logplots.mws

Improved Logarithmic Plotting

Carl Devore <devore@math.udel.edu>

6 July 2001

In this worksheet, I present a module with procedures for logarithmic plotting in 2 or 3 dimensions. All plot structures are handled except for animations and ISOSURFACEs. There are procedures for producing 3d plots with logarithmic scaling on any axes as well as a procedure that will improve the axes of logarithmic plots produced by other programs.

If you use this module, please send me email, either to comment on it, make suggestions for improvement, or just to say "hi": devore@math.udel.edu. The author can be reached by phone at (302) 738-2606 or (302) 831-3176 or by paper mail at 655 E3 Lehigh Rd, Newark DE 19711, USA.

There are many features of the module that are not shown in the examples below. Most of these are commented in the code. Feel free to email me for any assistance.

Some features:

Procedure Logplot: Take the logarithms along any axis of any plot structure, except ISOSURFACEs. The grid or mesh spacing will NOT be even for 3d plots and will NOT be adaptive for 2d plots..

Procedure semilogplot3d: Take any plot3d command and use logarithmic scaling on either or both independent axes. The grid or mesh spacing WILL be even if the plot is not a parametric surface. Can handle parameterized domains.

View options:

If a view option is outside the range of the original plot, it is likely because the user wanted to leave a margin of a particular size. My processing of the view option leaves the same percentage of margin as the original non-log plot had.   In particular, it can be meaningful to have a negative lower viewpoint on a logged axis.   For example, suppose that an original, non-log axis had an actual range of 1-10 and a lower viewpoint of -1. The margin at the bottom of that graph is 1-(-1) = 2 which is 2/(10-1) = 22% of the actual range. We choose to maintain the 22%. The range on the log graph is 0-1, so the lower viewpoint should be -.22, which will appear as 10^(-.22) in the original units.

If a viewpoint cropped the original plot, then the log of that viewpoint is used.

Merging:

If the user left a margin, as described under "View options", perhaps it is because something is plotted in the margin. The user probably wants that object to remain fixed relative to the boundaries of the plot. In general we want to take a non-logarithmic plot containing several objects, take the logs of some of the objects, but have the other objects appear as fixed background. We can't just do nothing to the fixed objects: they need to be scaled to the new axes. Procedure "Merge" handles all this.

Tickmarks:

I have about 250 lines of code devoted to printing the tickmarks for the logarithmic axes. This is far more sophisticated than any other logarithmic plotting program that I know of. I have attempted to print reasonable tickmarks in all cases, whether the range is small or large, whether it spans many powers of 10 or none, whether the user requests few or many tickmarks, whether a view option crops the axis or expands it.

If the user has specified a tickmark as an equation, we simply take the log of the left side.

If the user has specified a tickmark as a number x, we make the equation log(x) = "x", where "x" is a minimal string representation of x.
If the user has specified a string s, and that string can be interpretted as a number x, we make the equation log(x) = s.
Note that the three types of tickmarks above can be mixed in the same list, unlike with regular plot commands.

If the user has specified a number of tickmarks rather than their values, or DEFAULT, we try to mimic what happens with the ordinary plot command, with some extra provisions for the logs: If the user specifies a number of tickmarks, we will print at least that many LABELLED tickmarks. If nothing is specified, the default is 6, just like the regular plot commands. Some unlabelled tickmarks may be put between the labelled ones.

Integer powers of 10, 10^i, where abs(i) > 2, are printed as "ei". For example 1000 is printed as "e3". This removes a lot of clutter from the plot. Leading zeroes, plus signs and other space wasters from Maple's scientific notation are eliminated. For every number printed on the tickmarks, I look at it as a floating point, in scientific notation, and as an integer (if applicable), and pick the shortest representation for printing.

If the desired number of labelled tickmarks can be achieved as integer powers of 10, then only integer powers of 10 are used, with each labelled EXPONENT being 1, 2, or 5 times a power of 10, and the interval between unlabelled exponents max(1, step/5), where step is the interval between labelled exponents.

If we need to subdivide between the powers of 10, then the first subdivision represents the numbers as multipliers unless the numbers can be represented as themselves in less than 3 characters. For example, 10 tickmarks btw 100 and 1000 would be e2, x2, x3, ..., x9, e3. But subdivsions less than the lowest printed integer power of 10 are always represented as themselves. Each 1/9 of a power of 10 is ticked off, some labelled and some unlabelled. The ones that are labelled are determined by the number desired to be labelled and the table `plot/logaxes_table`, which comes with Maple. If 5 is unlabelled, a long tick mark is used.

If there's still not enough labelled tickmarks, there are two possibilities:
A. There are currently two or more tickmarks.
B. There's less than two tickmarks, in which case the range must be small.
In case A, we subdivide each current tick interval. In case B, the tick values we choose are pretty much the same as would be chosen by an oridinary plot command.

