An Overview of *Maple*

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

General Information

OBJECTIVE: Learn the basic features, characteristics, and language structure of *Maple* .

The working environment for *Maple* is called a worksheet. This Overview is an example of a *Maple* worksheet. Each worksheet is divided into sections as indicated by the plus and minus signs on the left margin of the page. If you start to type in a worksheet, by default Maple takes the typing as an executable command. The right arrow " **[>** " distinguishes sections that contain executable commands from sections that contain non-executable text. You can change an executable section to a text or title section by clicking on the line you want to change, and then clicking the **T** button on the toolbar or by pressing **Ctrl+t** on the keyboard.

You can type several commands in a single section by separating them with semicolons (**;**) and/or colons (**:**). If a semicolon follows a command, the output from the command is displayed. If a colon follows a command, its output is suppressed. One or the other is required at the end of each command. In a section with executable commands, pressing **Shift-enter** moves the cursor to a new line in the same section without executing the commands in the section. If you wish to start a new section you can press **Ctrl+j** (inserts new section after the present one), **Ctrl+k** (inserts new section before the present one), or you can click the **[>** button on the toolbar (which does the same as **Ctrl+j**).

To execute a *Maple* command or group of commands in a section, place the cursor anywhere in the section containing the command(s) to be executed and press **Enter** . All of the commands in the section are executed in sequence from the first to the last. If you wish to execute the entire worksheet, you can select that option from the "Edit" pull-down menu.

In mathematical formulas, always use the times symbol (*) for multiplication.

There are four palettes available for typing standard mathematical forms in commands and in text. These are found in the "View" pull-down menu.

The "Help" pull-down menu is extremely useful when using *Maple* . Formats for *Maple* commands are given with explanations of how they function, variables are defined, and examples that can be cut and pasted into a worksheet are provided.

There is a set of **Technology Guidelines** in a separate subsection at the end of the **Introduction** to each module. Until you get used to *Maple* , you should read these guidelines before continuing with the rest of the module. The following shows what the guidelines say.

Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down *Maple* and start it up again.

Built-In Commands

Maple commands consist of a word or a string of words followed immediately (i.e., no spaces) by a series of arguments enclosed in a pair of round brackets **( )** . The following are examples.

```
> restart;
```

```
> sin(Pi/4);
```

$$\frac{\sqrt{2}}{2}$$

```
> sqrt(16);
```

$$4$$

```
> abs(x);
```

$$|x|$$

```
> expand((a-b)^2)/(c+d)^3;
```

$$\frac{a^2 - 2\,a\,b + b^2}{(c+d)^3}$$

```
> solve(x^2=2, x);
```

$$\sqrt{2}, -\sqrt{2}$$

Words in a command string are typically in lowercase letters. Commands containing multiple-words are typed with no spaces between the words. Also, there is no space between the word in the command string and the opening round bracket that begins the argument list.

In the first four preceding commands, the list of arguments contains only one item, whereas, in the **solve( )** command, the list of arguments contains two items, the equation to solve and the symbol that is to be isolated in the solution. When a command has more than one argument, they are separated with commas.

Lists

Another important structure in *Maple* is the list. A list is any ordered array of items. In *Maple* , lists are enclosed in a pair of square brackets **[ ]** , and the items in a list are separated by commas. The following are examples.

> **[a,b,c,d];**

$$[a, b, c, d]$$

> **[a, a^2, a^3, a^4];**

$$[a, a^2, a^3, a^4]$$

> **[[1,2],[2,3],[3,4]];**

$$[[1, 2], [2, 3], [3, 4]]$$

The last list consists three items, and each of these items is a list of two items.

A list of lists can also be thought of as a matrix or an array. The **matrix( )** command shows a list of lists in more standard matrix form. Each list inside the list is a row of the matrix.

> **matrix([[1,2],[2,3],[3,4]]);**

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$$

Note that while the **matrix( )** form of a list makes it more readable, this form won't work in calculations.

The **seq( )** command is very useful for generating a list when you know a formula for the elements in the list, as in the following example.

> **[seq(2*i,i=1..5)];**

$$[2, 4, 6, 8, 10]$$

> **[seq(2,i=1..5)];**

$$[2, 2, 2, 2, 2]$$

The first argument of the **seq( )** command is a formula that gives each element of the sequence inside the list as a function of the index variable, **i** . The second argument specifies that the elements of the sequence are to be generated by replacing the index **i** in the formula with the integers 1 through 5, each in turn.

You can generate lists of lists in a variety of ways. For example, when the formula for the elements of the list is a function of two indices, you can nest **seq( )** commands.

> **[seq([seq(i*j,j=-2..2)],i=1..5)];**

[[-2, -1, 0, 1, 2], [-4, -2, 0, 2, 4], [-6, -3, 0, 3, 6], [-8, -4, 0, 4, 8], [-10, -5, 0, 5, 10]]

You can generate the same list like this.

> **[seq([-2*i,-i,0,i,2*i],i=1..5)];**

[[-2, -1, 0, 1, 2], [-4, -2, 0, 2, 4], [-6, -3, 0, 3, 6], [-8, -4, 0, 4, 8], [-10, -5, 0, 5, 10]]

2-D Plots

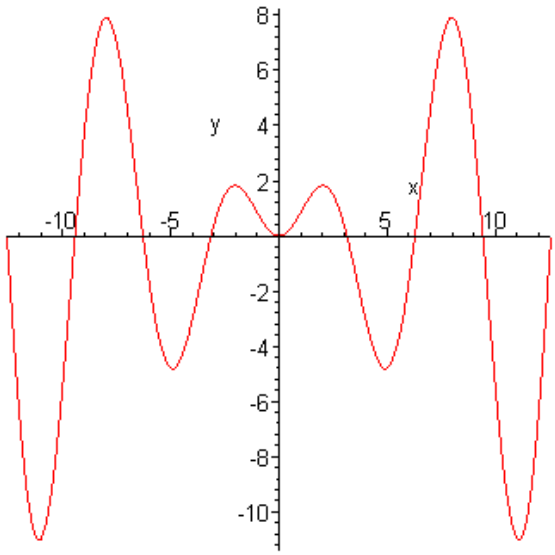
Numerous two-dimensional plot commands in *Maple* are used to generate graphical displays of functions and data. The **plots** package contains even more plot functions. To make the additional plot functions available for use in this worksheet, we read the **plots** package into memory.

