plot *(basic plotting)*

The `plot` command is probably the command you will use most often in Maple. The purpose of this command, of course, is to produce (two-dimensional) plots.

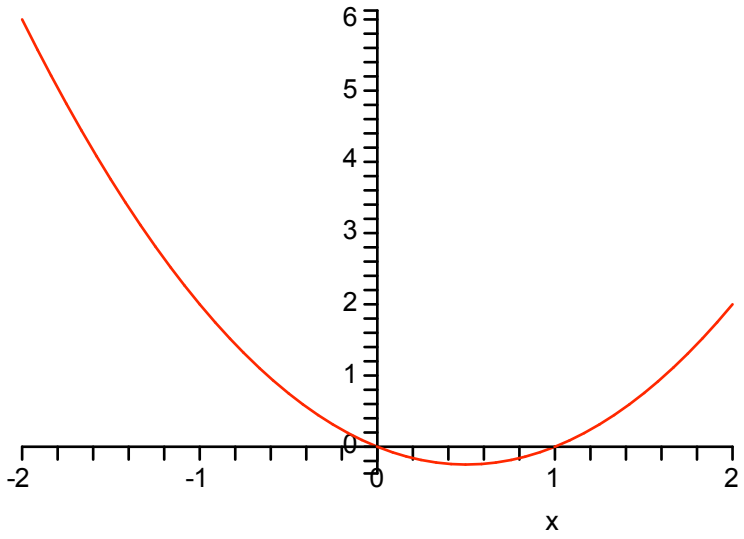
The syntax of the `plot` command in general follows the basic Maple

```
plot(what,how);
```

pattern, but both the "what" and the "how" can get pretty complicated.

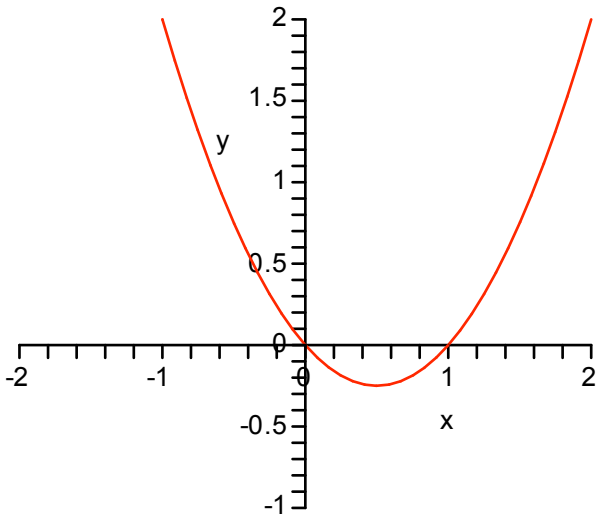
In the most basic form of the `plot` statement, "what" is an expression to be plotted and "how" indicates the domain on the horizontal axis over which the plot is to be displayed:

```
> restart:
> plot(x^2-x,x=-2..2);
```



Notice here that Maple automatically chose a scale on the vertical axis. The scale it chooses is such that the plot over the entire specified domain is visible (i.e., the graph does not "run off" the top of the plot). It is possible to restrict the range on the vertical axis as well, as follows:

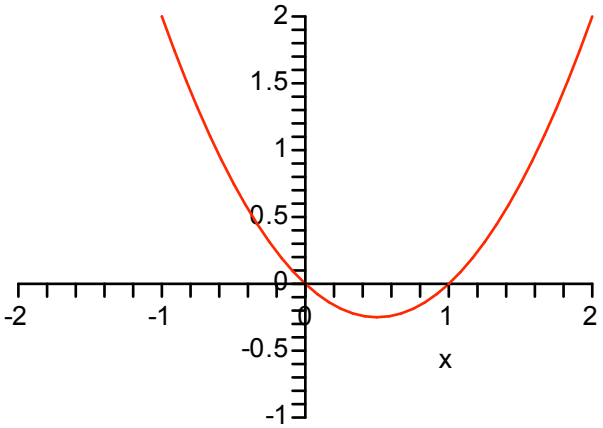
```
> plot(x^2-x,x=-2..2,y=-1..2);
```



There is another important difference between the two plots above besides the change of scale on the vertical axis -- namely, the vertical axis on the second plot has a label. Maple takes the axis labels from the left side of the domain and range specifications.

