Matlab tutorial

by Nick Aschenbach

In this tutorial you will learn a few of the basic functions of Matlab. First we will start working with basic mathematical functions, setting variables, and generating time series. This is followed by a short section on vectors. We will move on to graphing simple trigonometric functions to demonstrate how to use the plot function. There is a brief overview of how to generate scripts in Matlab called 'M-files' and finally we move on to how to code up ordinary differential equations. This tutorial assumes that you have some basic understanding of matrices and calculus.

Basics

First of all open up Matlab to make sure the program is installed and is working. Note that sometimes Matlab may not start if your default printer is an HP. If this is the case, then change or delete your current printer default setting to another printer.

Let's start with some simple math. In the Matlab command window type in the following expression:

```
2 + 2
```

Matlab returns

```
ans =
        4
```

This means that Matlab has evaluated the expression '2 + 2' and set the answer equal to a variable called 'ans'. It is possible to see the value of this variable by typing in:

```
ans
```

Matlab returns

```
ans =
        4
```

Now type in another expression, which ends with a semicolon ';' at the end of the line:

```
2 + 3;
```

Note that Matlab does not return anything, but enter in the following line:

```
ans
```

Matlab returns

```
ans =
        5
```

So you can see the calculation has been made. Ending a line with a ‘;’ is useful if it is desirable to suppress the output from a function, which will come in handy later in the tutorial.

It is very simple to define new variables in Matlab. Type in the following expression:

```
a = 3 + 5
```

Matlab returns

```
a =
        8
```

It is possible to define a new variable that incorporates another variable that has already set:

```
b = a + 2
```

Matlab returns

```
b =
        10
```

To see what variables are currently in memory there are few useful commands to use. First type in:

```
who
```

Matlab returns

```
Your variables are:
a                ans              b
```

To see a list of variables, memory allocation, and class of the variables in memory type in:

```
whos
```

Matlab returns:

Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	1x1	8	double array

Type in:

clear

Matlab returns nothing, but if ‘who’ or ‘whos’ are typed in, the list of variables is empty. This is useful if when starting a new calculation and it is desirable to clean up the variables stored in memory.

Vectors

Vectors are simple to enter into Matlab. To enter in a one dimensional vector type in the following text:

a = [1 2 3 4 5 6 7 8 9]

Matlab should return

a =
1 2 3 4 5 6 7 8 9

Alternately use a simple method to generate the same list:

a = 1 : 9

It is also possible to generate a vector that does not just step by one using this feature. This is often desirable when generating a time series:

0 : 2 : 20

Matlab returns:

ans =
0 2 4 6 8 10 12 14 16 18 20

Manipulating vectors using simple math is as easy as changing other variables. Type in the following expression:

b = a + 2

Matlab returns:

```
b =
     3     4     5     6     7     8     9    10    11    12
```

Adding two vectors together is extremely straightforward (given that they are the same size):

```
c = a + b
```

Matlab returns:

```
c =
     4     6     8    10    12    14    16    18    20
```

Plotting

Plotting is arguably one of the most important functions in Matlab. It is also fairly easy to use. The function plot(X, Y) plots vector X versus vector Y.

To generate a time series vector from zero to 2π and set it equal to the variable 'x', enter in the following line:

```
x = 0 : 0.1 : 2*pi;
```

The constant 'pi' stands for the symbol π, which is approximately equal to 3.141592654.

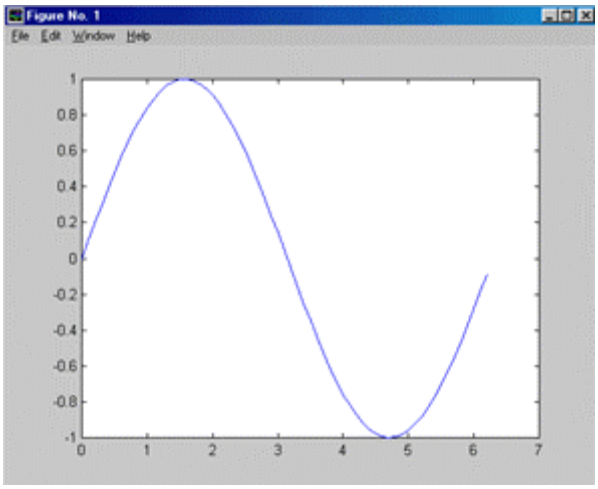
Define a new variable 'y' which depends on 'x':

```
y = sin(x);
```

