5.1 Computational Toolbox—Tools of the Trade: *Maple* Tutorial 2

File: *MapleTutorial2.mw*

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College

Introduction

The prerequisite to this tutorial is " *Maple* Tutorial 1." Tutorial 2 prepares you to use *Maple* to complete projects for this and subsequent chapters. The tutorial introduces the following functions and concepts: lists, plotting points, plotting lines connecting points, comments, and appending to lists. The module also gives examples along with Quick Review Questions for you to do with *Maple*. Execute all input cells to view the results of the examples.

Lists

Many *Maple* applications involve **lists**, and a number of built-in functions perform operations on lists. **Brackets, [ ]**, enclose the values in a list, such as follows:

```
> numList := [13, 36, 92];
```

We also employ a list to store an ordered pair, such as the following representation of the point (-3, 46):

```
> orderedPair := [-3, 46];
```

Quick Review Question 1 Do anything that is asked of you in cells that look like this one, marked as a Quick Review Question in boldface. Because such cells are text cells and not input cells, do not type in these cells. Instead, if a greater than prompt does not appear in an otherwise empty execution group below, from the *Insert* menu, *Execution Group* submenu, select *After Cursor*. Alternatively, use the shortcut indicated on that menu or click the icon ([>) to "Insert executable *Maple* input after the current paragraph."

For this question, assign to *ptLst* a list representing the following ordered pairs (points): (-3, 6), (5, 2), and (1, 12).

The elements of a list have a numeric order starting with 1, and we can refer to a particular element by using the name of the list and brackets, [ ], surrounding that number. For example, the second element of *numList* above is as follows:

```
> numList[2];
```

Besides using a number, such as 2, we can employ a variable, such as *i*, that has a value, so that *numList*[*i*] refers to the *i*-th element of the list. Consequently, a *for* loop can process the elements of a list individually by having the varying loop index specify the list element.

Quick Review Question 2 Using print and a*for* loop, print on separate lines the points of *ptLst* from Quick Review Question 1.

Frequently, a list occurs within a list, such as the following list that represents a matrix with two rows, each in brackets, and four columns:

```
> mat := [ [45, 99, 203, -29], [775, 31, -582, 62] ];
```

With *mat* having two elements, the rows, we reference the first row, [45, 99, 203, -29], as follows:

```
> mat[1];
```

We obtain first row's third column element using a row and column values in brackets, as follows:

```
> mat[1][3];
```

Alternatively, we employ the following notation that separates the indices with a comma:

```
> mat[1, 3];
```

Quick Review Question 3 Write commands using *ptLst* and brackets to return the following parts of *ptLst* from Quick Review Question 1:

a. The third point, (1, 12)

b. The first coordinate of the second point, 5. Use two pairs of brackets.

c. The first coordinate of the second point, 5. Use one pair of brackets.

d. The second coordinate of the first point, 6. Use two pairs of brackets.

e. The second coordinate of the first point, 6. Use one pair of brackets.

Graphing Points

To plot points in a list, we use ***listplot***, which the package ***plots*** defines. *Maple* contains a number of useful packages that the software loads only as requested. Using the ***with*** function, we load a needed package once in a worksheet. With the *plots* package loaded as the following command accomplishes, we can call the *listplot* function:

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

In the example below, two statements appear in one execution group, the assignment of a list to *pts* and *listplot*; and output for each command occurs after the input cell. After indicating the function(s) or data to graph, we list options and their values in the form ***option = value*** in calls to plotting functions, such as *plot* and *listplot*. The *listplot* command below contains two such option-value pairs. The function displays the points with the appropriate axes labels by employing the option ***labels***, which is also available for *plot*. The ***style*** option with value ***point*** indicates that the display should contain distinct points instead of lines connecting the points. The appearance of a "point" depends on the output device and might be, for example, a circle, diamond, or cross.

```
> pts := [[-3, 2], [2, 2], [-1, -1], [4, 3], [0, 0] ];
  listplot(pts, labels = [" x ", " y "], style = point);
```

We can still use *listplot* without loading the entire package by using the package name with the function name in brackets, such as ***plots*[*listplot*]** in the following command:

```
> plots[listplot](pts, labels = [" x ", " y "], style = point);
```

Another option that is important to maintain consistency when generating a number of plots for an animation is ***view***, which designates the viewing area for the graph. The option-value pair ***view* = [*xmin*..*xmax*, *ymin*..*ymax*]** indicates that for the display, the minimum and maximum values in the horizontal direction are *xmin* and *xmax*, respectively; and those for the vertical direction are *ymin* and *ymax*, respectively. The following command results in a plot from -3 to 2 in the *x* direction and from -1 to 2 in the *y* direction, causing the omission of one point:

```
> plots[listplot](pts, labels = [" x ", " y "],
      style = point,
   view = [-3..2, -1..2] );
```

Quick Review Question 4 Graph the points in list *ptLst* from Quick Review Question 1. Have

labeled axes.

