8.1 Computational Toolbox—Tools of the Trade: *Mathematica* Tutorial 3

File: *MathematicaTutorial3.nb*

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College
© 2006 by Princeton University Press

Introduction

The prerequisites to this tutorial are "*Mathematica* Tutorial 1" and "*Mathematica* Tutorial 2." Tutorial 3 prepares you to use *Mathematica* to complete projects for this chapter and is useful for material in subsequent chapters. The tutorial introduces the following functions and concepts: list functions, such as *Table*, *TableForm*, and *Transpose*; additional graphics options, such as *AspectRatio*, *PlotRange*, *RGBColor*, *Thickness*, and *Dashing*; *Show*; *Fit*; rules with /. and ->; reading from a file; and logarithm. The module also gives examples along with Quick Review Questions for you to do with *Mathematica*. Execute all input cells to view the results of the examples.

- **Do anything that is asked in Quick Review Questions, which appear in cells that look like this one. Because such cells are text cells and not input cells, do not type in these cells. Instead, move the cursor down until it changes from being vertical to being a horizontal-I bar. Click and start typing; a new input cell will form.**

Tables

Lists are essential to *Mathematica*, and we can employ the ***Table*** command to generate a list from repeated evaluation of an expression. The following form of the command generates a table of *imax* number of elements that are evaluations of *expr:*

```
Table[expr, {imax}]
```

For example, the following command generates a list consisting of five zeros:

```
Table[0, {5}]
```

Frequently, an index that varies from 1 to the number of iterations (*imax*) is useful in element generation, as in the following form of the *Table* command:

```
Table[expr, {i, imax}]
```

For example, the following command employs the index *x* in the expression to generate a list of successive positive powers of 2:

```
tbl = Table[2^x, {x, 6}]
```

The resulting value of *tbl* is {2, 4, 8, 16, 32, 64}.

■ Quick Review Question 1 In an example of *Do* and *Append* from "*Mathematica* Tutorial 2," we defined $g(x) = 3\sqrt{x}$, initialized the list *gLst* to be empty, and then stored $g(i)$ for the positive integers *i* less than 10 in *gLst*. Give a *Table* command to create *gLst* without defining *g*.

To have an initial index value (*imin*) other than 1, we can employ the following form of the *Table* command:

```
Table[expr, {i, imin, imax}]
```

Another form has a step size (*step*) after the maximum index:

```
Table[expr, {i, imin, imax, step}]
```

For example, we generate a table, *tbl*, of values from 1 to 16 with a step size of 3, namely the list {1, 4, 7, 10, 13, 16}, with the following assignment involving *Table*.

```
tbl = Table[i, {i, 1, 16, 3}]
```

■ Quick Review Question 2 In an example of *Do* from "*Mathematica* Tutorial 1," we initialized *dist* to be 0 and then seven times changed its value with the assignment *dist* = *dist* + 2.25 until *dist* became 15.75. Write a command to generate a list, *distLst*, containing these values from 0 through 15.75. Do not use *dist*.

We can use *Table* to form a list of ordered pairs, too. For example, after defining $f(x) = x^2$, we generate a table of pairs of values of *x* and *f*[*x*] in the following segment:

```
Clear[f];
f[x_] := x^2
fLst = Table[{x, f[x]}, {x, 0, 2.5, 0.5}]
```

The result is a list of ordered pairs, where the list and each ordered pair are in braces { } and the first coordinates step with an interval of 0.5.

Quick Review Question 3 In Quick Review Question 8 of "*Mathematica* Tutorial 2," you generated the following list (*lst*) of 30 points: {{0, 1.2}, {0.25, 1.44}, {0.5, 1.728}, …, {7.25, 237.376}}. The first coordinate was 0.25 times an index *i* that varied from 0 to 29. The second coordinate, which was initially 1, was 1.2 its previous value. Instead of using *Do* and *Append* as in the previous tutorial, first initialize the second coordinate (*y*) to be 1 and then create the list (*lst*) by using *Table*.

