Introduction to MATLAB

Els Pekalska, Marnok van der Schaar
Pattern Recognition Group, Faculty of Applied Sciences
Delft University of Technology
2007 - 2009

Send comments to els@ph.tudelft.nl
## Contents

1. **Introduction**
   - Preliminaries .................................................. 3

2. **Getting started with MATLAB**
   - 1.1 Input via the command-line .................................. 4
   - 1.2 Help facilities ............................................. 5
   - 1.3 Interrupting a command or program ....................... 6
   - 1.4 Path .......................................................... 6
   - 1.5 Workspace issues ............................................ 7
   - 1.6 Saving and loading data .................................... 7

3. **Basic syntax and variables**
   - 2.1 MATLAB as a calculator .................................... 7
   - 2.2 An introduction to floating-point numbers ............... 8
   - 2.3 Assignments and variables ................................ 9

4. **Mathematics with vectors and matrices**
   - 3.1 Vectors ...................................................... 10
     - 3.1.1 Colon notation and extracting parts of a vector ..... 11
     - 3.1.2 Column vectors and transposing .................... 12
     - 3.1.3 Product, divisions and powers of vectors .......... 13
   - 3.2 Matrices .................................................... 16
     - 3.2.1 Special matrices .................................... 16
     - 3.2.2 Building matrices and extracting parts of matrices 17
     - 3.2.3 Operations on matrices ............................... 20

5. **Visualization**
   - 4.1 Simple plots ............................................... 22
   - 4.2 Several functions in one figure .......................... 23
   - 4.3 Other 2D plotting features - optional .................. 24
   - 4.4 Printing ................................................... 25
   - 4.5 3D line plots ............................................. 26
   - 4.6 Plotting surfaces .......................................... 26
   - 4.7 Animations - optional .................................... 27

6. **Control flow**
   - 5.1 Logical and relational operators ........................ 28
     - 5.1.1 The command find .................................... 30
   - 5.2 Conditional code execution .............................. 31
   - 5.3 Loops ...................................................... 33
   - 5.4 Evaluation of logical and relational expressions in the control flow structures 35

7. **Numerical analysis**
   - 6.1 Curve fitting .............................................. 36
   - 6.2 Interpolation ............................................... 37
   - 6.3 Evaluation of a function ................................ 37
     - 6.3.1 Inline functions ..................................... 38
   - 6.4 Integration and differentiation .......................... 39
   - 6.5 Numerical computations and the control flow structures .................................................. 40

8. **Script and function m-files**
   - 7.1 Script m-files .............................................. 40
   - 7.2 Function m-file ............................................ 41
     - 7.2.1 Special function variables ........................... 43
     - 7.2.2 Local and global variables .......................... 44
     - 7.2.3 Indirect function evaluation - optional ............ 44
   - 7.3 Scripts vs. functions .................................... 46

9. **Text** .......................................................... 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Character strings</td>
<td>47</td>
</tr>
<tr>
<td>8.2</td>
<td>Text input and output</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>Cell arrays and structures - optional</td>
<td>52</td>
</tr>
<tr>
<td>9.1</td>
<td>Cell arrays</td>
<td>52</td>
</tr>
<tr>
<td>9.2</td>
<td>Structures</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>Optimizing the performance of MATLAB code</td>
<td>54</td>
</tr>
<tr>
<td>10.1</td>
<td>Vectorization - speed-up of computations</td>
<td>54</td>
</tr>
<tr>
<td>10.2</td>
<td>Array preallocation</td>
<td>55</td>
</tr>
<tr>
<td>10.3</td>
<td>MATLAB's tricks and tips</td>
<td>56</td>
</tr>
<tr>
<td>11</td>
<td>File input/output operations</td>
<td>59</td>
</tr>
<tr>
<td>11.1</td>
<td>Text files</td>
<td>60</td>
</tr>
<tr>
<td>11.2</td>
<td>Binary files - optional</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>Writing and debugging MATLAB programs</td>
<td>62</td>
</tr>
<tr>
<td>12.1</td>
<td>Structural programming</td>
<td>62</td>
</tr>
<tr>
<td>12.2</td>
<td>Debugging</td>
<td>64</td>
</tr>
<tr>
<td>12.3</td>
<td>Recommended programming style</td>
<td>64</td>
</tr>
</tbody>
</table>
1 Getting started with MATLAB

MATLAB is a tool for mathematical (technical) calculations. First, it can be used as a scientific calculator. Next, it allows you to plot or visualize data in many different ways, perform matrix algebra, work with polynomials or integrate functions. Like in a programmable calculator, you can create, execute and save a sequence of commands in order to make your computational process automatic. It can be used to store or retrieve data. In the end, MATLAB can also be treated as a user-friendly programming language, which gives the possibility to handle mathematical calculations in an easy way. In summary, as a computing/programming environment, MATLAB is especially designed to work with data as well as vectors, matrices and images. Therefore, PRTOOLS, a toolbox for Pattern Recognition purposes, and DPLIB, a toolbox for Image Processing, have been developed under MATLAB.

Under Windows, you can start MATLAB by double clicking on the MATLAB icon that should be on the desktop of your computer; on Unix systems, type `matlab` at the command line. Running MATLAB creates one or more windows on your screen. The most important is the Command Window, which is the place you interact with MATLAB; i.e., it is used to enter commands and display text results. The string `>>` is the MATLAB prompt (or `>>` for the Student Edition). When the Command Window is active, a cursor appears after the prompt, indicating that MATLAB is waiting for your command. MATLAB responds by printing text in the Command Window or by creating a Figure Window for graphics. To exit MATLAB use the command `exit` or `quit`.

1.1 Input via the command-line

MATLAB is an interactive system; commands followed by Enter are executed immediately. The results are, if desired, displayed on screen. However, execution of a command will be possible if the command is typed according to the rules. Table 1 shows a list of commands used to solve indicated mathematical equations (e. g. a, x and y are numbers). Below you find basic information to help you starting with MATLAB:

- Commands in MATLAB are executed by pressing Enter or Return. The output will be displayed on screen immediately. Try the following:

```
>> 3 * 7.5
>> 18/4
>> 3 * 7
```

Note that spaces are not important in MATLAB.

The result of the last performed computation is assigned to the variable ans, which is an example of a MATLAB built-in variable. It can be used in the next command. For instance:

```
>> 14/4
ans =
3.5000
>> ans^(-8)
ans =
5.4399e-04
```

5.4399e-04 is a computer notation of 5.4399 * 10^-4 (see Preliminaries). Note that ans is always overwritten by the last command.

- You can also define your own variables. Look how the information is stored in the variables a and b:

```
>> a = 14/4
a =
3.5000
>> b = a^(-8)
b =
5.4399e-04
```

Read Preliminaries to better understand the concept of variables. You will learn more on MATLAB variables in section 2.3.

- When the command is followed by a semicolon `;`, the output is suppressed. Check the difference between the following expressions:

```
>> 3 + 7.5
>> 3 + 7.5;
```
• It is possible to execute more than one command at the same time; the separate commands should then be divided by commas (to display the output) or by semicolons (to suppress the output display), e.g.:

```matlab
>> sin(pi/4), cos(pi); sin(0)
ans =
 0.7071
ans =
 0
```

Note that the value of `cos(pi)` is not printed.

• By default, MATLAB displays only 5 digits. The command `format long` increases this number to 15, `format short` reduces it to 5 again. For instance:

```matlab
>> 312/66
ans =
 5.5714
>> format long
>> 312/66
ans =
 4.7142857142857
```

• The output may contain some empty lines; this can be suppressed by the command `format compact`. In contrast, the command `format loose` will insert extra empty lines.

• To enter a statement that is too long to be typed in one line, use three periods `...` followed by Enter or Return. For instance:

```matlab
>> sin(1) + sin(2) - sin(3) + sin(4) - sin(5) + sin(6) - ...
  sin(8) + sin(9) - sin(10) + sin(11) - sin(12)
ans =
 1.0357
```

• MATLAB is case sensitive, for example, `a` is written as `a` in MATLAB; `A` will result then in an error.

• All text after a percent sign `%` until the end of a line is treated as a comment. Enter e.g. the following:

```matlab
>> sin(3.14159) % this is an approximation of sin(pi)
```

You will notice that some examples in this text are followed by comments. They are meant for you and you should skip them while typing those examples.

• Previous commands can be fetched back with the `[Del]` key. The command can also be changed, the `[←]` and `[→]` keys may be used to move around in a line and edit it. In case of a long line, Ctrl-a and Ctrl-e might be useful; they allow to move the cursor at the beginning or the end of the line, respectively.

