CS 105

Introduction to Scientific Computing

Topic #15 – Custom Functions

Matt Burlick
Stevens Institute of Technology

ASSIGNMENT 10

- Write a function to compute the hyperbolic sine
  - $sinh(x) = \frac{e^x - e^{-x}}{2}$
- Use the function to plot x vs sinh(x) for x=-π,.., π

NECESARY SKILLS

- How to write custom functions
  - File format
  - Input and outputs
- How to use custom functions in a script or another function

TOPICS

1. Making Custom Functions
2. Using Custom Functions

READING

- Section 5.1

WHAT'S A FUNCTION

- We already know what a function is
  - Something that takes inputs (maybe), does some job, and returns outputs (maybe)
- We've used tons of built-in MATLAB functions
  - disp
  - Input
  - plot
  - rand
  - isempty

CUSTOM FUNCTIONS

- Why might we want to make our own functions?
  - Recycling code!
  - Write once, use often!
- How do we make a custom function?
  - Make new files, one per function
  - Start file with special *header* which includes the keyword *function* and ends with keyword *end*
    - It's just a block of code (which may contain other blocks)!

FUNCTION HEADER

- How do we use functions?
  - Just like with built-in Matlab functions!
    - output = function_name(param1,param2,…)
      - Where param1,param2, etc.. have values
      - They're *inputs*
    - output is a variable we store the returned value in
- How do we create functions?
  - We start our functions file similarly:
    - **function** output=function_name(param1,param2,…)
  - Note the keyword *function*
  - The variable *output* must be set in the body of the function
    - If in fact the function is supposed to return something
  - param1, param2, … are just variables names to be used (maybe) in the body
    - They are *assigned* values when the function is *used*

EXAMPLE: PRINT ARRAY

- %file PrintArray.m
- function PrintArray(A)
```
        for i=1:length(A)
                disp(A(i));
        end
  end
```
- Note: This function doesn’t return anything!

EXAMPLE: SUMARRAY

- % file: SumArray(A)
- function s = SumArray(A)

```
function s = SumArray(A)
        s=0;
        for i=1:length(A)
                s = s + A(i);
        end
end
```

- Note: s is the return value, must be assigned in the function

EXAMPLE: GETMINMAX

- %file: GetMinMax
- function [mi,ma] = GetMinMax(A)

```
          mi=NaN;
          ma = NaN;
          if(length(A)>0)
                    mi=A(1);
                    ma=A(1);
          else
                    return;
          end
          for i=2:length(A)
                    if(A(i) > ma)
                              ma = A(i);
                    end
                    if(A(i) < mi)
                              mi = A(i);
                    end
          end
end
```

- Note: We return several values as a vector!

USING FUNCTIONS

- Using functions is often called *calling functions*
- Here we specify the function name, supply values for the inputs (maybe), assign variables for the outputs (maybe)
- X = input('Enter something: ');
- plot(x,y);
- PrintArray([4 3 4 0]);
- X = SumArray([4 3 4 0]);
- [x,y] = GetMinMax([4 3 4 0]);

LOCAL VARIABLES

- For our purposes variables are “local” to the script they’re in
  - So if a script or function calls a new function, the variables within the function are different than the variables outside the function

COMMANDS, SCRIPTS, AND FUNCTIONS

- So now we've talked about 3 ways to do code:
    1. Single commands
    2. Sequences of commands in a file as a *script*
    3. A *function* in a file to be used by other functions/scripts

INCLUDING FUNCTIONS

- If a function is in the same folder as the functions/scripts that use it we have no problem!
  - The computer can find them
  - Also true for Built-In functions
- What if there function is elsewhere?
  - We may want to have different folders to group different types of functions
  - To allow a function/script to *find* it we use ***addpath(path)*** where path is either the absolute or relative path to the function(s)

PATHS

- Absolute path let's us specify where it is on the computer from the *root* of the machine:
  - Windows: "C:/MyDocs/MyFunctions"
  - Mac/Linux/Unix: "\home\MyDocs\MyFunctions"
- Relative paths specify where a file is *relative* to the current location
  - Nice for if we move stuff around!
  - Uses a few special characters
    - '.' means "current folder/directory"
    - '..' means "parent folder/directory"
  - EX: addpath('./../MyFunctions/SortingFunctions');