If a view option is specified, then the range of tickmarks is based on the view.

RedoOpts:

These tickmarks are so much better than other logarithmic plotters that you'll probably want to take those other plots and "correct" them to use my tickmarks. Procedure RedoOpts does this.

Axes labels:

If an axis label is a "simple" identifier, that is, it is composed strictly of alphanumeric characters and underscores, then ` (log)` is prepended to the label.

```
>    restart;
```

```
>    plotsetup(inline);
```

```
>    read "C:\\Program Files\\Maple 8\\Users\\carl devore\\logplots.mpl";
```

$logplots :=$

```
module() export semilogplot3d, Logplot, RedoOpts, Merge, LogView, LogTicks, SmallRangeTicks, strip, minstring, LogLabel, LogOpt, LogOpts, Ranges; ... end module
```

```
> with(logplots);
```

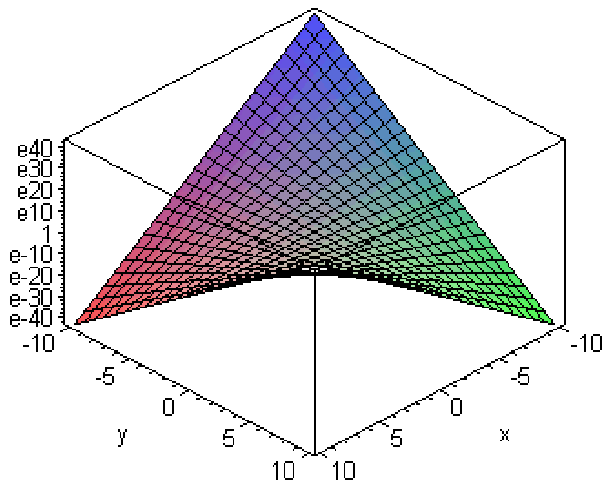
```
[LogLabel, LogOpt, LogOpts, LogTicks, LogView, Logplot, Merge, Ranges, RedoOpts, SmallRangeTicks, minstring, semilogplot3d, strip]
```

Some examples:

In order to appreciate the finer details of many of these examples, I suggest that you send the plot output to a separate window.

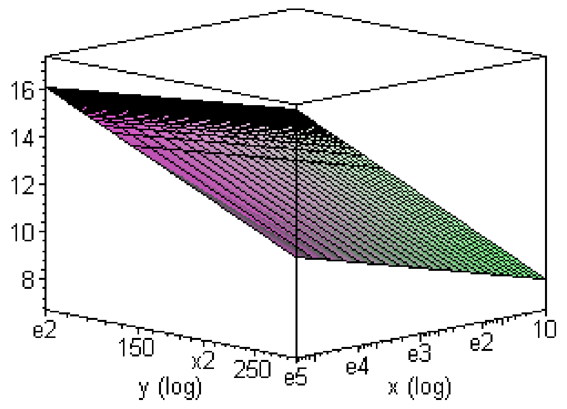
Take an oridinary 3d plot and use logarithmic scaling on the z-axis: (By default, only the last axis is logged.)

```
> Logplot(plot3d(exp(x*y), x= -10..10, y= -10..10, axes= boxed));
```



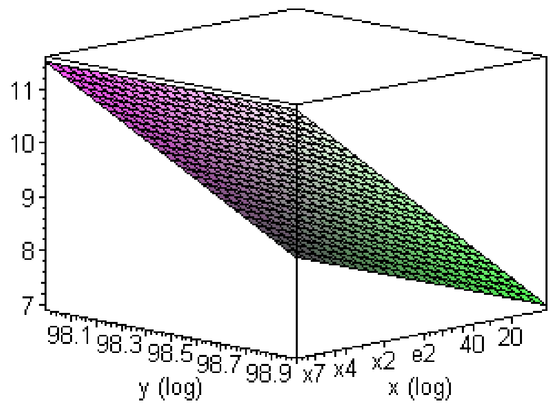
Take an ordinary 3d plot and use logarithmic scaling on the x and y axes. Note the 'logaxes' options specifies which axes are logged.

```
> Logplot(plot3d(ln(x*y), x= 10..10^5, y= 100..300,axes= boxed,orientation= [45,75])
      ,logaxes= [true,true,false]);
```



The problem with that plot is that the grid spacing is not even. That's where the semilogplot3d command comes in. The arguments to semilogplot are exactly the same as the arguments to plot3d, and the optional argument logaxes= [boolean, boolean] can be used to specify which independent axes are logged. By default, both are logged.

```
>    semilogplot3d(ln(x*y), x= 11..999, y= 98..99,axes= boxed,orientation= [45,75]);
```