> **with(plots):**

Warning, the name changecoords has been redefined

The **plot( )** command is used to graph functions of the form $y = f(x)$.

> **plot(x*sin(x), x=-4*Pi..4*Pi,labels=[x,y]);**

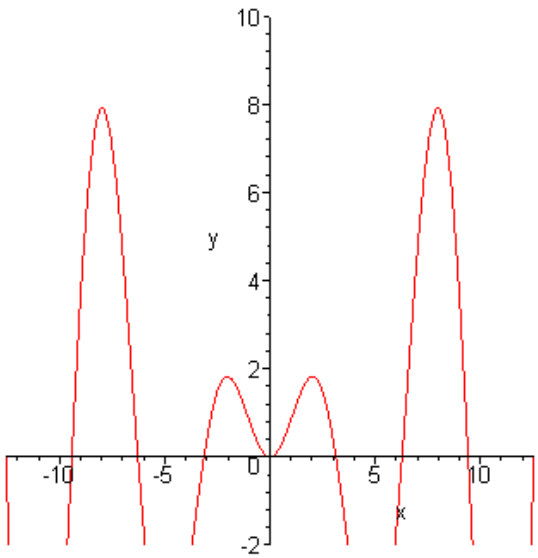


The first argument of the **plot( )** command is the function to graph, and the second specifies the independent variable, plus the left and right ends of the domain over which you wish to graph the function. The first two arguments are required, but you can add other optional specifications like **labels=[x, y]** in the preceding example. Additional options are presented below.

As is the case with any *Maple* command, a colon (**:**) after a **plot( )** command suppresses displaying the output. This is useful when you wish to generate a graph and save it to display later, usually in combination with other graphs. Suppressing the output of a graph also helps to reduce storage requirements for the worksheet.

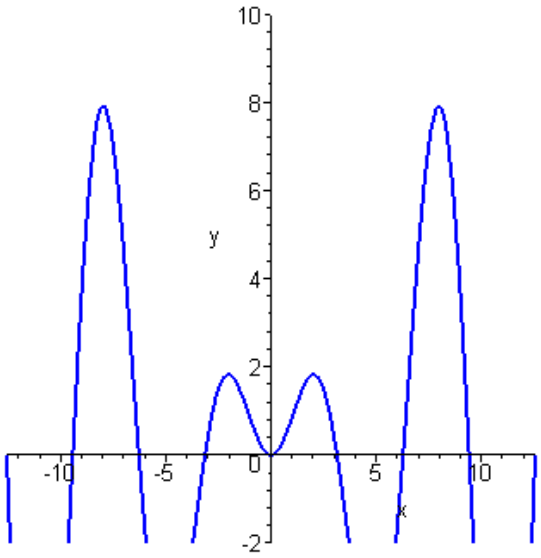
By default, *Maple* selects the range (vertical limits) for the graph. However, you can override this by specifying your own vertical limits as follows.

> **plot(x*sin(x), x=-4*Pi..4*Pi, y=-2..10,labels=[x,y]);**



You can format a graph by specifying a wide variety of options after the first two required arguments in the **plot( )** command.

> **plot(x*sin(x), x=-4*Pi..4*Pi, y=-2..10, color=blue, thickness=2, labels=[x,y]);**

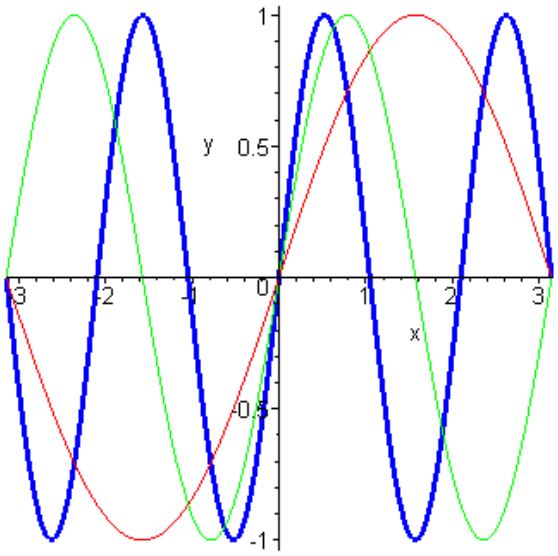


The **plot** options are used to change the attributes of the graph, such as the color and line thickness, as shown in the preceding example. Colors can be specified with a color name like **blue** or **orange** . *Maple* recognizes 21 predefined color names. For additional colors, an RGB designation can be used in place of the color name. For example, the color blue can be designated by **COLOR(RGB 0, 0, 1)** . This RGB representation of the color blue can replace the word **blue** in the preceding **plot( )** command. Each of the three numbers in the RGB specification ranges from 0 to 1 and represents the amount of red, green, and blue, respectively, in the color.

Other options can be added to **plot( )** commands. Commas are used separate whatever options are specified. The following command plots three functions on the same graph; each line is a different color, and the third one is

thicker.