The following form for *Table* generates a nested list that is a two dimensional array, where for each value of *i* from *imin* to *imax*, *j* varies from *jmin* to *jmax*:

```
Table[expr, {i, imin, imax}, {j, jmin, jmax}]
```

For example, the next command produces a list that is a 4-by-7 rectangular array:

```
prod = Table[x * y, {x, 1, 4}, {y, 3, 9}]
```

At the highest level, *prod* contains four lists, one for each value of *x* from 1 through 4. Each of the four lists contains seven elements, one for each value of *y* from 3 through 9. At this lowest level, each element is the product of the corresponding values of *x* and *y*. Thus, the value of *prod* is a partial multiplication table.

■ **Quick Review Question 4 Write a command to generate a table of values of x^y, where for each *x* value of 2.0, 2.1, 2.2, 2.3, 2.4, and 2.5, *y* takes on values 1, 3, and 5. Thus, the output is {{2., 8., 32.}, {2.1, 9.261, 40.841}, {2.2, 10.648, 51.5363}, {2.3, 12.167, 64.3634}, {2.4, 13.824, 79.6262}, {2.5, 15.625, 97.6563}}.**

Displaying Tables

We might find the information in list *flst* = {{0, 0}, {0.5, 0.25}, {1., 1.}, {1.5, 2.25}, {2., 4.}, {2.5, 6.25}} above easier to understand if the format is in two columns instead of as one long list. The command ***TableForm*** displays an argument list of lists as a rectangular array, or matrix, of elements. The following command produces the list in the two column output:

```
TableForm[fLst]
```

Thus, we can think of this six-element list *flst* of ordered pairs as a rectangular array of six rows and two columns.

TableForm is a formatting command. Thus, we should not assign the output of *TableForm* to a variable that we expect to hold a list.

■ **Quick Review Question 5 Write commands to display the following lists as rectangular arrays:**

a. *lst* of 30 points from Quick Review Question 3 in the *Table* section

■ **b. *prod* from the end of the *Table* section**

■ **c. The table from Quick Review Question 4. Copy the command that is the answer for that question to a new cell and have the argument for *TableForm* be the *Table* command, not a variable.**

Transpose

In the module on "Empirical Models," we deal with some examples where the first and second coordinates of data are in separate lists that need to be incorporated into one list of ordered pairs. For example, suppose for an hour a scientist measures amounts (in milligrams) of residues from a chemical reaction every 12 minutes, or 0.2 hours. The following command assigns to *tlst* the list of times, {0, 0.2, 0.4, 0.6, 0.8, 1.}:

```
tlst = Table[k, {k, 0, 1, 0.2}]
```

The following *rlst* is a list of residue measurements:

```
rlst = {0.00, 0.05, 0.16, 0.23, 0.55, 1.00};
```

The following expression produces on output line a list of two lists, each with 6 elements:

```
{tlst, rlst}
```

Using *TableForm*, we display this list as a rectangular array of two rows and six columns:

```
TableForm[{tlst, rlst}]
```

The first row consists of the times, while the second stores the measured residues. To generate a list of ordered pairs of corresponding times and residues, we take the **transpose** of the list of lists {*tlst*, *rlst*}. Without changing the original list, the transpose function in *Mathematica*, ***Transpose***, returns a list with the rows and columns swapped. Thus, the following statement assigns the list of ordered pairs to *trans:*

```
trans = Transpose[{tlst, rlst}]
```

The formatting command *TableForm* shows the transposed list in six rows and two columns:

```
TableForm[trans]
```

Definition. The **transpose** of a matrix (rectangular array) is a matrix with the rows and columns exchanged from the original matrix.

■ **Quick Review Question 6 Write a statement to generate a list *xLst* of positive integers from 1**

through 9. From Quick Review Question 1, *gLst* stores the corresponding values of *g*(*x*). Write commands to assign to *pairsLst* the list of ordered pairs and to display *pairsLst* as a rectangular array.

Additional Graphics Options

In "*Mathematica* Tutorial 1," we covered basic graphing of a function with *Plot*. In this tutorial, we discuss some additional features that are helpful in producing meaningful scientific visualizations.

The **aspect ratio** of a graphics is its width divided by its height. For example, if a graphics is 6 cm wide and 3 cm high, then its aspect ratio is 6 / 3 = 2; and if the dimensions are reversed, the aspect ratio is 3/ 6 = 1/2. *Mathematica* decides the aspect ratio that it considers best for each graph. To override the default, we can employ the ***AspectRatio*** option with **->** and the number for the aspect ratio. For graphs where the graphing interval is the same on the horizontal and vertical axes, *AspectRatio*->1 causes a unit length on one axis to be the same as on the other. Such specifications can be helpful for visualization of a graphics without distortion.

