Introduction to MATLAB: Program Flow

Eduardo Rossi
University of Pavia
erossi@eco.unipv.it

September 2014
Two crucial aspects of programming

- Conditional programming:
  1. Logical operators
  2. `if`, `elseif`, `else`.

- Recursive programming
  1. `for` loops
  2. `while` loops
Read your program. There is only one route the program will take. You can follow it as well.

Statements are executed in order;

A statement can be a *loop or conditional* statement, repeating or skipping some statements.

The order can also be broken by *break* or *continue*

A statement can call a *function*: The statements within the function are executed in order, until the end of the function

And that is all, any program follows these lines.
Logical Operators

Logical operators are fundamental when writing a code as they allow for:

- conditional branching in flow of control
- changes in the path of the program due to a condition
MATLAB represents true and false with numerical symbols, in particular, false=0 and true=1 (Boolean conditions). The logical operators are

- `==`: is ... equal to ...?
- `~=`: is ... not equal to ...?
- `>`: is ... greater than ...?
- `<`: is ... less than ...?
- `>=`: is ... greater than or equal to ...?
- `<=`: is ... less than or equal to ...?

To combine more than one conditional statement, use

- `&&` (and)
- `||` (or)
Logical Operators

Other useful operators are `all`, `any`, `find`, and `is*`.

- `all(condition)`: is the `condition` satisfied for all the elements?
- `any(condition)`: is the `condition` satisfied for any of the elements?
- `find(condition)`: find the elements that satisfy the `condition`.
- `isnan(condition)`: is the element of the matrix `not-a-number`?
- `isinf(condition)`: is the element of the matrix `infinite`, `Inf`?
- `isfinite(condition)`: is the element of the matrix `finite`?
- `isreal(condition)`: is the input a `real` matrix?
- `ischar(condition)`: is the input a `character` array?
- `isempty(condition)`: is the input a `empty` matrix?
- `isequal(condition)`: are all the input elements equal?
- `islogical(condition)`: is the input a `logical` matrix?
- `isscalar(condition)`: is the input a `scalar`?
- `isvector(condition)`: is the input a `vector` (row or column)?
Flow of Control

The usefulness of logical operators turns out to be fundamental when the code involves conditional branching, that changes the normal flow of the program.

- modify the sequence of instructions,
- modify the order in which they are executed.
- different portion of the code to be executed depending on whether certain conditions are met.
Flow of Control

The `if` command followed by a logical condition begins the conditional branch. Each subsequent conditional statement must begin with `else if` or `else`. Example

```matlab
if logical_1
    Code to run if logical_1 true
else if logical_2
    Code to run if logical_2 true and logical_1 false
else if logical_3
    Code to run if logical_3 true and logical_j false, j < 3
    ...
    ...
else
    Code to run all logicals false
```

Note that all the logical expression must be scalar values.
Exercise

- Generate a random number from a standard Gaussian distribution.
- If the number is negative, then compute the square of it and print the output;
- Otherwise compute the cube and print the output.
Other useful commands are given by the switch/case statements. In particular, they are useful when we have to deal with complex statements and can be completely replicated using only if / elseif flow control blocks. The switch statement switches among several cases based on expression and executes one set of statements selected from an arbitrary number of alternatives. The syntax is

```matlab
switch switch expr
  case case expr
    statement,...,statement
  case case expr2
    statement,...,statement
  ... otherwise
  statement,...,statement
end
```
Example:

```matlab
method = 'bilinear';
switch lower(method);
    case {'linear','bilinear'};
        disp('Method is linear');
    case 'cubic';
        disp('Method is cubic');
    case 'nearest';
        disp('Method is nearest');
    otherwise;
        disp('Unknown method.');
end;
```
Exercise

Load the MATLAB file GE_2010_2014.mat.

