Variables and Functions

Jeff Freymueller

September 27, 2010

Topics for Today

- A quick review of variables
- How MATLAB handles variables
- Advanced variable types
  - Cell arrays
  - Structures (structs)
- Functions
  - Built-in functions
  - Designing your own functions
  - Sensible use of functions

Variables Review: name vs. value

- Every variable has a name and a value – don't mix up the two.
  - The variable is a box in which you can store something. The name is written on the box, and the value is what you store inside.
  - Some languages have a few restricted words that cannot be used for variable names, usually because these are the names of commands or control structures.
- MATLAB really treats all variable values, even strings, as arrays of numbers.

Variable review: assignment vs. reference

- Are you putting the value in, or taking it out?
- Assignment is when you store a value in a variable
  - `deg2rad = pi/180;`
- Reference is when you access the value.
  - “`pi`” above is a reference. It is replaced by the value of the variable called `pi`.
  - Some languages use a special symbol when you reference the value of a variable, but MATLAB does not

Audience Participation

For each statement, identify all variable assignments and references:

- `h = 6.62606896*10^-34;`
- `h_bar = h/(2*pi);`
- `b == a_row(4);`
- `c = (a^2 + b^2)^0.5;`
- `j = j + 1;`

Variables in MATLAB

- MATLAB treats all variables as arrays (vectors or matrices). Program’s roots are in linear algebra.
  - Scalars are just 0-dimensional arrays (single values)
  - Values assigned using = : “a_row = [1 2 3]”
  - Values assigned using = : “a_col = [1; 2; 3]”
- Most of the time, you don’t need to worry about the variable type, MATLAB handles it invisibly.
- But you do have to remember that there is a difference between a row vector and a column vector.

Matrix, Row Vector, Column Vector

- A matrix $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- A row vector $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

- A column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

  – They are not the same:

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 + 4 + 9 = 14$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

Bracketology

- MATLAB uses three different kinds of brackets, parentheses, braces, all meaning different things
- [ ] Square brackets
  - Vectors, arrays and matrices are contained inside
- ( ) Parentheses
  - Access a particular element of an array by putting the indices inside parentheses
- { } Braces or Curly brackets
  - Like parentheses, except for cell arrays

Bracketology

- [ ] Square brackets
  - Vectors, arrays and matrices are contained inside
  - `a_row = [1 2 3]; a_col = [1; 2; 3];`
  - `also_a_col = [1 2 3]';`
- ( ) Parentheses
  - Access a particular element of an array by putting the indices inside parentheses
  - What is the value of `a_row(2)`?
  - `a_row(3) = 5;`
- { } Braces or Curly brackets
  - Like parentheses, except for cell arrays
  - `my_cell{1} = 'Label';`
  - `my_cell{2} = [1 2 3];`

Math Operators

- +, -, *, .*, /, ,/, ^, .^, \, .\
- Basic math operators (+, -, *, /, ^) operate on arrays.
  - * is actually matrix multiplication, / will invert a matrix
  - a/b is a*inverse(b) while a\b is inverse(a)*b
- Element-wise operators (.*, ./, .^, .\) operate on an element by element basis
  - `c = a.*b` means c(i) = a(i)*b(i), for all elements I
  - Why is there no .+ nor .- ??
- Order of operators is normal math order. If you are not sure, use parentheses to be sure.
- `help ops` or `doc ops`

Special “Numbers”

- MATLAB handles complex numbers seamlessly
  - `5 + sqrt(-1)` evaluates to 5.0000 + 1.0000i
  - If you do not define a variable “i”, MATLAB will use that symbol for sqrt(-1).
- Not a Number (NaN) is a very handy “number”
  - Use NaN to represent missing values
    - DO NOT use “9999” for missing values!
  - Any arithmetic operation with NaN produces NaN
  - The function “isnan” finds all the NaNs in its argument
    - idx = isnan(has_a_nan)

Array vs. Cell array

- Cell arrays behave a bit differently than regular arrays.
  - Every element of a regular array must be the same kind. Not so for cell arrays. *Each element of a cell array is a container that can hold any one thing*.
  - Cell arrays are really useful for strings, and also are returned by some functions that read files.
  - You can do numerical operations across regular arrays, but not cell arrays.
- Cell arrays let you make an array of structures, for example.