Now it is simple to plot 'x' versus 'y':

```
plot(x, y);
```

Note that a new plot pops up, which should look like the following:



Close the plot and return to the Matlab command window. Another useful way to produce a time series is by using a function 'linspace(x1, x2, N)'. The function generates a vector from 'x1' to 'x2' splitting it into 'N' pieces. Generally it is desirable to keep 'N' relatively large so that your plots looks smooth. Generate a new time series for 'x' using this function:

```
x = linspace(-pi, pi, 100);
```

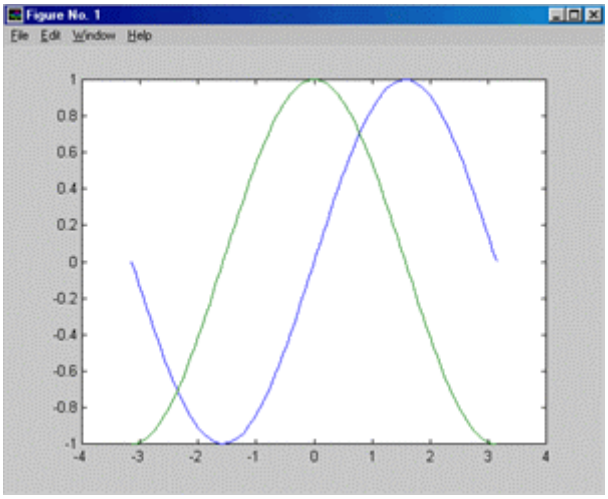
Let's define two new variables that depend on our time series:

```
y = sin(x);
z = cos(x);
```

To plot both of these functions on the same graph type:

```
plot(x, y, x, z);
```

A new figure should pop up that looks like the following:

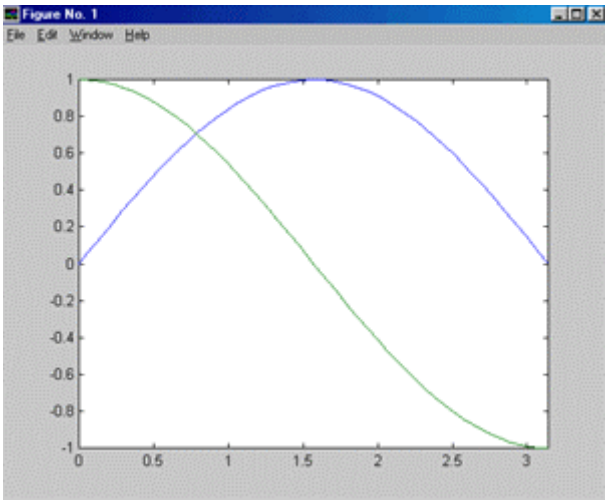


Note that both the sin(x) and cos(x) graphs appear in different colors. For more information on the plot function type ‘help plot’.

Let’s say you were interested in looking only at one part of the plot, but did not want to generate a new time series and dependent function. Use the function ‘axis([xmin xmax ymin ymax]);’. Let’s modify the above graph so that we look at the section of the plot from x = 0 to π:

axis([0 pi –1 1]);

The resulting graph should appear like the following plot:



It is always nice to be able to add text and other information to your graphs. There are a handful of functions designed for this purpose. To generate a titles for graphs use the function ‘title(‘text’)’. To label the x- and y-axes use the functions ‘xlabel(‘text’)’ and ‘ylabel(‘text’)’ respectively.

It is also possible to add text labels directly to the plot using the function ‘gtext(‘text’)’. This function brings us the currently selected graph and puts up a cross-hair. Move the cross-hair to the desired area on the graph and click where you want to place your text.

