Lab 1: Introduction to MATLAB

DSP Practical
Signal Processing Group
Technische Universität Darmstadt

October 28, 2014

1 Introduction

1.1 What is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. MATLAB itself is often used efficiently as a powerful graphical calculator.

This tutorial helps you to learn the MATLAB language, which lets you construct commands to create and process variables. Parts of its content are based on the "Getting Started" videos in [1]. The tutorial can be used interactively, i.e., while running MATLAB. You can try out commands, read in the online help or have a look at some MATLAB demos. You should be able to reproduce all of the plots and calculations in the tutorials by cutting and pasting text from the tutorials into the MATLAB command window or an m-file.

1.2 How to Get Help

To get help, generally feel encouraged to use the reference list at the end of this document. For example, there is an officially recommended tutorial section [1], which includes "Getting Started" videos and interactive MATLAB tutorials. A good list of available MATLAB tutorials, some in German, can

be found on [2]. Also, the classic primer by Kermit Sigmon [5], or a book on numerical computations by Cleve Moler [3] could be used as a second reference.

After starting MATLAB you should see the MATLAB prompt `>>` in the command window. MATLAB has a fairly good online help; type

```
>> help commandname
```

for more information on any given command. All you need to know is the name of the command. You can also find MATLAB command or function names via keyword search by typing

```
>> lookfor keyword
```

You can get the value of a particular variable at any time by typing its name. For example, the predefined constant π:

```
>> pi
ans =
    3.1416
```

You can have more than one statement on a single line by separating them with either a semicolon or comma. Also, you will see that as long as you do not assign a variable to a specific operation or result, MATLAB will store it in a temporary variable called `ans`.

2 A Short Outline of the Language

This section provides information about the basic syntax of MATLAB commands. We start off by introducing simple commands. You can create variables by entering them to the command window. For example,

```
>> a=1
a =
    1
>> b=3;
```

Note that MATLAB displays the value of the assigned variable. This can be avoided by adding a semicolon `;` after the command. You can carry out operations on your variables. For example,

```
>> c=a+b
c =
    4
>> d=cos(a)
d =
    0.5403
```

Important arithmetic operators in MATLAB are `+ - * / ^`, standing for addition, subtraction, multiplication, division and exponentiation. You can find basic MATLAB functions, such as `cos sin abs` etc. in the MATLAB function reference.

2.1 Vector Operations

MATLAB is an array-based language where variables can be vectors or matrices. You use square brackets `[]` to construct arrays. To create a row vector, you can type

```
>> t=[1 2 3 4 5]
t =
    1  2  3  4  5
```

Alternatively, you can use the colon operator `:` to simplify the creation of equally spaced arrays. The very same vector is obtained by typing

```
>> t=1:5
t =
    1  2  3  4  5
```

Other increments (spacing between vector elements) can be specified as follows

```
>> t=0:0.01:1;
```

which creates a vector with elements $0, 0.01, 0.02, \ldots, 1$. You can carry out operations on vectors just like simple scalars. For example,

```
>> x=sin(2*pi*t);
```

creates a sine wave. For visualization, you can plot `x` against `t` using the plot command

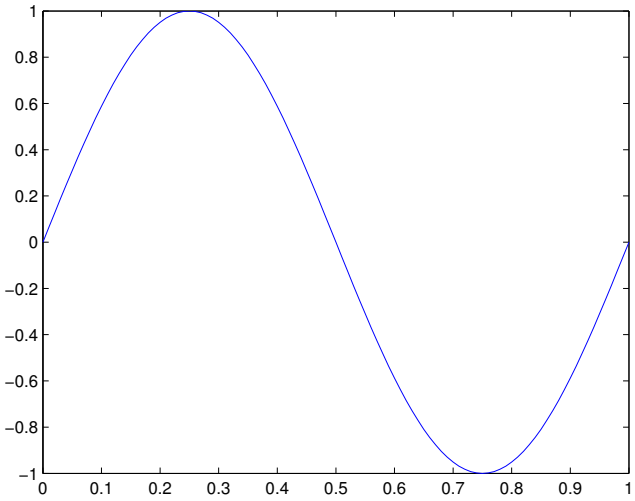

Figure 1: A simple plot of a sine wave.

```
>> plot(t,x)
```

The resulting plot is shown in Figure 1. If you want to see all currently existing variables in the MATLAB workspace, type

```
>> whos
  Name   Size   Bytes  Class

  a      1x1        8  double array
  b      1x1        8  double array
  c      1x1        8  double array
  d      1x1        8  double array
  t      1x101    808  double array
  x      1x101    808  double array
```

