Section 18.1 Parametric plotting

In the two dimensional plotting we have done so far, the expression given as the first argument to \texttt{plot} typically involves a variable which is mentioned again in the second argument, e.g. \texttt{plot(t^2+1, t=0..2)}; The plot uses the value of the plot variable (\texttt{t}, in the example) as the first coordinate (horizontal or "x" axis), while the value of the expression specifies the second coordinate of the graph (vertical or "y" axis). Given the plot expression and the range of the plotting variable, we can easily infer that points such as (0,1), or (.5, .5^2+1=1.25) will be on the graph. In other words, the set of points \{ \{t,1+t^2\} | t \in 0..1 \} will be used as the basis for drawing the plot.

The kind of plotting we have done so far is good at drawing curves where the second coordinate is a function of the plot variable, and the first coordinate is just the value of the plot variable.

There is another form of plotting where the first and second coordinates are both expressions of the plot variables. This is called \textit{parametric plotting}.

\texttt{plot} will do a parametric plot if its first argument is a list of three items. The first item is an expression for the \texttt{x} (horizontal axis) coordinate, the second item is an expression for the \texttt{y} (vertical axies) coordinate, and the third item is a variable with an associated range. The \texttt{x} and \texttt{y} expressions should involve this variable.

### Figure 18.1.1 Format of plot used for parametric plotting

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{plot( [ x-expression , y-expression , var = plot range ] )} \\
\hline
\end{tabular}
\end{center}

- x-expression and y-expression should evaluate to numerical values when \texttt{var} has a numerical value.
- Values from the plot range are selected for \texttt{var}.
- A curve described by the (x-expression, y-expression) coordinates as \texttt{var} varies over the plot range is drawn.

### Example 18.1.1 Parametric plotting

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{plot( [cos(t), 2*sin(t), t = 0..2 \pi ], scaling = constrained) } \\
\hline
\end{tabular}
\end{center}

The set of points that will be plotted will be drawn from the set of points
\{ (\cos(t), 2*\sin(t)) | t \in 0..2\pi \}, e.g. points such as (\cos(0), 2*\sin(0)) = (1,0), (\cos(.1), 2*\sin(.1)) = (}
The scaling is constrained so that the x-axis is drawn to the same scale as the y-axis. Otherwise, the plot will be drawn so that the vertical and horizontal ranges take up the same space.

This is the same curve, but the default scaling (unconstrained) is used. The default makes the vertical and horizontal ranges take up the same space, but that magnifies the horizontal scale by a factor of two.

This curve has points on it such as 
\( \cos 0, \sin 0 = (1, 0) \), 
\( \cos .1 \cdot .1, \sin .1 \cdot .1 = (0.09950041653, 0.009983341665) \), 
\( \cos(\pi) \cdot \pi, \sin(\pi) \cdot \pi = (-\pi, 0) \), etc. This curve can't be drawn using the other style of plotting because the curve does not depict a function of \( t \) since there are several points on it for values of \( t \) such as \( t=10 \) or \( t=0 \).

Because the horizontal and vertical ranges are the same, the axes are drawn using the same scaling even though the default scaling is in effect. If we said scaling=constrained in the plot command, we would see the same thing, for this plot.
Section 18.2  Plotting with Maple procedures

Figure 18.2.1 Plotting with Maple procedures

Procedures that work with symbolic inputs and return symbolic or numeric results should be okay to plot. Procedures that fail with symbolic inputs but work with numeric inputs need to be quoted, e.g. plot('f(x)',x=0..1);

We have seen how to use plot to draw the graphs of various kinds of functions such as \( \sin, \cos, \exp, \) etc. We have learned how to define our own Maple procedures with \( \rightarrow \) and with \texttt{proc}...\texttt{end proc}. It is tempting and natural to want to plot such procedures as well. However, there are complications due to the way that Maple evaluates expressions. If the procedure returns a symbolic expression as a result, then plot will work with it. However, if the procedure needs numeric values for arguments, then trying to plot with the procedure will produce an error.

Example 18.2.1  Procedures that need numeric inputs don't work with plot

<table>
<thead>
<tr>
<th>( \text{plot}(\sin(x), x = 0..1) )</th>
<th>This is the plotting we are used to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sin(x) )</td>
<td>( \sin(x) ) \hspace{1cm} (1.2.1)</td>
</tr>
<tr>
<td>( f := (x) \rightarrow 2 + \frac{1}{x} )</td>
<td>( x \rightarrow 2 + \frac{1}{x} ) \hspace{1cm} (1.2.2)</td>
</tr>
<tr>
<td>( f(y), f(.1) )</td>
<td>( 2 + \frac{1}{y}, 12. ) \hspace{1cm} (1.2.3)</td>
</tr>
<tr>
<td>( \text{plot}(f(x), x = 1..2) )</td>
<td></td>
</tr>
</tbody>
</table>
Giving \( f \) a symbolic or numeric argument returns a result. We demonstrate this by creating a sequence with where each kind of argument is used.