Definition The **aspect ratio** of a graphics is its width divided by its height.

Although we specify the part of the domain to graph, *Mathematica* decides on an appropriate part of the range unless we override the software's decision with another option, ***PlotRange***. Designation of the option has two forms. ***PlotRange->{c, d}*** indicates that *Mathematica* should display the vertical part of the graph from height *c* to height *d*; and ***PlotRange->{{a, b}, {c, d}}*** specifies horizontal portion as being from *a* to *b* as well as the vertical portion from *c* to *d*. The plot of $f(x) = x^2$ with domain from -2 to 2, range (*PlotRange*) [0, 4], the same proportions (*AspectRatio*) on both axes, and the axes labeled (*AxesLabel*) as follows results in the graph shown:

```
Clear[f];
f[x_] := x^2;

Plot[f[x], {x, -2, 2},
  PlotRange -> {0, 4},
  AspectRatio -> 1,
 AxesLabel -> {"x", "y"}
]
```

■ **Quick Review Question 7**

a. Graph $y = \ln(x^2)$ from 0.1 to 6 without any options except *AxesLabel*.

■ **b. Copy the command from Part a to a new cell. In that cell, specify that the graph should be shown from 0 to 6 on the *x*-axis and from –2, to 4 on the *y*-axis.**

■ **c. Copy the command from Part b to a new cell. In that cell, specify that the graph should have an aspect ratio of 1. Observe the impact of each option on the graph.**

Frequently, we wish to show more than one plot in the same graphics for comparisons. To do so, we group the functions in braces in the *Plot* command. The following segment defines *g* and plots *f* from above and *g* in the same graphics:

```
Clear[g]
g[x_] := f[x] + 1;
Plot[{f[x], g[x]}, {x, -3, 3}]
```

In such situations, to differentiate clearly between the two graphs, we can show them with different colors for display on a monitor or color printer, or we can employ assorted thicknesses or dashing for a black-and-white printer. With the option ***PlotStyle*** we can indicate a color for each function's graph using ***RGBColor***. The arguments of *RGBColor* are three floating point numbers between 0 and 1 that indicate the amount of red, green, and blue in the composite color. As a result of the segment below, the first function, *f*, appears in red because its associated *RGBColor* indicates full red (1) and no (0) green or blue. The second function, *g*, goes with the second *RGBColor*, which indicates blue. As with the functions, the two *RGBColor* options are grouped in braces.

```
Plot[{f[x], g[x]}, {x, -3, 3},
 PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 0, 1]}
]
```

■ **Quick Review Question 8 Go to the cell above and include the function $3x + 2$ in green. Make the changes on the input; do not retype the cell.**

To distinguish grays on a black-and-white printer, instead of the *PlotStyle* of *RGBColor*, we can designate the thickness as a fraction of the width of the graph, such as ***Thickness*[0.01]**. The graphics displays the result of the following command with the function *g* thicker than *f*.

```
Plot[{f[x], g[x]}, {x, -3, 3},
  PlotStyle -> {Thickness[0.01], Thickness [0.02]}
]
```

■ **Quick Review Question 9 Copy the input cell from Quick Review Question 8 and paste it below. On the pasted cell, have the graphs display in black-and-white with progressively thicker lines.**

Alternatively, we can indicate a plot style of ***Dashing***[{*r1*, *r2*, ... }] that indicates the graph is to be dashed with segments that are of lengths *r1*, *r2*, ..., where each *ri* is a fraction of the width of the graph and where the segments repeat cyclically. Thus, *Dashing*[{0.01, 0.04}] indicates to alternate line segments that are 0.01 = 1% the width of the graphics with spaces that are 4% the width; while *Dashing*[{0.02}] has alternating line segments and spaces, each 2% the width of the graphics. To have more than one plot style for a function's graph, we group the options in braces. The example below, has no extra specifications for the graph of *f* so that it appears as a solid line and indicates that the graph of *g* should be a thicker, dashed line. The options for each function appear in a list, and *f*'s option list is empty, { }.

```
Plot[{f[x], g[x]}, {x, -3, 3},
   PlotStyle -> {{}, {Thickness[0.01], Dashing[{0.02}]}}]
```

