Introduction to Matlab

First part: Morse code

Pouyan R. Fard & Prof. Dr. Stefan Kiebel

6. April 2017

Assignment

Basic value assignment:

- `A = 5` ← Variable name: A, value: 5.
- `B = 7`
- `C = A+B` ← Variable name: C, value: The values of A and B added.

Naming variables:

- Combinations of letters, numbers and underscores.
- Must begin with a letter
- Case sensitive

Strings:

- `First_Name = 'Dario';`
- `Last_Name = 'Cuevas';`
- `Full_Name = [First_Name, , Last_Name];` ← This is called concatenation.

Vectors

A vector is a collection of numbers. Examples:

- The heights of every person in the room.
- The number of pieces of bread eaten in the past four days.
- The reaction times of every subject in an experiment.

What to do with vectors:

- Defining a vector: `First5Numbers = [1,2,3,4,5];` ← Notice the square brackets used to define. Every element is separated by a comma.
- Reading the value of one element (indexing): `First5Numbers(3) <Enter>` ← round brackets (parentheses) for indexing.
- Changing the value of one element: `First5Numbers(3) = -20` ← Just like indexing, but with an equal sign.
- Operations and changing of values, all in one: `First5Numbers(5) = First5Numbers(1) + First5Numbers(4);`

Indexing vectors

Say `A = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];` We can:

- Read the first three elements: `A(1:3) <Enter>`.
- Change the first two to some other value: `A(1:2) = A(5);`
- Read the first and fifth elements: `A([1,5]) <Enter>` ← Notice the square brackets inside the parentheses.
- Change the values of elements 1 and 5-10 to -1: `A([1,5:10]) = -1`.
- Note that `A(1:3)` is the same as `A([1:3])`. The square brackets are not necessary but they do not hurt either.

Indexing vectors

Say `A = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];` We can:

- Read the first three elements: `A(1:3) <Enter>`.
- Change the first two to some other value: `A(1:2) = A(5);`
- Read the first and fifth elements: `A([1,5]) <Enter>` ← Notice the square brackets inside the parentheses.
- Change the values of elements 1 and 5-10 to -1: `A([1,5:10]) = -1`.
- Note that `A(1:3)` is the same as `A([1:3])`. The square brackets are not necessary but they do not hurt either.

Mini exercises. Do each in just one line of code.

1. Generate the vector A as before.
2. Change the values of the second and seventh elements to -1.
3. Set the values of elements 1 to 5 to those of elements 6 to 10.

Concatenation

For two vectors `A = [1, 2, 3];` and `B = [-1,-2];` we can:

- Add new elements: `A = [A,4];` or `B = [0,B, -3];` ← Use square brackets.
- Join them together (called concatenation): `C = [A,B];`
- For an empty vector `D = [];`, add more things: `D = [D,1:10];`

Concatenation

For two vectors `A = [1, 2, 3];` and `B = [-1,-2];` we can:

- Add new elements: `A = [A,4];` or `B = [0,B, -3];` ← Use square brackets.
- Join them together (called concatenation): `C = [A,B];`
- For an empty vector `D = [];`, add more things: `D = [D,1:10];`

Mini exercise:
For the vectors A and B above, create a vector E whose elements are: the first two elements of A, the first element of B, the last element of A and the last element of B, in that order.

The colon operator

Notice that `A([1,2,3,4])` is the same as `A(1:4)`. In reality, `[1,2,3,4]` is the same as `1:4`. That is the colon operator. The syntax is: start:step:end. Examples:

- Numbers for 1 to 100: `A1to100 = 1:100;`
- Even numbers from -10 to 10: `Evens = -10:2:10;` ← How many elements does it have?
- Numbers from 0 to 1 in steps of size 0.001:
  `ZeroToOne = 0:0.001:1;`
- Numbers from 10 to 1 (order matters!):
  `Inverse_order = 10:-1:1;`

The colon operator

Notice that `A([1,2,3,4])` is the same as `A(1:4)`. In reality, `[1,2,3,4]` is the same as `1:4`. That is the colon operator. The syntax is: start:step:end. Examples:

- Numbers for 1 to 100: `A1to100 = 1:100;`
- Even numbers from -10 to 10: `Evens = -10:2:10;` ← How many elements does it have?
- Numbers from 0 to 1 in steps of size 0.001:
  `ZeroToOne = 0:0.001:1;`
- Numbers from 10 to 1 (order matters!):
  `Inverse_order = 10:-1:1;`

Mini exercises. Do each in just one line of code.

