EXERCISES

Introduction to MATLAB: Programming

I. Class Materials

1. Download Programming.tar OR Programming.zip

From a web browser:

Download the file **Programming.tar** or **Programming.zip** from
http://web.mit.edu/acmath/matlab/IntroMATLAB to a local directory.
On Windows, if you do not have WinZip, download **Programming.zip**.

Alternatively, on Athena:

athena% add acmath
athena% cp /mit/acmath/matlab/IntroMATLAB/Programming.tar **.**

2. Extract this session's sub-directories and files

On Athena *(or the UNIX shell on Mac OS X)**:***

tar –xvf Programming.tar

On laptops:

Use your computer's utilities, such as double click or WinZip on Windows or StuffIt on Mac. Without WinZip on Windows, double click on **Programming.zip** and select File->Extract All. Your local work directory should now contain the following directories and files:

Programming

Exercise_One

cubicfitplot.m	modelfitplot.m

Exercise_Two

loadmesh.m	grid_x.dat
loadpoints.m	grid_y.dat
plotdata.m	interp_spline_x.dat
plotscript.m	XYZ_point_coordinates.txt
plotprogram.m	

Exercise_Three

veloctiyprogram.m	orbitalvelocity.m
velocityscript.m	

You may place and rename directories and files any way you wish. For consistency, we shall refer to the directory **Programming** as the work directory for these exercises.

II. Start MATLAB

On Athena:

athena% cd Programming
athena% add matlab
athena% matlab &
>> desktop

On laptops:

Launch MATLAB and navigate to the work directory **Programming**.

III. Exercise 1: Model Fit to Data Points

Purpose

To practice the following in MATLAB:

- Writing and understanding for loops.
- Using functions such as length, size and zeros to pre-define vectors and matrices.

Background

We shall use the same data that we worked with in Example Two of Session 2: Linear Algebra and Calculus and in Examples Two and Three in Session 3: Graphics. The difference is that here we shall also fit our own model to the data points, instead of using only built-in functions.

1. Open modelfitplot.m and cubicfitplot.m in the MATLAB Editor

```
>> cd Exercise_One
>> edit cubicfitplot.m
>> edit modelfitplot.m
```

2. Read and understand modelfitplot.m and cubicfitplot.m

- Both files are **script M-Files** that are meant to run as programs. Read all lines and try to understand them, before running the programs form the Command Window.
- Note the use of built-in function length to find the number of elements in a vector:
  N = length(x)
- Note the use of function zeros to create a vector of a certain size in modelfitplot.m:
  y = zeros(1, N)
- Note the use of a **for statement** to perform commands repetitively over an interval; for example, in file modelfitplot.m:

```
for i = 1 : N
        y(i) = 60 – 30 * cos( pi/3 + x(i)*pi/6);
end
```

  The program goes through a **for loop** to calculate y(i) for every x(i), using a non-linear equation. Try this with your own model (i.e. with your own equation).

3. Execute the M-files in the Command Window

Run the two programs from the Command Window and see the output.

```
>> cubicfitplot
>> modelfitplot
```

IV. Exercise 2: Function M-Files and Script M-Files

Purpose

To practice the following in MATLAB:

- Creating new functions in function M-Files.
- Writing **script M-files** and **function M-Files** to create MATLAB programs.
- Using built-in functions and variables such as uigetfile and nargin for program input.
- Writing **if statements** for program flow control and using **relational operators**.
- Working with **local variables** in functions and scripts.

Background

We used the same data in Example Three of Session 1: Interface and Basics and Example One in Session 3: Graphics. Here we define and use in a program three new functions: loadpoints, to import coordinates of a point set; loadmesh, to import coordinates of a surface; and plotdata, to plot a set of points and a surface on an annotated figure with customized appearance.