Generally, variables can also be complex with `i` or `j` denoting the imaginary part, such as

```
>> z=3+4i
z =
    3.0000 + 4.0000i
```

2.2 Matrix Operations

You can enter matrices by using the semicolon to separate rows. For example,

```
>> A = [1 2 3;4 5 6;7 8 10]
A =
    1   2   3
    4   5   6
    7   8  10
```

Alternatively, you can use functions to create matrices. For instance, a random matrix of 4 rows by 4 columns is obtained by

```
>> data=rand(4,4)
data =
    0.9501  0.8913  0.8214  0.9218
    0.2311  0.7621  0.4447  0.7382
    0.6068  0.4565  0.6154  0.1763
    0.4860  0.0185  0.7919  0.4057
```

You can find the dimension of an array using the `size` function

```
>> size(data)
ans =
    4  4
```

You can perform matrix operations, such as the matrix transpose

```
>> B=A'
B =
    1   4   7
    2   5   8
    3   6  10
```

or matrix multiplication (remember that the order matters when multiplying two matrices)

```
>> C=A*B
C =
    14   32   53
    32   77  128
    53  128  213
```

You can also perform element-wise operations by adding a dot before the operation symbol. As an example, element-wise multiplication is done using

```
>> C=A.*B
C =
     1    8   21
     8   25   48
    21   48  100
```

where the corresponding elements of `A` and `B` are multiplied. Note that the operant dimensions have to agree when using element-wise operations.

You can calculate the inverse of `A` by typing

```
>> inv(A)
ans =
    -0.6667  -1.3333   1.0000
    -0.6667   3.6667  -2.0000
     1.0000  -2.0000   1.0000
```

and multiply it by `A` to confirm you get the identity matrix:

```
>> inv(A)*A
ans =
     1.0000         0   0.0000
          0   1.0000         0
    -0.0000  -0.0000   1.0000
```

2.3 Selecting Array Elements

You can select elements or sections of an array by using indexing. For the variable `A`, the value of row 2 and column 3 is

```
>> A(2,3)
ans =
      6
```

For the variable `data`, rows 1 to 2, columns 2 to the end, can be selected by typing

```
>> data(1:2,2:end)
ans =
    0.8913  0.8214  0.9218
    0.7621  0.4447  0.7382
```

You can set values in this way, too. Take the variable `data` and set rows 1 to 2 and all the columns to 0 by

```
>> data(1:2,:)=0
data =
         0       0       0       0
         0       0       0       0
    0.6068  0.4565  0.6154  0.1763
    0.4860  0.0185  0.7919  0.4057
```

Note that the colon operator `:` used on its own in indexing specifies all elements, in this case all columns. Note also that the array indexing in MATLAB starts with 1.

2.4 Programming Structures

This section introduces two simple programming structures in MATLAB: `if-else` statements and `for` loops.

The typical form of `if-else` statements is as follows: The `if` command evaluates a logical expression and executes a group of statements when the expression is true. You can use optional `elseif` and `else` commands that allow for alternative groups of statements. An `end` command terminates the last group of statements. For example,

```
>> if c<0
s=-1;
elseif c>0
s=1;
else
s=0;
end
```

This code sequence sets variable `s` to be -1 or $+1$ if `c` is negative or positive, respectively. If `c` is zero, `s` is set to zero.

