

Intro to Matlab for ATOC 5051: Introduction to Physical Oceanography or, Notions for the Motions of the Oceans

Baylor Fox-Kemper

September 24, 2008

1 Contacts

The professor for this class is: Baylor Fox-Kemper

bfk@colorado.edu

303-492-0532

Office: Ekeley room S250B

<http://cires.colorado.edu/science/groups/foxkemper/classes>

2 Getting Help!

The most important commands in matlab are ‘help’ and ‘lookfor’. The first one allows to to get a description of any matlab function, for example, ‘`>> help plot`’ tells you about the function named ‘plot’. The second one allows to to search for keywords within a function description (in case you don’t know or can’t remember the name of the function).

3 The Basics: Matlab is Matrix Laboratory

Matlab is based on matrix algebra. So, when you think about data, you think about making arrays/vectors/matrices of data. In that way, it can be a lot like a spreadsheet program, but it is much more powerful because 1) it can handle much larger quantities of data, and 2) you can use pre-programmed solver routines to get things done.

The following contains a number of examples. Type them into matlab and see what happens! Try changing them up a bit, and see what happens then. Good hunting!

3.1 Scalars

Some simple examples of matlab scalar (that is one number arrays) at work::

```
>> A=1
A =
    1
>> B=2
B =
    2
>> B*A
ans =
    2
```

```

>> B+A
ans =
    3
>> size(A)
ans =
    1    1
>> size(B)
ans =
    1    1
>> sqrt(B)+exp(A)
ans =
    4.1325

```

3.2 Vectors

Now, let's consider vectors. You can make a horizontal vector:

```

>> A=[1 1 1 1]
A =
    1    1    1    1

```

Or a vertical one:

```

>> B=[1;1;1;1]
B =
    1
    1
    1
    1

```

You can't add together a horizontal and a vertical vector:

```

>> A+B
??? Error using ==> plus
Matrix dimensions must agree.

```

But you can use the transpose single quote ' to transpose a matrix, or in this case convert an horizontal vector to a vertical one.

```

>> A'+B
ans =
    2
    2
    2
    2

```

The * operator is a vector or matrix multiply, which in this case is the dot-product of A and B:

```

>> A*B
ans =
    4

```

While the .* operator multiplies component-by-component (if the vectors are the same shape...):

```
>> A' .* B
ans =
     1
     1
     1
     1
```

3.3 Matrices

Matrices behave in much the same way as vectors, except now there are both rows and columns.

```
>> A=[1 2 3; 4 5 6]
A =
```

```
     1     2     3
     4     5     6
```

```
>> B=[1 4; 2 5; 3 6]
B =
```

```
     1     4
     2     5
     3     6
```

```
>> A+B
```

```
??? Error using ==> plus
Matrix dimensions must agree.
```

```
>> A+B'
```

```
ans =
     2     4     6
     8    10    12
```

```
>> A.*B
```

```
??? Error using ==> times
Matrix dimensions must agree.
```

```
>> A.*B'
```

```
ans =
     1     4     9
    16    25    36
```

```
>> A*B
```

```
ans =
    14    32
    32    77
```

There are many special commands for generating matrices. The most important are:

```
>> ones(5)
```

```
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
>> ones(3,2)
```

```
ans =
```

```

    1    1
    1    1
    1    1
>> zeros(5)
ans =
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
>> zeros(3,2)
ans =
    0    0
    0    0
    0    0
>> eye(5)
ans =
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
>> rand(5)
ans =
    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787

```

3.4 Accessing a Submatrix

You don't have to use the whole matrix all at once. The colon `:` plays a special role in accessing a subset of the matrix. Used alone, it means 'that whole row'. Used with a number before (A) and after(B), it means, 'that whole row from A to B'. Used with three numbers it means, 'that whole row from A to B jumping by C (A:C:B)'. For example:

```

>> C=[1 2 3 4 5 6; 7 8 9 10 11 12]
C =
    1    2    3    4    5    6
    7    8    9   10   11   12
>> C(1,:)
ans =
    1    2    3    4    5    6
>> C(2,:)
ans =
    7    8    9   10   11   12
>> C(:,2)
ans =
    2
    8

