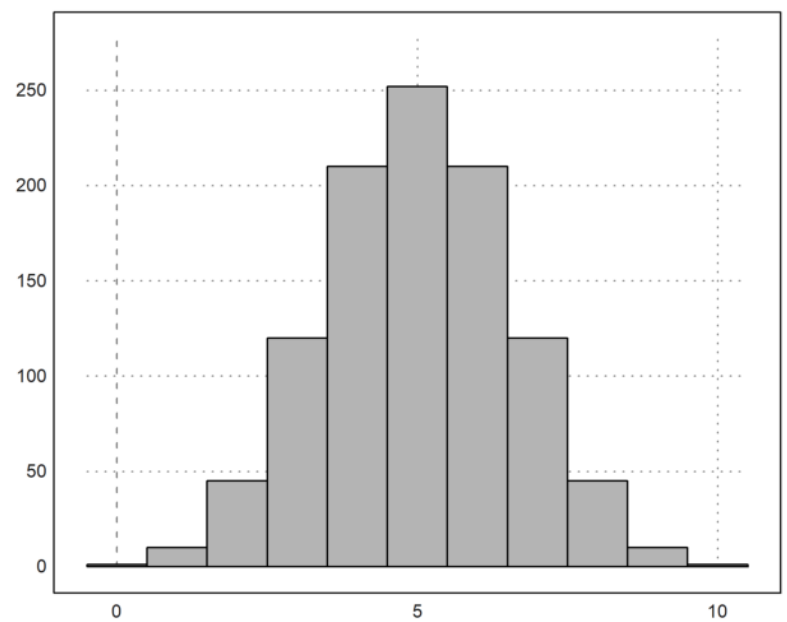



```
>n=10; i=0:n; ...
>plot2d(i,bin(n,i)/2^n,a=0,b=10,c=0,d=0.3); ...
>plot2d(i,bin(n,i)/2^n,points=true,style="ow",add=true,color=blue):
```



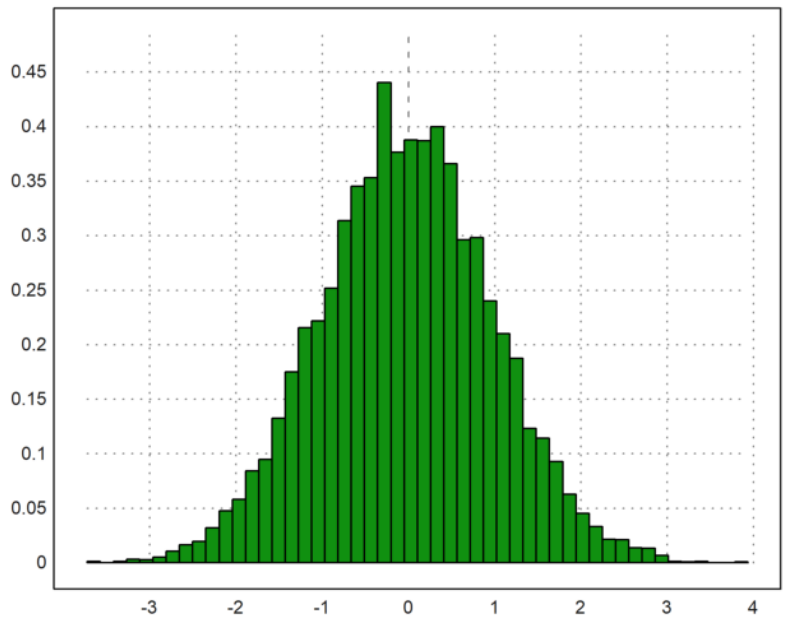
Moreover, data can be plotted as bars. In this case, x should be sorted and one element longer than y. The bars will extend from x[i] to x[i+1] with values y[i]. If x has the same size as y, it will be extended by one element with the last spacing.
Fill styles can be used just as above.

```
>n=10; k=bin(n,0:n); ...
>plot2d(-0.5:n+0.5,k,bar=true,fillcolor=lightgray):
```

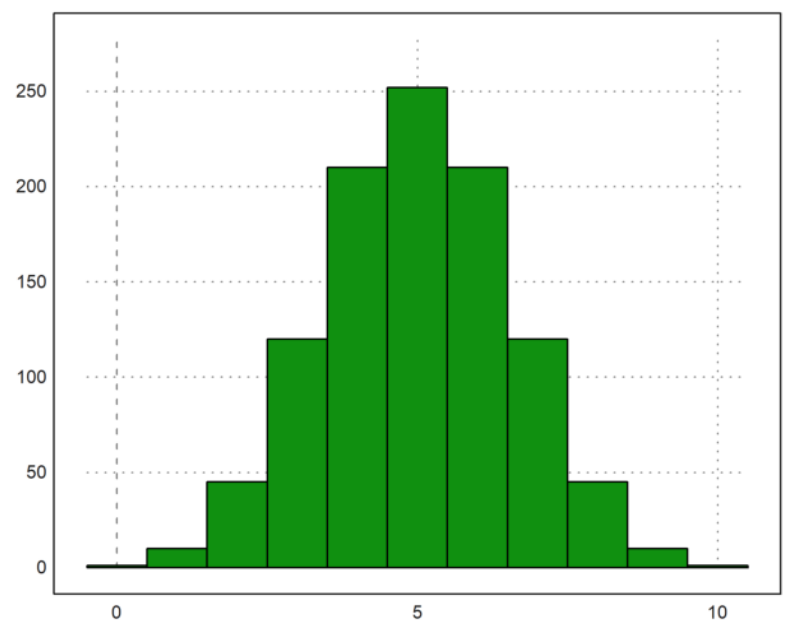


The data for bar plots (bar=1) and histograms (histogram=1) can either be explicitly given in xv and yv, or can be computed from an empirical distribution in xv with >distribution (or distribution=n). Histograms of xv values will be computed automatically with >histogram. If >even is specified, the xv values will be counted in integer intervals.

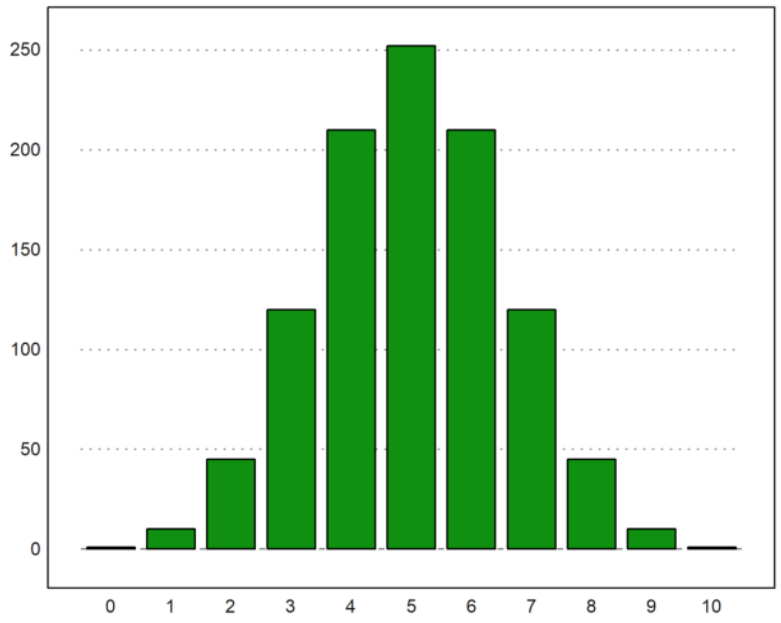
```
>plot2d(normal(10000),distribution=50):
```



