Introduction to MATLAB

Matlab is a 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 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 course. Please read through the more comprehensive and verbose Matlab Intro (available at the course website) and familiarize yourself with Matlab.

All the machines in this CIP Pool have Matlab installed (under Linux). After starting up the program, you will be presented with an interactive window where you can type in commands as indicated 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.

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

1. Useful linear algebra

Let's define a vector **b** as:

$$\mathbf{b} = \begin{pmatrix} 5 & 10 & 17 \end{pmatrix} \quad \rightarrow \quad \mathbf{b} = \begin{bmatrix} 5 & 10 & 17 \end{bmatrix}$$

and a 3 by 2 matrix **D** as:

$$\mathbf{D} = \begin{pmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \end{pmatrix} \quad \rightarrow \quad \mathbf{D} = \begin{bmatrix} 1\,2;\ 4\,3;\ 5\,6 \end{bmatrix}$$

The transpose (denoted with ^{T}) is given by: $\rightarrow$ type: D' and b'

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

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

$$\mathbf{A} \cdot x = rhs$$

where $\mathbf{A}$ is a $n \times m$ matrix and rhs is a $n \times 1$ vector whose coefficients are both known, and x is a $m \times 1$ vector with unknown coefficients.

If we take $\mathbf{A} = \mathbf{D}$ and $rhs = \mathbf{b}^T$, x becomes (check !):

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Exploring MATLAB

To start the program on the Linux machines type `matlab` at the UNIX prompt. The MATLAB environment, including the command window, is starting.

1. Type $2 + 3$ You'll get the answer.
   Type $2 + 3 * 9 + 5 \hat{}\; 2$

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
» 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 `p`⇑ will bring up the last command line that started with a `'p'`.

5. Create an array of x-coordinates

```
» dx=2
» x=[0:dx:10]
```

6. and of y-coordinates as a function of x

```
» y=x.^2 + exp(x/2)
```

7. and plot it:

```
» plot(x,y)
```

8. 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')
```

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

Now, create x and y arrays, for example `x=[1:5]; y=x;`

9. Play with the 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 the two vectors that returns a scalar

```
» x'*y
```

the dyadic product that 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.

MATLAB scripting & loops

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 command in a text editor of your choice. Matlab scripts are text files that end with the suffix ".m".

12. Use the built in editor (or another text editor e.g. vi) 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 anywhere 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. Alternatively, within the Matlab editor, you can press `F-5` 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.

15. Create an array `na=100; a=sin(5*[1:na]/na); plot(a);` 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 built-in function `sum`

    » `sum(a)`

18. Exercise: Create an 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.

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`

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

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 of less than 0.5.

26. Exercise: Reproduce the linear algebra exercises in the beginnig of this document. Hint: If you want to solve the linear equations $\mathbf{A}x = Rhs$ for x, you can use the backslash operator: $x = \mathbf{A}\backslash Rhs$

Functions & Structures

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 `mysqr.m` in your working directory, you can the use your function just like a regular Matlab command:

```
» y=[2,3,4]
» mysqr(y);
```

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 actually as 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-variables as you want which are each addressed with a ".". For example, if dealing with eartquakes, 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.

Online reference

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

`http://www.dynamicearth.de/compgeo/Tutorial/Day1/mlintro.pdf`]