```

```

>> C(2,4:5)
ans =
    10    11
>> C(2,1:2)
ans =
     7     8
>> C(2,1:2:end)
ans =
     7     9    11
>> C(2,1:3:end)
ans =
     7    10
>> C(2,1:1:end)
ans =
     7     8     9    10    11    12
>> C(2,end:-1:1)
ans =
    12    11    10     9     8     7

```

3.5 Higher-order Tensors

Of course, you can have more indices on your variables,

```

>> A=ones(3,4,5)
ans(:,:,1) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
ans(:,:,2) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
ans(:,:,3) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
ans(:,:,4) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
ans(:,:,5) =
     1     1     1     1
     1     1     1     1
     1     1     1     1

```

But you won't be able to easily use the matrix multiply and other matrix-based arithmetic. However, it is easy to convert a submatrix into a real matrix:

```

>> A(:,2,:)
ans(:,:,1) =
     1

```

```

1
1
ans(:,:,2) =
1
1
1
ans(:,:,3) =
1
1
1
ans(:,:,4) =
1
1
1
ans(:,:,5) =
1
1
1
>> squeeze(A(:,2,:))
ans =
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

```

3.6 Quiet Mode

To stop matlab from spitting everything back at you, end each statement with a semi-colon

```

>> A=ones(3)
A =
1 1 1
1 1 1
1 1 1
>> A=ones(3);
>>

```

4 Getting and Saving Data: load, save, and loadsave

Matlab is very good at loading and saving its own kind of files, .mat files. Another useful function is 'who' which tells you the names of defined variables.

```
>> who
```

Your variables are:

```
A B C ans
\end{verbatim}
```

To make a .mat file, you just do the following:

```
\begin{verbatim}
>> save vars.mat

```

```
\end{verbatim}
```

To save only a few of the variables, you list them after the desired filename:

```
\begin{verbatim}
```

```
>> save varsAB.mat A B
```

Now, to check what's in the files, we first clear the memory with 'clear', then

```
>> clear
```

```
>> load varsAB.mat
```

```
>> who
```

Your variables are:

```
A B
```

```
>> load vars.mat
```

```
>> who
```

Your variables are:

```
A B C ans
```

Load/Save are also capable of handling text files (with the flag `-ascii`), but you usually need to pretreat the text files, e.g., by deleting headers on columns.

Furthermore, standard unix and ms-dos commands, e.g., `ls`, `dir`, `cd`, work as expected showing the contents of directories and changing the local directory,

4.1 nload and ncx

The add-on function `nload` is available from the `ncx` toolbox linked from the course webpage. It allows you to load a `netcdf`-formatted dataset from a webpage almost as though it were a matlab file.

The website has a link to `ncx` called `netcf on matlab`. Installing this software, and adding the installed directories (with subdirectories) to the matlab path will allow you to browse, load, and save `netcdf` files.

Perhaps the easiest way to get data into matlab is using `nload`:

```
>> nload('file.nc')
```

This loads the entire contents of the file into matlab's memory buffer. Your file may be too large to do it all at once, so you can load subsets

```
>> nload('file.nc', 'Var1', 'Var2')
```

Where `Var1` and `Var2` are the names of variables in the dataset. How do find out the names? Opening the dataset in the `ncx` browser works, or using features from the `netcdf` toolbox:

```
%%THIS IS THE CRUCIAL STEP%%
```

```
%%IT SAYS OPEN THE netcdf FILE WITH A HANDLE (named nc here)%%
```

```
>> nc=netcdf('file.nc')
```

```
%%Now we can use this handle to manipulate the data in the file:
```

```
>> vars=var(nc)
```

```
>> vars{1}
```

