

# 1 Intro and MATLAB basics

## 1.1 Introduction

Many physical phenomena, such as tsunami waves, elastic earthquake waves, heat diffusion, groundwater flow, bending of lithospheric plates, erosion or also the motion of rocks on geological timescales can be described by so-called partial differential equations. The appropriate governing equations to describe these phenomena were typically derived centuries ago (you learn more about it in a class on continuum mechanics). Here we will not rederive them, but rather focus on solving them. In many cases it is not possible to solve the equations analytically (i.e. write down the solution on a piece of paper), or if it is possible the solutions are very lengthy. At the same time, however, it is relatively straightforward to solve them with the help of a computer. Many phenomena such as weather prediction, but also geological processes, are so nonlinear that they can only be solved with the help of a computer.

Although there are a variety of numerical techniques that one can use (finite element, finite volume, spectral methods), the main focus here is on the Finite Difference Method which is widely used and maybe the most intuitive one to learn.

The focus here is on the practical problem of going from an equation to a solution. At the beginning of each lecture, a short introduction will be given and the rest of the lecture consists in writing code and solving exercises. The computer language we will use is MATLAB, which has a number of neat features, such as plotting or solving of linear systems of equations.

If you have never used MATLAB before, you likely will have to install it on your ZDV account. For this, open the program 'ZDV Apps' and move MATLAB 2011 to your local account. After logout/login MATLAB should be available under 'programs'. In case you already have an older version installed in your laptop, you can use that one. We do not support MATLAB clones like Octave or SciLab.

## 1.2 Useful linear algebra

MATLAB is entirely vector or linear algebra based. It is therefore useful to remind you of some of the linear algebra that you learned a long time ago.

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

$$\mathbf{b} = ( 5 \quad 10 \quad 17 )$$

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

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

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

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

Matrix-vector multiplication:

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

Vector-vector multiplication:

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

Matrix-matrix multiplication:

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

In numerical modeling, we frequently end up with linear system of equations of the form:

$$\mathbf{A}\mathbf{c} = \mathbf{Rhs}$$

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

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

### 1.3 Exploring MATLAB

MATLAB is a vector based computer language, which is available for Windows, MAC, Unix and Linux. It comes with its own programming language, which is a bit slow but still extremely useful due to its simplicity.

#### 1.3.1 Getting started

To start the program on the ZDV windows machines, make sure that matlab is installed locally (using ZDV Apps) and start **matlab** by clicking the icons. The MATLAB command window starts.

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

```
>>x=3
>>y=x.^2
>>z=x*y
>>pi
>>a=x*pi
```

3. Type **demo** and explore some examples.
4. Type **help**. You see a list of all help functions. Type **help log10** to get information about the log10 command.

#### 1.3.2 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.

### 1.3.3 Matrixes 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`

```
>>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 matrixes

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

the matrix product - returns a matrix.

Some commands (try them):

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

10. Create 2D matrixes.

A useful function is `meshgrid`, which creates 2D arrays:

```
>>[x2d,y2d] = meshgrid(0:.1:5,1:.1:8)
```

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

### 1.3.4 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 textfiles that end with the suffix ".m".

12. Open a the MATLAB editor and create a file "mysurf.m", which you save in your home-directory.

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 textfile is saved (your home directory). Type **mysurf** from within MATLAB and you should see the plot.

### 1.3.5 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.

### 1.3.6 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**

### 1.3.7 IF command

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

### 1.3.8 FIND command

24. Ask **help find**. Find which bed has the maximum thickness: **find(thickness==max(thickness))**.

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

### 1.3.9 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 **A $\mathbf{c}$ =Rhs** for **c**, you can use the backslash operator: **c = A\Rhs**