You can use a `for` loop to repeat a group of statements for a fixed number of times. `for` loops start with the `for` keyword and the definition of the iteration variable, followed by the group of statements and a terminating `end` command. For example,

```
>> for n=1:10
y(n)=log(n)-0.2*n;
end
```

will produce a row vector y with values $\log(n) - 0.2n$ for $n = 1, 2, \ldots, 10$. Note that MATLAB allows you to get the same vector without using a `for` loop, simply by typing

```
>> n=1:10;
>> y=log(n)-0.2*n;
```

2.5 Plotting with MATLAB

As you have seen above, simple x-y plots can be created using the `plot` command. For example, plotting the previously defined variable `y` against variable `n` can be done using

```
>> plot(n,y)
```

You can plot multiple lines in one plot by using the `hold` function. For example,

```
>> y2=sin(n/2);
>> hold on
>> plot(n,y2,'g:*')
>> hold off
```

produces an additional green, dotted line with markers. You can put labels to the x and y axis using

```
>> xlabel('n')
>> ylabel('y')
```

and add a legend to the graph using

```
>> legend('curve 1','curve 2')
```

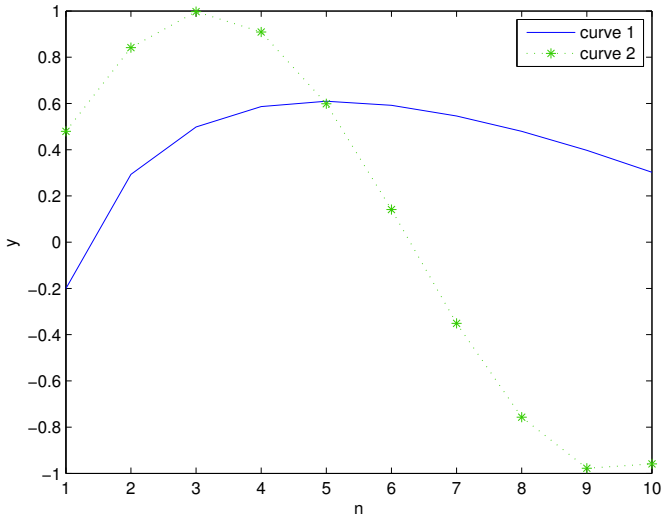

Figure 2: A plot with multiple lines, labels and a legend.

The created plot is shown in Figure 2.

Additionally, you can add a title using the `title` command, or put multiple plots into one figure by using the `subplot` command. For more details, we refer to the corresponding help files.

2.6 Reading and Saving Data

You can save all variables on the MATLAB workspace using the `save` command. Typing

```
>> save mydata
```

creates a "mydata.mat" file in the current directory. You can clear individual variables or the entire workspace using the `clear` command, such as

```
>> clear y
>> clear all
```

You can read the workspace data back into MATLAB with the `load` command:

```
>> load mydata
```

2.7 MATLAB Programs

This section will show you how to create and run a MATLAB program. Surely, you have noticed that the command window is designed to execute individual commands, one at a time. You can automate the execution of many commands or build a complex application by writing a MATLAB program.

A MATLAB program is a collection of MATLAB commands and functions, also known as statements, stored on the disk as a file with extension `.m`, known as an m-file. You can create a new MATLAB program with the MATLAB editor, which can be started using

```
>> edit functionname
```

For example, take the following code sequence

```
f(1)=1;
f(2)=2;
for i=3:8
  f(i)=f(i-1)+f(i-2);
end
```

which creates the first 8 Fibonacci numbers, and save it to `fibonacci.m`. You can run this type of MATLAB program, called a script, by clicking the "Run" button in the MATLAB editor or calling the file name in the command window:

```
>> fibonacci
```

which will store the variable f to the workspace.

```
>> f
f =
    1   2   3   5   8  13  21  34
```

Note that to call an m-file by its name, it has to be located in the current directory or in the MATLAB search path.

As well as MATLAB script files you can also create MATLAB function files. MATLAB function files start with the `function` keyword; the function name should match the name of the file. You can have optional input arguments and specify one or more output arguments. As an example, simply modify `fibonacci.m` as follows:

```
function f=fibonacci(n)
% FIBONACCI fibonacci sequence
f(1)=1;
f(2)=2;
for i=3:n
  f(i)=f(i-1)+f(i-2);
end
```

Note that comments can be made after the percent symbol `%`. Save the changes and call your function, assigning the result to variable `f` while specifying an input argument of 10.

```
>> f=fibonacci(10)
f =
    1   2   3   5   8  13  21  34  55  89
```

You can call a MATLAB function or script from another function or script, allowing you to create large multi-file applications.