It is possible not to give the label for the vertical axis, if you don't want it printed. If the "what" to plot is an expression, the variable in the domain specification must be specified, however. This is illustrated by the following:

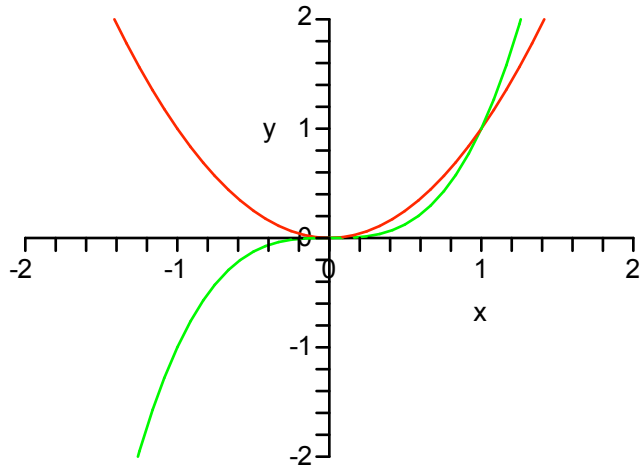
```
> plot(x^2-x,x=-2..2,-1..2);
```




```
>
```

Plotting more than one curve on the same axes: It is possible to do this. But Maple looks for the first (un-parenthesized) comma in the `plot` syntax to dilineate the "what" from the "how". Thus, your list of things to plot must be enclosed within braces `{ }`. For example:

```
> plot({x^2,x^3},x=-2..2,y=-2..2);
```



Common errors in basic plotting: The most common syntax error to make while plotting is to forget the braces when you are plotting more than one curve on the same axes. Other than syntax errors, the most common mistakes to make when plotting involve incorrect specification of variables. There are two kinds of errors:

1. Using a domain value that already has a specific value. It is important to make sure that the variable that is supposed to vary during the plot isn't already declared to be a constant (perhaps in the distant past during the Maple session). Making this error results in an error message, because Maple thinks you are trying to assign a new value to a constant ("invalid arguments"):

```
> x:=3:

> plot(x^2,x=-2..2);
Error, (in plot) invalid arguments
```

(Now we reset the value of x so that we don't run into the problem we have just illustrated.)

```
> x:='x';
```

$$x := x$$

2. Not specifying a domain variable, or specifying the wrong domain variable. If your expression involves `t`, then you must let `t=-2..2`, not `x=-2..2` (or whatever the range is). This kind of mistake results in the dreaded "invalid arguments" or "empty plot" messages:

```
> plot(t^3,x=-2..2);
Warning, unable to evaluate the function to numeric values in the region;
see the plotting command's help page to ensure the calling sequence is
correct

Error, empty plot
> plot(t^3,-2..2);
Warning, unable to evaluate the function to numeric values in the region;
see the plotting command's help page to ensure the calling sequence is
correct

Error, empty plot
```

FANCIER PLOTTING:

