Getting started with MATLAB
Local Guide

Matthew Leingang

December 7, 1999

This is to get people up and running on the very useful and powerful mathematical computer application MATLAB. Some of the instructions are very specific to the time and location of this tutorial. The rest serve as a general introduction.

1 Starting Matlab

First, log on to the machine in front of you. You should be given an xterm window. We are going to be running MATLAB in a Unix environment on these Xwindows machines, because that's what's available right now. So over the course of this tutorial you will come to know a little bit about Unix, and about the text editor Emacs.

If in the future you are running MATLABon a Mac or PC, things will be much different. There will be more windows with menus that speed up certain tasks, and a text editor is built in, so there is no need to run Emacs. So many of these instructions not pertaining to MATLAB can be ignored.

1.1 Logging on to `ice`

Right now, MATLABis not installed on the math department's main Unix computer (abel), so we have to use it on the FAS machines. So we have to open a connection to it. So at your prompt, type

```
% ssh -l username ice
```

and hit **Return**. Here *username* is your `fas` username, not your `math` one. You will be prompted for a password; enter it. Or you might see this:

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)?
```

Type `yes`, then hit **Return**. You will finally get a prompt that looks like

```
ice%
```

If you ever come back to this document and want to run MATLAB on abel (assuming it's been installed), you can skip all of this subesection. You'll see a different prompt, probably having the word "abel" in it, but most everything else should work the same.

1.2 Prior to starting

First, it's a good idea to create a directory to put all of our MATLAB stuff in. (Directories are like folders.) So type:

```
ice% mkdir matlab
```

This creates a directory called `matlab` in your home directory in which you can keep all your M-files, et cetera. You only have to do this once.

Secondly, we are going to have to edit some text files, so let's start a text editor.

```
ice% emacs &
```

You should get a new window opening. We'll get back to it in a moment, but it was important to start emacs before the next step, which is:

```
ice% matlab
```

This will actually start matlab. You'll get a welcome message that looks like this:

```
                          < M A T L A B >
              Copyright 1984-1999 The MathWorks, Inc.
                 Version 5.3.0.10183 (R11)
                        Jan 21 1999


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, type tour or visit www.mathworks.com.

>>
```

The `>>` is the prompt; this is where you tell MATLAB what to do.

2 Take the tour

Just to see some of the things MATLAB can do, you might want to take the tour as suggested by the welcome message. To do this, type

```
>> tour
```

at the prompt. Play with the window that opens up, and you can see some good demonstrations.

3 Basic Input

3.1 Matrices

See pp. 3–10 of the document *Getting Started with* MATLAB, copyright 1984–1998 by The Math Works, Inc.

4 Writing a function

4.1 Level I: numbers

Any function you write for MATLAB has to be edited and stored as its own M-file. So let's create one.

Remember that Emacs window? Emacs is a plain text editor that we will use to create a file to define a function. With your mouse pointer in that window, hit the keys `C-x C-f` (that `C-` means to hold down the **Control** key while you hit the next key), and you'll see the `Find File:` prompt in the bottom of the window. Alternatively, you can select Files→Open File... from the menu bar.

Type in `matlab/logistic.m'` at the Emacs prompt. You should get a blank screen and the message `(New File)` in the status bar. You've begun a new file, which is going to be stored in your `matlab` directory, in which we will define a MATLAB function.

Type these lines into your file:

```
function f = logistic(x,r)
% LOGISTIC(X,R) provides the logistic map
%   logistic(X,R) = R X (1 - X)
f = r * x * (1 - x);
```

The first line declares your function. It basically tells MATLAB "this file will define the function 'logistic', which takes two variables, call them x and r." The next two lines are comments, and that's why they begin with `%`. These tell the *user* that 'logistic' is a function and how it's defined. It is important to document your code lest you forget months later what this function was for! The only real line is the fourth, which actually defines the function. Save the file. This is done by entering `C-x C-s`, or by selecting Files→Save Buffer from the menu bar.

Now go back into MATLAB, and try

```
>> help logistic
```

You should see

```
LOGISTIC(X,R) provides the logistic map
  logistic(X,R) = R  X (1 - X)
```

There's your documentation. Now let's actually try it. I'll put your input and the computer's output one right after the other.

```
>> logistic(0.1,3)

ans =

    0.2700
```

Let's iterate by hand. Notice that the output is preceded by the message `ans=`. This means you can use `ans` in the next line, and 0.2700 will be substituted in. A second technique you'll learn is the built-in history buffer. MATLAB remembers your commands. Hit the up arrow, and you'll see exactly what you just typed. Change 0.1 to `ans` and hit **Return**. You'll see:

```
>> logistic(ans,3)

ans =

    0.5913
```

Keep hitting ↑ and **Return**, and you can continue to iterate the logistic map.

4.2 Level II: vectorizing

We haven't used one of the best capabilities of MATLAB: to work with matrices. We just gave it numbers. So let's soup up our logistic map to accept matrices as well. The answer we want should just be the map on each entry.

Multiplying matrices is no problem with MATLAB. In fact, the `*` in the file will work with matrices. Try:

```
>> logistic([1 2 ; 3 4],[5 6;7 8 ])

ans =

   -68  -125
   -92  -169
```

So it worked without our even knowing that we were writing it to accept matrices. The problem is that it doesn't really give us the answer that we want. If $f_r(x)$ were the logistic map, we probably meant by this input that we wanted the matrix

$$\begin{matrix} f_5(1) & f_6(2) \\ f_7(3) & f_8(4) \end{matrix}.$$

Instead, what we got was the matrix

$$\begin{matrix} 5 & 6 & 1 & 2 \\ 7*8 & & 3*4 & \end{matrix}\left(\begin{matrix} 1 & 1 \\ 1*1 & \end{matrix} - \begin{matrix} 1 & 2 \\ 3*4 & \end{matrix}\right)$$

Which is the logistic map with multiplication of matrices as matrices.

To perform the component-wise multiplication of matrices, replace `*` with `.*`. That is, change the fourth line of `logistic.m` to read

```
f = r .* x .* (1 - x);
```

Then save the file and try it again in MATLAB:

```
>> logistic([1 2 ; 3 4],[5 6;7 8 ])

ans =

     0   -12
   -42   -96
```

This is the right "vectorization" of the logistic map.

5 Scripting

5.1 A bifurcation table

Let's use this to make a logistic bifurcation diagram. To do this, first we have to make a large table. We'll try to create a matrix of high iterates of the logistic map f_r for various values of r.

Let $r = (r_1, \ldots, r_k)$ be a vector of r-values. Given a row vector $x = (x_1 \ldots, x_k)$, we can create the new vector

$$f_r(x) = (f_{r_1}(x_1), \ldots, f_{r_k}(x_k)),$$

and iterating this, we'd get a sequence of vectors

$$(f_r)^j(x) = \left(f_{r_1}^j(x_1), \ldots, f_{r_k}^j(x_k)\right).$$

The algorithm is the following. Take a random vector $x \in \mathbb{R}^k$. Iterate the map f_r enough times, say N. Call the resulting vector x again. Then create a $k \times M$ matrix A, where the nth row of A is the nth iterate of x. Finally, we transpose the matrix A, given a matrix where each row is the Nth to the $(N+M)$th iterates of f_r for a single value of r.

Begin a new file in the `matlab` directory called `logistic_limit.m`. Put in it the folowing:

```
function A = logistic_limit(R,M,N)
% LOGISTIC_LIMIT(R,M,N) limit orbit of the logistic map
%   For each value of r in R, a (row) vector of the Mth to the (M+N)th
%   iterates of the logistic map at r is generated.


x = rand(1,length(R));
A = zeros(N,length(R));

for n = 1:M
x = logistic(x,R);
```

```
end

for n = 1:N
A(n,:) = x;
x = logistic(x,R);
end

A=A';
```

Look at the code carefully. First we create matrices x (to have random entries) and A which will first have zeros. It's safer to initialize any big matrix you're going to create. Then we iterate f M times. Then we begin iterating again, this time setting a row of A each time. That's what the notation `A(n,:) = x` does; it sets the nth row of A to x. When we're done, we transpose A and quit.

Try it out:

```
>> R = linspace(0,4,100);
>> A = logistic_limit(R,100,100);
>> plot(R,A,'r.');
```

The `linspace` command generates a row vector of 100 values between 0 and 100. Take out the semicolon if you actually want to see R. We then use our `logistic_limit` function to make the matrix A.

5.2 Some graphics

Finally, we plot the rows of A versus the elements of R with the `plot` command. The notation `'r.'` is a sort of plotting style; it says use red dots. Try `'rs'`, `'c^'`, and `'go'` to see others.

To get some more detail, let's make a bigger R matrix.

```
>> R = linspace(3,4,1000);
>> A = logistic_limit(R,100,100);
>> plot(R,A,'r.');
```

Now you notice that the point sizes are too big. So enter:

```
>> plot(R,A,'r.','MarkerSize',5);
```

and this should be more like it.

5.3 synthesis

Let's put these together in one fell swoop. Make a new file, called `logistic_diagram.m`, with the following text in it:

```
% LOGISTIC_DIAGRAM is a script which will generate the logistic
% bifurcation diagram of the limiting orbit of logistic(x,r)^n
% versus r.  You are prompted to enter in the values of r to test
```

```
% along with the values of N (number of iterates to get to limiting
% behavior) and M (number of itererates to plot thereafter).

R = input('Please input a vector of R values: ');
N = input('Give the number N of iterates to allow transients to die down: ');
M = input('Give the number M of iterates to plot: ');

A = logistic_limit(R,N,M);
plot(R,A,'r.','MarkerSize',5);
```

Now try it:

```
>> logistic_diagram
Please input a vector of R values: linspace(3,4,1000)
Give the number N of iterates to allow transients to die down: 100
Give the number M of iterates to plot: 200
```

The diagram should appear automatically. This has the advantage that you don't have to deal with the plotting commands independently of the number crunching, but you'll also have to change this file if you want to change the plot style.