Chapter 2

Scalars and Variables

In this chapter, we will discuss arithmetic operations with scalars (numbers, really an array with only one element) and variables. Scalars can be used directly in calculations and equations, or be assigned to variables.

2.1 Arithmetic Operators with Scalars:

Operation	Symbol	MATLAB	C++
Addition	+		
Subtraction	-		
Multiplication	*		
Right Division	/	5 / 3 evaluates to 1.6667	5 / 3 evaluates to 1
Left Division	\	5 \ 3 = 3 / 5 evaluates to 0.6	NA
Exponentiation	^	5 ^ 3 means 5^3 evaluates to 125	NA

2.2 Order of Precedence

Order	MATLAB	C++
First	Parentheses. For nested parentheses, the innermost are executed first.	same
Second	Exponentiation - right to left	NA
Third	Multiplication, division in order left to right	same
Fourth	Addition and subtraction in order left to right	same

2.3 Examples:

The following examples demonstrate the order of operations

```
>> 7+5/2-6          %division is performed first followed by
                    % addition and subtraction

ans =               % if no variable is denoted, the result is
                    % stored in "ans"
```

```
    3.5000


>> (7+5)/2-6              %the evaluation order can be
                          % modified by the use of ( )

ans =

     0


>> (7+5)/(2-6)

ans =

    -3

>> 7^(1/3) + 32^0.3      % ( ) are performed first
                         % followed by exponentiation
                         % followed by the addition
ans =

    4.7414

>>
```

2.4 Numerical Precision of MATLAB Output

All arithmetic is done to double precision. MATLAB automatically prints integer values as integers and floating point numbers to four decimal digits of accuracy. Exponential format is automatically used when the value of a number falls outside the range of numbers that can be printed using the default format.

However the default formatting can be altered by using the format command. Once the format command is entered, all the output that follows is displayed in the specified format. Several of the available formats are listed below.

Command	Description	Example

format	`Default Same as short`	`>>format`
		`>> 150/7`
		
		`ans =`
		
		`21.4286`

Command	Description	Example
format short	Scaled fixed-point format with 4 decimals 0.001 <= number <= 1000	>>format short >> 150/7 ans = 21.4286
format long	Scaled fixed-point with 14 decimals 0.001 <= number <= 100	>>format long >> 150/7 ans = 21.42857142857143
format short e	exponential format with 4 decimal places	>> format short e >> 150/7 ans = 2.1429e+001
format long e	exponential format with 14 decimal places	>> format long e >> 150/7 ans = 2.142857142857143e+001
format short g	Best of fixed or floating point format with 5 digits	>> format short g >> 150/7 ans = 21.429
format long g	Best of fixed or floating point format with 15 digits	>> format long g >> 150/7 ans = 21.4285714285714
format bank	Fixed format for dollars and cents Two decimal places	>> format bank >> 150/7 ans = 21.43

Command	Description	Example
format rat	Approximation by ratio of small integers	>> format rat >> 10/30 ans = 1/3
format compact	Suppress extra line-feeds	
format loose	Puts the extra line-feeds back in	

2.5 Identifiers

An identifier is a programmer-defined name that represents some element of your program, i.e., data, function name, object name. Variable names are examples of identifiers. Please make them readable and indicative of what the variables are used for. You may be tempted to declare variables with names like Bob -- please restrain yourself. The rather nondescript name, Bob, gives no clue as to the variable's purpose. (But it may refer to yourself--ego issues??, father--good for you, but not here, person you are in love with--nice, but.) It is better to use variable names that describe what it represents. This way of coding helps produce self-documenting programs, which means you get an understanding of what the program is doing just by reading its code.

Here are the specific rules that must be followed with all identifiers:

Identifiers	
MATLAB	**C++**
begin with a letter, followed by any combination of letters, numbers, and the underscore character. Only the first 63 characters are significant; if more than 63 are used, the remaining characters will be ignored.	begin with a letter or underscore followed by any combination of letters, numbers, and underscore. No limit on the number of characters.
Case Sensitive name, NAME, Name are all different	Case Sensitive name, NAME, Name are all different
Cannot be a keyword	Cannot be a keyword
Avoid using the names of built-in functions for a variable, i.e., avoid using: cos, sin, pow, sqrt, etc.. Once a function name is used to define a variable, the function cannot be used	Avoid using the names of built-in functions for a variable. Once a function name is used to define a variable, the function cannot be used.

