NTNU
Norwegian University of
Science and Technology
Department of Electronics and
Telecommunications

An introduction to MATLAB

Version 1.2

Anders Gjendemsjø

Contents

1 Introduction

MATLAB, short for Matrix Laboratory, is a simple and flexible programming environment for a wide range of problems such as signal processing, optimization, linear programming and so on. The basic MATLAB software package can be extended by using add-on toolboxes. Examples of such toolboxes are: Signal Processing, Filter Design, Statistics and Symbolic Math. Comprehensive documentation for MATLAB is available at

`http://www.mathworks.com`

In particular, an excellent (extensive) getting started guide is available at

`http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf`

There is also a very active newsgroup for MATLAB related questions,

`comp.soft-sys.matlab`

MATLAB is an interpreted language. This implies that the source code is not compiled but interpreted on the fly. This is both an advantage and a disadvantage. MATLAB allows for easy numerical calculation and visualization of the results without the need for advanced and time consuming programming. The disadvantage is that it can be slow, especially when bad programming practices are applied.

2 Using MATLAB

2.1 MATLAB Help

MATLAB has a great on-line help system accessible using the help command. Typing `help <function>` will return text information about the chosen function. For example to get information about the built-in function `sum` type:

```
help sum
```

To list the contents of a toolbox type `help <toolbox>`, e.g. to show all the functions of the signal processing toolbox enter

```
help signal processing
```

If you don't know the name of the function but a suitable keyword use the `lookfor` followed by a keyword string, e.g.

```
lookfor 'discrete fourier'
```

To explore the extensive help system use the "Help menu" or try the commands `helpdesk` or `demo`.

2.2 Matrices, vectors and scalars

MATLAB uses matrices as the basic variable type. Scalars and vectors are special cases of matrices having size 1x1, 1xN or Nx1. In MATLAB there are a few conventions for entering data:

- Elements of a row is separated with blanks or commas.
- Each row is ended by a semicolon, `;`.
- A list of elements must be surrounded by square brackets, `[ ]`.

Example 2.1. Creating basic variables.

```
x = 1 (scalar)

y = [2 4 6 8 10] (row vector)

z = [2; 4; 6; 8; 10] (column vector)

A = [4 3 2 1 0; 1 3 5 7 9] (2 x 5 matrix)
```

Regularly spaced values of a vector can be entered using the following compact notation

```
start:skip:end
```

Example 2.2. A more compact way of entering variables than in Example 2.1.

```
y= 2 : 2 : 10

A=[4:-1:0;1:2:9]
```

If the skip is omitted it will be set to 1, i.e the following are equivalent

```
start:1:end and start:end
```

To create a string use the single quotation mark `"'"`, e.g. by entering

```
x = 'This is a string'
```

2.3 Indexing matrices and vectors

Indexing variables is straightforward. Given a matrix M the element in the i'th row, j'th column is given by `M(i,j)`. For a vector v the i'th element is given by `v(i)`.
Note that the lowest allowed index in MATLAB is 1. This is in contrast with many other programming languages (e.g. JAVA and C), as well as the common notation used in signal processing, where indexing starts at 0.
The colon operator is also of great help when accessing specific parts of matrices and vectors, as shown below.

Example 2.3. This example shows the use of the colon operator for indexing matrices and vectors.

```
A(1,:)      returns the first row of the matrix A

A(:,3)      returns the third column of the matrix A

A(2,1:5)    returns the first five elements of the second row

x(1:2:10)   returns the first five odd-indexed elements of the
            vector x
```

2.4 Basic operations

MATLAB has built-in functions for a number of arithmetic operations and functions. Most of them are straightforward to use. Table 1 lists the some commonly used functions. Let x and y be scalars, M and N matrices.

	MATLAB
xy	`x*y`
x^y	`x^y`
e^x	`exp(x)`
$\log(x)$	`log10(x)`
$\ln(x)$	`log(x)`
$\log_2(x)$	`log2(x)`
MN	`M*N`
M^{-1}	`inv(M)`
M^T	`M'`
$det(M)$	`det(M)`

Table 1: Common mathematical operations in MATLAB.

- **Dimensions** - MATLAB functions `length` and `size` are used to find the dimensions of vectors and matrices, respectively.
- **Elementwise operations** - If an arithmetic operation should be done on each component in a vector (or matrix), rather than on the vector (matrix) itself, then the operator should be preceded by ".", e.g `.*`, `.^` and `./`.

Example 2.4. Elementwise operations, part I

Let $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. Then `A^2` will return $AA = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$,

while `A.^2` will return $\begin{bmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.

Example 2.5. Elementwise operations, part II

Given a vector x, find a vector y having elements $y(n) = \frac{1}{\sin(x(n))}$.

