An Introduction to MATLAB Programming

Center for Interdisciplinary Research and Consulting

Department of Mathematics and Statistics

University of Maryland, Baltimore County

www.umbc.edu/circ

Winter 2008

Mission and Goals: The Center for Interdisciplinary Research and Consulting (CIRC) is a consulting service on mathematics and statistics provided by the Department of Mathematics and Statistics at UMBC. Established in 2003, CIRC is dedicated to support interdisciplinary research for the UMBC campus community and the public at large. We provide a full range of consulting services from free initial consulting to long term support for research programs.

CIRC offers mathematical and statistical expertise in broad areas of applications, including biological sciences, engineering, and the social sciences. On the mathematics side, particular strengths include techniques of parallel computing and assistance with software packages such as MATLAB and COMSOL Multiphysics (formerly known as FEMLAB). On the statistics side, areas of particular strength include Toxicology, Industrial Hygiene, Bioequivalence, Biomechanical Engineering, Environmental Science, Finance, Information Theory, and packages such as SAS, SPSS, and S-Plus.

This document is under constant development and will periodically be updated. Standard disclaimers apply.

Acknowledgements: We gratefully acknowledge the efforts of the CIRC research assistants and students in Math/Stat 750 Introduction to Interdisciplinary Consulting in developing this tutorial series.

1 Introduction

In this tutorial we introduce the basics of programming in Matlab. We will start by reviewing the elements of structured programming; next, we will begin our discussion of Matlab programming. The flow of control and the related Matlab keywords will be mentioned along with several examples for the purpose of illustration. Another topic which we will discuss is the use of Matlab functions. At the end, we will point the audience to the appropriate sections of Matlab's documentations for more information on Matlab programming. The outline of the major objectives of this workshop is the following:

- Basics of Structured Programming
- Basic Input and Output
- Flow of Control in a Program
- Matlab Functions
- Further Help

2 Structured Programming in Matlab

Matlab provides extensive math and graphics functions along with a powerful high-level programming language. Programmers can choose to program using the classical structured programming approach, but it is also possible to do object-oriented programming in Matlab. Our discussion, however, will be limited to structured programming in Matlab.

2.1 Structured Programming

Structured programming (modular programming) is a software development paradigm in which problems are solved in a top-down fashion. As a program gets larger, it is sub-divided into sub-programs to provide code that is easier to read, debug, and maintain; also this approach facilitates reuse of commonly used procedures (functions). A natural starting point in our discussion of structured programming in Matlab will be a quick review of the elements of structured programming.

2.2 Basic Constructs of Structured Programming

Recall that an algorithm is a finite set of instructions for accomplishing a given task which given an initial state will terminate in a defined end state. And a computer program is an implementation of a given algorithm in a programming language. In

the discussion that follows, we will use Matlab programming language as our implementation tool.

There are three main programming constructs:

1. Sequence
2. Branch (Decision)
3. Loop

The Sequence construct refers to writing a group of programming statements in a sequence. The Branch construct enables us to change the flow of control *if* a given condition is satisfied, and finally the Loop construct enables the program to run a statement (or a group of statements) a number of times.

3 Some Technical Details

Before we go deeper in our discussion of Matlab programming, we take care of some technical details in this section. We will first mention naming conventions for Matlab program files; next, we will talk about Matlab's program development environment, and then have a brief discussion of input and output.

3.1 File Names in Matlab

Matlab programming codes are saved in files with extension `.m`. This gives rise to the so-called Matlab M-files. For example, a program that computes the roots of a quadratic polynomial may be saved in a file named `quadRoots.m`. An M-file may contain a Matlab script or a Matlab function; the distinction between the two types will become clear in the course of our discussion.

3.2 Development Environment

Matlab programs can be created using any text editor. In a Unix environment, one can use editors such as vi, emacs, nedit, etc. In Windows environment, notepad and alike can be used. In addition, Matlab provides a user-friendly editor where programmers can create their programs (and also debug them). To access Matlab's editor, type the following in Matlab command prompt,

```
edit
```

this opens Matlab's editor (Figure 1).

