Short introduction to Matlab

Attilio Frangi

13 janvier 2006

1 Matlab commands

This document serves as an introductory and simple guide to common Matlab commands employed in the FEM codes analysed in the sequel. For further reference, please consult the complete Matlab documentation.

1.1 The command window

The command line in the command window allows users to send commands to Matlab. The command `dir`, for instance, tells Matlab to list the contents of the current directory :

```
>> dir

.                   elast2d_tri3        example.m           matlab.pdf
..                  elast3d_tet4        matlab.aux          matlab.tex
coupled2d_tri3      elastaxi_tri3       matlab.log          matlab_commands.tex
```

We can change the current directory by means of the usual `DOS` commands. Suppose we are currently in a directory `matlab`. First we move to parent directory, then list the contents and finally return back into `matlab`.

```
cd ..
>> dir

.         SISM.pdf  pc1       pc3       pc5       pc7       pc9
..        matlab    pc2       pc4       pc6       pc8       utility

>> cd matlab
```

1.2 Basic operations

We define variables `a=1` and `b=2`. The semicolons prevent Matlab echo. Next we perform algebraic operations on `a` and `b` and save the result in `c` :

```
>> a=1;
>> b=2;
>> c=a+b+a*b-2

c =

     3
```

We now create the row vector `a`. Remark that we can use the same name previously employed for a scalar variable. Next we define vector `b` as the transpose of `a`.

```
>> a=[1 2 3];
>> size(a)

ans =

     1     3

>> b=a'

b =

     1
     2
     3

>> size(b)

ans =

     3     1
```

Finally we compute the scalar product `d` between `a` and `b` :

```
>> d=a*b

d =

    14
```

On the contrary, the product of a column vector and a row vector yields the tensor (outer) product. For instance :

```
>> mat=b*a

ans =

     1     2     3
     2     4     6
     3     6     9
```

As expected, `mat` has a null determinant :

```
>> det(mat)

ans =

     0
```

We now add the identity matrix to `mat` and evaluate the determinant again :

```
>> mat=mat+eye(3)

mat =

     2     2     3
     2     5     6
     3     6    10

>> det(mat)

ans =

    15
```

The function `eye` is an example of Matlab built-in function. A very useful feature is the `in-line help`, which can be invoked typing on the command line `help` followed by the name of the command we wish to investigate. Frequent use of this in-line help, when reading the FEM codes, is strongly suggested.

```
>> help eye

 EYE Identity matrix.
    EYE(N) is the N-by-N identity matrix.

    EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1's on
    the diagonal and zeros elsewhere.

    EYE(SIZE(A)) is the same size as A.

    See also ONES, ZEROS, RAND, RANDN.
```

Suppose now we want to solve the system `mat*x=b`, where `x` is the unknown solution vector. Then we simply type :

```
>> x=mat\b

x =

    0.0667
    0.1333
    0.2000

>> mat*x

ans =

     1
     2
     3
```

A very powerful feature of Matlab is the ability to operate on several elements of matrices and or vectors at the same time. In this example a row vector full of zeros is created (`help zeros`) and the first three elements are initialized. Then the second and fifth are given specific values.

```
>> a=zeros(1,5);
>> a(1:3)=[1 3 4]

a =

     1     3     4     0     0

>> b=[2 5];
>> a(b)=[-2 3]

a =

     1    -2     4     0     3
```

This feature can be also employed for matrices, as the following example clarifies :

```
>> a=zeros(4,4);
>> b=[1 2; 3 4];
>> ir=[2 4];
>> ic=[2 3];
>> a(ir,ic)=b

a =

     0     0     0     0
     0     1     2     0
     0     0     0     0
     0     3     4     0
```

This possibility will be often exploited especially when assembling global rigidity matrices.
It is often necessary to enter a complete matrix from an m file (see further on) or from the command line. Lines of the matrix are delimited by the semicolon symbol as follows

```
>> a=[1 3 4; 2 5 7; 10 11 13]

a =

     1     3     4
     2     5     7
    10    11    13
```

