GEOS F436/636 Beyond the Mouse

Christine (Chris) Waigl
University of Alaska Fairbanks – Fall 2018
Week 2: more on variables, matrices, vectors...

Topics for week 2

- Review / clean-up week 1
- More data structures:
  - Vectors & matrices
  - Cell arrays
  - (looking ahead)Structs
- Binary operators

Review & clean-up from week 1

Programming languages reminder

A programming language is an artificial language that is made up of a set of symbols (**vocabulary**) and grammatical rules (**syntax**) to instruct a machine.

Interpreter	**Compiler**
Translates human-authored code to machine-executable code one statement at a time. Can be accelerated with a just-in-time (JIT) compiler.	Translates human-authored code in a separate step as a whole. Faster.
Often comes with an interactive command line, also called REPL (read-eval-print loop).	Generates intermediate code files which can be combined with specialized library files.
Continues translating until the first error is encountered. Then stops.	Generates complex log files that make it harder to debug.

Comments, and uncommenting

- Comments are pieces of text in your script (or the command line) that are not executed.
- You can **(you should!)** use them to explain what you're doing for the benefit of your future self, or any collaborators. You can use them to annotate your code with, for examples, units of variables you are using. You can use them to add code samples that didn't work **(but label them as "this didn't work"!).**
- A comment is everything that comes after the % sign.
- Sometimes code samples come with commented lines that you are supposed to turn into code by removing the % sign. This is calle ***uncommenting.***

Helpful hints when using the IDE

- Very useful keys: **arrow-up** (in the Command window: retrieves previously typed commands) and **tab** (completes what you have started typing)
- Any folder on your computer can be "added to the MATLAB path". This means that MATLAB will find the functions stored in them.
- The ellipsis (dot-dot-dot: ...) is used to continue lines:

```
>> mystr = ["my text ", "is very very very very very ", ...
"long"];
```

- Three useful commands:
  - `clear all` ← removes variables, functions from your Workspace
  - `clc` ← removes everything from your Command Line (but the history is stll there!)
  - `close all` ← closes pop-up windows
- Some people like to start each script with `"clear all; clc; close all".`

Indenting. Pseudocode.

Pseudocode is a notation that resembles a simplified programming language. You use pseudocode, often written by hand, to sketch out program design or work mentally through an algorithm (= a step-by-step method for solving a task).

Indenting (adding space at the beginning of line, often in sets of 4 spaces) is used to make your program structure more apparent. Some languages require indenting schemes (Python is very strict about them!), others, like MATLAB, are more lax. Imagine indents like the outline of a written piece. Pseudocode example:

```
IF viscosity_of_lava between 10^3 and 10^4:
    lava_type = 'basaltic'
ELSE IF viscosity_of_lava < 10^8:
    lava_type = 'andesic'
ELSE:
    lava_type = 'rhyolite'
```

More on numeric variables

Computer memory can only store data in the form of binary data: smallest unit of data, which everything is build out of, is a **bit**: a single number that can be 0 or 1.

Integers. Imagine a 2x4 egg carton, where each position can be occupied (1) or not (0):

1	1	0	0
1	1	1	0

← 5 eggs left

Each configuration corresponds to a (binary) number.

00000001 = 1, 000000010 = 2, 00000011 = 3 ...

How many are there? Answer: 2^8 = 256.

⇒ We can use this scheme to express either the numbers from 0 to 255 (unsigned integer) or from -128 to +127 (signed integer). int(8) or uint(8)

Real numbers. This is much harder for real numbers. Computer memory is finite, so we can *never* represent a transcendental number like π precisely. Some computers use fixed-point representations like xxxx.xxxx (fixed number of places). But this makes it impossible to represent very small *and* very large numbers at the same time. Floating point: ± xxxx * 2^yyyy .

Matrices and vectors

The name "MATLAB" comes from "matrix laboratory"

Everything is already a matrix! (Simple numbers = 1x1 matrix.)

```
>> a = [1, 2, 3, 4, 5]   ← row vector
>> b = [1 2 3 4 5]       ← also a row vector, also:
>> c = [1; 2; 3; 4; 5]   ← column vector

Try:
>> a == b               ← remember, this tests for identity
>> b == c
>> a == c'               ← the apostrophe is the transpose operator
>> a * a
>> a * c                ← did you expect this? It's the dot product
>> a .* a               ← ... and this is the element-wise product
```