• To recall the most recent command starting from e.g. `c`, type `c` at the prompt followed by the `[↑]` key. Similarly, `cos` followed by the `[↑]` key will find the last command starting from `cos`.

Since MATLAB executes the command immediately, it might be useful to have an idea of the expected outcome. You might be surprised how long it takes to print out a 1000 x 1000 matrix!

### 1.2 help-facilities

MATLAB provides assistance through extensive online help. The `help` command is the simplest way to get help. It displays the list of all possible topics. To get a more general introduction to `help`, try:

```matlab
>> help help
```

If you already know the topic or command, you can ask for a more specified help. For instance:

```matlab
>> help ops
```

gives information on the operators and special characters in MATLAB. The topic you want help on must be exact and spelled correctly. The `lookfor` command is more useful if you do not know the exact name of the command or topic. For example:
<table>
<thead>
<tr>
<th>Mathematical notation</th>
<th>MATLAB command</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a + b)</td>
<td>(a + b)</td>
</tr>
<tr>
<td>(a - b)</td>
<td>(a - b)</td>
</tr>
<tr>
<td>(ab)</td>
<td>(a \times b)</td>
</tr>
<tr>
<td>(\frac{a}{b})</td>
<td>(a \div b)</td>
</tr>
<tr>
<td>(x^2)</td>
<td>(x^2)</td>
</tr>
<tr>
<td>(\sqrt{x})</td>
<td>(sqrt(x))</td>
</tr>
<tr>
<td>(\log_2)</td>
<td>(log(x))</td>
</tr>
<tr>
<td>(\pi)</td>
<td>(pi)</td>
</tr>
<tr>
<td>(4 \times 10^3)</td>
<td>(4e3)</td>
</tr>
<tr>
<td>(i)</td>
<td>(1) or (j)</td>
</tr>
<tr>
<td>(3 - 4i)</td>
<td>(3-4i)</td>
</tr>
<tr>
<td>(e, e^x)</td>
<td>(exp(1), exp(x))</td>
</tr>
<tr>
<td>(\sin x, \arctan x)</td>
<td>(\sin(x), \arctan(x))</td>
</tr>
</tbody>
</table>

Table 1: Translation of mathematical notation to MATLAB commands.

>> lookfor inverse

displays a list of commands, with a short description, for which the word inverse is included in its help-text. You can also use an incomplete name, e.g. lookfor inv. Besides the help and lookfor commands, there is also a separate mouse driven help. The helpwin command opens a new window on screen which can be browsed in an interactive way.

Exercise 1.
- Is the inverse cosine function, known as \(\cos^{-1}\) or \(\arccos\), one of the MATLAB’s elementary functions?
- Does MATLAB have a mathematical function to calculate the greatest common divisor?
- Look for information on logarithms.

Use help or lookfor to find out.

1.3 Interrupting a command or program

Sometimes you might spot an error in your command or program. Due to this error it can happen that the command or program does not stop. Pressing Ctrl-C (or Ctrl-Break on PC) forces MATLAB to stop the process. Sometimes, however, you may need to press a few times. After this the MATLAB prompt (>>) re-appears. This may take a while, though.

1.4 Path

In MATLAB, commands or programs are contained in m-files, which are just plain text files and have an extension ‘.m’. The m-file must be located in one of the directories which MATLAB automatically searches. The list of these directories can be listed by the command path. One of the directories that is always taken into account is the current working directory, which can be identified by the command pwd. Use path, addpath and repath functions to modify the path. It is also possible to access the path browser from the File menu-bar, instead.

Exercise 2.
Type path to check which directories are placed on your path. Add you personal directory to the path (assuming that you created your personal directory for working with MATLAB).

1.5 Workspace issues

If you work in the Command Window, MATLAB memorizes all commands that you entered and all variables that you created. These commands and variables are said to reside in the MATLAB workspace. They might be easily recalled when needed, e.g. to recall previous commands, the \(F1\) key is used. Variables can be verified with the commands who, which gives a list of variables present in the workspace, and when, which includes also information on name, number of allocated bytes and class of variables. For example, assuming that you performed all commands from section 1.1, after typing who you should get the following information:
>> who
Your variables are:
a    ans    b    x

The command `clear <name>` deletes the variable `<name>` from the MATLAB workspace, clear or clear all removes all variables. This is useful when starting a new exercise. For example:

>> clear a x
>> who
Your variables are:
ans    b

1.6 Saving and loading data

The easiest way to save or load MATLAB variables is by using (clicking) the File menu-bar, and then selecting the Save Workspace as... or Load Workspace... items respectively. Also MATLAB commands exist which save data to files and which load data from files.

The command `save` allows for saving your workspace variables either into a binary file or an ASCII file (check Preliminaries on binary and ASCII files). Binary files automatically get the '.mat' extension, which is not true for ASCII files. However, it is recommended to add a '.txt' or '.dat' extension.

Exercise 3.
Learn how to use the `save` command by exercising:

>> a1 = sin(pi/4);
>> c1 = cos(pi/4); c2 = cos(pi/2);
>> str = 'hello world';
>> save % saves all variables in binary format to matlab.mat
>> save data % saves all variables in binary format to data.mat
>> save nndata a1, c1 % saves numeric variables a1 and c1 to nndata.mat
>> save strdata str % saves a string variable str to strdata.mat
>> save allosc.dat c* -ascii % saves c1, c2 in 8-digit ascii format to allosc.dat

The `load` command allows for loading variables into the workspace. It uses the same syntax as `save`.

Exercise 4.
Assuming that you have done the previous exercise, try to load variables from the created files. Before each `load` command, clear the workspace and after loading check which variables are present in the workspace (use `who`).

>> load % loads all variables from the file matlab.mat
>> load data a1 c1 % loads only specified variables from the file data.mat
>> load str data % loads all variables from the file strdata.mat

It is also possible to read ASCII files that contain rows of space separated values. Such a file may contain comments that begin with a percent character. The resulting data is placed into a variable with the same name as the ASCII file (without the extension). Check, for example:

>> load allosc.dat % loads data from allosc.dat into variable allosc
>> who % lists variables present in the workspace now

2 Basic syntax and variables

2.1 MATLAB as a calculator

There are three kinds of numbers used in MATLAB: integers, real numbers and complex numbers. In addition, MATLAB has representations of the non-numbers: Inf, for positive infinity, generated e.g. by 1/0, and NaN, Not-a-Number, obtained as a result of the mathematically undefined operations such as 0/0 or ∞ - ∞.
You have already got some experience with MATLAB and you know that it can be used as a calculator. To do that you can, for example, simply type:

```matlab
>> (23+17)/7
```

The result will be:

```matlab
ans =
56.8571
```

MATLAB has six basic arithmetic operations, such as: +, -, *, / or \ (right and left divisions) and ^ (power). Note that the two division operators are different:

```matlab
>> 19/3
ans =
6.3333
>> 19\3, 3/19
ans =
0.1579
ans =
```

Basic built-in functions, trigonometric, exponential, etc, are available for a user. Try help elfun to get the list of elementary functions.

**Exercise 5.**

Evaluate the following expressions by hand and use MATLAB to check the answers. Note the difference between the left and right divisors. Use help to learn more on commands rounding numbers, such as: round, floor, ceil, etc.

- \(2/2 + 3\)
- \(8/5\)
- \(8 \times (5/4)\)
- \(7 - 5 \times 4/9\)
- \(6 - 2/5 + 7^{-2} - 1\)
- \(10/2\times 3 + 2 \times 4\)

```matlab
>> 2\2 + 3
ans =
3.2
>> 8/5
ans =
1.6
>> 8 \times (5/4)
ans =
5.0
>> 7 - 5 \times 4/9
ans =

3.8889
>> 6 - 2/5 + 7^{-2} - 1
ans =

5.4706
>> 10/2\times 3 + 2 \times 4
ans =

16.0
```

**Exercise 6.**

Define the format in MATLAB such that empty lines are suppressed and the output is given with 15 digits.

Calculate:

```matlab
>> pi
ans =
3.14159265358979
>> sin(pi)
ans =
```

Note that the answer is not exactly 0. Use the command format to put MATLAB in its standard-format.

---

### 2.2 An introduction to floating-point numbers

In a computer, numbers can be represented only in a discrete form. It means that numbers are stored within a limited range and with a finite precision. Integers can be represented exactly with the base of 2 (read Preliminaries on bits and the binary system). The typical size of an integer is 16 bits, so the largest positive integer, which can be stored, is \(2^{16} = 65536\). If negative integers are permitted, then 16 bits allow for representing integers between \(-32768\) and \(32767\). Within this range, operations defined on the set of integers can be performed exactly.

