MFE Programming Workshop Class 1

Robert Richmond

November 2, 2015

UCLA Anderson

Goals

- Learn to program in R and in Matlab
- What does programming mean?
    - Language syntax
    - Debugging
    - Finding solutions
    - Translating math to code
- This is just the beginning, you'll develop these skills throughout the program

R vs Matlab

- Both are useful and you will use both in the MFE program
- My view:
  - R is good for data munging, statistics, regressions, etc.
  - Matlab is good for simulations, numerical solvers, etc.
- This workshop will demonstrate these differences

Structure

- I will talk for 30-60 minutes at the beginning of each class
- For the remainder of the time you will break into groups and work on programming tasks
- Tasks are designed to introduce you to the building blocks that will be used for course assignments throughout the MFE program
- This course is a programming course with emphasis on methods for finance:
    - You *will* see finance terms and math
    - You *may* not understand all of the finance, but you will learn it throughout the program
- The key skills will be translating mathematical algorithms into code and developing the ability to find helpful resources

Questions

Any questions before we start?

The interface

- Matlab is more than just a programming language
- Lets take a look at the interface

Hello world

- I can't break programming tradition!

Code

```
disp('hello world')
```

Output

```
hello world
```

Documentation

- The command `help` will be very useful
  - try `help disp` now to get information on the `disp` function we just used
- *very* Useful resources can be found under the help menu including programming tutoritals
- A similarly useful command is `doc`
  - `doc disp`

Actually writing code

- Matlab has you structure code in *.m* files
    - Scripts (now), functions (later)
- Click the new script button of press File-New and create a new script
- Type in the code examples and run them as we go
    - Highlight the region and select evaluate selection

Variables and operators

- Assignment is done using =
- Matlab works like a fancy calculator
- using ; suppreses the output of a given line
- You can use ; to put multiple statements on a line

Example 1

```
x = 1;
x+1
```

Example 2

```
x=3; y=4;
x*y
```

Comments

- Get in the habit of commenting your code
  - Other people have to read and understand it
  - You have to read it and understand if 1 year down the road
- Comments start with %

Example

```
% declare a variable
x = 2;
% operate on it
x*2
```

Matrices

- *most* objects in matlab are matrices/vectors
- Create vecotors or matrices using `[ stuff ]`

Example 1

```
mymatrix = [1 2;
            3 4;
            5 6;];
mymatrix*2
```

Special Matrices

- Some special matrices can be created using `eye, NaN, zeros, ones`

Example

```
N = 4;
myidentity = eye(N)
ans = myidentity
```

Special Matrices

- `eye(N)` is the identity matrix of size N*N
- `NaN` will create a matrix with elements that are "Not a number". This is useful for initilazing variables before use
- `zeros` is a matrix of zeros
- `ones` is a matrix of ones
- `repmat` is incredibly useful creating matrices are are replicated multiple times in a given dimension

Special Matrices

- You can pass multiple parameters to these functions

Example

```
N = 4;
M = 3;
mymat = zeros(N,M)
ans = mymat
```

The : operator

- You can create sequences of numbers with :
- You can use two : operators to create sequences skipping elements

Example

```
x = 1:5;
ans = x
```

Example 2

```
y = 1:2:10;
ans = y
```

Accesing matrix elements (1)

- Using () you can access matrix subsets
- Indexes are rows followed by columns

Example

```
A = [1 3;
     8 4;
     6 2];
A(1,2)
```

Accesing matrix elements (2)

- You can use : to access multiple elements
- : by itself means all elements in that dimension

Example

```
A = [1 3 8;
     8 4 4;
     6 2 5];
A(:,1:2)
```

Accesing matrix elements (3)

- `end` accesses to the end of that dimensions

Example

```
A = [1 3 8;
     8 4 4;
     6 2 5];
A(2,2:end)
```

Accesing matrix elements (4)

- You can also assign to elements

Example

```
A = zeros(3,3);
A(2,:) = 5;
ans = A
```

Combining matrices

- You can combine matrices with [ ]

Example

```
A = eye(3);
B = zeros(3,4);
out = [A B];
ans = out
```

Matrix Operations (1)

- Operators + and - work element-wise on matrices

Example

```
A = eye(3);
A - 1
```

Matrix Operations (2)

- * is matrix multiplication
  - Dimensions need to be correct!

Example

```
A = magic(3);
B = ones(3);
ans = A*B
```

Matrix Operations (3)

- .* and ./ operate element wise

Example

```
A = eye(3);
ans = A./2
```

Matrix Operations (4)

- .* and ./ operate element wise

Example

```
A = eye(2);
B = [1 2;
     3 4];
ans = A./B
```

Matrix Operations (5)

- We can solve equations using / and \
- Consider the matrix equation $Ax = b$

Example

```
A = [1 2;
     3 4;
     5 6];
b = [5; 4; 3];
x = A\b;
ans = x
```

Matrix Operations (6)

- You can invert matrices with `^(-1)` or with `inv`

Example

```
A = [1 2 6;
     3 4 8;
     5 6 9];
ans = inv(A)
```

Functions

- Matlab has countless functions that are already written for you
- `sin, cos, abs, max, min, ...`
- See `doc functionname` for details on these functions

Function examples (1)

- You can use sum to get a sum of matrix elements across a dimension
- For example get the sum of the magic matrix down rows

Example

```
A = magic(4);
ans = sum(A,1)
```

Function examples (1)

- Get the max element of a vector

Example

```
myvec = [1;2;6;2;4;8;5];
mymax = max(myvec);
ans = mymax
```

Function examples (2)

- Get the max element of a vector
- AND its position
- What is going on here?
    - `max` actually returns multiple values, I assign these to a vector
    - the second value returned is the index of the maximum

Example

```
myvec = [1;2;6;2;4;8;5];
[mymax myidx] = max(myvec);
ans = myidx
```

Function examples (3)

- `size` is useful for finding the size of a matrix

Example

```
A = ones(3,5);
[M N] = size(A);
ans = M
```

Conditionals

- Matlab allows for conditional statements using `if`
- The operator == tests for equality
    - that is *two* = signs
    - This is different than assignment with =

Example

```
x = -10
% create your own abs
if(x < 0)
    myabs = -x
else
    myabs = x
end
ans = myabs
```

Looping (1)

- Loops can be created using `for` and `while`

Example

```
x = 0;
for i = 1:10
    x = x+i;
end
ans = x
```

Looping (2)

- Loops can be created using `for` and `while`

Example

```
x = 0;
i = 0;
while i < 10
    x = x+i;
    i = i+1;
end
ans = x
```

Performance of looping and an example

- Although loop performance in Matlab has improved, there are often better ways to approach things
- Lets look at 3 possible ways to calculate and Lp Norm of a vector x:

$$\left(\sum_{i=1}^{N} |X_i|^p\right)^{1/p}$$

  - Looping
  - Combining built in functions
  - Using one built in function

Example take aways

- Don’t reinvent the wheel
- Google is your friend: “matlab my goal”

Functions

- Matlab allows you to write your own function
  - and you should!
- Put logic into individual functions that you know do what you want and then call them
- Functions are declared in their on *.m* file

Example

```
[out1 out2] = function(in1, in2)
% this is my function documentation

% this is where the function logic goes
end
```

The search path

- Matlab has a path that it looks for the *.m* files that define your functions
- You can change the current working directory of matlab from the interface
- You can also add specific directories to your path `path(path,'newpath')`
- See `help path` for more info