Interactive Matlab Course

2009-2010

March 22, 2010

Contents

Chapter 1

Basic elements of MATLAB

1.1 What is MATLAB?

MATLAB is a computer program designed for technical calculations. Its name is an abbreviation of "Matrix Laboratory". In this program, matrix computations are implemented in a straightforward manner. However, many other pre-programmed routines are available. In addition, it is possible to implement user-defined calculation routines. MATLAB allows users to implement calculations in relatively short programming time. When these calculations have been performed, they can be visualised by means of several plot-routines.

MATLAB can be extended with so called toolboxes. In general, these are collections of MATLAB functions that are tailored for special classes of problems. The '(Extended) Symbolic Math Toolbox' is an exception. To a great extent, this toolbox consists of ingredients from the computer algebra package MuPAD. MATLAB performs calculations with the aid of matrices. These are rectangular schemes of numbers, which are also called arrays. The 'Symbolic Math Toolbox' is quite different from MATLAB itself, since symbolic calculations hardly make any use of arrays. For this reason it is important to know whether a function used is a MATLAB function or a 'Symbolic Math Toolbox' function. This will not be completely clear at the outset, but it will become clearer after regular use of MATLAB.

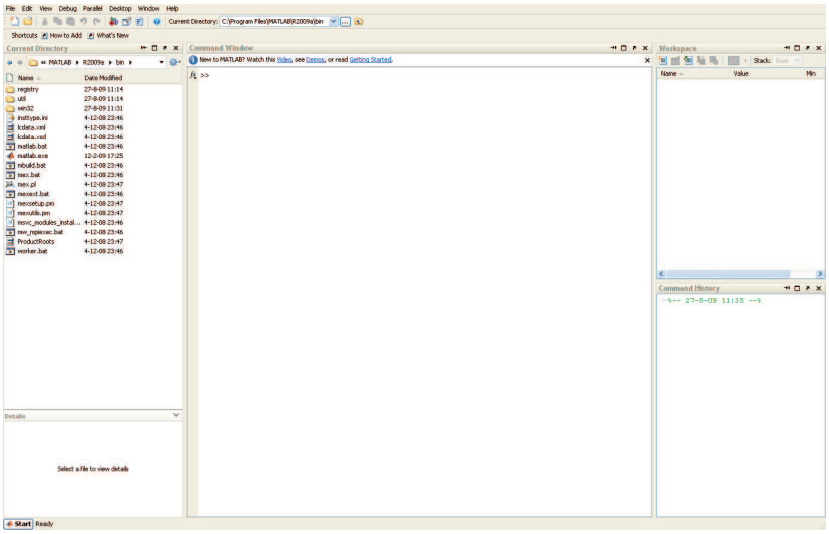
1.2 Starting and stopping

MATLAB can be started by clicking on the selecting the "MATLAB" option in your startup tree or possible on your Windows desktop. When MATLAB is started, a window similar to the one below should appear:

The main window in MATLAB is the command window, positioned at the center, in which commands can be typed after the MATLAB prompt:

```
>>
```

Commands are entered with the 'return' key. The command is then executed by MATLAB. If MATLAB is ready, after possible outputs a new prompt appears.



The window on the left is the "Current directory" window. The "Current directory" window depicts the files in the current directory. This is the first user-governed directory where MATLAB will look for files or functions. On the top right you will find the "Workspace" window, in which all declared variables are shown. The bottom right window is the "Command history", where you can find all commands you have (recently) entered. By double-clicking on such a command, MATLAB reuses the command.

Stopping MATLAB can also be done in different ways:

- By means of the command:

  ```
  >> quit
  ```

- Via the menu File\Exit MATLAB
- Clicking on the cross in the upper right of your window.

1.3 Commands in MATLAB

By clicking on the "Command Window", after the MATLAB prompt `>>`, a command can be typed. After having pushed the 'return' key, the command is executed, and possibly the result appears on the screen. A command and its result looks like this:

```
>> a = 1 + 2 + 3
a =
    6
```

The result of 1 + 2 + 3 is assigned to the variable *a*. You can now use this variable in succesive operations, such as:

```
>> b = 2*a + a/3
b =
    14
```

In MATLAB, variables are introduced by assigning a value. Note, that the value can be numerical values, matrices (called arrays), or other types. You can use the variable, and later possibly assign a new value to the variable. It is not possible to perform calculations with variables that have not been assigned a value.
A command does not need to start with an assignment of the form `variable =`. In such a case, the result is automatically assigned to the variable *ans*. You can then use *ans* further.

```
>> 4*a + 1
ans =
    25
>> ans*ans
ans =
    625
```

If you now want to know what the value of a is, the following command suffices:

```
>> a
a =
    6
```

Normally, MATLAB displays the output below the command. The output is suppressed by ending the command with a semicolon. After having entered the command $s = 1 + 2$, the screen looks as follows

```
>> s = 1 + 2;
```

The value 3 has been assigned to the variable s. If you want to know the value of this variable, you can either look in the " Workspace" window on the top left of your MATLAB window, or request the value of s as follows

```
>> s
s =
    3
```

If a command is longer than one line, you can end the line with three dots, and then press the 'return' key. You can then continue on the following line. After completing the command, the screen looks as follows:

```
>> s = 1 + 2 + ...
3 + 4
s =
    10
```

MATLAB	Standard
`a+b`	$a+b$
`a-b`	$a-b$
`a*b`	ab
`a/b`	$\frac{a}{b}$
`a^b`	a^b

Table 1.1: Standard operations

MATLAB	Standard
`sin(x)`	$\sin(x)$
`sqrt(x)`	$\sqrt{x}$
`cos(x)`	$\cos(x)$
`exp(x)`	e^x
`tan(x)`	$\tan(x)$
`log(x)`	$\ln(x)$
`asin(x)`	$\sin^{-1}(x)$
`log10(x)`	$\log_{10}(x)$
`acos(x)`	$\cos^{-1}(x)$
`abs(x)`	$\|x\|$
`atan(x)`	$\tan^{-1}(x)$
`sign(x)`	$\text{sign}(x)$
`mean(x)`	$\text{mean}(x)$
`std(x)`	standard deviation
`min(x)`	$\min(x)$
`max(x)`	$\max(x)$
`rand(x,y)`	returns $x \times y$ array of random numbers, distributed uniformly in [0,1] *
`randn(x,y)`	returns $x \times y$ array of random numbers, distributed normally with mean 0 and variance 1*
`round(x)`	Rounding to nearest integer
`floor(x)`	Rounding to smaller integer
`ceil(x)`	Rounding to larger integer

* Omitting y results in $x \times x$ array and
omitting x,y yields a single number

Table 1.2: Mathematical operations

Mathematical expressions

You cannot enter mathematical expressions literally. The Tables 1.1-1.2 make clear how mathematical expressions can be entered. The variables in these tables are to be interpreted as numbers.

Remark: The order of operations in MATLAB is the standard order: first raising to a power, then multiplication and division, and after that addition and subtraction. To deviate from this sequence, parentheses '(' and ')' need to be used to define the order of calculations, e.g. $\frac{1}{\exp(3)+1}$ is obtained by running `1/(exp(3)+1)`.

Variable	Explanation
ans	This variable contains the result of the last calculation that has not been assigned to another variable.
eps	The value of this variable is approximately $2.2204 \cdot 10^{-16}$ and is being used internally by MATLAB. This number is the default computation accuracy of MATLAB which is used to round of all numbers for storage in the computer memory. Do not confuse eps with the number e, because $e = \exp(1) = 2.7182...$
i or j	The complex number i with the property that $i^2 = -1$.
pi	3.1415...
Inf	This is the value infinity. When one divides 1 by 0, 1/0, the result will be Inf.
NaN	This variable is a representation of 'Not a Number'. A calculation with NaN always results in NaN. The command 0/0 produces NaN.

Table 1.3: Predefined variables in MATLAB

Interruption

A (long) MATLAB calculation can be interrupted with 'ctrl-c'. After this, a new prompt appears and new commands can be entered. This interruption is especially useful when a programmed loop is running that is corrupt.

Incomplete commands

If a command is incomplete or invalid and you press the 'return' key, there are two possibilities:

- An error message and a new MATLAB prompt appear. You can start over again.
- The flickering cursor is at the left of the line below the command. In this case there are two possibilities.
  - The command can be finished and after that evaluated.
  - Using 'ctrl-c', you can interrupt entering the command. A new MATLAB prompt appears.

Variables

The name of a variable in MATLAB has to start with a letter. After that, the name can consist of an arbitrary number of letters, numbers or symbols like '_' and '-'. MATLAB does distinguish between upper and lower case letters.
MATLAB uses some special variables, shown in Table 1.3.

Warning: in principle, it is possible to assign a value to the internal MATLAB variables presented in Table 1.3. This can influence your calculations. *Therefore, it is wise never to assign a value to an internal MATLAB variable.* Commands like `>>pi=10` should be avoided at all times! If by accident you have given a different value to an internal MATLAB variable, you can remove this value by using `clear` or the workspace browser.

Command	Explanation
who	Gives a list of the variables in use.
whos	Gives a list of the variables in use as well as some extra information.
clear	Removes all variables.
clear x y	Removes the variables x and y.

Table 1.4: Important commands.

On the top right of the default MATLAB window, you will find the "Workspace" window where all variables in use are listed. The browser also indicates to which class the variables belong. If necessary, you can remove variables by using the browser. The special variables above will not be listed
Some other important commands for the management of variables are given in Table 1.4.

1.4 Help facilities

In principle all information about MATLAB can be found in MATLAB's help facilities. Since the MATLAB commands and functions can often be used in more general situations, the information you will obtain through the help facilities will often be more general than necessary. You will have to filter out the essential ingredients from the information that is offered.

Below we list the most important ways to use the MATLAB help facilities.

- **Help** Functions - By typing the command `>> help 'function name'` in the Command Window, an M-help file appears in the Command Window. This file contains a short description of the function, the syntax, and other closely related help functions. This is MATLAB's most direct help facility, and you will often use this facility to quickly check how a certain function works and which notation should be used. If more extensive results are needed, try the command `doc`.

- **Look for** - By typing the command `lookfor 'topic'` in the Command Window, you can call up all help possibilities which contain the specific search word. You can use this possibility to look for a function of which you do not remember the function name exactly. In this manner, all related topics appear in the Command Window.

- **Help Browser** - You can use the 'Help Browser' to find and browse information about Mathworks products. This help browser contains different ways to obtain the correct information, like lists, a global index and a search function. You will especially use this help facility if you are looking for a function of which you do not remember the name. More information about the Help Browser can be found in the next paragraph.

- **Other help facilities** - You can use product specific help facilities, have a look at Mathworks' *demos* by typing demos in the Command Window, consult the Technical Support, look for documentation about other Mathworks products, have a look at a list or other books, and take part in a MATLAB newsgroup. More information about these topics can be found in the MATLAB Help Browser or on the site www.mathworks.com.

