Topic 5: Introduction to Programming in Matlab

CSSE, UWA

Starting with a great calculator...

• MATLAB is a high level language that allows you to perform calculations on numbers, or arrays of numbers, in a very simple way.
• For example at the prompt within the MATLAB command window you can type
  $$\gg 3 + 4$$
• MATLAB will evaluate this for you and report the answer
  $$\text{ans} = 7$$
• Since MATLAB is interpreted anything you can do in a program you can do from the command line, and vice versa
  • quick and easy prototyping

Variables

• You can also work with variables
  $$\gg a = 4$$
  $$\gg b = 6$$
  $$\gg c = a * b$$
• These instructions set a variable called ‘a’ to the value of 4, a variable called ‘b’ to 6, and then multiplies ‘a’ and ‘b’ and stores the answer in a variable called ‘c’. (Computers use ‘*’ to indicate multiplication).
• The variables ‘a’, ‘b’ and ‘c’ remain available for use in other calculations.
• Note that if the value of ‘a’ is changed later in the program the value of ‘c’ remains unchanged (unlike a spreadsheet program where all values would be recalculated when you change one value)

Comments

• An important part of programming is documenting your work. A program should convey both a message to the computer and a message to the human programmers who read the program.
• You need to document your programs so that your assumptions and solution techniques can be readily examined for their correctness by you and others.
• MATLAB uses the the percent symbol (%) to indicate the start of a comment statement.
• Anything on a line after a % symbol is ignored by MATLAB.
  $$\gg 3 * 4 6 + 7 \text{ (I would never do this)}$$
  $$\text{ans} = 12$$
Suppressing output

• By default, Matlab will display the answer to every assignment statement you enter. For example:
  >> halfPi = pi / 2
  halfPi =
  1.5708
• You can suppress the outputting of the answer by adding a semicolon (;) to the end of a statement. For example, the command:
  >> halfPi = pi / 2;
will still set the value of the variable halfPi to 1.5708, but will not print the answer on the screen.
(Important when you come to dealing with big matrices!)

Simple Expressions

• All the commands listed above form expressions. An expression is a valid statement that MATLAB can interpret or 'understand'.
• A statement is valid if it satisfies the syntax (the 'grammar') of the language.
• As the course progresses we will cover more and more of MATLAB's syntax, but at this stage you will find that any simple mathematical expression, expressed correctly, will have valid syntax.
• For example:
  b = 4 * (a + 3); % is valid
  c = a +          % invalid -incomplete expression.
  d = (b + 3)) / 6; % invalid -unmatched brackets.

Operators

• The operators used for writing basic mathematical expressions are:
  • +  addition
  • -  subtraction
  • *  multiplication
  • /  division
  • ^  exponentiation
• A common mistake can be to omit the multiplication operator when writing expressions, for example writing
  b = 4 (a+3);  rather than
  b = 4*(a+3);

Operator Precedence

• Operators have precedence, that is, in an expression involving several operators some will be applied before others. This ensures uniqueness in interpreting an expression.
• Expressions are evaluated from left to right with exponentiation having highest precedence, followed by multiplication and division with equal precedence, and addition and subtraction at an equal (lower) precedence.
  a = b+c*d^e/f-g;
  will be evaluated as
  a = b+(c*(d^e)/f)-g;
• Where appropriate use brackets (even if they are not strictly necessary) to make expressions easier to read and interpret.
Identifiers/Variables

- An **identifier** or a **variable** in a programming language gives a name to a specific memory location. When a command such as
  \[ x = 5.2 \]
  is evaluated the system chooses a memory location, associates it with the identifier \( x \), and stores the value 5.2 in that location. Thus the name ‘\( x \)’ serves as a place-name for the location in memory where our value is stored.
- We never have to worry what the actual memory location is.

Variable Names

- MATLAB has some rules regarding the names of identifiers/variables (more commonly we use the term ‘variable’).
  1. Variable names must start with a letter, this can then be followed by any number of letters, numerals or underscores. Punctuation characters are not allowed as many of these have a special meaning to MATLAB.
  2. Variable names are case sensitive. Items, items and iTeMs are all separate variables.
- (Most computer languages have these rules)
- MATLAB maintains some special variables, some of these are:
  - \( \text{ans} \) - the default variable name used for results of calculations.
  - \( \text{pi} \) - the ratio of circle circumference to its diameter.
  - \( i \) and \( j \) - the square root of -1.