However, this is not valid for other real numbers. In practice, computers are integer machines and are capable of representing real numbers only by using complicated codes. The most popular code is the floating point standard. The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, meaning that the decimal point can float. Note that most floating-point numbers that a computer can represent are just approximations. Therefore, care should be taken that these approximations lead to reasonable results. If a programmer is not careful, small discrepancies in the approximations can cause meaningless results. Note the difference between e.g. the integer arithmetic and floating-point arithmetic:
Integer arithmetic:  Floating-point arithmetic
2 + 4 = 6  18/7 = 2.5714
3 * 4 = 12  2.5714 * 7 = 17.9998
25/11 = 2  10000/3 = 3.3333 + 03

When describing floating-point numbers, precision refers to the number of bits used for the fractional part. The larger the precision, the more exact fractional quantities can be represented. Floating-point numbers are often classified as single precision or double precision. A double-precision number uses twice as many bits as a single-precision value, so it can represent fractional values much better. However, the precision itself is not double. The extra bits are also used to increase the range of magnitudes that can be represented.

MATLAB relies on a computer’s floating point arithmetic. You could have noticed that in the last exercise the value of \( \sin(\pi) \) was almost zero, and not completely zero. It came from the fact that both the value of \( \pi \) is represented with a finite precision and the \( \sin \) function is also approximated.

The fundamental type in MATLAB is double, which stands for a representation with a double precision. It uses 54 bits. The single precision obtained by using the single type offers 32 bits. Since most numeric operations require high accuracy the double type is used by default. This means, that when the user is inputting integer values in MATLAB (for instance, \( k = 4 \)), the data is still stored in double format.

The relative accuracy might be defined as the smallest positive number \( \epsilon \) that added to 1, creates the result larger than 1, i.e. \( 1 + \epsilon > 1 \). It means that in floating-point arithmetic, for positive values smaller than \( \epsilon \), the result equals to 1 (in exact arithmetic, of course, the result is always larger than 1). In MATLAB, \( \epsilon \) is stored in the built-in variable eps \( \approx 2 \times 10^{-16} \). This means that the relative accuracy of individual arithmetic operations is about 15 digits.

\[ \text{END INTERMEZZO} \]

### 2.3 Assignments and variables

Working with complex numbers is easily done with MATLAB.

**Exercise 7.** Choose two complex numbers, for example \(-3 + 2i\) and \(5 - 7i\). Add, subtract, multiply, and divide these two numbers.

During this exercise, the complex numbers had to be typed four times. To reduce this, assign each number to a variable. For the previous exercise, this results in:

```matlab
>> z = -3 + 2*i; u = 5 - 7*i;
>> yi = z + w; y2 = z - w;
>> y3 = z * w;
>> y4 = z / w; y6 = v / z;
```

Formally, there is no need to declare (i.e. define the name, size and the type of) a new variable in MATLAB. A variable is simply created by an assignment (e.g. \( z = -3 + 2i \)), i.e. values are assigned to variables. Each newly created numerical variable is always of the double type, i.e. real numbers are approximated with the highest possible precision. You can change this type by converting it into e.g. the single type\(^*\). In some cases, when huge matrices should be handled and precision is not very important, this might be a way to proceed. Also, when only integers are taken into consideration, it might be useful to convert the double representations into e.g. int32\(^*\) integer type. Note that integer numbers are represented exactly, no matter which numeric type is used, as long as the number can be represented in the number of bits used in the numeric type.

Bear in mind that undefined values cannot be assigned to variables. So, the following is not possible:

```matlab
>> clear x;
>> f = x^2 + 4 * sin(x)
```

It becomes possible by:

```matlab
>> x = pi / 3; f = x^2 + 4 * sin(x)
```

Variable name begins with a letter, followed by letters, numbers or underscores. MATLAB recognizes only first 31 characters of the name.

**Exercise 8.**

Here are some examples of different types of MATLAB variables. You do not need to understand them all now, since you will learn more about them during the course. Create them manually in MATLAB:

\(^*\) a variable \( s \) is converted into a different type by performing e.g. \( s = \text{single}(a) \), \( s = \text{int32}(a) \) etc.
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ans</td>
<td>the default variable name used for storing the last result</td>
</tr>
<tr>
<td>pi</td>
<td>π = 3.14159...</td>
</tr>
<tr>
<td>eps</td>
<td>the smallest positive number that added to 1, creates a result larger than 1</td>
</tr>
<tr>
<td>inf</td>
<td>representation for positive infinity, e.g. 1/0</td>
</tr>
<tr>
<td>nan or NaN</td>
<td>representation for not-a-number, e.g. 0/0</td>
</tr>
<tr>
<td>i or j</td>
<td>i = j = -1</td>
</tr>
<tr>
<td>nargin/nargout</td>
<td>number of function input/output arguments used</td>
</tr>
<tr>
<td>realmin/realfmax</td>
<td>the smallest/largest usable positive real number</td>
</tr>
</tbody>
</table>

Table 2: Built-in variables in MATLAB.

```matlab
>> this_is_my_very_simple_variable_today = 5 \% check what happens; the name is very long
>> x = 5
\% what is the problem with this command?
>> X = [1 2; 3 4; 5 6]
\% a matrix
>> c = 'e'
\% a character
>> str = 'Hello world'
\% a string
>> a = ['j'; 'e'; 'b'; 'i'; 'n']
\% try to guess what it is
```

Check the types by using the command who. Use clear <name> to remove a variable from the workspace.

As you already know, MATLAB variables can be created by an assignment. There is also a number of built-in variables, e.g. pi, eps or i, summarised in Table 2. In addition to creating variables by assigning values to them, another possibility is to copy one variable, e.g. b into another, e.g. a. In this way, the variable a is automatically created (if a already existed, its previous value is lost):

```matlab
>> b = 10.5;
>> a = b;
```

A variable can be also created as a result of the evaluated expression:

```matlab
>> a = 10.5; c = a^2 + sin(pi*x)/4;
```

or by loading data from text or '.*.mat' files.

If `min` is the name of a function (see help min), then a defined, e.g. as:

```matlab
>> b = 5; c = 7;
>> a = min (b, c);
\%
```

will call that function, with the values b and c as parameters. The result of this function (its return value) will be written (assigned) into a. So, variables can be created as results of the execution of built-in or user-defined functions (you will learn more how to built own functions in section 7.2).

Important: do not use variable names which are defined as function names (for instance mean or error)! If you are going to use a suspicious variable name, use help <name> to find out if the function already exists.

### 3 Mathematics with vectors and matrices

The basic element of MATLAB is a matrix (or an array). Special cases are:

- a 1 x 1 matrix: a scalar or a single number;
- a matrix existing only of one row or one column: a vector.

Note that MATLAB may behave differently depending on the input, whether it is a number, a vector or a 2D matrix.

#### 3.1 Vectors

Row vectors are lists of numbers separated either by commas or by spaces. They are examples of simple arrays.

First element has index 1. The number of entries is known as the length of the vector (the command length exists as well). Their entities are referred to as elements or components. The entries must be enclosed in [ ]:

```matlab
[1, 2, 3]
```

There is always one exception of the rule: variable i is often used as counter in a loop, while it is also used as $i = \sqrt{-1}$. 

