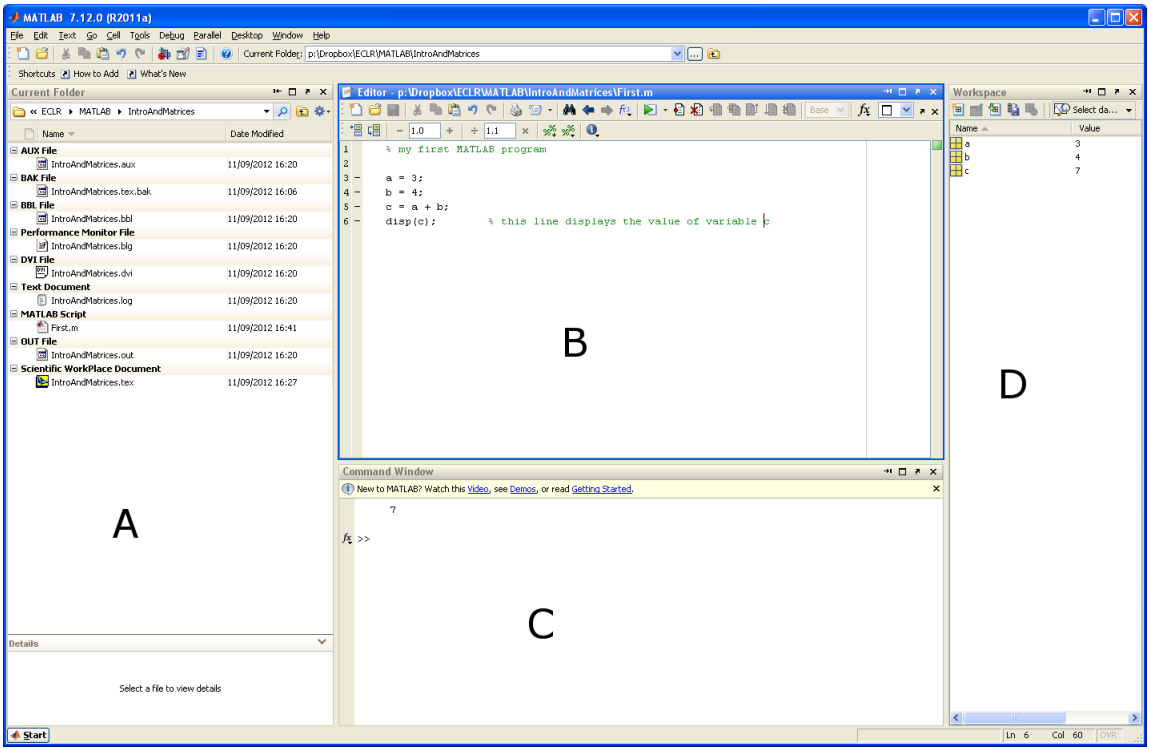
An Introduction to MATLAB and the use of matrices

Ralf Becker

1 Programming in MATLAB

We now consider how to create, edit and execute m–files. But initially we will have a brief look at the MATLAB screen.



This is how MATLAB looks on my machine although it may be arranged slightly differently on yours. However the 4 main elements A to D will always be there and I will briefly explain them

- Window A: This is the current directory.
- Window B: This is the editor window in which you can change MATLAB scripts and functions.

- Window C: This is called the command window. You can do basic clalculations in here (Type 3+2 and press ENTER) and any outputs of your MATLAB code will be printed in here.
- Window D: In this window you will be able to look at individual variables that have been defined in your code.

2 Creating and executing script m-files

An m–file can be created using the MATLAB Editor. To do this, click on `File` at the top of the MATLAB window and then select `New` followed by m–file. You will then be in the editor. Enter the commands:

```
1 C = [5 4; 3 2]; % Create 2x2 matrix
2 dc = det(C);    % Calculate determinant
3 disp(dc);       % Display value of dc
```

Everything following a % are comments and are ignired by MATLAB. It is good practice to use a lot of comments.

Now save the program into a file called test1.m. For now you created a collection of commands (three to be precise). What you now want is to tell MATLAB that you want it to execute these commands (of course you will typically have many more than three commands).

To execute this program type test1 on the command line in the MATLAB Command window and then hitting enter[1]. You can then seen that test1.m causes det(C) to be printed into the command window. At this stage you will also see two objects in the workspace (Window D), namely `C` and `dc`. You can use them in new calculations in the command window, say type `2*dc` and the ENTER.

If you want to edit an already existing m file you merely need to open it. It will appear in the editor window and away you go.