Vectors are just 1xn or nx1-dimensional matrices

- 1 x n ← the first number is the # of rows, the second the # of columns
- n x 1 ← this is a column vector

Other ways to make a vector. Play around with these, change the numbers...

```
>> 1:10
>> 1:2:10
>> linspace(0, 100, 1)        ← also try logspace(...)
>> 3 * 1:10
>> a = exp(1:10)
>> [a, 1, 2, 3]
>> [a'; 1; 3;]
```

Accessing the elements of a vector

1. Accessing them individually:

```
>> a = linspace(1, 10, 50)
>> a(17)
>> a(17:2:28)    ← this is sometimes called slicing
```

2. Accessing all of them in order

```
>> for element = a
  disp(a);  ← start this line with an indent
end
```

simple plotting

```
>> xs = linspace(-10, 10, 101);
>> ys = -3 * xs.*xs + 10;
>> plot(xs, ys);
```

Remove semicolons (;) to take a look at the values.

```
>> grid;
>> title('My firsts plot!!!');
```

... and now matrices

We can build matrices out of vectors, or use functions that create matrices:

```
>> A = [[1, 2]; [3, 4]]
>> B = ones(4)
>> C = zeros(4, 2)
>> D = C'
>> E = magic(6)
>> F = rand(2, 4)
>> G = [F, D]
>> H = [F; D]
>> A(1, 2)
>> F(1:2, 3:4)
```

Try:

- Element-wise multiplication with .*
- Element-wise addition, subtraction
- Apply matrix multiplication with *
- Do you need the . for addition/subtration?
- length and size functions

You will get many error messages! That's ok! Read them!

Remember character arrays?

```
>> my_name = 'chris waigl';
>> a_letter = 'a';
>> a_letter + 1                    ← what do you expect?
>> a_letter == 97                  ← what do you expect?
>> my_name + 1            ← huh?
```

Characters are represented (encoded) via integers! And they are treated as such. (This is where the new MATLAB strings, available in v. 2016 and later, are useful. But we can concern ourselves with them later!).

Also remember the concatenation operator: [my_name, a_letter]

What if I want to group data of different types?

Cell arrays are similar to matrices, but allow you to mix data types. They are created with the {} operator.

```
>> C = {'one', 'two', 'three'; ones(2), 2, 3} ← 2 x 3 cell array
>> C{2, 3}      ← individual element, use {}
>> C{2, 1}      ← ALSO an individual element, which is a matrix
>> C(2, 1:2)    ← make a sub-array, use()
```

Structs (structure arrays) give you mixed data *with labels.* Super useful!

```
>> mydata.date = '2018-03-01';
>> mydata.unit = 'm/s';
>> mydata.description = 'measured velocity of UAV';
>> mydata.values = [2.4, 3.5, 4.6, 1.6, 2.9];
>> mydata
```

Relational operators

Operators are equivalent to functions. They're just written differently (= have a different **syntax**)

```
>> plus(3, 4)
>> isequal(vec1, vec2)
```

← the same as `3 + 4`

← use this for vectors instead of == : it compares the whole vector. More with `doc is*`

Logical (Boolean) operators

Some logical operations (try replacing "true" by "false", or 1, or 0):

```
>> b = 2 == 3          ← this is a false statement, so b evaluates to 0
>> b && true           ← logical AND: BOTH have to be true
>> b || true           ← logical OR: ONLY ONE has to be true
>> xor(b, true)        ← exclusive OR: EXACTLY ONE has to be true
>> ~b                  ← logical negation ("not b")
```

For vectors, use & (AND) and | (OR) for **element-wise** operation

```
>> [ 0 1 1 0 ] & [ 1 0 1 0 ] ← evaluates to [ 1 1 1 0 ]
```

Try all of these by replacing `b` with `ans`. We will explore in the lab...

Don't forget to use "save your workspace".

You can save your workspace in a .mat file. And load it next time!

BRING YOUR OWN USB STICK.

AND/OR FIND A NICE WORKSTATION TO RETURN TO

Optional reading

- Hahn & Valentine ch.2 through 2.6, ch. 6.1, 6.2, 6.3. **or** Attaway, ch 1.4, 1.5, 3.1, 5.2, 5.3
- If you're interested in how you can represent rational and real numbers in fixed and floating point representation (a fascinating topic that we don't have time to go into very deeply), this is a good, gentle introdution: http://www.science.smith.edu/dftwiki/index.php/CSC231_An_Introduction_to_Fixed-_and_Floating-Point_Numbers