A number of operations can be done on vectors. A vector can be multiplied by a scalar, or added/subtracted to/from another vector with the same length, or a number can be added/subtracted to/from a vector. All these operations are carried out element-by-element. Vectors can be also built from the already existing ones.

```
>> v = [-1, 2, 7]; w = [2, 3, 4];
>> u = v + w  % an element-by-element sum
ans =
   1  5 11
>> vv = v + 2   % add 2 to all elements of vector v
vv =
   1  4  9
>> t = [2*v, -w]
ans =
   -2  4 14 -2  -3  -4
```

Also, a particular value can be changed or displayed:

```
>> v(2) = -1  % change the 2nd element of v
v =
  -1  -1  7
>> v(2)  % display the 2nd element of v
ans =
   7
```

3.1.1 Colon notation and extracting parts of a vector

A colon notation is an important shortcut, used when producing row vectors (see Table 3 and help colon):

```
>> 2:5
ans =
   2   3   4   5
>> -2:3
ans =
   -2   -1   0   1   2   3
```

In general, \texttt{first:step:last} produces a vector of entities with the value \texttt{first}, incrementing by the \texttt{step} until it reaches \texttt{last}:

```
>> 0.2:0.5:2.4
ans =
   0.2000   0.7000   1.2000   1.7000   2.2000
>> -3:3:10
ans =
   -3    0    3    6    9
>> 1.5:-0.5:-0.5  % negative step is also possible
ans =
   1.5000   1.0000   0.5000    0   -0.5000
```

Parts of vectors can be extracted by using a colon notation:

```
>> x = [-1:2:6, 2, 3, -2]  % -1:2:6 => -1 1 3 5
x =
   -1   1   3   5   2   3   -2
```
3.1.2 Column vectors and transposing

To create column vectors, you should separate entries by new lines or by a semicolon ';

```matlab
>> z = [1
    7]
ans =
    1
    7
```

```matlab
>> u = [-1; 3; 5] .
ans =
    -1
    3
    5
```

The same operations as on row vectors can be performed on column vectors. However, you cannot for example add a column vector to a row vector. To do that, you need an operation called transposing, which converts a column vector into a row vector and vice versa:

```matlab
>> u';
ans =
    -1
    3
    5
```

% u is a column vector and u' is a row vector

```matlab
>> v = [-1 2 7];
```

% v is a row vector

```matlab
>> u + v
??? Error using ==> +
Matrix dimensions must agree.
```

% you cannot add a column vector u to a row vector v

```matlab
>> u' + v
ans =
    -2
    5
    12
```

If z is a complex vector, then z' gives the conjugate transpose of z, e.g.:

```matlab
>> z = [1+2i, -1+i]
ans =
    1.0000 + 2.0000i
    -1.0000 + 1.0000i
```

% this is the conjugate transpose

```matlab
>> z'
ans =
    1.0000 - 2.0000i
    -1.0000 - 1.0000i
```

% this is the traditional transpose

```matlab
>> z.'
ans =
    1.0000 + 2.0000i
    -1.0000 + 1.0000i
```
3.1.3 Product, divisions and powers of vectors

You can now compute the inner product between two vectors \( x \) and \( y \) of the same length, \( x^T y = \sum_i x_i y_i \), in a simple way:

\[
\begin{align*}
\text{>> u} & \quad \text{[}-1, 3, 6\text{]} & \quad \% \text{ a column vector} \\
\text{>> v} & \quad \text{[}-1, 2, 7\text{]} & \quad \% \text{ a column vector} \\
\text{>> u \cdot v} & \quad \% \text{ you cannot multiply a column vector by a column vector} \\
\text{?? Error using ==> *} \\
\text{Inner matrix dimensions must agree.}
\end{align*}
\]

\[
\text{>> u' \ast v} \quad \% \text{ this is the inner product}
\]

\[
\text{ans} = \quad 42
\]

Another way to compute the inner product is by the use of the dot product, i.e. \( \cdot \), which performs element-wise multiplication. For two vectors \( x \) and \( y \), of the same length, it is defined as a vector \( [x_1y_1, x_2y_2, \ldots, x_ny_n] \). Thus, the corresponding elements of two vectors are multiplied. For instance:

\[
\begin{align*}
\text{>> u \ast v} & \quad \% \text{ this is an element-by-element multiplication} \\
1 & \quad 6 \\
6 & \quad 35 \\
\text{>> sum(u \ast v)} & \quad \% \text{ this is an another way to compute the inner product} \\
\text{ans} = \quad 42 \\
\text{>> x} & \quad \text{[4 3 1]} \quad \% \text{ z is a row vector} \\
\text{>> sum(u' \ast z)} & \quad \% \text{ this is the inner product} \\
\text{ans} = \quad 10 \\
\text{>> p \ast z'} & \quad \% \text{ since z is a row vector, u' \ast z' is the inner product} \\
\text{ans} = \quad 10
\end{align*}
\]

You can now tabulate easily values of a function for a given list of arguments. For instance:

\[
\begin{align*}
\text{>> x} & \quad \text{[1:0.5:4]} \\
\text{>> y} & \quad \text{sqrt(x) \ast cos(x)} \\
y = & \quad \begin{bmatrix} 
0.6403 & 0.0886 & -0.6885 & -1.2667 & -1.7147 & -1.7520 & -1.3073 
\end{bmatrix}
\end{align*}
\]

Mathematically, it is not defined how to divide one vector by another. However, in MATLAB, the operator \( ./ \) is defined to perform an element-by-element division. It is, therefore, defined for vectors of the same size and type:

\[
\begin{align*}
\text{>> x} & \quad \text{[2:2:10]} \\
x = & \quad \begin{bmatrix} 
2 & 4 & 6 & 8 & 10 
\end{bmatrix} \\
\text{>> y} & \quad \text{[6:10]} \\
y = & \quad \begin{bmatrix} 
6 & 7 & 8 & 9 & 10 
\end{bmatrix} \\
\text{>> x/y} \\
\text{ans} = & \quad \begin{bmatrix} 
0.3333 & 0.5714 & 0.7500 & 0.8889 & 1.0000 
\end{bmatrix} \\
\text{>> z} & \quad \text{[1:3]} \\
z = & \quad \begin{bmatrix} 
1 & 2 & 3 
\end{bmatrix} \\
\text{>> x/z} \\
\text{Warning: Divide by zero.} \\
\text{ans} = & \quad \begin{bmatrix} 
2.0000 & \text{Inf} & 4.0000 & 3.3333 
\end{bmatrix} \\
\text{>> z/x} \\
\text{Warning: Divide by zero.} \\
\end{align*}
\]
### Table 3: Manipulation of (groups of) matrix elements.

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(1,:)</td>
<td>A(:,1)</td>
</tr>
<tr>
<td>A(:,j)</td>
<td>j-th column of A</td>
</tr>
<tr>
<td>A(i,:)</td>
<td>i-th row of A</td>
</tr>
<tr>
<td>A(k,:,:)</td>
<td>(k-1)(k+1) x (n-m+1) matrix with elements A_{ij} with 1 ≤ i ≤ k, 1 ≤ j ≤ n</td>
</tr>
<tr>
<td>v(1,:)</td>
<td>vector-part ( (v_1, v_{j+1}, ..., v_n) ) of vector v</td>
</tr>
</tbody>
</table>

