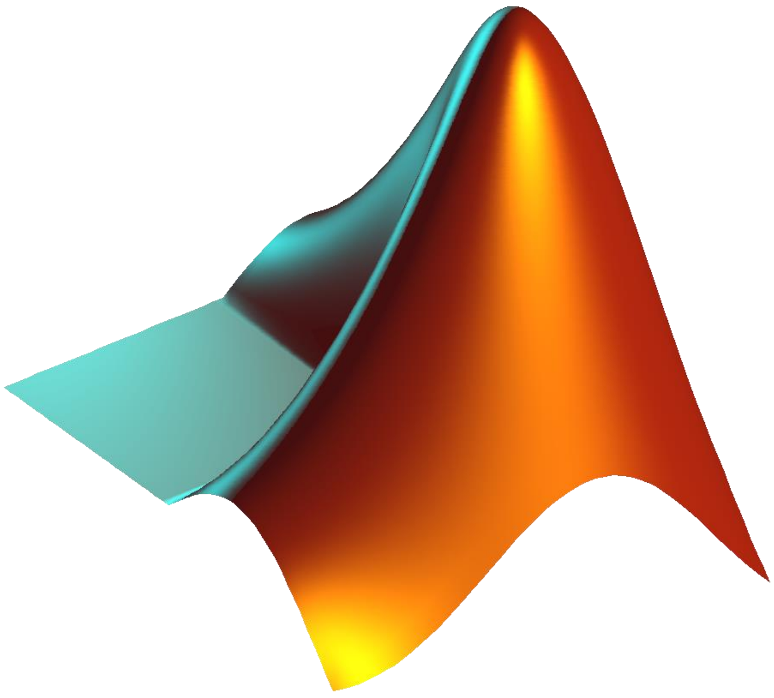
# Practical Course Matlab/Simulink

# MATLAB Fundamentals



## Table of Contents

## List of Tables

## 0 General Information and Advice

The following exercises cover the MATLAB Fundamentals. Each exercise will cover one basic feature of programming with MATLAB.

It is recommended that you write the MATLAB code you produce during this session into an M-File (MATLAB script). This makes it easier for your supervisor to help you in case of problems. You can also save and keep the file for your records.

At the beginning of each exercise, delete all existing variables.

## 1 Arrays: Vectors and Matrices

In this exercise you will learn how to handle numeric arrays. Create a script called "Exercise_01.m" containing your results. Use the comment operator ("%") to separate subtasks.

### Exercise

(1) Use the doc function to find out about the `magic` function.
(2) Create a 3x3 matrix containing all integers from one to nine, that has equal row and column sums and store it in the variable $A$.
(3) Use the colon (:) operator to create the column vector

$$b = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

(4) Solve the following equation for $x$:

$$Ax = b$$

by

a) multiplying the inverse of $A$ to the vector $b$:

$$x = A^{-1}b$$

b) using MATLAB'S more efficient operation for solving linear equations.
Hint: Type `doc inv` in the MATLAB command window to find out more

Use the `tic` and `toc` commands to compare the performance of both operations. Compare the result to the information you retrieved from the `inv`. (You may have to compare the operations several times due to effects of memory allocation)

(5) Manipulate A using elementwise multiplication, reshaping and sorting to create a column vector containing the square numbers from 1 to 9. Several solutions may be possible.
(6) Create a 3x3 matrix B containing only zeros and use linear indexing to fill it in using increasing even numbers to get the following result:

$$B = \begin{bmatrix} 2 & 8 & 14 \\ 4 & 10 & 16 \\ 6 & 12 & 18 \end{bmatrix}$$

(7) Create three new variables B_subs, B_lin and B_log which are all equal to B. Find a way to use subscript indexing, linear indexing and logical indexing to divide every element of $B$ by 4 that has no remainder from this operation (that divisible by 4).

## 2 Data Types

In this exercise you will learn how to handle data types in MATLAB. The table below contains an excerpt of the data of attendants at a party. There are different ways to arrange this data in MATLAB. Save your results in a script called "Exercise_02.m".

**Table 1: Attendants**

| Name | Age | Attendance | Company |
|---|---|---|---|
| Markus Müller | 53 | Yes | Marta Müller<br>Michael Müller<br>Martina Müller |
| Peter Schmidt | 58 | Yes | Ursula Schmidt |
| Beate Maier | 46 | No | Gustav Maier |
| Ursula Leitner | 36 | Yes | - |

### Exercise

(1) Create a cell array called “CellData” containing all data from Table 1, such that each row acts as a container for the data types in Table 2. Arrange the company cell array such that the first column represents the surname and the second column the first name.

**Table 2: Data types**

| Name | Class |
|---|---|
| FirstName | Char |
| Surname | Char |
| Age | Double |
| Attendance | Logical |
| Company | Cell |

*Hint*: If it is more convenient for you, you can use the variable editor to fill in the information after finishing the first entry. Open the variable editor interactively or by entering the `openvar` command.

(2) Save the CellData variable in a file called CellData.mat.

(3) Use the `struct` command to create a structure array called StructData containing all data. Use the Names from Table 2 as field names.

(4) Extract Mr. Müller’s data and save each data field to a variable which is called as in Table 2. Check all variable in your workspace using the `whos` command.

(5) Create a table containing the same data as in the structure. Use the documentation on tables to find the most efficient way.

(6) Find the mean age of all attending guests (hint: use logical indexing to extract a structure array containing attending guests).

(7) Save the expression created in step (6) to a function handle called `getMean(data)`.

## 3 Program Structures

**Table 3: Calculation modes**

| Name | Description |
|---|---|
| sum | Calculate the sum of all integers from 1 to n |
| factorial | Calculate the factorial of n |
| fibonacci | Calculate the $n^{th}$ element of the fibonacci series |

Program structures are an important part of MATLAB. In this exercise, you will learn how to implement a control flow for your program. You control the execution of the code by using conditional statements and loops.

(1) Use the `input` command to retrieve a user input regarding a calculation mode according to Table 3. Use an `if` statement to display the description that belongs to the user's choice in the command window.
(2) Repeat exercise (1) using a `switch` statement.
(3) Use a `for`-loop to determine the sum of all integers from 1 to n. Set n to 100 and test your program.
(4) Use a `while`-loop to implement a factorial calculation. Compare the code from exercise (3) and (4).
(5) Use either loop statement to implement the Fibonacci series. It is defined as follows:

$$a_0 = 0, a_1 = 1, a_n = a_{n-1} + a_{n-2}$$

(6) Calculation time can be long for large numbers of n. Using the tic and toc commands, implement a check of the calculation time and abort the loop if a maximal calculation time is exceeded. Increase n and reduce the maximal calculation time until it is exceeded.

## 4 Scripts and Functions

Scripts can be used to automate tasks, that have to be executed repeatedly. Scripts work with data from the base workspace. In contrary, functions have their own workspace, which is separate from the base workspace. Functions can be used to encapsulate and reuse code blocks and thus arranged in a clean manner.

### Exercise

(1) Create a script containing the Fibonacci program from Exercise 3 and run it. Observe the base workspace using the Workspace Browser.
(2) Clear all variable from the base workspace. Implement a function that takes an integer n as input and outputs the nth element of the Fibonacci series. Add a new section in the script from (1), invoke the created function and display the output in the command window.
(3) Scripts and functions can call any function in the search path. Create a function called "`fibonacci_recursive`", that recursively calls itself to find the two previous elements. Be sure to cover the cases, where there are no previous values (n=0, n=1), using a conditional statement. Compare the speed of the recursive function to the function containing the `for`-loop using the `tic` and `toc` commands for large numbers n.
(4) Create a function called SphereData with one input for the radius of the sphere and three outputs for
   - Volume
   - Surface
   - And circumference.

   Implement nested functions with no input and a single output to calculate each output of the parent function.
(5) Convert each nested function from (4) to a local function. What changes to the function header have to be made?