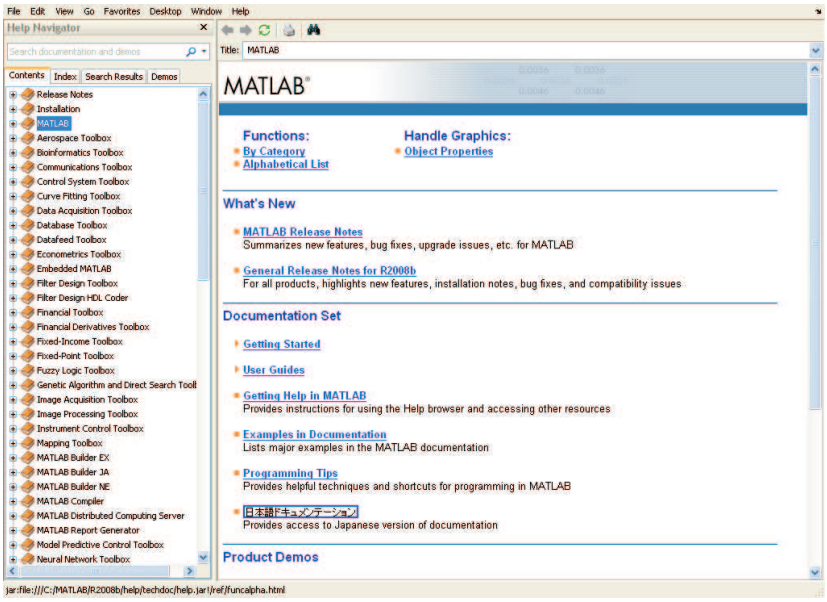
Use of the Help Browser

You can use the "Help Browser" to look for documents and to have a look at MATLAB documents and other Mathworks products. The "Help Browser" is a html viewer that is integrated in the MATLAB desktop. With this browser you can look at the HTML documents belonging to MATLAB.

You can open the "Help Browser" in three different ways:

- By clicking on the help button in the MATLAB task bar.
- By typing `helpbrowser` or `doc` in the Command Window.
- By selecting the menu Help\Product Help in the Menu bar.

When the "Help Browser" is opened, a window similar as the one below will appear:



The Help Browser consists of two parts. The left half, also called the Help Navigator, can be used to find information. In the right half you can look at the documentation. Below, the different tabs of the navigator are listed:

- Contents - Within this tab you can browse a list (arranged in a tree structure) with the contents of the documentation.
- Index - Here, you can find information by entering key words.
- Search results - Above the different tabs, you can enter a search word. Results of your search will be shown in this tab. Results found in the documentation will be listed separately from results found in the demos.
- Demos - Within this tab, you can browse through the demos that are present.

The best way to search

When you know the name of a MATLAB command you want to use, e.g. the function `plot`, the fastest way to obtain information is to type
`>> help 'function name'`, in this case `>> help plot`.This will show you some help in the command window. An alternative is to use the command `doc plot,` that will open the help browser containing a more extensive explanation, sometimes containing examples.

If you do not know the name of a function or do not know whether such a function exists, using the command `lookfor` function or the Help Browser would be appropriate.

1.5 Arrays

MATLAB uses matrices (or arrays) as the basic calculation unit. An array is a rectangular scheme of numbers, called elements, arranged in m rows and n columns. The simplest array contains only one column or one row.

An array with one row is also called a row or a row vector, and an array with one column is also called a column or a column vector. If the difference between consecutive elements in a row is always the same, the row can be formed with the command:

'variable' = 'initial value':'step size':'final value'

Here 'initial value' is the first element of the row, 'final value' is the last element of the row, and 'step size' is the difference between the consecutive elements. For example:

```
>> x = 1:-0.2:0
```

gives the row

```
x =
    1.0000    0.8000    0.6000    0.4000    0.2000         0
```

If the difference between consecutive elements of the row is +1, 'step size' can be omitted. For example,

```
>> x = 1:5
```

gives the row

```
x =
    1     2     3     4     5
```

For arrays with m rows and n columns, the numbers or elements a_{ij} are sorted in different rows and columns:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

MATLAB command	**Result**
`a(1,2)`	Gives the element on the first row and second column of a.
`a(1,:)`	Gives the first row of a.
`a(:,3)`	Gives the third column of a.
`a(:,2)=[10;10;10]`	Changes the second column of a into a column with 10's only. Only works if aa is a $n \times 3$ array, $n \geq 2$
`b(1)`	the first element of b in case b contains either one row or one column.
`a(1,[3,1])`	an array consisting of the 3^{rd} and 1^{st} elements of the first row of a.

Table 1.5: Using array indices

Such an array can be entered in different ways:

- Between '[' and ']' the elements are separated by spaces or commas, and the rows are separated by a semicolon. The commands are:

  ```
  >> a = [1 2 3;4 5 6;7 8 9]
  ```

  or

  ```
  >> a = [1,2,3;4,5,6;7,8,9]
  ```

- Between '[' and ']' where the elements are separated by spaces or commas, and each row is entered on a new line by using the 'return' key. The command now is

  ```
  >> a = [1 2 3
          4 5 6
          7 8 9]
  ```

In each of the cases above, the result is

```
a =
    1     2     3
    4     5     6
    7     8     9
```

Elements of arrays can be indicated by means of their index. In the case of row and column vectors, one index suffices. This can be used according to the Table 1.5.

Arrays (both matrices and vectors) can be concatenated into new arrays. If the arrays are arranged in a row, the number of rows in the arrays have to be equal. If the arrays are arranged in column, the number of columns in the arrays have to be equal. The concatenation of the arrays $a = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} 6 & 5 \end{bmatrix}$ proceeds as follows:

```
>> [a,b]

ans =
    2     3     4     6     5
```

Array operations

An 'array operation' is an operation (like addition or subtraction) that is applied to corresponding elements of two arrays of the same form. When an operation is applied to all elements of one array, one also speaks of an 'array operation'.

The operations addition and subtraction are automatically interpreted as array operations. After the assignments

```
>> a = [1,2,3]; b = [6,5,4];
```

addition of the arrays results in

```
>> a+b

ans =
    7     7     7
```

and subtraction of the arrays results in

```
>> a-b

ans =
    -5    -3    -1
```

Note that the arrays `a` and `b` should be of the same size.

Array operations, such as multiplication '*', division '/' and raising to a power '^', are indicated by putting a dot '.' in front of the operation sign. In these array operations, the operation is performed elementwise, e.g. the 1,1-element of `a.*b`, yields a(1,1)*b(1,1) when `a` and `b` are arrays of appropriate sizes. Hence

```
>> a.*b

ans =
    6    10    12

>> a./b

ans =
    0.1667    0.4000    0.7500

>> a.^b

ans =
    1    32    81
```

For all these operations, the following conventions hold:

1. If in an array operation one of the arrays is a number, this number is interpreted as an array of which each element equals this number, and of which the size is equal to the size of the other array.
2. If each element of an array is to be multiplied by a scalar, one does not need to put a dot in front of the multiplication sign.

Examples:

```
>> 2.^b

ans =
    64    32    16

>> [2 2 2].^b

ans =
    64    32    16

>> a./2

ans =
    0.5000    1.0000    1.5000

>> 3*a

ans =
    3     6     9
```

Detailed information about these operations can be obtained with the command `help arith`.

Producing an array of function values

Consider the function $f(x) = \frac{x^3}{1+x^2}$.
We want to assign the row vector $[f(0), f(0.2), \ldots, f(1.8), f(2)]$ to the variable y. First we make an array with the arguments [0, 0.2, $\cdots$, 1.8, 2].

```
>> x = 0:0.2:2

x =
    Columns 1 through 7

         0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000

    Columns 8 through 11

    1.4000    1.6000    1.8000    2.0000
```

For making the row of function values, we use array operations. Here, using dots is necessary, since the function $f(x)$ should be evaluated per element of x.

```
>> y = x.^3./(1+x.^2)

y =
    Columns 1 through 7

         0    0.0077    0.0552    0.1588    0.3122    0.5000    0.7082

    Columns 8 through 11

    0.9270    1.1506    1.3755    1.6000
```

In fact, this boils down to a componentwise division of a row of numerators and a row of denominators.

Remark: Multiplication of matrices will be discussed in Chapter 4.

Relational array operations

Relational array operations are operations where arrays are compared elementwise. For example, for the matrices

$$A = \begin{bmatrix} 3 & 5 & 20 \\ 11 & 9 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 3 & 11 \\ 8 & 2 & 4 \end{bmatrix},$$

the command for finding out which elements in A are larger than corresponding elements in B is given by:

```
>> A > B

ans =
    0     1     1
    1     1     0
```

Entries with a '1' (true) indicate the elements of A that are larger than corresponding elements of B. Otherwise the entry will be '0' (false). The result is thus an array with boolean entries. You can only compare the elements of array A and B if the arrays A and B have the same size, or if one of the arrays is a scalar. For example,

```
>> A > 2
```

gives:

```
ans =
    1     1     1
    1     1     0
```

Operation	Meaning
>	greater than
>=	greater than or equal to
<	lower than
<=	lower than or equal to
==	equal to
~=	not equal to

Table 1.6: Relation operators

The relational operations are given in the Table 1.6. Relational operations are often used in combination with the function `find`. Look at the result of the command `help find`.

Other array operations

In MATLAB, the mathematical standard functions `sin, cos, sqrt, atan, asin`, etc. automatically operate on arrays.

```
>> a = [1 2 3];
>> sin(a)

ans =
    0.8415    0.9093    0.1411

>> sqrt(a)

ans =
    1.0000    1.4142    1.7321
```

Consider the function $f(x) = x\sin(x)$. A row of function values can be generated with the following command:

```
>> x = 0:0.25:4*pi;
>> y = x.*sin(x)
```

The result is an array with the corresponding 51 function values.

Some other operations are

```
>> [nr,nc] = size(A)
```

The result gives the size of A, i.e., the number of rows `nr` and the number of columns `nc`. The function

```
>> sum(A)
```

calculates the column sums in a rectangular array A and gives the sums in a row vector. (A column sum is the sum of the elements of a column.) If A is a row vector, then the sum of the elements of A is calculated. The function

```
>> prod(A)
```

does the same for the product.

Transposition

In an array (matrix), the rows and columns can be interchanged. This is called transposition. The result is called the transposed array. In MATLAB, transposition can be performed with the function `transpose`. After the commands

```
>> a = [1 2 3]

a =
    1     2     3

>> A = [1 2;3 4;5 6]

A =
    1     2
    3     4
    5     6
```

transposition proceeds as follows:

```
>> transpose(a)

ans =
    1
    2
    3
```

and

```
>> transpose(A)

ans =
    1     3     5
    2     4     6
```

Transposition can also be performed by using an accent (’). The commands `transpose(A)` and `A’` give the same result for arrays containing *real valued* elements, but not for matrices having *complex* elements. Additional information about this will be provided in Chapter 4.

1.6 Graphs

To draw a graph, you first need to open a graphic screen. This is done with the command:

```
>> figure
```

Graphs are drawn with the command `plot`. The command

```
>> plot(v)
```

where v is a row vector containing the numbers v_1 until v_n, generates a plot where the points (1,v_1), (2,v_2) until (n,v_n) are connected by straight lines. The command

```
>> plot(v,w)
```

with v and w two row vectors with the same number of elements, plots the points (v_i ,w_i), and connects them by straight lines. The command

```
>> plot(v,w,x,y)
```

plots the points (v_i ,w_i) and (x_i ,y_i) in the same figure.

To determine the appearance of your graph, after plotting a vector you can add the color and linestyle of graph. For example, the command

```
>> plot(x,y,'r--')
```

will plot y versus x, with a red dashed line. Type `help plot` to see other line, marker and color options.