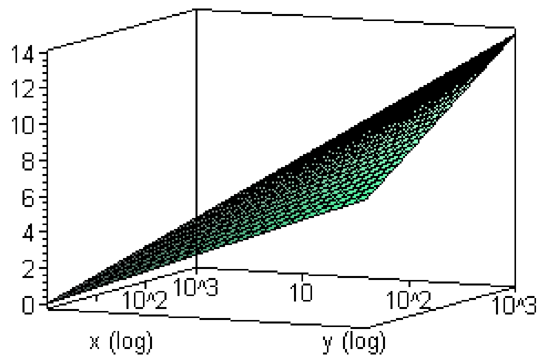


The tickmarks that start with "x" represent multipliers of the next lower printed power of 10. So x2 = 2x10^2 = 200, etc.

The first argument can be a set, can be in operator or expression form, or can be parametric, just like plot3d. Any arguments that can be passed to plot3d can be passed to semilogplot3d.

The axes ranges can be parametric. In this case, I will use strings that can be interpretted as numbers for the tickmarks on one of the axes.

```
>     semilogplot3d(ln(x*y), x= 1..1000, y= x..1000, axes=boxed, orientation= [-25,100]
            ,tickmarks= [["10", "10^2", "10^3"] $ 2, DEFAULT]);
```

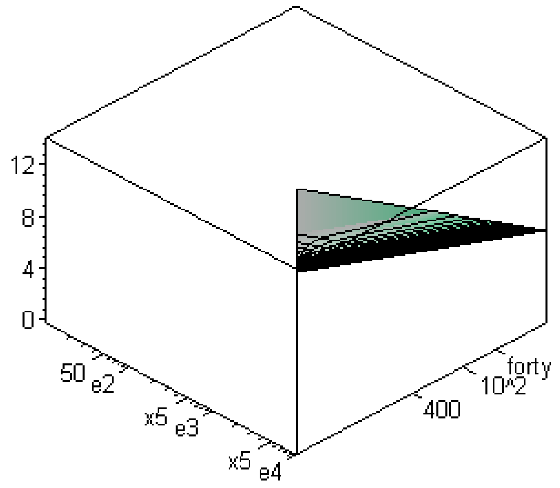


If the plot3d is parametric, you can't get even grid spacing. Note also that you can mix strings and numbers in the tickmarks. Any string that can be interpretted as a number can be used.

```
>     forty:= 40;
```

$$forty := 40$$

```
>     semilogplot3d([10*s, 10*t, log(s*t)], s= 1..1000, t= s..1000, axes= boxed
            ,tickmarks= [["forty", 400, "10^2"], DEFAULT $ 2]
            );
```

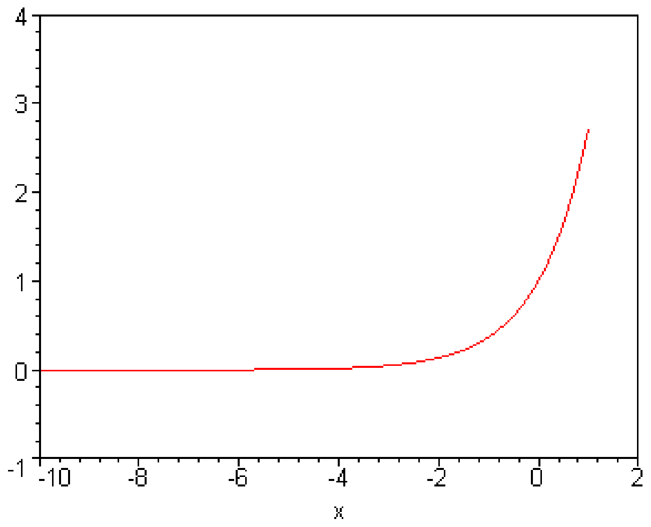


An example of improving the axes on Maple's own logplot command.

```
>   with(plots):
```

```
Warning, the name changecoords has been redefined
```

```
>   plot(exp(x), x= -10..1, view= [-10..2, -1..4], axes= boxed);
```



```
>   P:= %:
```