Types

- Most computer languages have the concept of a **type** of a variable. For example, a variable may be an **integer**, a **character**, or a **floating point number**. The type specifies how the value of that variable is stored in the computer.
- For example, some of the types available in the Java programming language include:

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Stores a character using 16 bits (unsigned).</td>
</tr>
<tr>
<td>int</td>
<td>Stores an integer using 32 bits (two’s complement).</td>
</tr>
<tr>
<td>float</td>
<td>Stores a floating point value (a non-integer) using 32 bits (4 bytes). These 32 bits are used differently from the storage of an integer.</td>
</tr>
<tr>
<td>double</td>
<td>Stores a floating point value using 64 bits (double precision)</td>
</tr>
</tbody>
</table>

Type Casting

- When two variables of different types are combined together (e.g. added together), some form of conversion, or **casting**, has to be performed to match the types.
- Without casting, strange results will occur because in binary terms, we are adding “apples and oranges” – combining things that are represented in completely different ways.
- In the case of adding an integer to a float, typically the integer would be **promoted** to become a float. The result would be a float.
- Fortunately, Matlab has very few types. The types we are likely to encounter are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Characters</td>
</tr>
<tr>
<td>double</td>
<td>Double precision numbers</td>
</tr>
<tr>
<td>complex</td>
<td>Complex numbers represented using two doubles</td>
</tr>
</tbody>
</table>
Weak Typing

- In almost all cases, Matlab automatically determines the type that should be used for a variable and casts as necessary.
- Matlab also automatically handles any conversions between types that need to be made to evaluate an expression.

Weakly typed

- Unlike many programming languages, Matlab does not require a programmer to make any distinction between integers, doubles, or complex numbers. (This is great, but could there be disadvantages? How could strong typing be an advantage?)
- For example, we could type:
  ```matlab
  >> a = 'B';       % Characters enclosed in quotes
  >> b = 1;
  >> c = 1.5;
  >> d = 2 + 3*i;   % Can also write d = 2 + 3i
  >> e = c*d;       % Matlab automatically handles the complex multiplication
  ```

Getting to know the type in Matlab

- The `whos` command prints out information about the currently active variables. For example:
  ```matlab
  >> whos
  Name      Size         Bytes  Class
  a         1x1              2  char array
  b         1x1              8  double array
  c         1x1              8  double array
  d         1x1             16  double array (complex)
  e         1x1             16  double array (complex)
  Grand total is 5 elements using 50 bytes
  ```

Release Memory

- The `clear` `variableName` command will remove a variable from memory:
  ```matlab
  >> clear a
  >> whos
  Name      Size         Bytes  Class
  b         1x1              8  double array
  c         1x1              8  double array
  d         1x1             16  double array (complex)
  e         1x1             16  double array (complex)
  Grand total is 4 elements using 48 bytes
  ```

The Distinction between Assignment and Equality

- When we write:
  ```matlab
  >> a = 2;
  >> b = a;
  >> a = a*3;
  ```
  the `=` sign in this code does not mean equality in the mathematical sense (clearly it cannot). The `=` symbol really means the operation of assignment, sometimes read “becomes equal to”.

- We can read the expression `a = a*3` as:
  1. Read the value stored at `a`.
  2. Multiply the value by `3`.
  3. Assign this calculated value to the memory location referred to by `a`.
A close look at assignment

- The value of 2 stored at the memory location referred to by \( b \) remains unchanged.

\[ \begin{array}{c}
\text{Memory} \\
\cdot\cdot\cdot 5 \cdot\cdot\cdot \\
a \quad b
\end{array} \]

- Pseudo-code often uses the symbol ← (or :=) to denote assignment rather than the = symbol, so that the distinction between the two is made clear.
  
  eg. \( a \leftarrow a \times 3 \)

- Testing for equality between two values is done with a different operator (more later).

Script files

- Matlab allows commands to be entered interactively at the command prompt, but this is not really appropriate for extended sequences of commands. Instead, we can create a script file - a file that contains a sequence of commands (a script) for Matlab to follow.

- A script file is simply a text file that contains the sequence of Matlab commands/expressions to follow. Script files can be created using any text editor (e.g. Emacs) or by using the Matlab internal editor.

- By convention, script files should be saved with a .m ending.

- Script files are executed by typing the name of the script without the .m ending.

- The commands in the script file are executed as if they had been typed in at the command prompt.

Many expressions on one line

- You can put several expressions on one line with the semi-colon (;) operator (though I wouldn’t normally advise it).

- For example:
  
  \[
  \begin{array}{c}
  \gg a = 'B'; \ b = 1; \ c = 1.5;
  \end{array}
  \]

- The ; acts as a separator between expressions.

- If you want the result of each expression in the line to be printed to the screen, commas (,) are used as statement separators. For example:
  
  \[
  \begin{array}{c}
  \gg a = 'B', \ b = 1, \ c = 1.5
  \end{array}
  \]

One Expression on Many lines

- You can break an expression over more than one line using ... to indicate a continuation.

- For example:
  
  \[
  \begin{array}{c}
  \gg a = 1 \ldots \ + 2 \ldots + 3;
  \end{array}
  \]

- In general, seek to format your expressions to aid readability and understanding.