Suppose that we want to draw the graph of the function $f(x) = \frac{\sin(x)+2x}{1+x^2}$ on the interval $[0, 2\pi]$ consisting of 100 linearly spaced points. Note that the smoothness of the graph is of course determined by the number of point. The graph can be plotted by using the following commands:

```
>> x = 0:2*pi/99:2*pi;
>> y = (sin(x)+2*x)./(1+x.^2);
>> figure
>> plot(x,y)
```

When drawing a graph, the axes are chosen automatically. You can choose the lengths of the axes yourself with

```
>> axis([XMIN XMAX YMIN YMAX])
```

which takes care that the graph is drawn in the rectangle XMIN $\leq x \leq$ XMAX and YMIN $\leq y \leq$ YMAX.

After each new plot command, the old plot will in general disappear. If you want to draw different plots with different plot commands in the same figure, you have to give the following command after the first plot command

```
>> hold on
```

MATLAB then holds the graphs. The command

```
>> hold off
```

restores the state in which every new plot command makes the previous plot disappear.

The following commands are also important for handling the graphical screen.
You can make the graphical screen visible by either activating it, or by using the command

```
>> shg
```

The graphical screen remains the the same until either a new plot command is given or until it is erased. The command for erasing the graphical screen is

```
>> clf
```

The command

```
>> grid
```

draws a grid in the graph.

```
>> title('title')
```

puts the title indicated by the string between the quotes above the plot. In the example above, the title will be the word 'title'.

```
>> text(x,y,'string')
```

puts the text 'string' at the point (x,y) of the last plot.

```
>> xlabel('text')
```

writes information along the x-axis.

```
>> ylabel('text')
```

does the same for the y-axis.

Other graph types
You can use other plot functions in MATLAB as well. The most important are given in Figure 1.1.

- `bar(x)` produces a bar-plot of vector x.

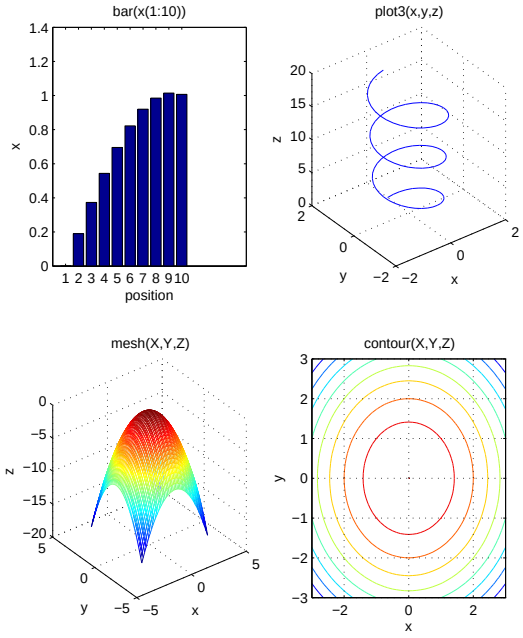


Figure 1.1: Plot functions

- `plot3(x,y,z)` produces a three-dimensional curve between the points, whose coordinates are given in the vectors x, y, and z.
- `mesh(X,Y,Z)` produces a three-dimensional surface between the grid points, whose coordinates are given in the matrices X, Y, and Z.
- `contour(X,Y,Z)` produces level curves in x, y plane, such that all points on a line have the same value for z. Since the inputs in this functions are gridpoints, the inputs X, Y, and Z should be matrices.

You can save a figure by selecting "File\Save as" and select the required file directory, filename and file type. An other option is to use the `print` command. Type `help print` for additional information. When you want to be able to open and modify your figures again in MATLAB, you should save your figure as a .fig file.

1.7 Script m-Files

If you have to execute a series of commands repeatedly, it might be better to write the commands in a so called 'script file'. The file should have a name with the extension '.m'.
As an example, let us consider the drawing of the function $f(x) = |x \sin(2\pi x)|$ on the interval [0,2] by using a 'script file': Through the menu 'File\New\M-file' the 'MATLAB Editor/Debugger' is opened. Type the following lines:

```
x = 0:0.1:2;
y = abs(x.*sin(2*pi*x));
plot(x,y)
title('f(x)= |x sin(2 pi x)| on the interval [0,2]')
xlabel('x')
ylabel('f(x)')
axis([0 2 0 2])
shg
```

Save the file through the menu 'File\Save' under the name 'picture.m'. If you now give the command `picture` in MATLAB, then the graph of $f(x)$ on the interval [0,2] will appear. An other option to save and run a file written in your editor window is to press the button  on top of it. Be careful, that this will overwrite a possible previous file.

It is better to save the commands that generate a figure than to save the figure itself.

1.8 User defined functions

Function m-files
In MATLAB you can also define your own functions. MATLAB assumes by default that all functions act on arrays. Therefore, you must keep in mind the rules for array operations when writing your own functions. You can then combine your own functions with MATLAB functions. A function definition has to be saved in a 'function file', which is a file with the extension '.m'. The name of the file has to be the same as the name of the function, and should have the extension '.m'.

Consider the function $f(x) = x^2 + e^x$. In MATLAB we will write a function with the same name. The definition of the function has to be saved in the file 'f.m'. Through the menu 'File\New\M-file' or by typing `edit` on the command line, the 'MATLAB Editor/Debugger' is opened. If you now type the lines:

```
function y = f(x)
y = x.^2 + exp(x);
```

and you save the file under the name 'f.m', then within MATLAB the function $f(x)$ is available.

Some comments on the file above are in order.

- The first line of the file has to contain the word 'function'. The variables used are local; they will not be available in your 'Workspace'.
- If x is an array, then y becomes an array of function values.
- The semicolon at the end prevents that at every function evaluation unnecessary output appears on your screen.

Always test the functions you have defined yourself!
You can change the definition of $f(x)$ by typing the command `edit f` in the command prompt.

Warning: m-files should not be given the name of existing variables or MATLAB functions. Conversely, after you have defined a function yourself, you should not give variables the same name as your function, otherwise the function will not work any more (see Exercise 1.25).

Anonymous functions
Sometimes it is more convenient to define your function at the command line. MATLAB functions produced at the command line are called *anonymous functions*. For example, consider again the function $f(x) = x^2 + e^x$. We can create the anonymous function as follows:

```
>> f = @(x)(x.^2+exp(x))
```

Here, f is the name of the function, `@` is the function handle, `x` is the input argument and `x.^2+exp(x)` is the function.

Anonymous functions can have multiple input arguments. The general structure of anonymous functions is `name = @(input arguments) (functions)`.

The anonymous function will be evaluated in the same way as the function m-files. Thus the value of $f(1.2)$ will be obtained by the command

```
>> f(1.2)

ans =

    4.7601
```

1.9 Saving and loading data

Very often, results of MATLAB calculations need to be reused elsewhere. The values of variables can be saved in a '.mat file'. This can be done with the command

```
>> save filename
```

which results in all variables in use being saved in the file with the name *'filename.mat'*. The extension '.mat' does not necessary need to be given. If only a few variables need to be saved, these variable should be specified after the file name, e.g. only the variables x and y are saved in the file filename.mat with the command

```
>> save filename x y
```

A saved '.mat' file can be read into MATLAB with the command `load`

```
>> load filename
```

where the extension '.mat' can be omitted. The saved variables and their values are stored in the workspace and can now be used again.

Besides loading data stored in '.mat' files, MATLAB is useful for processing data which is obtained from external sources, e.g., experimental measurements. Typically this data is available as a plain text file organized into columns. MATLAB can easily handle tab or space-delimited text.

There is more than one way to read data into MATLAB from a external data file. The simplest, though least flexible, procedure is to use again the `load` command to read the entire content of the file in a single step. The `load` command requires that the data in the file is organized into a rectangular array. No column titles are permitted. One useful form of the load command is:

```
>> load measurements.txt
```

where 'measurements.txt' is the name of the file containing the data. The result of this operation is that the data in 'measurements.txt' is stored in a variable called 'measurements'. Files with various extension can be used. Note, that any extension except '.mat' indicates to MATLAB that the data is stored as plain ASCII text. A '.mat' extension is reserved for a file which has stored MATLAB variables.

Suppose you had a simple ASCII file named 'meas_xy.txt' that contained two columns of numbers. The following MATLAB statements will load this data into the variable 'meas_xy', and then copy it into two vectors, x and y.

```
>> load meas_xy.txt;     %  read data into the meas_xy variable
>> x = meas_xy(:,1);     %  copy first column of my_xy into x
>> y = meas_xy(:,2);     %  and second column into y
```

After applying these commands, the data stored in column 1 of the text file is stored in the MATLAB variable x, and the data stored in column 2 of the text file is stored in the MATLAB variable y. You are now able to process this data with MATLAB.

Another option to import data in MATLAB is to select "File\Import Data". In the window opening, you can select your data file. Selecting next in this window, various import options are accessable. For example, you can choose whether MATLAB should split columns between spaces, or between commas. Furthermore, you can choose the number of header rows. On the righthand side of your window, you will see a preview how MATLAB will import the data. If you are satisfied, select finish.

Chapter 2

The numerical toolbox

2.1 Introduction

The MATLAB package itself is the numerical toolbox. We will make some general remarks and treat some important commands. Look at the help information of all commands mentioned. The number of digits of numbers that are displayed on the screen can be changed with the commands `format long` or `format short`. The calculations themselves do not change. Internally, MATLAB performs calculations with 16 decimal points.
In this chapter, we are going to investigate the function $f(x) = x^3 - x^2 - 3\arctan(x) + 1$ on the interval [-2,3]. In what follows we will assume that an function m-file of the function is available.

2.2 Graph of a function

A possible way to draw the graph of $f(x)$ is using the command `plot`. You should draw the graph using that command when the function $f(x)$ is described by an array of function values. However, when you have a function file, for instance 'f.m', which describes the specific function $f(x)$ there is an alternative way. It is based on the command `ezplot`. For example,

```
>> ezplot('f(x)',[-2 3])
```

The command `ezplot` stands for easy-to-use function plotter. It plots the function $f(x)$ over a specific domain. The expression $f(x)$ has to be put between quotes ('). The plotting interval is given as a row vector consisting of the left endpoint and the right endpoint. The advantage of using this command is that it is easy ('ez') to change the interval to investigate the graph of $f(x)$ in more detail.

2.3 Zeroes

In general it is hard to determine the zeroes of functions exactly. However, they can be approximated numerically. The function $f(x) = x^3 - x^2 - 3\arctan(x) + 1$ has three zeroes on

the whole real line.
The first zero is on the interval [-2,-1], and the signs of the function values $f(-2)$ and $f(-1)$ are different. Use the command `fzero`:

```
>> fzero(@f,[-2,-1])

ans =
    -1.2780

>> fzero(@f(x),[-2,-1])

ans =
    -1.2780
```

So it is possible to use either 'f' or 'f(x)'.

```
>> f(ans)

ans =
    -4.4409e-016
```

You can see from the last output that MATLAB has found an approximation for the zero.

Remark: When `fzero` does not find a zero of a function f, this does not mean the function does not have a zero. When the wrong interval is selected, the function is discontinuous or does not cross $f(x) = 0$, no zero can be found, although it may exist. This can be the case, for example, when searching for a zero of $f = x^2$. Note that in this case the zero of the function is identical to its minimum.