3 Some basic MATLAB commands

To work with MATLAB you need to know a few basic things. Data are saved and handled in matrices (vectors and scalars are just special cases of matrices). Let's say we have a matrix A. In

[1]For this to work, the current folder (see the field just underneath the menus needs to be set to the folder in which you saved this m file

order to access particular elements of that matrix you need to follow the following conventions:

- `C = zeros(4,3)`, this creates a new matrix C with 4 rows and 3 columns and all values 0.
- `D = ones(7,1)`, this creates a new matrix (vector) D with 7 rows and 1 column and all values 1.
- `A = [1 2; 3 4]`; this enters the following matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  Elements in one row are seperated by a space (although you could also use a `,` and a `;` indicates that the next element is the first element in a new row.
- `b = A(2,3)`, this saves the element from the 2nd row and 3rd columns of A in the new variable b.
- `b = A(2,:)`, this saves the entire 2nd row in variable b
- `B = A(4:end,:)`, this saves rows 4, 5, 6, up to the last (end) row of A in variable B (assuming A has at least 4 rows).
- `b = A(:,5)`, this saves the entire 5th column in variable b
- `B = A(:,3:end-1)`, this saves the following columns of A in a new matrix B: columns 3, 4, 5, etc. up to and including the last but one column.
- `A(6,2) = 4`, this puts the value 4 on the (6,2) element of A
- `A(:,1) = ones(size(A,1),1)`, this replaces the entire first column of matrix A with a vector of 1s. Note that to ensure that the vector of ones that is to be put into the first column of A is created to have exactly the same number of rows as A.
- `E = zeros(4,3,2)`, this creates a three dimensional array!
- `c = 1:5`, creates a (15) row vector with the numbers 1 to 5
- `c = 2:3:14`, creates a row vectors with numbers starting at 2 up to 14, but in steps of 3

If any of the above commands is followed by a semicolon(`;`) then the result of the operation will be saved in MATLAB's workspace but not printed onto the screen. This should be your default mode. If you want to see something on the screen you can either leave away the semicolon, or better use the command `disp(VARIABLENAME);`.

Mathematical operations are pretty straightforward in MATLAB. You just have to use the conventional mathematical commands:

- `C+D`, if C and D are matrices their dimensions will have to be identical. If one of them (say C) is a scalar that value will be added to each element of the other (say D).
- `C*D`, if C and D are matrices their dimensions will have to be such that a matrix multiplication is allowed
- `C.*D`, multiplies each element in C with the corresponding element in D. The matrices have to have identical dimensions for this operation to work. The `.` in front of an operation indicates an element-by-element operation.
- `a = b^c`, calculates b^c.
- `E = C'`, saves the transpose of C in the new variable E.
- `E = C'*C`, calculates C'C and saves it in the new variable E.
- `A = [B C]`, creates a matrix A, which consists of the two matrices B and C attached to each other sideways. B and C need to have the same number of rows. (horizontal concatenation)
- `A = [B; C]`, creates a matrix A, which stacks the two matrices B and C on top of each other. B and C need to have the same number of columns. (vertical concatenation)

4 Some basic MATLAB functions for matrices

In a later module you can read about functions in MATLAB. But in short they are little pieces of code that perform some pre-specified operations. In the context of matrices you may have some painful memories of having to calculate the inverse of a matrix by hand. Wouldn't it be nice if MATLAB could do this with one little command. Well you are in luck, MATLAB can. The inverse of the matrix A can be calculated by using `inv(A)`.

Well, this is only a half truth. MATLAB will still have to do all the painstaking operations required for the calculation of a matrix inverse, but it will do this all by itself and in the background as one clever programmer wrote a little piece of software that you call up, by typing `inv(A)`. Such a piece of software is called MATLAB. In a later module you can learn how to write such functions yourself. In fact it is very likely that you will wan to do this to make your life easier.

Here is a list of useful matrix functions in MATLAB:

- `[nr,nc] = size(A)`, the size command returns information on how many rows (nr) and how many columns (nc) the matrix A has (You choose the names for the output variables,

here nr and nc, yourself). You can achieve the same by the following two seperate commands

```
nr = size(A,1)
nc = size(A,2)
```

- `r = rank(A)`, this delivers the rank of a matrix
- `d = det(A)`, The inverse of A
- `C = inv(A)`,The trace of A

5 Error messages and coding practice

Error messages is MATLAB's way of communicating with you (in educational speak giving you formative feedback on your effort to writing a program). They usually contain two pieces of information.

1. Where in the code the mistake occurred
2. What went wrong, which gives you a hint on what to do to fix it. This part is at times cryptic at best.

You should be careful and not think that just because MATLAB did not give you an error message everything is alright. All the absence of MATLAB error messages means is that MATLAB could indeed undertake all instructions it obtained from you. It DOES NOT imply that all the instructions which you passed on to MATLAB are indeed the instructions you had intended. Also, sometimes you get a whole list of error messages. However, only tackle the error that occurs first in the code. All other error messages may disappear after having fixed the first problem.

This implies the following rules for good program writing:

1. Use extensive comments in your code in order to write down what you intended to do at a particular place in the code. Everything that follows a % in the code is ignored by MATLAB, and hence is a comment intended to yourself or someone else
2. Write the code bit by bit and check its workings as you write it in small steps
3. Where possible check whether the results you obtain are sensible