Plotting and Flow Controls

1 Basic Plotting

1.1 Introduction

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

1.2 Creating Simple Plots

The basic MATLAB graphing procedure, in two dimensions, is summarized as follows. Let's take a vector of x-coordinates, x = (x_1,...,x_n), and a vector of y-coordinates, y = (y_1,...,y_n). Each pair of points in x and y corresponds to the x and y values of a point in a typical graph. Therefore, you need to prepare x and y in an identical array form; namely, x and y are both row vectors or column vectors of the same length. The MATLAB command to plot y vs x is plot(x,y) where each of the points is joined by a straight line.

1.3 Plotting a Trigonometric Function

1.3.1 Description

Let's start by looking at the basic sine function plotted from 0 to 2π radians. To plot the function sin(x) on the interval [0, 2π], we first create a vector of x-values ranging from 0 to 2π, then compute the sine of these values, and plot the result. When you are creating the vector containing the x-values, you can indicate at what increment you want the values to be plotted by using the colon operator. For example, if you want a plot ranging from 0 to 2π with a coordinate plotted every $\pi/4$ units, the vector of x-values would be **x**=[0:pi/4:2*pi].

Note: If you do not input an increment when using the colon operator,such as **x**=[0:2*pi], MATLAB automatically increments by 1. This may not result in a smooth curve. For the following example, **use an increment of $\pi/100$** for the vector ranging from 0 to 2π.

1.3.2 Exercise 1

Use MATLAB to plot the function sin(x) on the interval given in section 1.3.1. Try changing the size of the increment to smaller and bigger values to see the difference in the resulting plot.

Hint: Use either the colon operator or `linspace`.

1.4 Customizing Your Plot

1.4.1 Specifying line styles and colors

By default, MATLAB uses different line styles and colors to distinguish the data sets plotted in the graph. However, you can change the appearance of these graphic components or add annotations to the graph to further explain your data. Therefore, it is possible to specify line styles, colors, and markers using the `plot` command. The general syntax is `plot(x,y,'style-color-marker')`, where `'style-color-marker'` is a triplet of values from Table 1. For a complete listing of the combinations of colors and symbols, enter `help plot` or `doc plot`.

1.4.2 Multiple Data Sets in One Plot

Multiple (x,y) pairs arguments create multiple graphs with a single call to plot. For example, these statements plot three related functions of x: y1 = 2cos(x), y2 = cos(x), and y3 = 0.5cos(x), in the interval $0 \leq x \leq 2\pi$.

Table 1: Elementary functions

Symbol	Color	Symbol	Line Style	Symbol	Marker
k	Black	-	Solid	+	Plus sign
r	Red	- -	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	-.	Dash-dot	.	Point
c	Cyan	none	No line	x	Cross
m	Magenta			s	Square
y	Yellow			d	Diamond

1.4.3 Exercise 2

Type these commands into MATLAB to see what the plot looks like.

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,'-.')
>> xlabel('X Values')
>> ylabel('Cosine Functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')
>> title('Typical Example of Multiple Plots')
>> grid on
>> axis([0 2*pi -3 3])
```

Note how the command `axis` is used above to reset the automatic range of the axes. Alternatively, the command `hold` can be used to plot several graphs on the same plot. With the command `hold on`, each new plot is overlayed on the previous ones. The command `hold off` stops further overlaying of figures. The command `grid on` turns on a grid and the command `grid off` turns it off. The command `legend`, which creates a legend on the plan, is helpful when multiple plots are used. However, it can be replaced by the command `gtext`, where you can put text anywhere in the plot window using the mouse. See the MATLAB documentation for more information about these commands.

2 Flow Controls

MATLAB has several flow control constructs. Here are the most commonly used:

- `if, else`, and `elseif`
- `for`
- `while`
- and others, such as `switch, break, continue`, etc.

2.1 Relational and Logical Operators

A relational operator compares two variables by determining whether a comparison is true or false. Relational operators are shown in Table 2.

Note that the "Equal to" relational operator consists of two equal signs (==) with no space between them, since = is reserved for the assignment operator.

Table 2: Relational and Logical Operators

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
|	OR operator
~	NOT operator

2.2 Output Commands

MATLAB automatically generates an output when commands are executed without a semicolon. In addition to this automatic display, MATLAB has several commands that can be used to generate displays and outputs.

Two commands that are frequently used to generate outputs are `disp` and `fprintf`. The main differences between these two are summarized in Table 3.

Table 3: `disp` and `fprintf` Commands

`disp`	. Simple to use
	. Provides limited control over the appearance of output
`fprintf`	. Slightly more complicated than `disp` command
	. Provides total control over the appearance of output

Example:

```
fprintf('pi = %7.5f\n', pi)
pi = 3.14159
```

2.3 The "if...end" Structure

2.3.1 Description

If you want to make a decision in MATLAB using some condition, you can use the if construct. The simplest form of the if statement is,

```
if expression
   statements
end
```

where "expression" is a condition that can contain a logical or relational operator and "statements" are commands that will be executed if the the "expression" is true.

Here are some examples based on the quadratic equation ($ax^2 + bx + c = 0$).

```
1. discr = b*b - 4*a*c;
   if discr < 0
      disp(Warning: discriminant is negative, roots are imaginary);
   end

2. discr = b*b - 4*a*c;
   if discr < 0
      disp(Warning: discriminant is negative, roots are
      imaginary)
```

```
   else
      disp(Roots are real, but may be repeated)
   end
```

```
3. discr = b*b - 4*a*c;
   if discr < 0
      disp(Warning: discriminant is negative, roots are
      imaginary)
   elseif discr == 0
      disp(Discriminant is zero, roots are repeated)
   else
      disp(Roots are real)
   end
```

Note:

- `elseif` has no space between `else` and `if`
- no semicolon is needed at the end of lines containing `if, else,` and `end`
- indentation of if block is not required, but it makes it easier to read the code
- the end statement is required

2.3.2 Exercise 3

If an integer x is given, write an if statement that will display the words "even" or "odd" based on whether the integer is even or odd. (Hint: Use mod function)

2.4 The For Loop

2.4.1 Description

In the for loop, the execution of a command is repeated for a fixed and predetermined number of times. The command syntax is

```
for variable = expression
    statements
end
```

Usually "expression" is a vector of the form i:s:j; where i and j are initial and final data indices respectively, while s is an increment.

Example of a for loop:

```
>> a = ones(1,4);
>> for ii = 1:4
   a(ii) = ii;
end

>> a  =

      1   2   3   4
```

Note that without the semicolon after the statement within the for loop, the vector **a** will be displayed during each iteration of the loop.

2.4.2 Exercise 4

Given a matrix A = [2 5 7; 4 1 8], write a for loop that will replace all the even numbers with 0. (Hint: You may need a for loop inside another for loop. This is called a "nested loop")

2.5 The While Loop

2.5.1 Description

This loop is used when the number of passes is not specified beforehand. The looping continues until a stated condition is satisfied. The while loop has the form:

```
while expression
      statements
end
```

The statements inside the while loop are executed as long as "expression" is true.
Example of a while loop:

```
n = 5
while n < 10
    disp('n is less than 10')
    n = n + 1;
end
```

"n is less than 10" will be displayed 5 times because that is how many times the loop will cycle through before n = 10, and the loop ends.

2.5.2 Exercise 5

Use a while loop to determine how many terms in the sum 1+2+3+... are needed for the sum to exceed one million.