18 Functions!

User-Defined Functions

Lecture 7 (Feb 13)
CS100M – Spring 2007

Announcements

- Prelim 1 Conflicts
  - Our exam: Thursday, Feb 22, at 7:30pm
  - You *must* contact Kelly Patwell (see website) if you have any scheduling difficulties due to other exams
- Register your clicker!
  - See the announcement on the 100M website
  - We need the registration to know which student goes with which clicker
- For this week, section is back in the lab

Functions

- There are lots of functions that are built-in to Matlab
  - General math: max, min, abs, sqrt
  - Trigonometry: sin, cos, tan, asin, acos, atan
  - Exponential: exp, log, log2, log10
  - Integer computation: round, floor, ceil, fix, mod
- Matlab is designed so that a user can add new *user-defined* functions
- Goals for how a user-defined function should behave
  - Should have input
  - Should have output
  - Should be able to use a function without clobbering user's variables
  - Should be able to use it just like we use a predefined function

Simple Example Function

- Goal: a function that computes $f(x) = x^2+4x+4$
- Code to do this (stored in an m-file):

```
function y = f(x)
% Compute f(x) = x^2 + 4*x + 4
y = x^2 + 4*x + 4;
```

- Using this function (at the Command Window)

```
>> f(3)
ans =    25
>> f(0)
ans =     4
>> f(4)
ans =    36
```

Script vs. Function Example

- Suppose we have the following two m-files (i.e., files with .m suffix)

```
% g(x) = x^2 + 4*x + 4
y = x^2 + 4*x + 4;
```

```
function y = f(x)
% f(x) = x^2 + 4*x + 4
y = x^2 + 4*x + 4;
```

- We can do "the same stuff" with both, but the script is more cumbersome

```
>> x = 10;
>> g;
>> z = y;
```

```
>> z = f(10);
```

- For the script, anything that used to be stored in x or y is now gone

General Form for a User-Defined Function

```
function outputArg = functionName(arg1, arg2, ...)
% One line comment describing the function
% Additional description of function
<executable code which at some point assigns to outputArg>
...
```

- arg1, arg2, … are defined when the function's code begins execution
  - These input variables (called function *parameters*) hold the function *arguments* used when the function was called
- outputArg does not have a value until something is assigned to it

Returning Multiple Values

```
function [outArg1, outArg2,...] = functionName(arg1, arg2, ...)
% One line comment describing the function
% Additional description of function
<code which at some point assigns to outArg1 and outArg2>
...
```

- This kind of function is called using something like this
  `[x, y] = coords(angle)`
- The first returned value is stored into x, the next into y, etc.

Scripts vs. Functions

- The programs you have been using until now have all been *scripts*
- A script is executed line-by-line just as if you are typing it into the Command Window
  - A change to a variable within the script is a change to the variable in the Command Window workspace
- A function has its own *private* workspace (for its variables) that does not interact with the Command Window workspace
  - Variables are *not* shared between workspaces even if variables have the *same name*

A Function Example

- Goal: Choose a uniform-random number between L and U
- Recall: We needed a random number between 1 and 9 for Project 1
  - We used: n = 1 + 8*rand(1);
- We can make this into a function:

```
function number = myRand(L, U)
% myRand(L,U) is a random number between L and U
number = L + (U-L)*rand(1);
```

- This is used as: n = myRand(1, 9);

Why Use Functions?

- Functions keep *driver programs* clean by keeping coding details in separate, non-interacting files
- Functions can be independently tested
- Functions provide a useful *level of abstraction*, allowing one to easily re-use code
  - E.g., you don't need to know the details of how sqrt or sin are implemented

To Execute y = myFunction(x)

- Matlab looks for an m-file that matches the function name
- Arguments are *copied* into the function's local parameters
  - This scheme (*copying* values into parameters) is called *pass-by-value*
  - Some programming languages use other argument-passing schemes (but Java also uses *pass-by-value*)
- The function's code is executed using the function's own private workspace
- Once a function has been executed, its workspace is deleted
  - Except for the output-value which, in this example, is assigned to y
  - If a function is called again, it starts with a new, *empty* workspace

Comments in Functions

- Some comments in a function are treated specially
  - The entire block of comments after the function statement is printed whenever a user types **`help functionName`** at the Command Window
  - The *first line* of this comment block is searched whenever a user types **`lookfor someWord`** at the Command Window
- Every function should have a comment block (after the function statement)
  - With a first line that succinctly describes what the function does
  - And, if necessary, additional lines that describe how one uses the function