Introduction to Matlab

Tyler Ransom[1]
Duke University

July 3, 2012

[1]Slides adapted with permission from Justin Trogdon.

Outline

- Matlab user interface
- Accessing Matlab at Duke
- How Matlab works
- Operators
- Elementary Matrices
- Flow Control
- Loops
- Working with Data
- Help in Matlab
- Functions

User Interface

There are five main windows on the user interface:

1. **Command Window**: Interactive command line where user can create matrices, perform calculations, etc.
2. **Workspace**: List of variables that are available to the user
3. **Command History**: Lists recent commands issued by the user
4. **Current Folder**: Gives a directory listing of the current directory (User can change the directory at the top of the screen, or by typing "cd " in the command window)
5. **Variable Editor**: Allows user to view specific observations of a variable

Accessing Matlab at Duke

Four ways:

1. Windows machines in the Bowling Alley
   - Log in and double-click on Matlab icon (make sure to change directory to H:\)
2. Linux machines in the Bowling Alley
   - Log in, open the terminal and type "`matlab &`" or "`matlab -nodisplay`" (batch mode)
3. SSH terminal from a laptop or non-Duke desktop
   - Log in, type "`matlab`" (interactive) or "`matlab -nodisplay`" (batch)
   - Interactive mode typically runs very slowly when not on the Duke network
   - Batch mode runs very quickly but only has the command prompt, so the user can't see the workspace or command history
4. Personal license
   - Students can purchase a license of Matlab (Student Version) from MathWorks for $100. The license is valid so long as it's being used for coursework. Non-student personal licenses are also available for purchase, but can be quite expensive.

How Matlab Works

- User gives it a text file (.m), which the program interprets line-by-line
- This is equivalent to evaluating each line at the Command Line

There are two types of m-files in Matlab:

1. **Script files**: Series of command line commands (read line-by-line)
2. **Function files**: Take arguments as inputs and give outputs

Remember that nothing a user does is reproducible unless it is contained in a .m file.

Operators

```
%           %in-line comment
;           %suppresses output
A(:,1)      %references the first column and all rows of A
A(1,:)      %references the first row and all columns of A
A = []      %creates an empty matrix named A
A(index,:)  %subsets the rows of A by some logical vector (index)
[A; B]      %vertically concatenates matrices A and B
[A B]       %horizontally concatenates matrices A and B
A.B         %creates a structure A with sub-matrix B
A'          %transpose of A
A+B         %matrix addition of A and B
A-B         %matrix subtraction of B from A
A*B         %matrix multiplication of A and B
```

Always use brackets to concatenate matrices and parentheses to reference matrix elements

Operators

```
1  A^B          %matrix exponentiation; B must be a scalar and A must be square
2  A.*B         %element-by-element multiplication of A and B (must be same size)
3  A\B          %postmultiplies A inverse by B
4  A/B          %postmultiplies A by B inverse
5  kron(A,B)    %Kronecker product of A and B
6  A.^B         %element-by-element exponentiation of A by B
7  (A ≤ B)      %creates a logical array that is 1 if A≤B, 0 otherwise
8  (A == B)     %creates a logical array that is 1 if A=B, 0 otherwise
9  (A ≥ B)      %creates a logical array that is 1 if A≥B, 0 otherwise
10 A & B        %creates a logical array that is 1 if both A and B have ...
       non-zero elements at that location
11 A | B        %creates a logical array that is 1 if either A or B have ...
       non-zero elements at that location
12 ¬A           %same as A==0
```

