Introduction to MATLAB
Reading

- *Spencer and Ware* (2008), secs. 1-7, 9-9.3, 12-12.4.

- For reference: matlab online help desk

1   Introduction

MATLAB is commercial software that provides a computing environment that allows for sophisticated ways of developing and debugging computer code, executing programs, and visualizing the output. MATLAB is also a computer language (sort of a mix between C and Fortran) and this exercise for you to work through is mainly concerned with some of the language aspects that we will use extensively throughout the book. Please read through the more comprehensive and verbose MATLAB Intro and familiarize yourself with MATLAB.

We will assume that your Windows, Mac, or Linux machine has MATLAB installed. After starting up the program with the graphical user interface enabled, you will be presented with a number of windows, including an interactive window ("shell") where you can type in commands as we indicate below. Please also familiarize yourself with the other components of the development environment, such as the built-in editor for MATLAB programs, which are called "m-files", so that you can be more efficient in writing and debugging codes. There are numerous MATLAB -provided help resources accessible through the environment, including video tutorials, access to the help pages, along with extensive documentation on the web.

Also note that there is a free clone of MATLAB called octave. Given that MATLAB often uses freely available computational routines underneath the hood, it was fairly easy to reproduce the computational basics of MATLAB.

However, the MATLAB people also added a bunch of proprietary visualization tools which are not available in octave. Another alternative is to use the freely available Python language and its MatPlotLib package, but we will not have time to explore such intriguing options.

MATLAB is entirely vector or linear algebra based. It is therefore useful to briefly review some basic linear algebra.

2   Useful linear algebra (reprise)

Let's define a vector $\underline{b}$ as:

$$\underline{b} = \left(\begin{array}{ccc} 5 & 10 & 17 \end{array} \right)$$

---

and a 3 by 2 matrix $\underline{\underline{D}}$ as:

$$\underline{\underline{D}} = \begin{pmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \end{pmatrix}$$

The transpose (denoted with ^{T}) is given by:

$$\underline{\underline{D}}^T = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix}$$

$$\underline{b}^T = \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix}$$

Matrix-vector multiplication:

$$\underline{\underline{D}}^T \underline{b}^T = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix} = \begin{pmatrix} 130 \\ 142 \end{pmatrix}$$

Vector-vector multiplication (dot product):

$$\underline{b}\underline{b}^T = \begin{pmatrix} 5 & 10 & 17 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix} = \begin{pmatrix} 414 \end{pmatrix}$$

Matrix-matrix multiplication:

$$\underline{\underline{D}}^T \underline{\underline{D}} = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 42 & 44 \\ 44 & 49 \end{pmatrix}$$

If you don't know what's going on here, and what the rules for such multiplications are, please consult sec. **??**.

In numerical modeling, or in geophysical inverse problems, we frequently end up with linear system of equations of the form:

$$\texttt{Ac} = \texttt{Rhs}$$

where $\texttt{A}$ is a $n \times m$ matrix and $\texttt{Rhs}$ is a $n \times 1$ vector whose coefficients are both known, and $\texttt{c}$ is a $m \times 1$ vector with unknown coefficients. If we take $\texttt{A} = \underline{\underline{D}}$ and $\texttt{Rhs} = \underline{b}^T$, $\texttt{c}$ is (check!):

$$\texttt{c} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

3 Exploring MATLAB

3.1 Getting started

To start the program on the Linux machines type `matlab` at the UNIX prompt, or click on the relevant Windows item. The MATLAB environment, including the command window, starts. (If you want to avoid bringing up the whole environment on Linux, use "matlab -nojvm" for no-java-virtual-machine.)

1. Type `2+3`. You'll get the answer. Type `2 + 3*9 + 5^2`.

2. Type the following commands and note how MATLAB deals with vectors

```
>>x=3
>>x=3;
>>x
>>y=x^2
>>x = [2,  5.6]
>>y=2 * x;
>>y=x^2;
>>y=x.^2
>>y = [3,  4]
>>x * y
>>x * y'
>>x .* y
>>pi
>>a=x*pi
```

3. Type `demo` and explore some examples. Also note the introductory tutorial videos you might want to watch later.

4. Type `help`. You see a list of all help functions. Type `help log10` to get information about the log10 command. Type `help logTAB` where `logTAB` means typing log and then pressing the TAB key without adding a white space. Notice the command completion selection within the MATLAB shell. Note also that you can use the Up and Down arrows to retrieve previous commands and navigate through your command history, and `pUP` will bring up the last command line that started with a `p`. MATLAB also offers a graphical user interface (GUI) to explore all of its features: click help in the menu bar, then product help. Moreover, the function browser offers you a graphical way to find the suitable function for what you are trying to accomplish. The function browser can be found under help, functions browser, or can be brought up using a keyboard shortcut: `Shift+F1`.

3.2 Vectors/arrays and plotting

5. Create an array of x-coordinates

```
>>dx=2
>>x=[0:dx:10]
```

6. Y-coordinates as a function of x

```
>>y=x.^2 + exp(x/2)
```

7. Plot it:

```
>>plot(x,y)
```

8. Exercise: make a plot of a parametric function. What is it?

```
>>t=0:.1:2*pi
>>x=sin(t); y=cos(t); plot(x,y,'o-')
>>xlabel('x')
>>ylabel('y')
>>axis image, title('fun with plotting')
```

Exercise: make an ellipse out of it with short radius 1 and long radius 2. Also change the color of the curve to red.

