Lecture 3: Functions

Math 98, Spring 2019

Reminders

Instructor: Eric Hallman
Login: !cmfmath98
Password: c@1analog
Class Website: https://math.berkeley.edu/~ehallman/98-sp19/
Assignment Submission: https://bcourses.berkeley.edu
Homework 2:

1. Due Feb 7 by 11:59pm on bCourses
2. Collaboration welcome

HW2 Pointers

- `mod(m,n)` gives the remainder when `m` is divided by `n`.
- It's fine to define the function `isPerfect` locally rather than in a separate file.

Functions: Recap

Recap of the basic format for a Matlab function:

```
%%%Name.m%%%
function [output vars] = Name(input)

% code here

end
```

The function name should be the same as the name as the M-file.

Functions v. Scripts

Scripts:

- No inputs or outputs–Matlab just executes all commands
  - (Unless you use `input`)
- Operates on existing data in the workspace
- Variables created remain in the workspace

Functions:

- Accept inputs and return outputs
- Create their own separate workspace
- Only requested output variables get saved

Functions v. Scripts

Functions do not access variables stored in the main Workspace.

```
%%%exampleFunction.m%%%

function w = exampleFunction(x,y)
w = x + y + z;
end
```

```
>> z = 5; a = exampleFunction(2,3);
Undefined function or variable 'z'.
```

Functions v. Scripts

Functions do not save variables back to the main Workspace unless they are requested as outputs.

```
%%%exampleFunction.m%%%

function a = exampleFunction(x,y)
a = x + y; b = 101;
end
```

```
>> a = exampleFunction(2,3); disp(a);
    5
>> disp(b)
Undefined function or variable 'b'.
```

Functions v. Scripts

Because functions use their own workspace, variables named inside a function cannot conflict with variables of the same name outside the function.

```
%%%exampleFunction.m%%%

function a = exampleFunction(x,y)
b = 100; a = x + y + b;
end
```

```
>> b = -300; a = exampleFunction(40,5); disp(a);
    145
>> disp(b);
    -300
```

Local Functions

We can define more than one function in a single file.

```
%%%myStats.m%%%
function avg = myStats(x);
% takes a vector and returns the average.
    n = length(x);
    avg = myMean(x,n);
end

function m = myMean(v,n)
% it takes a vector and its length, returns the mean
    m = sum(v)/n;
end
```

Only the first function (the **main** function) can be called from other programs or the command line.

Local Functions

We can also define local functions in scripts:

```
v = 1:5;
L = myLength(v);
fprintf('the length of v is %f', L);

function len = myLength(x)
    len = sqrt(sum(x.^2));
end
```

Any function definitions must come at the end of the script.

Commenting

As with built-in Matlab functions, we can use comments and `help` to inform how each function is properly used.

```
>> help myStats
  takes a vector and returns the average.
>> help myStats>myMean
  it takes a vector and its length, returns the mean
```

nargin/return

When used in the code for a function, nargin is the number of inputs specified by the user. Handy when setting default values for inputs.

```
%%%addMe.m%%%
%Input: one or two floating point numbers
%Output: addMe(x,y) returns x + y; addMe(x) returns 2*x
function s = addMe(x,y)
    if (nargin == 1)
        s = x + x;
    elseif (nargin == 2)
        s = x + y;
    else
        fprintf('Read the comments!');
        return
    end
end
```

return automatically halts the function.

Exercise

Write a function `myCosine(theta,units)` that returns the cosine of an angle. If the second parameter is 'deg', convert the angle to radians with a local function `DegToRadians(x)` before using Matlab's `cos`. In all other cases (including no second parameter), assume the angle is in radians.

```
>> myCosine(180, 'deg')
    -1
>> myCosine(pi, 'rad')
    -1
>> myCosine(pi)
    -1
```

Problem

We would like to find the roots of the polynomial

$$p(x) = x^3 + bx + c$$

for various numbers $b, c \in \mathbb{R}$.

- How can we produce this family of functions?
- What tools does Matlab have to solve this problem?

Nested Functions

Nested functions are functions defined within other functions.

```
function f = makeCubic(b,c)
    function y = myCubic(x)
        y = x.^3 + b*x + c;
    end

    f = @myCubic;
end
```

They can access variables in the workspace of the parent function, and don't need to be defined at the end of the code in the parent function.

Function Handles

A function handle is a Matlab variable that allows us to reference functions indirectly. Use them to include functions as inputs to or outputs from other functions.

```
>> integral(cos,0,1)
Error using cos
Not enough input arguments.
>> integral(@cos,0,1)
    0.8415
```

Anonymous Functions

A way to define functions in the middle of a Matlab script or in the command line. Takes the form `functionName = @(inputs)(output)`, and returns the function handle `functionName`.

```
>> f = @(x,y)(x^2-y);
>> f(10, 3)
    97
>> fzero( @(x)(x^2-2), 1.5)
    1.4142
```

Useful when defining functions with simple expressions.

Anonymous Functions: Basic Plotting

`fplot(@f, [xMin,xMax])` plots a given function on the desired interval.

```
>> fplot(@cos, [-1, 3]);
>> xlabel('hello'); ylabel('world');
>> title('My Title', 'FontSize', 15);
```

As with `integral` and `fzero`, `fplot` must take a function handle as its first argument.

Anonymous Functions

We can use anonymous functions to get behavior similar to our function `makeCubic(b,c)` from earlier:

```
>> b = 3; c = 5;
>> f1 = @(x)(x^3 + b*x + c);
>> fzero(f1,0)
    -1.1542
>> b = 2; c = -1;
>> f2 = @(x)(x^3 + b*x + c);
>> fzero(f2, 0)
    0.4534
```

Question: does changing the values of `b` and `c` change the function `f1`, or will `f1` and `f2` be different functions?

Exercise

Write a function `makeScrambler(a,p)` that returns a function `f` such that

$$f(x) = x^a \pmod{p}.$$

Then do the same thing using an anonymous function definition.

128A Assignment

Implement a function `newton.m` of the form

```
function r = newton(x0, f, p, n)
% x0: initial estimate of the root
% f : function and derivative handle [ y, yp ] = f(x,p)
% p : parameters to pass through to f
% n : number of steps
```

Solve $f(x) = \frac{1}{x} + \ln x - 2 = 0$ and tabulate the number of correct bits at each step of the iteration.

- "Number of correct bits" $\approx$ -log2(relative error)