1. Open all M-files in the Exercise_Two directory in the MATLAB Editor

```
>> cd Exercise_Two
>> edit loadmesh.m
>> edit loadpoints.m
>> edit plotdata.m
>> edit plotscript.m
>> edit plotprogram.m
```

2. Read and understand all M-Files

- These files are part of two programs: plotscript and plotprogram. Read and understand all lines in the files, before running the programs from the Command Window.
- plotscript.m and plotprogram.m are **script M-files**.
- loadmesh.m, loadpoints.m, and plotdata.m are **function M-files**. They define three new functions: loadmesh, loadpoints, and plotdata, respectively.
- Note the format of a **function M-file**; for example, plotdata.m:
    - The first line is the **function's definition**, including **arguments** the function takes (xdata, ydata, zdata, X, Y, Z for function plotdata) and returns (none here):
      `function plotdata(xdata, ydata, zdata, X, Y, Z)`
    - The first commented line is the **H1 Line** (one line description of the function):
      `% PLOTDATA creates a customized plot from surface and point data.`
    - The first paragraph of comments is the **Help paragraph** for the function. Type the following in the Command Window and see what happens:
      `>> help plotdata`
    - The rest is the function's body, including command lines and comments.

- Note the use of built-in functions disp and error to display output in the Command Window when a function is executed; for example, in loadmesh.m:
  disp('Loading mesh data …')
  error('Number of arguments should be 3.')
- Note the use of built-in function uigetfile, for example in loadpoints.m:
  [filename, pathname] = uigetfile({ '*.txt', 'Get Text Files' } , 'Pick a file');
  At runtime, this will open a browser for file selection. What are the **input arguments** of uigetfile? (A **cell array** and a **string**.) What **output arguments** does uigetfile return?
- Note the use of built-in variable nargin to check the number of input arguments passed to a function; for example, in loadpoints.m:
  if nargin == 1
- In the statement above, note the use of **relational operator** == (meaning "equal to").
- Note the construction of a simple **if, else statement**, for example, in loadmesh.m:
  if nargin == 3
  commands for loading three files if nargin is equal to 3
  else
  command for error if nargin is not 3
  end
- What is the difference between the **script M-files** plotscript.m and plotprogram.m?
  plotscript.m includes an entire program and uses only built-in MATLAB functions.
  plotprogram.m also uses the three new functions defined in three **function M-Files**:
  [X, Y, Z] = loadpoints ('XYZ_point_coordinates.txt');
  plotdata(x, y, z, X, Y, Z)
- Note that in a **script M-file** all arguments passed to functions are specific and already exist; for example, in plotscript.m:
  data = load('XYZ_point_coordinates.txt');
  and in plotprogram.m:
  [X, Y, Z] = loadpoints ('XYZ_point_coordinates.txt');
- Now note that a function can take any arguments before specific variables for them are created; for example, in loadpoints.m, filename can be <u>any file name</u>:
  function [X, Y, Z] = loadpoints(filename)
- Note that **local variables** in **function M-File**s "live" only while the functions are being executed, and do not interfere with the same variable names being used in **script M-files**. For example, see X, Y, and Z in loadpoints.m and in plotscript.m and plotprogram.m.
- A **local variable** used in a **script M-File**, on the other hand, is shared with other script M-Files and the base workspace. This means that if you run plotscript.m, and change the values of X, Y, and Z, plotprogram.m will also know the new values of X, Y, and Z.
- Note that once a function is defined in a **function M-File**, this function can be called with different arguments form a **script M-File**. For example, in plotprogram.m:
  plotdata(x, y, z, X, Y, Z);
  plotdata(x1, y1, z1, X, Y, Z);

3. Execute M-files from the Command Window

- Use functions defined in **function M-Files** in the Command Window. For example:
  >> loadmesh('grid_x.dat', grid_y.dat', 'interp_spline_z.dat')

- Run the two programs plotscript and plotprogram, and explain what happens in terms of specific command lines in the **script M-Files** and in the **function M-Files**:
  >> plotscript
  >> plotprogram

V. Exercise 3: Orbital Velocity – An Interactive Program

Purpose

To practice the following in MATLAB:

- Creating **new functions** in function M-Files.
- Writing an interactive **MATLAB program** using **script and function M-Files**.
- Using functions such as input for program input from the Command Window.
- Writing **if** and **switch statements** for program flow control.
- Using **relational** (e.g. <=) and **logical** (e.g. ||) **operators** in a program.
- Working with strings and string-specific functions such as strcmp.