### Table 4: Frequently used matrix operations and functions.

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>r = rank(A)</td>
<td>( \text{r} ) becomes the rank of matrix A</td>
</tr>
<tr>
<td>r = det(A)</td>
<td>( \text{r} ) becomes the determinant of matrix A</td>
</tr>
<tr>
<td>r = size(A)</td>
<td>( \text{r} ) becomes a row-vector with 2 elements: the number of rows and columns of A</td>
</tr>
<tr>
<td>r = trace(A)</td>
<td>( \text{r} ) becomes the trace (sum of diagonal elements) of matrix A</td>
</tr>
<tr>
<td>r = norm(v)</td>
<td>( \text{r} ) becomes the Euclidean length of vector v</td>
</tr>
<tr>
<td>C = A + B</td>
<td>sum of two matrices</td>
</tr>
<tr>
<td>C = A - B</td>
<td>subtraction of two matrices</td>
</tr>
<tr>
<td>C = A * B</td>
<td>multiplication of two matrices</td>
</tr>
<tr>
<td>C = A .+ B</td>
<td>'element-by-element' multiplication (A and B are of equal size)</td>
</tr>
<tr>
<td>C = A .^ B</td>
<td>power of a matrix (1 ≤ B; can also be used for A^-1)</td>
</tr>
<tr>
<td>C = A ./ B</td>
<td>'element-by-element' division (A and B are of equal size)</td>
</tr>
<tr>
<td>A / B</td>
<td>finds the solution to the least squares sense to the system of equations ( AX = B )</td>
</tr>
<tr>
<td>A .' / B</td>
<td>finds the solution of ( XA = B ), analogous to the previous command</td>
</tr>
<tr>
<td>C = inv(A)</td>
<td>( C ) becomes the inverse of A</td>
</tr>
<tr>
<td>C = null(A)</td>
<td>( C ) is an orthonormal basis for the null space of A obtained from the singular value decomposition</td>
</tr>
<tr>
<td>C = orth(A)</td>
<td>( C ) is an orthonormal basis for the range of A</td>
</tr>
<tr>
<td>C = rref(A)</td>
<td>( C ) is the reduced row echelon form of A</td>
</tr>
<tr>
<td>L = eig(A)</td>
<td>( L ) is a vector containing the (possibly complex) eigenvalues of a square matrix A</td>
</tr>
<tr>
<td>D = eig(A)</td>
<td>produces a diagonal matrix ( D ) of eigenvalues and a full matrix ( X ) whose columns are the corresponding eigenvectors of ( A )</td>
</tr>
<tr>
<td>S = svd(A)</td>
<td>( S ) is a vector containing the singular values of A</td>
</tr>
<tr>
<td>U, S, V = svd(A)</td>
<td>( U ) is a diagonal matrix with non-negative diagonal elements in decreasing order; columns of ( U ) and ( V ) are the accompanying singular vectors</td>
</tr>
<tr>
<td>X = linspace(a, b, n)</td>
<td>generates a vector ( x ) of ( n ) equally spaced points between ( a ) and ( b )</td>
</tr>
<tr>
<td>X = logspace(a, b, n)</td>
<td>generates a vector ( x ) starting at ( 10^a ) and ending at ( 10^b ) containing ( n ) values</td>
</tr>
<tr>
<td>A = eye(n)</td>
<td>( A ) is an ( n \times n ) identity matrix</td>
</tr>
<tr>
<td>A = zeros(n, n)</td>
<td>( A ) is an ( n \times n ) matrix with zeros (default ( m = n ))</td>
</tr>
<tr>
<td>A = ones(n, n)</td>
<td>( A ) is an ( n \times n ) matrix with ones (default ( m = n ))</td>
</tr>
<tr>
<td>A = diag(v)</td>
<td>gives a diagonal matrix with the elements ( v_1, v_2, ..., v_n ) on the diagonal</td>
</tr>
<tr>
<td>X = tril(A)</td>
<td>( X ) is lower triangular part of ( A )</td>
</tr>
<tr>
<td>X = triu(A)</td>
<td>( X ) is upper triangular part of ( A )</td>
</tr>
<tr>
<td>A = rand(A, m)</td>
<td>( A ) is an ( n \times m ) matrix with elements uniformly distributed between 0 and 1</td>
</tr>
<tr>
<td>A = randn(m, n)</td>
<td>( A ) is an ( m \times n ) matrix with elements standard normal distributed</td>
</tr>
<tr>
<td>r = max(A)</td>
<td>( r ) is a vector with the maximum value of the elements in each column of ( A )</td>
</tr>
<tr>
<td>r = max(A)</td>
<td>( r ) is the maximum of all elements if ( A ) is a vector</td>
</tr>
<tr>
<td>r = min(A)</td>
<td>( r ) is the minimum</td>
</tr>
<tr>
<td>r = sum(A)</td>
<td>( r ) is the sum</td>
</tr>
</tbody>
</table>

15
3.2 Matrices

Row and column vectors are special types of matrices. An \( n \times k \) matrix is a rectangular array of numbers
having \( n \) rows and \( k \) columns. Defining a matrix in MATLAB is similar to defining a vector. The generalization
is straightforward, if you see that a matrix consists of row vectors (or column vectors). Commas or spaces are
used to separate elements in a row, and semicolons are used to separate individual rows. For example, the
matrix \( \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \) is defined as:

\[
A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}
\]

% row by row input

\[
A = \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\]

Another examples are, for instance:

\[
A2 = \begin{bmatrix} 1:4 \\ -1:2:6 \end{bmatrix}
\]

\[
A2 = \\
1 & 2 & 3 & 4 \\
-1 & 1 & 3 & 5
\]

\[
A3 = \begin{bmatrix} 1 \\ -4 \\ 7 \end{bmatrix}
\]

\[
A3 = \\
1 & 3 \\
-4 & 7
\]

From that point of view, a row vector is a \( 1 \times k \) matrix and a column vector is an \( n \times 1 \) matrix. Transposing
a vector changes it from a row to a column or the other way around. This idea can be extended to a matrix,
where transposing interchanges rows with the corresponding columns, as in the example:

\[
A2 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -1 & 1 & 3 & 5 \end{bmatrix}
\]

\[
A2^T
\]

% transpose of A2

\[
A2^T = \\
1 & -1 \\
2 & 1 \\
3 & 3 \\
4 & 5
\]

\[
\text{size}(A2)
\]

% returns the size (dimensions) of A2: 2 rows, 4 columns

\[
\text{size}(A2)
\]

\[
\text{size}(A2^T)
\]

\[
\text{size}(A2^T)
\]

3.2.1 Special matrices

There is a number of built-in matrices of size specified by the user (see Table 4). A few examples are given
below:

\[
E = []
\]

% an empty matrix of 0-by-0 elements!

\[
E = \\
[]
\]

\[
\text{size}(E)
\]

\[
\text{size}(E)
\]

\[
I = \text{eye}(3);
\]

% the 3-by-3 identity matrix
\[ x = [2; -1; 7]; I*x \] 
\[ x \] is such that for any 3-by-1 \( x \) holds \( I*x = x \)

\[ R = \begin{bmatrix} 2 & 3 & -2 \\ 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{bmatrix} \] 
% create a diagonal matrix with \( r \) on the diagonal

\[ A = \begin{bmatrix} 1 & 2 & 3; \\ 4 & 5 & 6; \\ 7 & 8 & 9; \end{bmatrix} \] 
% extracts the diagonal entries of \( A \)

\[ B = \text{ones}(3,2) \] 
\[ B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \]

\[ C = \text{zeros}(\text{size}(C')) \] 
% a matrix of all zeros of the size given by \( C' \)

\[ D = \text{rand}(2,3) \] 
% a matrix of random numbers; you will get a different one!

\[ v = \text{linspace}(1,2,4) \] 
% a vector is also an example of a matrix

\[ v = \begin{bmatrix} 1.0000 \\ 1.3333 \\ 1.6667 \\ 2.0000 \end{bmatrix} \]

3.2.2 Building matrices and extracting parts of matrices

It is often needed to build a larger matrix from the smaller ones:

\[ x = [4; -1], \ y = [-1 \ 3] \]

\[ y = \begin{bmatrix} -1 \\ 3 \end{bmatrix} \]

\[ X = [x \ y'] \] 
% \( X \) consists of the columns \( x \) and \( y' \)

\[ X = \begin{bmatrix} 4 & -1 \\ -1 & 3 \end{bmatrix} \]

\[ T = [-1 \ 3 \ 4; 4 \ 5 \ 6]; \ t = 1:3; \]
% add to \( T \) a new row, namely the row vector \( t \)

\[ T = \begin{bmatrix} -1 & 3 & 4 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} \]

\[ C = [1 \ 5; 4 \ 5; 0 \ 2]; \] 
% \( C \) is a matrix of the 3-by-2 size; check \( \text{size}(C) \)

\[ T2 = [T \ 0] \] 
% concatenate two matrices
% extract the 3rd row of A
ans =  
7 8 9  
% extract the 2nd column of A
ans =  
2 5 8 1 5  
% extract the rows 1st and 2nd of A
ans =  
1 2 3  
4 5 8  
% extract a part of A
ans =  
4 5  
4 5  

As you have seen in the examples above, it is possible to manipulate (groups of) matrix-elements. The commands are shortly explained in Table 3.

The concept of an empty matrix [] is also very useful in MATLAB. For instance, a few columns or rows can be removed from a matrix by assigning an empty matrix to it. Try for example:

```matlab
>> C = [1 2 3 4; 5 6 7 8; 1 1 1 1];
>> D = C; D(:,2) = [];
```

% now a copy of C is in D; remove the 2nd column of D
```
>> C ((1,3), :) = []
```

% remove the rows 1 and 3 from C

Exercise 11.
Clear all variables. Define the matrix: \( A = \begin{bmatrix} 1 & 4 & 5 & 8; 1 & 1 & 1 \end{bmatrix} \). Predict and check the result of:
- \( x = A(:,3) \)
- \( y = A(3,1:4) \)
- \( B = A(1, 2 : 2) \)
- \( A(1, 1) = 9 + A(2, 3) \)
- \( A(2 : 3, 1 : 3) = [0 0 0; 0 0 0] \)
- \( A(2 : 3, 1 : 2) = [1 1; 3 3] \)
- \( D(2, :) = [] \)

Exercise 12.
Define the matrices: \( T = \begin{bmatrix} 1 & 3 & 4 & 1 & 8 & -4 & 3 \end{bmatrix}; A = \begin{bmatrix} diag(-1:2; 3) \end{bmatrix} \). Perform the following operations on the matrix A:
- extract a vector consisting of the 2nd and 4th elements of the 3rd row
- find the minimum of the 3rd column
- find the maximum of the 2nd row
- compute the sum of the 2nd column
- compute the mean of the 1st and 4th rows
- extract the submatrix consisting of the 1st and 3rd rows and all columns
- extract the submatrix consisting of the 1st and 2nd rows and the 3rd, 4th and 5th columns
- compute the total sum of the 1st and 2nd rows
- add 3 to all elements of the 2nd and 3rd columns

Exercise 13.
Let \( A = \begin{bmatrix} 2 & 4 & 1; 6 & 7 & 2; 3 & 5 & 9 \end{bmatrix} \). Provide the commands which:
- assign the first row of \( A \) to a vector \( x \);
- assign the last 2 rows of \( A \) to a vector \( y \);
- add up the columns of \( A \);
- add up the rows of \( A \);
- compute the standard error of the mean of each column of \( A \) (i.e. the standard deviation divided by the square root of the number of elements used to compute the mean).
A part can be extract from a matrix in a similar way as from a vector. Each element in the matrix is indexed by a row and a column to which it belongs. Mathematically, the element from the i-th row and the j-th column of the matrix A is denoted by $A_{ij}$. MATLAB provides the $A(i,j)$ notation.