Here are some other functions that are useful to know:

clf	clears the current graph
figure	opens a new figure to plot on so the previous plot is preserved. To go back to a previous plot ‘n’ by using ‘figure(n)’
subplot(r,c,p)	breaks the current window into a matrix of ‘r x c’ matrix of plottable axes and selects the ‘p-th’ graph for the current plot.
close	closes the current window
loglog()	same as plot, but both axes are on a log base 10 scale
semilogx()	same as plot, but the x-axis is on a log base 10 scale

semilogy()	same as plot, but the y-axis is on a log base 10 scale
grid	adds a grid to the current plot

M-Files

One performs automated tasks in Matlab tasks or scripts in Matlab by writing what are called M-files. It is useful to use these M-files to define multiple variables or to define ordinary differential equations as seen later. It is possible to type anything into these files that can be typed in Matlab, but it is necessary to save them someplace Matlab can find it. This is usually in the 'MATLAB\bin\' directory.

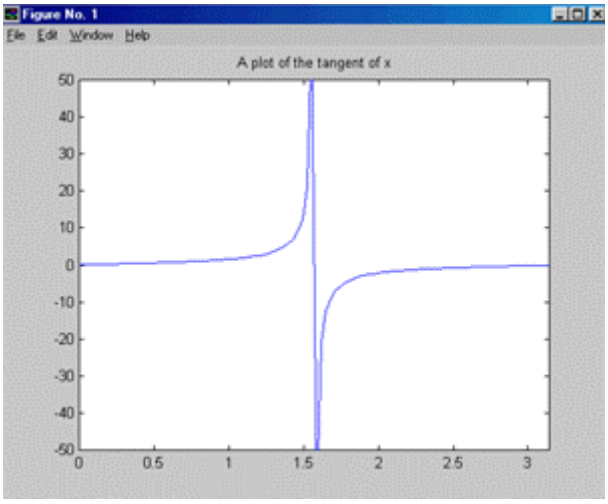
To create a new M-file in the Windows operating systems click File-New-M-file. This opens up a program called the M-file Editor / debugger. Type the following lines of code into the editor:

```
x = linspace(0, pi, 100);
y = tan(x);
plot(x, y);
title('A plot of the tangent of x');
axis([0 pi -50 50]);
```

Then click File-Save As… Make sure the path is set to the 'MATLAB\bin\' directory. In the 'File name' box type in 'tanplot'. Save the file and close the M-file editor program. From the Matlab command window type in:

tanplot

The following figure should appear:



Ordinary Differential Equations

All of the following examples draw on simple population models from ecology. Let's start out with probably the most simple system of all, which models exponential growth. We are going to code up the following ODE:

$$\frac{dN}{dt} = rN$$

In the model let's assume that the initial population is 2.0. The model has one parameter 'r' which determines how fast the population will grow. Open the M-File editor and type in the following lines of code:

```
function dy = popgrowth(t, y)

% create and empty vector
dy = zeros(1, 1);

% growth rate parameter
r = 0.25;

% simple exponential ODE
dy(1) = r*y(1);
```

Note a few things first. The word 'function' is highlighted in blue and tells Matlab that we are defining a new function. The lines in green that start with '%' are comments and are ignored by the Matlab interpreter. Save this file in the 'MATLAB\bin\' directory as 'popgrowth'.

Go back to the Matlab command window and type in the following line:

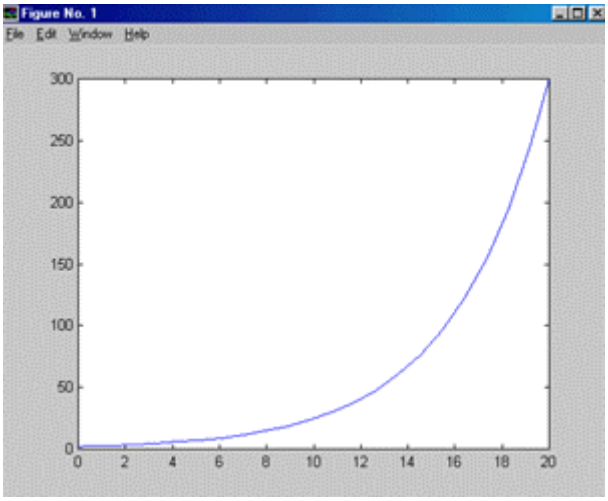
[t, y] = ode23s('popgrowth', [0 20], [2]);

This may seem a little complicated, but it is easy to understand if it is broken down into its individual parts. First of all we are creating two new vectors. One is for time 't' and the other vector is for the population. The vectors are being set equal to the output from a function called ode23s. This function takes care of numerically solving the function, which is located in the file we just saved as 'popgrowth.m'. There are two other arguments in this function, which are both vectors. The first vector '[0 20]' tells the integrator how long to do the numerical integration. The second vector is the initial condition for our ODE.