Is is worth stressing that matrices are stored in memory as vectors. Suppose we have an (M,N) matrix `a`. Then Matlab implicitly creates a vector the first M coefficients of which are the coefficients of the first column of `a`, followed by the M coefficients of the second column and so on. We say that Matlab uses a column-wise vector storage of matrices
For instance, using the matrix `a` just defined, if we ask Matlab to write `a(4)`

```
>> a(4)

ans =

     3
```

we obtain the fourth coefficient in the equivalent matrix vector

1.3 Matrix operations versus coefficient operations

Some operators, like the product * operator, the exponential operator ^ and others, can have different effects if applied to matrices as a whole or to single coefficients.
Let us first define two matrices :

```
>> a=[1 3 4; 2 5 7; 1 1 0];
>> b=[4 1 0; 0 2 5; 5 4 3];
```

Then as expected :

```
>> a*b

ans =

    24    23    27
    43    40    46
     4     3     5
```

is the classical row-vector product of matrices. On the contrary, if we add a point before the product operator

```
>> a.*b

ans =

     4     3     0
     0    10    35
     5     4     0
```

Matlab yields a matrix whose coefficients are the product of the corresponding coefficients in `a` and `b`. Essentially, whenever defined, the dot operator placed before a second operator tells Matlab to perform operation coefficient-wise and not on matrices as a whole

1.4 M files

Pieces of code can be stored in files with extension `.m`. These files can be executed by simply typing the name of the file itself in the command line. Suppose we have created the file `example.m` containing the following lines which define the elastic coefficient matrix for 2D plane stress (typed exactly as we were working with the command line) :

```
E=100; nu=.3;
A=E/(1-nu^2)*[1,nu,0;
              nu,1,0;
              0,0,(1-nu)/2];
```

The file can be conveniently created using the Matlab editor which can be invoked from the command line by typing `edit`. Then from the command line first we execute `example`, then we display matrix `A` contents :

```
>> example
>> A

A =

  109.8901   32.9670         0
   32.9670  109.8901         0
         0         0   38.4615
```

1.5 Flow control

Test construct `if`. Suppose we create the following if_file.m file :

```
if a>0,
 b=0;
else
 b=1;
end
```

Then, from the command line :

```
>> a=1;
>> if_file
>> b

b =

     0

>> a=-2;
>> if_file
>> b

b =

     1
```

A second example is the definition of the material stiffness matrix `A` which will be employed in the FEM codes. Variable `type` can have the values 1 (for plane stress) or 0 (for plane strain). `E` and `nu` are material parameters

```
if type==1,  % plane stress
 A=E/(1-nu^2)*[1,nu,0;
               nu,1,0;
               0,0,(1-nu)/2];
else  %plane strain
 A=E/((1+nu)*(1-2*nu))*[1-nu,nu,0;
                        nu,1-nu,0;
                        0,0,(1-2*nu)/2];
end
```

Loop `for`. Suppose we have created the for_file.m file :

```
a=zeros(1,10);
for i=1:10,
 a(i)=2*i-1;
end
```

Vector `a` is created as a row of ten zeros (use `help zeros` to learn more about `zeros`). Then the loop over `i` is used to initialize it :

```
>> for_file
>> a

a =

     1     3     5     7     9    11    13    15    17    19
```

This is not the most efficient way to perform this initialization. A much better way would be :

```
>> a=[1:2:20];
```

This command should be interpreted as follows : create a vector whose entries are obtained starting from 1 and adding 2 until the number is not greater than 20.

`Switch` construct. According to the value of `Action` different outputs are produced

```
switch Action
case 'Initialize'
 sprintf('%s','You have chosen to Initialize')
case 'Refresh'
 sprintf('%s','You have chosen to Refresh')
otherwise
 sprintf('%s','No valid choice has been made')
end
```

For instance :

```
>> Action='Refresh';
>> switch_file

ans =

You have chosen to Refresh
```

1.6 Functions

It is often convenient to collect a series of commands into a function with inputs and outputs. The function must be saved in an `.m` file having the same name of the function itself. In the sequel the file `ex_func.m` is shown. Function `ex_func` accepts as input `a` and computes the output `b` according to some rule :

```
function b=ex_func(a)

if a>0,
 b=sqrt(a/3);
else
 b=sqrt(-a);
end
```

