Getting Started with Matlab

1. Start Matlab by double-clicking the Matlab icon.
2. Use the Matlab toolbar to set the appropriate working directory. Use Z: or a subfolder of Z: if using a mathlab computer. (When Matlab launches, the directory is on the local hard drive. Your files are in a particular directory on the department file server, accessible to you as though they were on a separate drive called Z. If the directory you want isn't listed, you can find it by pressing the button just to the right of the directory list labeled "…".)
3. Use the Matlab toolbar to open any m files that you need to edit.
4. For convenience, shrink the main window to the bottom right quarter of the screen and the edit window to the left half of the screen. The main window is divided into two portions, and you can change the widths of these. The working area on the right is all you really need to see, so move the divider far to the left.
5. Matlab statements can be entered in the main window and are executed when you hit Enter. Values assigned to names by these statements are retained for future instructions.
6. If you want to repeat a statement or run a statement again with a minor change, you can use the up arrow key to recall earlier statements. Each time you press the key, you scroll back farther up the list of statements run in that session.
7. Ending an instruction with a semicolon suppresses the output of the result. Generally it is best to end instructions with semicolons unless you particularly want to see the result.
8. Matlab has many built-in functions that solve problems; for example, there is a function that finds solutions of algebraic equations. You can also define your own "built-in" functions that do computations or prepare graphs. When Matlab executes a user-defined function, it outputs any statements in the function file that aren't ended with semicolons. This allows you to use a function file to generate a graph.

Lists in Matlab

1. **`A=[10 0.1];`** defines a list of two numbers. These can be accessed separately as **`A(1)`** and **`A(2)`**. It is convenient to use a single list when you want to pass model parameters to a function.
2. **`t=[0:l0];`** defines a list of consecutive integers, starting with 0 and ending with l0. These can be accessed individually. Thus, $t(2)=1$ and $t(11)=10$.
3. **`x=[0.1:0.05:1];`** defines a list of real numbers, starting with 0.1, incremented by 0.05, and ending with 1.
4. You can create lists by applying arithmetic operations to the elements of an existing list. Thus, **`y=exp(t);`** creates a list in which each y_k is computed as $y_k=\exp(t_k)$. Arithmetic done on lists sometimes requires special symbols. If x and y are lists and you want a list with elements $x_k y_k$, you need **`x.*y`** rather than x*y. The extra "**`.`**" is needed for division and exponentiation also. (Arithmetic on lists makes Matlab really useful, but it is tricky.)

Plots in Matlab

1. Plots are created from lists of the ***x*** and ***y*** values of the points to be plotted. These lists have to have the same number of entries. If you have data points of the form (***t***,***N***) stored in lists ***t*** and ***N***, the command **`plot(t,N)`** will produce a solid blue curve joining the specified points. Other plot styles are specified in the form **`plot(t,N,'stylespecs'),`** where stylespecs is a strong of characters that specify the style or color. Common stylespecs are **`b`**, **`g`**, **`k`**, and **`r`**, for blue, green, black, and red; **`-`**, **`--`**, **`:`**, and **`-:`** for solid, dashed, dotted, and dash-dot; and **`+`**, **`o`**, **`x`**, **`s`**, and **`d`** to mark the points with + signs, circles, "**x**", a square, and a diamond. The defaults are blue, solid, and unmarked. If you want both a solid curve and points marked with **x**, the simplest thing to do is use the plot command twice, once to get the points and once to get the curve; this allows you to use different colors for the points and the curves.
2. **`Plot`** can either replace the previous graph (the default) or add to the previous graph. If you want to add to the graph, such as when you want a red curve with blue points, use the command **`hold on`** after the first plot. This way, the first plot replaces any existing plot, but subsequent plots are added to it. If at some point you want to start over again, the command **`hold off`** will cause the next and subsequent plots to replace the previous graph.
3. Plots can be modified using a variety of commands. **`xlim([0 T])`** and **`xlabel('\it{t}','FontSize',12)`** are examples. The first overrides Matlab's default routine for determining axis limits and the second labels the ***x*** axis with an italic ***t***. Labels on the ***y*** axis come out sideways unless you specify vertical orientation. **`ylabel('\it{N}','FontSize',12,'Rotation',0)`** labels the y axis with an italic ***N*** oriented vertically.

Functions in Matlab

Here is a simple example of a Matlab function. It could be used as a separate file called seqfnc.m or it could be used within another function file. The purpose of the function is to calculate N_{t+1} for the model $N_{t+1}=N_t+RN_t(1-N_t/K)$, where ***R*** and ***K*** are parameters. The function requires two arguments, ***N*** and ***A***, and calculates a value ***y***; ***N*** is just N_t and ***A*** is a list of the parameters ***R*** and ***K***. The advantage of combining the parameters into a list is that fewer modifications are needed if the function is changed to one with a different number of parameters.

```
function y=seqfnc(N,A)
R = A(1);
K = A(2);
y = N+R*N*(1-N/K);
```

The command **`N(t+1)=seqfnc(N(t),[2 10])`**, inside a loop that increments ***t***, will compute the next ***N*** value using the parameters ***R***=2 and ***K***=10.