1. Generate a vector called `Big_Vector`, with all numbers from 1 to 100.
2. Set all odd-numbered elements to zero.
3. Multiply the even-numbered elements of `Big_Vector` by 2.

Cells

Cells are like vectors, but instead of storing numbers in each element, you can store whatever you want. Examples:

- `MyCell{1} = 5;` ← notice that indexing of cells is with curly brackets.
- `MyCell{2} = 1:10;` ← The second element of the cell is a vector.
- `MyCell{3} = [5,2,-10];` ← Another vector.
- `MyCell{4} = 'We can also store strings in cells';`.
- `MyCell{5} = [];` ← This means that element 5 is 'empty'.

You can index cells like with vectors, except with curly brackets:

- `Five = MyCell{1};`
- `MyCell{2} = 2*MyCell{2};`

When an element of a cell is a vector, you can access the elements of the vector:

1. `MyCell{2}(5) <Enter>` will return the fifth element of the second element of MyCell.
2. `MyCell{4}(1:5) <Enter>` will return the phrase 'We can'.

Writing the dictionary

For this part of the exercise, check the section called The Dictionary in the Morse.pdf file.

For loops

To repeat a command many times, use the For loop:

```
for i = start:step:end
  commands to be repeated
end
```

For loops

To repeat a command many times, use the For loop:

```
for i = start:step:end
  commands to be repeated
end
```

Examples:

- ```
  sumI = 0;
  for i=1:100
    sumI = sumI + i;
  end
  ```
  ```

For loops

To repeat a command many times, use the For loop:

```
for i = start:step:end
  commands to be repeated
end
```

Examples:

- ```
  sumI = 0;
  for i=1:100
    sumI = sumI + i;
  end
  ```
- ```
  X = [1, -10, 5, 32, 1];
  sumX = 0;
  for i=1:5
    sumX = sumX + X(i);
  end
  ```

For loops

To repeat a command many times, use the For loop:

```
for i = start:step:end
  commands to be repeated
end
```

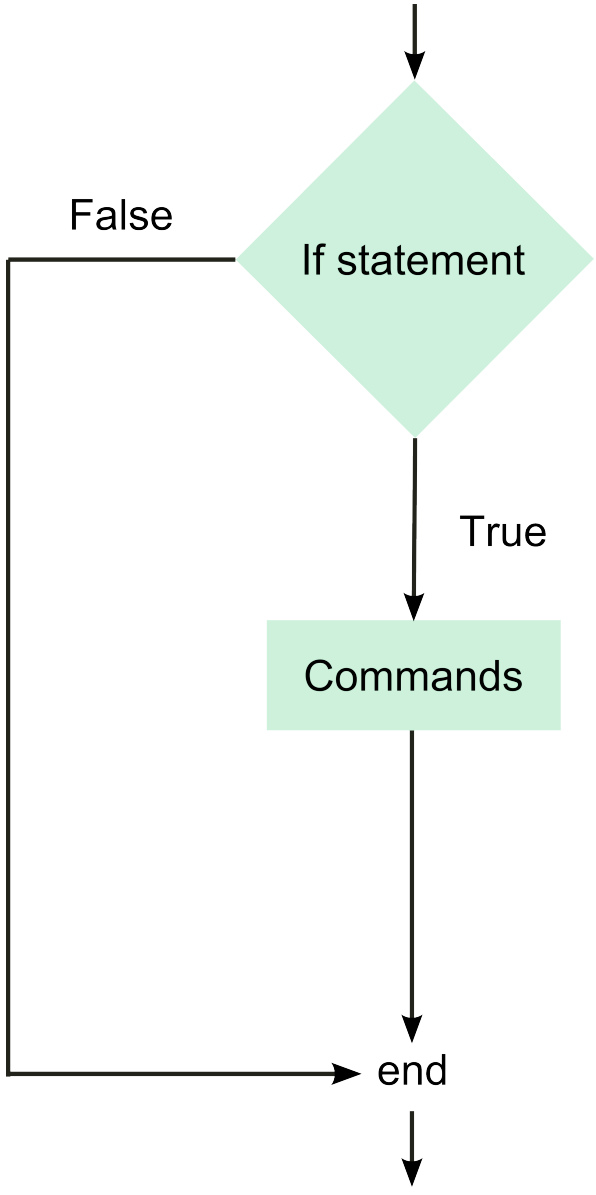
Examples:

- ```
  sumI = 0;
  for i=1:100
    sumI = sumI + i;
  end
  ```
- ```
  X = [1, -10, 5, 32, 1];
  sumX = 0;
  for i=1:5
    sumX = sumX + X(i);
  end
  ```

Mini exercises.

1. Calculate the multiplication of all the elements in a vector.
2. Calculate the area of circles of radii 1, 3 and 10. Save the results in a vector called Areas ($A = 2\pi r^2$).
3. Calculate the volume of cylinders whose circular faces have the areas as in (2), and with heights 10, 15 and 25. ($V = Ah$). Do not save the results; just print them using the command `display(V)`, where V is the current calculated volume.

If statements

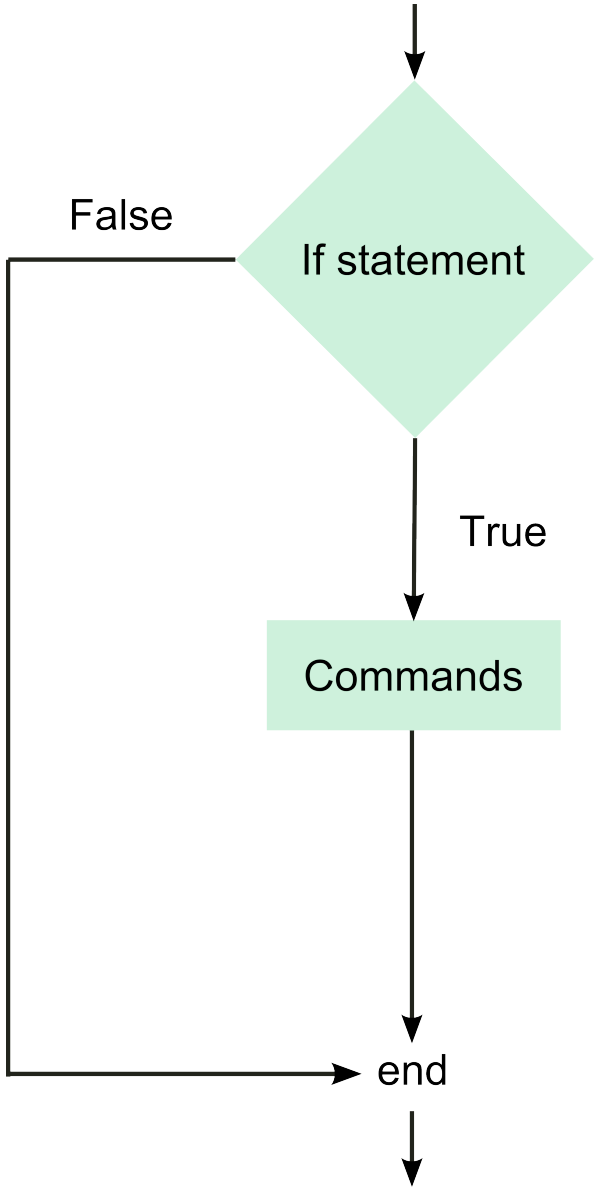


When you want to run a piece of code only under specific circumstances, the IF statement is your friend. Examples:

1. Check whether a number is positive before obtaining its square root:

```
if A>0
  SQRofA = sqrt(A);
end
```

If statements



When you want to run a piece of code only under specific circumstances, the IF statement is your friend. Examples:

1. Check whether a number is positive before obtaining its square root:

```
if A>0
  SQRofA = sqrt(A);
end
```

2. If an element of a vector is negative, make it positive:

```
vect = [1, -5, 10, 52, -0.1];
for i=1:numel(vect)
  if vect(i)<0
    vect(i) = -vect(i);
  end
end
```

Conditionals

The following symbols can be used as conditions in an IF statement, for two numbers A and B:

1. $A < B$: A smaller than B
2. $A > B$: A bigger than B
3. $A == B$: A equals B
4. $A \sim= B$: A is different from B
5. $A <= B$: A is smaller than or equal to B
6. $A >= B$: A is bigger than or equal to B

You can also combine conditions in an IF statement:

1. Cond1 && Cond2: Both conditions have to be true
2. Cond1 || Cond 2: At least one has to be true

Example:

```
if A<B && A<C
  display('A is the smallest from ABC')
end
```

Medium-sized exercise

Define two vectors as follows: `A = [1, 10, 2, -5, 50, 80]` and `B = [90, -5, 90, 2]`. Now, write a script that counts the number of elements that these two vectors have in common (the answer should be 2). You will need two nested For loops and an IF statement; the first For should run through the elements of A, the second through the elements of B.

Use the `numel` Matlab function to make it a general script, i.e. a script that works for any two vectors A and B.

Translation from text to Morse

For this part of the exercise, check the section called The Translation in the Morse.pdf file.

Operations with vectors

For a scalar `A = 5` and two vectors `Vec1 = 1:10;` and `Vec2 = 11:20;`, we can do the following:
Sum:

- `Vec1 + Vec2 <Enter>` ← They are added element-wise and the result is a vector of the same size as Vec1 and Vec2.
- `A + Vec1 <Enter>` ← A is added to each element of Vec1
- `A - Vec2 <Enter>`

Operations with vectors

For a scalar `A = 5` and two vectors `Vec1 = 1:10;` and `Vec2 = 11:20;`, we can do the following:
Sum:

- `Vec1 + Vec2 <Enter>` ← They are added element-wise and the result is a vector of the same size as Vec1 and Vec2.
- `A + Vec1 <Enter>` ← A is added to each element of Vec1
- `A - Vec2 <Enter>`

Multiplication:

- `A*Vec1 <Enter>` ← Each element of Vec1 is multiplied by A.
- `Vec1.*Vec2` is called the element-wise multiplication. Works as the sum.
- `Vec1./Vec2` is the element-wise division.

Operations with vectors

For a scalar `A = 5` and two vectors `Vec1 = 1:10;` and `Vec2 = 11:20;`, we can do the following:
Sum:

- `Vec1 + Vec2 <Enter>` ← They are added element-wise and the result is a vector of the same size as Vec1 and Vec2.
- `A + Vec1 <Enter>` ← A is added to each element of Vec1
- `A - Vec2 <Enter>`

Multiplication:

- `A*Vec1 <Enter>` ← Each element of Vec1 is multiplied by A.
- `Vec1.*Vec2` is called the element-wise multiplication. Works as the sum.
- `Vec1./Vec2` is the element-wise division.

Mini exercise:

1. Create a vector Vec3 such that `Vec1-Vec3` is a vector of ten ones.
2. Create a variable B such that `Vec1+B-Vec3` is a vector of ten zeros.
3. Create a vector with 100 elements, whose value is all 1 using element-wise division.

The beep

For this part of the exercise, check the section called The Sound File in the Morse.pdf file.

Scripts

- Scripts are successions of commands. Executed in the order found (from top to bottom).
- % at the beginning of a line means that it's a comment and won't be executed.
- Use ; at the end of each command to suppress the output of that command.
- To run the script, use F5.
- Use %% to divide the script in independent cells.
- To run a cell, press ctrl+Enter.
- Script names can have letters, underscores and numbers. Just like variables.
- All will be saved to the workspace (command window). Variables will be overwritten.
- You can execute a script within another script by just writing its name.

Functions

A function is defined as:

```
function [out1, out2, ...] = functionName(in1, in2,...)
  Content of function
end
```

- It's also a succession of commands
- All variables are stored in a temporary workspace and deleted afterward.
- You can reuse names of variables that are in your main workspace without changing them.
- You cannot use variables from outside of the function unless passed as inputs.

To call a function:

```
[var1, var2,...] = functionName(in1, in2,...);
```

Last part of the Morse Code

For this part of the exercise, check the section called 'Turning it into a Function' in the Morse.pdf file.