Numerical Methods in Engineering Sciences

Lecture 1: Brief introduction to MATLAB

Pablo Antolin

pablo.antolinsanchez@unipv.it

October 29th 2013

How many of you have used MATLAB before?

How many of you have used C/C++ before?

How many of you have used Python before?

Wikipedia says

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

Main features

- MATLAB = "MATrix LABoratory" originally based on LAPACK & BLAS
- It is an interpreted programming language (high-level scripting language): flow control, functions, input/output, classes (recent feature), ...
- Dynamically typed
- Mainly use for matrix analysis: solving linear systems, eigen-values & eigen-vectors, ...
- 2D & 3D data visualization
- Interfaces with programs written in C, C++, Fortran, Java

MATLAB help

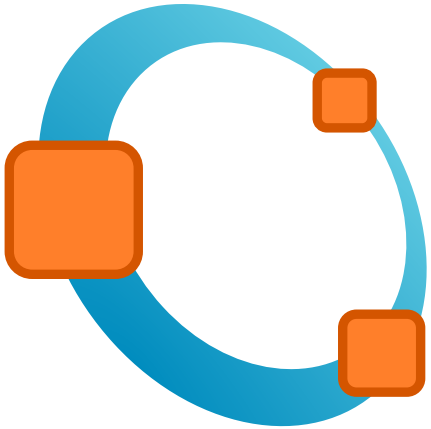
MATLAB is very well documented

`http://www.mathworks.com/help/matlab/`

Octave: a good alternative to MATLAB

Main features

- `http://www.gnu.org/software/octave/`
- GPL license. Free software.
- Multiplatform: Windows, OSX, GNU/Linux
- Syntax 99% compatible with MATLAB
- Most of the common MATLAB functionalities are mimicked (including PDE toolbox)



The commands are typed in the *prompt* of the *command window* (in particular, programs are launched there). The result of the commands is visualized in the *command window* also.

```
>>
```

Some examples:

```
>> 2 + 3
ans =
     5
>> sin(2)
ans =
    0.9093
>> sin(pi)
ans =
   1.2246e-16
>> [1 2; 3 4] + [10 20; 30 40]
ans =
    11    22
    33    44
>> % This is a comment
```

```
>> a = 2
a =
     2
>> b = 3
b =
     3
>> a + b
ans =
     5
>> whos % Which are the defined variables?
  Name        Size                    Bytes  Class
  a           1x1                         8  double
    array
  ans         1x1                         8  double
    array
  b           1x1                         8  double
    array
Grand total is 3 elements using 24 bytes
```

```
>> A = [1 2; 3 4]; B = [10 20; 30 40]; C = A + B
   ;
>> C
C =
    11    22
    33    44
>> whos
  Name        Size                    Bytes  Class
  A           2x2                        32  double
    array
  B           2x2                        32  double
    array
  C           2x2                        32  double
    array
  a           1x1                         8  double
    array
  ans         1x1                         8  double
    array
  b           1x1                         8  double
    array
Grand total is 15 elements using 120 bytes
```

```
>> row_vector = [1 2 3]
row_vector =
     1     2     3
>> col_vector = [1; 2; 3]
col_vector =
     1
     2
     3
>> row_vector + col_vector
??? Error using ==> +
Matrix dimensions must agree.
>> row_vector * col_vector
ans =
    14
```

Asking for help

```
>> help name_of_the_command
```

For example

```
>> help numel
 numel    Number of elements in an array or
    subscripted array expression.
     N = numel(A) returns the number of elements,
     N, in array A.

     N = numel(A, INDEX1, INDEX2, ...) returns in
     N the number of
     subscripted elements in array A(index1,
    index2, ...).

     MATLAB implicitly calls the numel builtin
    function ...
```

For extracting submatrices the command ":" is used

```
>> A = [2 4 6 ; 5 7 9; 2 3 4; 6 7 8]
A =
     2     4     6
     5     7     9
     2     3     4
     6     7     8
>> A(2, :)
ans =
     5     7     9
>> A(2:3, 1:2)
ans =
     5     7
     2     3
```

The "." operator

```
>> a = rand(1, 20) ;
>> b = rand(1, 20) ;
>> c = a .* b ;
>> size (c)
ans =
      1    20
```

This is equivalent to

```
for i = 1 : 20
  c(i) = a(i) * b(i) ;
end
```

Also ...

```
>> d = a .^ 2 ;
>> f = a ./ 2 ;
```

MATLAB as a programming language.
The files that contain the code are the M-files (`*.m`). There exists two types:

- Scripts: they have input/output variables. All the variables are created in the main memory (the same memory used by the *command window*)
- Functions: they receive and input and return an output. They create local variables and interact with the main memory only with the input/output variables. Sintax:

```
function [output1, output2] = namefunction (
    input1, input2)
  % Comment (it will appear in the help of
    the function)
  ......
  ......
  ......
  output1 = .....
  output2 = .....
  return
```

Let's consider the script test_script.m:

```
clear all
A = [1 2 3; 4 5 6; 7 8 9] ;
B = [0; 1; 2] ;
C = A * B
```

When executed, it produces:

```
>> test_script
C =
      8
     17
     26
>> who
Your variables are:
A   B   C
```

Let's consider the function test_function.m:

```
function add = test_function (add1, add2)
    add = add1 + add2 ;
    return
end
```

When executed, it produces:

```
>> test_function([1 2], [3 4])
ans =
     4     6
>> who
Your variables are:
A    B    C    ans
```