This can be easily be done in MATLAB by typing

```
y=1./sin(x)
```

Note that using `/` in place of `./` would result in the (common) error `Matrix dimensions must agree`.

2.5 Complex numbers

MATLAB has excellent support for complex numbers with several built-in functions available. The imaginary unit is denoted by i or (as preferred in electrical engineering) j. To create complex variables $z_1 = 7 + j$ and $z_2 = 2e^{j\pi}$ simply enter

```
z1 = 7 + j
```

```
z2 = 2*exp(j*pi)
```

Table 2 gives an overview of the basic functions for manipulating complex numbers, where z is a complex number.

	MATLAB
$\mathrm{Re}(z)$	`real(z)`
$\mathrm{Im}(z)$	`imag(z)`
$\lvert z \rvert$	`abs(z)`
$\angle z$	`angle(z)`
z^*	`conj(z)`

Table 2: Manipulating complex numbers in MATLAB

3 Graphical representation of data

MATLAB provides a great variety of functions and techniques for graphical display of data. The flexibility and ease of use of MATLAB's plotting tools is one of its key strengths. In MATLAB graphs are shown in a figure window. Several figure windows can be displayed simultaneously, but only one is *active*. All graphing commands are applied to the active figure. The command `figure(n)` will activate figure number `n` or create a new figure indexed by `n`.

3.1 Tools for plotting

In this section we present some of the most commonly used functions for plotting in MATLAB.

- `plot` - The `plot` and `stem` functions can take a large number of arguments, see `help plot` and `help stem`. For example the line type and color can easily be changed. `plot(y)` plots the values in vector `y` versus their index. `plot(x,y)` plots the values in vector `y` versus `x`. `plot` produces a piecewise linear graph between its data values. With enough data points it looks continuous.
- `stem` - Using `stem(y)` the data sequence `y` is plotted as stems from the x axis terminated with circles for the data values. `stem` is the natural way of plotting sequences. `stem(x,y)` plots the data sequence `y` at the values specified in `x`.
- `xlabel('string')` - Labels the x-axis with string.
- `ylabel('string')` - Labels the y-axis with string.
- `title('string')` - Gives the plot the title string.

To illustrate this consider the following example.

Example 3.1. In this example we plot the function $y = -x^2$ for $x \in [-2, 2]$.

```
x = -2:0.2:2;
y = x.^2;

figure(1);
plot(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple plot');
```

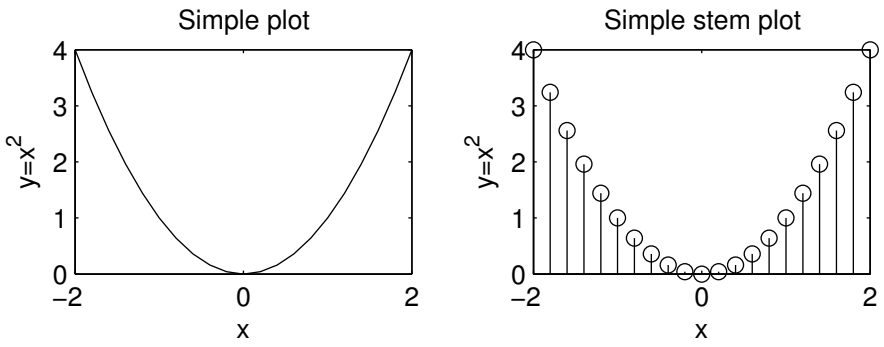


Figure 1: Basic plots.

```
figure(2);
stem(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple stem plot');
```

The result is shown in figure 1.

Some more commands that can be helpful when working with plots:

- `hold on / off` - Normally hold is off. This means that the plot command *replaces* the current plot with the new one. To add a new plot to an existing graph use `hold on`. If you want to overwrite the current plot again, use `hold off`.
- `legend('plot1','plot2',...,'plot N')` - The legend command provides an easy way to identify individual plots when there are more than one per figure. A legend box will be added with strings matched to the plots.
- `axis([xmin xmax ymin ymax])` - Use the `axis` command to set the axis as you wish. Use `axis on/off` to toggle the axis on and off respectively.
- `subplot(m,n,p)` Divides the figure window into m rows, n columns and selects the pth subplot as the current plot, e.g `subplot(2,1,1)` divides the figure in two and selects the upper part. `subplot(2,1,2)` selects the lower part.
- `grid on / off` - This command adds or removes a rectangular grid to your plot.