2.6 Data Types: most common

Computer programs collect pieces of data from the real world and manipulate them in various ways. There are many different types of data--character, strings, whole numbers (integers), fractional numbers, boolean, etc. C++ offers many data types. However, in the very broadest sense, there are only two: numeric and character.

	MATLAB	C++
Most common	double char	bool, char, short, int, unsigned int, long, float, double
double	scalars or arrays of 64-bit double-precision floating-point numbers. Can hold real, imaginary, or complex values. The real and imaginary components of each variable can be positive or negative numbers in the range of 10^{-308} to 10^{308}	Can hold real values. The variable can be positive or negative in the range of 10^{-308} to 10^{308}
char	scalars or arrays of 16-bit values, each representing a single character. Used to hold character strings. Automatically created whenever a single character or a character string is assigned to a variable name.	a single character

2.7 Declarations:

In the early days of computing, language design was heavily influenced by the decision to use compilation or interpretation as a mode of execution. For example, some compiled languages require that programs must explicitly state the data-type of a variable at the time it is declared or first used. On the other hand, some languages take advantage of the dynamic aspects of interpretation to make such declarations unnecessary. In FORTRAN 77, variables starting with "I" -"N" are an integer; all others were float, unless otherwise indicated. In many scripting languages, data-type of a variable is determine by what is stored into it. Nowadays, the differences between the two styles of execution have largely been dealt with by more sophisticated designs. Most so-called interpreted languages use an intermediate representation.

A long explanation to lead into: MATLAB does not require variable declarations; C++ does.

MATLAB:	C++:
No declaration needed. The variable is created when a value is assigned to a variable. The datatype is determined from the assignment. variable = numerical value variable = computed expression	Must be declared datatype variable;

`x = 1.23`
`y = 3 * x - 12`
`str = 'This is a character string'`
 	
Note: Those are single quotes '...'
NOT double quotes as in C++
	`double x = 1.23;`
	`string str = "This is a character string";`

2.8 Predefined Variables

A number of frequently used variables are already defined within MATLAB.

Variable	Represents
`ans`	Variable that has the value of the last expression that was not assigned to a specific variable. If the user does not assign the value of an expression to a variable, MATLAB automatically stores the result in `ans`
`pi`	The value of Π
`eps`	The smallest difference between two numbers. Equals 2^(-52), which is approximately 2.2204e-016
`Inf and inf`	used for infinity
`i`	Defined as sqrt(-1) which is: 0 + 1.0000i
`j`	Same as i
`NaN`	Stands for Not-a-Number. Used when MATLAB cannot determine a valid numeric value. For example 0.0/0.0 and inf - inf
`realmax`	largest positive floating point number
`realmin`	smallest positive floating point number

Variable	Represents

`clock`	Current date and time as date vector.
	CLOCK returns a six element date vector containing the current time and date in decimal form:
	CLOCK = [year month day hour minute seconds]
	The first five elements are integers. The seconds element is accurate to several digits beyond the decimal point.
	FIX(CLOCK) rounds to integer display format.
	`>> fix(clock)`
	`ans =`
	`2004 3 29 22 35 24`

`date`	Current date as date string.
	S = DATE returns a string containing the date in dd-mmm-yyyy format.
	`>> s = DATE`
	
	`s =`
	
	`28-Mar-2004`

These variables can be redefined. The variables `ans, pi, eps, inf, NaN, clock, and date` are usually NOT redefined. The other predefined variables i and j are sometimes redefined (common in association with loops) when complex numbers are not involved in the application.

2.9 Commands for Managing Variables

The following commands can be useful to eliminate variables from memory or to obtain information about variables that have been used. These commands are usually used within the `Command Window`.

Command	What it Does
`clear`	Removes all variables from memory
`clear x y z`	Removes only variables `x, y,` and `z` from memory
`who`	Displays a list of the variables currently in memory
`whos`	Displays a list of the variables currently in memory and their size together with information about their bytes and class

2.10 I/O

There are many ways to get data into variables. One way is to define them explicitly, by assigning in values from computations, and by using data that are loaded into MATLAB from an external file. In this section, we are concerned only with I/O of variables.

MATLAB has functions for the basic input of variables from the keyboard and for formatted output of variables.

2.11 Interactive input

There are several command that can be used for interactive input: `input, keyboard, menu,` and `pause`. These can be used within the `Command Window`, a script or function file for interactive user input.

2.11.1 Input

The input command is one of the easiest, simplest, and useful for basic input.