% this is also possible: what do you get here?
>> [G; diag(5:6); ones(3,2) T] % you can concatenate many matrices
ans =
     1     4     0     5     0
     8     8     2     0     6
     1     1    -1     3     4
     1     1     4     5     6
     1     1     1     2     3

A part can be extracted from a matrix in a similar way as from a vector. Each element in the matrix is indexed by a row and a column to which it belongs. Mathematically, the element from the i-th row and the j-th column of the matrix A is denoted by $A_{ij}$. MATLAB provides the $A(i,j)$ notation.

% this is not possible: A is a 3-by-3 matrix!
??? Index exceeds matrix dimensions.
>> A(2,3) = A(2,3) + 2*A(1,1) % change the value of A(2,3)
A =
     1     2     3
     4     5     8
     7     8     9

It is easy to extend a matrix automatically. For the matrix A it can be done e.g. as follows:

% assign 5 to the position (5,2); the uninitialized elements become zeros
>> A(5,2) = 5
A =
     1     2     3
     4     5     8
     7     8     9
     0     0     0
     0     5     0

If needed, the other zero elements of the matrix A can be also defined, by e.g.:

% assign vector [2, 1, 2] to the 4th row of A
>> A(4,:) = [2, 1, 2];
% assign: A(5,1) = 4 and A(5,3) = 4
>> A(5,1) = 4; A(5,3) = 4;
% how does the matrix A look like now?
>> A

Different parts of the matrix A can be now extracted:
Table shows some frequently used matrix operations and functions. The important ones are dot operations: dot products and matrix-matrix products. In the case of the dot operation, there are simple operations, dot division and dot power. These operations work as for vectors: they are also available for matrix-matrix operations. A few examples of basic operations are provided below:

\[ \begin{array}{c}
\text{add two matrices; only for is needed instead of } \oplus \\
\text{subtract 2 from all elements of matrix } B \\
\text{divide all elements of the matrix } B \text{ by } A \\
\text{multiply element-by-element} \\
\text{this is possible; equivalent to } A \cdot \text{ones(size(B))} \div B
\end{array} \]

\[ \text{max}(A) \]

\[ \text{min}(A) \]

\[ \text{sum}(A) \]

\[ \text{mean}(A) \]

\[ \text{prod}(A) \]

\[ \text{mean}(A) \]

\[ \text{max}(A) \]

\[ \text{min}(A) \]

\[ \text{sum}(A) \]

\[ \text{mean}(A) \]

\[ \text{prod}(A) \]

\[ \text{mean}(A) \]

\[ \text{max}(A) \]

\[ \text{min}(A) \]

\[ \text{sum}(A) \]

\[ \text{mean}(A) \]

\[ \text{prod}(A) \]

\[ \text{mean}(A) \]

\[ \text{max}(A) \]

\[ \text{min}(A) \]
Exercise 40.
Let \( A = \text{call}(5 \cdot \text{randn}(6, 6)) \). Perform the following:
- Find the indices and list all elements of \( A \) which are smaller than -3;
- Find the indices and list all elements of \( A \) which are smaller than 5 and larger than -1;
- Remove those columns of \( A \) which contain at least one 0 element.
Exercise with both: logical indexing and the command \text{find}. □

5.2 Conditional code execution

Selection control structures, if-blocks, are used to decide which instruction to execute next depending whether expression is \text{TRUE} or not. The general description is given below. In the examples below the command \text{disp} is frequently used. This command displays on the screen the text between the quotes.

- if ... end

\[
\begin{align*}
\text{Syntax} & : \\
\text{if logical_expression} \\
& \quad \text{statement1} \\
& \quad \text{statement2} \\
& \quad \ldots \\
& \quad \text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Example} & : \\
& \quad \text{if (a > 0)} \\
& \quad \quad b = a; \\
& \quad \quad \text{disp ('a is positive');} \\
& \quad \quad \text{end}
\end{align*}
\]

- if ... else ... end

\[
\begin{align*}
\text{Syntax} & : \\
\text{if logical_expression} \\
& \quad \text{block of statements} \\
& \quad \text{evaluated if TRUE} \\
\text{else} \\
& \quad \text{block of statements} \\
& \quad \text{evaluated if FALSE} \\
& \quad \text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Example} & : \\
& \quad \text{if (temperature > 100)} \\
& \quad \quad \text{disp ('Above boiling.');} \\
& \quad \quad \text{toohigh = 1;} \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{disp ('Temperature is OK.');} \\
& \quad \quad \quad \text{toohigh = 0;} \\
& \quad \quad \text{end}
\end{align*}
\]

- if ... elseif ... else ... end

\[
\begin{align*}
\text{Syntax} & : \\
\text{if logical_expression1} \\
& \quad \text{block of statements evaluated if TRUE} \\
\text{elseif logical_expression1 is TRUE} \\
& \quad \text{block of statements evaluated if TRUE} \\
\text{elseif logical_expression2} \\
& \quad \text{block of statements evaluated if TRUE} \\
& \quad \text{block of statements evaluated if no other expression is TRUE} \\
& \quad \text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Example} & : \\
& \quad \text{if (height > 190)} \\
& \quad \quad \text{disp ('very tall');} \\
& \quad \quad \text{elseif (height > 170)} \\
& \quad \quad \quad \text{disp ('tall');} \\
& \quad \quad \quad \text{elseif (height < 150)} \\
& \quad \quad \quad \quad \text{disp ('small');} \\
& \quad \quad \quad \text{else} \\
& \quad \quad \quad \quad \quad \text{disp ('average');} \\
& \quad \quad \text{end}
\end{align*}
\]

Important: To do the exercises below, a script m-file can be of use. It is an external file that contains a sequence of MATLAB commands. What you need to do is to open an editor, enter all commands needed for the solution of a task, save it with the extension '.m' (e.g. nystask.m) and then run it from the Command Window, by typing nystask. To open the MATLAB editor, go to the File menu-bar, choose the Open option and then m-file. A new window will appear where you can write your scripts and save them on the disk. All commands in the script will be executed in MATLAB. You will learn more on script m-files in section 7.1. The script file must be saved in one of the directories in MATLAB's path.

An example: A script short.m can be created from the two lines given below and run by calling short.
Exercise 53.
Use `fmin` (if needed when applicable) to find the maximum of \( f(x) = \frac{1 - x}{(x-1)^2 + 0.1} \) in the interval \([0, 0.5]\). Choose an error tolerance such that the maximum is correct to \(10^{-6}\). If \(x_m\) denotes the computed solution, check the answer by evaluating \( f \) at \(x_m + 10^{-6}\) and \(x_m - 10^{-6}\). The values should be smaller at both these neighboring points. **Hint:** There is no function \(f(x)\) that achieves the maximum. Find the minimum of \(-f(x)\), instead.

Exercise 54.
Determine the maximum of the function \( f(x) = x \cos(x) \) over the interval \(10 < x < 15\).

Exercise 55.
Find the zero, i.e. \(x_0\), of the functions \( f(x) = \cos(x) \) and \( g(x) = \sin(2x) \) around the point 2. Use a command from Table 8. Check that the values are approximately zero for \(x_0\) and the signs of \(f\) at \(x_0 \pm 10^{-4}\).

### 6.4 Integration and differentiation

The integral, or the surface underneath a 2D function, can be determined with the command `trapz`. `trapz` does this by measuring the surface between the x-axis and the data points, connected by the straight lines. The accuracy of this method depends on the distance between the data points:

\[
\int x = 0:0.5:10; y = 0.5 * \sqrt{x} + x.* \sin(x);
\]

\[
\text{integral1} = \text{trapz}(x,y)
\]

\[
\text{integral1} = 15.1665
\]

\[
\int x = 0:0.05:10; y = 0.5 * \sqrt{x} + x.* \sin(x);
\]

\[
\text{integral2} = \text{trapz}(x,y)
\]

\[
\text{integral2} = 15.2846
\]

A more accurate result can be obtained by using the command `quad`, which also numerically evaluates the integral of the function \(f\) specified by a string or by an inline definition. Let \( f = \frac{1}{(x-0.1)^2 + 0.1} + \frac{1}{(x-1)^2 + 0.1} \):

\[
\int f = \frac{1}{(x-0.1)^2 + 0.1} + \frac{1}{(x-1)^2 + 0.1};
\]

\[
\text{integral1} = \text{quad}(f,0,2)
\]

\[
\text{integral2} = \text{quad}(f,0,2)
\]

You can also add an extra parameter to `quad` and `quad2`, specifying the accuracy.

Exercise 56.
Find the integral of the function \( f(x) = e^{-x^2/2} \over (-3,3)\). Exercise with different MATLAB commands and different accuracy. **Hint:**

Differentiation is done by determining the slope in each data point. Numerically this is done by first fitting a polynomial, followed by differentiating this function:

\[
\int x = 0:10;
\]

\[
y = [-10, .24, 1.02, -1.58, 2.84, 2.76, 7.75, .35, 4.53, 5.22, 7.51];
\]

\[
p = \text{polyfit}(x,y,5)
\]

\[
p = 0.0026 -0.0654 0.3684 -0.6988 0.2747 0.0877
\]

\[
p = \text{polyder}(p)
\]

\[
dp = 0.0132 -0.2216 1.0903 -1.3776 0.2747
\]

\[
x1 = \text{inspace}(0,10,200);
\]

\[
x = \text{polyval}(p,x1);
\]

\[
dx = \text{polyval}(dp,x1);
\]

\[
xplot(x,y,'g-*',x1,2,'b',x1,dx,'r-');
\]

\[
\text{legend('data points', 'curve fit', 'derivative')}
\]

Exercise 57.
Determine the integral and derivative of the 4th polynomial fitted through \((x,y)\) with \(x = 4:13\) and \(y = [12.84, 12.00, 7.24, -0.96, -11.16, -20.96, -27.00, -24.98, -9.56, 25.44] \). **Hint:**
<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fplot('f', [min.x max.x])</code></td>
<td>plots the function f on [min.x, max.x]</td>
</tr>
<tr>
<td><code>fmin('f', [min.x max.x])</code></td>
<td>returns the x-value for which the function f reaches minimum on [min.x, max.x]</td>
</tr>
<tr>
<td><code>fzero('f', x0)</code></td>
<td>returns the x-value for which the function f is zero in the neighborhood of x0</td>
</tr>
</tbody>
</table>

Table 8: Evaluation of a function.

```matlab
>> x = 0:pi/8:2*pi;
>> y = sin(8*x);
>> plot(x,y,'b')
>> hold on
>> fplot('sin(8*x)',[0 2*pi],'r')
>> title('sin(8*x)')
>> hold off
```

Execution of these commands results in a figure with an almost straight blue line and a red sine. `fplot` can also be used to plot a defined function:

```matlab
>> f = 'sin(8*x)';
>> fplot(f,[0 2*pi],'r')
>> title(f)
```

### 6.3.1 Inline functions

It might be useful to define a function that will be only used during the current MATLAB session. MATLAB offers a command `inline` to define the so-called inline functions in the Command Window, e.g.:

```matlab
>> f = inline('cos(x).*sin(2*x)')
f =
 Inline function:
 f(x) = cos(x).*sin(2*x)
>> g = inline('sqrt(x.^2+y.^2)','x','y')
g =
 Inline function:
 g(x,y) = sqrt(x.^2+y.^2)
```

You can evaluate this function in a usual way:

```matlab
>> f(-2)
ans =
   -0.3149
>> g(3,4)
ans =
    5
>> A = [1 2; 3 4];
>> B = [2 3; 4 5];
>> g(A,B)
ans =
   2.2361    3.6056
   8.0000    6.4031
```

Let \( f(x) = \frac{1}{x-0.1} + \frac{1}{x-1} \). The point at which \( f \) takes its minimum is found using `fmin` (or `fminbnd` in MATLAB version 6.0). By default, the relative error is of 1e-4. It is possible, however, to obtain a higher accuracy, by specifying an extra parameter while calling `fmin`.

```matlab
>> format long
% you need this format to see the change in accuracy
>> g = inline('1./(x-0.1).^2+0.1+1./(x-1).^2+0.1');
>> fplot(f,[0 2]);
>> x1 = fmin([0,0.3,1],[0,1e-8]);
x1 = f(x1);
xmin = f(x1);
x2 = fmin([0,0.3,1],[0,1e-8]);
x2 = f(x2);
% compare the two answers
```

38
\[ x = -5:5; \]
\[ y = 0.2 \cdot x^3 - x + 2; \]  % a 3rd order polynomial
\[ y = y + \text{randn}(11,1); \]  % add some noise to \( y \)

Determine the 3rd order polynomial fitting \( y \) (note that because of added noise the coefficients are different that originally chosen). Try some other polynomials as well. Plot the results in a figure.

### 4.2 Interpolation

The simplest way to examine an interpolation is by displaying the function plots with the command `plot`. The neighboring data points are connected with straight lines. This is called a linear interpolation. When more data points are taken into account the effect of interpolation becomes less visible. Analyze:

\[ x_1 = \text{linspace}(0,2\pi,2); \]
\[ x_2 = \text{linspace}(0,2\pi,4); \]
\[ x_3 = \text{linspace}(0,2\pi,16); \]
\[ x_4 = \text{linspace}(0,2\pi,256); \]
\[ \text{plot}(x_1,\sin(x_1),x_2,\sin(x_2),x_3,\sin(x_3),x_4,\sin(x_4)) \]
\[ \text{legend}('2 points','4 points','16 points','256 points') \]

There exist also MATLAB functions which interpolate between points, e.g. `interp1` or `spline`, as a 1D interpolation and `interp2`, as a 2D interpolation. Perform the following commands:

\[ n = 50; \]
\[ x = \text{linspace}(0.5,5); \]
\[ y = \sin(x) \cdot \sin(6\cdot x); \]
\[ \text{subplot}(2,1,1); \text{plot}(x,y); \]
\[ \text{hold on} \]
\[ p = \text{randperm}(n); \]
\[ p = p(1:\text{round}(n/2)); \]
\[ px = x(p); \]
\[ py = y(p); \]
\[ \text{plot}(px,py,'ro-') \]
\[ \text{legend('Original','Crude version','Nearest neighbor interpolation') } \]
\[ \text{subplot}(2,1,2); \text{plot}(px,py,'ro-'); \]
\[ \text{hold on} \]
\[ yx = \text{interp1(px,py,x,'linear')}; \]
\[ \text{plot}(x,yx,'g'); \]
\[ yx = \text{spline(px,py,x);} \]
\[ \text{plot}(x,yx,'k') \]
\[ \text{axis tight}; \]
\[ \text{legend('Crude version','Linear interpolation','Spline interpolation') } \]

You can also see a coarse approximation of the function peaks:

\[ X,Y,Z = \text{peaks}(10); \]
\[ [X_1,Y_1] = \text{meshgrid}(-3:-5:3,-3:-5:3); \]
\[ Z1 = \text{interp2}(X,Y,Z,X_1,Y_1); \]
\[ \text{mesh}(X_1,Y_1,Z1); \]

### 4.3 Evaluation of a function

As said before, the command `plot` plots a function by connecting defined data points. When a function is constant and not very interesting over some range and then acts unexpectedly wildly over another, it might be misinterpreted by using `plot`. The command `fplot` is then more useful. An example is:
while (abs (x - xold) > tolerance) | (iter < max_iter)

Try to understand the difference and confirm your expectations by running solve.cos2. What happens to iter? ■

6 Numerical analysis

Numerical analysis can be used whenever it is impossible or very difficult to determine the analytical solution. MATLAB can be used to find, for example, the minimum, maximum or integral of a certain function.

There are two ways to describe data with an analytical function:

- **curve fitting or regression**: finding a smooth curve that best fits (approximates) the data according to a criterion (e.g., best least square fit). The curve does not have to go through any of the data points.