2.4 Extrema

In general also extrema cannot be determined exactly. In principle (the locations of) extrema can be approximated numerically.
From the graph, you can read the approximate location of the extrema of $f(x)$. There is a minimum in the interval [0,2] which can be determined with the command `fminbnd`:

```
>> x1 = fminbnd(@f,0,2)

x1 =
    1.0878

>> x1 = fminbnd(@f(x),0,2)

x1 =
    1.0878

>> f(x1)
```

```
ans =
    -1.3784
```

So the function $f(x)$ has a minimum at $x = 1.0878$ with value -1.3784. Again, it is possible to use 'f' or 'f(x)'.
The function $f(x)$ has a maximum on the interval [-1,0]. Unfortunately, MATLAB does not have a function that can give the location of the maximum directly. However, the location of a maximum of $f(x)$ is the same as the location of a minimum of $-f(x)$.
The following commands will now be clear.

```
>> x2 = fminbnd(@(x)-f(x),-1,0)

x2 =
    -0.5902

>> f(x2)

ans =
    2.0456
```

The function $f(x)$ has a maximum at $x = -0.5902$ a with value 2.0456.
With `fminbnd`, the location of one minimum in the indicated interval is approximated. You have to calculate the value of the minimum yourself. The precision with which the locations of extrema are calculated, can be adapted.

2.5 Numerical integration

The integral $\int_{-2}^{3}(x^3 - x^2 - 3\arctan(x) + 1)dx$ *can* be calculated exactly. Its value is $\frac{115}{12} - 9\arctan(3) + \frac{3}{2}\log(2) + 6\arctan(2) \approx 6.0245344593535$. However, integrals of the form $\int_a^b f(x)dx$ can in general not be determined exactly.

There are different commands to approximate an integral numerically. Also refer to the help facilities. We will use the command `quad` (which is an abbreviation of quadrature) and let the result appear with 15 digits.

```
>> format long
>> quad('f',-2,3)

ans =
    6.02456548407172
```

It is not possible to use 'f(x)' as the first argument. The approximation already differs from the exact value above after the fifth digit. In principle, `quad` tries to give the first four decimal points correctly. The precision of the approximation can be improved by using some extra

input arguments. Refer to the help facilities to find out the meaning of these extra input arguments.

```
>> quad('f',-2,3,[10^-7 10^-7])

ans =
    6.02453446058438
```

2.6 Polynomials

MATLAB has some commands for working with polynomials in one variable. An n-th degree polynomial in one variable has to be represented in MATLAB by means of a row vector containing the coefficients of the polynomial, where the first component of the row vector is the highest coefficient. For example, the polynomials $4z^2 + 2z - 4$, $6x^4 - 3x^2 + 5$, $s^3 - 2s$ are represented by respectively the row vectors [4 2 -4], [6 0 -3 0 5] and [1 0 -2 0].

If v is a row vector defining the coefficients of a polynomial and x is a number, then the command

```
>> polyval(v,x)
```

calculates the value of the polynomial represented by v in the point x. x can also be a vector. In this case, the values of the polynomial in the points represented by x are calculated and stored in a row vector. For example, we have

```
>> polyval([1 -2 0 1],[1 2 3 4])

ans =
    [0 1 10 33]
```

i.e., 0, 1, 10, 33 are the values of the polynomial $x^3 - 2x^2 + 1$ in the points $x = 1, 2, 3, 4$ respectively.

Very often polynomial data fitting is used to fit a polynomial function through measurement data. The data fitting is done by means of the so-called least squares method. In this method, one attempts to generate a n^{th}-degree polynomial

$$y_n(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n, \tag{2.1}$$

where the $n+1$ coefficients are calculated in such a way that the line is as close as possible to the measurement points. In MATLAB, these coefficients may be calculated with the function `polyfit(x,y,n)`. Here, x and y are arrays of measurement points and n is the degree of the polynomial: $n = 1$ is linear, $n = 2$ is quadratic, etc. The output is an array of coefficients. For example,

```
>> coeff = polyfit(y,x,1)
```

results in a linear approximation $a_0 x + a_1$ of the function $y(x)$, with coefficients:

```
>> a0 = coeff(1)
>> a1 = coeff(2).
```

To evaluate this data fitting, you can use the command `polyval` again. For example, calculate some points of the line for $x \in [0, 7]$ in steps of $\Delta x = 0.1$, i.e. $x = [0, 0.1, 0.2, \ldots 6.9, 7.0]$:

```
xp = 0:0.1:7;
yp = polyval(coeff,xp);
```

Chapter 3

The symbolic toolbox

3.1 Introduction

In this chapter we will learn how to use MATLAB to do symbolic calculus. Although MATLAB is originally a tool for numerical computations, thanks to the (Extended) Symbolic Math Toolbox, it is possible to perform symbolic computation into the numeric environment of MATLAB software. Many mathematical operations can be done. A few examples are differentiation, integration, calculation of limits, substitution of variables, simplifying algebraic expressions and solving equations.

There are a few fundamental data types in MATLAB. An overview of the most common data types is given in Table 3.1. A data type defines a set of values, and the allowable operations on those values. In the previous chapter we saw already that results of numerical computations are influenced by the precision of the numbers you are working with. In order to perform symbolic computations, MATLAB introduces a data type `sym` called a **symbolic object**. A symbolic object is a data structure that stores a string representation of the symbol. Symbolic Math Toolbox software uses symbolic objects to represent symbolic variables, expressions, and matrices. In this chapter we mainly discuss how to perform symbolic computation using symbolic objects.

Data type	**Meaning**
`int16`	16-bit integer number
`double`	Double precision number (default)
`single`	Single precision number
`logical`	Logical number,i.e. booleans
`char`	Character string
`sym`	Symbolic object

Table 3.1: Common data types

3.2 Expressions with variables

In MATLAB, you can work with variables that have been given a value. In principle, in MATLAB you get an error message when you work with a variable that has not been given a value.

```
>> clear x
>> exp(-x)+sin(x)


??? Undefined function or variable 'x'.
```

The 'value' given to a variable does not need to be a numerical value. For example, the 'value' of a variable could also be a 'string'. This is an expression between quotes ('). In fact, we have already been using this with `ezplot`, `fzero` en `fminbnd`. Consider the result of

```
>> f='exp(-x)+sin(x)'

f =
    exp(-x)+sin(x)

>> ezplot(f,[0 2*pi])
```

The variable f has now been introduced, and has been given a string as its value. However, the variable x is still unknown to MATLAB, since it has not been given a value.

The function f has a zero in the interval [3,4] and a minimum in the interval [4,5].

```
>> fzero(f,[3,4])
Zero found in the interval: [3, 4].

ans =
    3.1831

>> fminbnd(f,4,5)

ans =
    4.7213
```

These commands do not give an error message, because of the fact that by specifying the search interval we have temporarily given x a value (or rather, a range of values). Another possibility is to give a variable a symbolic 'value'. This goes as follows.

```
>> x = sym('x')

x =
    x
```

or

```
>> syms x
```

which gives the same result. You can give a symbolic value to more than one variable at the same time in the following way:

```
syms x y z
```

You can now manipulate with these variables to your heart's desire.

```
>> v = x^3 + y^2 + 1

v=
    x^3+y^2+1

>> w = sin(z)

w =
    sin(z)

>> v*w

ans =
    (x^3+y^2+1)*sin(z)
```

3.3 Substitutions

You can replace symbols you have defined by a value.

```
>> syms a b
>> y = 3*sin(a)+cos(b)

y =
    3*sin(a)+cos(b)
```

In the line below, the symbol a is replaced by the value 2.

```
>> y = subs(y,a,2)

y =
    3*sin(2)+cos(b)

>> subs(y,b,5)

ans =
    3.0116
```

Substitution of the value 2 for a and the value 5 for b can also be done at one sweep in the following way:

```
>> y = 3*sin(a)+cos(b)

y =
    3*sin(a)+cos(b)

>> subs(y,{a,b},{2,5})

ans =
    3.0116
```

3.4 Differentiation and integration

In MATLAB, you can differentiate and integrate symbolic expressions. We illustrate these commands by means of an example:

```
>> syms x y
>> y = atan(x)
```

Differentiating y with respect to x, respectively, once and three times:

```
>> diff(y,x)

ans =
    1/(1+x^2)

>> diff(y,x,3)

ans =
    8/(1+x^2)^3*x^2-2/(1+x^2)^2
```

The indefinite integral $\int \arctan(x)dx$:

```
>> int(y,x)

ans =
    x*atan(x)-1/2*log(x^2+1)
```

The definite integral $\int_2^7 \arctan(x)dx$:

```
>> int(y,x,2,7)

ans =
    7*atan(7)-1/2*log(2)-1/2*log(5)-2*atan(2)
```

Matlab	Result
`sym(sqrt(2))`	sqrt(2)
`sym(2.3654)`	11827/5000
`double(sin(2)+log(3))`	2.0079

Table 3.2: Converting numerical or symbolic expressions

This is the exact value of the integral. The numerical value is obtained by

```
>> double(ans)

ans =
    6.6367
```

The command `double` gives the numerical value of an exact number.

By far most integrals cannot be calculated explicitly. Consider the example of trying to calculate the integral of '$f(x) = e - x(1 + x^3)^{1/2}$' over the interval [1,4].

```
>> int(exp(-x)*sqrt(1+x^3),x,1,4)

ans =
    int(exp(-x)*(1+x^3)^(1/2),x = 1..4)

>> double(ans)

ans =
    1.0017
```

MATLAB returns the integral, because it cannot calculate it explicitly. The function `double` gives a numerical approximation of the integral.

3.5 Numerical values

The function `sym` converts numbers into exact expressions, while the function `double` does the reverse. Exemplary results of both expressions are given in Table 3.2. You can get an idea of the magnitude of an exact expression by using the command `double`.

3.6 Manipulation of expressions

First perform the command

```
>> syms a b c d x y
```

Matlab	Meaning of the command
`expand((a+b)^3)`	Get rid of the brackets.
`factor(a^3+3*a^2*b+3*a*b^2+b^3)`	Factorise.
`[n d] = numden(a/b+c/d)`	Reduce to a common denominator.
`simplify((x\^2+2*x+1)/(x+1))`	Simplify the expression.

Table 3.3: Manipulation of expressions

Several manipulations of symbolic expression can be performed, stated in Table 3.3. If you want to write the expression '$x/(1+x/(1+x))$' as a numerator divided by a denominator, you can proceed as follows:

```
>> syms x
>> y = x/(1+x/(1+x));
>> [n d] = numden(y)

n =
    x*(1+x)
d =
    1+2*x
```

The numerator and denominator cannot be simplified further.