> **plot([sin(x),sin(2*x),sin(3*x)], x=-Pi..Pi, color=[red,green,blue],thickness=[0,0,3],labels=['x','y']);**



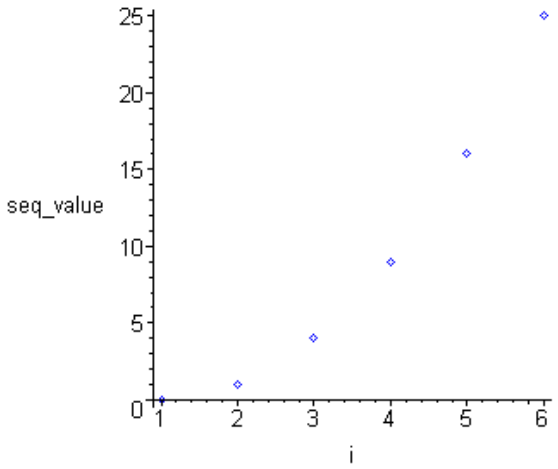
Note that inside the list for each option, the attributes for the three graphs are specified in order from first to last as they occur in the list of functions to plot. For example, in the option **color = [red, green, blue]** , **red** is the color for **sin(x)** , **green** is for **sin(2*x)** , and **blue** is for **sin(3*x)** .

The **listplot( )** command is used to plot lists where only the *y* -coordinate values of the plot points are specified. By default, the *x* -coordinate values are taken to be 1, 2, ...

> **plotvalues:=[seq(i^2,i=0..5)];**

$$plotvalues := [0, 1, 4, 9, 16, 25]$$

> **listplot(plotvalues, style=POINT, color=COLOR(RGB,0,0,1),labels=['i','seq_value']);**

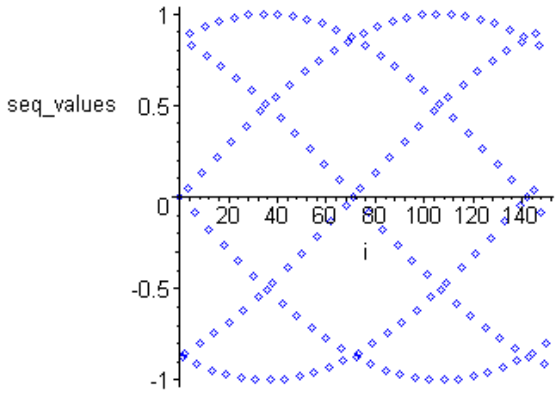


When both coordinates of the points in a list are specified, the **pointplot( )** command is used.

> **plotvalues:=[seq([i,sin(200.0*i)],i=0..150)];**

plotvalues := [[0, 0.], [1, -0.8732972972], [2, -0.8509193596], [3, 0.04418244833], [4, 0.8939696482], [5, 0.8268795405], [6, -0.08827860647], [7, -0.9128960386], [8, -0.8012247907], [9, 0.1322023528], [10, 0.9300395044], [11, 0.7740052149], [12, -0.1758679022], [13, -0.9453665637], [14, -0.7452739741], [15, 0.2191899743], [16, 0.9588472821], [17, 0.7150871818], [18, -0.2620839590], [19, -0.9704553311], [20, -0.6835037939], [21, 0.3044660827], [22, 0.9801680398], [23, 0.6505854941], [24, -0.3462535712], [25, -0.9879664388], [26, -0.6163965734], [27, 0.3873648118], [28, 0.9938352975], [29, 0.5810038040], [30, -0.4277195126], [31, -0.9977631538], [32, -0.5444763096], [33, 0.4672388590], [34, 0.9997423364], [35, 0.5068854299], [36, -0.5058456682], [37, -0.9997689800], [38, -0.4683045816], [39, 0.5434645394], [40, 0.9978430324], [41, 0.4288091145], [42, -0.5800220013], [43, -0.9939682551], [44, -0.3884761651], [45, 0.6154466557], [46, 0.9881522157], [47, 0.3473845053], [48, -0.6496693167], [49, -0.9804062732], [50, -0.3056143889], [51, 0.6826231461], [52, 0.9707455558], [53, 0.2632473946], [54, -0.7142437835], [55, -0.9591889311], [56, -0.2203662670], [57, 0.7444694726], [58, 0.9457589699], [59, 0.1770547548], [60, -0.7732411813], [61, -0.9304819014], [62, -0.1333974471], [63, 0.8005027172], [64, 0.9133875624], [65, 0.08947960859], [66, -0.8262008375], [67, -0.8945093388], [68, -0.04538701259], [69, 0.8502853525], [70, 0.8738841005], [71, 0.001205773842], [72, -0.8727092242], [73, -0.8515521297], [74, 0.04297781983], [75, 0.8934286579], [76, 0.8275570414], [77, -0.08707747601], [78, -0.9124031876], [79, -0.8019456992], [80, 0.1310070662], [81, 0.9295957552], [82, 0.7747681232], [83, -0.1746807940], [84, -0.9449727830], [85, -0.7460773922], [86, 0.2180133629], [87, 0.9585042389], [88, 0.7159295404], [89, -0.2609201424], [90, -0.9701636955], [91, -0.6843834480], [92, 0.3033173338], [93, 0.9799283813], [94, 0.6515007257], [95, -0.3451221337], [96, -0.9877792255], [97, -0.6173455949], [98, 0.3862528954], [99, 0.9937008950], [100, 0.5819847620], [101, -0.4266292888], [102, -0.9976818246], [103, -0.5454872882], [104, 0.4661724572], [105, 0.9997142394], [106, 0.5079244547], [107, -0.5048051710], [108, -0.9997941699], [109, -0.4693696232], [110, 0.5424519790], [111, 0.9979214602], [112, 0.4298980930], [113, -0.5790393553], [114, -0.9940997675], [115, -0.3895869536], [116, 0.6144958432], [117, 0.9883365559], [118, 0.3485149344], [119, -0.6487521948], [120, -0.9806430812], [121, -0.3067622508], [122, 0.6817415058], [123, 0.9710343691], [124, 0.2644104475], [125, -0.7133993468], [126, -0.9595291857], [127, -0.2215422394], [128, 0.7436638887], [129, 0.9461500012], [130, 0.1782413499], [131, -0.7724760235], [132, -0.9309229456], [133, -0.1345923475], [134, 0.7997794799], [135, 0.9138777582], [136, 0.09068048061], [137, -0.8255209332], [138, -0.8950477288], [139, -0.04659151087], [140, 0.8496501091], [141, 0.8744696333], [142, 0.002411545932], [143, -0.8721198824], [144, -0.8521836616], [145, 0.04177312885], [146, 0.8928863686], [147, 0.8282333391], [148, -0.08587621895], [149, -0.9119090100], [150, -0.8026654419]]

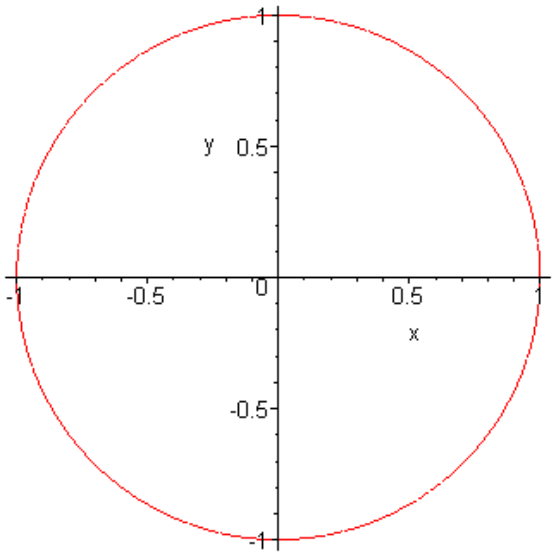
> **pointplot(plotvalues, color=COLOR(RGB,0,0,1),labels=['i','seq_values']);**



Just as you can in the **plot( )** command, you can also specify formatting options in the **listplot( )** command, as shown in the preceding examples.

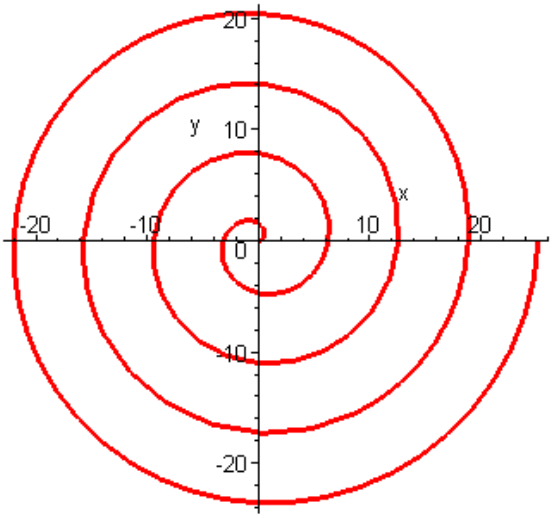
Another useful two-dimensional plot command is **implicitplot( )** . You can use this command to plot curves for mathematical relations that are not functions; that is, they cannot be written in the form $y = f(x)$. For example, you can graph the curve $x^2 + y^2 = 1$ as follows.

```
> implicitplot(x^2+y^2=1,x=-1..1,y=-1..1,labels=[x,y]);
```



Graphs of curves expressed in parametric form are plotted with the **plot( )** command. For example, if the x-coordinates of the points on a curve are given by $x = t \cos t$, and the y-coordinates are given by $y = t \sin t$, then the curve can be graphed like this.

```
> plot([t*cos(t), t*sin(t), t=0..8*Pi], labels=[x,y], color=COLOR(RGB,1,0,0), thickness=3,
  scaling=constrained,labels=[x,y]);
```

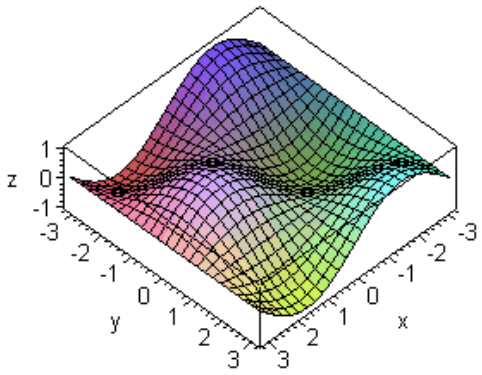


3-D Plots

There are numerous *Maple* commands for plotting three-dimensional functions and data. In this section we introduce two of them, and in later modules we will introduce more.

The **plot3d( )** command is used to plot functions of two independent variables, that is, functions of the form $z = f(x, y)$.

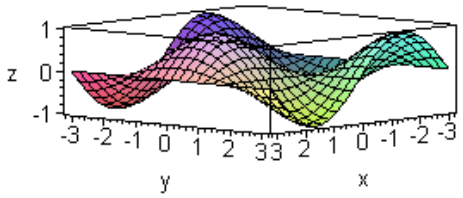
```
> plot3d(sin(x)*cos(y), x=-Pi..Pi, y=-Pi..Pi, labels=[x,y,z], axes=BOXED, scaling=CONSTRAINED);
```



The first three arguments of the **plot3d( )** command are required. They are: 1) the function to graph, 2) the first independent variable and its limits for the graph, and 3) the second independent variable and its limits. As is the case with two-dimensional graphics, you can specify a number of formatting options after the first three arguments in the command.

To change the viewpoint for the graph, you can include an **orientation** indicator as an option in the **plot3d( )** command.

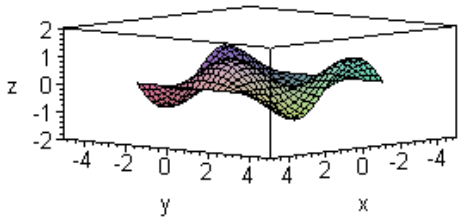
> **plot3d(sin(x)*cos(y), x=-Pi..Pi, y=-Pi..Pi, labels=[x,y,z], axes=BOXED, scaling=CONSTRAINED, orientatio [42,84]);**



If you want to look at the graph from different viewpoints, you can click anywhere on the graph and drag the mouse. The plot will automatically adjust to give you a different view.

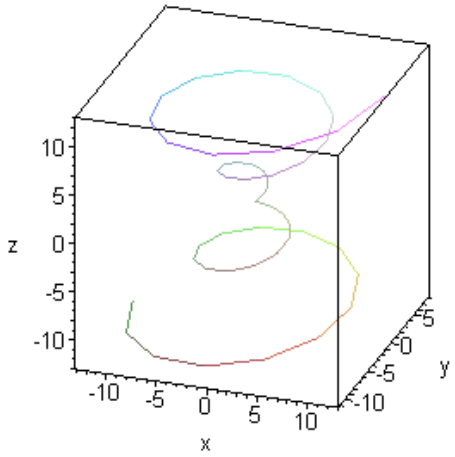
Another way of changing the view of the plot is to set the **view** option. This option allows you to zoom in and out of the plot without changing the plot range.

> **plot3d(sin(x)*cos(y), x=-Pi..Pi, y=-Pi..Pi, labels=[x,y,z], axes=BOXED, scaling=CONSTRAINED, orientatio [42,84], view=[-5..5, -5..5, -2..2]);**



The second three-dimensional plot command is **spacecurve( )** . You can use this command to plot three-dimensional curves that are specified in parametric form. For example, if the x -coordinates of the points on a curve are given by $t \cos t$, the y -coordinates by $t \sin t$, and the z -coordinates by t , then the curve can be graphed as follows.

> **spacecurve([t*cos(t),t*sin(t),t], t=-4*Pi..4*Pi, labels=[x,y,z], orientation=[-71,65], axes=boxed);**



Assignment Commands

In *Maple* the **colon-equals** (**:=**) is used to assign items (values, lists, graphs, etc.) to variable names. Whenever a colon-equals assignment command is executed, *Maple* evaluates the expression on the right side of the colon-equals and assigns the result to the variable name on the left side. If the expression on the right side includes variables that have not been assigned values, then *Maple* evaluates the right-side expression to the extent possible with values that are available, and then assigns the resulting expression to the variable on the left side of the colon-equals. Here are some examples.

```
> restart;
```

```
> x:=2;
```

$$x := 2$$

```
> y:=x+2;
```

$$y := 4$$

```
> z:=y^3+w;
```

$$z := 64 + w$$

```
> s:=x+y+z;
```

$$s := 70 + w$$

In the third assignment command, the variable w has no assigned value, but y does. Therefore, evaluation gives an expression in terms of w, and this expression is assigned to z .

You can always check to see what has been assigned to a variable by typing the name of the variable followed by a semicolon.

> **x;**

$$2$$

> **y;**

$$4$$

> **z;**

$$64 + w$$

> **s;**

$$70 + w$$

Once a value or expression has been assigned to a variable, the variable name is replaced with the assigned value or expression wherever it is used in subsequent expressions, as shown in the following examples.

> **x^2;**

$$4$$

> **3*y^2;**

$$48$$

> **sin(z);**

$$\sin(64 + w)$$

The **unassign( )** command removes assignments that you have made to variables.

> **unassign('x');**

If you check a variable name after it has been unassigned, you see that it is not replaced with a value or expression.

> **x;**

$$x$$

If you use the variable name in an expression, it will be treated as a simple variable with nothing assigned to replace it.

> **x+2;**

$$x + 2$$

You can unassign more than one variable at a time.

> **unassign('x','y','z');**

Redefining a variable automatically clears whatever was previously assigned to it. Therefore, it is unnecessary to unassign a variable name when a new assignment is made to it. The following example demonstrates this feature.

> **a:=2;**

$$a := 2$$

> **a;**

$$2$$

> **a:=12345;**

$$a := 12345$$

> **a;**

$$12345$$

It is extremely important to keep close track of the variable names that you use in a *Maple* session. If you use a new variable name in an expression or a command and you are assuming that it is clear of assignments, you should first use the **unassign( )** command to erase all previous assignments to the new variable name. Failure to do this can produce misleading and confusing results. The following commands illustrate what can happen if you forget to unassign a variable name before you use it. Suppose we want to use **s** in an expression and we have forgotten that we previously assigned another expression to **s** .

> **expand((s+2)*(s-3));**

$$4824 + 139\,w + w^2$$

Now we fix the problem.

> **unassign('s');**

> **expand((s+2)*(s-3));**

$$s^2 - s - 6$$

That's more like it!

There is another item that is related to the issue of variable-name management. If you finish a *Maple* worksheet or one of our modules and wish to start another one, it is best to save your work and then execute the **restart** command at the beginning of the new module or worksheet. This clears any assignments that have been made to variables. Otherwise, the assignments that were made in the old worksheet are carried over into the new one. Executing the **restart** command also removes any packages that were previously read into memory. Therefore, if you want to use a package after you **restart** *Maple* , you need to reload it.

Special Packages

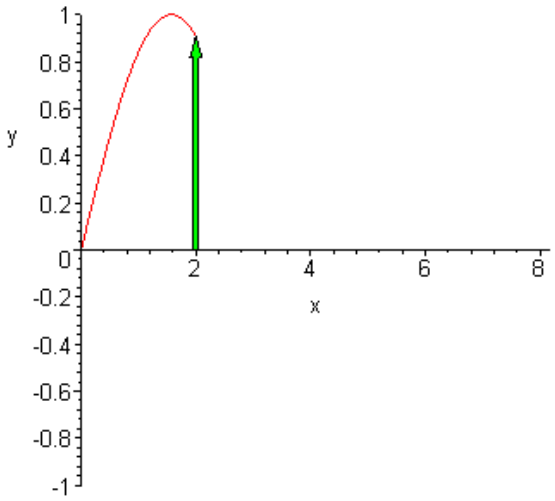
While there are many built-in commands that are available for use as soon as you open a *Maple* worksheet, there are many more specialized commands that are contained in packages. These packages are not automatically available in a *Maple* worksheet, and in order to use the specialized commands that they contain, you have to read them into computer memory. You do this by typing **with(*packagename*)** , and then executing the command. Here is an example.