Flow control commands:

- if
- switch and case
- for
- while
- continue
- break

Example:

```
counter = 0 ;
for i = 1 : 10
    counter = counter + i ;
    if counter == 5
        break ;
    end
end
```

Example:

```
if I == J
  A(I,J) = 2;
elseif abs(I-J) == 1
  A(I,J) = -1;
else
  A(I,J) = 0;
end
```

Example: quadrature (by using mid point rule) in $[-1,1]^2$
We want to write a function called `mid_point` that computes an approximation of the integral of a generic f in the square $[-1,1]^2$ by means of the quadrature formula

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dxdy = \sum_{T\in\mathcal{T}_h} f(x_T,y_T)|T|$$

where $\mathcal{T}_h$ is a discretization of the domain, and (x_T, y_T) are the coordinates of the baricenter of every discrete element $T \in \mathcal{T}_h$.
Let's do it by using square elements. n elements per side.

```
function integral = mid_point(num_elements)
    % Computing center of squares
    size_elements = 2.0 / num_elements ;
    x0 = 1.0 - size_elements / 2.0 ;
    x = linspace ( -x0, x0, num_elements) ;
    coords = zeros (num_elements ^ 2, 2) ;
    k = 1 ;
    for i = 1 : num_elements
        for j = 1 : num_elements
            coords(k, 1) = x(j) ;
            coords(k, 2) = x(i) ;
            k = k + 1 ;
        end
    end
    % Computing integral
    area_element = size_elements ^ 2 ;
    values = myfunction(coords) * area_element ;
    integral = sum (values) ;
    return ; % This is optional
end
```

Let's try with the function $f(x, y) = 1 + x + y$

$$\int_{-1}^{1} \int_{-1}^{1} f(x, y)\, \mathrm{d}x\, \mathrm{d}y = 4$$

```
function value = myfunction(points)
    value = 1 + points(:, 1) + points(:, 2) ;
    return ;
end
```

```
n = [1, 2, 4, 8, 16, 32, 64, 128] ;
exact_solution = 4 ;
format long ;
err = zeros (size (n)) ;
for ii = 1 : numel(n)
    err(ii) = ...
        abs(exact_solution - mid_point(n(ii))) ;
end
figure ;
loglog (n .* n, err)
xlabel ('number of points') ;
ylabel ('error') ;
axis equal ;
```

```
function integral = mid_point(func, num_elements)
    % Computing center of squares
    size_elements = 2.0 / num_elements ;
    x0 = 1.0 - size_elements / 2.0 ;
    x = linspace ( -x0, x0, num_elements) ;
    coords = zeros (num_elements ^ 2, 2) ;
    k = 1 ;
    for i = 1 : num_elements
        for j = 1 : num_elements
            coords(k, 1) = x(j) ;
            coords(k, 2) = x(i) ;
            k = k + 1 ;
        end
    end
    % Computing integral
    area_element = size_elements ^ 2 ;
    values = func(coords) * area_element ;
    integral = sum (values) ;
    return ; % This is optional
end
```

Let's try now with the function $f(x, y) = 1/2 + \sin 10x \cos 15y$

$$\int_{-1}^{1} \int_{-1}^{1} f(x, y)\, \mathrm{d}x\, \mathrm{d}y = 2$$

```
function value = myfunction(points)
    value = 0.5 + sin(10.0 * points(:, 1)) .*
   cos(15.0 * points(:, 2)) ;
    return ;
end
```

```
n = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512] ;
exact_solution = 2 ;
format long ;
err = zeros (size (n)) ;
for ii = 1 : numel(n)
    err(ii) = ...
        abs(exact_solution - mid_point(
   @myfunction, n(ii))) ;
end
figure ;
loglog (n .* n, err)
xlabel ('number of points') ;
ylabel ('error') ;
axis equal ;
```

Now, try you!

$$f(x,\, y) = 1/2 + \sin xy \cos xy$$

in the rectangle $[0, 1] \times [-1, 1]$

$$\int_0^1 \int_0^1 f(x,\, y)\, \mathrm{d}x\, \mathrm{d}y = 1$$

Solution

```
function integral = mid_point(func,num_elements)
    size_elements_x = 1.0 / num_elements ;
    size_elements_y = 2.0 / num_elements ;
    x0 = size_elements_x / 2.0 ;
    x1 = 1.0 - size_elements_x / 2.0 ;
    x = linspace ( x0, x1, num_elements) ;
    y1 = 1 - size_elements_y / 2.0 ;
    y = linspace (-y1, y1, num_elements) ;
    coords = zeros (num_elements, 2) ;  k = 1 ;
    for i = 1 : num_elements
        for j = 1 : num_elements
            coords(k, :) = [x(j), y(i)] ;
            k = k + 1 ;
        end
    end
    area_element = size_elements_x * size_elements_y ;
    values = func (coords) * area_element ;
    integral = sum (values) ;
end
```

Solution

The function for f

$$f(x, y) = 1/2 + \sin xy \cos xy$$

is

```
function value = myfunction3(points)
    value = 0.5 ...
          + sin(points(:, 1) .* points(:, 2))...
         .* cos(points(:, 1) .* points(:, 2)) ;
    return ;
end
```

Solution

Just one element is needed for integrating f exactly

```
exact_solution = 1 ;
format long ;
err = abs(exact_solution...
              - mid_point(@myfunction3, 1))
```

Try a different function!