■ **Quick Review Question 10 Copy the input cell from Quick Review Question 9 and paste it below. On the pasted cell, have the graphs display in black-and-white. Display *f* having a thicker line (1.5% the graphics width) that is dashed with alternating lengths of 1%, 2%, and 3% of the graphics width; display *g* with no extra options; and display 3*x* + 2 dashed with each dash and space being 4% the graphics width.**

Showing Several Graphics Together

Several times in Module 8.3 on "Empirical Models," we need to display a *ListPlot* of data points and a *Plot* of a function in the same graphics. To do so, we generate each graphics, saving the results in variables, and then display the combination using the ***Show*** command with the variables as arguments. For example, the following segment defines a list of points (*pts*), displays the points with *ListPlot*, and stores the graphics in variable *lp:*

```
pts = { {-3, 20}, {2, 20}, {-1, -10}, {4, 30}, {0, 0} };
lp = ListPlot [pts, AxesLabel -> {"x", "y"} ,
  PlotStyle → {PointSize[0.03]}];
```

The next statement, plots a cubic polynomial, storing the graphics in variable *plt:*

```
plt = Plot[-3 + 9 x + 3 x2 - x3, {x, -3, 5}]
```

With the *Show* command below, we display the data points and cubic together. The result illustrates that the cubic somewhat captures the trend of the data.

```
Show[lp, plt]
```

Line Graphics Element

Line is a two-dimensional primitive graphics elements. We can generate this element by having a list of the beginning and ending points as the argument to the *Line* function, as in the following format:

```
Line[ { {x1, y1}, {x2, y2} } ]
```

For a two-dimensional graphical image, we can employ ***Graphics*** with the following format, where the argument can be a *Line* primitive:

```
Graphics[primitives]
```

This command generates the image but does not draw the line. To complete the process of displaying the image, we employ *Show*. For example, to display a line from point (1.309, 2.138) to point (1.68, 5.66), we employ the following command:

```
Show[Graphics[Line[{{1.309, 2.138}, {1.68, 5.66}}]]]
```

We can specify various directives for the graphics, such as color, line thickness, and dashing. To do so, we place the graphics directive and graphics primitive in a list with the directive appearing first. Thus, the following command with braces surrounding *Thickness* and *Line* displays a thick line:

```
Show[Graphics[
 { Thickness[0.02], Line[{{1.309, 2.138}, {1.68, 5.66}}] }
]]
```

■ **Quick Review Question 11 Replace each xxxxxx to complete the command below to display a line through points (0, 0) and (1, 3). Have graphics directives to designate a dashed line with segments of length 3% and *RGBColor* levels of 0.4 for red, green, and blue:**

```
xxxxxx[ xxxxxx[
  xxxxxx  xxxxxx[xxxxxx 0.03 xxxxxx],
  xxxxxx[0.4, 0.4, 0.4],
  xxxxxx[ xxxxxx {0, 0}, {1, 3} xxxxxx]
   xxxxxx
 ]]
```

Fit

In Module 8.3 on "Empirical Models," we investigate discovering functions that capture the trend of data. To do so, we employ the *Mathematica* function ***Fit***. One form of the call to *Fit* is as follows:

Fit[*listOfPoints*, {f_1, f_2, ..., f_n}, *var*]

Mathematica finds coefficients (a_1, a_2, …, a_n) for f_1, f_2, ...,f_n and returns the function $a_1 f_1 + a_2 f_2 + \ldots + a_n f_n$ in independent variable *var* that "best" fits the data in a list of points, *listOfPoints*. For example, the following command fits a cubic polynomial to the set *pts* = { {-3, 20}, {2, 20}, {-1, -10}, {4, 30}, {0, 0} } above:

```
fitCubic = Fit[pts, {1, x, x^2, x^3}, x]
```

After plotting *fitCubic*, we show together the graphics of the points and the cubic:

```
Show[lp, fitCubic]
```

- **Quick Review Question 12**

  a. Plot the list of points stored in variable *pts2* below and save the graphics in variable *pts2Plot*. Have the points be green and a slightly larger size than those in the previous cell.

```
pts2 = { {0.4, 0.16}, {0.6, 0.23}, {0.8, 0.55}, {1., 1.}};
```