3.3 A Word on Input and Output

Before doing any programming we recall that a typical computer program (in structured programming) receives data as input, does some processing on the given data and provides output. Hence, input and output constitute an important part of any computer program. The programming examples in this tutorial do not involve interactive (or batch) data input. However, in every example, we will create some output. The commands used for providing output are `disp` and `fprintf`. We will discuss these commands at appropriate places throughout this tutorial. Although our discussion of input and output will be limited in this tutorial, we point out that Matlab provides flexible file I/O options as well; for more information, the reader can refer to Matlab's documentations.

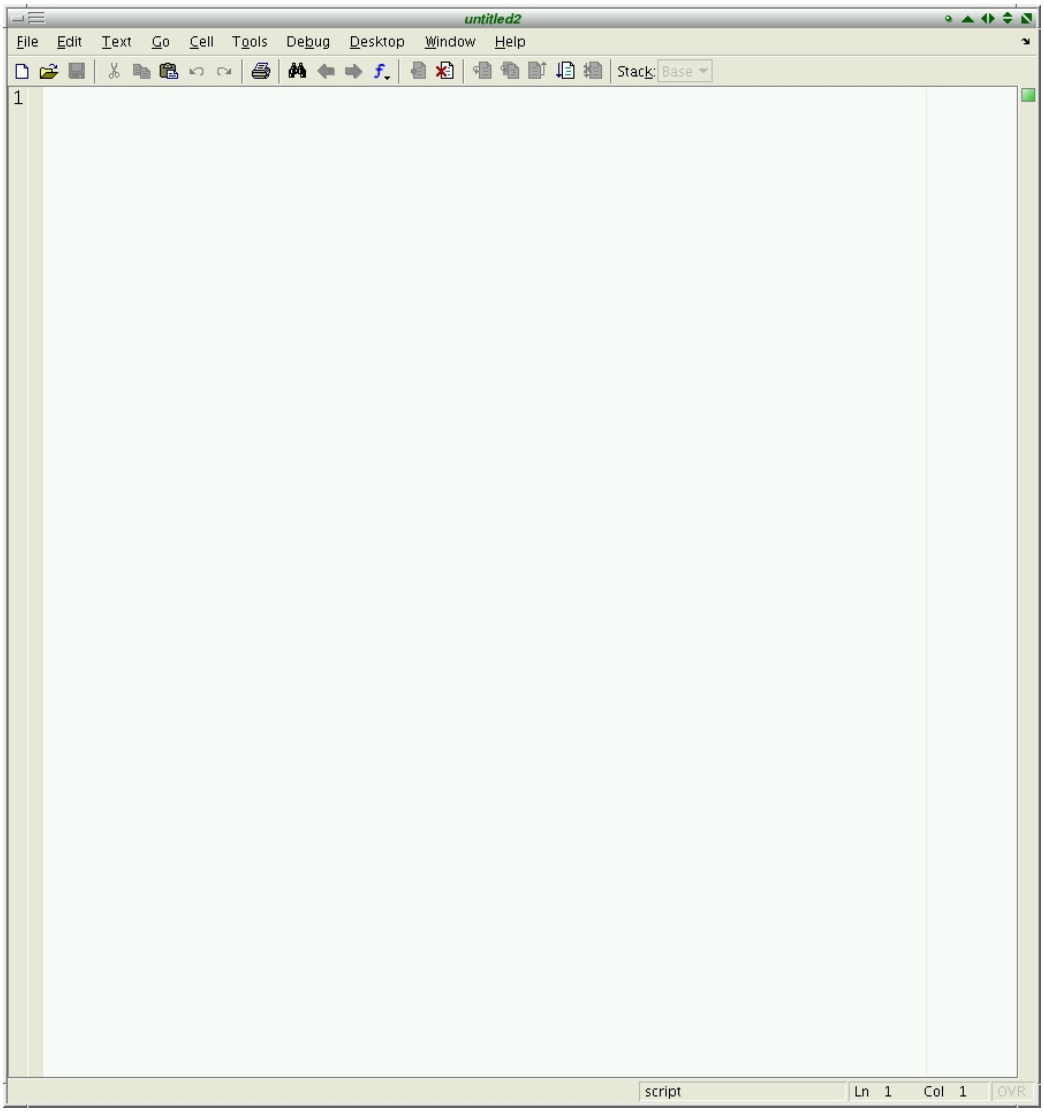

Figure 1: Matlab's program development environment.