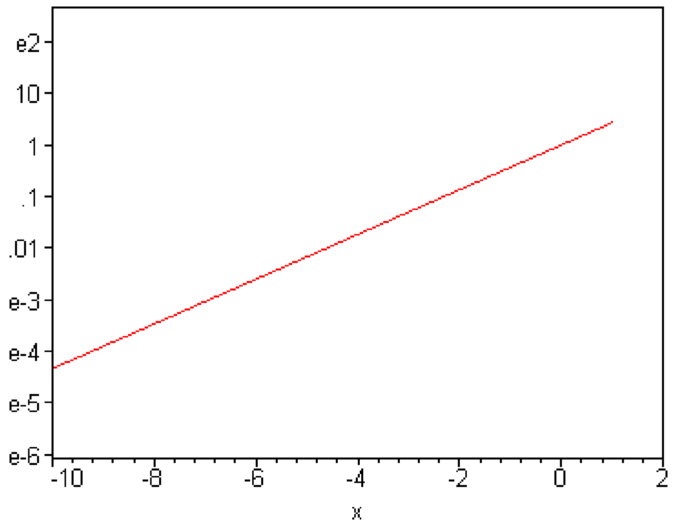
We can't logplot that because of the negative view.

```
>   logplot(exp(x), x= -10..2, view= [-10..2, -1..4], axes= boxed);
```

```
Error, (in plots/dologplot) the view of the y-axis must be strictly positive for a log plot
```

Use procedure RedoOpts to correct that. Note that the view option goes outside the logplot command.

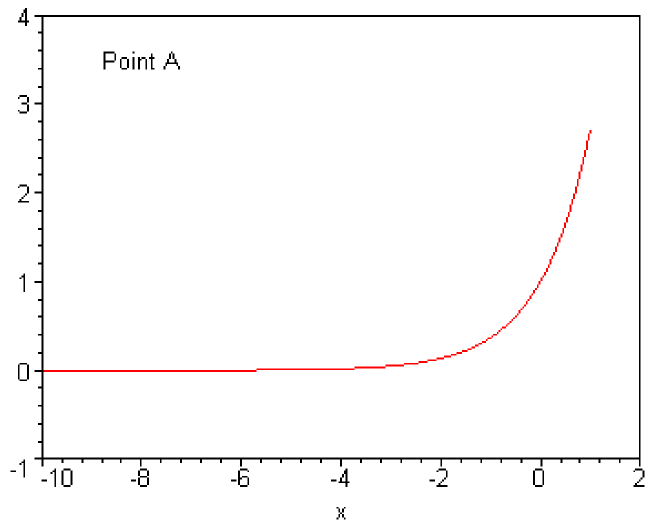
```
>   RedoOpts(logplot(exp(x), x= -10..1, axes= boxed), view= [-10..2, -1..4]);
```



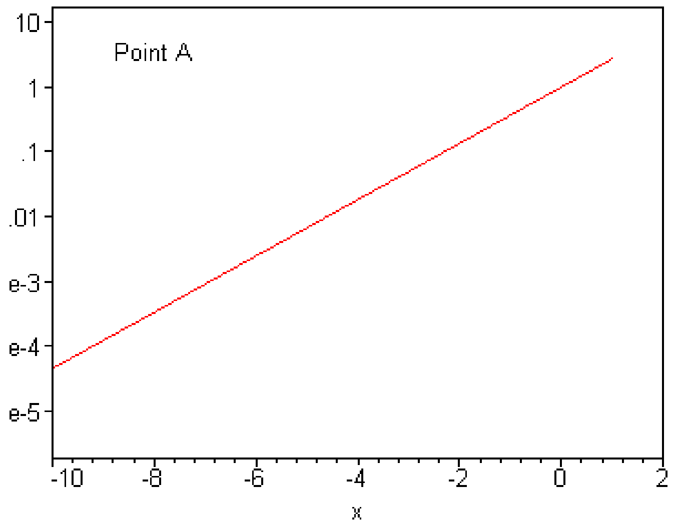
```
>     L:= %:
```

Note that the same size margin top and botton are maintained. So if the original had text in those margins, it will fit the same in the new margins.

```
>     display(P
           ,textplot([-8, 3.5,`Point A`])
           ,axes= boxed
           ,view= [-10..2, -1..4]
           );
```



```
>     Logplot(%);
```



But if the text object contains has a negative y-coordinate, we can't take it's log. For these cases, I wrote procedure Merge.

The first argument to Merge must be a plot structure P returned by a procedure from this module (Logplot, semilogplot3d, RedoOpts, Merge, or LogOpts).   P must have a VIEW large enough to contain all the objects. The remaining arguments, which can be in any order, can be

1. other plot structures that are scaled in the same original (i.e., non log) units as the first argument and are to appear in the same plot with the logarithmic plot, but should not have their log taken.

2. any other plot options that can be used with the display command except view and tickmark options.

It is not necessary to use the "logaxes= " options to specify which axes are logarithmic. Since this procedure only works with log plots produced by this module, it can remember which axes are logarithmic.

