Introduction to Scientific Computing and Problem Solving

Lecture 8
Functions II

Assignments

- Review lecture slides, diaries, and m-files.
- Homework 2 is due Thursday, 2/28.
- Lab 3 begins Sunday, 2/24.
- You can always check assignments on the course web site.
- Grades now posted on MyCourses.

www.cs.brown.edu/courses/csci0040
mycourses.brown.edu

Introduction to Functions

- Functions let us
  - Break up a big program into small chunks
    - Easier to write because we are focused
    - Help make a large program clearer
  - Write some code once and reuse it in the program many times

Functions in MATLAB

- You can create function files that can be used just like the built-in functions.
- First line: function definition

```
function [output arguments] =
     function_name(input arguments)
```

- Input arguments
  - List of variables used in the function that provide input when the function is "called"
- Output arguments
  - List of variables used in the function and that transfer output from the function

Using Your Function

- Use from Command Window, script file, or from another function
- Assigning output to a variable

```
average_grade = CalcCS4HWGrade(1)
```

- Using in an expression

```
total_weight = weight_ring(d) +
    weight_base(l,w,h)
```

- Type in Command Window

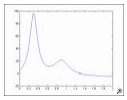
```
>> PlotMyData(x)
```


Anonymous Functions

- Good for short (one-line) calculations used frequently in a longer program.
- Example: converting Fahrenheit to Celsius.
- General form

```
function_name = @ (arguments) expression
FtoC = @ (F) 5*(F-32)./9
cube = @ (x) x^3
circle = @ (x,y) 16*x^2+9*y^2
```

Function Functions



- MATLAB built-in function **`fzero`** can find the zeros (values of *x* where *f(x)* = 0) of any function *f(x)*. How do we describe *f(x)* to **`fzero`**?
- *Function functions* like **`fzero`** accept functions as arguments in two ways
  - Function handle
  - Using the name of the function in a string

Function Handles

- A unique value associated with any function (user-defined, built-in, anonymous)
- Obtained by

  `@function_name`

  `@cos`

  `@FtoC` (as function file)

  `FtoC` (as anonymous function)
- Function functions must use input arguments consistently with the input function.

Function Function Example

- Funplot: evaluates and graphs a function over a specified range returning a matrix of values at each extreme and the midpoint.

$$f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$$

- Use user-defined function
- Use anonymous function

`m-files: funplot, Fdemo, lect7_2`

Function Names in Strings

- Older method, less efficient.
- Pass name of function as a string

```
'cos'
'FtoC'
```

- Evaluate

```
var = feval('function name', arguments)
```

`m-files: funplotS`

Subfunctions

- A function file can contain more than one user-defined function.
- First function defined, the "primary," is how the function is known to the rest of the program.
- Other functions, "subfunctions," are only known inside the function file and each has its own workspace (local variables).

Example 7-4
Mean and Standard Deviation

- Write a function that calculates mean (average) and standard deviation of a list of numbers.

$$\bar{x} = \frac{(x_1 + x_2 + \ldots + x_n)}{n}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{i=n} (x_i - \bar{x})^2}{n-1}}$$

`m-files: stat, Ex7_4`

Nested Functions

- Subfunctions have separate workspaces (variables)
- By nesting function definitions, variables can be shared

```
function y = A(a1, a2)
…
      function z = B(b1, b2)
      …
      end
      function w = C(c1, c2)
      …
      end
end
```

m-files: statNest

Example 7-5
Exponential Growth and Decay

- Growth rate (or decay rate) is proportional to current value of the function.
  - Microorganisms in culture
  - A virus like SARS or West Nile
  - Processing power of computers (Moore's Law)
  - Nuclear chain reaction
  - Multilevel marketing
  - Temperature changes of an object (Newton's Law of Cooling)
  - Similarities between two languages with a common root

Example 7-5 (cont.) Exponential Growth and Decay

- General model

$$A(t) = A_0 e^{kt}$$

where $A(t)$ and A_0 are the quantity at time t and time 0, respectively, and k is a constant.