Background

This example, which we also used in Exercise Two in Session 1: Interface and Basics, is based on NASA's education site: http://exploration.grc.nasa.gov/education/rocket/rktrflght.html. You can read more theoretical background in the handout for Session 1 or on the above web site.

1. Open all M-files in the Exercise_Three directory in the MATLAB Editor

```
>> cd Exeircise_Three
>> edit orbitalvelocity.m
>> edit velocityscript.m
>> edit velocityprogram.m
```

2. Read and understand all M-Files

- These files are part of two programs: velocityscript and velocityprogram. Read and understand all files, before running the programs from the Command Window.
- velocityscript.m is the same **script M-File** that we used in Exercise Two in Session 1: Interface and Basics to compute orbital velocities in **matrix form**.
- velocityprogram.m is a **script M-File**, which is **an interactive program**, i.e. a program that allows users to enter input in the Command Window at runtime.
- orbitalvelocity.m is a **function M-file**, which defines a new function: orbitalvelocity.
- In **function M-file** orbitalvelocity.m identify the **function's definition**, **H1 Line**, and **Help paragraph**. What happens if you type in the Command Window the command:
  >> help orbitalvelocity
- Note the use of the built-in function input, which allows users to enter numerical or string input in the Command Window at runtime; for example, in velocityprogram.m:
  Re = ('Enter the mean radius of the planet: ');
  units = ('What units? E for English or M for metric: ', 's');
- Note the use of **relational operators** such as < (meaning "less than") and **logical operators** such as || (meaning OR); for example, in orbitalvelocity.m:
  if nargin < 3 || nargin > 4

- Note the construction of **if, elseif, else statements**, for example, in orbitalvelocity.m:
  ```
  if nargin < 3 || nargin > 4
      commands to execute if nargin is smaller than 3 or higher than 4
  elseif nargin == 3
      commands to execute if nargin is equal to 3
  else
      commands to execute if nargin is anything else
  end
  ```
- Note the construction of **switch, case statements**, for example, in orbitalvelocity.m:
  ```
  switch units
      case {'m', 'metric'}
          commands to execute if the variable units is either 'm' or 'metric'
      case {'e', 'english'}
          commands to execute if the variable units is either 'e' or 'english'
      otherwise
          commands to execute if units is anything else
  end
  ```
- Note the use of built-in functions for **strings**; for example, in velocityprogram.m:
  `if strcmp(units, 'm')`
  compares two strings – units and 'm' – and returns a logical true or false. Also:
  `units = lower(units)`
  turns all characters of units into lower-case characters. And in orbitalvelocity.m:
  `strV = num2str(V)`
  converts the numeric value of V into a string.
- What is the difference between the **script M-files** plotscript.m and plotprogram.m?
  plotscript.m includes an entire program, which only uses built-in MATLAB functions.
  plotprogram.m also uses a new function defined in orbitalvelocity.m.
  Note how much shorter a program is when put in matrix form!
- Note that in a **script M-file** all commands must be explicit, i.e. all arguments passed to functions are specific and already exist; for example, in velocityprogram.m:
  `V = orbitalvelocity(Re, g0, altitude, units)`
- Now note that the **arguments** R, G, H, and units of orbitalvelocity can take any values:
  `function V = orbitalvelocity(R, G, H, units)`
- Write MATLAB code to add a third planet as an option for selection in the interactive program velocityprogram. Which file(s) do you need to modify?

3. Execute M-file velocityprogram.m in the Command Window

- Run program velocityprogram and explain what happens in terms of specific command lines in M-Files velocityprogram.m and orbitalvelocity.m.
  ```
  >> velocityprogram
  ```
- Use the new function orbitalvelocity from the Command Window; for example:
  ```
  >> orbitalvelocity(1079, 5.3, 200, 'e')
  ```

VI. Demos

Demos of MATLAB programs written at MIT will be presented in this and next sessions.