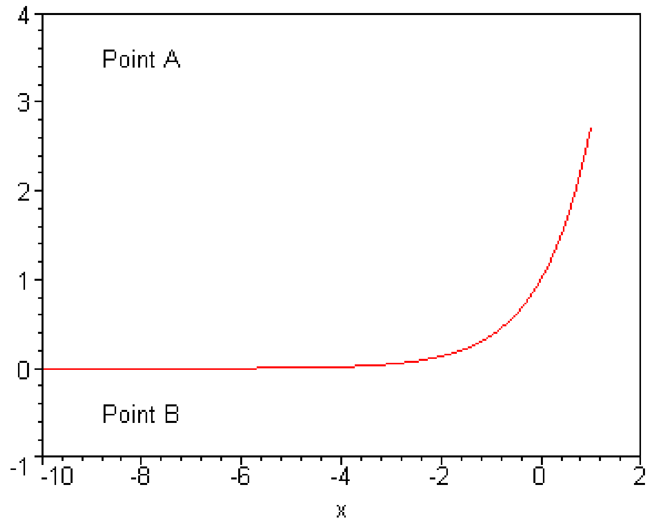
Example:

L was created above with the RedoOpts procedure.

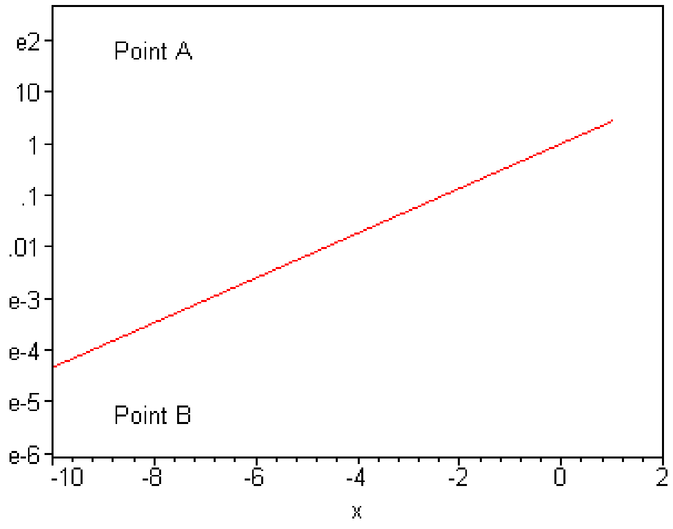
```
>     T1:= textplot([-8, 3.5, `Point A`]):

>     T2:= textplot([-8, -0.5, `Point B`]):

>     display(P,T1,T2);
```



```
>   Merge(L, T1, T2);
```



In this next example, I use use merge to not only maintain the same placement of an object, but also to prevent its shape from being distorted by the logarithm.

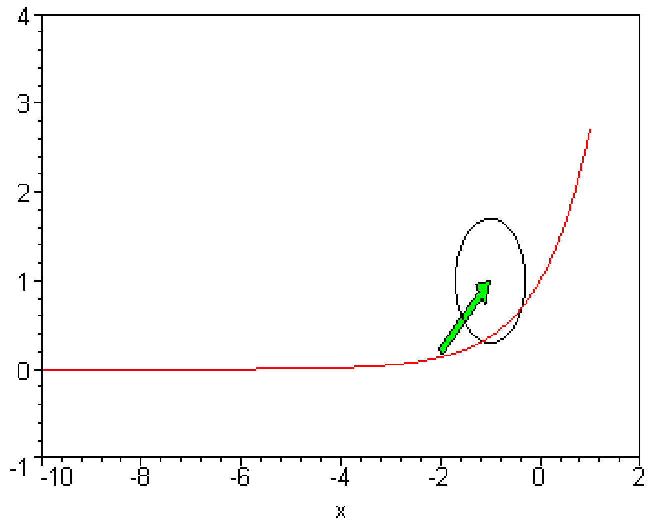
```
>   with(plottools):
```

```
Warning, the name arrow has been redefined
```

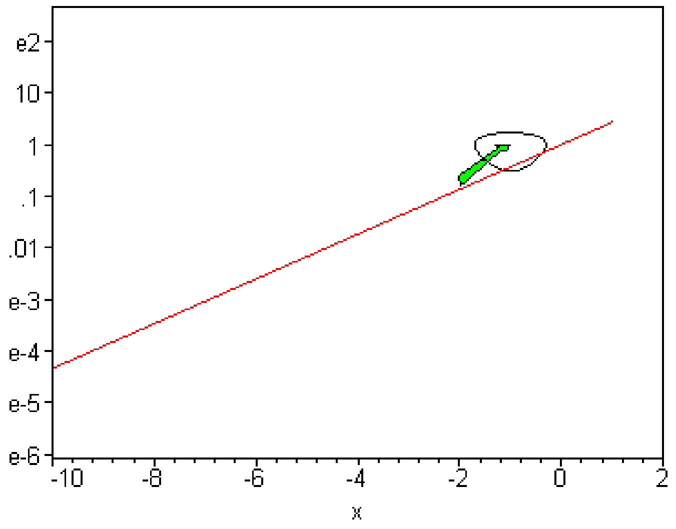
```
>   A:= arrow([-2,.2], [-1,1], .1, .3, .2, color= green):
```

```
> C:= circle([-1,1],.7):
```