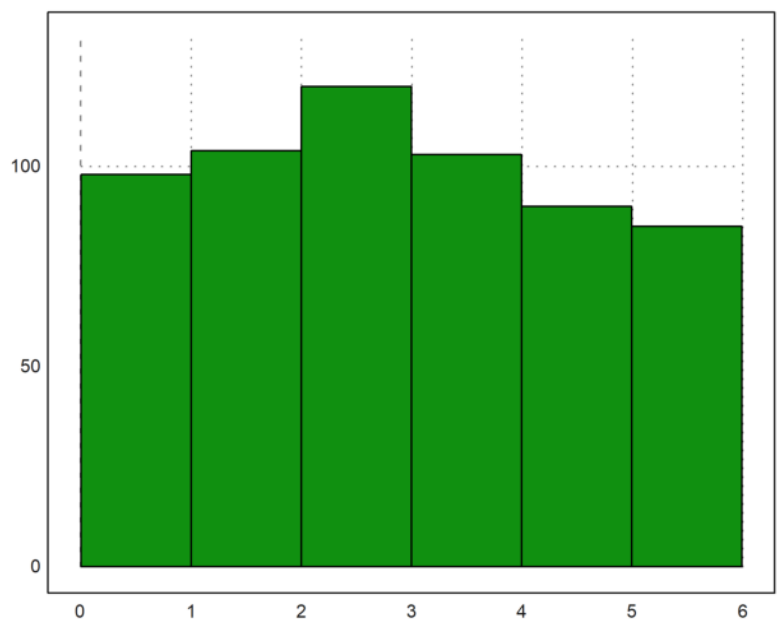

```
>k=0:10; m=bin(10,k); x=(0:11)-0.5; plot2d(x,m,>bar):
```




```
>columnsplot(m,k):
```

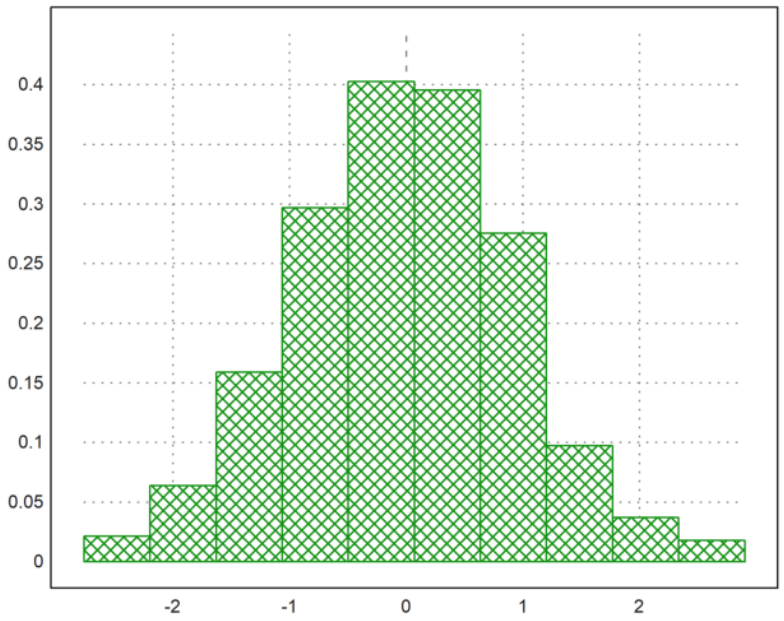



```
>plot2d(random(600)*6,histogram=6):
```



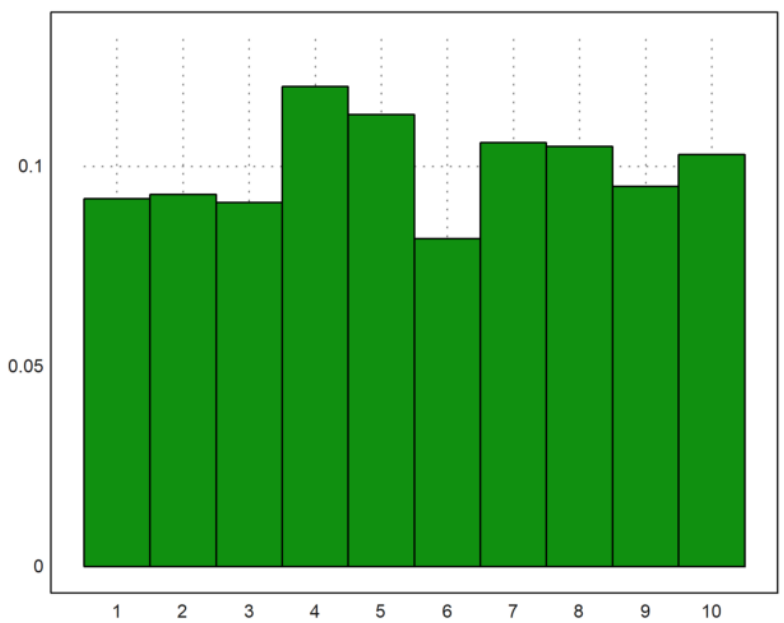
For distributions, there is the parameter distribution=n, which counts values automatically and prints the relative distribution with n sub-intervals.

```
>plot2d(normal(1,1000),distribution=10,style="\/"):
```



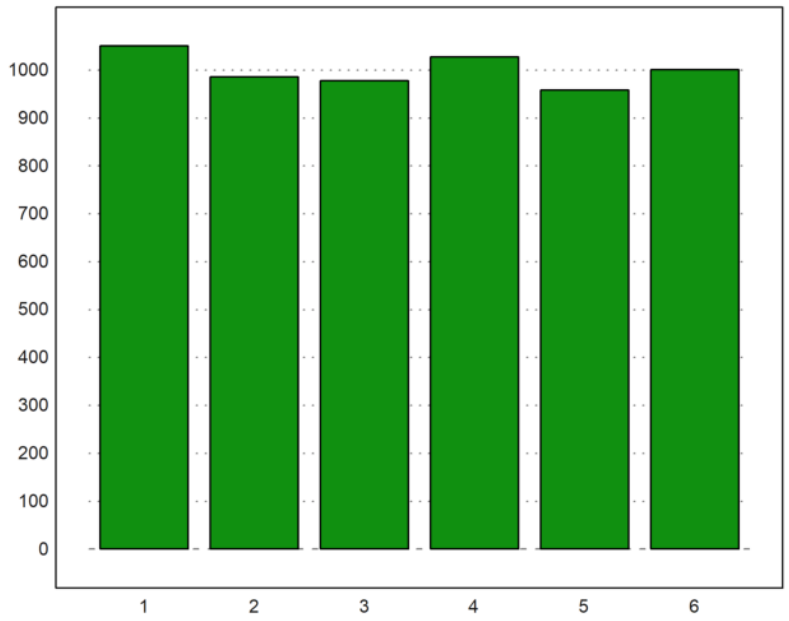
With the parameter even=true, this will use integer intervals.

```
>plot2d(intrandom(1,1000,10),distribution=10,even=true):
```

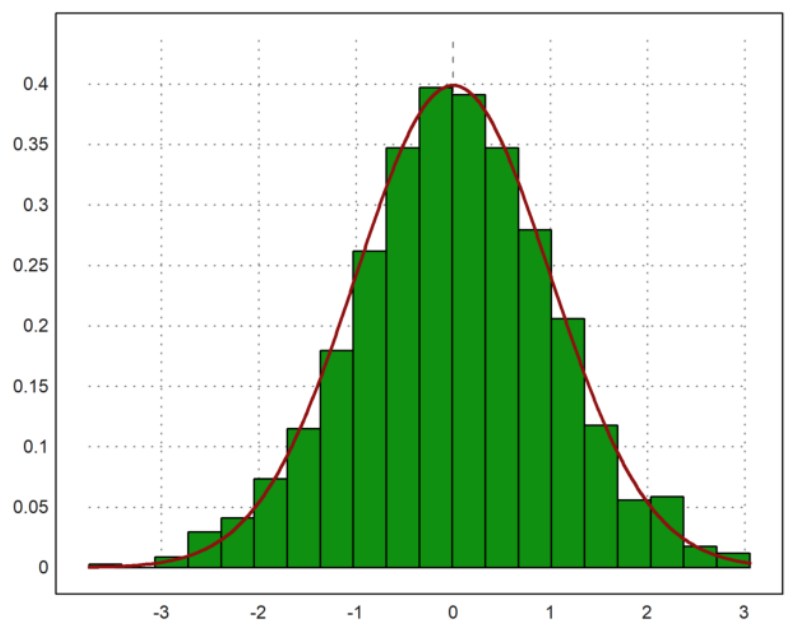


Note that there are many statistical plots, which might be useful. Have a look at the tutorial about statistics.

```
>columnsplot(getmultiplicities(1:6,intrandom(1,6000,6))):
```