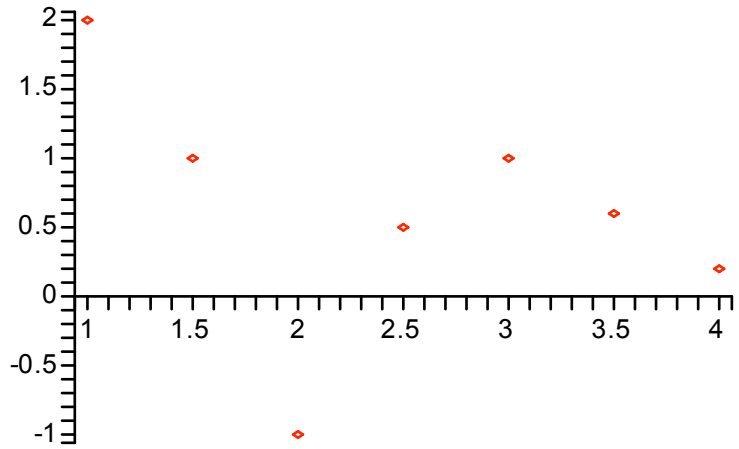
The `plot` command is incredibly powerful and versatile. All of the ins and outs of plot options take a fair amount of getting used to. We will cover a few of them here.

Plotting points: It is possible to have Maple plot points. This is often useful when comparing empirical data with a mathematical model. There are two ways to do this, depending on how the points are generated. If you have a list of specific points to plot, you can assign them to a name as follows (you may replace the name "ptlst" with any of your own choosing -- except those in the list of "reserved words"):

```
> ptlst:=[1,2],[1.5,1],[2,-1],[2.5,0.5],[3,1],[3.5,0.6],[4,0.2
  ]:
>
```

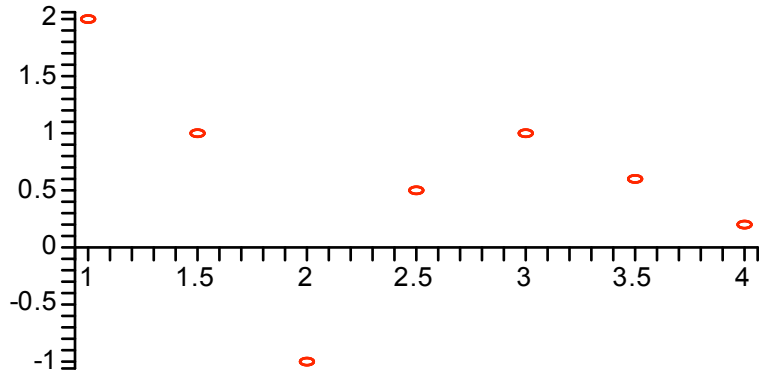
In this statement, the variable `ptlst` is a list of points. Each point is an ordered pair of numbers enclosed in square brackets (this is different from the usual convention in mathematics). To plot the list, we must enclose the entire list in square brackets, as follows:

```
> plot([ptlst],style=POINT);
```



Maple can use other symbols for the points, including circles and boxes. The optional phrase: `symbol=circle` or `symbol=box` is used for this purpose, as follows:

```
> plot([ptlst],style=POINT,symbol=circle);
```

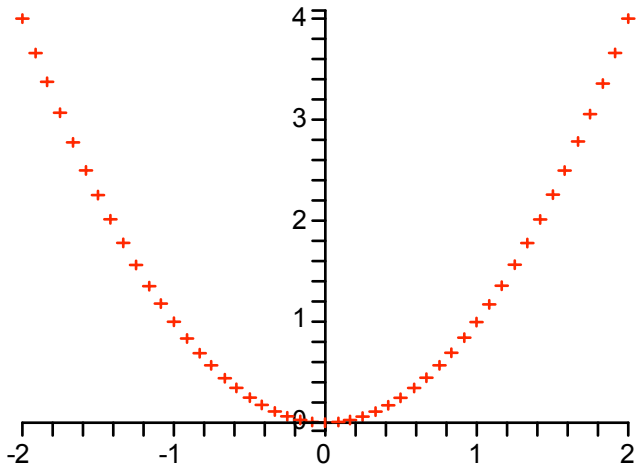



```
>
```

If you replace `style=POINT` with `style=LINE`, the dots will be connected by straight lines (in the order the points were given -- this can make for interesting-looking plots if the points are mixed up).

