Machine Learning

Prof. Jose L. Mendoza-Cortes

Scientific Computing Department, Dirac Science Building
Materials Science and Engineering, High Performance Materials Institute
Florida State University
jmendozacortes@fsu.edu

Condensed Matter Theory, National High Magnetic Field Laboratory
Florida State University
mendoza@magnet.fsu.edu

Chemical and Biomedical Engineering
Florida State University — Florida A&M University — College of Engineering
mendoza@eng.famu.fsu.edu

Web: http://mendoza.eng.fsu.edu/

Overview

1. Review last class
2. Anonymous Functions
3. Recursion

Remember

- MATLAB is case sensitive!.
- If you declare a **variable** `Temp` and then use `temp`, MATLAB will give an error
- If you define **function** `Pressure` but then try to run `pressure` with F5 or run, MATLAB will give you an error (assuming your function needs inputs).

Camel Case convention

- This useful to keep track of your code and be organized.
- **Variables** named with a single word are all lower case. Example: `position`
- **Variables** named using more than one word, should include capitalization on every word but he first. Examples: `centerPosition`, `centerOfGravity`
- **Function** names should always start with a capital letter starting each word Examples: `CalculateCenterPosition`

Review on functions: Early example

```
function [P] = Pressure(T,V)
% Defition: Calculates the pressure in ideal gas
% Use: Pressure (Temperature, Volume)
% Input: T = temperature (K), V = volume (L)
% Output: P = Pressure (Atm)
R=8.314;
P = R.*T./V;
end
```

Code 1: Script

```
thePressureis = Pressure(298,10)
```

Remember the difference with local variables

Variables defined inside scripts are valid after executing the script (i.e. you will see them in the workspace window, e.g. 298, 10).
However variables within a function are said to be **local** and erased after the function is executed (e.g. `R` on this case).

Another way to call assign the output

Code 2: Another way to assign the output

```
clear; clc;
darklord = Pressure(298,10)
%R=8.314
```

Code 3: The output for the script above is

```
darklord =
247.7572
```

Remember

Remember that you can assign the output to whatever variable you choose. On this example we assign the pressure into a variable called `darklord`, whereas before we call it `thePressureis`

Review on functions: subfunction scheme

```
function [ P ] = Pressurews( T,V )
P = Pressuresubfunc(T,V);
end
function [P] = Pressuresubfunc(T,V)
R = 8.314; P = R.*T./V;
end
```

Main function versus sub-function

The first function is the main (or primary) function

- It is the only function that is accepted in the command window and other functions and script.
- If we run the function `Pressuresubfunc` in the command window we get an error. This is because the subfunction `Pressuresubfunc` is **local** function, which is analog to the **local** variable above.

Review on functions: Input fashion

```
function Pressureinputfashion
%Helpcomments go here
R=8.314;
T=input('Give me the temp (K): ');
V=input('Give me the Vol (K): ');
disp(' ')
disp('The pressure is: ');
disp(R.*T./V)
end
```

Running a function directly, notice the difference

Notice how we can run the function directly! this because the function does not need inputs, but the input are requested while the function is running. This is just useful to know.

Code 4: Filename = stats.m

```
function [ mymean mystdev ] = stats( numbers )
%This will calculte the mean and the standard deviation
%Use: [mean standardeviation] = stat (arrayofnumber)
%input: number = array of numbers
%output: mymean = geometric mean
%        mystdev = standard deviation
n = length (numbers);
mymean = sum(numbers)./n;
mystdev = sqrt(...
    sum((numbers-mymean).^2./(n-1)));
end
```

Code 5: Script

```
numbers       = [15 20 30 17 18]
[mean, sigma] = stats(numbers)
```

Notice the assignment of the variables as outputs are different than the output names in the function (mymean vs mean).

More notes on the input command

```
clc; clear;
n = input ('Give me a number: ')
%Accepts a number
name = input ('what is your name: ','s')
% Accepts a string
disp = (n)
%value can be a number, string, expression
disp = (name)
%value can be a number, string, expression
```

Something useful to know

The input command tells the computer to wait for you to give it a value. Test the script above in your own computer.

Notes - Anonymous Functions

Anonymous Functions:

- is a function that is not stored in a program file, but is associated with a variable.
- can accept inputs and return outputs, just as standard functions do.
- However, they can contain only a single executable statement.
- the main of an anonymous function is that you can define in the same script so you do not have to create another file to store your function

Remember

Any function (implicit or explicit) is a way to outsource part of your code that would be repetitive. Outsourcing part of your code that will repeat many times into a function makes your code clear and allow you to use that same part later when you need it.

Implicit Functions: Example 1

```
clc; clear; close;
% Anonymous functions explanation
TestFun = @(x) x.^3
%Now let's test the function
a = TestFun(3);
y = 5 ;
b = TestFun(y);

fplot(TestFun, [-10 10])
fzero(TestFun, [-10 10])
%fzero finds when TestFun changes sign
```

Notice how we add the @() when we define the implicit function but we do not put the @ when we call the function.