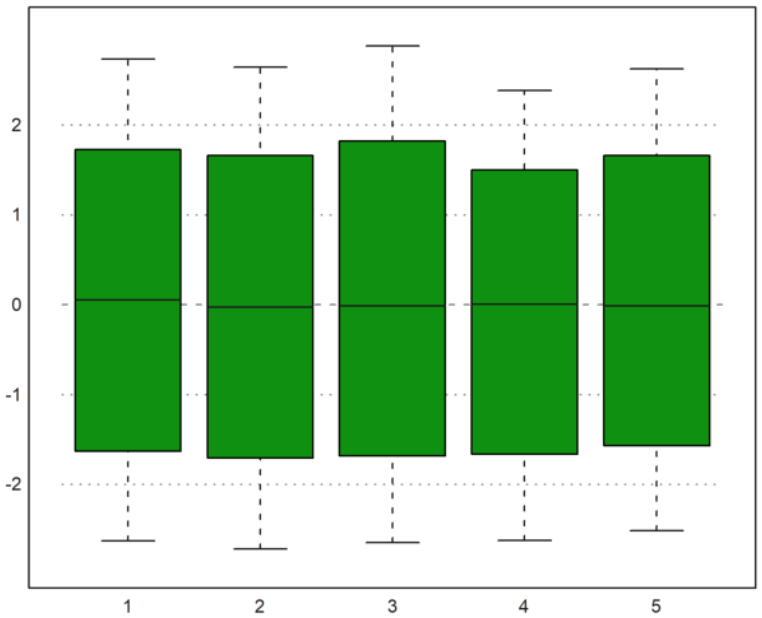



```
>plot2d(normal(1,1000),>distribution); ...
>  plot2d("qnormal(x)",color=red,thickness=2,>add):
```



There are also many special plots for statistics. A boxplot shows the quartiles of this distribution and lots of outliers. By definition, outliers in a boxplot are data which exceed 1.5 times the middle 50% range of the plot.

```
>M=normal(5,1000); boxplot(quartiles(M)):
```



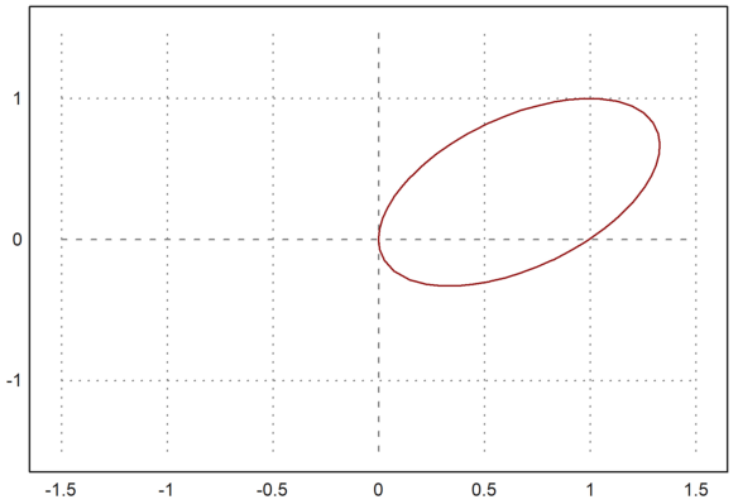
Implicit Functions

Implicit plots show level lines solving f(x,y)=level, where "level" can be a single value or a vector of values. If level="auto", there will be nc level lines, which will spread between the minimum and the maximum of the function evenly. Darker or lighter color can be added with >hue to indicate value of the function. For implicit functions, xv must be a function or an expression of the parameters x and y, or, alternatively, xv can be a matrix of values.
Euler can mark the level lines

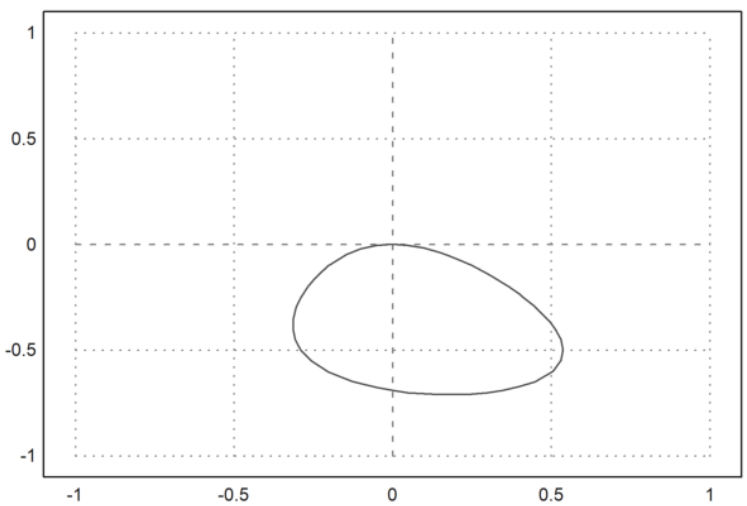
$$f(x,y) = c$$

of any function.
To draw the set f(x,y)=c for one or more constants c you can use plot2d() with its implicit plots in the plane. The parameter for c is level=c, where c can be vector of level lines. Additionally, a color scheme can be drawn in the background to indicate the value of the function for each point in the plot. The parameter "n" determines the fineness of the plot.

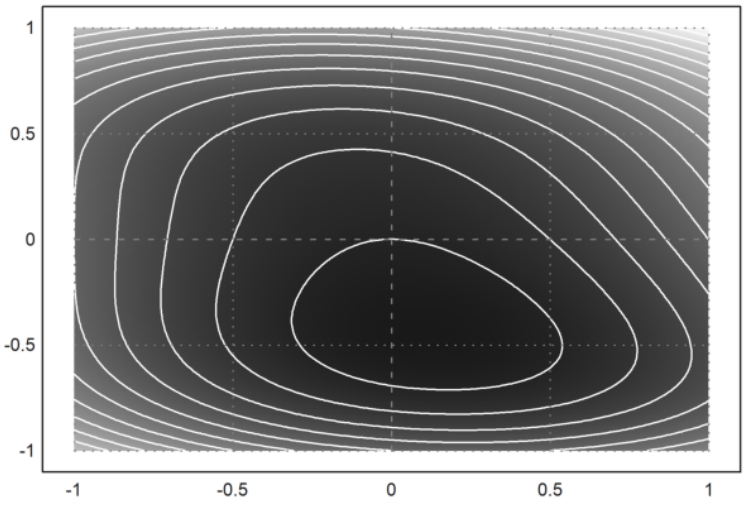
```
>aspect(1.5);
>plot2d("x^2+y^2-x*y-x",r=1.5,level=0,contourcolor=red):
```



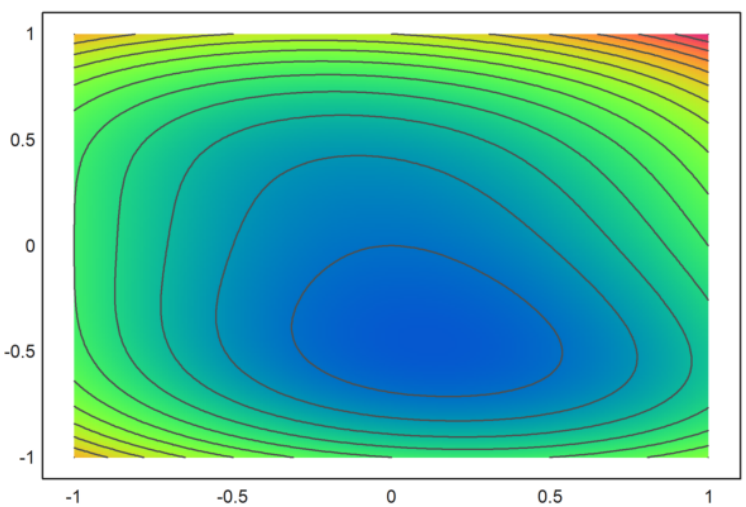

```
>expr := "2*x^2+x*y+3*y^4+y"; // define an expression f(x,y)
>plot2d(expr,level=0): // Solutions of f(x,y)=0
```




```
>plot2d(expr,level=0:0.5:20,>hue,contourcolor=white,n=200): // nice
```