3.7 Symbols, strings and numbers

We will investigate the function '$f(x) = 2xe^x - \sin(x)$' on the interval [0,π] with MATLAB. Execute the following commands.

```
>> syms x
>> y = 2*x*exp(-x)-sin(x)
```

Now the variables x en y are defined as symbols in MATLAB. You can draw the function $f(x)$ in MATLAB with the command

```
>> ezplot(y,[0,pi])
```

From the graph you can immediately see that f has a maximum, a minimum and two zeroes on the interval [0,π]. The locations of the extrema and zeroes cannot be determined exactly, but have to be approximated numerically. However, the MATLAB functions `fminbnd` and `fzero` operate on strings, i.e., expressions between quotes ('). The expressions can be expressions in the variable x. You can simply convert a symbolic expression into a string with the function `char`.

```
>> fzero(char(y),[0.5,1])
Zero found in the interval: [0.5, 1].
```

```
ans =
    0.8030

>> fminbnd(char(y),1.5,2)

ans =
    1.8409

>> fzero(char(y),[2.5,3])
Zero found in the interval: [2.5, 3].

ans =
    2.7923
```

The locations of the minimum and the zeroes have been found. Also the location of the maximum can easily be found.

```
>> fminbnd(char(-y),0,0.5)

ans =
    0.3384
```

It will often occur that you have to convert a symbolic expression into a string or vice versa. With class you can always find out what type of expression you are working with.

```
>> class(y)

ans =
    sym

>> z = char(y);
>> class(z)

ans =
    char
```

Suppose that you want to make an array of function values of f without first defining your own function. Let us say that we want to make the array [(-1), (-0.6), (-0.2), ..., (1)]. This proceeds as follows.

```
>> x = -1:0.4:1;
>> z = vectorize(y)

z =
    2.*x.*exp(-x)-sin(x)

>> eval(z)
```

```
ans =
    -4.5951   -1.6219   -0.2899    0.1288    0.0939   -0.1057
```

To check the result you might do the following

```
>> syms x
>> subs(y,x,0.6)

ans =
    0.0939
```

The interrelationship of symbols, strings and numbers may make using MATLAB difficult. You can only get a feeling for this by practising a lot. We give one more example.

```
>> syms x
>> 2.3456 + x

ans =
    1466/625+x
```

The fact that the numerical value 2.3456 has been converted into the exact fraction 1466/625, raises the suspicion that the last result has become a symbolic value, because in MATLAB itself every exact fraction is immediately converted into a numerical value. A check with `class` confirms this suspicion.

Chapter 4

Linear algebra

4.1 Introduction

MATLAB is a software package for performing matrix calculations. In this context, a matrix is nothing else but a rectangular scheme of numbers that can also be interpreted as an array. As mentioned before, the name MATLAB is an abbreviation of 'matrix laboratory'. More than any other scientific computation programs, MATLAB encourages and expects the user to make heavy use of arrays, vectors, and matrices.

As the name MATLAB suggests, the basic calculation unit is a matrix. For matrix calculations, MATLAB knows two kinds of operations:

- matrix operations, as they play a role in linear algebra,
- array operations, which simply act identically on each element of an array.

You have seen the array operations already in Chapter 1. Array operations are distinguished from the matrix operations that are common in Linear Algebra by the fact that a dot '.' is put in front of the operation sign. This chapter mainly focusses on the matrix operations following the rules of linear algebra.

4.2 Matrices

Let m,n be natural numbers. An $m \times n$ matrix is a rectangular scheme of numbers, which are arranged in m rows and n columns:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{4.1}$$

The numbers a_{ij}, where $1 \leq i \leq m, 1 \leq j \leq n$, are called elements or entries of the matrix. In this notation, i is called the row index and j the column index. In general, this matrix

is briefly referred to as A, or alternatively as a_{ij}. Essentially, MATLAB only knows one calculation unit, namely an $m \times n$ matrix. In MATLAB, a scalar is interpreted as a 1×1 matrix.

4.2.1 Entering matrices

A matrix can be entered in different ways:

1. Between [and], where the rows are separated by a semicolon, and the elements are separated by spaces or commas. For example:

   ```
   >> A = [1 2 3;4 5 6;7 8 9]
   ```
   or
   ```
   >> A = [1,2,3;4,5,6;7,8,9]
   ```

2. Between [and], where every row starts on a new line and the elements are separated by spaces or commas. For example:

   ```
   >>  A = [1  2  3
            4  5  6
            7  8  9]
   ```

3. Via external files in which the values of the variables are stored or are being calculated.

A matrix element can consist of all MATLAB elements. For example, the command:

```
>> a = [-1.3 sqrt(3) (1+2+3)*4/5]
```

You can also build a matrix out of other matrices to form a new matrix. For example,

```
C = [A;a]
```

produces matrix

$$C = \begin{bmatrix} A \\ a \end{bmatrix}. \tag{4.2}$$

When doing this, the sizes of the matrices have to be compatible!

4.2.2 Matrices with symbolic elements

The command `sym` constructs symbolic numbers, variables and objects. Read what `help sym` tells about this command. With the command `syms` multiple symbolic objects can be constructed at the same time. In this case, the names of the objects have to start with a letter. Symbolic objects can be used as matrix elements.

```
>> syms a b
>> A = [1 2 a
        3 b 4
        1 2 3]}
A =
    [ 1 2 a]
    [ 3 b 4]
    [ 1 2 3]
```

4.2.3 Entering vectors

Here, we recall from Chapter 1 a useful method to enter a specific type of vectors. A row vector is in MATLAB a matrix with one row, while a column vector is a matrix with one column. So vectors are entered in the same way as matrices. If the difference between the consecutive elements of the row is the same all the time, the row can also be entered by using

'variable' = 'initial value':'step size':'final value'

where 'initial value' is the first element of the row, 'final value' is the last element of the row, and 'step size' is the difference between the consecutive elements. For example:

```
>> x = 1:-0.2:0

x =
    1.0000    0.8000    0.6000    0.4000    0.2000         0
```

If the difference between the consecutive elements of the row equals one and the elements should be increasing, you can omit 'step size'. For example:

```
>> x = 1:5

x =
    1     2     3     4     5
```

To form a column vector, it is often easiest to first form a row vector, and then to transpose this row vector (see also Chapter 1). For example,

```
>> x = [1:5]'
```

gives the transpose of the row [1 2 3 4 5], i.e.,

```
x =
    1
    2
    3
    4
    5
```

4.2.4 Special matrices

Besides by direct input, some special matrices can also be formed directly with MATLAB commands. These matrices are:

- An $m \times n$ zero matrix, i.e., a matrix in which all elements have the value zero, is formed with the command `zeros(m,n)`.
- An $m \times n$ matrix whose elements all equal one is formed with the command `ones(m,n)`.
- An $m \times n$ matrix with random numbers, uniformly distributed between 0 and 1, is formed with the command `rand(m,n)`.
- An $n \times n$ diagonal matrix, i.e., a matrix in which only the diagonal elements (the elements with equal row and column indices) have a non-zero value, is formed with the command `diag(v)`, where v is a vector containing the n diagonal elements.
- An $m \times n$ unit matrix, i.e., a diagonal matrix for which all diagonal elements equal one, is formed with the command `eye(n)`.

If you want to give a matrix the same size as a given matrix A, then you can do this by using the command `size`. For example, the command

```
>> zeros(size(A))
```

forms a zero matrix with the same size as the matrix A. Of the above commands, only the command `diag` accepts a matrix as its argument. The result of the command `diag(A)`, where A is a matrix, is a column vector consisting of the diagonal elements of A.

4.2.5 Indices

Often it is desirable to work with parts or separate elements of matrices. Hereto, each element of a matrix can be accessed by means of indices. The command for this has the general form `>> A(m,n)`. Here A is the matrix of which the element with row index m and column index n is being specified. If A is a vector, one index suffices. For the matrix A in exercise 4.1, `>> A(3,2)` gives the value of this element, namely -2. Among others, this enables you to change one single element in the matrix. The command:

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

changes the original value -2 of the element on the third row and second column of A into the new value 3. All other elements of A remain the same. Instead of indicating one element of A, you can also indicate more elements of A at the same time. This is done by giving a vector of indices instead of one single index.

```
>> A(3,[2 4])
```

gives the elements a_{32} and a_{34} from A arranged as in A, i.e., as a 1 x 2 matrix. Hence,

the command

```
>> B=A([1 2 3],[2 4])
```

gives the 3 x 2 matrix consisting of the elements of A in the first three rows and columns two and four of A. For example, applying the previously mentioned command on a 4×4 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

results in

$$B = \begin{bmatrix} a_{12} & a_{14} \\ a_{22} & a_{24} \\ a_{32} & a_{24} \end{bmatrix}.$$

In view of the remarks from paragraph 4.2.4, this could also have been accomplished with the command

```
>> A(1:3,[2 4])
```

When only ':' is used, all rows or columns are meant. This can appear in the following forms:

- `A(:,j)` is the j^{th} column of A.
- `A(i,:)` is the i^{th} row of A.
- `A(:,:)` is the same as A.

This possibility is especially useful if you want to perform operations with whole rows or columns at once. For example, for our original matrix A, the command:

```
>> A(2,:)-7*A(1,:)
```

gives the row vector that results from subtracting seven times the first row of A from the second row of A. With the command:

```
>> A(2,:) = A(2,:)-7*A(1,:)
```

the same result is calculated and assigned to the second row of A. Thus, this command has changed the matrix A.

4.3 Matrix operations in MATLAB

With MATLAB it is possible to perform operations and commands on matrices. For this, MATLAB knows a lot of commands. A number of these operations will be/have been treated in the course Linear Algebra. At this moment, the following MATLAB operations are of importance.

Multiplication

The multiplication of two matrices is performed in a special manner. For a $m \times n$ matrix $A = a_{ij}$ and a $n \times p$ matrix $B = b_{jk}$, we define the product AB of A and B to be the $m \times p$ matrix $C = c_{ik}$, where

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \tag{4.3}$$

So,

$$\begin{bmatrix} \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \cdots & b_{1k} & \cdots \\ \cdots & b_{2k} & \cdots \\ & \vdots & \\ \cdots & b_{nk} & \cdots \end{bmatrix} \begin{bmatrix} & \vdots & \\ \cdots & c_{ik} & \cdots \\ & \vdots & \end{bmatrix} \tag{4.4}$$

The product of the matrices A en B can only be formed if the sizes of A and B are compatible: the number of columns of A has to be equal to the number of rows of B. If AB exists, BA does not need to exist. If AB as well as BA exists, we generally do not have that $AB = BA$. Within MATLAB, the matrix product of A and B is always defined if either A or B is a number. Multiplication of a matrix by a number (scalar multiplication) boils down to multiplication of all matrix elements by that number. The symbol for matrix multiplication in MATLAB is `*`, i.e.,

```
>> A*B
```

Addition and subtraction

Addition

```
>> A+B
```

and subtraction:

```
>> A-B
```

of two matrices A and B is performed by addition and subtraction of the separate elements of the matrix. In MATLAB, these operations are defined if the sizes of the matrices are the same or if one of the matrices is a scalar. In the latter case the scalar is added to or subtracted from every element of the matrix.

Raising to a power

The command

```
>> A^p
```

raises the matrix A to the p^{th} power. If p is a positive integer, A^p is calculated by repeated multiplication of A by itself. The matrix A needs to be square (i.e., the number of rows is equal to the number of columns) in order to be able to perform this operation.