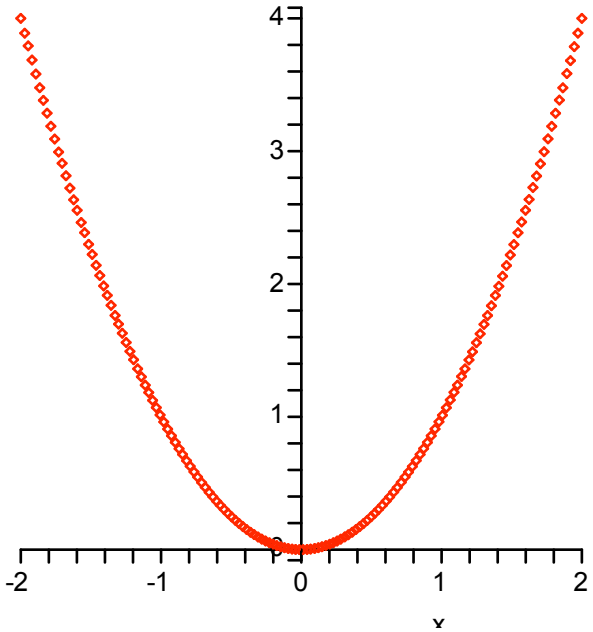
If the points come from evaluating an expression at several values of x, you can use `plot` in its usual form, but specify `style=POINT` (and a symbol option if you like):

```
> plot(x^2,x=-2..2,style=point,symbol=cross);
```



The minimum number of points plotted this way is about 50. You can insist that more points be plotted using the "numpoints" (number of points) option as follows (we do not show the plot):

```
> plot(x^2,x=-2..2,style=POINT,numpoints=150);
```




```
>
```

COMBINING PLOTS, Labelling Plots ...

Maple's fanciest specialty plotting functions are contained in a separate library called "`plots`". For basic plotting, there are two commands from the `plots` library which are especially useful: `textplot` and `display`. To load these two commands into the computer memory, use the statement:

```
> with(plots,textplot,display);
```

[*textplot*, *display*]

The `display` command is useful for combining different kinds of plots into one picture. The various kinds of plots that can be combined are standard plots of expressions, plots of points, plots of text (what `textplot` is for, useful for labelling things), and animations.

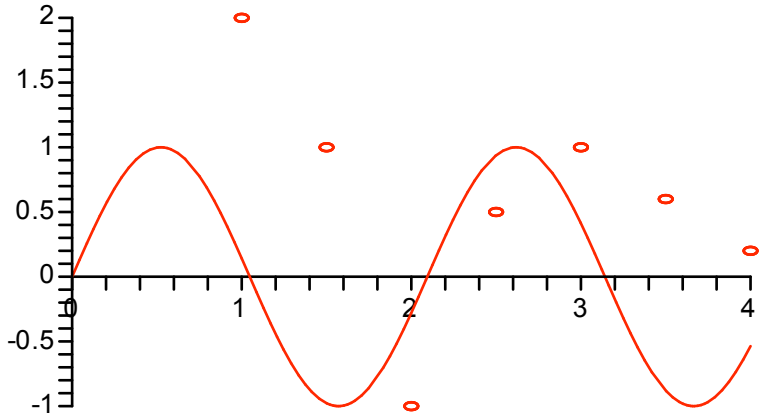
For example, suppose we wish to combine the point plot of the variable "`ptlst`" we defined above, and a plot of the function `sin(3*x)`. To do this, we define two separate plots, assign them to variables, and then display them together as follows:

```
> plot1:=plot([ptlst],style=POINT,symbol=circle):

> plot2:=plot(sin(3*x),x=0..4):

>
```

When defining and assigning plots, it is very advisable to use a colon rather than a semicolon. The thing that gets assigned to the variable (`plot1` and `plot2` in these examples) is Maple's list of internal instructions for producing the plot -- a long, complicated sequence of computer-speak that is best left undisplayed.