Note that parentheses can also be used to create logical variables

Elementary Matrices

Matlab has commands to create "common" matrices so that users don't have to hand-code them

```
1  eye(n)      %n-by-n identity matrix
2  ones(n,m)   %n-by-m matrix with the number 1 in every element
3  zeros(n,m)  %n-by-m matrix with the number 0 in every element
4  m:step:n    %row vector from m to n increasing (or decreasing) by step ...
       each time
5  rand(n,m)   %n-by-m matrix of Uniform[0,1] random numbers
6  randn(n,m)  %n-by-m matrix of Normal(0,1) random numbers
7  mvnrnd(mu,sigma) %matrix of multivariate Normal(mu,sigma) random numbers ...
       (dimension is same as mu)
8  length(A)   %lists number of elements of the longest dimension of A
9  size(A,d)   %gives the number of elements of the dth dimension of A
10 diag(A)     %captures diagonal elements of A if A is square. If A is a ...
       vector, creates a diagonal matrix from the elements of A
11 triu        %replaces lower triangular elements of A with zeros
12 tril        %replaces upper triangular elements of A with zeros
13 find(A>1)   %finds, e.g. elements of A that are larger than 1
```

Note that all "(n,m)" commands can be generalized to more than two dimensions by using "(n,m,p,...,z)"

Matrix Relations

Commands that help users know the properties of matrices in the workspace

```
1 isreal(A)    %returns a 1 if all elements of A are real numbers
2 isempty(A)   %returns a 0 if A has at least one element
3 issparse(A)  %returns a 1 if A is sparse (has many zeros)
4 isnan(A)     %returns a matrix of 0s or 1s corresponding to the elements ...
      of A that are not a number (NaN)
5 isfinite(A)  %returns a matrix of 0s or 1s corresponding to the elements ...
      of A that are infinite
```

Special Variables

Special variables reserved by Matlab

```
1 ans      %the value (answer) of the most recent command line input
2 inf      %positive infinity
3 -inf     %negative infinity
4 NaN      %missing value (Not a Number) -- this is case sensitive
5 eps      %machine epsilon; equal to 2^-52
6 realmax  %largest number Matlab can recognize; equal to 10^308
7 realmin  %smallest number Matlab can recognize; equal to 10^-308
```

Note also that Matlab can do other mathematical functions on matrices (e.g. $\log(A)$ takes the log of all elements of A)

Flow Control

- Suppose a user wants to control which part of the script is executed
- This technique is referred to as *flow control*
- Flow control in Matlab is very similar to other languages (C, Fortran)

Flow Control - if

```
1 if [some condition]
2     [some code];
3 elseif [some other condition]
4     [some code];
5 else
6     [some code];
7 end
```

Conditions must be logical statements.
`elseif` and `else` statements are optional

Flow Control - switch

```
switch [some expression]
case [some case]
    [some code];
case [some other case]
    [some code];
otherwise
    [some code];
end
```

Case expressions are either scalar numbers or strings (not logical statements).
Matlab goes through each case expression and executes the code below it if it matches the switch expression.
I have never used `switch`, but it is useful if the user needs to execute different code depending on a finite number of differing case scenarios

Flow Control - error and return

A user can check that code is running according to plan by forcing Matlab to give an error if something is amiss

```
1 if [condition that should hold doesn't]
2    error('This code doesn`t work');
3 end
```

A user can issue the `return` command to force Matlab to exit the script

Looping

- Matlab has two types of loops: `for` and `while`
- These loops literally execute commands repeatedly as long as the looping conditions are satisfied
- Loops in Matlab are useful for mechanizing code, but very slow because they are interpreted line by line
- `for` and `while` are basically the same, but have subtle differences
- Loop syntax:

```
for loopvar = starting_value:stepsize:stopping_value
    [code user would like to repeat]
end

while [looping condition]
      [code user would like to repeat]
end
```

Looping: `for` vs. `while`

Use a `for` loop when

- You know a command should be execute a certain number of times
- You can't use a built-in Matlab command to accomplish your design

Use a `while` loop when

- You don't know how many times the commands need to be repeated
- You can't use a built-in Matlab command to accomplish your design

Looping Example

- Suppose a user wants to "mechanize" how she is creating a matrix. For example, suppose she wants to create a matrix B that is 2^k for each row k in the first column, and $\ln(k)$ for each row k in the second column.
- If k is small, she could hand code it:

```
1 B(1,1) = 2^1;
2 B(1,2) = log(1);
3 B(2,1) = 2^2;
4 B(2,2) = log(2);
```

Looping Example - Continued

- However, if k is large, hand coding is tedious and more vulnerable to user mistakes.
- Instead, the user can write a `for` loop to accomplish the task:

```
1 B = zeros(1e5,2); %initialize the size of the matrix for speed
2 for k=1:length(B) %allow the loop length to change automatically ...
      according to the size of B
3     B(k,1) = 2^k;
4     B(k,2) = log(k);
5 end
```

- As you can see, loops are very powerful in performing repetitive tasks
- Matlab, however, is slow at doing loops (since it interprets commands line-by-line), so loops should generally be avoided if at all possible

When not to loop

- Suppose a user wants to delete rows in a vector V if elements of V are larger than 5
- Two ways to do this:

```
for i=1:length(V)
    if V(i)>5;
       V(i)=[];
    end
end
```

- A much easier way to do this is to make use of Matlab's logical indexing capabilities

```
V(V>5)=[];
```

While loops

Consider the `for` loop from before

```
1 B = zeros(1e5,2); %initialize the size of the matrix for speed
2 for k=1:length(B) %allow the loop length to change automatically ...
      according to the size of B
3     B(k,1) = 2^k;
4     B(k,2) = log(k);
5 end
```

We can accomplish the same task using a `while` loop:

```
1 B = zeros(1e5,2);
2 k = 1
3 while k≤length(B) %Note the use of the weak inequality
4     B(k,1) = 2^k;
5     B(k,2) = log(k);
6     k = k+1;
7 end
```

While loops - continued

- In their simplest form, `while` loops and `for` loops look very similar and accomplish the same task
- To illustrate the (subtle) difference, suppose a user wants to repeatedly execute code until some criterion is met
- However, the user may not know how many times the code should be repeated
- The `while` loop takes care of this in a way the `for` loop can't

```
1 % This code takes a matrix A and element-wise multiplies it by a matrix ...
      F, then divides by the loop counter until its norm reaches zero
2 A = rand(15);
3 F = eye(size(A));
4 i = 1;
5 while norm(F,1) > 0,
6    F = A.*F/i;
7    i = i + 1;
8 end
```

Looping Summary

- Subsetting `if` statements in Matlab should always be taken care of through logical indexing
- There are some cases when looping can't be avoided (e.g. merging two datasets, creating matrices that systematically vary by element location [like the example from two slides previous])
- `while` loops are useful for executing commands where the number of iterations is not known from the outset (e.g. contraction mappings)
- In fact, structural economists use `while` loops often because many problems can be characterized by a two-step estimation procedure which the `while` loop is especially tailored to deal with

Working with data in Matlab

- Matlab-formatted data comes in a .mat file
- `load mydata.mat` loads the data onto the workspace
- `load mydata.mat var1 var2 var3` loads variables `var1 var2 var3` of `mydata.mat` onto the workspace
- Be careful about which directory you're in when loading data. Matlab usually opens to its default directory, which is generally not the directory a user is working in
- `save` works just like load, except it saves variables in the workspace into a .mat file
- e.g. `save mydata1.mat var1 var2 var3`

Loading non-Matlab-formatted data

- The easiest way to import non-mat files is through the import wizard (Start▶Matlab▶Import Wizard) in the lower left corner of the Matlab interface
- If importing from other software (e.g. Stata or SAS), easiest way is to go from .dta▶.csv▶.mat
- StatTransfer also supports Matlab, though it is unavailable on Econ department machines (but available at Perkins Library)
- Matlab does not allow string components in data matrices, so make sure you don't export value labels from other software
- Also, users may need to manually rename variables once the data matrices have been imported

Reproducibly importing data

- The import wizard gives users the option to generate m-code that is reproducible; however, I've never been successful in re-running this reproducible code
- Matlab has built-in functions to import .csv or fixed-format file types:
- `csvread('xyz.csv')`
- `csvwrite('xyz.csv', A)`
- `dlmread('xyz.dat', 'delimiter string')`
- `dlmwrite('xyz.dat', A, 'delimiter string')`
- `A=importdata('xyz.csv', delimiter, nheaderlines)` is the best way to import data, because it automatically converts unreadable elements to `NaN`

Dealing with missing values

- Most of the problems in data conversion lie in mapping missing values from one software to another
- Missing values are coded as `NaN` in Matlab; in Stata or SAS they are a dot (.)
- StatTransfer automatically converts . to `NaN`
- Without StatTransfer, need to do a find-and-replace of . (or blank) to `NaN` prior to importing to Matlab (unless using `importdata`)
- `importdata` does this automatically
- Matlab 2011 has an import utility that allows the user to convert non-numeric values to NaN within the Matlab import wizard

Describing data

```
1  [val,idx]=sort(A)      %sorts each column of A in ascending order
2  [val,idx]=sortrows(A) %same as sort, but preserves row groupings (just ...
       like sorting in Stata or SAS)
3  [val,idx]=unique(A)    %strips out unique elements of A (no repeats)
4  [val,idx]=max(A)       %max element of A
5  [val,idx]=min(A)       %min element of A
6  mean(A)                %mean
7  median(A)              %median
8  var(A)                 %variance
9  std(A)                 %standard deviation
10 sum(A)                 %summation
11 hist(A)                %histogram
12 corrcoef(A)            %correlation coefficient of A
13 cov(A)                 %covariance of A
```

Note that summary statistics can be extended to multiple dimensions
e.g. `max(A,2)` is the maximum across columns of A (instead of down rows)

Describing data with missing observations

Summary statistics from the previous slide will return `NaN` if any element of the vector (or matrix) is `NaN`. There are two ways to get around this:

- `mean(¬isnan(A))` returns the mean ignoring missing values
- A faster way is to use `nanmean(A)`, which accomplishes the same task
- Most other commands listed in the previous slide have "nan" counterparts
- Other built-in functions will explicitly mention in the help menu that they ignore `NaN` values

Accessing Matlab's help

There are a number of ways to view documentation on a certain command, or find commands that accomplish something of interest:

- Type `help commandname` in the command window if you know the name of a command , but want to learn more about its syntax or usage
- Type `lookfor word` in the command window to get a list of Matlab commands that have `word` in the command description
- Google "matlab commandname" for help on the `commandname` command (Matlab's official documentation for the command is usually the top hit)
- Google something more general like "how to sort rows in matlab." This will usually bring up the Matlab documentation, but may also lead to valuable discussion sites like StackOverflow

It is rare to come across a problem or question that no one else has already thought of or solved

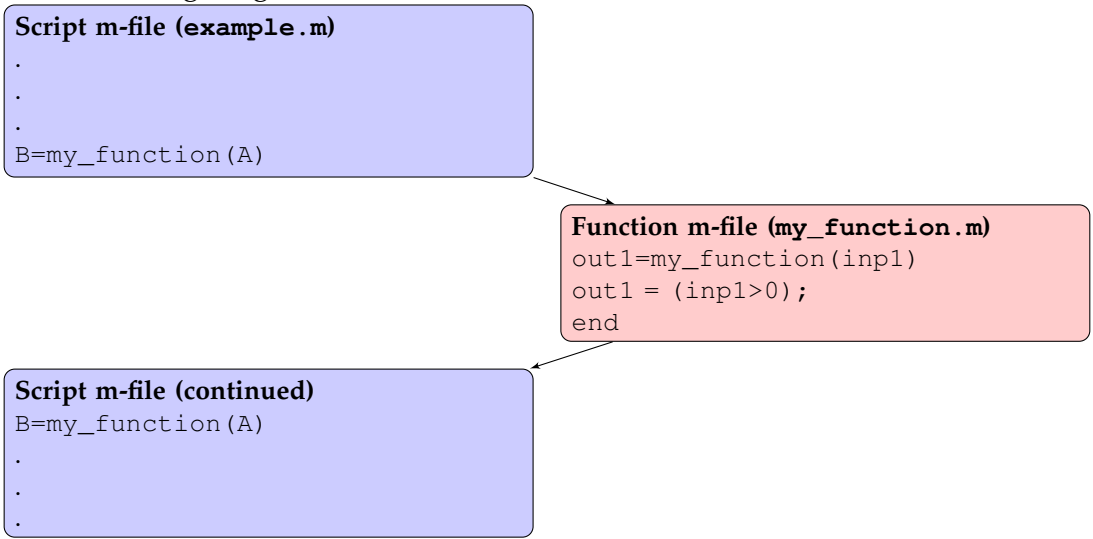
Functions

- Functions are an integral part of Matlab and must be used when estimating some econometric models (e.g. MLE)
- Functions are powerful in their own right because they allow users to replicate code in a systematic fashion
- Function syntax:

```
1 [output1,output2,..,outputN] = function_name(input1,input2,..,inputK)
2 % Comments that describe what the function does
3 [code which user would like to have repeatedly executed and which uses ...
      all inputs and creates all outputs]
4 end
```

Using functions

The following diagram illustrates how functions are called in Matlab:



Using functions - Continued

Helpful reminders when working with functions

- Always name a function m-file the same name as the function title
- Once Matlab enters a function, only function inputs are available in the workspace
  - The exception to this is that all `global` variables are passed to the function
- Once Matlab re-enters the script m-file (after executing the function), only the function output is available (in addition to the original workspace that existed before the function call)
- Other functions can be referenced within functions. Users may also create an arbitrary number of nested functions (i.e. functions within functions)—I typically don't do this

Helpful function variables

- These are only valid inside a function m-file
- Most useful for implementing flow control in a function

```
1 nargin    %lists the number of inputs to the function
2 nargout   %lists the number of outputs from the function
3 varargin  %shorthand for ''all inputs after this point''
4 varargout %shorthand for ''all outputs after this point''
```

Accessing function outputs

- Suppose a user only wants to see the first and fifth outputs of a function
- Matlab 2009 and later allows the use of $\sim$ to suppress function outputs, e.g. `[x,~,~,~,z] = myfun(X,Y,Z);`

Function handles

- Matlab allows users to create functions "on the fly" (without needing to create a m-file)
- A function handle is a function without a name, which usually has a simple expression (and only a few inputs)
- The function handle syntax is `@(x)f(x)`
- e.g. To create a function handle for $f(x) = x^2 + 2x - 8$, the correct syntax would be `@(x)x.^2+2.*x-8`
- Note that function handle operators need to be vectorized (`.*`, `.^`)
- Can also be functions of multiple variables

Variable scope

- As was mentioned previously, when Matlab enters a function it leaves behind the workspace from the file (or function) it came from
- Users can make certain variables available to all functions by issuing a `global` statement at the top of each file
- All variables that are not contained in the `global` statement are not transferred
- Changes to a `global` variable in one function are brought into the the next function, so be careful!
- This is useful in situations where a user needs to pass a large number of variables and doesn't want to create a function with a large number of inputs

Example 1: Three ways to evaluate code

Suppose that a user wants to calculate the product Z of two matrices X and Y. She could do this in three ways:

1. Type into the command line
2. Save as a line in a script m-file
3. Create a function to do the product

Example 1 – Command Line

Supposing X and Y already exist, type `Z=X*Y;` into the command line, and press enter.

Example 1 – Script

Supposing X and Y already exist, type `Z=X*Y;` into line 1 of the editor.

Example 1 – Function

Create a new script and type the following:

```
1  Z = matrixproduct(X,Y)
2  % This function calculates the product of two matrices (X and Y)
3  Z=X*Y;
4  end
```

Save the script as `matrixproduct.m` In a new script, type

```
Z=matrixproduct(X,Y);
```