2.11.1.1 Numeric Input

```
numApplies = input('How many apples >> ')
```

gives the user the prompt in the text string and then waits for input from the keyboard. The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R. If the user presses the return key without entering anything, **input** returns an empty matrix.

MATLAB	C++

`numApples = input('How many apples >> ')`
 	
`How many apples >> <CR>`
 	
`>> numApples`
 	
`numApplies =`
 	
`[]`
	`int numApples;`
	`cout << "How many apples >> ";`
	`cin >> numApples;`
	
	`//if a <CR> is hit, the program`
	`//waits forever for an integer`

`numApples = input('How many apples >> ')`
 	
`How many apples >> 5`
 	
`>> numApples`
 	
`numApplies =`
 	
`5`
	`int numApples;`
	`cout << "How many apples >> ";`
	`cin >> numApples;`

2.11.1.2 STRING INPUT

```
name = input('What is your name: >> ','s')
```

gives the prompt in the text string and waits for character string input. The typed input is not evaluated; the characters are simply returned as a MATLAB string.

MATLAB	C++

`>> name = input('What is your name: >> ', 's')`
`What is your name: >> Mary Lou`
 	
`name =`
 	
`Mary Lou`
	`string name;`
	`cout << "What is your name: >> ";`
	`cin >> name;`
	
	`//if  a <CR> is hit, the program`
	`//waits forever for an integer`

The text string for the prompt may contain one or more `'\n'`. The `'\n'` means skip to the beginning of the next line. This allows the prompt string to span several lines. To output just a `'\'` use `'\\'`.

2.11.2 Keyboard

The command keyboard inside a script or function file (M-file) returns control to the keyboard at that point where it encounters the command. The execution of the function or script is NOT terminated. The command window prompt changed '>>' to 'k>>' to show the status. At this point, you can check variables already computed, change their values, and issue any valid MATLAB commands. Control is returned to the script or function by typing the word `return` and pressing the `return` key <CR>.

This command is useful for debugging purposes. Sometimes, in long programs, you may want to check intermediate results, plot them, see if the computation is behaving as expected, and then let the execution continue if all is well with the world.

2.11.2.1 MENU

The command menu generates a menu of choices for user input.
`CHOICE = MENU(HEADER, ITEM1, ITEM2, ... )` displays the `HEADER` string followed in sequence by the menu-item strings: `ITEM1, ITEM2, ... ITEMn.` Returns the number of the selected menu-item as `CHOICE`, a scalar value. There is no limit to the number of menu items.

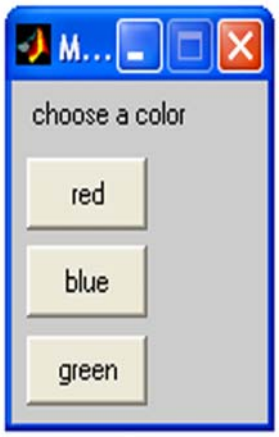
`CHOICE = MENU(HEADER, ITEMLIST)` where `ITEMLIST` is a string cell array is also a valid syntax.

On most graphics terminals `MENU` will display the menu-items as push buttons in a figure window, otherwise they will be given as a numbered list in the command window (see example, below).

Command window example:

```
>> menuChoice = MENU('choose a color','red','blue','green')
```

displays on the screen:



When the user chooses a value in response to the prompt, it is returned (i.e., `menuChoice = 2` implies that the user selected `Blue`).

2.11.2.2 PAUSE

The command pause temporarily halts "pauses" the current process. It is used with or without an optional argument:

pause	suspends execution of the current script or function until the user "hits any key" (I'm still trying to find that key labeled "any"--anybody find it yet?)
pause(n)	suspends execution for n seconds

2.11.3 Output to Monitor

Data can be displayed on the monitor or written to files in a number of ways.

2.11.3.1 UNFORMATTED OUTPUT OF TEXT AND OF STRINGS: DISP

One command that is frequently used to generate output on the screen is the **disp** command. The **disp** command is used to display text, or the value/elements of a variable without displaying the name of the variable. The format is:

```
disp('text as string') or
disp(name of the variable)
```

Every time the **`disp`** command is executed, the display it generates appears on a new line.