- Write a function that finds $A(t)$ knowing A_0 and $A(t_1)$ at some other time t_1 and then solve:
  - The population of Mexico was 67 million in 1980 and 79 million in 1986. Estimate the population in 2000.
  - The half-life of a radioactive material is 5.8 years. How much of a 7 gram sample will be left after 30 years?

Example 7-5 (cont.) Exponential Growth and Decay

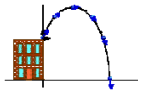
- First find the constant k

$$k = \frac{1}{t_1} \ln \frac{A(t_1)}{A_0}$$

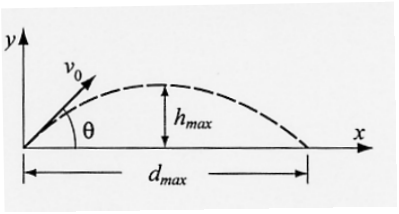
- Write a MATLAB function expGD with arguments *A0*, *At1*, *t1*, *t* to find the value of *A* at time *t*.

`m-files: expGD`

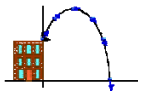
Example 7-6
Motion of a Projectile



- Applying the laws of motion, write a function to plot the trajectory of an object with initial velocity of 230 m/s at an angle of 39°. Find the maximum height and distance.



Example 7-6 (cont.)
Motion of a Projectile



- Initial velocity v_0 has horizontal and vertical components

$$v_{0x} = v_0 \cos\Theta \text{ and } v_{0y} = v_0 \sin\Theta$$

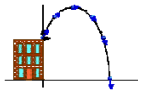
- Vertical motion is governed by

$$v_y = v_{0y} - gt \text{ and } y = v_{0y}t - \frac{1}{2}gt^2$$

- Highest point is reached when

$$t_{hmax} = \frac{v_{0y}}{g} \text{ and } h_{max} = \frac{v_{0y}^2}{2g}$$

Example 7-6 (cont.) Motion of a Projectile



- Total flying time: solve for $y = 0$

$$y = v_{0y}t - \frac{1}{2}gt^2 = 0 \text{ or } t_{tot} = \frac{2v_{0y}}{g} = 2t_{hmax}$$

- Horizontal velocity is constant and position is

$$x = v_{0x}t$$

`m-files: trajectory`

Program Design

- Data
  - How do we represent the values of interest to us?
  - What variables shall we name?
- Algorithm
  - How do we organize the sequence of calculations?
- Implementation
  - How do we break this up into manageable pieces?
  - What are the MATLAB statements we need to perform?

Example 7-7
Grade Book

- Write a MATLAB function to calculate final grades in a course with homework assignments, midterm exams, and a final.
  - Homeworks: 6 @ 10 points each, total 10% of grade
  - Midterms: 3 @ 100 points each, total 45% of grade
  - Final: 1 @ 100 points, 45% of grade
- The input is a matrix with a row for each student's grades.
- The output is a column vector with each student's final grade.

Example 7-7 (cont.)
Grade Book

- Data
  - StudentScores

	← *Homework* →						*Midterms*			*Final*
Student 1	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	h_{16}	m_{11}	m_{12}	m_{13}	f_1
Student 2	h_{21}	h_{22}	h_{23}	h_{24}	h_{25}	h_{26}	m_{21}	m_{22}	m_{23}	f_2
Student 3	h_{31}	h_{32}	h_{33}	h_{34}	h_{35}	h_{36}	m_{31}	m_{32}	m_{33}	f_3
Student 4	h_{41}	h_{42}	h_{43}	h_{44}	h_{45}	h_{46}	m_{41}	m_{42}	m_{43}	f_4

  - HomeworkWeight, MidtermWeight, FinalWeight
  - HomeworkMaxPoints, MidtermMaxPoints, FinalMaxPoints

Example 7-7 (cont.)
Grade Book

- Algorithm

```
% Assign the weights
% Assign the maximum points
% Sum the individual scores
% Compute the weighted score
```

- Implementation

```
function Grades = GradeBook(StudentScores)
function Total = SumScores(first, last)
```

m-files: GradeBook, Ex7_7