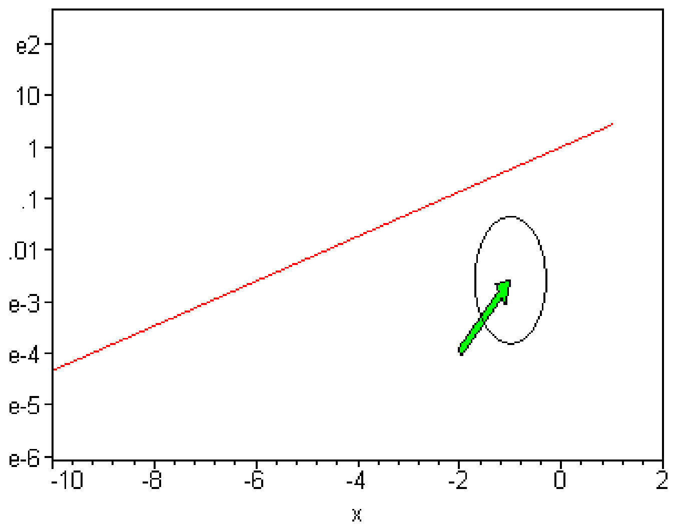
```
> display(P,C,A);
```



```
> Logplot(%);
```



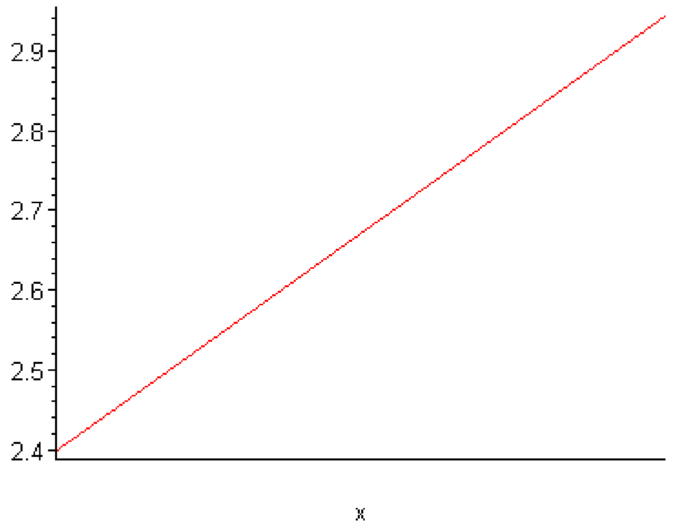
```
> Merge(L, C, A);
```



Note that the ellipse and arrow appear the same, and in the the same place, in both the immediately above plot and its non-log counterpart.

By default, RedoOpts treats the last axis as the log axis. Here we redo a semilogplot.

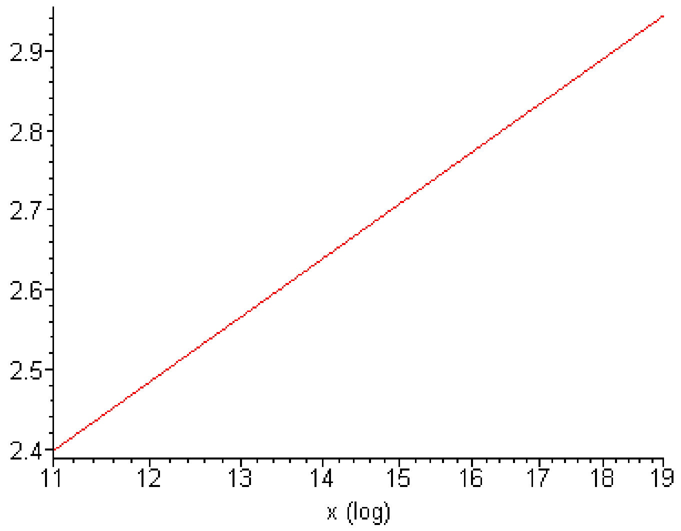
```
>     semilogplot(ln(x), x= 11..19);
```



```
>     P:= %:
```

The lack of tickmarks on the x-axis is annoying. Use RedoOpts.