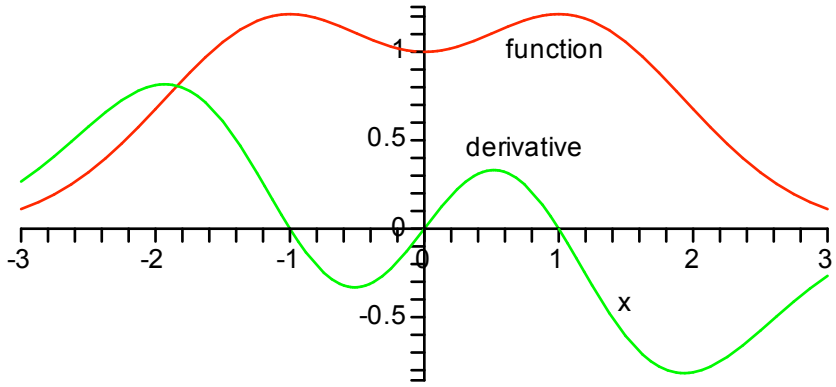
The display command uses the standard `display(what,how)` Maple syntax. In this case "what" is a set of "plot structures", and "how" is often an expression of the form `view=[a..b,c..d]`, which specifies the horizontal and vertical ranges to be displayed:

```
> display({plot1,plot2},view=[0..4,-1..2]);
```



Another reason to use `display` is to attach labels to objects in your plots. You do this by putting the labels in a separate plot called a `textplot`. The `textplot` command takes as its argument a single or a set of "text objects", all of which look like `[a,b,`words`]` -- it places the words inside the quotes (they are both left quotes, on the keyboard to the left of the numeral 1) on the plot so that they are centered at the point (a,b). To see this at work, we plot a function and its derivative, and label them on a graph:

```
> y:=(1+x^2)*exp(-x^2/2): d:=diff(y,x):

> F:=plot({y,d},x=-3..3):

> G:=textplot({[1,1,`function`],[0.75,0.45,`derivative`]}):

> display({F,G});
```



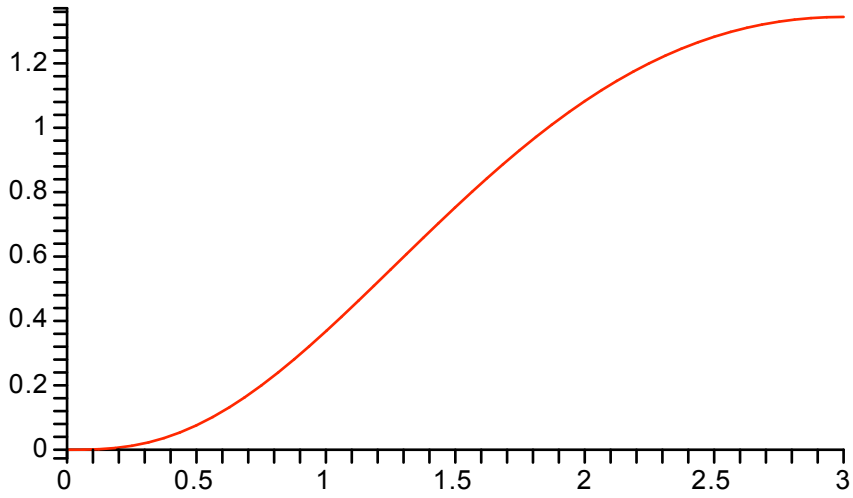
Optional special topic: A word about plotting functions (as opposed to expressions), and one situation in which it is a good idea: Sometimes, you will have the relationship you

want to plot in the form of a function, rather than an expression, for example:

```
> f:=x->x^3*exp(-x):
```

In such a situation, you can simply plot `f(x)`, which is an expression, using the information given above. Alternatively, you may use `plot` in the following form:

```
> plot(f,0..3);
```



Usually, there is no particular reason to favor one version of `plot` over the other. However, it is imperative not to confuse them. Neither statement

```
> plot(f(x),0..3);
```

```
Warning, unable to evaluate the function to numeric values in the region;
see the plotting command's help page to ensure the calling sequence is
correct


Error, empty plot
```

nor

```
> plot(f,x=0..3);
```

```
Error, (in plot) invalid plotting of procedures, perhaps you mean plot(f,
0 .. 3)
```

will work correctly.