Hence, from the command line :

```
>> b=ex_func(27)

b =

     3

>> b=ex_func(-4)

b =

     2
```

1.7 File I/O

It is often required to read and write data on files. Imagine we have the following lines saved in dam.msh. This file is actually one of the input files which will be used in the FEM codes.

```
$NOD
13
1 0 0 0
2 5 0 0
3 0 10 0
4 2.499999999999938 0 0
5 4.000000000000028 1.999999999999943 0
6 3.000000000000051 3.999999999999898 0
7 2.000000000000072 5.999999999999856 0
8 1.000000000000069 7.999999999999861 0
9 0 7.500000000000055 0
10 0 5.000000000000111 0
11 0 2.500000000000167 0
12 1.87818608546323 2.00364788202003 0
13 1.313048020223624 3.825819278107899 0
$ENDNOD
```

Then we create an M file to read data. Before we can do this we need to introduce structures. A structure is simply an object with many fields. For instance we can define the structure `analysis` with the field `nnod` (the number of nodes) and the field `nel` (the number of elements) :

```
>> analysis.nnod=2

analysis =

    nnod: 2

>> analysis.nel=10

analysis =

    nnod: 2
    nel: 10
```

It is useful when we want to collect several pieces of information under a single name, like `analysis` in the example.
Now let us read the file. First we have to open it, in order to make it accessible :

```
fmid=fopen(dam.msh,'r');
```

Then read the string `$NOD`, and the following one (which contains the number of nodes). The number of nodes is directly read from the string

```
tline = fgets(fmid);
tline = fgets(fmid);
analysis.nnod=sscanf(tline,'%d');
```

The code `'%d'` tells Matlab that we want to read an integer. Then, line by line, all the nodal coordinates are read

```
coor=zeros(analysis.nnod,2);
for i=1:analysis.nnod,
 tline = fgets(fmid);
 h=sscanf(tline,'%d %f %f %f')';
 coor(i,:)=h(2:3);
end
fclose(fid)
```

Please remark the different codes `%d, %f` which are used for reading intergers and floats, respectively.
Remark also the transposition sign in the last `sscanf` which is required since `sscanf` creates by default a column vector.

1.8 Debugging

A very powerful feature is the possibility to debug applications. In Figure 2 the debugging buttons are evidenced in the tool-bar.
The first one on the left, in particular, can be used to set and remove break-points. In this case a break point as been set as shown by the arrow. The break point is graphically represented by a red dot.
If now we launch the m-file form the command line, the analysis will stop at the chosen line and a green arrow will appear on the left. If we now point with the cursor any of the variables defined above the breakpoint, Matlab will automatically show the value. The same info can be obtained from the command line :

```
K>> nnod

nnod =

     5
```

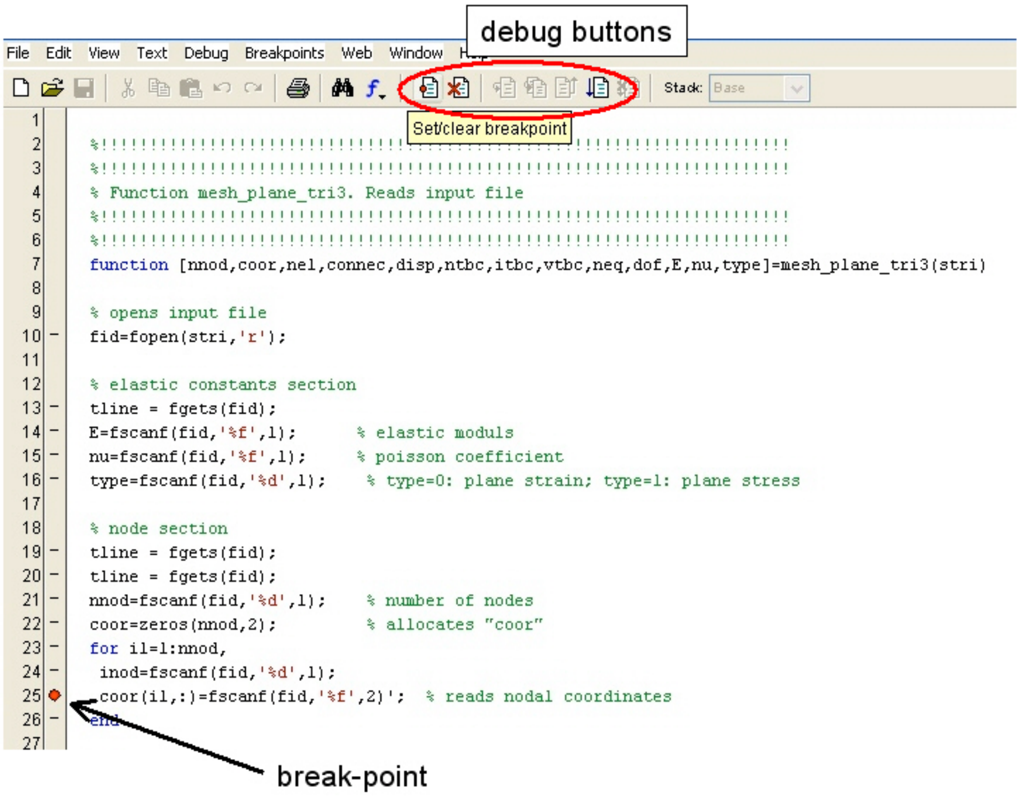

FIG. 1 – Debugging commands

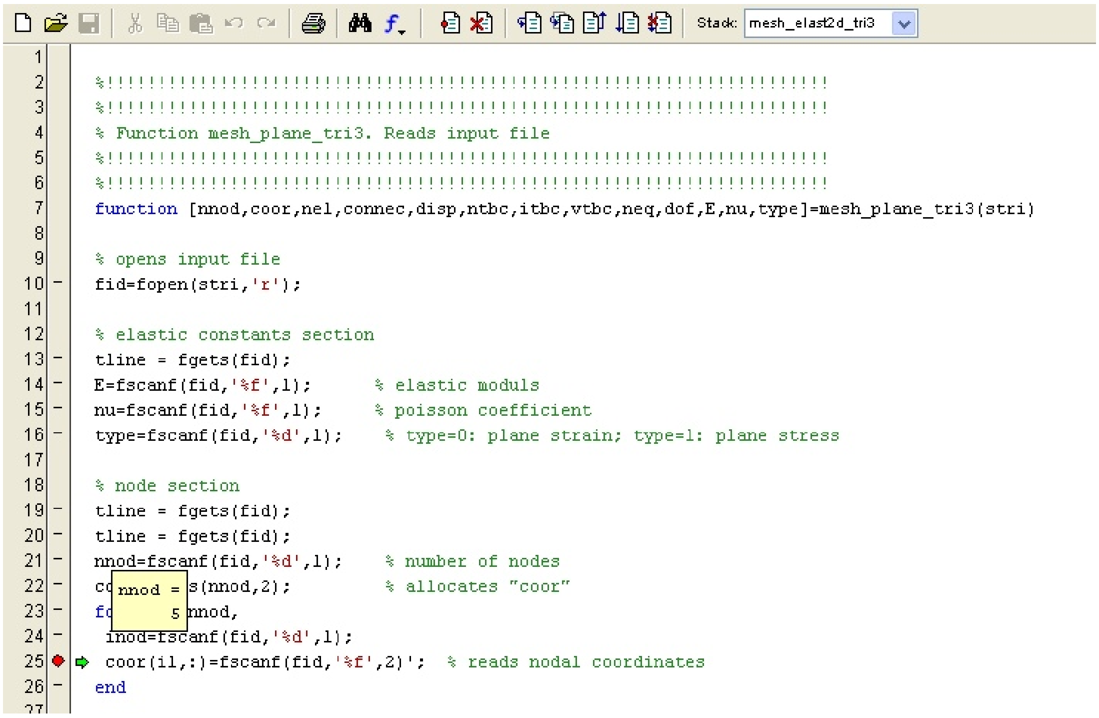

FIG. 2 – Analysis stopped at a break point