If you want to save data to a file, the **`disp`** command will not work. See `fprintf`.

```
gEarth = 9.81;  %acceleration on Earth
disp('========================================')
disp('Earth:')
```

```
disp(gEarth)

========================================
Earth:
    9.8100
```

C++ Equivalent

```
gEarth = 9.81;
cout << "========================================" << endl;
cout << "Earth: " << endl;
cout << gEarth;
```

2.11.4 Formatted Output of Variables: fprintf

MATLAB uses the function `fprinf` for formatted output of messages and numbers. The general syntax for this function is:

```
count = fprintf(fid,format,A,...)
```

`count = fprintf(fid,format,A,...)` formats the data in the real part of matrix `A` (and in any additional matrix arguments) under control of the specified `format` string, and writes it to the file associated with file identifier `fid`. `fprintf` returns a count of the number of bytes written.

Argument `fid` is an integer file identifier obtained from `fopen`. (It may also be `1` for standard output (the monitor) or `2` for standard error. See `fopen` for more information.) Omitting `fid` causes output to appear on the monitor.

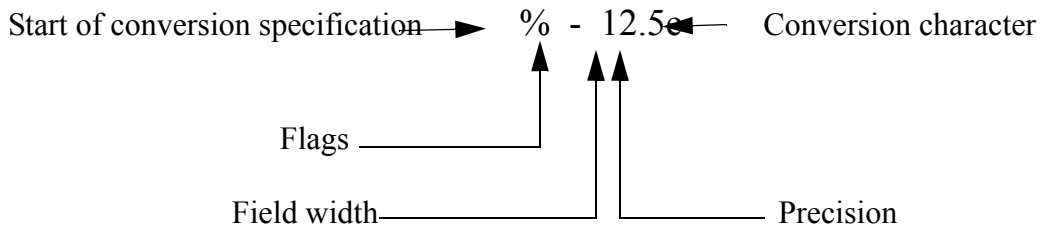
2.11.5 Format String

The format argument is a string containing C language conversion specifications. A conversion specification controls the notation, alignment, significant digits, field width, and other aspects of output format. The format string can contain escape characters to represent non-printing characters such as newline characters and tabs.

Conversion specifications begin with the % character and contain these optional and required elements:

- Flags (optional)
- Width and precision fields (optional)
- A subtype specifier (optional)
- Conversion character (required)

You specify these elements in the following order:



2.11.6 Flags

You can control the alignment of the output using any of these optional flags.

Character	Description	Example
A minus sign (-)	Left-justifies the converted argument in its field.	`%-5.2d`
A plus sign (+)	Always prints a sign character (+ or -).	`%+5.2d`
Zero (0)	Pad with zeros rather than spaces.	`%05.2d`

Examples	
MATLAB Code	**Outut**
`x = 5.1200`	
`>> fprintf('example%-6.2frun',x)`	`example5.12  run`
`>> fprintf('example%+6.2frun',x)`	`example +5.12run`
`>> fprintf('example%06.2frun',x)`	example005.12run

2.11.7 Field Width and Precision Specifications

You can control the width and precision of the output by including these options in the format string.

Character	Description	Example
Field width	A digit string specifying the minimum number of digits to be printed.	%6f
Precision	A digit string including a period (.) specifying the number of digits to be printed to the right of the decimal point.	%6.2f

2.11.8 Conversion Characters

Conversion characters specify the notation of the output.

Specifier	Description
`%c`	Single character
`%d`	Decimal notation (signed)
`%e`	Exponential notation (using a lowercase `e` as in `3.1415e+00`)
`%E`	Exponential notation (using an uppercase `E` as in `3.1415E+00`)
`%f`	Fixed-point notation
`%g`	The more compact of `%e` or `%f`, as defined in [2]. Insignificant zeros do not print.
`%G`	Same as `%g`, but using an uppercase `E`
`%i`	Decimal notation (signed)
`%o`	Octal notation (unsigned)
`%s`	String of characters
`%u`	Decimal notation (unsigned)
`%x`	Hexadecimal notation (using lowercase letters `a-f`)
`%X`	Hexadecimal notation (using uppercase letters `A-F`)

Conversion characters %o, %u, %x, and %X support subtype specifiers. See Remarks (below) for more information.