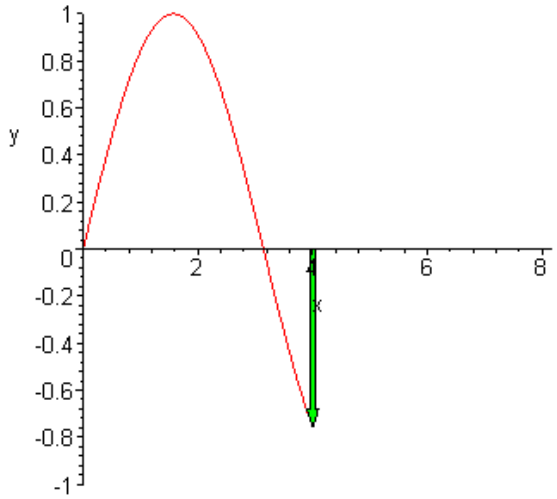
```
> restart;
```

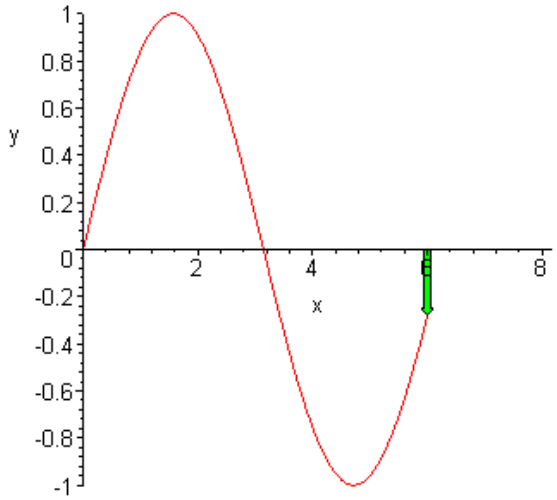
```
> with(plottools):
```

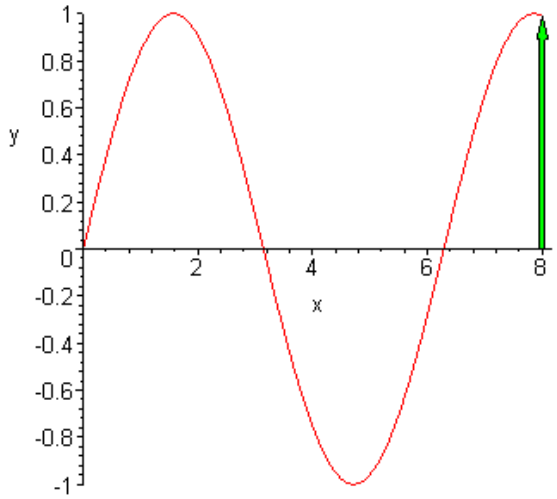
Now we can use the **arrow( )** command that is available after the **plottools** package is read into memory.

```
> for a from 2 to 8 by 2 do
    arr1:=arrow([a,0], [a,sin(a)], .1 , .2 , .1, color=green):
    p1:=plot(sin(x), x=0..a,labels=[x,y]):
    print(plots[display]({p1, arr1}, view=[0..8.2, -1..1]));
  od:
```









The command in the preceding section is a loop that creates the series of graphs. The first command inside the loop

(indented) defines the arrow and the second indented command defines the plot of sin x . The third indented command displays the two plot objects, the arrow and the curve, together. Colons after the first two commands inside the loop suppress showing the arrow and the graph until they are displayed together by the third command. The **od:** at the end closes the loop. The specification in the **for_do_od** command causes the variable **a** , the upper **x** -limit on the graph, to vary from 2 to 8 in steps of 2.

To find out what *Maple* packages are available type **?index** to open the help files, and then select **packages** from the list.

Before you use any of the specialized commands available in a package, be sure to read in the package first. A package needs to be read in only once during a *Maple* session unless you **restart** . If you try to execute a package command before you have read in the package, the command won't work and you may get an error message. You can fix the problem by reading in the package. Then the command will work. The commands in the following section illustrate what happens when you try to execute a package command before you load the package, and how to fix the problem.

```
> restart;
  unassign('y');
  implicitplot(y=sin(x), x=0..2*Pi, y=-1..1, filled=true);
```

$$\text{implicitplot}(y = \sin(x), x = 0 .. 2\pi, y = -1 .. 1, \mathit{filled} = \mathit{true})$$

The **implicitplot( )** command didn't work.

Now read in the **plots** package.

```
> with(plots);
```

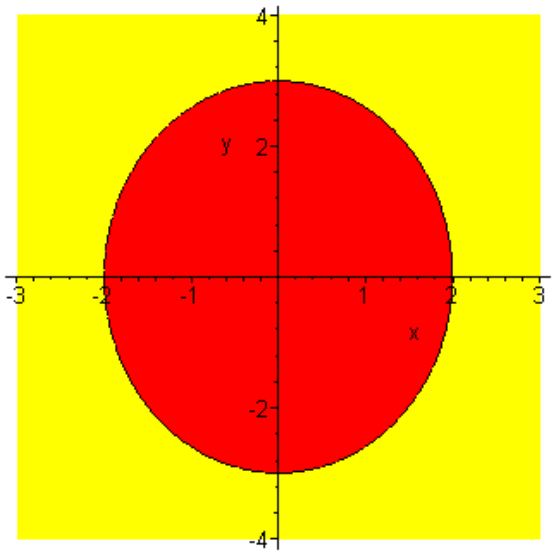
Warning, the name changecoords has been redefined

[*animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, cylinderplot, densityplot, display, display3d, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, replot, rootlocus, semilogplot, setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot, surfdata, textplot, textplot3d, tubeplot*]

Note that all the functions available in the **plots** package are listed. To suppress the list of functions, use a colon in place of the semicolon after the **with( )** command.

Now the **implicitplot( )** command will work.

```
> implicitplot(9*x^2+4*y^2=36, x=-3..3, y=-4..4, filled=true);
```



Making Your Own Commands

As a user and programmer in *Maple,* you can design and construct your own functions and commands.

Suppose, for example, that you are going to use the function $x^2 \sin(x)$ over and over again in a program. You may wish to create a new function, as illustrated by the following definition.