In the command below, we make the points bigger and assign the graph to a variable, *lp*, for later reuse. To have larger points, we employ the option ***symbolsize*** with value (here, 15) being the number of pixels (dots) across. The default symbol size is 10.

```
> listplot(pts,
      labels = [" x ", " y "] , style = point,
      symbolsize = 15 );
```

Quick Review Question 5 Copy to a new execution group the answer of Quick Review Question 4 to plot the list of points stored in variable *ptLst*. Adjust the command to have the points appear larger.

Lines Connecting Points

Sometimes, it is helpful to visualize the path of an entity, such as an animal or a molecule, whose movement we are simulating. To generate the path, we employ the default value for the option *style* or explicitly indicate ***style = line***. The subsequent graph displays line segments joining pairs of adjacent points.

For example, suppose [-1, 0, -1, 0, 1, 2, 3, 2, 1, 0] is a list (*ylst*) of *y* values, where each element is randomly one more or less than the previous element. Considering each *y* value to occur at sequential ticks of the clock, we can draw a line graph to display the trend of the *y* values over time. Just plotting *ylst* causes the first coordinates of the points to be 1, 2, ..., 10 and the corresponding second coordinates to be from *ylst*. The segment to display the trend of *y* over time follows:

```
> ylst := [-1,0,-1,0,1,2,3,2,1,0]:
  listplot(ylst, labels = ["t", "y"],  style = line );
```

Quick Review Question 6 Plot the list of points stored in variable *ptLst* from Quick Review Question 1 of the "Lists" section. Label the axes, and connect the points with line segments.

Comments

Any program, regardless of the language, should have ample comments to explain the code. It is amazingly easy forget what was done only a few minutes earlier. It takes far less time to enter the comments as we type the program than to figure out the code later. A descriptive software engineering phrase is "Write once, read many times."

We have been using *Maple* worksheet execution groups with styles, such as *Text*, to record comments. However, for longer segments of code, internal comments are also helpful. In *Maple*, comments appear after #, and the computer ignores text starting with a pound sign for the rest of the line. Examples of comments follow:

```
> # A comment can appear on a line by itself; and if it
  # continues to the next line, we must use another pound sign.
  # Also, a can document code on a line, such as follows:
  lst := []:    # initialize list to empty
```

Quick Review Question 7 Write a statement to assign 80 to the variable *v*, and in a comment on the same line indicate that the variable represents "velocity in km/hr."

Appending to a List

In simulations with *Maple*, we frequently build a list, one element at a time. To do so, we employ list notation with brackets and the Maple function ***op***, which returns the operands or sequence of elements in a

list. In the following segment, we assign a list to the variable *lst* and then display the sequence of elements using *op:*

```
> lst := [1, 2, 3, 4];
  op(lst);
```

To obtain an appended list, we incorporate the sequence of elements from the original list, *op*(*expr*), and the additional element, *elem*, in brackets, as follows:

```
[op(expr), elem]
```

The segment below returns the value of the list *lst* with a new element, 5, on the end. As the output of *lst* on the last line indicates, however, the operation returns the appended list but does not change the original value of *lst*.

```
> [op(lst),5] ;
  lst;
```

To have *lst* be the appended list, we assign the appended list to *lst*, as follows:

```
> lst := [op(lst),5] ;
  lst;
```

The appended element can be a list itself. In the following example, *pts* originally represents a list of two points. After execution, *pts* contains a third point, the origin.

```
> pts := [[1, 5], [-2, 7]];
  pts := [op(pts), [0, 0]] ;
```

In a programming language, such as C, C++, or Java, when using a loop to count, we must initialize the counter to be zero before the loop. Similarly, we must initialize a list before performing the append operation. When building a list from scratch, that initial value is usually an **empty list, [ ]**. The segment below defines a function $g(x) = 3\sqrt{x}$ using the *Expression* palette or ***sqrt(x)*** for the **square root** of *x* and then stores *g*(*i*) for the positive integers *i* less than 10 in the list *gLst*, which is originally empty. Finally, we have *Maple* return the value of *gLst*.

```
> g := x-> 3*sqrt(x):
  gLst := []:
  for i from 1 to 10 do
     gLst := [op(gLst), g(i)] ;
  end do:
  gLst;
```

Quick Review Question 8 Using the comments in the cell below as a guide, write a *Maple* segment to generate a list, *lst*, of 30 points. Initialize the list to be empty and *y* to be 1. Inside the body of a for loop with an integer index *i* that goes from 0 through 29, make *x* be 0.25 times *i*, make *y* be 1.2 times its previous value, and append the ordered pair of values to the list. After creation of the list of points, plot the points with line segments joining adjacent points and labeled axes.

```
> # initialize list lst to be empty
  # initialize y to be 1
  # for i going from 0 through 29 in a loop do the following:
  #      assign to x the expression 0.25 times i
  #      assign to y the expression 1.2 times previous value of y
  #      assign to lst the list with ordered pair of x and y appended
```

```
    # plot the points
```