Example 3.2. This example illustrates `hold`, `legend` and `axis`

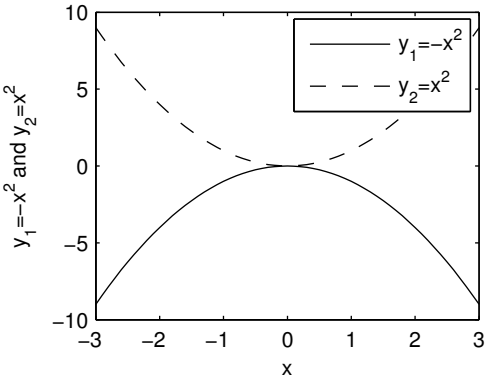
```
x = -3:0.1:3; y1 = -x.^2; y2 = x.^2;

figure(1);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
legend('y_1=-x^2','y_2=x^2');

figure(2);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
legend('y_1=-x^2','y_2=x^2');
axis([-1 1 -10 10]);
```

The result is shown in figure 2.

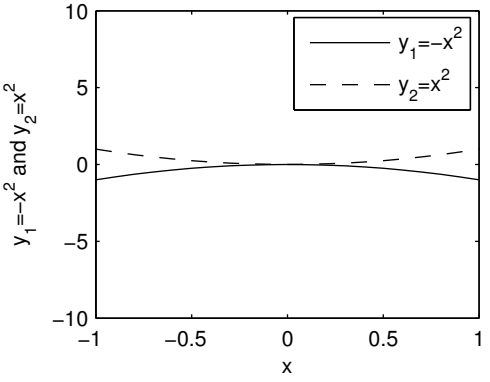
Example 3.3. In this example we illustrate `subplot` and `grid`.

```
x = -3:0.2:3; y1 = -x.^2; y2 = x.^2;

subplot(2,1,1);
plot(x,y1);
xlabel('x'); ylabel('y_1=-x^2');
grid on;

subplot(2,1,2);
plot(x,y2);
xlabel('x');
ylabel('y_2=x^2');
```

The result is shown in figure 3.



(a) Plot with x from -3 to 3.



(b) Plot with x from -1 to 1.

Figure 2: MATLAB plot illustrating hold, legend, axis.

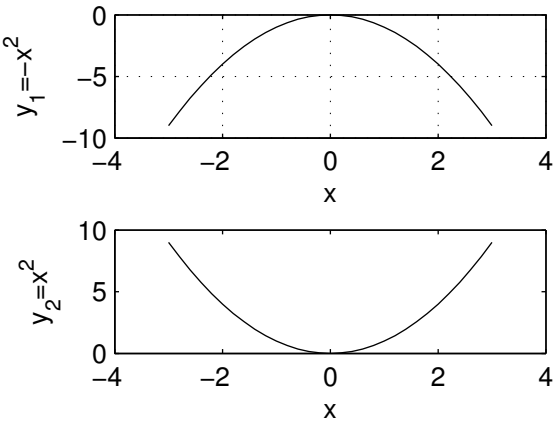


Figure 3: MATLAB plot illustrating subplot and grid

3.2 Printing and exporting graphics

After you have created your figures you may want to print them or export them to graphic files. In the "File" menu use "Print" to print the file or "Save As" to save your figure to one of the many available graphics formats. Using these options should be sufficient in most cases, but there are also a large number of adjustments available by using "Export setup", "Page Setup" and "Print Setup".

3.3 3D plots

We end this section on graphics with a sneak peek into 3D plots. The new functions here are `meshgrid` and `mesh`. In the example below we see that `meshgrid` produces `x` and `y` vectors suitable for 3D plotting and that `mesh(x,y,z)` plots `z` as a function of both `x` and `y`.

Example 3.4. Creating our first 3D plot.

```
[x,y] = meshgrid(-3:.1:3);
z = x.^2+y.^2;
mesh(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z=x^2+y^2');
```

The result is shown in figure 4.

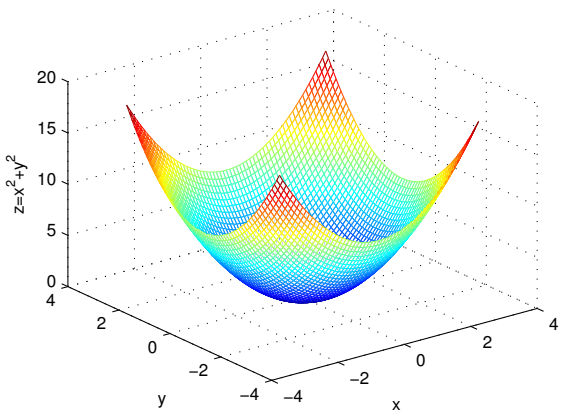


Figure 4: 3D plot

---

4 Improving your MATLAB code

In this section we present programming tips for improving your MATLAB code.