- **interpolation**: the data are assumed to be correct, desired is a way to describe what happens between the data points. In other words, given a finite set of points, the goal of interpolation is to find a function from a given class of functions (e.g., polynomials) which passes through the specified points.

The figure below illustrates the difference between regression (curve fitting) and interpolation:

![Curve fitting and interpolation](image)

In this section we will briefly discuss some issues on curve fitting, interpolation, function evaluation, integration and differentiation. It is also possible to solve differential equations which are shortly treated in "The Student Edition of Matlab" book.

6.1 Curve fitting

MATLAB interprets the best fit of a curve by the 'least squares curve fitting'. The curve used is restricted to polynomials. With the command `polyfit` any polynomial can be fitted to the data. `polyfit (x, y, n)` finds the coefficients of a polynomial of degree n that fits the data (finds the linear relation between x and y). Let's start with finding a linear regression of some data:

```matlab
>> x = 0:0.1:
>> y = [-10 1.02 1.68 2.84 2.76 2.99 4.05 4.83 5.22 7.51]
>> p = polyfit (x, y, 1)               % find the fitting polynomial of order 1
p = 0.6772   -0.3914
```

The output of `polyfit` is a row vector of the polynomial coefficients, the solution of this example is therefore $y = 0.6772x - 0.3914$.

**Exercise 51.**

Execute the above example. Plot the data and the fitted curve in a figure. Determine the 2nd and 9th order polynomial of these data as well, and display the solution in a figure.

**Hint:** `x1 = linspace(xmin,xmax,n)`: results in a vector with n element evenly distributed from `xmin` till `xmax`. `x = polyval(p,x1)`: calculates the values of the polynomial for every element of `x1`. ■

**Exercise 52.**

Define x and y as follows:
A simple example how to use the loop construct can be to draw graphs of \( f(x) = \cos(nx) \) for \( n = 1, \ldots, 9 \) in different subplots. Execute the following script:

```matlab
figure
hold on
x = linspace(0, 2*pi);
for n = 1:9
    subplot(3, 3, n);
    y = cos(n*x);
    plot(x, y);
    axis tight
end
```

Given two vectors \( x \) and \( y \), an example use of the loop construction is to create a matrix \( A \) whose elements are defined, e.g., as \( A_{ij} = x_i y_j \). Enter the following commands to a script:

```matlab
n = length(x);
m = length(y);
for i = 1:n
    for j = 1:m
        A(i, j) = x(i) * y(j);
    end
end
```

and create \( A \) for \( x = [1 \ 2 \ -1 \ 5 \ -7 \ 2 \ 4] \) and \( y = [3 \ 1 \ -5 \ 7] \). Note that \( A \) is of size \( n \times m \). The same problem can be solved by using the while-loop, as follows:

```matlab
n = length(x);
m = length(y);
i = 1; j = 1;
while i <= n
    while j <= m
        A(i, j) = x(i) * y(j);
        j = j + 1;
    end
    i = i + 1;
end
```

Exercise 45.
Determine the sum of the first 50 squared numbers with a control loop.

Exercise 46.
Write a script to find the largest value \( n \) such that the sum: \( \sqrt{1^2} + \sqrt{2^2} + \ldots + \sqrt{n^2} \) is less than 1000.

Exercise 47.
Use a loop construction to carry out the computations. Write short scripts.

1. Given the vector \( x = [1 \ 6 \ 3 \ 9 \ 0 \ 1] \), create a short set of commands that will:
   - add the values of the elements (check with sum);
   - computes the running sum (for element \( j \), the running sum is the sum of the elements from 1 to \( j \); check with `cumsum`);
   - computes the sine of the given \( x \)-values (should be a vector).

2. Given \( x = [4 \ 1 \ 6 \ -1 \ -2 \ 2] \) and \( y = [6 \ 2 \ -7 \ 1 \ 5 \ -1] \), compute matrices whose elements are created according to the following formulas:
   - \( a_{11} = y_1 / x_1 \);
   - \( b_1 = x_1 y_1 \) and add up the elements of \( c \);
   - \( c_{11} = x_1 / (2 + x_1 + y_1) \);
   - \( d_{11} = 1 / \max(x_1, y_1) \).

3. Write a script that transposes a matrix \( A \). Check its correctness with the MATLAB operation: \( A' \).

4. Create an \( m \times n \) array of random numbers (use `rand`). Move through the array, element by element, and set any value that is less than 0.5 to 0 and any value that is greater than (or equal to) 0.5 to 1.

5. Write a script that will use the random-number generator `rand` to determine:
The statements following the first case where the expression matches the choice are executed. This construction can be very handy to avoid long if elseif else end constructions. The expression can be a scalar or a string. A scalar expression matches a choice if expression == choice. A string expression matches a choice if strcmp(expression, choice) returns 1 (is true) (strcmp compares two strings).

**Example**

```
method = 2;
switch method
    case (1,2)
        disp('Method is linear.');
    case 3
        disp('Method is cubic.');
    case 4
        disp('Method is nearest.');
    otherwise
        disp('Unknown method.');
end
```

**Exercise 44.**

Assume that the months are represented by numbers from 1 to 12. Write a script that asks you to provide a month and returns the number of days in that particular month. Alternatively, write a script that asks you to provide a month name (e.g. 'June') instead of a number. Use the switch- construction.

### 5.3 Loops

Looping control structures, loops, are used to repeat a block of statements until some condition is met. Two types of loops exist:

- **The `for` loop** that repeats a group of statements a fixed number of times;

  **Syntax**

  ```
  for index = first:step:last
    block of statements
  end
  ```

  **Example**

  ```
  sumax = 0;
  for i=1:length(x)
    sumax = sumax + x(i);
  end
  ```

  You can specify any step, including a negative value. The index of the `for`-loop can be also a vector. See some examples of possible variations:

  **Example 1**

  ```
  for i=1:2:n
    ...
  end
  ```

  **Example 2**

  ```
  for i=-1:3
    ...
  end
  ```

  **Example 3**

  ```
  for x=0:0.5:4
    disp(x^2);
  end
  ```

  **Example 4**

  ```
  for x=[25 9 81]
    disp(sqrt(x));
  end
  ```

- **The `while` loop**, which evaluates a group of commands as long as expression is TRUE.

  **Syntax**

  ```
  while expression
    statement1
    statement2
    statement3
    ...
  end
  ```

  **Example**

  ```
  N = 100;
  iter = 1;
  masum = 0;
  while iter <= N
    masum = masum + iter;
    iter = iter + 1;
  end
  ```
Exercise 41.
In each of the following questions, evaluate the given code fragments. Investigate each of the fragments for various starting values given on the right. Use MATLAB to check your answers (be careful, since those fragments are not always the proper MATLAB expressions):

1. if \( n > 1 \)
   \[ m = n + 2 \]
   else
   \[ m = n - 2 \]
   end

2. if \( s <= 1 \)
   \[ t = 2s \]
   elseif \( s < 10 \)
   \[ t = 9 - s \]
   elseif \( s < 100 \)
   \[ t = \sqrt{s} \]
   else
   \[ t = s \]
   end

3. if \( t >= 24 \)
   \[ z = 3t + 1 \]
   elseif \( t < 9 \)
   \[ z = t^2 - 2t \]
   else
   \[ z = -t \]
   end

4. if \( 0 < x < 7 \)
   \[ y = 4x \]
   elseif \( 7 < x < 88 \)
   \[ y = -10x \]
   else
   \[ y = 333 \]
   end

Exercise 42.
Create a script that asks for a number \( N \) and computes the drag coefficient \( C \), depending on the Reynolds number \( N \) (make use of the if ... elseif ... construction). The command input might be useful here (use help if needed).

\[
C = \begin{cases} 
0, & N \leq 0 \\
\frac{24}{N}, & N \in [0, 0.1] \\
\frac{24}{N} (1 + 0.14 N^{0.7}), & N \in [0.1, 1e3] \\
0.43, & N \in [1e3, 5e5] \\
0.19 - 8c4/N, & N > 5e5 
\end{cases}
\]

Check whether your script works correctly. Compute the drag coefficient for e.g. \( N = -3e3, 0.01, 56, 1e3, 366 \) (remember that e.g. 3e3 is a MATLAB notation of \( 3 \times 10^3 \)).

Exercise 43.
Write a script that asks for an integer and checks whether it can be divided by 2 or 3. Consider all possibilities, such as: divisible by both 2 and 3, divisible by 2 and not by 3 etc (use the command rem). Another selection structure is switch, which switches between several cases depending on an expression, which is either a scalar or a string.