To conclude this short outline of MATLAB, you are encouraged to browse for examples and demos in the help section of MATLAB, which can be opened by typing

```
>> doc
```

3 Preparation

1. What does MATLAB stand for?

2. Fundamentally, there is one data type in MATLAB, a rectangular array of numbers, stored in double precision format. It is convenient to think of three special cases. Name and describe them briefly.

3. Provide the MATLAB commands to create a vector `v` with elements -1 -0.9 -0.8 ... 1. Next, use a `for` loop and `if`/ `else` statements to determine the sign of each element of `v` and write it to a new vector `s`.

4. The following matrix is considered

$$G = \begin{pmatrix} 0.6 & 1.5 & 2.3 & -0.5 \\ 8.2 & 0.5 & -0.1 & -2.0 \\ 5.7 & 8.2 & 9.0 & 1.5 \\ 0.5 & 0.5 & 2.4 & 0.5 \\ 1.2 & -2.3 & -4.5 & 0.5 \end{pmatrix}$$

   - What is the size of G? Is it square?
   - Determine the indices of elements that contain the value 0.5.
   - Give indices of elements that contain negative values.

5. Use the MATLAB help (while operating MATLAB or surfing the web) and make sure you understand how to use

   - arithmetic operators, matrix operation and element-wise operations (`help arith`)
   - relational and logic operators (`help relop`, etc.)
   - programming structures (`help if`, `help for`, etc.)
   - plotting functions (`help plot`, etc.), e.g., how to plot multiple lines in one graph, having different color, line style or marker symbols.

6. What is the difference between MATLAB scripts and MATLAB functions?

4 Experiments

4.1 Magic Matrices

The special properties of magic matrices or "Dürer" matrices are often used when introducing MATLAB. Enter the following command

```
>> M=magic(5);
```

What is special about `M`? Try `sum(M)` and `sum(M')`. Next determine

- the first row of `M`, the third column of `M`, column 1 to 3, row 2 to the end of `M`.
- all indices of elements with values > 10, and indices of elements < 4.

4.2 Fibonacci Numbers

The Fibonacci numbers are computed according to the following relation:

$$f_n = f_{n-1} + f_{n-2}, \quad f_0 = f_1 = 1$$

Construct a function `fibonacci.m` by using copy/paste from the text. Your function should have input argument n and return a vector containing the first n Fibonacci numbers.

1. Use your function to calculate the first 12 Fibonacci numbers.
2. Compute the sequence of ratios f_n/f_{n-1} for $n = 1, \ldots, 12$. It is claimed that this ratio converges to the value of the "golden ratio" $\phi = \frac{\sqrt{5}+1}{2}$. Show this approximation using an appropriate plot.

4.3 Statistical Measurements

Uniformly distributed random numbers can be generated in MATLAB using the `rand` function. For example, a random vector of size 1000×1 can be constructed by entering

```
>> x=rand(1000,1);
```

1. Determine the minimum and maximum value of `x`. Further calculate the mean and the standard deviation of `x`.
2. Generate new random numbers using the transformation $y = 4x - 2$. Calculate mean and standard deviation of `y`. Could you have predicted the approximate result?
3. Repeat 1. using `randn` instead of `rand` to generate your random numbers. Also, have a look at the histograms using the `hist` function. Can you tell what `randn` generates?

4.4 An Optimization Example

We want to find the optimal dimensions of a common beverage can ("Cola Dose"), such that the surface area (material) is minimized while the volume

is kept fixed at 330 ml. We assume a simplified cylinder shape with radius r and height h, whose surface area and volume are

$$A = 2\pi r^2 + 2\pi r h \quad \text{and} \quad V = \pi r^2 h.$$

To determine the optimal r and h, proceed as follows:

- Arrange the second equation to $h = \frac{V}{\pi r^2}$ and substitute this into the first equation to obtain $A(r) = 2\pi r^2 + \frac{2V}{r}$.
- Take $V = 330$ (in ml $\equiv$ cm^3) and plot $A(r)$ on the interval $0.5 \leq r \leq 10$ (in cm).
- Find the minimum of $A(r)$, determine the optimal r and highlight it in your plot. Does your result (r and h) match a common beverage can?