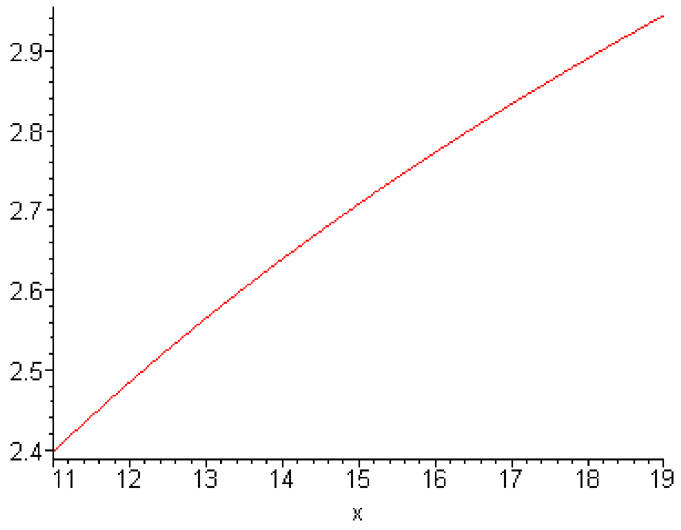
```
>     RedoOpts(P, logaxes= [true, false]);
```



```
> L:= %:
```

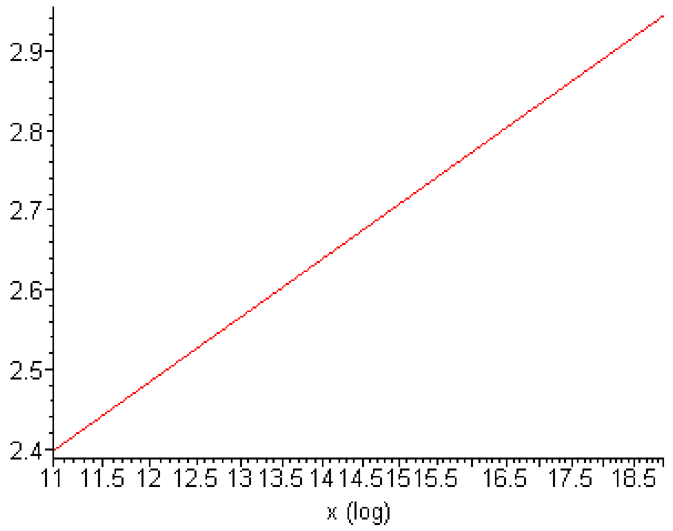
Compare with the tickmarks from the ordinary plot command.

```
> plot(ln(x), x= 11..19);
```



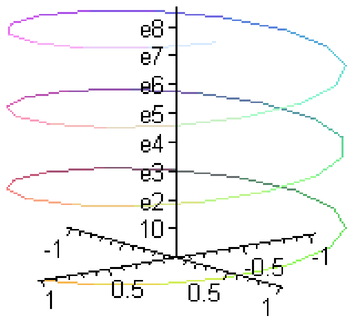
Redo it with more tickmarks. Note that RedoOpts can automatically figure out which axes are logarithmic when it is passed a plot generated by this module.

```
> RedoOpts(L, xtickmarks= 10);
```



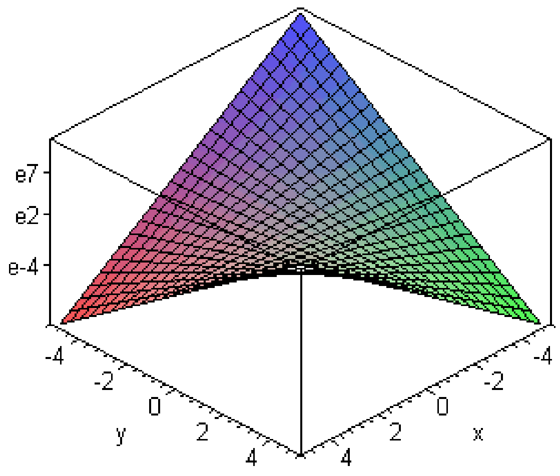
Log plots of other types of plots:

```
>       Logplot(spacecurve([cos(t), sin(t), exp(t)], t= 0..20, axes= normal, orientation= [52,73]));
```



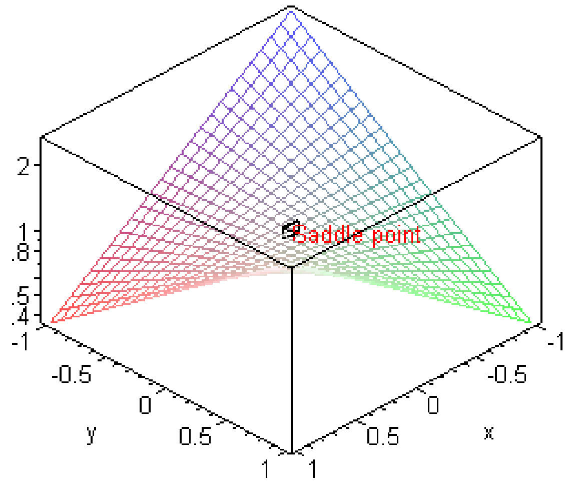
An example with specified tickmarks:

```
>       Logplot(plot3d(exp(x*y), x= -5..5, y= -5..5, tickmarks= [DEFAULT $ 2, [.0001, 100, 10^7]], axes= boxed));
```



A complicated example with text plots and point plots.