Implicit Functions: Example 2

```
1
2  p1 = @(x) Pressure(298,x);
3  p2 = @(x) Pressure(398,x);
4  p3 = @(x) Pressure(498,x);
5
6  figure(1)
7  fplot(p1, [0.01 .1],'rs:'); hold on;
8  fplot(p2, [0.01 .1],'bd-')
9
10 figure(2)
11 fplot(p1, [0.01 .1],'r.-'); hold on;
12 fplot(p2, [0.01 .1],'b--')
13
14 figure(3)
15 subplot(2,1,1);fplot(p1, [0.01 .1],'rs:')
16 subplot(2,1,2);fplot(p2, [0.01 .1],'bd-')
```


Implicit Functions: Example 2 (Continuation)

```
1
2  figure(4)
3  subplot(2,2,1);fplot(p1, [0.01 .1],'rs:')
4  subplot(2,2,2);fplot(p2, [0.01 .1],'bd-')
5  subplot(2,2,[3 4]);fplot(p3, [0.01 .1],'cx-')
6
7  figure(5)
8  subplot(2,2,1);fplot(p1, [0.01 .1],'rs:')
9  subplot(2,2,2);fplot(p2, [0.01 .1],'bd-')
10 subplot(2,2,3);fplot(p3, [0.01 .1],'cx-')
11 subplot(2,2,4);
12 fplot(p1, [0.01 .1],'rs:');
13 hold on;
14 fplot(p2, [0.01 .1],'bd-');
15 fplot(p3, [0.01 .1],'cx-');
```

Notes - Recursion

Recursion:

- is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself.

Parts of a Recursive Algorithm

1. Base Case (i.e., when to stop)
2. Work toward Base Case
3. Recursive Call (i.e., call ourselves)

The "work toward base case" is where we make the problem simpler. The Base Case is the solution to the "simplest" possible problem.

Sum up an array

File: `SumArrayR.m`

```
function[sum] = SumArrayR(list)
  if (length(list) == 0 )
    sum = 0;
  else
    sum = list(1) + ...
      SumArrayR (list(2:end));
  end
end
```

Now try in the command line

```
>> list=[1:5]
>> SumArrayR(list)
```


Notes - Recursion

What is the definition of Factorial !?

$$N! = N * (N-1) * (N-2) * ... * 2 * 1$$

Hmmm, but what does

$$(N-1) * (N-2) * ... * 2 * 1$$

equal?
Answer: $factorial(N-1)$
What if we combine these definitions.
Definition of Factorial:

$$factorial(N) = N! = N * factorial(N-1)$$

Lets write a recursive factorial function.

File: `FindFact.m`

```
function[res]= FindFact(N)
   if (N == 0)
     res = 1;
   else
   res = N*FindFact(N-1);
   end
end
```

Now try in the command line

```
>> FindFact(5)
```

Compare with MATLAB's function: factorial(x)

How to continue to the next line of code

This is important

The **ellipsis** (three consecutive periods, ...) at the end of the line means 'to be continued'. This is useful to keep your code in one window.

Code 6: For codes and numerical variables

```
a = [20; 21; 22]; b = {'Eng'; 'Sci';...
'Math'} %Notice the {} instead of []
X=table(a, b, ...
'VariableNames',{'age','Major'})
```

Code 7: For strings

```
quote = ['the computer will do ',...
'what you tell them to do',...
'not what you want them to do']
```

Test these commands/script in your computer

How to continue your code in the same window

```
%This is going to be a cont... of strings
clc; clear;
n = 'Dark'; l = ' lord';
donotsayit = [n l]
quote = ['The computer will do ',...
    'that you tell them to do not want you '...
    'want them to do']
```

This is useful if you want to keep your code in the same window. If you use the ellipsis MATLAB will assume the code continues in the same line, however for the programmer is easier to see all in the same window.

Vectorization: The most important thing to remember

SUPER IMPORTANT

To make a variable into a vector can be done in two main ways. For more complex codes this might not be trivial.

Code 8: Vectorization form 1

```
counter = 0;
for vector = 0:0.5:10
    counter = counter + 1
    y(counter) = sin(vector);
end
```

Notice how the variable `counter` acts as a way to store the numbers into an array, while `sin(vector)` is simply evaluating the sine of each value of the variable `vector`

Vectorization: The most important thing to remember

SUPER IMPORTANT

To make a variable into a vector can be done in two main ways. For more complex codes this might not be trivial.

Code 9: Vectorization form 1

```
vector= 0:0.5:10
y=sin(vector)
```

Notice that on this case, we declare the variable `vector` as a vector from the begining and then operate on it. This is because of MATLAB can operate directly on arrays.

See you next class

"Just as there is not royal road to geometry, there is no royal road to programming".- Euclid and J. V. Guttag

The computer will do what you TELL them to do NOT what you WANT them to do.- Someone in the Internet (Perhaps)

Think twice, code once.- Anonymous

The sooner you start to code, the longer the program will take.- R. Carlson

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.- M. Fowler

Simplicity is the soul of efficiency.- A. Freeman

If you cannot grok the overall structure of a program while taking a shower, you are not ready to code it.- R. Pattis

Appendix: Scripts included

Code 10: anonymous_function_ex.m

```
clc; clear; close;
% Anonymous functions explanation
TestFun = @(x) x.^3
%Now let's test the function
a = TestFun(3);
y = 5 ;
b = TestFun(y);

fplot(TestFun, [-10 10])
fzero(TestFun, [-10 10])
%fzero finds when TestFun changes sign
```

Try these commands in your own workstation, i.e. have the lectures on one half side of your screen and Matlab/Octave-GUI on the other half.

Check the scripts/functions under the directory for this note number (X): /NX_Notes_directory