Transposition

Let $A = a_{ij}$ be an $m \times n$ matrix. Then the transpose of A, denoted by A^T, is the $n \times m$ matrix $B = b_{ij}$ with $b_{ij} = a_{ji}$. So:

$$A^T = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}^T = \begin{bmatrix} a_{mn} & \cdots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \cdots & a_{11} \end{bmatrix} \tag{4.5}$$

In MATLAB, the command `transpose(A)` or `A'` (the latter only for real valued matrices) calculates the transpose of the matrix A. To give an example:

```
>> [1 2 3]'
```

forms the column matrix:

```
>> 1
   2
   3
```

If the matrix A contains complex, non-real, elements, then the command `A'` does not only reflect the matrix A with respect to the diagonal, but it also takes the complex conjugate of every element. If you only want to calculate the reflection of A with respect to the diagonal, you can do this with the command `A.'`. We illustrate this by means of two examples.

```
>> A = [2+3*i 4+5*i
        2     3]

A =
    2.0000+3.0000i 4.0000+5.0000i
    2.0000         3.0000

>> A'

ans =
    2.0000-3.0000i   2.0000
    4.0000-5.0000i   3.0000

>> A.'
```

```
ans =
   2.0000+3.0000i   2.0000
   4.0000+5.0000i   3.0000
```

You need to take this into account when dealing with symbolic matrix elements (or you have to define your symbolic elements as reals).

```
>> syms a b
>> A = [a b
        1 2]

A =
    a, b
    1, 2

>> A'

ans =
    [ conj(a),       1]
    [ conj(b),       2]

>> A.'

ans =
    [ a, 1]
    [ b, 2]
```

No division

A division operator is not defined for matrices. However, sometimes one needs to find a matrix X such that $AX = B$ for given matrices A and B, where A and B are of appropriate dimensions. This is discussed in Section 4.4.

Matrix functions in MATLAB

In this paragraph we mention some commands that have a matrix as their argument.

The command:

```
>> inv(A)
```

calculates the inverse of a square matrix A if A is invertible, i.e., it calculates a matrix B such that $AB = BA = I$.

If no inverse exists, MATLAB will give a warning the matrix is singular. Possibly, it will present a matrix, with Inf as elements, e.g.

```
>> inv([1 1;0 0])
Warning: Matrix is singular to working precision.
```

```
ans =
   Inf   Inf
   Inf   Inf
```

Sometimes, an inverse with finite elements is found by MATLAB, which may be inaccurate. In that case, MATLAB will present the following warning:

```
>> inv([1e-8 0;0 1e8])
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.000000e-016.
ans =

  1.0e+008 *
    1.0000         0
         0    0.0000
```

The command:

```
>> det(A)
```

calculates the determinant of A. A has to be a square matrix.

The command:

```
>> rank(A)
```

gives the rank of the matrix A.

The command:

```
>> eig(A)
```

gives a vector containing the eigenvalues of the matrix A. The matrix A has to be square. This command is often used in the form

```
>> [S,D] = eig(A)
```

This results in matrices S and D, so that $AS = SD$. Every column of S is an eigenvector of A, while D is a diagonal matrix with diagonal elements equal to the eigenvalues of A associated with the eigenvectors in the corresponding columns of S.

Remark: The commands above can also be used for symbolic matrices. However, you have to be aware of the fact that there are other matrix functions that cannot be used for symbolic matrices. Furthermore, there are also commands that can only be used for symbolic matrices.

4.4 Solving sets of linear equations

The most common tasks in numerical linear algebra are solving a set of linear equations of the form $Ax = b$.

Several types of linear systems

$$Ax = b \tag{4.6}$$

exist:

1. There may exist a unique vector x that solves (4.6). In that case, the system is called non-singular. An example of such a system is:

   $$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} x = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, \tag{4.7}$$

   which has a unique solution $x = \begin{pmatrix} -4 \\ 4.5 \end{pmatrix}$.

2. There may be an infinite number of possible vectors x that solve (4.6). In that case, the system is called underdetermined. For example, the system

   $$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \tag{4.8}$$

   has solutions $x = \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$, $x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and many others.

3. There may not be a vector x that solves (4.6). In that case, the system is called overdetermined. Examples of such systems are:

   $$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} x = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix} \tag{4.9}$$

   and

   $$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 3 \end{pmatrix}. \tag{4.10}$$

In MATLAB it is possible to determine the solution of a set of linear equations directly. For square or rectangular linear systems the method of first resort is the backslash operator. The solution of $Ax = b$ is implemented directly in MATLAB as:

```
>> x = A\b
```

You can find more information about this command with `help mldivide`.

Sometimes MATLAB gives one of the following warnings after the command `A\b`:

```
>>  Warning: Matrix is close to singular or badly scaled.
    Results may be inaccurate. RCOND = ....
```

or

```
>>  Warning: Matrix is singular to working precision.
```

We will not go into the exact meaning of this warning. However, if this warning appears, you should not trust the result that appears on the screen.

If none of these error messages appears, this does not mean yet that the result that appears on the screen is the 'real' solution of $Ax = b$. Namely, in case of overdetermined systems, a solution to this equation does not exist. MATLAB, however, will give the least squares 'solution'.
Therefore, it is important to check the value of x given by MATLAB by typing in `A*x`, and comparing the result to the value of b. If $Ax \neq b$, then a solution to the equations does not exist, and MATLAB has given the least squares 'solution'. If you do find $Ax = b$, you know that MATLAB has given a solution. However, there may still be more solutions. To find the general solution, you will have to use a different method, for example use the method explained below.

You can also use the command `A\b` for symbolic matrices. A warning will be shown if the solution does not exist.

4.4.1 The row reduced echelon form

The row reduced echelon form can be used to solve a system of linear equations

$$Ax = b \tag{4.11}$$

Here, the derivation of the row reduced echelon form is given for several specific systems. Afterwards, the interpretation of the row reduced echelon form of a matrix is discussed.

Nonsingular system

Consider the set of equations (4.11) with $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $b = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$. Define a new matrix $C = [A, b] = \begin{pmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{pmatrix}$. Now, (4.11) is found by solving $C(:, [1, 2])x = C(:, 3)$. We will try to update C until this matrix clearly has a solution vector x.

A solution vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ should satisfy the equations:

$$1x_1 + 2x_2 = 5, \tag{4.12}$$
$$3x_1 + 4x_2 = 6. \tag{4.13}$$

Subtracting three times the upper equation from the lower yields:

$$\begin{aligned} 1x_1 + 2x_2 &= 5, && (4.14)\\ -2x_2 &= -9. && (4.15) \end{aligned}$$

This set of equations is represented in a new matrix $C_1 = \begin{pmatrix} 1 & 2 & 5 \\ 0 & -2 & -9 \end{pmatrix}$. Dividing the lower equation by -2 yields the set of equations

$$\begin{aligned} 1x_1 + 2x_2 &= 5, && (4.16)\\ x_2 &= 4.5, && (4.17) \end{aligned}$$

represented in a matrix $C_2 = \begin{pmatrix} 1 & 2 & 5 \\ 0 & 1 & 4.5 \end{pmatrix}$. Now, one can substract twice the lower equation from the upper equation, such that:

$$\begin{aligned} x_1 &= -4, && (4.18)\\ x_2 &= 4.5, && (4.19) \end{aligned}$$

is obtained. This is represented in the matrix $C_3 = \begin{pmatrix} 1 & 0 & -4 \\ 0 & 1 & 4.5 \end{pmatrix}$. Clearly, the solution vector $x = \begin{pmatrix} -4 \\ 4.5 \end{pmatrix}$ can be obtained from both the last set of equations and the matrix C_3.

The matrix C_3 is called the row reduced echelon form of C. When this matrix has an identity matrix at the lefthand side, the righthandside is the unique solution of the system (4.11). In this case, the system is nonsingular.

Overdetermined system

The row reduced echolon form does not always contain the identity matrix. Suppose $A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$ and $b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$. In that case, we find the subsequent matrices

$$C = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 3 \end{pmatrix}, \quad (4.20)$$

$$C_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad (4.21)$$

$$C_2 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (4.22)$$

Here, C_2 is the row reduced echelon form. Note, that the last line of this matrix represents $0x_1 + 0x_2 = 1$. Of course, this equation cannot be solved, such that there exist no solution satisfying $A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$. The system is overdetermined.

Underdetermined system
Suppose $A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$ and $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$. In that case, we find the subsequent matrices

$$C = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \end{pmatrix}, \tag{4.23}$$

$$C_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.24}$$

Here, C_1 is the row reduced echelon form. Note, that the last line of this matrix represents $0x_1 + 0x_2 = 0$, which is satisfied for all x. The first line represents $x_1 + 2x_2 = 1$, which is satisfied as soon as $x_2 = 0.5 - 0.5x_1$. Therefore each x satisfying this relation is a solution of the system. The system is underdetermined.

The row reduced echelon form of C, called C_{rref}, can thus be used to see whether the system is nonsingular, overdetermined or underdetermined. If C_{rref} has an identity matrix with the size of A on the lefthand side, the system is nonsingular and has one solution. If the last row of C consists only of zeros, the matrix is underdetermined and has infinitely many solutions. If the last row of C consists of only zeros, except the last element, than the system is overdetermined and has no solutions.

In MATLAB, `rref(C)` obtains the row reduced echelon form of the matrix C.

4.4.2 Solving sets of equations with the Symbolic Toolbox

The MATLAB Symbolic Toolbox also enables you to solve equations in which the coefficients are symbolic expressions. The command with which symbolic (not necessarily linear) equations can be solved is solve. This command is used as follows:

```
>> solve('expression1','expression2','unknown1','unknown2')
```

Here *expression1* and *expression2* are equations, and *unknown1* and *unknown2* are the variables that need to be solved from these equations. For example:

```
>> syms a b c d p q
>> [x1,x2] = solve('a*x1+b*x2=p','c*x1+d*x2=q','x1','x2')

x1 =
    (-b*q+p*d)/(-b*c+d*a)
x2 =
    (-c*p+q*a)/(-b*c+d*a)}
```

More detailed information about this command can be obtained through the online help.

4.5 Numerical aspects of the use of MATLAB

When using MATLAB, you have to keep in mind that it is a numerical package. In numerical algorithms, the aspects of efficiency, rounding errors and data errors play a role. We will

briefly illustrate the last two aspects.

Remark: When calculating numerically within the Symbolic Toolbox, you will make use of the numerical Maple algorithms instead of the numerical MATLAB algorithms.

Number representations and rounding errors All values with which MATLAB calculates are represented with a finite number of digits. The computer rounds all real numbers to a number that fits the representation that is employed.

Example (standard output representation in MATLAB):
$\frac{1}{3}$ = `0.3333`
$\sqrt{(}2)$ = `1.4142`
e^{-10} = `4.5400e-005`
e^{10} = `2.2026e+004`
$\frac{1}{10}$ = `0.1000`
$\frac{1}{10^6}$= `1.0000e-006`

Internally, MATLAB calculates with a bigger precision than is represented on the screen. The fact that the output representation can give rise to unexpected results, becomes clear from exercise 4.21: Changing the output representation.

There we see that 'rounding errors' can influence the result. Use `help format` for more information about the possible representations of MATLAB output. The role played by rounding errors in calculations can often be reduced by cleverly organising the algorithms.

Data errors Other errors that can occur are 'data errors'. In exercises 4.22 and 4.23 we will illustrate that, when solving $Ax = b$, a small change in the data can have big consequences for the solution. Here the intrinsic properties of the matrix A play a role.

Chapter 5

Differential equations

5.1 Differential equations

This chapter introduces methods to analyse ordinary differential equations. In this course, no other types of differential equations are discussed. Therefore, in the following chapter, all differential equations are assumed to be ordinary, i.e. only contain derivatives with respect to time. In addition, most differential equations will be autonomous, i.e. independent on time.