4.1 Script files

Script files, also called M-files as they have extension `.m`, make MATLAB programming much more efficient than entering individual commands at the command prompt. A script file consists of MATLAB commands that together perform a specific task.
The M-file is a text file which can be created and edited by any plain text editor like Notepad, `emacs` or the built-in MATLAB editor. To create a script in MATLAB use: **File - New - M-file** from the menu. An example script is shown below.

Example 4.1. Example script.

```
n = 0:pi/100:2*pi; % create an index vector
y = cos(2*pi*n);   % create a vector y
plot(n,y)          %plot y versus n
```

As shown above the **%-sign** allows for comments. Saving the script as `foo.m` it can be executed as `foo` from the command prompt or by clicking the run button in the MATLAB editor.
Script files are very practical and should be the preferred alternative compared to the command prompt in most cases.

4.2 Program flow

As in most programming languages program flow can be controlled by using statements such as `for`, `while`, `if`, `else`, `elseif` and `switch`. These statements can be used both in `.m` files and at the command prompt, the latter being highly inconvenient. Below we show some examples. Use `help` to get more details.

- **for** - To print "Hello World" 10 times we write

  ```
  for n=1:10
      disp('Hello World');
  end
  ```

  `for` loops can in many cases be avoided by vectorizing your code, more about that in section 4.5.

- **if, else and elseif** - Classics that never goes out of style.

```
 if a == b
    a = b + 1
elseif a > b
    a = b - 1
else
    a = b
end
```

4.3 Creating MATLAB Functions

Sometimes it is convenient to create your own functions for use in MATLAB. Functions are program routines, usually implemented in M-files. Functions can take input arguments and return output arguments. They operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

Example 4.2. Create a function for calculating the sum of the $N+1$ first terms of geometric series. Let $N < \infty$.

Solution: The sum of the $N+1$ terms of a geometric series is given by $ssum = \sum_{n=0}^{N} a^n$. An implementation of this sum as a function accepting the input arguments `a` and `N` is shown below.

```
function ssum = geom(a,N)
    n=0:N;
    ssum = sum(a.^n);
end
```

The function `geom` can then be called, e.g from the command prompt. The function call `geom(0.9,10)` returns 6.8619.

To illustrate some more MATLAB programming we take on the task of creating a MATLAB function that will compute the sum of an arbitrary geometric series, $\sum_{n=0}^{N} a_n$.

Example 4.3. Create a function to calculate the sum of an arbitrary geometric series.

Solution: For $N < \infty$ we know that the sum converges regardless of a. As N goes to ∞ the sum converges only for $|a| < 1$, and the sum is given by the formula $\sum_{n=0}^{\infty} a_n = \frac{1}{1-a}$. A possible implementation is given as:

```
function ssum = geomInf(a,N)
    if(N==inf)
        if(abs(a)>=1)
            error('This geometric series will diverge.');
        else
            ssum=1/(1-a);
        end
    else
        n=0:N;
        ssum = sum(a.^n);
    end
end
```

4.4 Learning from existing code

Wouldn't it be great to learn from the best? Using the command `type` followed by a function name the source code of the function is displayed. As the built in functions are written by people with excellent knowledge of MATLAB, this is a great feature for anyone interested in learning more about MATLAB.

4.5 Vectorizing loops

As mentioned earlier, in MATLAB one should try to avoid loops. This can be done by vectorizing your code. The idea is that MATLAB is very fast on vector and matrix operations and correspondingly slow with loops. We illustrate this by an example.

Example 4.4. Given $a_n = n$ and $b_n = 1000 - n$ for $n = 1, ..., 1000$. Calculate $\sum_{n=1}^{1000} a_n b_n$ and store it in the variable `ssum`.

Solution: It might be tempting to implement the above calculation as

```
a = 1:1000;
b = 1000 - a;

ssum=0;
for n=1:1000 %poor style...
    ssum = ssum +a(n)*b(n);
end
```

Recognizing that the sum is the inner product of the vectors a and b, ab^{T},we can do better:

```
ssum = a*b' %Vectorized, better!
```

4.6 Other useful details

- A **semicolon** added at the end of a line tells MATLAB to suppress the command output to the display.
- MATLAB and **case sensitivity**. For variables MATLAB is *case sensitive*, i.e. `b` and `B` is different. For functions it is *case insensitive*, i.e. `sum` and `SUM` refers to the same function.
- Often it is useful to **split a statement** over multiple lines. To split a statement across multiple lines, enter three periods "..." at the end of the line to indicate it continues on the next line.

Example 4.5. Splitting $y = a + b + c$ over multiple lines.

```
y = a...
  + b...
  c;
```