2.12 Examples

MATLAB/C code	Output
`y = 25.123456` `x = 123.450000`	
`>> fprintf( 'example %6f',y)`	`example 25.123456`
`>> fprintf( 'example %6.2f',y)`	`example  25.12`
`>> fprintf( 'example %6.4f',y)`	`example 25.1235`
`>> fprintf('example %e',x)`	`example 1.234500e+002`
`>> fprintf('example %E',x)`	`example 1.234500E+002`
`>> fprintf('example %g',x)`	`example 123.45`
`>> fprintf('example %6.4g',x)`	`example  123.5`

MATLAB/C code	Output

x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f %12.8f\n', y)
fclose(fid)
	creates the text file: exp.txt
	containing:
	0.00 1.00000000
	0.10 1.10517092
	...
	1.00 2.71828183

>> fprintf('A unit circle has circum- ference %g.\nAnd area %g',2*pi*1,pi*1*1)	A unit circle has circumference 6.28319. And area 3.14159
Print a single quote >> fprintf(1,'It''s Friday.\n')	It's Friday.
>> B = [8.8 7.7; 8800 7700] >> fprintf(1,'X is %6.2f meters or %8.3f mm\n', 9.9, 9900,B)	X is 9.90 meters or 9900.000 mm X is 8.80 meters or 8800.000 mm X is 7.70 meters or 7700.000 mm
Convert to HEX >> a = [6 10 14 44]; >> fprintf('%9X\n',a + (a<0)*2^32)	6 A E 2C

2.12.1 Escape Characters

This table lists the escape character sequences you use to specify non-printing characters in a format specification.

Description	MATLAB	C++
Backspace	\b	\b
Form feed	\f	\f
New line	\n	\n
Carriage return	\r	\r
Horizontal tab	\t	\t
Backslash	\\	\\
Single quotation mark	\'' (two single quotes)	\’
Percent character	%%	NA
alarm	NA	\a
Double quote	NA	\”

2.13 Remarks

The fprintf function behaves like its ANSI C language namesake with these exceptions and extensions.

If you use fprintf to convert a MATLAB double into an integer, and the double contains a value that cannot be represented as an integer (for example, it contains a fraction), MATLAB ignores the specified conversion and outputs the value in exponential format. To successfully perform this conversion, use the fix, floor, ceil, or round functions to change the value in the double into a value that can be represented as an integer before passing it to fprintf.

The following, non-standard subtype specifiers are supported for the conversion characters `%o, %u, %x, and %X`.

`b`	The underlying C data type is a double rather than an unsigned integer. For example, to print a double-precision value in hexadecimal, use a format like '`%bx`'.
t	The underlying C data type is a float rather than an unsigned integer.

For example, to print a double value in hexadecimal use the format '%bx'

The fprintf function is vectorized for nonscalar arguments. The function recycles the format string through the elements of A (columnwise) until all the elements are used up. The function then continues in a similar manner through any additional matrix arguments.

Note `fprintf` displays negative zero (-0) differently on some platforms, as shown in the following table.

	Conversion Character		
Platform	**%e or %E**	**%f**	**%g or %G**
PC	0.000000e+000	0.000000	0
SGI	0.000000e+00	0.000000	0
HP700	-0.000000e+00	-0.000000	0
Others	-0.000000e+00	-0.000000	-0

2.14 FILE I/O

As with many other programming languages, there are times when you don't want to type all those numbers from the keyboard or you want to use that output to feed into another program or routine. This is when you need FILES.

2.14.1 File Input/Output

Command
% open an existing file fid = fopen(filename, permission) Permissions are: 'r' read 'w' write (create if necessary) 'a' append (create if necessary) 'r+' read and write (do not create) 'w+' truncate or create for read and write 'a+' read and append (create if necessary) 'W' write without automatic flushing 'A' append without automatic flushing fid1 = fopen("data.txt") % open for reading fid2 = fopen ("temp.txt",'w') % open for writing
% close an open file fclose state = fclose(fid) %fid is the id connected with the file state = fclose(fid1) %closes "data.txt" from above example
% read binary data from a file fread If you wish further info, use "help fread"
%writes binary data to a file fwrite
%read formatted data from a file fscanf S = fscanf(fid,'%s') reads (and returns) a character string. A = fscanf(fid,'%5d') reads 5-digit decimal integers.
% writes formatted data to a file fprintf
%read strings in specified format sscanf for further info, use "help sscanf"

Command
% writes data in formatted string sprintf
% reads a line from file discarding newline character fgets
% reads a line from file including newline character fgetl
% rewinds a file frewind
% sets the file position indicator fseek
% gets the current file position indicator ftell
% inquires file I/O error status ferror

2.15 References

[1] Kernighan, B.W. and D.M. Ritchie, *The C Programming Language*, Second Edition, Prentice-Hall, Inc., 1988.

[2] ANSI specification X3.159-1989: "Programming Language C," ANSI, 1430 Broadway, New York, NY 10018.