The differential equations can be used to describe the behavior of a system over time. Hereto, the changes in time of a system are prescribed as a function of the current state of the system.

For example, the dynamics of a mechanical system with mass $m = 1$, spring with stiffness $k = 1$ and damping $c = 0.2$ can be described by the differential equation:

$$m\ddot{x} + c\dot{x} + kx = 0, \tag{5.1}$$

where $x(t)$ denotes the position of the mass. Here, the time derivatives $\ddot{x} = \frac{d^2x(t)}{dt^2}$ and $\dot{x} = \frac{dx(t)}{dt}$ represent the acceleration and velocity, respectively.

Another possible notation is the following form

$$\dot{q} = Aq, \tag{5.2}$$

where $A = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix}$ and the state vector $q = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$ is chosen.

Hence, since q is a 2-dimensional vector, it is possible to depict the direction field of a differential equation in the phase plane, i.e. the plane $x, \dot{x}$. Hereto, depict at certain predefined coordinates the vector $\dot{q}$. For the differential equation shown above, this yields Figure 5.1. This figure is called the direction field, which provides insight in the behavior of the system for all initial conditions inside its domain.

To obtain a trajectory satisfying the system equations, an initial condition should be specified. When an initial condition is specified, one can look for a trajectory with this initial condition

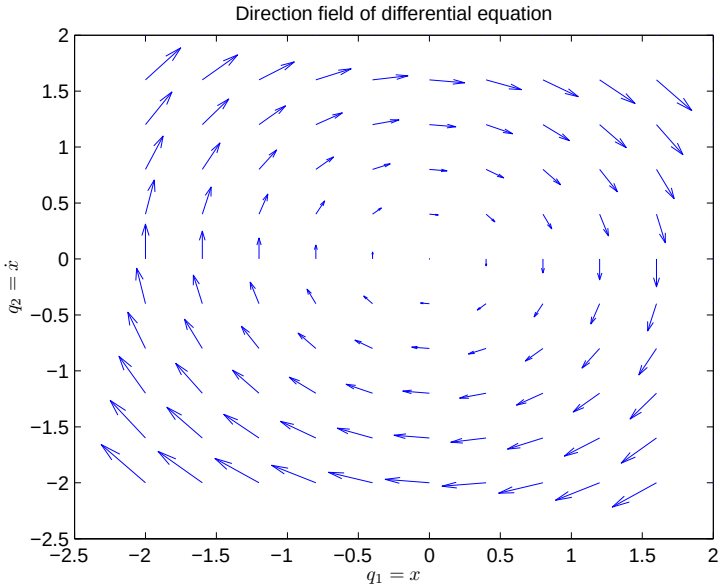


Figure 5.1: Direction field

that satisfies the differential equation. For example, in the above example, a system with initial position $x_0 = 1.5$, $\dot{x}_0 = 0$ has the solution:

$$x(t) = 1.5e^{-\frac{c}{2m}t} cos\left(\sqrt{\frac{k}{m}}\sqrt{1 - \frac{c^2}{4km}}t\right). \tag{5.3}$$

Plotting this function in time yields the trajectory depicted in Figure 5.2.

This trajectory can also be plotted in the phase plane, such that one obtains Figure 5.3.

As can be seen in this figure, the vector field is always locally tangent to the trajectories $q(t)$. Therefore, these pictures can be used to visualize many different trajectories. In our example, according to the phase plane representation, all trajectories apparently encircle the origin and converge to it. Such dynamics is expected from a weakly damped system.

5.2 Solving differential equations

In the example in the previous section, the solution of the differential equation

$$\dot{x} = f(x) \tag{5.4}$$

could be computed analytically. Although this is always possible for linear differential equations, in general no analytical solution exists. In that case, one has to use numerical solution techniques to approximate a solution.

Usually, solving a differential equation numerically requires a correct solver to obtain accurate results. However, a numerical obtained solution will give an engineer less information as an

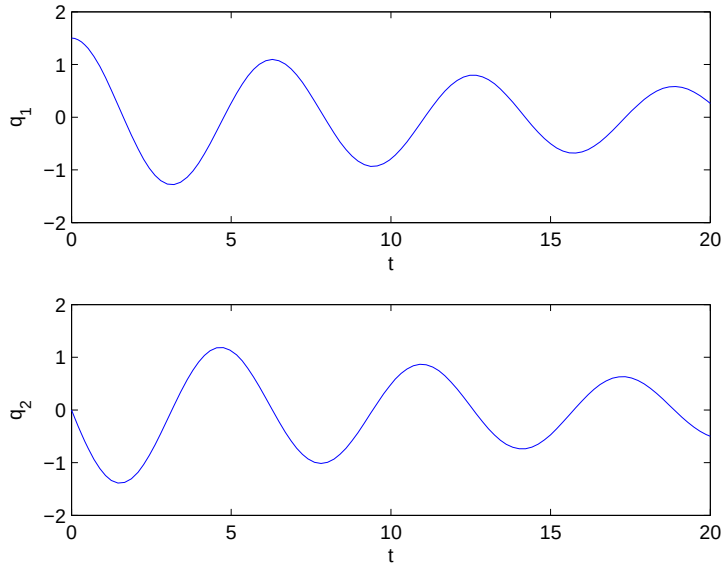


Figure 5.2: Trajectory in time

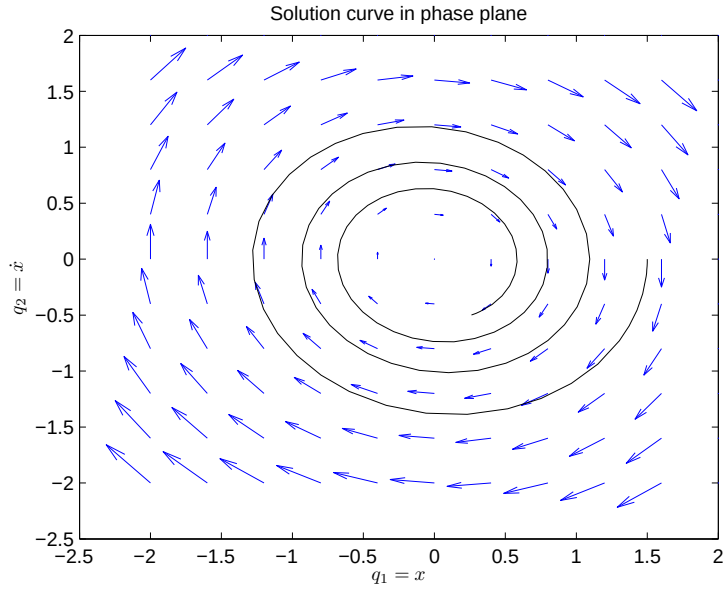


Figure 5.3: Trajectory in phase plane

analytical expression. For example, with a complete algebraic expression, it may be possible to determine the trajectory for different initial conditions or prove stability of a system. In general, a new numerical solution has to be found, when e.g. the initial conditions change.

The following section explains the use of numerical solvers, and how to interpret the results. In Section 5.3, the use of MATLAB for analytical solutions of differential equations is discussed.

5.2.1 Solving differential equations numerically

For the numerical calculation (or rather, approximation) of the solution of a differential equation, one uses algorithms that are related to the Euler algorithm. Briefly, the Euler algorithm boils down to the following. Even though one does not know the solution of the differential equation explicitly, one does know the derivative of the solution in every point (this is given by the right hand side of the differential equation), or, in other words, one knows what the direction field of the differential equation is. This direction field is now used to approximate the solution. From a given initial value, the next point of the solution is calculated by moving from the initial value in the direction of the direction field. From the point obtained in this way, one does the same again, etc. So the quality of the approximation of the solution depends on the initial value, the direction field, and the step size.

In MATLAB, differential equations can be solved numerically with the commands `ode45`, `ode23` or `ode15s`. The underlying algorithms of `ode45` and `ode23` make use of Runge-Kutta-Fehlberg integration with variable step size, i.e., the algorithm increases the step size when the solution varies less. `ode23` uses the second and third order formulas, while `ode45` uses the fourth and fifth order formulas. The routine `ode15s` uses another integration routine. This routine is specialized in problems, that the previous routines have problems to deal with: so-called stiff differential equations.

The mentioned routines can be applied to sets of first order differential equations of the form:

$$\frac{dy}{dt} = f(t, y) \tag{5.5}$$

Here y is the state vector and f the vector valued function that gives the time-derivative of the state vector as a function of t and y.

Throughout this Section, extensive use is made of function-files to facilitate numerical solving of differential equation. For an explanation of these, see Section 1.6, under "User defined functions".

The form in which `ode23` is used, is:

```
>> [t,y] = ode23('filename',[t0,t1],y0)
```

Here 'filename' is the name of the .m file in which the differential equation is defined, $[t_0, t_1]$ the time-interval over which the differential equation is solved, and $y0$ the vector with initial values. The algorithm in the program `ode23` first determines in a point the derivative y' of the function defined in the function file 'filename'. With this, the next point is determined, etc. The algorithm itself determines with which time steps it goes through the interval $[t_0, t_1]$.

Since `ode45` works with higher order formulas, less integration steps are needed with this command. As a consequence of this, `ode23` in general gives less smooth figures than `ode45`. The output of ode gives a vector t with the time instances at which the solution x has been calculated, and a row vector y with associated solution values.

Choice of solver
For most differential equations, both `ode23` and `ode45` are suited. The difference between `ode23` and `ode45` is rather subtle: with the same accuracy, `ode23` will need more time steps, though each time step is calculated faster.

However, for a certain class of differential equations, the stiff differential equations, the previously mentioned solvers will not be able to find an accurate solution, or may need excessive computation times for taking very small time steps. In that case, the problem might be of a class called "Stiff differential equations". For these differential equations, the `ode15s` is a better choice.

5.2.2 Sets of differential equations

The commands `ode23` and `ode45` can also be applied for sets of differential equations. For example, consider the set of differential equations

$$\begin{aligned} \dot{x}_1 &= 5x_1 - 2x_2 \\ \dot{x}_2 &= 7x_1 - 4x_2 \end{aligned} \tag{5.6}$$

The initial values $x_1(0) = 2$ and $x_2(0) = 8$ define exactly one solution.