4.5 The Moving Average

Let us have a look at some climate data, and discover how the moving average can be used to highlight long-term trends. Enter the following command

```
>> load glob_warm.mat
```

which loads variables `year` and `Ta` into your workspace. `Ta` is a vector of the yearly averaged temperature anomaly, recorded over the years 1850-2007. Here, anomaly means difference to the previous year.

To get rid of short-term fluctuations, we apply a moving average and therewith smooth our data x_n for $n = 1, \ldots, N$. A new value $\tilde{x}_n$ is obtained by averaging x_n over $2m + 1$ surrounding values:

$$\tilde{x}_n = \frac{x_{n-m} + \cdots + x_{n-1} + x_n + x_{n+1} + \cdots + x_{n+m}}{2m + 1}$$

To implement the moving average, proceed as follows:

- Use a `for` loop to iterate over all elements of your data vector.
- Take into account the edges separately by using an `if-else` statement. We consider edges as iterations, when fewer summands than $2m + 1$ are available for averaging, i.e., for $n < m + 1$ and $n > N - m$.

- To carry out an averaging operation, first select all desired vector elements, then use the `sum` function and divide by the number of summands.

Experiment with different values for m and look at the result. Finally, plot the original series and the smoothed version for the case $m = 7$ in one plot. Use a legend and label the axes properly.

4.6 A Signal Processing Example

1. Generate and plot a digital sinusoidal signal, proceed as follows:
   - Construct a digital time vector, `n=0:100;`
   - Set a 'continuous-time' frequency `F=1;` (in Hz) and a sampling time `T=0.05;` (in seconds).
   - Generate a signal `s=sin(2*pi*F*n*T);`
   - Plot your signal using `plot(n,s)` vs. discrete time, or using `plot(n*T,s)` vs. continuous-time. The command `stem(n,s)` is also useful to emphasize the discrete-time nature of sampled signals.
2. Sure, that looks like a sine-wave. Let us look at it in the frequency domain (we will explain the FFT and the periodogram later in the course):
   - Compute a 128-point FFT, `S=fft(s,128);`
   - Calculate the "spectrum" by `P=S.*conj(S);`
   - Generate a normalized frequency vector, `f=(0:127)/128;`
   - Plot the spectrum of your signal using `plot(f,P)` (vs. normalized frequency) or `plot(f/T,P)` (vs. frequency in Hz).
3. Now, let us add a disturbance and see if we can remove it by filtering!
   - Add a 4 Hz distrubance, `s2=s+sin(2*pi*4*n*T);`
   - Plot `s` and `s2` and compare.

- Set the numerator coefficient of a digital filter as `b=[1 1 1 1]/4;` and denominator coefficients as `a=1`. Calculate the frequency response of your filter `[H,w]=freqz(b,a);` and plot it by using `plot(w/pi/2/T,abs(H))` (magnitude vs. frequency in Hz)
- Filter your signal, `sf=filter(b,a,s2);`
- Look at the recovered signal in both time and frequency domain. Observe that the disturbance is not gone, but suppressed!

References

[1] List of officially recommended MATLAB tutorials. The MathWorks, Inc. 2008.
`www.mathworks.com/academia/student_center/tutorials`

[2] Good reference list of MATLAB tutorials (in German, "Skripte zur Einführung in MATLAB")
`www.ant.uni-bremen.de/teaching/glab/matlab/literatur`

[3] Cleve Moler. "Numerical Computing with MATLAB". The MathWorks, Inc. 2004.
`www.mathworks.com/moler`

[4] A three day MATLAB tutorial from MIT.
`www.mit.edu/people/abbe/matlab/main.html`

[5] Kermit Sigmon. "MATLAB Primer". Third edition 1998.