- **b. Clear *x*'s value. Give the command to have *curveFit* store the equation of the line to fit "best" *pts2*. Recall that the general equation of a line is $mx + b = m(x) + b(1)$.**
- **c. Plot the line from Part b, *curveFit*, from 0.4 to 1.0 and store the graphics in variable *curvePlot*.**
- **d. Show the graphics for *pts2Plot* and *curvePlot* together.**
- **e. Copy the command from Part b to a new cell. Edit the copied cell to fit a quadratic, $a_2x^2 + a_1x + a_0(1)$, to the data.**
- **f. Copy the command from Part c to a new cell and re-execute to plot the quadratic.**
- **g. Copy the command from Part d to a new cell and re-execute to show the data and quadratic in the same graphics.**

Fit gives the least-squares fit to a list of data. In Module 8.3 on "Empirical Models," we explain least-squares fit and its utility in working with data.

Rules

Mathematica transformation rules enable us to change expressions from one form to another. A **rule** using **->**, such as

lhs **->** *rhs*

transforms *lhs* into *rhs*. The **rule operator /.** with the following form indicates to apply a rule or a list of rules (*rules*) to an expression (*expr*):

expr **/.** *rules*

More specifically, the following command replaces every occurrence of *z* with *value* in the expression *expr*:

expr **/.** *z* **->** *value*

For example, suppose *fitData* is 0.0225362 + 0.75118 *z*. To replace *z* with *x*3.9 in the expression *fitData* and to define a function *f* equal to the result, we use the following command:

```
f[x_] := fitData /. z -> x^3.9
```

Thus, we are defining $f(x)$ as $0.0225362 + 0.75118\, x^{3.9}$.

We can make several substitutions (*xValue* for *x*, *yValue* for *y*, etc.) by forming a list of the rules after /., such as follows:

expr **/. {***x* **->** *xvalue*, *y* **->** *yvalue*, … **}**

■ **Quick Review Question 13 Clear values of *x* and *y*. Add rule replacement to the following line to return an expression replacing *u* with 2^x and *v* with ln(*y*). Remember that the natural logarithm function in *Mathematica* is *Log*.**

```
u + 7 v
```

Reading from a File

Files can store huge amounts of data and simplify input. Links to data files for various projects appear on the textbook's website. For example, Module 8.3 on "Empirical Models" uses the file *DanWoodEM.dat*, which follows, and several much larger files:

```
1.309   2.138
1.471   3.421
1.490   3.597
1.565   4.340
1.611   4.882
1.680   5.660
```

The *Mathematica* function ***ReadFile*** can read a file of numbers or data of other types into a table. One form of the command is as follows:

ReadList["*FileName*", *type*]

The file name, which appears in quotation marks, must be in the path of where *Mathematica* looks, or we must specify the full path name of the file. A common type is ***Number***, which is the type of an integer or real number. To have pairs of numbers grouped into two-element lists, we give the template {*Number*, *Number*} as the type. For example, the file *DanWoodEM.dat* consists of two columns of data for *x*- and *y*-values. To read the data into a list of points with the coordinates of a point in braces and to store the resulting table in the variable *pts*, we employ the following command:

```
pts = ReadList["DanWoodEM.dat", {Number, Number}]
```

■ **Quick Review Question 14 Replace each xxxxxx to complete the command to read the data of *DanWoodEM.dat* into a list, *lst*, of 12 numbers:**

```
lst = xxxxxx DanWoodEM.dat xxxxxx
```

Logarithms

See the section on "Logarithmic Functions" from Module 8.2 on "Function Tutorial" for a discussion of logarithms.

In *Mathematica*, the common logarithm of n is ***Log*[10, *n*]**. Thus, the following call to the function returns 3:

```
Log[10, 1000]
```

The natural logarithm of n is ***Log*[*n*]** in *Mathematica*. Thus, *Log*[50.0] returns 3.91202 because $e^{3.91202}$ is 50.0. Bases other than e or 10 are permissible as long as the base is greater than 1. In general, *Mathematica*'s ***Log*[*b*, *n*]** is $\log_b n$.

- **Quick Review Question 15**

  a. Plot e^x and $\ln x$ on the same graph.

- **b. Evaluate $\log_{10} 7$ as a floating point number.**