To plot this system out over time type in the following line:

plot(t, y);

The resulting plot will appear like the following:



After finishing coding up the first ODE, it is easy to see that it is not very biologically realistic. While populations might exhibit exponential behavior for short periods of time, it is impossible to keep this up.

Let's modify our ODE to take into account density dependent population growth. The logistic model is defined by the following equation:

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right)$$

There is one new parameter 'K', which represents the carrying capacity. Open up the M-File editor or click 'File-Open'. Click on the 'popgrowth.m' file we worked on earlier. Modify the code so that it looks like this:

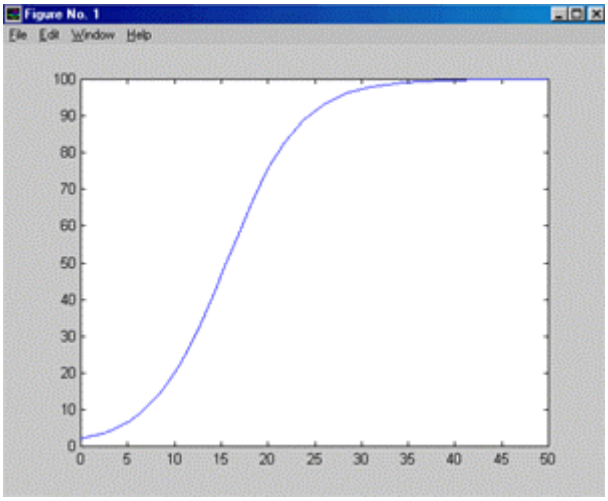
```
function dy = popgrowth(t, y)

% create and empty vector
dy = zeros(1, 1);

% growth rate parameter
r = 0.25;

% carrying capacity parameter
K = 100;

% simple logistic ODE
dy(1) = r*y(1)*(1-y(1)/K);
```

Save the M-file then go back to the main Matlab command window. Type the following two lines:

[t, y] = ode23s('popgrowth', [0 50], [2]);

plot(t, y);

The graph should appear like the following:



Note that initially, when 'N' is small the system grows almost exponentially. As 'N' increases the growth of the population slows down and reaches an equilibrium at N = 100.

Let's look at a system of ODEs that has two equations. Another classic model in ecology is the lokta-volterra predator-prey model, which is represented by the following equations:

$$\frac{dN}{dt} = rN - aCN$$
$$\frac{dC}{dt} = faCN - qC$$

Where

N is the number of prey
C is the number of predators
r is the growth rate for prey
a is the attack efficiency of predators
f is the rate at which predators turn prey into offspring
q is the starvation rate for predators

Open a new M-File and type in the following commands:

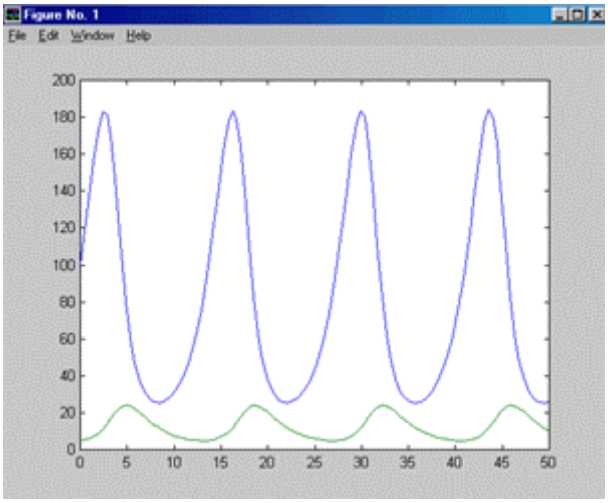
```
function dy=predatorprey(t, y)

dy=zeros(2, 1);

% Parameters
r = 0.6;
```

```
a = 0.05;
f = 0.1;
q = 0.4;

% ODEs
dy(1) = r*y(1) - a*y(1)*y(2);
dy(2) = f*a*y(1)*y(2) - q*y(2);
```