3.3 Matrices and 3D plotting

First create x and y arrays, for example: `x=[1:5];y=x;`
9. Play with matrix product of x and y. Typing

```
>>x.*y
```

performs an element by element product of the two vectors (note the dot)

```
>>x'
```

returns the transpose

```
>>x*y.'
```

the "dot" or scalar product of two matrices

```
>>x'*y
```

the matrix product - returns a matrix.
Some commands (try them):

```
>>ones(1,5), zeros(6,1)
>>length(x)
>>whos
```

10. Create 2D matrices.
A useful function is meshgrid, which creates 2D arrays:

```
>>[x2d,y2d] = meshgrid(0:.1:2*pi,1:.1:2*pi)
```

You can get the size of an array with:

```
>>size(x2d)
```

11. Plotting of the function `sin(x2d.*y2d)`.

```
>>z2d = sin(x2d.*y2d)
>>surf(x2d,y2d,z2d)
>>mesh(x2d,y2d,z2d)
>>contour(x2d,y2d,z2d), colorbar
>>contourf(x2d,y2d,z2d), colorbar
```

Some cool stuff (1)

```
>>[x2d,y2d,z2d] = peaks(30);
>>surf(x2d,y2d,z2d); shading interp
>>light; lighting phong
```

Some cool stuff (2): perform the example given at the end of

```
>>help coneplot;
```

Other useful commands:
`clf`: clear current active figure
`close all`: close all figure windows

3.4   MATLAB scripting

By now you must be tired from typing all those commands all the time. Luckily there is a MATLAB script language which basically allows you to type the commands in a text editor. MATLAB scripts are text files that end with the suffix ".m".

12. Use the built in editor (or another text editor *e.g.* Emacs) and create a file "mysurf.m".

13. Type the plotting commands from the last section in the text file. A good programming convention is to start the script with `clear`, which clears the memory of MATLAB. Another good programming practice is to put lots of comments inside a MATLAB script. A comment can be placed after %, *e.g.* `% this is my first MATLAB script`.

14. Start the script from within MATLAB by going to the directory where the text file is saved. Type `mysurf` from within MATLAB and you should see the plot pop up in a new figure window. Alternatively, within the MATLAB editor, you can press F5 to run. Also note that there are various debugging features in the editor that are very helpful, such as real-time syntax checking and addition of breakpoints.

3.5 Loops

Create an array `na=100; a=sin(5*[1:na]/na); plot(a)`.
15. Ask instructions on using "for":

>>`help for`

16. Compute the sum of an array:

>>`mysum=0; for i=1:length(a), mysum = mysum + a(i); end; mysum`

17. Compare the result with the MATLAB inbuilt function `sum`

>>`sum(a)`

18. Exercise. Create x-coordinate array: `dx=0.01; y=cos([0:dx:10])`. Compute the integral of y=cos(x) on the x-interval $0 < x < 10$. Use sum(y) and write a MATLAB -script. Compare it with sin(10), the analytical solution.

3.6 Cumulative sum

19. Create a number of sedimentary layers with variable thickness.

>>`thickness = rand(1,10); plot(thickness)`

20. Compute the depth of the interface between different layers.

>>`depth(1)=0; for i=2:length(thickness), depth(i) = depth(i-1)+thickness(i); end; plot(depth)`

21. Compare the results with the built in MATLAB function cumsum:

>>`bednumber=1:length(depth)`
>>`plot(bednumber,depth,bednumber,cumsum(thickness))`

22. What causes the discrepancy? Try to remove it, ask `help cumsum`

3.7 IF command

23. Ask `help if`. Find maxima of the above array `thickness`, and compare it with the in built function `max(thickness)`

3.8 FIND command

24. Ask `help find`. Find which bed has the maximum thickness:
`find(thickness==max(thickness))`. Is there a way to do this without invoking the find command? find out by typing `help max`

25. Find the number of beds with a maximum thickness less than 0.5.

3.9   Matrix operations

26. Exercise: Reproduce the linear algebra exercises in the beginning of this document. Hint: If you want to solve the system of linear equations `Ac=Rhs` for `c`, you can use the backslash operator: `c = A\Rhs`

3.10   Functions

MATLAB allows you to declare functions that return a value and use m-files to store those functions. If you save

```
function xs = mysqr(x)
xs = x.^2;
```

as a `mysqr.m` in your working directory, you can then use your function just like a regular MATLAB command.

```
y=[2,3,4]
mysqr(y);
```

3.11   Variables and structures

MATLAB stores all regular variables as arrays of size 1×1 which are by default of type "double". To write more efficient programs, you might at times consider declaring integers as actual integers.

   More importantly, MATLAB affords you with the possibility to collect variables that logically belong together into a "structure". This variable will hold as many sub-variable as you want which are each addressed with a ".". For example, if dealing with earthquakes, you might want to use a structure like

```
quake.lon = 100.1;quake.lat = 120.1;quake.depth = 15;
```

The benefit of this is that you can now, for example, pass "quake" to functions and the function will locally know that quake actually has the components lon, lat, and depth which can be addressed within the subroutine.

26. Exercise: Write and test function that has two inputs, `x` and a `polynomial`. The `polynomial` structure should have two entries, the order of the polynomial expansion n and a vector a with n entries that hold the coefficients such that the function returns

$$y = \sum_{i=1}^{n} a_i x^{n-1} \tag{1}$$

Bibliography

Spencer, R. L., and M. Ware (2008), *Introduction to Matlab*, Brigham Young University, available online, accessed 07/2008.