Structures

- Suppose you have a set of variables that go together. A structure (or struct) lets you package them together, making it easy to keep track of things.
- A ***struct*** has one or more ***fields***, which are named, and each can store a value (or vector, or array, or a struct, or …)
- You define a struct by naming its fields and assigning a value to each (or use [] for an empty value).
- Access a field like this: weather.year, weather.temp(5)
  - Or getfield(weather, “year”) or getfield(weather, name) where name is a variable.

Struct Example 1

- Let’s suppose you have a data file, and you need to keep track of the file meta-data as well as the data values. For example, a SAR image. Possible fields are:
  - amplitude, phase : arrays of data values
  - x, y : positions of each (georeferenced) pixel
  - satellite_name
  - track_num, frame_num : identify the image
  - More meta-data

Example 1 continued

- Define the structure:
  - My_image = struct(‘satellite_name’, [], ‘track_num’, [], ‘frame_num’, [], ‘x’, [], ‘y’, [], ‘amplitude’, [], ‘phase’, []);
  - This is a set of label, value pairs. [] means an empty array. You could put values here, but be aware that MATLAB won’t always do what you expect if some values are arrays.
- Now populate it with values
  - My_image.satellite_name = ‘Envisat’;
  - My_image.track_num = 1745;
- Access the values
  - Small_x_values = (My_image.x < -500);

Struct Example 2

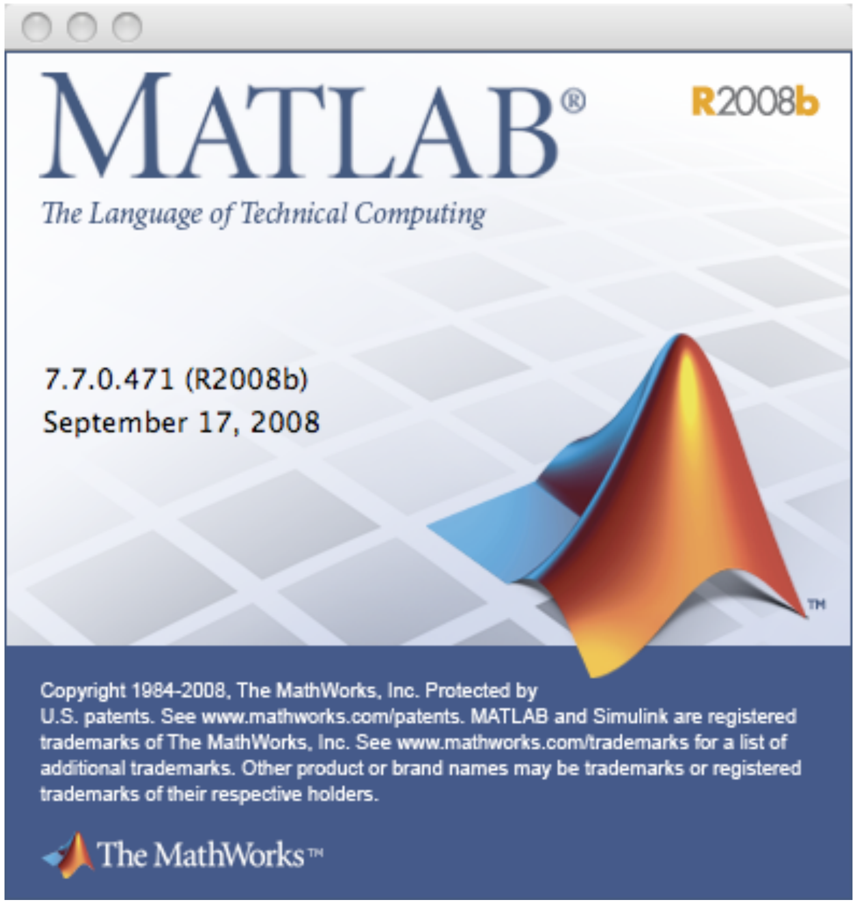
- Suppose you have a data set that has a number of arrays of the same size. There might also be some other information.
  - For each day of year, you have temperature, pressure and humidity readings
- What are the fields?
  - Year (scalar)
  - Day_of_year (array)
  - temperature, pressure, humidity (arrays)
- In this case, it makes sense to make an array of structs, which will allow you to do some numerical operations on the elements, like finding the minimum, maximum or mean.