****Gives output about variable 1****

```
>> vars(2)
```

****Gives output about variable 2, etc.****

The netcdf toolbox is particularly good about opening only a subset of the data in a file. For example, if by doing the `vars=var(nc)` trick above, we find out that there is a variable named T that we want to get the data from, we can do:

```
>> nc{'T'}
>> nc{'T'}(1) % gives the first value of T
>> nc{'T'}(1:5) % gives the first 5 values of T
>> nc{'T'}(1:5, 10:100) % gives the 1:5, 10:100 block of T (assuming T is a 2d matrix)
>> nc{'T'}(:) % gives all of T as a vector
>> nc{'T'}(:, :) % gives all of T as a matrix (assuming it's a 2d matrix)
```

You can also find lots of info about using the netcdf toolbox by typing:

```
>>help netcdf
```

There is also a user's guide with examples on the netcdf toolbox webpage.

4.1.1 Trouble Installing

Netcdf libraries are constantly changing, as some of you have found. If the netcdf libraries included with `ncx` don't work for you (e.g., matlab complains about a missing `mexcdf53` function), you can try installing the `mexnc` binaries after installing `ncx`. I have added a link to the binaries on the webpage.

4.2 loaddap

The add-on function `loaddap` is available from the Matlab OPenDap toolbox linked from the course webpage. It allows you to load an oceanographic dataset from a webpage almost as though it were a matlab file.

4.3 ASCII files

If all else fails above, but you are able to download an ascii file version of your dataset, matlab is fairly capable at interpreting such files. The function `load -ascii` will do very simply organized datasets, but crashes if there are column titles, missing values, etc. The more powerful function `textscan` can be instructed to ignore comments, process nans, etc., but it takes a little TLC to get it working.

5 Making Plots

The important plot commands are (use `help 'command'` for more detail):

- `plot` (plots scatter and line plots), e.g., `plot(1:10,exp(1:10))`
- `plot3` (plots 3d scatter and line plots), e.g., `plot231:10,exp(1:10),sin(1:10))`
- `figure` (generates a new figure and window)
- `subplot` (generates a subplot within a figure for making paneled figures)
- `contour` (generates a contour plot), e.g., `contour(1:10,1:20,sin((1:20)'*(1:10)))`

- `contourf` (generates a filled contour plot), e.g., `contourf(1:10,1:20,sin((1:20)'*(1:10)))`
- `axis` (subselects the figure axes)
- `saveas` (allows you to save a figure as a jpg or pdf, etc.)
- `pcolor` (generates a shaded plot), e.g., `pcolor(1:10,1:20,sin((1:20)'*(1:10)))`, often used with `shading('interp')` or `shading('flat')`
- `scatter` (allows you to make a map of scattered data, with colored dots representing a dependent variable's values)
- `caxis` (allows you to change the limits of most color-coded plots, so you choose the max and min of the colorbar)
- `colorbar` (adds a colorbar to a figure based on the `caxis` values)

6 Doing Stats and Calculations

These are really easy! Some examples:

```
>> A=rand(4,5)
A =
    0.7577    0.1712    0.0462    0.3171    0.3816
    0.7431    0.7060    0.0971    0.9502    0.7655
    0.3922    0.0318    0.8235    0.0344    0.7952
    0.6555    0.2769    0.6948    0.4387    0.1869
>> mean(A)
ans =
    0.6371    0.2965    0.4154    0.4351    0.5323
>> mean(A(1,:))
ans =
    0.3348
>> mean(A(:,1))
ans =
    0.6371
>> std(A(:,1))
ans =
    0.1694
>> std(A)
ans =
    0.1694    0.2909    0.4009    0.3829    0.2975
>> var(A)
ans =
    0.0287    0.0846    0.1607    0.1466    0.0885
```

7 Saving a Script or Function

The last piece worth mentioning here is the `.m` file format. You can type a series of commands into a file named (for example) `doit.m`. Then, while in the same directory as `doit.m`, if you type

```
>> doit
```

it will execute the commands.

If you define a function (see `help function`), then you can call that function from within the directory. A function is written as a `.m` file, but has added capabilities, e.g., input, output arguments, private variables, etc.