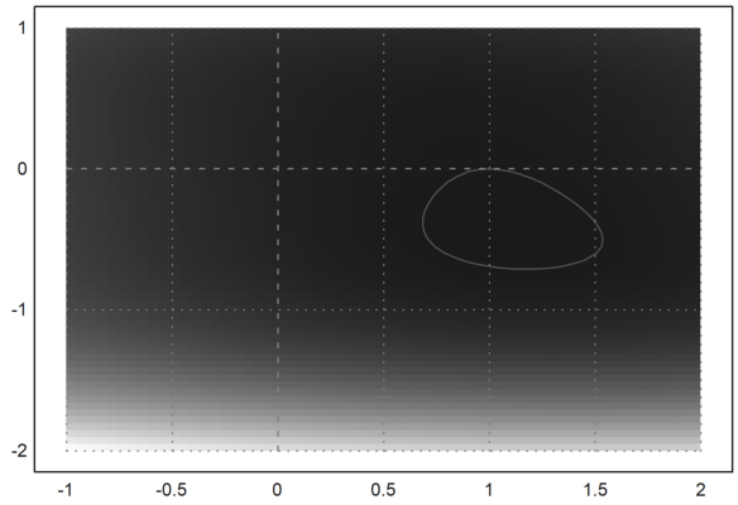



```
>plot2d(expr,level=0:0.5:20,>hue,>spectral,n=200,grid=4): // nicer
```

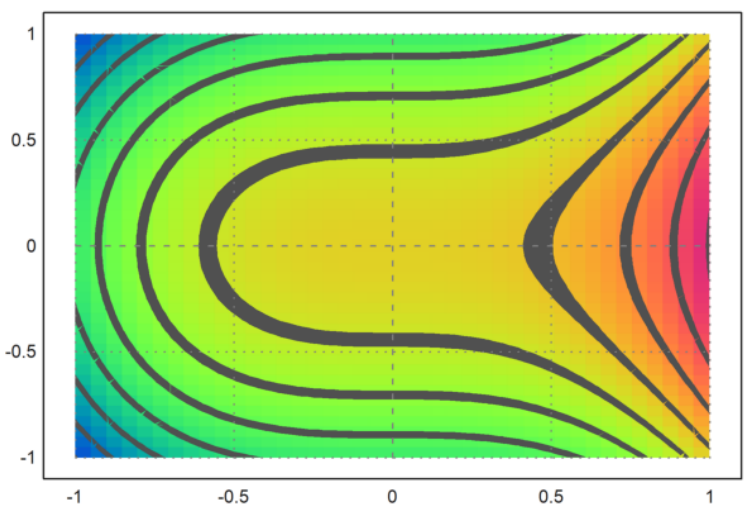


This works for data plots too. But you will have to specify the ranges for the axis labels.

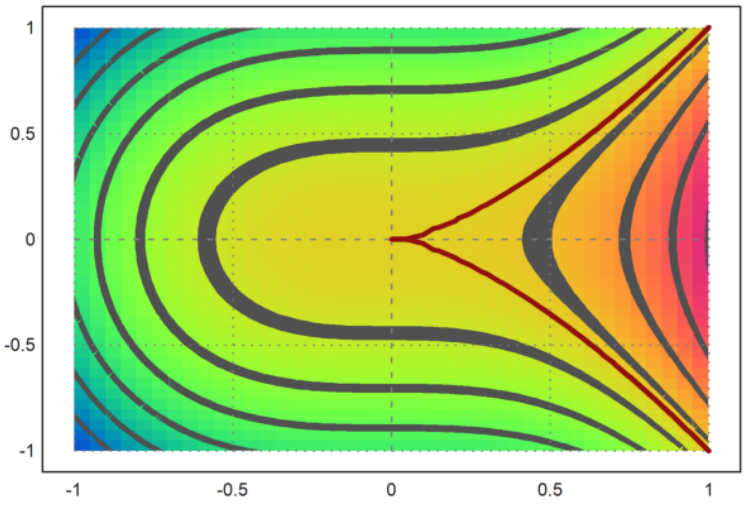
```
>x=-2:0.05:1; y=x'; z=expr(x,y);
>plot2d(z,level=0,a=-1,b=2,c=-2,d=1,>hue):
```



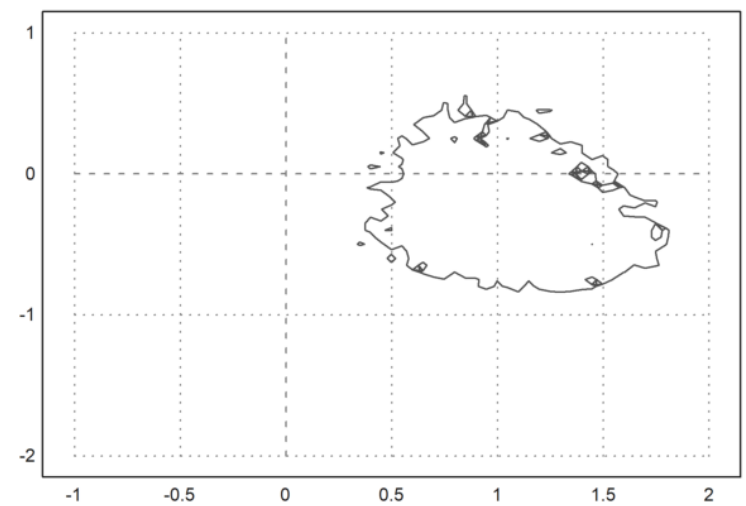

```
>plot2d("x^3-y^2",>contour,>hue,>spectral):
```



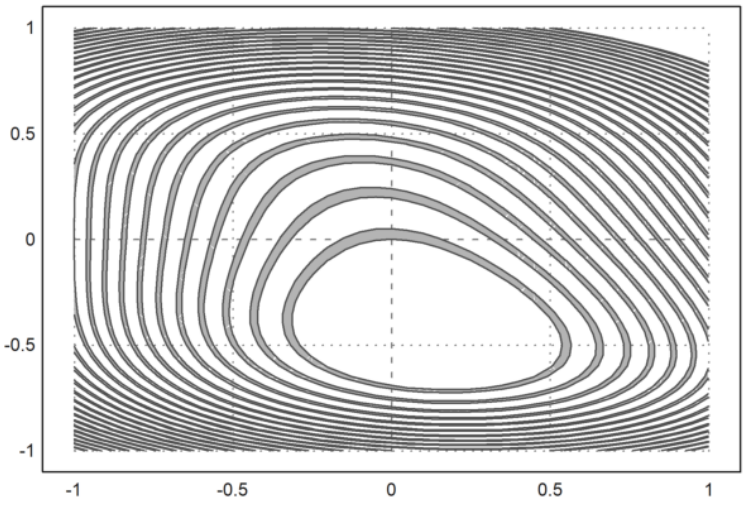

```
>plot2d("x^3-y^2",level=0,contourwidth=3,>add,contourcolor=red):
```



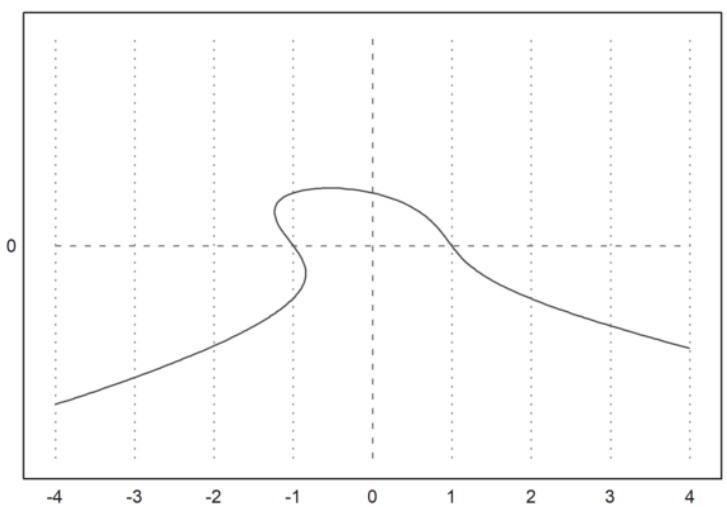

```
>z=z+normal(size(z))*0.2;
>plot2d(z,level=0.5,a=-1,b=2,c=-2,d=1):
```



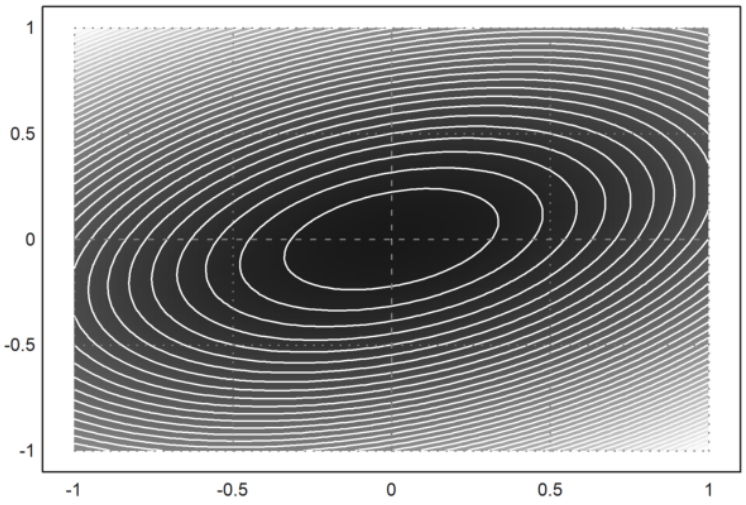

```
>plot2d(expr,level=[0:0.2:5;0.05:0.2:5.05],color=lightgray):
```




```
>plot2d("x^2+y^3+x*y",level=1,r=4,n=100):
```




```
>plot2d("x^2+2*y^2-x*y",level=0:0.1:10,n=100,contourcolor=white,>hue):
```

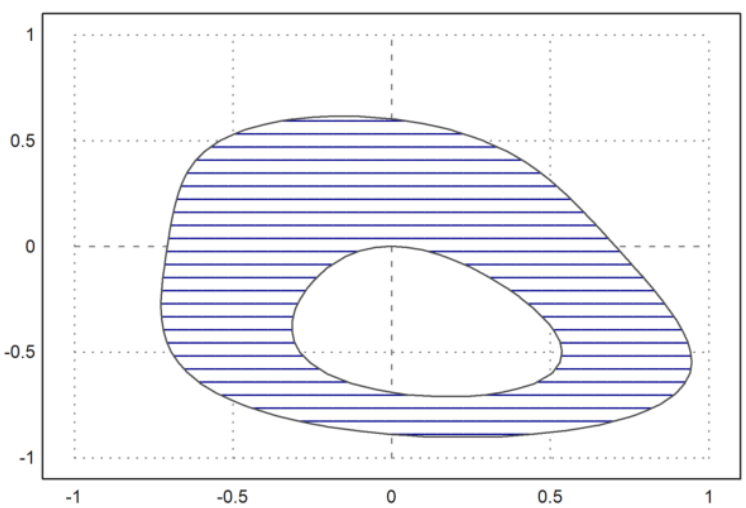


It is also possible to fill the set

$$a \le f(x,y) \le b$$

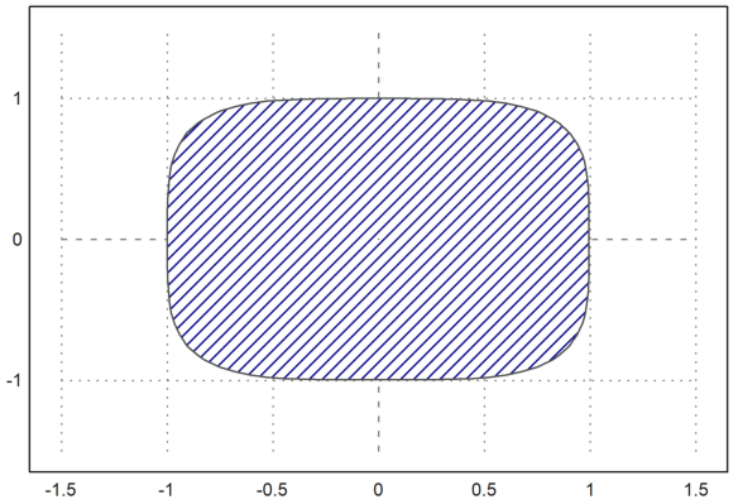
with a level range.
It is possible to fill regions of values for a specific function. For this, level must be a 2xn matrix. The first row are the lower bounds and the second row contains the upper bounds.

```
>plot2d(expr,level=[0;1],style="-",color=blue): // 0 <= f(x,y) <= 1
```

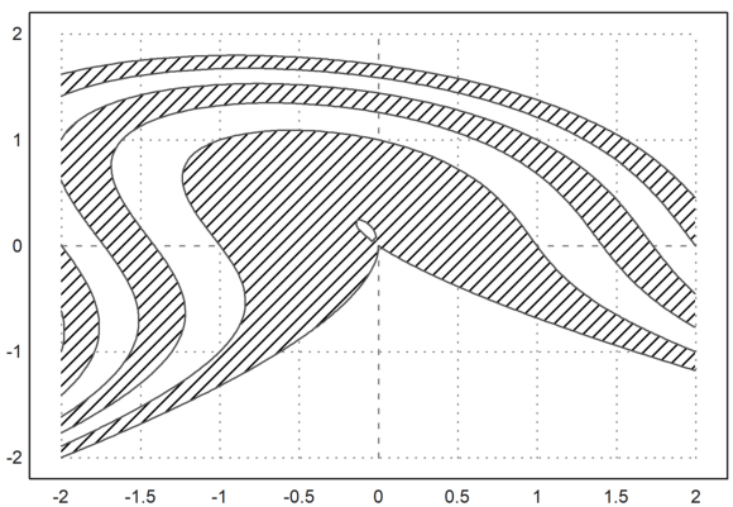


Implicit plots can also show ranges of levels. Then level must be a 2xn matrix of level intervals, where the first row contains the start and the second row the end of each interval. Alternatively, a simple row vector can be used for level, and a parameter dl extends the level values to intervals.

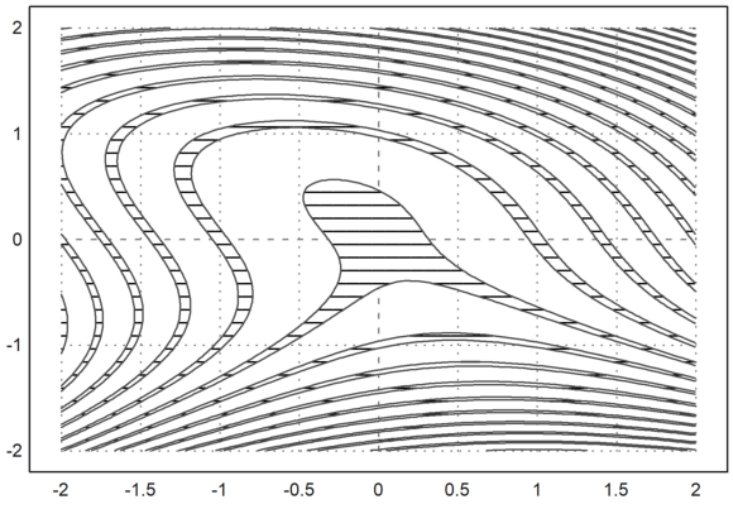
```
>plot2d("x^4+y^4",r=1.5,level=[0;1],color=blue,style="/"):
```



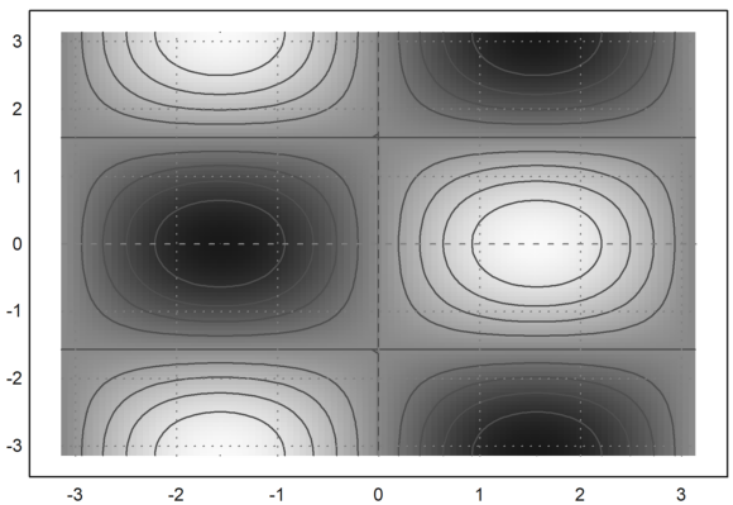

```
>plot2d("x^2+y^3+x*y",level=[0,2,4;1,3,5],style="/",r=2,n=100):
```




```
>plot2d("x^2+y^3+x*y",level=-10:20,r=2,style="-",dl=0.1,n=100):
```




```
>plot2d("sin(x)*cos(y)",r=pi,>hue,>levels,n=100):
```

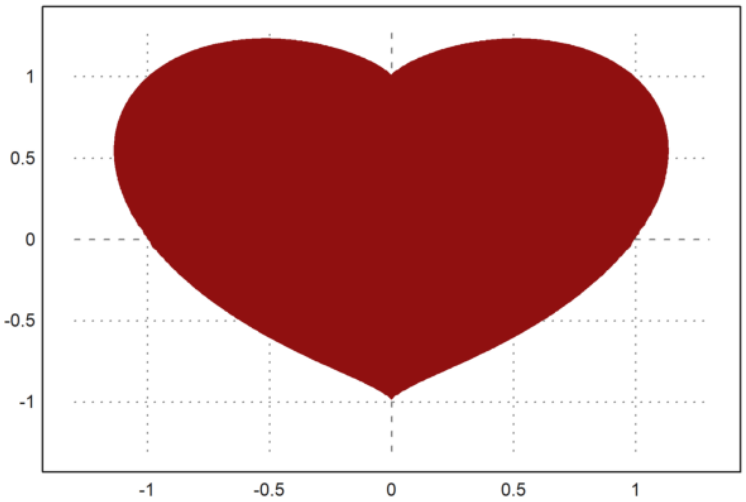


It is also possible to mark a region

$$a \le f(x,y) \le b.$$

This is done by adding a level with two rows.

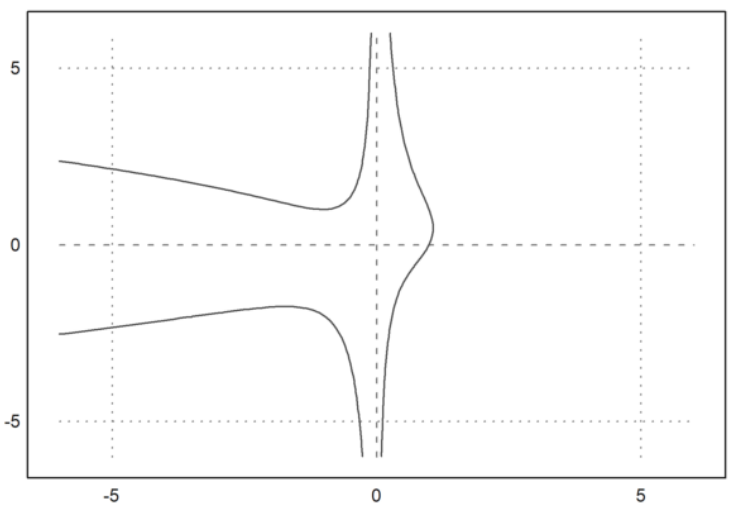
```
>plot2d("(x^2+y^2-1)^3-x^2*y^3",r=1.3, ...
>  style="#",color=red,<outline, ...
>  level=[-2;0],n=100):
```



It is possible to specify a specific level. E.g., we can plot the solution of an equation like

$$x^3 - xy + x^2y^2 = 6$$

```
>plot2d("x^3-x*y+x^2*y^2",r=6,level=1,n=100):
```



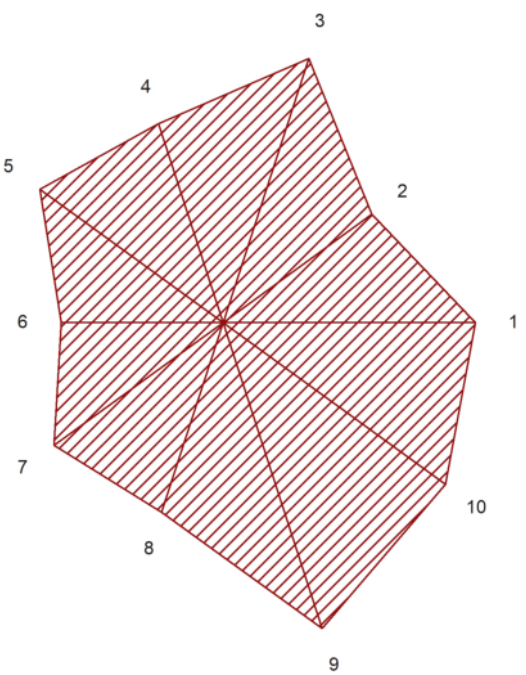

```
>function starplot1 (v, style="/", color=green, lab=none) ...
```

```
    if !holding() then clg; endif;
    w=window(); window(0,0,1024,1024);
    h=holding(1);
    r=max(abs(v))*1.2;
    setplot(-r,r,-r,r);
    n=cols(v); t=linspace(0,2pi,n);
    v=v|v[1]; c=v*cos(t); s=v*sin(t);
    cl=barcolor(color); st=barstyle(style);
    loop 1 to n
```

```
    polygon([0,c[#],c[#+1]],[0,s[#],s[#+1]],1);
    if lab!=none then
      rlab=v[#]+r*0.1;
      {col,row}=toscreen(cos(t[#])*rlab,sin(t[#])*rlab);
      ctext(""+lab[#],col,row-textheight()/2);
    endif;
  end;
  barcolor(cl); barstyle(st);
  holding(h);
  window(w);
endfunction
```

There is no grid or axis ticks here. Moreover, we use the full window for the plot.
We call reset before we test this plot to restore the graphics defaults. This is not necessary, if you are sure that your plot works.

```
>reset; starplot1(normal(1,10)+5,color=red,lab=1:10):
```



Sometimes, you may want to plot something that plot2d cannot do, but almost.
In the following function, we do a logarithmic impulse plot. plot2d can do logarithmic plots, but not for impulse bars.

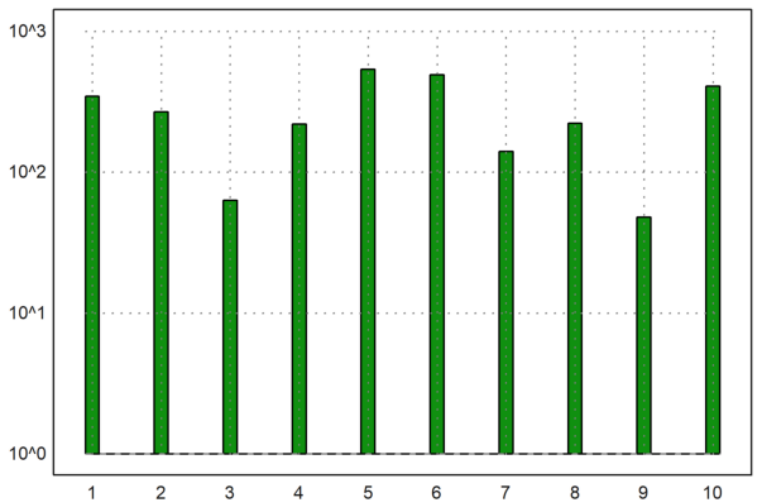
```
>function logimpulseplot1 (x,y) ...
```

```
  {x0,y0}=makeimpulse(x,log(y)/log(10));
  plot2d(x0,y0,>bar,grid=0);
  h=holding(1);
```

```
    frame();
    xgrid(ticks(x));
    p=plot();
    for i=-10 to 10;
      if i<=p[4] and i>=p[3] then
         ygrid(i,yt="10^"+i);
      endif;
    end;
    holding(h);
  endfunction
```

Let us test it with exponentially distributed values.

```
>aspect(1.5); x=1:10; y=-log(random(size(x)))*200; ...
>logimpulseplot1(x,y):
```



Let us animate a 2D curve using direct plots. The plot(x,y) command simply plots a curve into the plot window. setplot(a,b,c,d) sets this window.
The wait(0) function forces the plot to appear on the graphics windows. Otherwise, the redraw takes place in sparse time intervals.

```
>function animliss (n,m) ...
```

```
  t=linspace(0,2pi,500);
  f=0;
  c=framecolor(0);
  l=linewidth(2);
  setplot(-1,1,-1,1);
  repeat
    clg;
    plot(sin(n*t),cos(m*t+f));
    wait(0);
    if testkey() then break; endif;
    f=f+0.02;
  end;
  framecolor(c);
```

```
  linewidth(l);
  endfunction
```

Press any key to stop this animation.

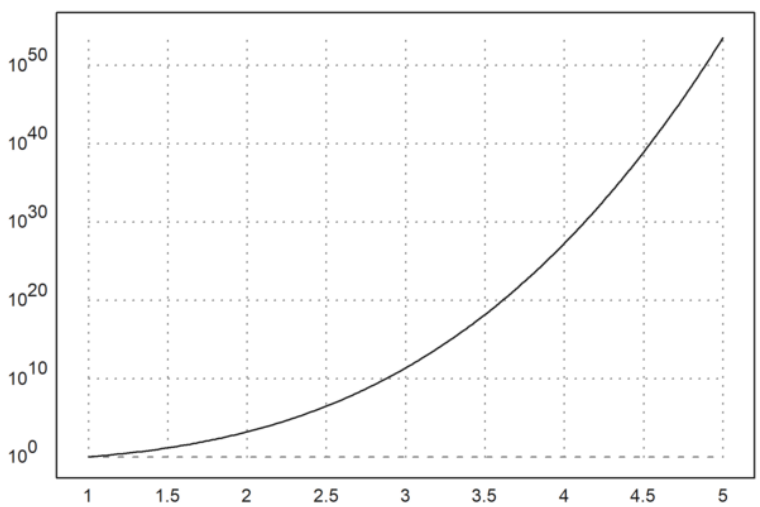
```
>animliss(2,3); // lihat hasilnya, jika sudah puas, tekan ENTER
```

Logarithmic Plots

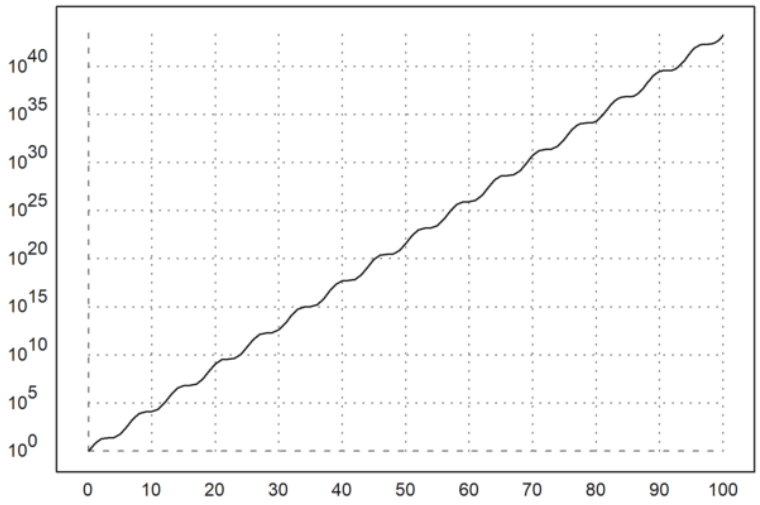
EMT uses the "logplot" parameter for logarithmic scales.
Logarithmic plots can be plotted either using a logarithmic scale in y with logplot=1, or using logarithmic scales in x and y with logplot=2, or in x with logplot=3.

- logplot=1: y-logarithmic
- logplot=2: x-y-logarithmic
- logplot=3: x-logarithmic

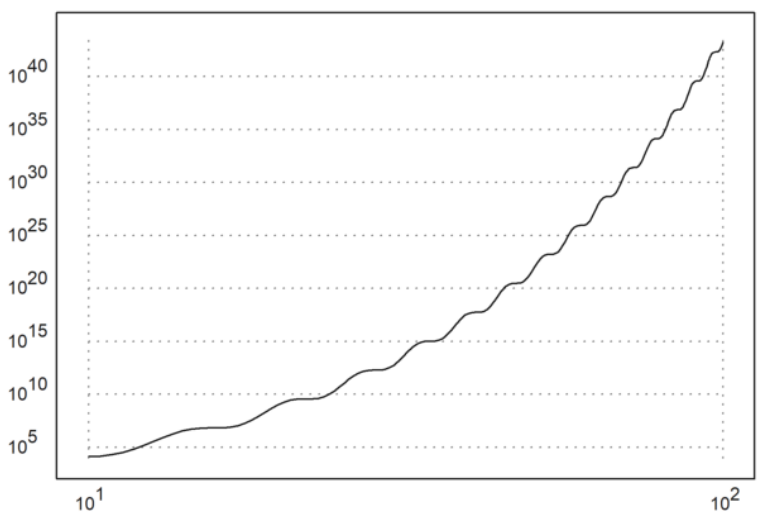
```
>plot2d("exp(x^3-x)*x^2",1,5,logplot=1):
```



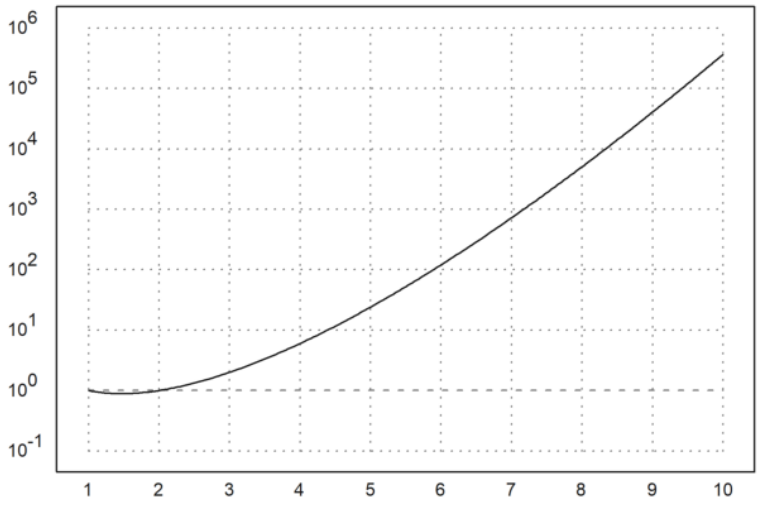

```
>plot2d("exp(x+sin(x))",0,100,logplot=1):
```