4 Flow of Control: Branch and Loop Structures

In this section, we discuss the two main programming constructs: Branch and Loop. The discussion will involve a general description the relevant commands in Matlab along with a set of examples to illustrate the use of the commands.

4.1 Branch Structure: If Statement

The `if` statement in Matlab allows us to change the flow of control in our computer programs based on a specified condition. We will briefly describe the syntax of the `if` statement below. The simplest way of using the `if` statement is as follows:

```
if <condition>
  statement 1
  statement 2
  .
  .
  .
end
```

We also have the following option:

```
if <condition>
  statement 1
  statement 2
  .
  .
  .
else
  statement 1
  statement 2
  .
  .
  .
end
```

Note that this second form of using the `if` statement provides a way to test for a condition and execute the appropriate statement (or set of statements) if a condition is true or false. The most general way of using the `if` statement is outlined below.

```
if <condition>
  statements
elseif <condition>
  statements
elseif <condition>
```

```
    statements
  .
  .
  else
    statements
  end
```

As an example, we look at the following simple problem. Given a raw score $s \in [0, 100]$ we want to determine the corresponding letter grade. The following Matlab code solves this problem. To enter the code, open Matlab's editor by typing

```
edit score1
```

at Matlab's command prompt and type the following piece of code.

```
s = 95; % put some score here
if s >= 90
  disp('A');
elseif s >= 80
  disp('B');
elseif s >= 70
  disp('C');
elseif s >= 60
  disp('D');
else
  disp('F');
end
```

Then, to execute the code, type

```
score1
```

at the Matlab command prompt.

Note that at this point, we are hard-coding the score at the beginning of the code; this will be made more general when we discuss the use of Matlab functions where we can write a function that receives the score as input as provides a letter grade as output.

Before we end the discussion of `if` statement, we mention the relational and logical operators in Matlab briefly. When discussing the `if` statement it is natural to discuss the logical operators, AND, OR, and NOT; in Matlab this operators are denoted as in Table 1. In addition, we can use the relational operators in the conditional expressions as specified in Table 2.

Logical Operator	Matlab Representation
AND	`&&`
OR	`||`
NOT	`~`

Table 1: Logical Operators in Matlab

Relational Operator	Matlab Representation
$<$ $\leq$	`<` `<=`
$>$ $\geq$	`>` `>=`
$=$	`==`
$\neq$	`~=`

Table 2: Relational Operators in Matlab

Remark: To be more precise, we mention that in Table 1 above, the operators `&&` and `||` are scalar AND and OR operators; we can also use the logical operations of AND and OR on vectors (or matrices) in which case we use `&` and `|` respectively.

4.2 Loop Structure

In this section, we discuss the loops and will introduce `for` and `while` loops. It happens frequently that we need to execute a statement (or group of statements) for a number of times in a computer program. An example is the implementation of Newton's method for finding the roots of a given function which entails computing successive iterates of the algorithm until a desired level of accuracy is attained (or stop in case of divergence).

4.3 For Loop

To execute a statement (or group of statements) a specified number of times we use the `for` loop. The basic usage of the `for` loop is as follows.

```
for var = a : s : b
  statements
end
```

The generic variable `var` is the loop control variable whose value will range from `a` to `b`. The variable `s` provides the step-size (increment). When we omit the step-size, the default will be one. The following simple examples illustrate the use of `for` loop: The statements

```
for i = 1 : 10
  fprintf('%d ', i);
end
```

```
fprintf('\n');
```

provide the following output:

```
1 2 3 4 5 6 7 8 9 10
```

The statements

```
for i = 1 : 2 : 10
  fprintf('%d ', i);
end
fprintf('\n');
```

provide the following output:

```
1 3 5 7 9
```

The statements

```
for i = 10 : -1 : 1
  fprintf('%d ', i);
end
fprintf('\n');
```

provide the following output:

```
10 9 8 7 6 5 4 3 2 1
```

Another point to note in the above statements is the use of `fprintf` command. This commands which is used to produce structured output operates similar to `printf` in C programming language. The `%d` in the format-string specifies an integer output (we can also use `%i` instead of `%d`) and the special character `'\n'` denotes a new-line.