> **restart;**

> **myfunction:=x->x^2*sin(x);**

$$myfunction := x \to x^2 \sin(x)$$

Maple displays the definition of the new function. The variable *x* is a dummy variable that will be replaced by real input in the argument list, whenever **myfunction( )** is called. The following examples illustrate how you to use **myfunction( )** after you have defined it.

> **myfunction(2);**

$$4 \sin(2)$$

> **myfunction(2.0);**

$$3.637189707$$

> **myfunction(r);**

$$r^2 \sin(r)$$

> **myfunction(a+b);**

$$(a+b)^2 \sin(a+b)$$

The built-in **map( )** command can be used to apply a function to each of the values in a list.

```
> map(myfunction,[2,4]);
```

$$[4 \sin(2), 16 \sin(4)]$$

The first argument of the **map( )** command is the name of the function to apply (without the round brackets), and the second element is the list of values to be used in place of the function's argument. In the preceding example, **myfunction** is mapped over the list **[2, 4]** .

You can design commands that perform more than one operation by using the **proc( )** command. To create your own command, use a colon-equals (**:=**) assignment. On the left side of the **:=** , place the name that you wish to use for your command, and, on the right side, place the **proc( )** command with the input variables inside the round brackets. Immediately after the **proc( )** command is a list of local variables, headed by the word **local** . The local variable list is then followed by a list of the commands that you want your procedure to execute. The following example shows how to construct a command called **buildplot( )** . We want **buildplot( )** to generate a list of decaying numbers, plot the list, and return the list as output. First, we read the **plots** package into memory, because **buildplot( )** uses the **listplot( )** command from this package.

```
> with(plots):
  buildplot:=proc(n)
   local i,plotlist;
   plotlist:=[seq(0.8^(i-1), i=1..n)];
   print(listplot(plotlist, style=point, labels=[index,number]));
   RETURN(plotlist);
  end;
```

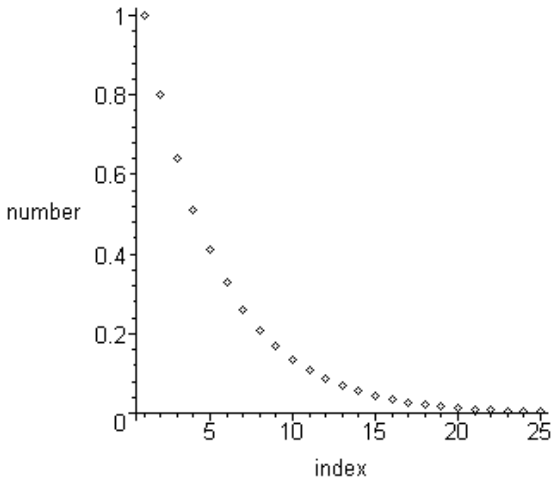
Warning, the name changecoords has been redefined

```
buildplot := proc(n)
local i, plotlist;
    plotlist := [seq(0.8^(i - 1), i = 1 .. n)];
    print(listplot(plotlist, style = point, labels = [index, number]));
    RETURN(plotlist)
end proc
```

In **buildplot( )** , the list of local variables contains the variable **plotlist** . Making a variable local to a command means that any assignments you make to the variable inside the procedure will not be available outside the procedure. If you type **plotlist;** after you execute the **buildplot( )** command, you will find that nothing has been assigned to **plotlist** . We will check it out below, after we execute the **buildplot( )** command.

We execute our new **buildplot( )** command the same way we would execute any other *Maple* command.

```
> buildplot(25);
```



[1., 0.8, 0.64, 0.512, 0.4096, 0.32768, 0.262144, 0.2097152, 0.16777216, 0.134217728, 0.1073741824, 0.08589934592, 0.06871947674, 0.05497558139, 0.04398046511, 0.03518437209, 0.02814749767, 0.02251799814, 0.01801439851, 0.01441151881, 0.01152921505, 0.009223372037, 0.007378697629, 0.005902958104, 0.004722366483]

Now we check the variable name **plotlist** .

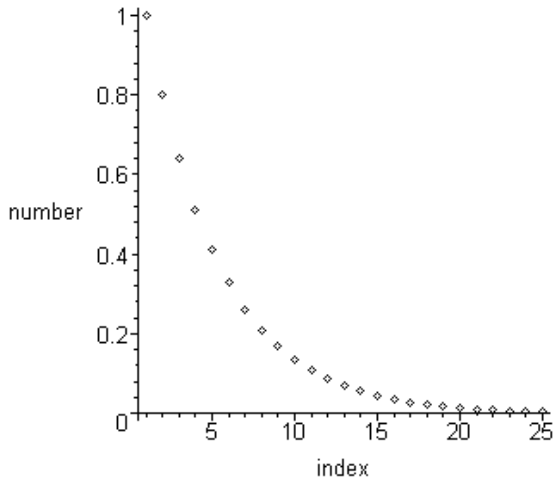
> **plotlist;**

plotlist

In the global environment outside the **buildplot( )** procedure, the variable **plotlist** is clear.

The **RETURN(plotlist)** command inside the procedure returns the list of values stored in the local **plotlist** variable. If you want to save the output list of values for later use, you can assign it to a variable name, as we do next.

> **decay:=buildplot(25);**



decay := [1., 0.8, 0.64, 0.512, 0.4096, 0.32768, 0.262144, 0.2097152, 0.16777216, 0.134217728, 0.1073741824, 0.08589934592, 0.06871947674, 0.05497558139, 0.04398046511, 0.03518437209, 0.02814749767, 0.02251799814, 0.01801439851, 0.01441151881, 0.01152921505, 0.009223372037, 0.007378697629, 0.005902958104, 0.004722366483]

If you use the variable name **decay** , you will get the list of numbers that **buildplot( )** returned as output.