1. Select the first and the fifth column of the data, corresponding to the daily open and closing prices.
2. Calculate the daily log returns based on daily opening and closing prices;
3. Write a code that takes a different path depending if the sample averages of the two returns series are equal or not. Allow for three cases: equality (no changes), average of opening > closing (compute the squares of opening returns), or opening < closing (compute the squares of closing returns).
4. Repeat the previous exercise using the switch, case statement.
Loops

In general, all the operations are executed row by row. Problem: excessively slow programming, when a series of operations has to be repeated sequentially. Solution: Loops!

- a very useful programming structure;
- allow to repeat a large number of operation in a sequence;
- combine with flow control blocks in very simple way;
- code become easy to read by the user;
In these cases, you should recur to the *loops*, such that your program code becomes easier to read and to compile by the user.

A *loop* opens a particular programming mode, where block of codes are executed in sequence. The general syntax of a loop is the following:

- **loop**: control expression to initialize the loop (with exit condition);
- **body of the loop**: the operation that are executed recursively;
for Loops

The for loop consists of three parts:
- an initialization part;
- a termination check;
- an increment part.

The for loop repeats the operations a pre-specified number of times. It can take any value in a range. Otherwise, it can be an existing vector of numbers with no orders.
Example:

```
S=0;
N=100;
for i=1:N;
S=S+i;
end;
SGauss=(N*(N+1))/2;
```

or, from an existing vector of numbers,

```
S=0;
vX=[7 8 10 2 1.5];
for i=vX;
S=S+i;
end;
```
Write a *for loop* that calculates the sum of the decimals between 0 and 2 \((0.1 + 0.2 + ... + 2)\).

Generate two vectors of random numbers from a Gaussian distribution.

Write a code block that would take a different path depending on whether the two vectors are simultaneously positive, both are negative, or they have different signs.
There are some cases when we want to repeat a section of MATLAB code until some logical condition is satisfied.

We cannot tell in advance how many times we have to go around the loop.

The loop command `while` repeats the block of code until a logical condition is violated.

The condition must be true when the loop starts, otherwise the loop will be ignored.

The condition must be updated inside the loop, otherwise the loop will repeat the operation an infinite number of times (Hint: use CTRL+C to stop a running program).
Example

The syntax of the loop is while CONDITION. Example:

S=0;
N=100;
i=0;
while i<=N;
    S=S+i;
i=i+1;
end;

Calculates the sum of the numbers between 0 and 100.
The command `while` can be used to construct complex flows of control, involving conditional branching. For example:

```matlab
K=20;
T=100;
Sigma=cov(randn(T,K));
diagonal=zeros(K,1);
for i=1:K;
    j=1;
    while j<=i;
        if j<i;
            j=j+1;
        else
            diagonal(i)=Sigma(i,j);
            j=j+1;
        end;
    end;
end;
```

The previous code extracts (in a very inefficient way!) the principal diagonal of a covariance matrix. I suggest to use the built-in functions of MATLAB, when it is possible, since loops increase drastically the execution time, while MATLAB functions are optimized.
Exercise

- Write a code to assess the fact that the loop is slower than the built-in functions to extract the diagonal element of a matrix.
- Generate a random number from a Gaussian distribution with mean 100 and variance 100.
- Start a while loop that at each iteration divides by two the result obtained at the previous iteration (the first division starts is based on the generated number).
- Set the stopping criterion when the result at the previous iteration is lower than 1;
- Print the number of iterations.
The command `break` in a loop determines the exit when a logical condition is verified. The general syntax is:

```plaintext
for (iterator=number;)
...
...
if (Condition)
  break;
end;
end;
```
he command `continue` forces the loop to skip to the next iteration when a condition is respected. The general syntax is

```matlab
for (iterator=number;)
    ...
    ...
    if (Condition)
        continue;
    end;
end;
```
K=20;
T=100;
Sigma=cov(randn(T,K));
diagonal=zeros(K,1);
for i=1:K;
    j=0;
    while j<i;
        j=j+1;
        if j<i;
            continue;
        end;
        diagonal(i)=Sigma(i,j);
    end;
end;