Example 2

- The easy way to make an array of structs is to put all the numerical values into cell arrays and use these to define the struct.
  - Note: some input routines give you the data in cell arrays already!
  - You'll get an error message if the cell arrays are different sizes.
- weather = struct('year', 2009, 'day_of_year', {225, 230, 235}, 'temp', {64, 69, 58}, 'pressure', {30.1, 30.5, 29.5}, 'humidity', {0.64, 0.34, 0.88});
- This produces a 3 element array of structs. You can access the values in different ways:
  - weather(2).temp
  - median([weather.pressure])

Let’s Explore MATLAB for a while

- Structs
  - Let’s look at the “timeseries” struct
  - This is an example of packaging a variety of related information into one “container”
  - If I want to store multiple timeseries, I could make a cell array and store each timeseries in one element.



7.7.0.471 (R2008b)
September 17, 2008

Copyright 1984-2008, The MathWorks, Inc. Protected by U.S. patents. See www.mathworks.com/patents. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Functions

- What is a function?
  - A set of mathematical operations that take some input values ("variables") and produce one or more output values.
    - Remember every value can be a scalar or a vector or matrix
  - A little black box of code that takes some inputs and produces one or more outputs
    - Some of the boxes in your flowchart might be implemented as functions
- Examples of built-in functions: sqrt, isnan, find, eye, zeros, ones, size, inv

Defining your own functions

- Save your function in its own .m file, named for the function
- Start with a function declaration
  - function output = my_func(input1, input2)
  - Function [out1, out2] = two_out(in1, in2, in3)
  - ***Be sure to give your function a descriptive name!***
- End with “return” (not required, but good habit)
- You can use any number of inputs and any number of outputs
  - The arguments to the function are ***passed by value***
    - `c = my_func(a, b);`
  - If you change the values of the input variables inside the function, those changes are lost when you exit
  - Some languages (like fortran) pass arguments by reference, so you can change any variable passed to a subroutine/function
- Any comment lines immediately afterward are used by help

A simple function example

Let’s make a “cuberoot” function.

- Create a file called cuberoot.m

```
function out = cuberoot(in);
out = in^(1/3);
return
```

- Use the function
  - `y = cuberoot(x);`
- What’s going on when you call the function
  - MATLAB takes the value of x, and assigns it to the variable in on the inside of the function
  - When the function exits, the value of out is assigned to y

Scope of variables

- The function exists in its own separate little space. It interacts with its calling routine only through the arguments it is passed and the values it returns.
  - It can only modify the values it returns, and not the arguments.
- Variables used inside the function are created when the function is called, and thrown out when it is done.
- You can re-use variable names inside a function that are also used somewhere outside.

More of your own functions

- As long as MATLAB knows where to find it, your function becomes just as much a part of the language as the built-in functions
  - Build up a set of your own useful functions!
    - MATLAB will always find functions in the current directory
    - Use addpath to specify other directories where it should look for your functions
  - Assemble small pieces into bigger tools!
- Be sure to use sensible names!