The one situation in which function-plotting is de riguer is when you have defined a function that contains an "if-then" clause (a step-function, or piecewise-defined function). For example:

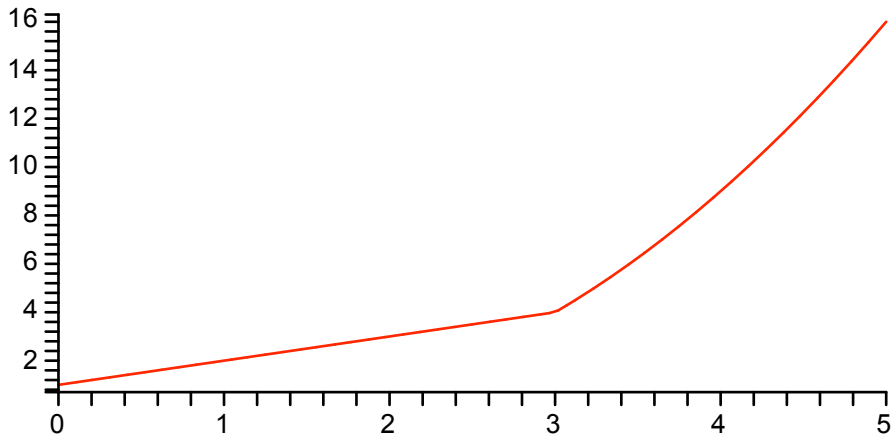
```
> f:=x-> if x<3 then x+1 else (x-1)^2 fi:
```

This function is equal to `x+1` if x is less than 3 and is equal to `(x-1)^2` otherwise. If we try to plot it ths usual way, we will get an error message:

```
> plot(f(x),x=0..5);
Error, (in f) cannot determine if this expression is true or false: x < 3
```

This is because Maple attempts to understand the function before it has a value for x. On the other hand, the following way will work:

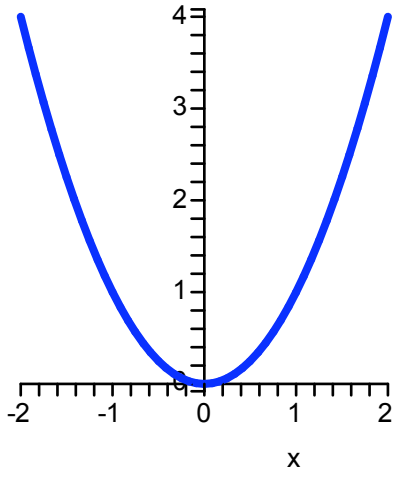
```
> plot(f,0..5);
```



(incidentally, from the plot we can see that f is probably continuous but not differentiable at x=3).

Plotting options: There are many options you can invoke when doing plots so that you can make the plot look like you want it. For example, you can control the color of a graph with the "color=" option (Maple knows many colors, for instance "color=red" or "color=green"), or make the curves plot thicker with the "thickness=" option (the plots above all use the default "thickness=1", but you can use bigger integers than 1 to get thicker plots). Another useful option is "scaling=constrained", which tells Maple to use the same scale on the x and y axes -- this makes circles look like circles rather than ellipses, and the slopes of lines are really what they appear to be. Here is a plot that uses all of these options:

```
> plot(x^2,x=-2..2,color=blue,thickness=3,scaling=constrained)
  ;
```


4
3
2
1
0
-2
-1
0
1
2
x

>