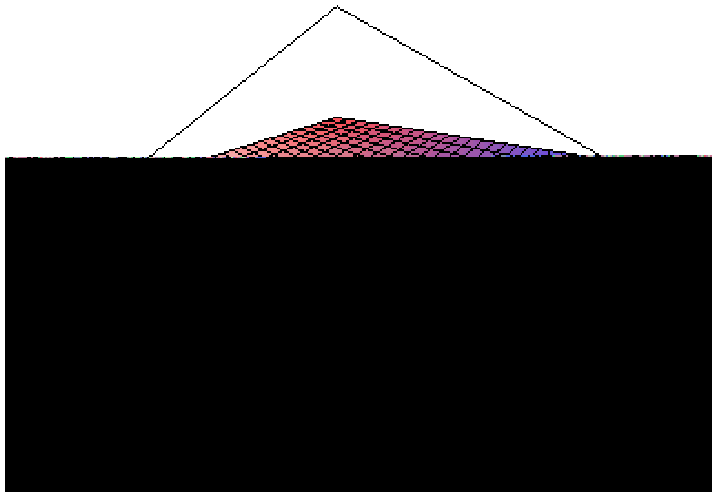
```
>     Logplot
        (display
          (plot3d(exp(x*y), x= -1..1, y= -1..1, axes= boxed, style= wireframe)
          ,pointplot3d([0,0,1], symbol= DIAMOND, symbolsize= 30, color= black)
          ,textplot3d([.1,.1,exp(.1*.1), `Saddle point`], color= red, align= {ABOVE, RIGHT})
          )
        );
```



To transform an animation, it is usually best to transform the pieces, then animate the whole. Doing it in the other order can cause some minor problems with the views and tickmarks.

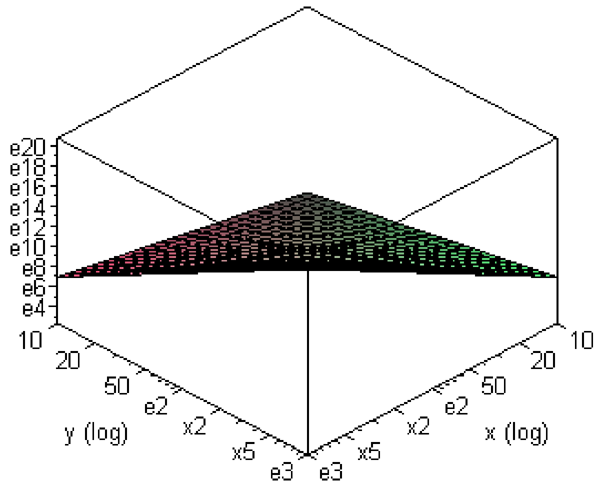
```
>    P:= Logplot(plot3d(exp(x*y), x= -5..5, y= -5..5, axes= boxed)):
```

```
>    display
       ([seq(display(P, orientation= [10*i, 25]), i= -17..18)], insequence= true);
```



To take the log of all axes in a 3d plot, and maintain even grid spacing, use a combination of logplot and semilogplot3d.

```
>    Logplot(semilogplot3d(exp(ln(x)*ln(y)), x= 10..1000, y= 10..1000, axes= boxed));
```



The module exports a procedure Ranges which returns a module which can be used to compute the actual ranges of any plot, regardless of whether it is logarihmic.

```
>    Ranges():-FindRanges(P);
```

```
-5. .. 5.000000000000, -5. .. 5.000000000000, -10.857362047581 .. 10.857362047581
```

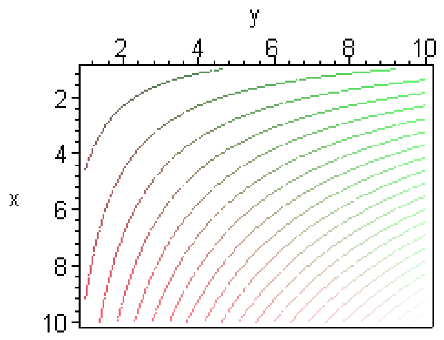
Note that the z-axis is logged.

```
>    Ranges():-FindRanges(plot(sin(x), x= -3..3));
```

```
-3. .. 3., -0.9999778183154985 90 .. 0.9999438135241671 68
```

Last example, a contourplot:

```
>    Logplot(contourplot3d(exp(x*y), x= 1..10, y= 1..10, axes= boxed, orientation= [0,0], contours= 20));
```



```
>
>
>
>
```