Plot works with the function we defined. Maple evaluates \( f(x) \), gets \( 2 + 1/x \), and hands this to plot to work with.

\[
g := \text{proc}(t)
\text{local} \text{val}; \nonumber
\text{val} := \text{int} \left( \sin(x), x = 1 .. t \right); \nonumber
\text{return} \text{val} \cdot 2; \nonumber
\text{end} \nonumber
\text{proc} \nonumber
\text{proc}(t) \nonumber
\text{local} \text{val}; \nonumber
\text{val} := \text{int} \left( \sin(x), x = 1 .. t \right); \nonumber
\text{return} 2 \times \text{val} \nonumber
\text{end proc} \nonumber
\]

\[
g(t), g(.1) \nonumber
2 \cos(1) - 2 \cos(t), -0.9094037188 \nonumber
\]

This defines a function, too.

This function succeeds at returning a result regardless of whether a symbol or a number is given to it. Of course, if you give it a symbol, you get a symbolic expression as a result.

Maple evaluates the set \( \{ f(t), g(t) \} \) and gets \( \{ 2 + 1/t \text{ and } 2 \cos(1) - 2 \cos(t) \} \) before plot starts to operate. It is that set of expressions that plot gets to operate on.
else return -val;
end if;
end proc

proc(t)
    local val;
    val := int(cos(x), x = 1..t);
    if 5 < t then
        return val
    else
        return -val
    end if
end proc

h(.5)
0.3620454462

h(z+1)
Error, (in h) cannot determine if this expression is true or false: 0 < z-4

plot(h(t), t = 0..1)
Error, (in h) cannot determine if this expression is true or false: 5 < t

This is a procedure that has an if statement in it. If statements (as opposed to piecewise expressions) need to be able to decide whether their conditions are true or false whenever they are evaluated. This procedure has only one condition, t>5

The evaluation of h(.5) works.. When it executes, the if statement's condition is "if 5<.5 ...." which is false. Thus h returns -val.

The function does not work when it is given a symbolic input for t. When h(z) is evaluated, t gets the symbolic value z+1. The if statement can't determine "if 5 < z+1" because z has no known value.

This is the reason why trying to plot h fails. Before plot starts to operate, h(t) is evaluated. t has no value because plot hasn't started to operate yet. The evaluation of h(t) fails, making the plot attempt fail.

There are two simple things to remember that will help you to deal with the situation

1. Maple always evaluates the arguments to a function before it starts doing the function. Thus
if you do plot( g(t+1), t = .... ), then g must work with the symbolic input \( t+1 \). It should return an expression that plot can work on, not give an error complaining about if statements or whiles that can't operate properly.

2. If you have a properly defined procedure that only works with numeric input, then you should *quote* the procedure invocation: `plot (g(t+1), t=...)`. This will delay evaluation until after plot starts working and has assigned \( t \) a numeric value.

Quotation should be used sparingly, only in situations where it is needed. Too much quotation may cause other problems.

**Example 18.2.2 Quotation allows plotting of a procedure that always needs numbers to work**

```plaintext
plot( h(z+1), z = 1..7 )
Error, (in h) cannot determine if this expression is true or false: 0 < z-4

plot('h(z+1)', z = 1..7)
```

With \( h \) defined as in the previous example, plot doesn't work because the evaluation of \( h(z+1) \) fails with its if statement.

By quoting the expression, evaluation is \( h(z+1) \) is delayed until after plot starts to work. At that point \( z \) has a numeric value so \( h \)'s if statement will work.

---

**Section 18.3 Animation with plots[animate]**

`plots[animate]` was first discussed in Section 6.7. *animate* requires at least three arguments:

1. The first argument to *animate* is the name of a plotting function (it could be a procedure you define that returns a plot structure).
2. The second argument is a list of arguments to be given to the plotting function.
3. The third argument is a variable (mentioned in the plot) whose value varies from frame to frame of the movie.
4. Other arguments are optional and can include equations of the form frames= ... (number of frames the movie should have), background= ... (a plot structure that should be drawn in every frame as "background" to the other things being drawn. This is also the place to put other options such as labels, axes style, etc.

<table>
<thead>
<tr>
<th>Example 18.3.1 Other ways of producing animations</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>As with display, we can use with to load all the functions in the plot package, in expectation of using several.</td>
</tr>
<tr>
<td>with(plots)</td>
<td>The first parameter to animate is the name of a plotting function. The second parameter is a list of arguments to be giving to that plotting function. The expression to be plotted has an extra parameter (in this case, $A$) that controls how things change from frame to frame. Thus the first frame will be plot $(\sin(0 \cdot x), x=0..10)$. The second frame will be plot$(\sin(0.083333 \cdot x), x=0..10)$, the next frame plot$(\sin(0.16667 \cdot x), x=0..10)$, etc. A increments in steps of 2/25 are being used because the range is from 0.2 and we are using the default number of frames, which is 25. Animate labels each frame with the value of $A$ being used.</td>
</tr>
<tr>
<td>animate(plot, $\left[ \sin(A \cdot x), x = 0 .. 10 \right], A = 0 .. 2$)</td>
<td>(1.3.1)</td>
</tr>
</tbody>
</table>
This is another example of animate. The first frame is the plot produced by `pointplot([0, -0^2+2*0], color=red)`. The second frame is `pointplot([1/10, -(1/10)^2+2*(1/10)], etc. The number of frames in the movie is specified as an additional argument.