As an example of use of `for` loops we can solve the following simple problem: Given a natural number n form an $n \times n$ Hilbert matrix whose (i, j)-component is defined as

$$H_{ij} = \frac{1}{i + j - 1}. \tag{4.1}$$

The following small program solves the problem.

```
n = 5; % change n to any value that you like
for i = 1 : n
  for j = 1 : n
    H(i,j) = 1/(i+j-1);
  end
end
disp(H);
```

We need to open Matlab's editor by typing

```
edit hilbert1
```

at the Matlab command prompt and then enter the code in the editor.

Note that there is a Matlab function `hilb` which does the same thing as the above program. Also, we point out the use of `disp` command in the above program. The `disp` command is also used to produce output, but unlike `fprintf` we do not have much control over how the output in displayed.

4.4 While Loop

In the previous section, we saw how to use `for` loops. In this section, we discuss another type of a loop where as opposed to the case with the `for` loop we do not know in advance the number of times the statements in the loop should execute. This type of loop provides more flexibility. The following is the syntax of a `while` loop.

```
while <condition>
  statements
end
```

The statement (or statements) in the body of a `while` loop will execute for as long as the condition in the beginning of the loop is true. One common use of a `while` loop is in iterative algorithms in numerical computations which will execute until a certain level of accuracy is achieved. As a an example, the general structure of a program implementing a line-search method in unconstrained optimization is as follows:

```
x = x0;
iter = 0;
while norm(Df(x)) > tol && (iter <= maxiter)
  iter = iter + 1;
  p = ... % some search direction
  a = ... % some step length
  x = x + a*p; % update
end
```

In the above program outline, `x0` is an initial guess, `tol` is a specified tolerance, and `maxiter` is the maximum number of iterations allowed. The routine will compute successive iterates $\mathbf{x}_{n+1} = \mathbf{x}_n + a\,\mathbf{p}$, with $\mathbf{p}$ being the search direction and a the step-length. The routine stops when the norm of the gradient of the objective function, $\|\nabla(f)\| \leq tol$.

To get some practice with `while` loops, we solve the following problem in Matlab: Given a positive integer n, we want to print the Fibonacci numbers less than or equal to n. Recall that the Fibonacci sequence is the sequence of numbers given by the following recurrence relation:

$$\begin{aligned} f_0 &= 1, \\ f_1 &= 1, \\ f_{n+1} &= f_n + f_{n-1}. \end{aligned}$$

The first few Fibonacci numbers are

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots.$$

The following piece of code solves the problem.

```
n = 30; % use any number you like for n
fprev = 0;
fcurr = 1;
while fcurr <= n
  fprintf('%i ', fcurr);
  temp = fcurr;
  fcurr = fcurr + fprev;
  fprev = temp;
end
fprintf('\n');
```

You can type the code in the Matlab's editor and save it as `fibonacci.m`.

We can see that a `while` loop is a more general type of loop than a `for` loop. Anything that one can accomplish with a `for` loop can be programmed using a `while` loop but the converse is not true. However, we point out that even though `while` loop is more general purpose than `for` loop, a `for` loop is a more appropriate command to use when we know in advance (outside the loop) how many times the statements in the body of the loop are to be executed.

5 Matlab Functions

In this section, we discuss Matlab functions. Using functions to break down a large program to smaller and more manageable units is the heart of modular programming and hence the discussion that follows is an integral part of our introduction to Matlab programming. We start by looking at the general structure of a Matlab function. In general, an m-file containing a Matlab function begins with the keyword `function`; in the function header we specify the name of the function and the input and output parameters. The following shows a typical situation.

```
function [out1 out2 ...] = funcName(in1, in2, ...)
  statements
```

For example, we can write a function that receives as input a raw score $s \in [0, 100]$ and returns a letter grade. Recall that we solved this problem earlier in this tutorial. All we need to do is to modify our code as below:

```
function g = score(s)
if s >= 90
  g = 'A';
elseif s >= 80
  g = 'B';
elseif s >= 70
  g = 'C';
elseif s >= 60
  g = 'D';
else
  g = 'F';
end
```

Note that the file containing the function `score` should be named `score.m`. If the file name and the function name do not match, Matlab will use the file name as the function name and ignore the function name given in the function header. To ensure the function works correctly, the following instructions (or alike) can be entered in Matlab's command prompt.

```
score(91)
score(88)
score(72)
score(65)
```

We can also make the routine generating the Fibonacci sequence more general by writing a function `fibo(n)` which receives n as input and prints Fibonacci numbers less than or equal to n (the reader is encouraged to do this as an exercise).