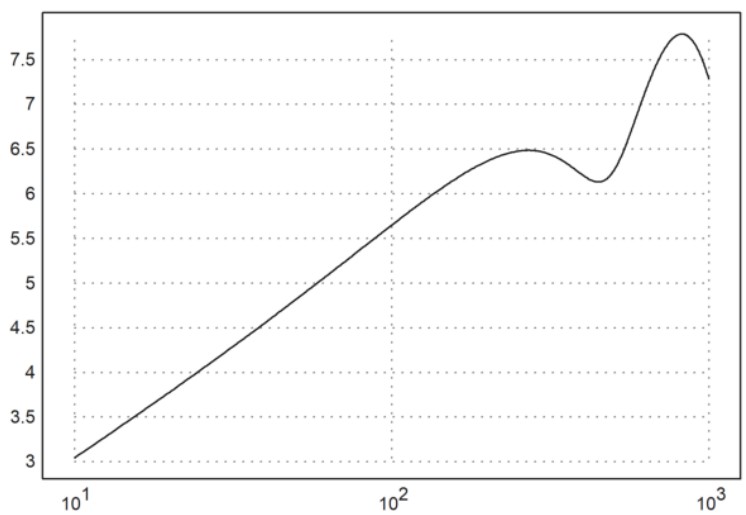

```
>plot2d("exp(x+sin(x))",10,100,logplot=2):
```




```
>plot2d("gamma(x)",1,10,logplot=1):
```

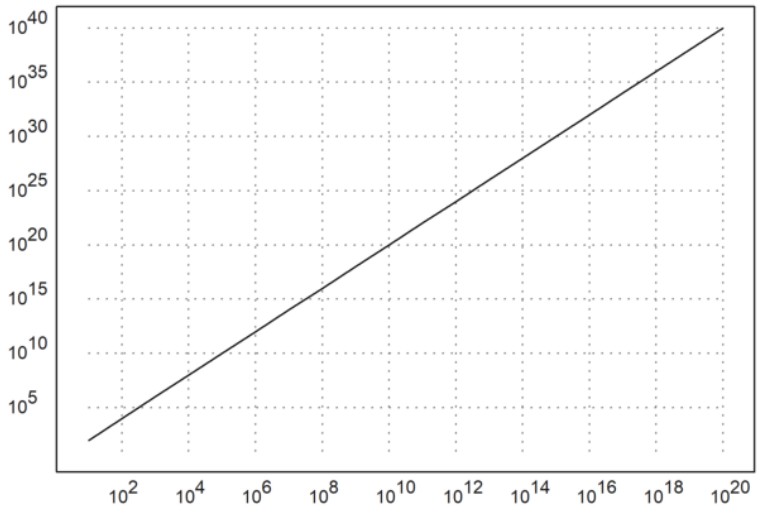



```
>plot2d("log(x*(2+sin(x/100)))",10,1000,logplot=3):
```



This does also work with data plots.

```
>x=10^(1:20); y=x^2-x;
>plot2d(x,y,logplot=2):
```



Rujukan Lengkap Fungsi plot2d()

```
function plot2d (xv, yv, btest, a, b, c, d, xmin, xmax, r, n,  ..
logplot, grid, frame, framecolor, square, color, thickness, style, ..
auto, add, user, delta, points, addpoints, pointstyle, bar, histogram,  ..
distribution, even, steps, own, adaptive, hue, level, contour,  ..
nc, filled, fillcolor, outline, title, xl, yl, maps, contourcolor, ..
contourwidth, ticks, margin, clipping, cx, cy, insimg, spectral,  ..
cgrid, vertical, smaller, dl, niveau, levels)
```

Multipurpose plot function for plots in the plane (2D plots). This function can do plots of functions of one variables, data plots, curves in the plane, bar plots, grids of complex numbers, and implicit plots of functions of two variables.
Parameters
x,y : equations, functions or data vectors
a,b,c,d : Plot area (default a=-2,b=2)
r : if r is set, then a=cx-r, b=cx+r, c=cy-r, d=cy+r

```
r can be a vector [rx,ry] or a vector [rx1,rx2,ry1,ry2].
```

xmin,xmax : range of the parameter for curves
auto : Determine y-range automatically (default)
square : if true, try to keep square x-y-ranges
n : number of intervals (default is adaptive)
grid : 0 = no grid and labels,

```
1 = axis only,
2 = normal grid (see below for the number of grid lines)
3 = inside axis
4 = no grid
5 = full grid including margin
6 = ticks at the frame
7 = axis only
8 = axis only, sub-ticks
```

frame : 0 = no frame
framecolor: color of the frame and the grid
margin : number between 0 and 0.4 for the margin around the plot
color : Color of curves. If this is a vector of colors,

```
it will be used for each row of a matrix of plots. In the case of
point plots, it should be a column vector. If a row vector or a
full matrix of colors is used for point plots, it will be used for
each data point.
```

thickness : line thickness for curves

```
This value can be smaller than 1 for very thin lines.
```

style : Plot style for lines, markers, and fills.

```
For points use
"[]", "<>", ".", "..", "...",
"*", "+", "|", "-", "o"
"[]#", "<>#", "o#" (filled shapes)
"[]w", "<>w", "ow" (non-transparent)
For lines use
"-", "--", "-.", ".", ".-.", "-.-", "->"
For filled polygons or bar plots use
"#", "#O", "O", "/", "\", "\/",
"+", "|", "-", "t"
```

points : plot single points instead of line segments
addpoints : if true, plots line segments and points
add : add the plot to the existing plot
user : enable user interaction for functions
delta : step size for user interaction
bar : bar plot (x are the interval bounds, y the interval values)
histogram : plots the frequencies of x in n subintervals
distribution=n : plots the distribution of x with n subintervals
even : use inter values for automatic histograms.
steps : plots the function as a step function (steps=1,2)
adaptive : use adaptive plots (n is the minimal number of steps)
level : plot level lines of an implicit function of two variables
outline : draws boundary of level ranges.
If the level value is a 2xn matrix, ranges of levels will be drawn
in the color using the given fill style. If outline is true, it
will be drawn in the contour color. Using this feature, regions of
f(x,y) between limits can be marked.
hue : add hue color to the level plot to indicate the function

```
value
```

contour : Use level plot with automatic levels
nc : number of automatic level lines
title : plot title (default "")
xl, yl : labels for the x- and y-axis
smaller : if >0, there will be more space to the left for labels.
vertical :

```
  Turns vertical labels on or off. This changes the global variable
  verticallabels locally for one plot. The value 1 sets only vertical
  text, the value 2 uses vertical numerical labels on the y axis.
```

filled : fill the plot of a curve
fillcolor : fill color for bar and filled curves
outline : boundary for filled polygons
logplot : set logarithmic plots

```
            1 = logplot in y,
            2 = logplot in xy,
            3 = logplot in x
```

own :

```
  A string, which points to an own plot routine. With >user, you get
  the same user interaction as in plot2d. The range will be set
  before each call to your function.
```

maps : map expressions (0 is faster), functions are always mapped.
contourcolor : color of contour lines
contourwidth : width of contour lines
clipping : toggles the clipping (default is true)
title :

```
  This can be used to describe the plot. The title will appear above
  the plot. Moreover, a label for the x and y axis can be added with
  xl="string" or yl="string". Other labels can be added with the
  functions label() or labelbox(). The title can be a unicode
  string or an image of a Latex formula.
```

cgrid :

```
  Determines the number of grid lines for plots of complex grids.
  Should be a divisor of the the matrix size minus 1 (number of
  subintervals). cgrid can be a vector [cx,cy].
```

Overview
The function can plot
- expressions, call collections or functions of one variable,
- parametric curves,
- x data against y data,
- implicit functions,
- bar plots,
- complex grids,
- polygons.

If a function or expression for xv is given, plot2d() will compute values in the given range using the function or expression. The expression must be an expression in the variable x. The range must be defined in the parameters a and b unless the default range should be used. The y-range will be computed automatically, unless c and d are specified, or a radius r, which yields the range r,r

for x and y. For plots of functions, plot2d will use an adaptive evaluation of the function by default. To speed up the plot for complicated functions, switch this off with <adaptive, and optionally decrease the number of intervals n. Moreover, plot2d() will by default use mapping. I.e., it will compute the plot element for element. If your expression or your functions can handle a vector x, you can switch that off with <maps for faster evaluation. Note that adaptive plots are always computed element for element. If functions or expressions for both xv and for yv are specified, plot2d() will compute a curve with the xv values as x-coordinates and the yv values as y-coordinates. In this case, a range should be defined for the parameter using xmin, xmax. Expressions contained in strings must always be expressions in the parameter variable x.