We create a non-animated plot structure that depicts a net that stretches from (1.8, 0) to (2.1, 0).

An additional argument to animate is an equation `background=p`, where `p` is a plot structure. It is drawn in every frame of the animation as "background". If we make `net` the background in our animation, we will see the flying point land in the "net".
Section 18.4 For the curious: why is *animate* invoked in the way that it is?

*animate(plot, [sin(A*x), x = 0 .. 10], A = 0 .. 2)* computes several plots, using *plot(sin(A*x), x = 0 .. 10)*. A distinct value of A in the range 0..2 is used for each plot, which constitutes a frame of the movie. We might think we should be able to say this directly, with *animate(plot(sin(A*x), x = 0 .. 10), A = 0 .. 2)*. However, Maple's "evaluate all arguments to functions before invoking them" rule gets in the way here. While sin(A*x) will work even if A doesn't yet have a value, plot will not be able to graph sin(A*x) -- even though it knows how to give x values, plot doesn't know about any values for A, so it would fail before *animate* was ever invoked. Thus, expecting plot(sin(A*x), x = 0 .. 10) as one of the arguments to *animate* isn't going to work.

*animate(plot, [sin(A*x), x = 0 .. 10], A = 0 .. 2)* can work because the only evaluation, that of sin(A*x), that occurs before *animate* gets its arguments, will succeed even with A and x having no numeric values. Then *animate* gets to work and supply a values for A and then invoke plot(sin(A*x), x = 0 .. 10).

The dilemma that language designers have is to balance consistency -- "evaluation of functions always works the same way" -- with convenience in particular circumstances by introducing exceptions -- "make it easier for users to remember how to start up *animate*". Introducing exceptions may slow down the execution of programs because more checking needs to be done at each point to determine whether the exceptional circumstances hold or not. The decisions made rarely satisfy everyone.

Section 18.Z Chapter summary
Example 18.1.1 Parametric plotting

\[ \text{plot}\left( [\cos(t), 2 \sin(t), t = 0 \ldots 2\pi], \text{scaling} = \text{constrained} \right) \]

The set of points that will be plotted will be drawn from the set of points
\[ \{ (\cos(t), 2 \sin(t)) | t \in 0 \ldots 2\pi \}, \] e.g. points such as \((\cos(0), 2 \sin(0)) = (1,0), (\cos(\pi), 2 \sin(\pi)) = (-1,0)\).

The scaling is constrained so that the x-axis is drawn to the same scale as the y-axis. Otherwise, the plot will be drawn so that the vertical and horizontal ranges take up the same space.

\[ \text{plot}\left( [\cos(t), 2 \sin(t), t = 0 \ldots 2\pi] \right) \]

This is the same curve, but the default scaling (unconstrained) is used. The default makes the vertical and horizontal ranges take up the same space, but that magnifies the horizontal scale by a factor of two.

\[ \text{plot}\left( [\cos(t) \cdot t, \sin(t) \cdot t, t = 0 \ldots 10 \cdot \pi] \right) \]

This curve has points on it such as \((\cos(0), \sin(0)) = (1,0), (\cos(\pi), \sin(\pi)) = (-1,0)\), etc. This curve can't be drawn using the other style of plotting because the curve does not depict a function of \(t\) since there are several points on it for values of \(t\) such as \(t=10\) or \(t=0\).

Because the horizontal and vertical ranges are the same, the axes are drawn using the same scaling even though the default scaling is in
effect. If we said scaling=constrained in the plot command, we would see the same thing, for this plot.

Figure 18.2.1 Plotting with Maple procedures

Procedures that work with symbolic inputs and return symbolic or numeric results should be okay to plot. Procedures that fail with symbolic inputs but work with numeric inputs need to be quoted, e.g. plot('f(x)',x=0..1);

Example 18.3.1 Other ways of producing animations

<table>
<thead>
<tr>
<th>Example</th>
<th>Commentary</th>
</tr>
</thead>
</table>
| with(plots)
(animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, | As with display, we can use with to load all the functions in the plot package, in expectation of using several. |

The first parameter to animate is the name
animate(plot, [sin(A*x), x = 0 .. 10], A = 0 .. 2)

A = 0.

animate(pointplot, [[x, -x^2 + 2*x], color = red], x = 0 .. 2, frames = 50)
x = 0.

net := pointplot([[1.8, 0], [2.1, 0]], color = green, symbolsize = 40)

PLOT(…)

animate(pointplot, [[x, -x^2 + 2*x], color = green, symbolsize = 40])
We create a non-animated plot structure that depicts a net that stretches from (1.8,0) to (2.1, 0).

An additional argument to animate is an equation `background=p`, where `p` is a plot structure. It is drawn in every frame of the animation as "background". If we make `net` the background in our animation, we will see the flying point land in the "net".