In Matlab, there are various ways of defining functions. Some examples include sub-functions, inline functions, anonymous functions, etc. While the discussion of the

different types of Matlab functions is outside the scope of this tutorial, the reader can refer to Matlab's documentations for further information. The various ways of defining functions in Matlab provide great flexibility in creating highly structured and elegant code, and hence to become an effective Matlab programmer it is essential to learn more about Matlab functions.

6 Getting Further Help

Matlab help facility provides extensive documentations describing various aspects of the software package. In particular, the Getting Started guide provides a rather detailed introduction to Matlab programming which can be a suitable starting point for beginners (see Figure 2). This Help Browser can be accessed by selecting `Help -> Matlab Help` on the top menu of the main GUI window.

Moreover, to view the documentations for a specific command, the user can use the `help` or `doc` command. For example, the following command will open the help browser and show the help page for the `while` command.

```
doc while
```

Basic documentation is also available within the command window by saying

```
help while
```

For many basic functions, the information provided in both ways is identical. For more sophisticated functions however, the additional examples provided in the Help Browser are often helpful (this is particularly true for graphics functions).

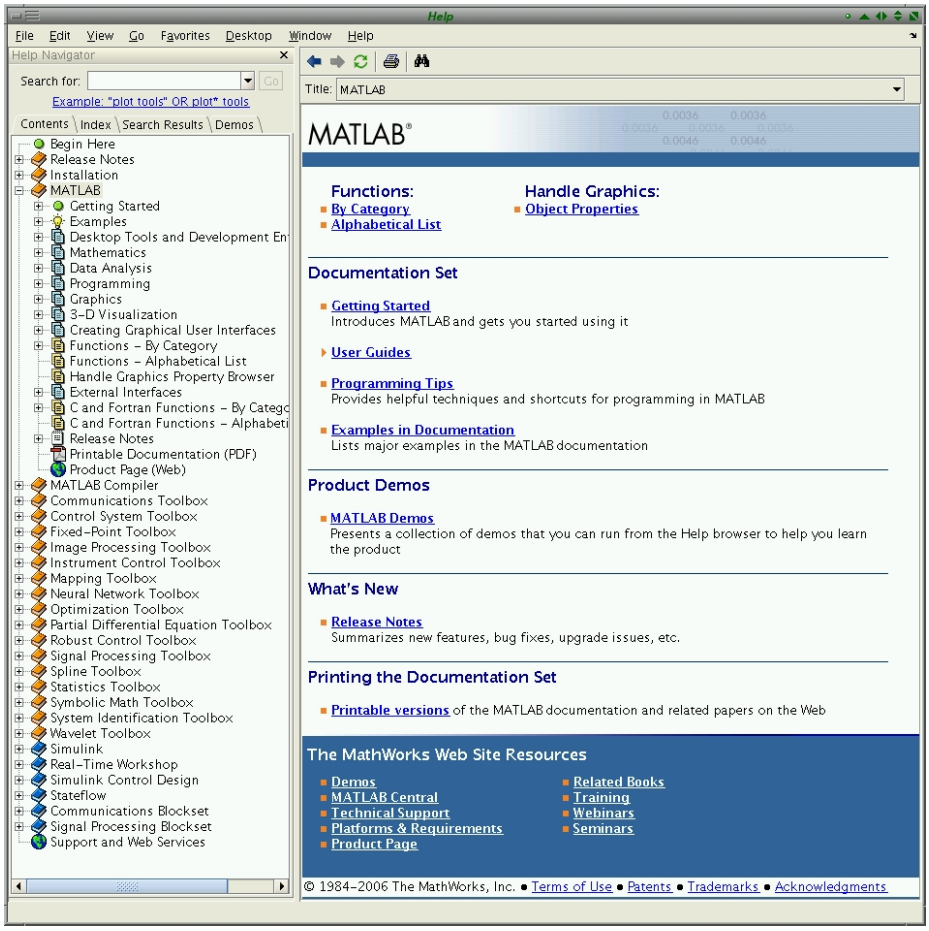

Figure 2: Matlab's Help-desk after expanding the MATLAB chapter.