> **decay;**

[1., 0.8, 0.64, 0.512, 0.4096, 0.32768, 0.262144, 0.2097152, 0.16777216, 0.134217728, 0.1073741824, 0.08589934592, 0.06871947674, 0.05497558139, 0.04398046511, 0.03518437209, 0.02814749767, 0.02251799814, 0.01801439851, 0.01441151881, 0.01152921505, 0.009223372037, 0.007378697629, 0.005902958104, 0.004722366483]

Elements of Lists

So far, we have used square brackets **[ ]** to enclose the elements of a list, and round brackets **( )** to enclose the arguments in *Maple* commands. Round brackets are also used as grouping symbols in mathematical expressions, like **(x + y)*(x - y)** for example.

Square brackets also have another use; it is to select elements from a list. Consider the following examples.

> **odds:=[1,3,5,7,9,11,13,15];**

odds := [1, 3, 5, 7, 9, 11, 13, 15]

To select the third element of **odds,** use a selector specification after the name of the list, as follows.

> **odds[3];**

5

The selector in the next example selects a series of values from the list and returns them in a new list.

> **odds[2..5];**

$$[3, 5, 7, 9]$$

For nested lists, that is, a list of lists, use multiple selectors like **[3, 2]** to retrieve elements. The first number points to an individual element in the outermost list, which may itself be a list. The second number points to an element inside the inner list. The following examples illustrate how this works.

> **oddeven:=[[1,2],[3,4],[5,6],[7,8],[9,10]];**

$$oddeven := [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]$$

The following command retrives the fouth list form **oddeven** .

> **oddeven[4];**

$$[7, 8]$$

To retrive the second element of the third list in **oddeven** type:

> **oddeven[3,2];**

$$6$$

Making Substitutions

In *Maple* , the equals sign (**=**) is used to express mathematical equality and to make substitutions into commands and/or mathematical expressions. In this section, we consider how the equals sign is used to make substitutions.

Normally, when the name of a command is used, the values enclosed by the round brackets after the command name are automatically substituted for the arguments of the command. These values are then used in the evaluating the command's procedure. The following example uses a function to show how this substitution works.

> **unassign('x','y','z');**

> **f:=(x,y)->sin(x)^2+cos(y)^2;**

$$f := (x, y) \rightarrow \sin(x)^2 + \cos(y)^2$$

> **f(Pi/4,Pi);**

$$\frac{3}{2}$$

> **f(theta,theta);**

$$\sin(\theta)^2 + \cos(\theta)^2$$

Another way to define a function is without using the mapping symbol (**->**). If a function is defined this way, then you need to explicitly specify the substitutions to make for the arguments. The next example shows this type of function definition, and how to make substitutions for the arguments using the **eval( )** and **subs( )** commands.

> **f:=sin(x)^2+cos(y)^2;**

$$f := \sin(x)^2 + \cos(y)^2$$

> **theta:=Pi/8;**

$$\theta := \frac{\pi}{8}$$

> **eval(f, [x=theta, y=theta]);**

$$\sin\left(\frac{\pi}{8}\right)^2 + \cos\left(\frac{\pi}{8}\right)^2$$

> **subs([x=theta, y=theta],f);**

$$\sin\left(\frac{\pi}{8}\right)^2 + \cos\left(\frac{\pi}{8}\right)^2$$

The difference between the **eval( )** command and the **subs( )** command is that **eval( )** more fully evaluates **f** after substituting for **x** and **y** , whereas **subs( )** does not evaluate **f** after substitution. Usually, the **eval( )** command is preferred.

Retrieving Output from Commands

If you execute a *Maple* command, the output is displayed in blue characters on the next line after the command.

> **solve(x^2=1,x);**

$$1, -1$$

Typing colon (**:**) after the command line suppress the output display.

> **solve(x^2=1,x):**

You can assign the output from a command to a variable name.

> **soln:=solve(x^2=1,x);**

$$soln := 1, -1$$

To access an individual solution, use a list-element selector after the name of the variable to which you assigned the output.

> **soln[1];**

$$1$$

> **soln[2];**

$$-1$$

Alternatively, you can assign the individual solutions to variable names up front.

> **firstsoln:=solve(x^2=1,x)[1];**

$$firstsoln := 1$$

> **secondsoln:=solve(x^2=1,x)[2];**

$$secondsoln := -1$$

This latter approach is not as efficient as the one preceding it because it requires solving the equation twice, whereas the first approach only solves it once.

Some commands give a function as output. For example, the following command solves the differential equation $\frac{d}{dx}y(x) = \sin(x)$ with the initial condition $y(0) = 0$.

> **restart:**
unassign('x,y'):
soln:=dsolve({diff(y(x),x)=sin(x),y(0)=0},y(x));

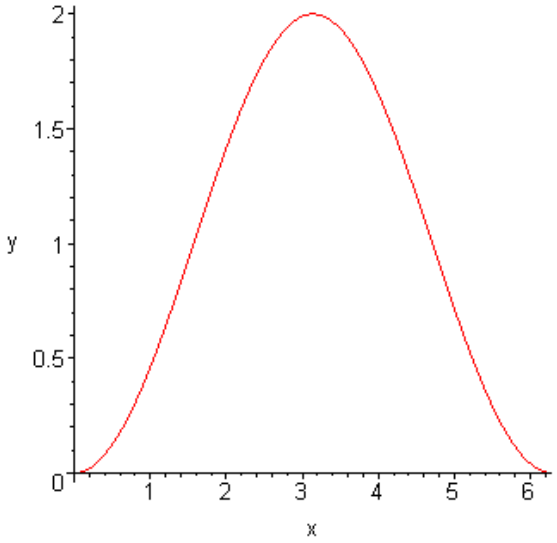
$$soln := y(x) = -\cos(x) + 1$$

To use the function $y(x)$ in other commands, such as **plot( )** , for example, we form a *Maple* function from the output of the **dsolve( )** command as follows.

> **y:=unapply(rhs(soln),x);**

$$y := x \rightarrow -\cos(x) + 1$$

> **plot(y(x),x=0..2*Pi,labels=['x','y']);**


2
1.5
y
1
0.5
0
1
2
3
4
5
6
x

>