Save the file as ‘predatorprey’. To plot both of these populations versus time first do the numerical integration and then use the plot command as before:

[t, y] = ode23s(‘predatorprey’, [0 50], [100 5]);
plot(t, y);

The resulting plot should appear like the following plot:

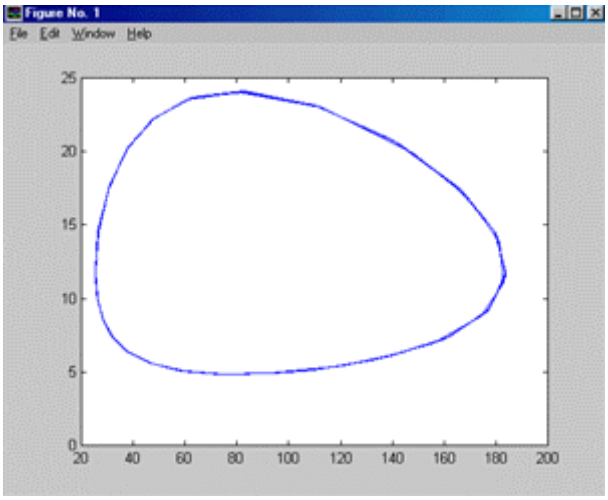


Note the oscillatory behavior of the system. As the prey population increases so does the predator population with a slight time delay. The increase in the predator population drives the prey population down and the cycle then repeats.

Another interesting plot that shows the oscillatory behavior nicely is plotting the first equation against the second. These types of plots are called phase plane or state space diagrams. They show the orbit along which the two populations travel. Type in the following line to produce this type of graph:

plot(y(:,1), y(:,2));

The result is a state space diagram with an egg shaped stable orbit for the two populations:



Let's extend what we've learned to a new system of three equations. Kermack and McKendrick developed the classic SIR model in 1927. Their model was used to quantitatively explain the dynamics of an epidemic. SIR is actually an acronym that stands for susceptible, infected, and removed or resistant. The susceptible (S) group is made up of healthy individuals who are available hosts for a disease and are assumed to have no prior immunity. The infected (I) group is made up of hosts that carry the disease. The removed (R) group is made up of individuals that have either recovered from the disease and gained immunity, individuals that have been quarantined, or individuals that have died from the disease. Their model can be described by the following set of equations:

$$\frac{dS}{dt} = -b\,SI$$
$$\frac{dI}{dt} = b\,SI - g\,I$$
$$\frac{dR}{dt} = g\,I$$

Where

β is the rate of transmission

1/γ represents the average length of infectivity

Many contemporary models use the classic SIR model as a starting point. In the literature you will encounter variations on the theme such as SIS, SIRS, SEI, SEIS, SEIRS, etc (note the (E) in these models stands for exposed and usually represents a latent period in the model). Many of these models are designed very specifically for particular diseases.

See if you can code up these equations on your own. There is example code on the internet if you get stuck:
http://www.biomath.medsch.ucla.edu/faculty/sblower/biomath209/tutorial/index.html

Exercises:

1. Code up the model using parameter values between 0.0 and 1.0 and plot out the state variables.
2. Try and figure out how to plot the prevalence of infection (100*I / [S + I + R]). This represents the fraction of individuals in the population who are infected at a given time.
3. The model makes certain assumptions about a disease:
    a. It is assumed that the disease is not lethal, but this is not always the case. How would you change the model to account for this?
    b. Once you have recovered it is assumed that you have permanent immunity. How would you account for immunity that declined over time?
    c. The population size does not change (i.e. there are no births or deaths). How would you change the model to account for these characteristics? A model that includes these factors is said to have vital dynamics.

Here are plots for the state variables and prevalence of infection: