ECO 503 Computational Precept Notes

Thomas Winberry

September 20th, 2013

1 Introduction

- The basic tradeoff faced by every programmer is between human time, spent writing code, and computing time, spent executing that code
- Good code takes longer to write but shorter to run

  - implication: only spend time refining your code if it will save at least that time over a more simple version of the code
- Different programming languages allow programmers to exploit this tradeoff in different ways:

  - machine language: series of 0's and 1's which are most efficient for the computer to read but hardest for the human to write

  - low-level languages (eg C++, Fortran): some commands which routinize common sequences of 0's and 1's. These take time for the computer to translate back into 0's and 1's.

  - high-level languages (eg Matlab, Python, R, Stata, Java): even more commands that take even more time to translate back to 0's and 1's.
- As a high-level language, Matlab has a lot of general purpose commands that can be used for many different situations. This generality is good for saving programming time but bad for computing time.
- Still, Matlab is better (ie faster) at some things and worse at other
- Your job is to take advantage of what Matlab is good at and avoid using what it is bad at
- What is Matlab good at? Matrices, matrix operations, some linear algebra
- What is Matlab bad at? Everything else

2 Installing Matlab

- Matlab licenses are in general expensive
- Luckily for you, Princeton has already paid for your license and lets you install it on your personal computer

- Go to http://www.princeton.edu/software/licenses/software/matlab/ and follow the instructions to install Matlab and some optional toolboxes. You will need to be on the Princeton network, either directly or indirectly through the VPN.

- Subtle point: actually two separate type of licenses:

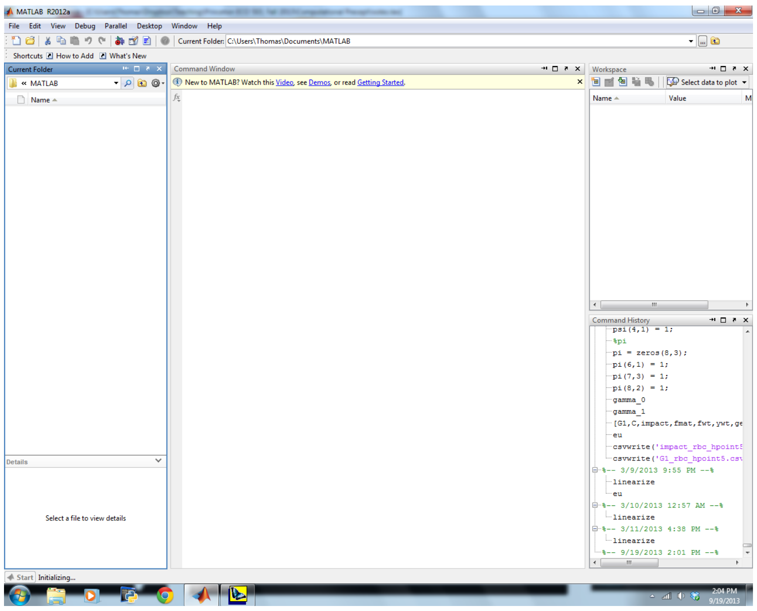
  (1) Core Matlab: do not need to be on the network once installed

  (2) Toolboxes: need to be on the network to use once installed

3 Getting to Know Matlab

- Once you install it, open up Matlab!

3.1 The Windows



New to MATLAB? Watch this Video, see Demos, or read Getting Started.

Select a file to view details

```
psi(4,1) = 1;
%pi
pi = zeros(8,3);
pi(6,1) = 1;
pi(7,3) = 1;
pi(8,2) = 1;
gamma_0
gamma_1
[G1,C,impact,fmat,fwt,ywt,ge
eu
csvwrite('impact_rbc_hpoint5
csvwrite('G1_rbc_hpoint5.csv
%-- 3/9/2013 9:55 PM --%
linearize
eu
%-- 3/10/2013 12:57 AM --%
linearize
%-- 3/11/2013 4:38 PM --%
linearize
%-- 9/19/2013 2:01 PM --%
```

- Current folder: tells you what directory you are working in and what stuff you have stored there. This will come up later when we talk about m files and reading / writing data

- Command window: where commands are entered

- Workspace: objects that you currently have loaded in the memory and can manipulate

- Command history: commands you've already inputted

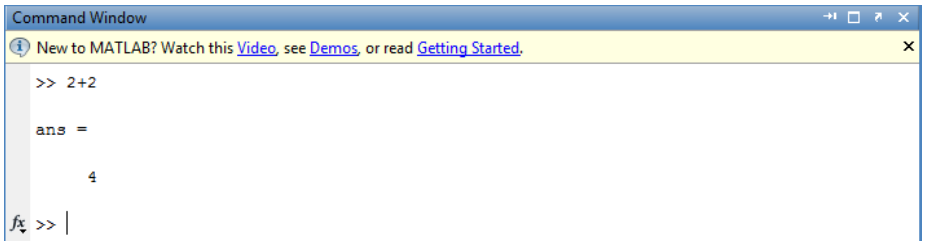
3.2 Commands

- You get Matlab to do something by giving it commands

- One way to give it a command is by typing it directly in the command window

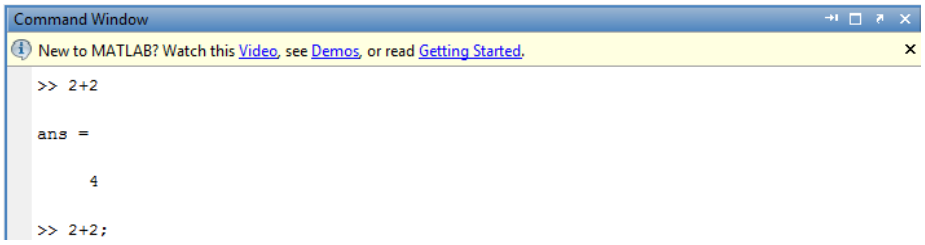
Example 1: Evaluating an Expression

- The easiest kind of command is to tell Matlab to evaluate something
- Here, I ask it to compute $2 + 2$:


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> 2+2

ans =

     4

>> 
```

- Matlab returns an object called "ans" which contains the result of the computation, 4
- This object is now in the workspace
- You can tell Matlab not to print out the answer to by adding a semicolon to the end of your command:


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> 2+2

ans =

     4

>> 2+2;
```

- You can go through your command history at the command line by using the up arrow

4 How Matlab Stores Data

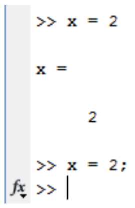
- Anything that is stored in the workspace (ie, the memory) is called "data" in programming terms
- In example 1, the variable "ans" was a type of data: a scalar number
- There are a lot of different types data: scalars, strings, vectors, matrices, arrays, etc.
- We'll go through the types you'll see most often, but remember that there are more that you may find useful in the future

4.1 Scalars

- Scalars are real numbers
- Technically, they can be stored as different types of data depending on whether they are integers, the level of precision, etc., but we'll gloss over that

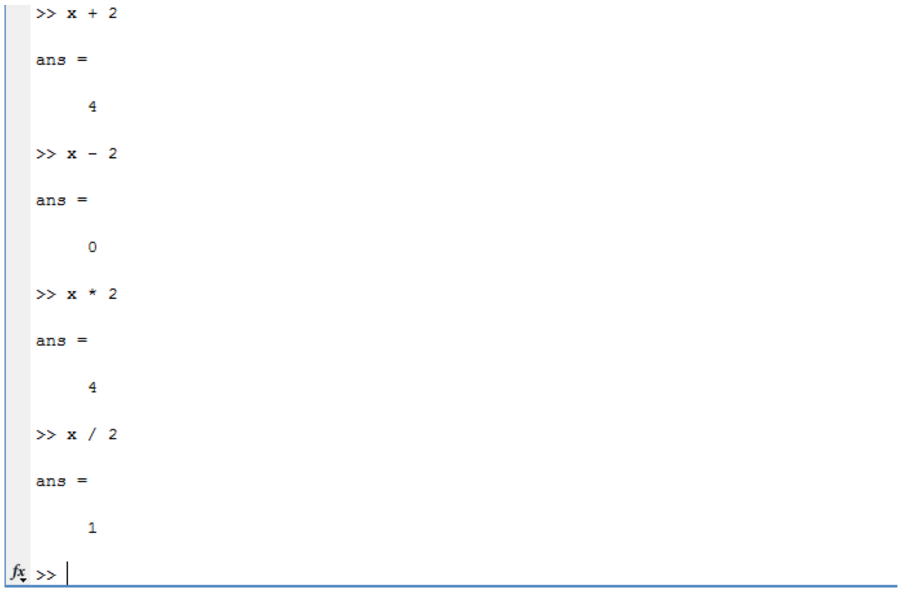
Example 2: Scalar

- Let's create a scalar x which is equal to 2:



```
>> x = 2

x =

     2

>> x = 2;
>> 
```

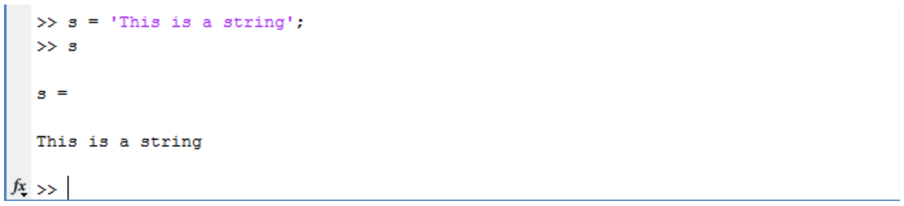
- Now, x is in your workspace and equal to the scalar 2
- You can play around with it by calling the variable x. Try telling Matlab to evaluate some operations involving x:



```
>> x + 2

ans =

     4

>> x - 2

ans =

     0

>> x * 2

ans =

     4

>> x / 2

ans =

     1

>> 
```

4.2 Strings

- Strings are not numerical variables, and you'll probably use them the least in your programming
- To create a string, place single quotes (') around a set of non numerical characters:



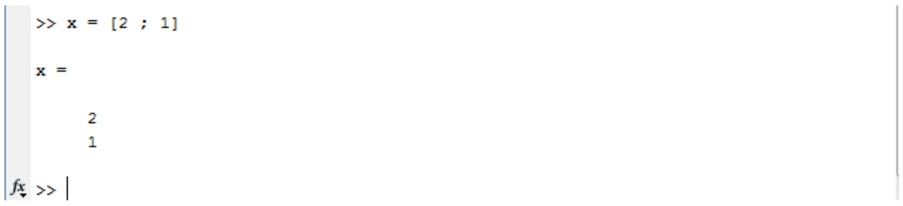
```
>> s = 'This is a string';
>> s

s =

This is a string

>> 
```

4.3 Vectors and Matrices

- Loosely speaking, these data types are groups of scalars
- Vectors have a dimension $N \times 1$ or $1 \times N$
- Matrices have dimension $M \times N$

Example 3: Vector

- To create the 2×1 vector $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, enter $x = [2; 1]$ into the command window:

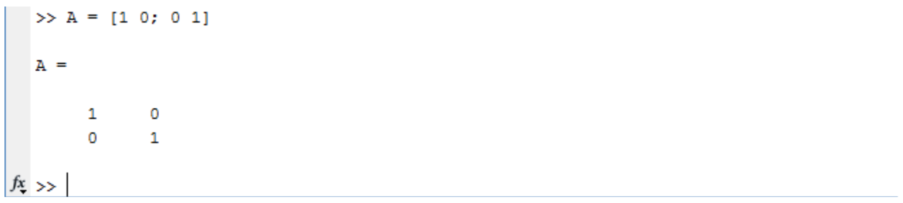


```
>> x = [2 ; 1]

x =

     2
     1

>> 
```

- If you want to create the 1×2 vector $x = (1, 2)$, enter $x = [2\ 1]$ into the command window
- Note that semicolons separate rows
- You will very rarely create a vector by typing it in like this, but now you have a vector x in your workspace. Play around with it!

Example 4: Matrix

- Now lets create the 2×2 matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ by manually entering it:



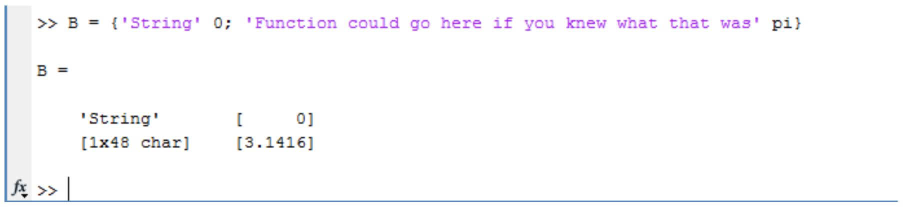
```
>> A = [1 0; 0 1]

A =

     1     0
     0     1

>> 
```

4.4 Cell Arrays

- Cell arrays are just arrays that can hold things other than scalars
- In fact they can hold anything!

Example 5: Cell Arrays

- Enter an array using { and } instead of [and]



```
>> B = {'String' 0; 'Function could go here if you knew what that was' pi}

B =

    'String'         [     0]
    [1x48 char]      [3.1416]

>> 
```

5 Matrices

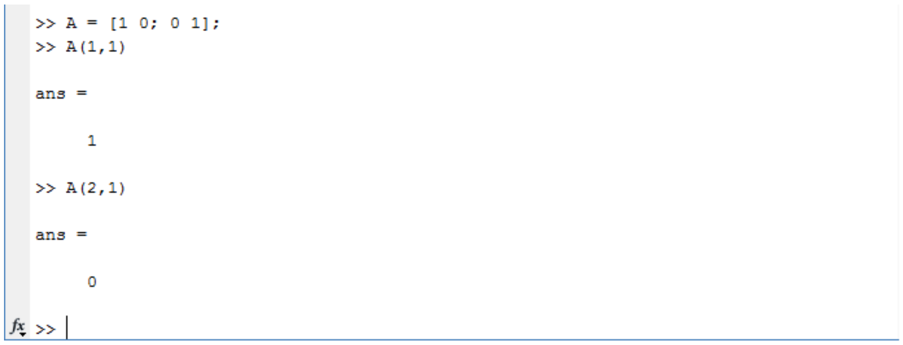
- Matrices are the bread and butter of Matlab
- As I said in the introduction, Matlab is optimized for matrix operations; anything else is much slower
- That means if you can frame what you're doing in terms of a matrix operation, you should do it
- This is more versatile than you might think!

5.1 Accessing Matrices

- Suppose you want to access only certain elements of a matrix

Example 6: Accessing one entry in a matrix

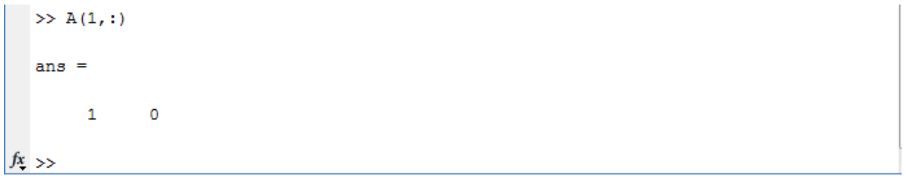
- One option is to call the exact index of the entry you want
- In Matlab, but not most other programming languages, indexes start at 1



```
>> A = [1 0; 0 1];
>> A(1,1)

ans =

     1

>> A(2,1)

ans =

     0

>> 
```

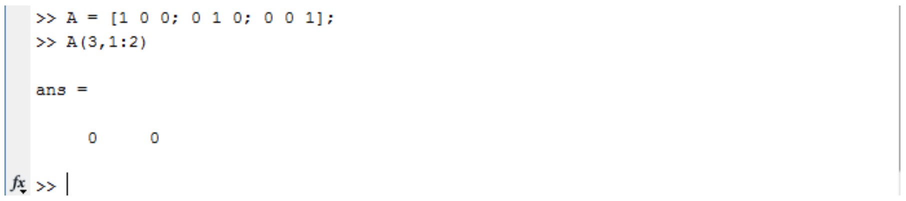
Example 7: Accessing multiple entries in a matrix

- Another option is to access multiple indices at once. We'll do this using something called colon notation.
- Suppose you want to access the first row of A. Just ask for $A(1,:)$



```
>> A(1,:)

ans =

     1     0

>> 
```

- The colon tells Matlab to get all the entries along that dimension
- You can also ask for a subset of (connected) indices. For example, lets make A the 3×3 matrix $A = [1\ 0\ 0; 0\ 1\ 0; 0\ 0\ 1]$ (without putting this into Matlab: which matrix is this?)
- Now suppose you want the first two entries in the third row; just ask for $A(3, 1:2)$:



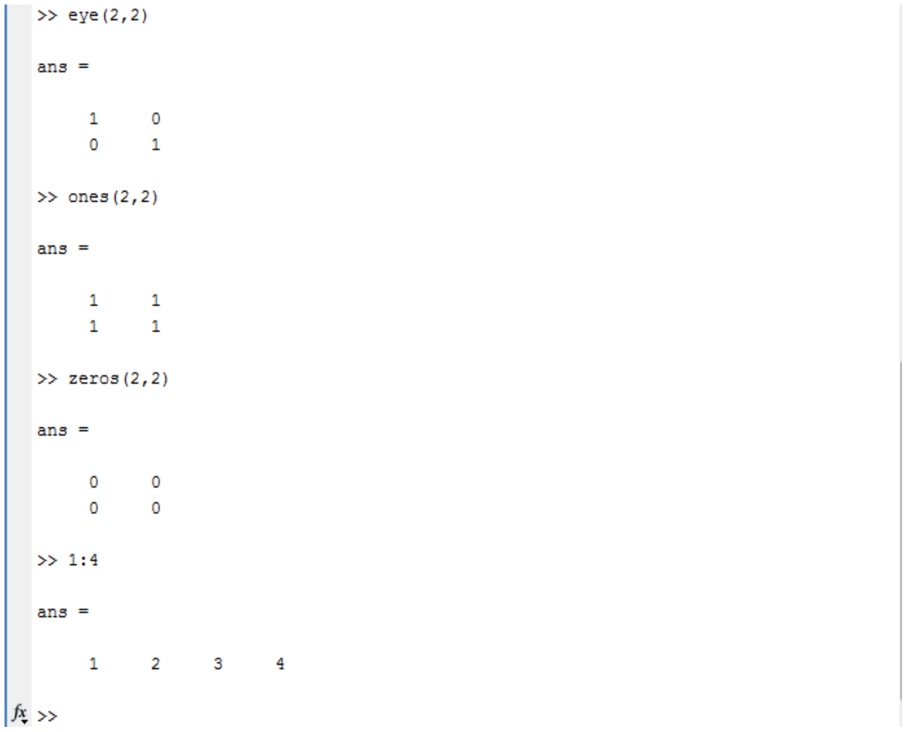
```
>> A = [1 0 0; 0 1 0; 0 0 1];
>> A(3,1:2)

ans =

     0     0

>>
```

5.2 Creating Matrices

- You will very rarely create matrices by hand like we did above
- You will instead generate matrices by manipulating some of Matlab's built in functions
- These are the building blocks of the operations you will do

Example 8: Creating matrices

- To create an $N \times M$ identity matrix, use $eye(N, M)$
- For a matrix full of 1's, use $ones(N, M)$
- For a matrix full of 0's, use $zeros(N, M)$
- For the $1 \times N$ vector $(1, 2, 3, ..., N)$, use $1 : N$



```
>> eye(2,2)

ans =

     1     0
     0     1

>> ones(2,2)

ans =

     1     1
     1     1

>> zeros(2,2)

ans =

     0     0
     0     0

>> 1:4

ans =

     1     2     3     4

>>
```

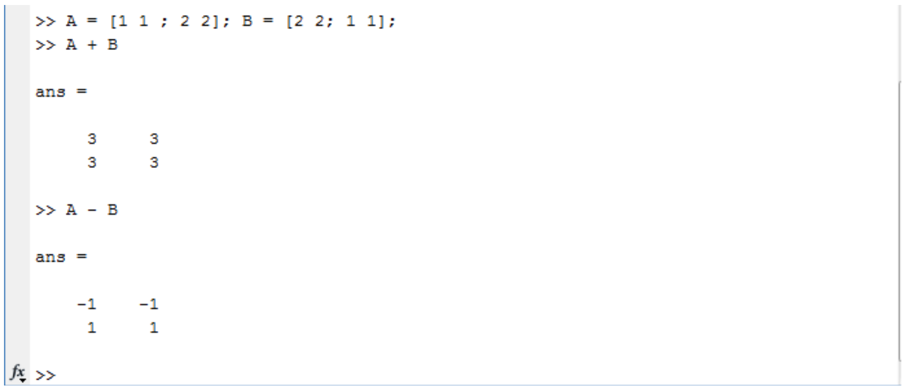
- There are many other functions to take advantage of; check them out!

5.3 Matrix Operations

- Now lets create some new data by manipulating pre-exististing matrices
- Again: this is the bread and butter of Matlab programming. I can't emphasize this enough!
- The same operations apply of course to vectors since in principle they are all the same thing
- Throughout the examples below, let $A = [1\ 1; 2\ 2]$ and $B = [2\ 2; 1\ 1]$

Example 9: Matrix addition and subtraction

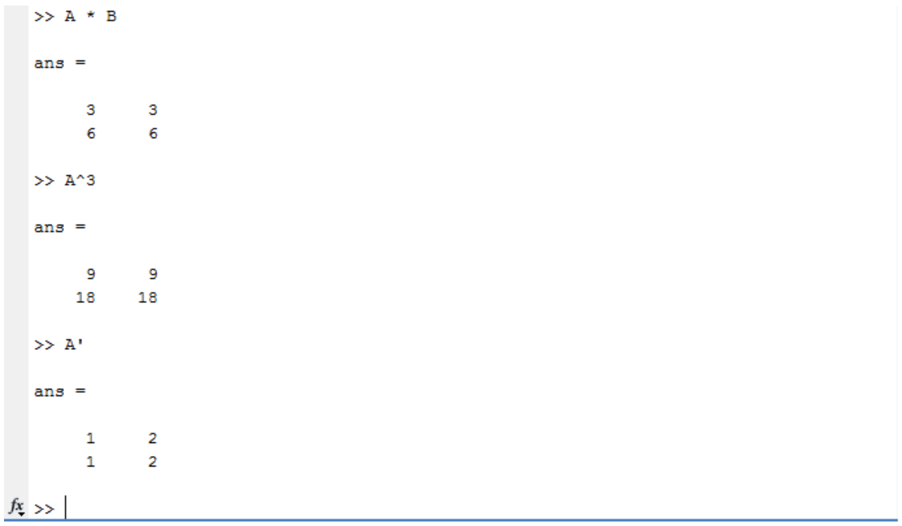
- Addition and subtraction act elementwise on matrices; that is, it just adds and subtracts the individual elements one by one



```
>> A = [1 1 ; 2 2]; B = [2 2; 1 1];
>> A + B

ans =

     3     3
     3     3

>> A - B

ans =

    -1    -1
     1     1

>>
```

Example 10: Matrix multiplication, power, and transpose

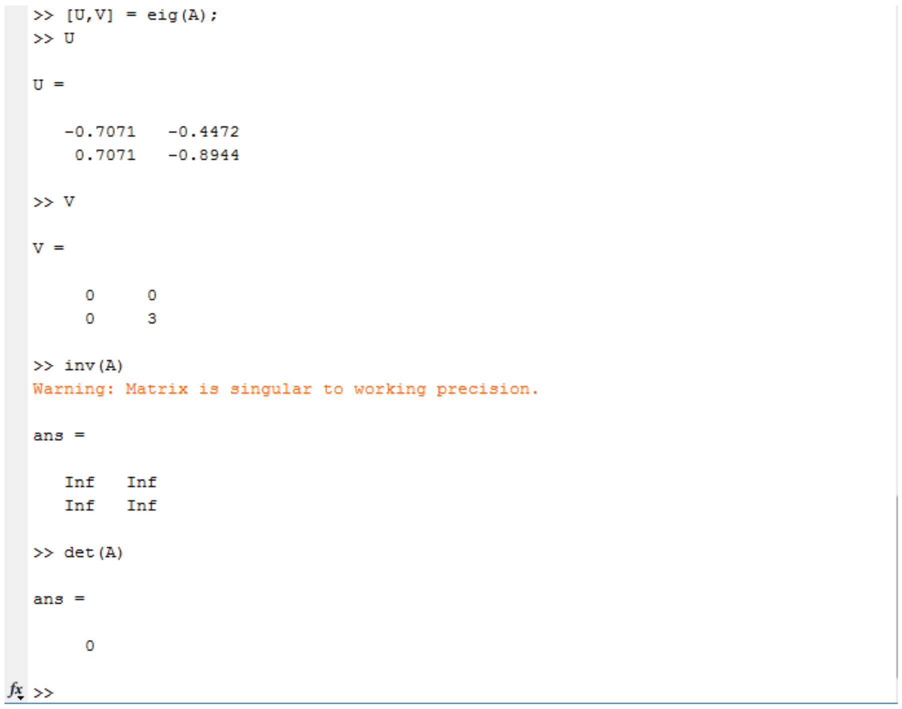
- Many operations do not act elementwise
- Most basic: matrix multiplication, performed by $*$
- Another is the matrix power ˆ or transpose $'$
- Again, many options here, so explore



```
>> A * B

ans =

     3     3
     6     6

>> A^3

ans =

     9     9
    18    18

>> A'

ans =

     1     2
     1     2

>>
```

Example 11: Matrix operations

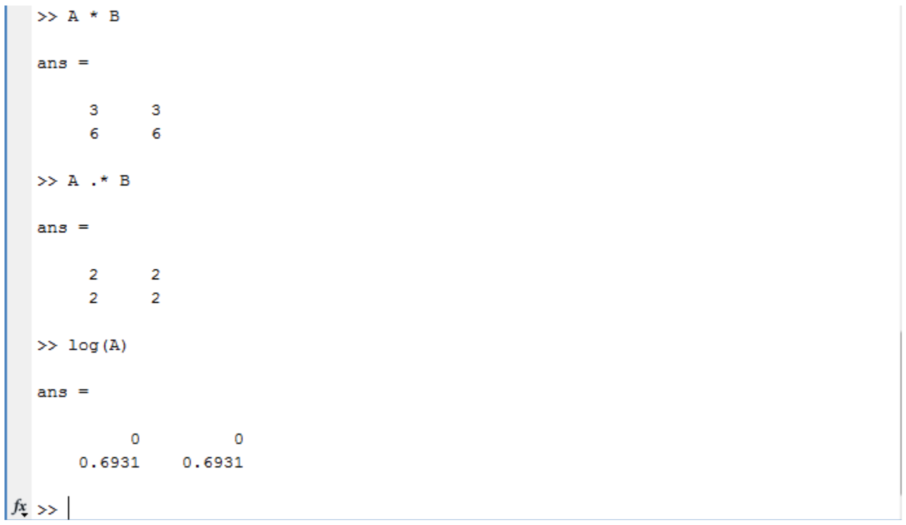
- Matlab also has a lot of built in operations that act on entire matrices; what follows is a very small sample of them
- $eig(A)$: computes the eigenvalues and eigenvectors of A
- $inv(A)$: computes the inverse of A
- $\det(A)$: computes the determinant of A
- A was a poor choice to illustrate these functions...



```
>> [U,V] = eig(A);
>> U

U =

   -0.7071   -0.4472
    0.7071   -0.8944

>> V

V =

     0     0
     0     3

>> inv(A)
Warning: Matrix is singular to working precision.

ans =

   Inf   Inf
   Inf   Inf

>> det(A)

ans =

     0

>>
```

Example 12: Some entrywise matrix operations

- Some operations are entrywise, which act on the individual entries of a matrix
- The basic matrix operations, like multiplication and power, can also be performed entrywise by adding a period (.) in front of the operator: $.*$, $.\hat{}$, etc
- Most built in mathematical functions, like log or exp also act elementwise

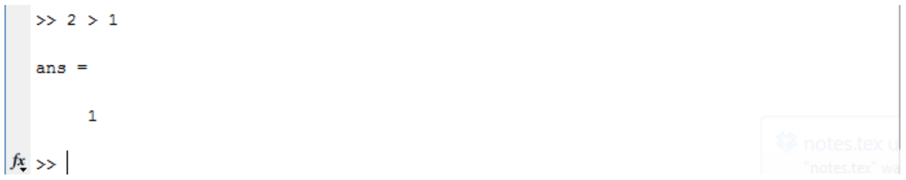

```
>> A * B

ans =

     3     3
     6     6

>> A .* B

ans =

     2     2
     2     2

>> log(A)

ans =

         0         0
    0.6931    0.6931

>> 
```

6 Boolean Statements and Loops

- A lot of times you will run into situations where you want to perform and operation only if a certain condition is satisfied
- So we need a way to think about the truth of certain statements
- These are called boolean statements

Example 13: Boolean statements

- What happens if you type 2 > 1 into the command window?
- This doesn't ask Matlab to return a number; instead, it tells you whether the statement is true or not (this is actually stored as an integer, a 1 if true or 0 if not)


```
>> 2 > 1

ans =

     1

>> 
```

- Of course there are other relations:

  >=: greater than or equal to

  <: strictly less than

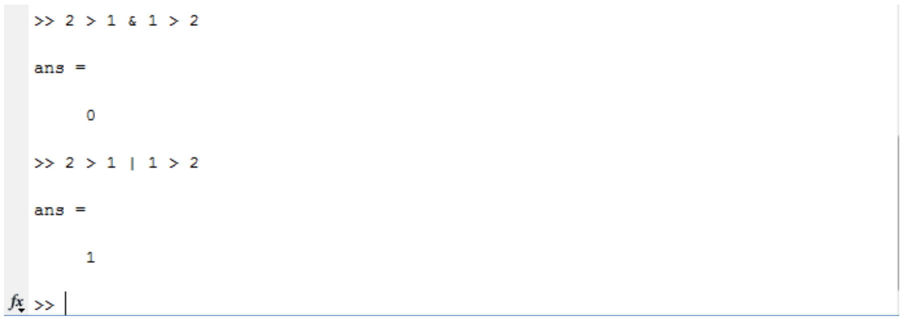
  <=: less than or equal to

  ==: equals (note there are two equal signs!)

  ˜: not

Example 14: Connecting boolean statements

- You can also connect boolean statements using & (and) or | (or)



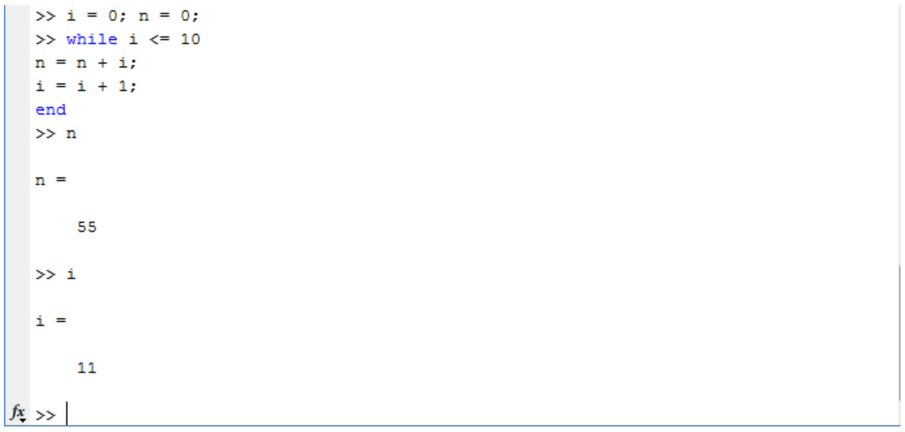
```
>> 2 > 1 & 1 > 2

ans =

     0

>> 2 > 1 | 1 > 2

ans =

     1

>>
```

6.1 Loops

- Loops are commands for Matlab to execute a given set of commands as long as a given boolean statement is true
- There are three different types: while loops, for loops, and if "loops"
- Most operations can be performed using any kind, but they have different strengths and weaknesses to keep in mind
- You can nest each of these different types of loops
- Avoid using loops if possible! They are not matrix operations and will tremendously slow down your code.

Example 15: While Loops

- While loops execute a given set of commands *while* a given boolean statement is true
- You state the condition by typing "while (boolean statement)"
- You then add the sequence of commands to execute under the while statement
- When you want the set of commands to stop, type "end"
- As an example, lets find the sum of the first 10 integers:

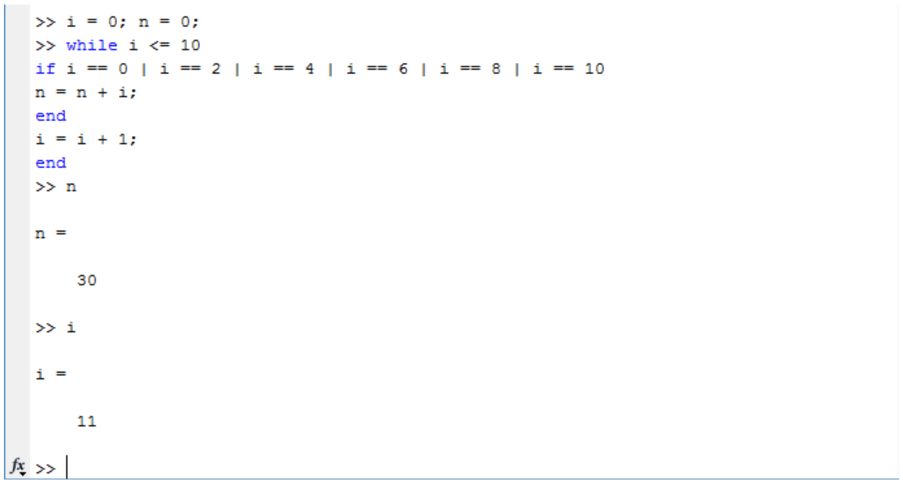


```
>> i = 0; n = 0;
>> while i <= 10
n = n + i;
i = i + 1;
end
>> n

n =

    55

>> i

i =

    11

>>
```

- Can you figure out what is going on the above code?
  - i: an integer variable that I use to index which of the integers I'm at
  - n: another integer variable that I use to store the sum.
- At each step, I update the value of n using the command "$n = n + i$". You couldn't get away with this in math unless $i = 0$, so what's going on here?
- It has to do with how Matlab reads the command.
  - first: reads the right hand side, $n + i$, and stores it in temporary memory
  - second: creates a new variable, n, which is equal to what is in the temporary memory
- This second step overwrites the old value of n with its new value

Example 16: If "Loops"

- If loops execute a given set of commands *if* a given boolean statement is true
- The structure is the same as a while loop: first type "if (boolean statement)," then a set of commands, then "end."
- Lets put in our while loop to create the sum of the first ten even integers:



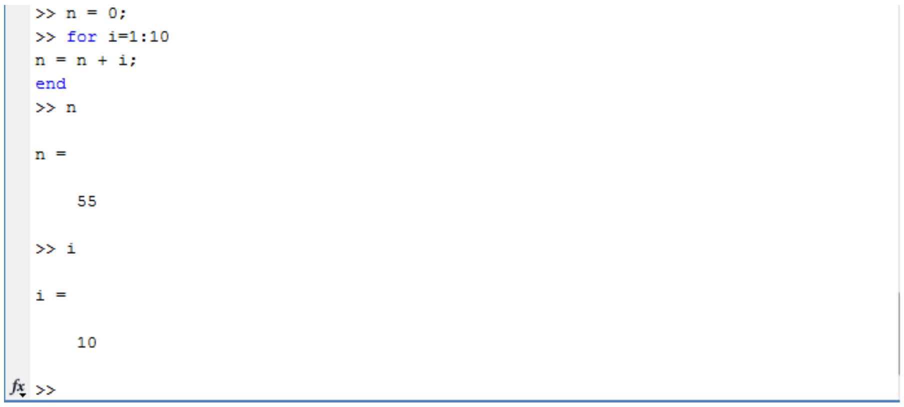
```
>> i = 0; n = 0;
>> while i <= 10
if i == 0 | i == 2 | i == 4 | i == 6 | i == 8 | i == 10
n = n + i;
end
i = i + 1;
end
>> n

n =

    30

>> i

i =

    11

>> 
```

- In this example, we put an if statement in the while loop; for each value of i the loop went through, it checked if i was even before performing the n = n + i operation
- Note that each boolean statement requires an end after the set of commands
- And of course, there are more efficient ways to find even integers

Example 17: For Loops

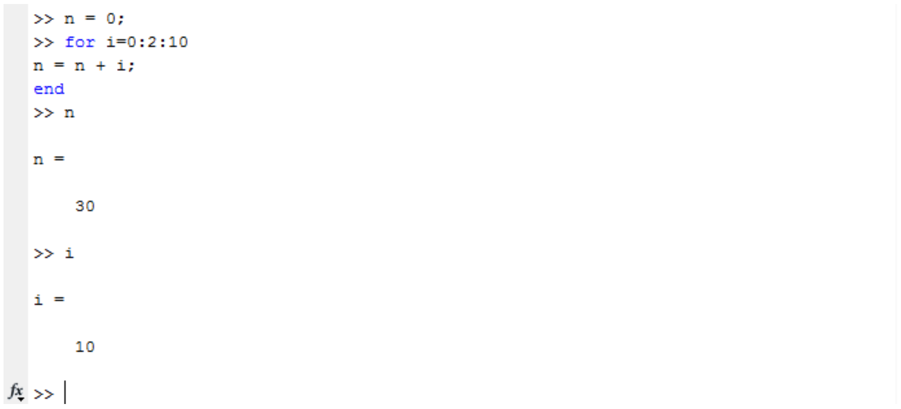
- Advance warning: for loops are not commands executed as long as a given boolean statement is true. But they share the same structure as while and if loops, so I include them here
- For loops execute a given set of commands *for* values of an "indexing variable" in some set

- A simple way to do this is to create an $N \times 1$ vector of increasing integers using the colon notation from above
- Now we're going to go through each entry of that variable and perform an operation for it
- Again, lets compute the sum of the first ten integers:



```
>> n = 0;
>> for i=1:10
n = n + i;
end
>> n

n =

    55

>> i

i =

    10

>>
```

- We can modify the list of integers i goes through to easily get the sum of the first ten even integers:



```
>> n = 0;
>> for i=0:2:10
n = n + i;
end
>> n

n =

    30

>> i

i =

    10

>> |
```

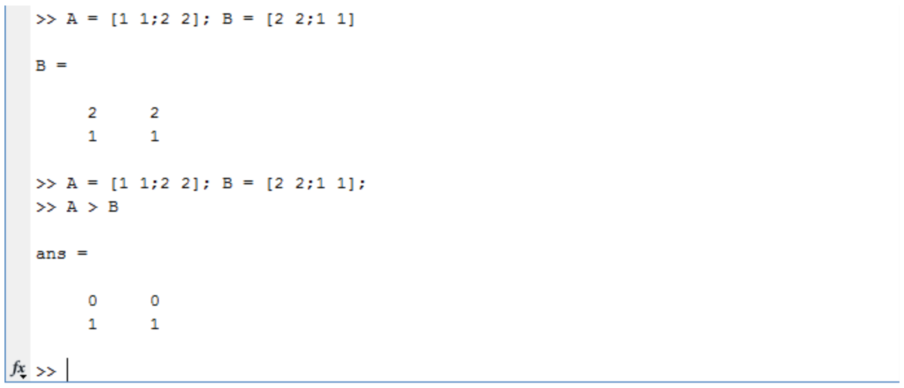
- The command "0:2:10" tells Matlab to make a vector with values going from 0 to 10, adding 2 each time; that is, 0:2:10=(0,2,4,6,8,10)

6.2 Matrices and Boolean Indexing

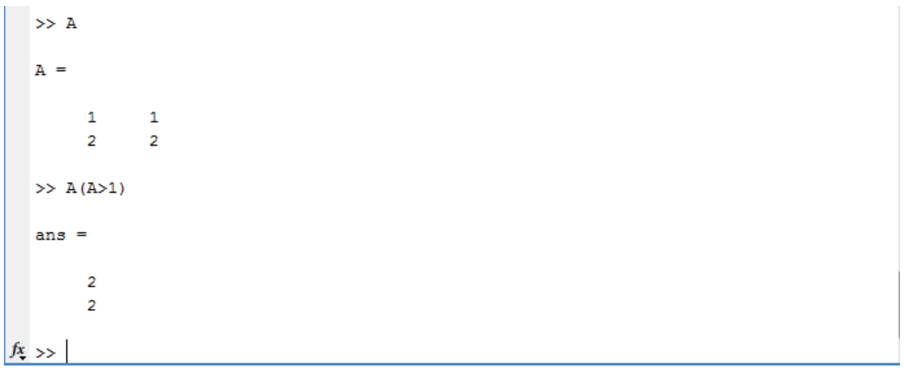
- Boolean statements can also involve matrices with the understanding that they are evaluated elementwise

Example 18: Matrix boolean statement

- A simple example is to look at the statement $A > B$:


```
>> A = [1 1;2 2]; B = [2 2;1 1]

B =

     2     2
     1     1

>> A = [1 1;2 2]; B = [2 2;1 1];
>> A > B

ans =

     0     0
     1     1

>> 
```

- Note that this returns another matrix, which is 1 where the boolean statement is true and 0 where its false
- A nice trick is to use Boolean statements to return entries of a matrix where the statement is satisfied
- This is probably easier shown in an example first:


```
>> A

A =

     1     1
     2     2

>> A(A>1)

ans =

     2
     2

>> 
```

- This is really the composition of two operations. First, Matlab figures out which indices of the matrix A satisfies $A > 0$. Second, Matlab returns the values of A corresponding to those indices

7 M Files

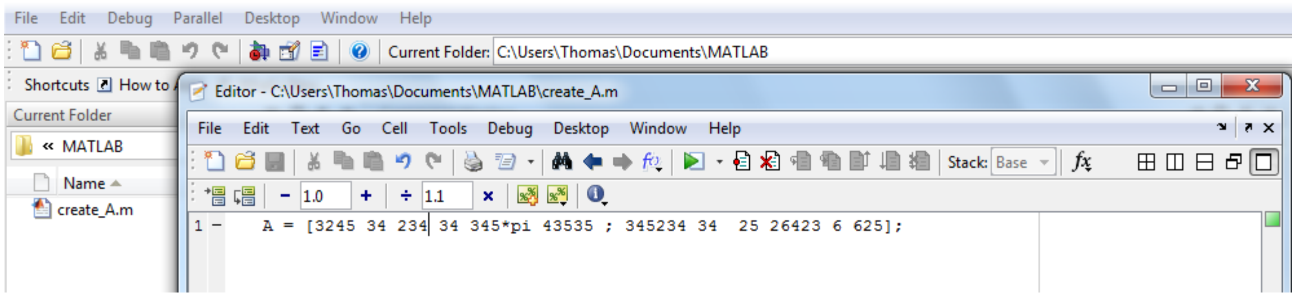
- We are already hitting the limit of what you can directly type into the command window
- Luckily, you can save series of commands into files called m files (so called because they end in ".m")
- Generally there are two types of m files: scripts and functions; we'll go over both
- If you want to call a particular m file (more on that in a minute), it needs to be in your working directory. Remember that's the window on the left of your screen.
- You can change which directly you're working in by changing that folder in the window or by typing commands

7.1 Scripts

- Scripts are series of commands that don't require any external input; they can run entirely on their own
- To call an m file script in your working directly, just type its name
- For example, if you have "matrix.m" in your directly, type "matrix" into the command window to call the file

Example 19: Scripts

- So how do you write an m file?
- From within Matlab, you can click on the new m file button to open up a simple text editor
- As you get more sophisticated, there are much better text editors out there, but for now Matlab's built in is fine
- The syntax for creating a complicated matrix A is below:


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\Thomas\Documents\MATLAB
Shortcuts How to
Current Folder
MATLAB
Name
create_A.m
Editor - C:\Users\Thomas\Documents\MATLAB\create_A.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1

```
A = [3245 34 234 34 345*pi 43535 ; 345234 34  25 26423 6 625];
```

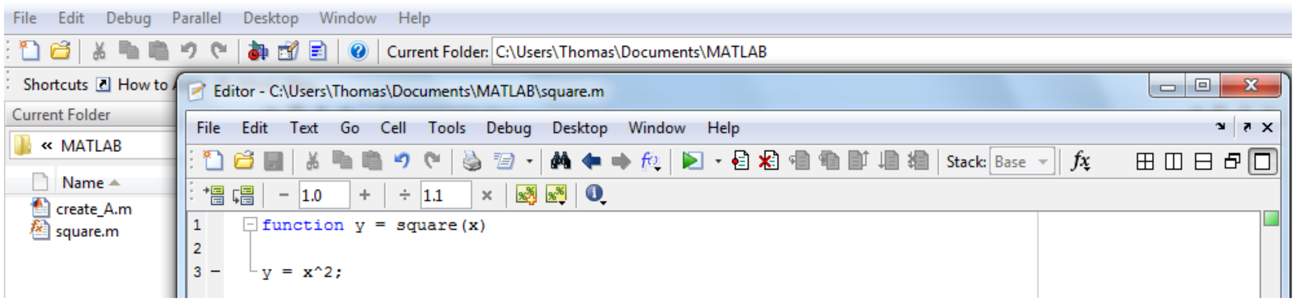
- Although this example is contrived, even here creating A via this script is nice because its very easy to edit A inside the m file rather than the command window
- Now lets run this script – which I've called "create_A.m" and create A in our command window by typing create_A. A will now show up in the workspace.

7.2 Functions

- Functions are series of commands that require an external input
- In practice you will use these all the time

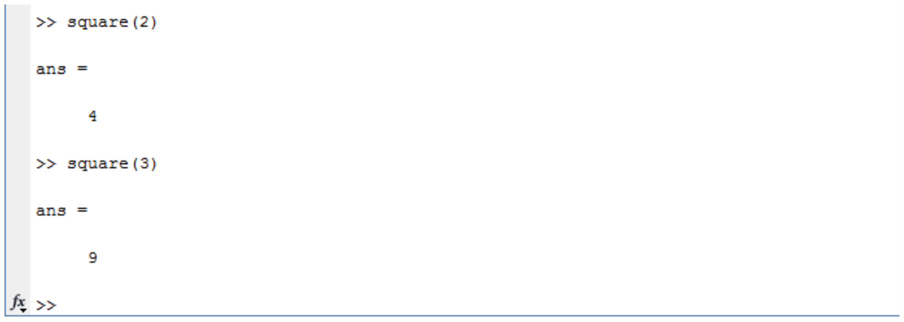
Example 20: Easy function

- A simple example is the function $f(x) = x^2$
- Here, x is the input, and the function f returns the squared value of x
- Here's the syntax to write this function as an m file:


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\Thomas\Documents\MATLAB
Shortcuts How to
Current Folder
MATLAB
Name
create_A.m
square.m
Editor - C:\Users\Thomas\Documents\MATLAB\square.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base

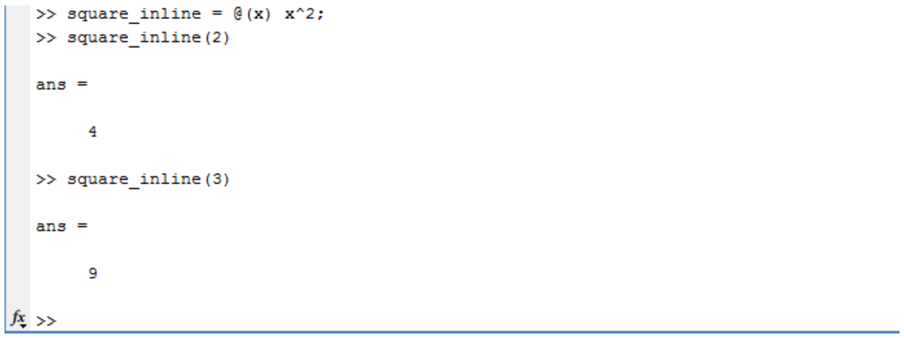
```
function y = square(x)

y = x^2;
```

- To declare an m file as a function, you need to write "function" on the first line, followed by what the function will output (here, y) equal to the function name and inputs (here, x)
- Then you write your commands. Here, I'm just assigning the value of the output y as the value of the input squared
- More complicated functions can have multiple inputs and outputs, or have different types of inputs and outputs (like matrices, strings, or even other functions)
- To call the function "square.m" from Matlab, I just do the following:



```
>> square(2)

ans =

     4

>> square(3)

ans =

     9

>>
```

- As an aside, you can write simple functions like this directly into the command line using something called "in-line" functions:

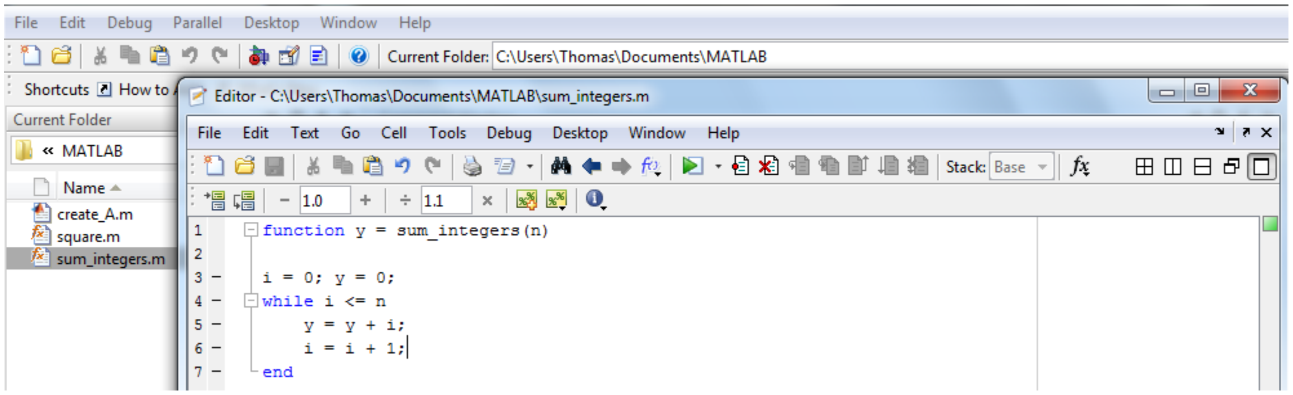


```
>> square_inline = @(x) x^2;
>> square_inline(2)

ans =

     4

>> square_inline(3)

ans =

     9

>>
```

- Here, the "@(x)" command tells Matlab that we're defining a function with inputs x; the rest then defines what the function returns

Example 21: Harder function

- Lets now do something harder: write a function that computes the sum of the first n integers, where the integer n is an input into the function.


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\Thomas\Documents\MATLAB
Shortcuts How to
Current Folder
MATLAB
Name
create_A.m
square.m
sum_integers.m
Editor - C:\Users\Thomas\Documents\MATLAB\sum_integers.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1

```
function y = sum_integers(n)

i = 0; y = 0;
while i <= n
    y = y + i;
    i = i + 1;
end
```

- Now if I want this sum I can just call the function from the command window:

```
>> sum_integers(2)

ans =

     3

>> sum_integers(20)

ans =

   210

>> sum_integers(200)

ans =

       20100

>> 
```

- Be an imaginative! Functions are very versatile

8 Miscellaneous

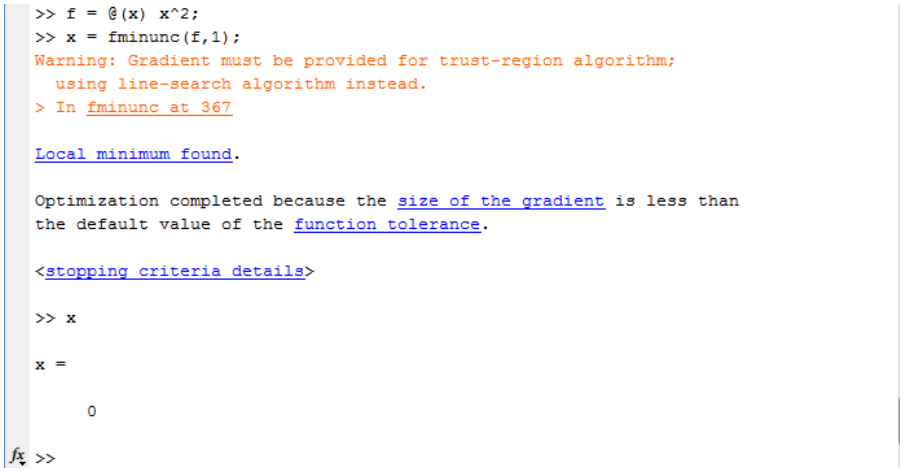
- Here I've collected some very useful things that Matlab can do that don't fit into the above categories

8.1 Operations on Functions

- Matlab also has many operations that you can perform on functions that you've written

Example 22: Minimization

- A simple example is function minimization
- Suppose we take our in-line function f =@(x) x^2 and want to find the value of x which minimizes f
- Lets use the operation "fminunc:"

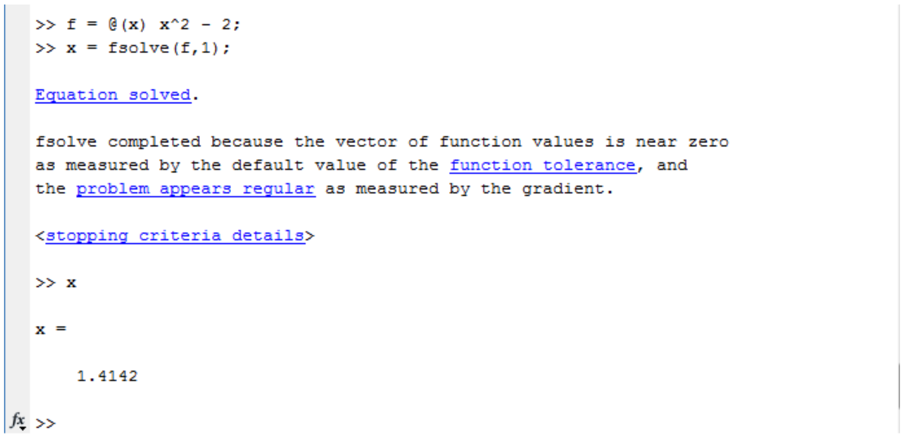


```
>> f = @(x) x^2;
>> x = fminunc(f,1);
Warning: Gradient must be provided for trust-region algorithm;
  using line-search algorithm instead.
> In fminunc at 367

Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

<stopping criteria details>

>> x

x =

     0

>>
```

- The function "fminunc" takes as inputs the function to be minimized (here, f) and an initial guess (here, 1).
- There are other options you can as inputs to fminunc as well. Check the help files or online resources for more information.
- Of course, you can also apply fminunc to functions you've written as m files
- There are a huge number of optimization functions in Matlab, each of which are better in some situations than others.
- If you're solving a hard optimization problem, go through the help files or online resources to find the one which is best fit to your situation

Example 23: Root finding

- Another common example is root finding, ie, equation solving
- Root finders find the value of x such that the function $f(x) = 0$
- Any system of equations can be rewritten in this form by bringing all terms to one side of the equation
- Lets solve f=@(x) x^2 - 2 using "fsolve:"



```
>> f = @(x) x^2 - 2;
>> x = fsolve(f,1);

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

>> x

x =

    1.4142

>>
```

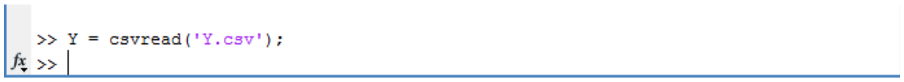
- Again, you can apply this to m files as well, and again, there are many types of root finders
- A word of caution: these general purpose operations on functions are not matrix operations, so they're not what Matlab is good at. Do not expect them to be efficient.
- One except to the above rule: Matlab is good at linear operations. So if you can make your inherently nonlinear equations look close to linear, you can expect it to work better.
- In the contrived example of root finding above, instead of using $x^2 - 2 = 0$, you would want to use $x - \sqrt{2} = 0$
- You can often do this in economic problems by taking logs
- Other Matlab commands are better suited at solving linear equations than fsolve

8.2 Reading / Writing Data

- We already saw one way to create a complicated data object was to code it up in a script m file and let it run
- But suppose someone gives you data as another file type; what are you supposed to do?
- Luckily, Matlab can read many of these file types

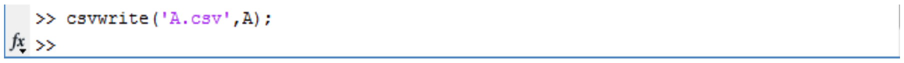
Example 24: .csv Files

- One of the most common and easiest to use of these types is .csv
- I've put a time series of real GDP in my Matlab directory, called Y.csv, and want to create a vector Y which contains that information
- I just have to use the "csvread" command:



```
>> Y = csvread('Y.csv');
>>
```

- If I want to save some of my matrices as csv files for other people, I instead use "csvwrite:"

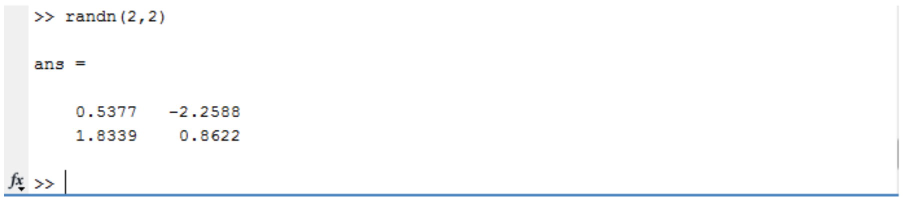


```
>> csvwrite('A.csv',A);
>>
```

- This command is good if you manipulate some data set and want to give that manipulated data to a friend (or another program of your own)

8.3 Random Variables

- Often in economics, you need to be able to simulate the external shocks to the model
- This means you need to be able to generate random variables
- There is a "philosophical" debate about whether this is truly possible on computers, but lets just pretend that you can (which seems to be a good approximation to reality anyway)
- Matlab can generate random variables which follow many different distributions



```
>> randn(2,2)

ans =

    0.5377   -2.2588
    1.8339    0.8622

>>
```

Example 25: Normally Distributed Variables

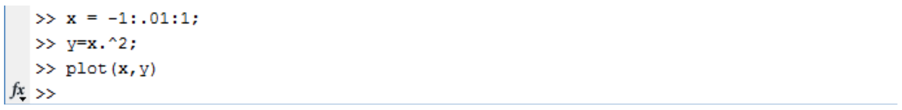
- Lets focus on normally distributed random variables, probably the most common you will see
- To create an $M \times N$ matrix of values drawn from the standard normal distribution, use "randn(M,N):"

8.4 Plots

- Matlab has very powerful plotting tools that can produce different types of plots for different types of data and offerss a huge degree of customization
- I'll just go through an extremely simple example and refer you to the help file or online resources for all of the other options

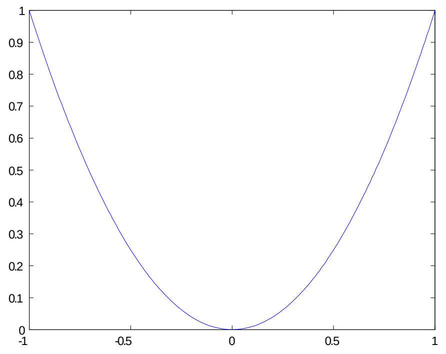
Example 26: Line Plots

- Suppose we want to plot the function $f(x) = x^2$
- Matlab doesn't have a command to just plot a function; instead, it plots data
- So, we'll plot a vector y on the vertical axis that is filled with the squared values of a vector x on the horizontal axis
- The commands to produce this plot are



```
>> x = -1:.01:1;
>> y=x.^2;
>> plot(x,y)
>>
```

- The plot itself is



9 Conclusion: Programming in Matlab

- I've only presented the basics of programming in Matlab, and is meant to help you get a feel for the environment
- As has been hinted at throughout, there are many other things that Matlab can do
- Find out what those things are! Matlab's help files or google searches can be very helpful
- Remember to focus on what Matlab does well: matrix operations
  - avoid loops!
  - if you think you need a loop, you're probably not thinking hard enough
- When writing code, make it readable both for yourself and others who might want to use it
  - lots of comments
  - descriptive variable names
  - re-use code where possible instead of repeating yourself
- Learning to program is a process of trial and error. But you have to get your hands dirty and try writing real code if you are going to learn anything.
- Increase your knowledge of what Matlab can do. If you can think of an operation you want done, its probably in Matlab somewhere. Look it up in the help files or online.