With the commands `ode23` and `ode45` this type of differential equations can be tackled. To start, we write the function file 'deq.m' in the 'MATLAB Editor/Debugger', with the contents:

```
function xdot = deq(t,x)
    xdot = [5*x(1)-2*x(2);7*x(1)-4*x(2)];
```

Note, that *xdot* has to be a column vector. The command:

```
>> [t,x] = ode23('deq',[0,1],[2,8])
```

now calculates on the time interval [0,1] the values of $x_1(t)$ and $x_2(t)$ with initial conditions $x_1(0) = 2$ and $x_2(0) = 8$. The variable x becomes an array with two columns, where the first column contains the values of $x_1(t)$ and the second column contains the values of $x_2(t)$. You can plot the solutions with the command:

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

Since x consists of two columns, two graphs are drawn in the same figure. If you only enter the command:

```
>> ode23('deq',[0,1],[2,8])
```

the graphs of $x_1(t)$ and $x_2(t)$ are drawn, since by default the 'OutputFcn' `odeplot` is used. Note that you could also have written the file 'deq.m' in the following way:

```
function xdot = deq(t,x)
    A = [5,-2;7,-4];
    x = [x(1);x(2)];
    xdot = A*x;
```

5.2.3 The direction field

The solutions of differential equation (5.2) are curves in the [t, x_1, x_2]-space, which are given by [t, $x_1(t)$, $x_2(t)$]. The direction field is given by the tangent vectors to the solution curves, i.e., by the vectors [1, $\dot{x}_1(t)$, $\dot{x}_2(t)$]. Note that you can derive the direction field by means of the right hand side of the differential equation, so you do not need to know the solutions explicitly. In general the explicit dependency on time is omitted and the solutions and the direction field are only plotted in the [x_1, x_2]-plane. So, in fact, the projection on the [x_1, x_2]-plane is being drawn. This plane is also called the phase plane or state space (the state of the system at time t is given by [$x_1(t)$, $x_2(t)$]). The projected solution is also called an integral curve. In a given point [$x_1(t_0)$, $x_2(t_0)$] of the state space, the direction vector can be calculated by using the differential equations given. In equation (5.2), the direction vector for the point [$x_1(t_0)$, $x_2(t_0)$] = [1,2] is given by [$\dot{x}_1(t_0)$, $\dot{x}_2(t_0)$] = [5*1-2*2, 7*1-4*2] = [1,-1]. The direction field is obtained by determining the direction vector in every point of the state space. To draw a direction field in MATLAB, you first need to indicate the points in the [x_1, x_2]-plane where you want to draw the direction vector. The command:

```
>> [X,Y] = meshgrid(-1:0.1:1,-2:0.2:2)
```

produces a matrix X for which every row is equal to the vector -1:0.1:1 and a matrix Y for which every column is equal to the vector -2:0.2:2. By combining these matrices elementwise, you obtain grid points [x,y], where x traverses the vector -1:0.1:1 and y traverses the vector -2:0.2:2, i.e., the points [X_{ij},Y_{ij}]. The matrices X and Y can now be used to perform calculations in the grid points. For example, $U = 5X - 2Y$ and $V = 7X - 4Y$ calculates the components [U,V] of the direction field in every grid point for equation 5.2. To draw the direction field in a number of grid points, you can use the MATLAB command `quiver`. The command:

```
>> quiver(X,Y,U,V)
```

draws the vectors with components [U_{ij},V_{ij}] in the points [X_{ij},Y_{ij}]. The vectors are scaled automatically. As a consequence of this, sometimes the direction field becomes somewhat unclear. In that case, reducing the number of grid points often clarifies the direction field.

5.2.4 Plotting of integral curves

The output of `ode23` for a set of two differential equations as in equation (5.2), say t and x where t is the time vector and x the corresponding states, can also be used for drawing the phase plane. To this end, you need to plot the solutions $x_1(t)$ and $x_2(t)$ in the $[x_1, x_2]$-plane. This can be achieved by plotting the second column of x versus the first column of x after you have used the command `ode23`:

```
>> plot(x(:,1),x(:,2))
```

This gives a figure with one integral curve. If you want to draw several integral curves with different plot commands in the same figure, then you have to enter the command **hold on** after the first plot command (see Chapter 1). If you draw the integral curves and the direction field in one figure, then the direction vectors will be tangent to the integral curves. You can also draw an integral curve by assigning the property `odephas2` to 'OutputFcn' in the ode-solver. The command:

```
>> ode23('deq',[0,1],[2,8],odeset('OutputFcn','odephas2'))
```

results in a two-dimensional integral curve from the given initial condition.

5.2.5 Example of numerical solution

The Duffing equation is a model for mechanical oscillations with a nonlinear spring. The dynamics of this system, at a certain parameter is described by:

$$\ddot{x} + 0.1\dot{x} - x + x^3. \tag{5.7}$$

To use MATLAB to compute the direction field, trajectories and solutions of this system, we rewrite this system in the state space form $\dot{q} = f(t, q)$, where $q = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$. The nonlinear function $f(t, q)$ becomes

$$\dot{q} = f(t, q) = \begin{pmatrix} q_2 \\ -0.1q_2 + q_1 - q_1^3 \end{pmatrix}. \tag{5.8}$$

This equation is implemented in the MATLAB routine duff.m, containing

```
function qdot=duffing(t,q);
qdot=[q(2);
    -0.1*q(2)+q(1)-q(1)^3];
```

In the following script file, the above mentioned function file is used, to plot first the direction field, afterwards the trajectory over time, and finally the solution curves in phase plane, together with the direction field. Since $f(t, q)$ is not explicitly dependent on t, the script uses `duffing(0,q)` to compute the direction field, this is valid at all times.

```
%% script file to analyse duffing equation.

clear all;
close all;

%% create grid for direction plot
[Q1,Q2]=meshgrid(-1.5:.3:1.5,-0.7:.14:0.7);

%% create matrices with elements of vector Qd
Qd1=zeros(size(Q1));
Qd2=zeros(size(Q2));
for i=1:11;
    for j=1:11;
        Qdot=duffing(0,[Q1(i,j);Q2(i,j)]);
        Qd1(i,j)=Qdot(1);
        Qd2(i,j)=Qdot(2);
    end
end

%% plot direction plot
figure(1);
quiver(Q1,Q2,Qd1,Qd2);
xlabel('q_1');
ylabel('q_2')
title('Direction field of Duffing equation');

%% create time vector and numerical solution with ode45
[t, Q]=ode45('duffing',[0,20],[1.5;0]);

%% plot trajectory in time
figure(2)
plot(t,Q(:,1))
xlabel('t');
ylabel('q_1');

figure(3)
plot(t,Q(:,2))
xlabel('t');
ylabel('q_2');

%% plot trajectory in phase space
figure(4);
quiver(Q1,Q2,Qd1,Qd2);
xlabel('q_1=x');
ylabel('q_2=dotx')
title('Solution curve in phase plane');
hold on;
```

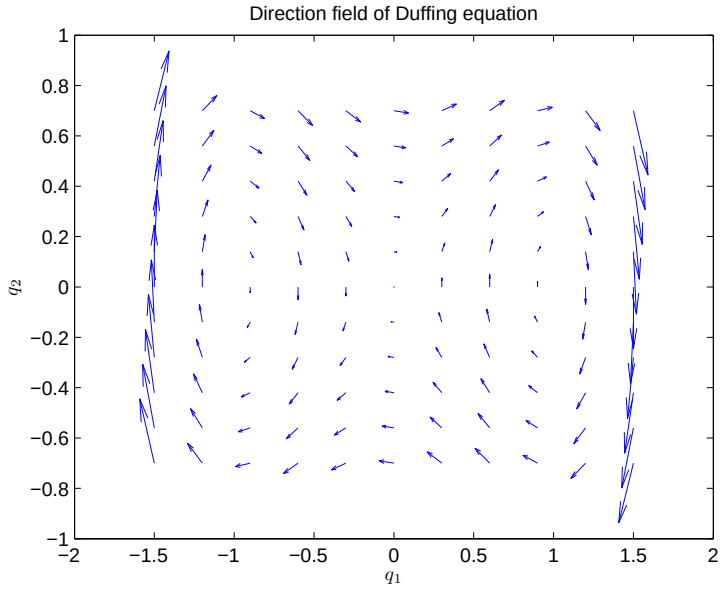


Figure 5.4: Direction field of Duffing equation

```
plot(Q(:,1),Q(:,2),'k');
```

Running these files yields the Figures 5.4-5.6.

5.3 Solving differential equations symbolically

In MATLAB you can solve differential equations in two ways: numerically (as described in the previous paragraph) and symbolically. The method to solve differential equations symbolically is described in this section. Note, that sometimes, no symbolic solution exist, or a solution exists, but cannot be found by MATLAB.

5.3.1 First order differential equations

With the command `dsolve` you can solve differential equations symbolically. For example, the general solution of the differential equation $\dot{x} = Ax$ is obtained by entering:

```
>> dsolve('Dx1=A*x1')

ans =
    exp(A*t)*C1
```

You can also add an initial condition:

```
>> dsolve('Dx1=A*x1','x1(0)=2')
```

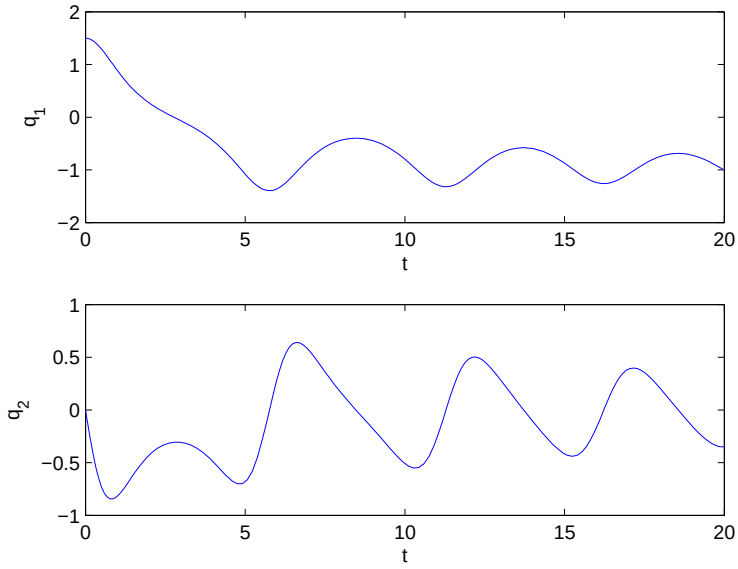


Figure 5.5: Time trajectory of Duffing equation

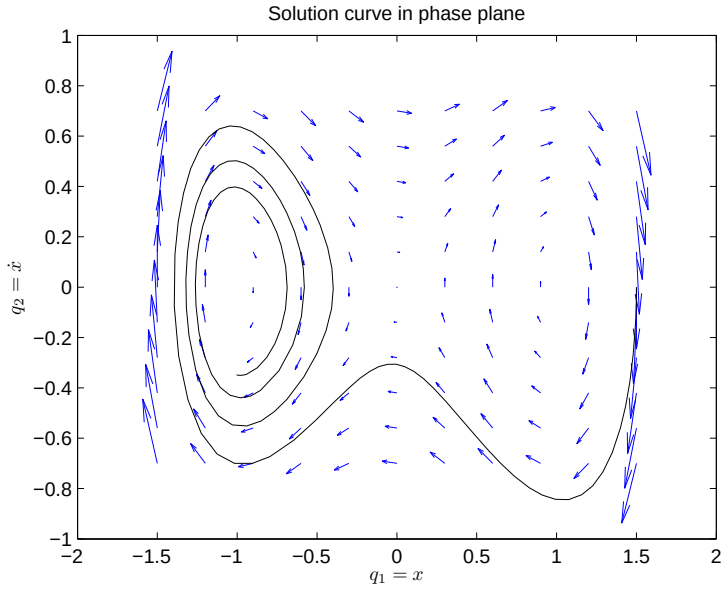


Figure 5.6: Trajectory of Duffing equation in Phase plane

```
ans =
    2*exp(A*t)
```

5.3.2 Sets of first order differential equations

You can also enter a set of differential equations. In this case you need to indicate what the output variables are. For example,

```
>> [x1,x2] = dsolve('Dx1=5*x1-2*x2','Dx2=7*x1-4*x2')

x1 =
    -2/5*C1*exp(-2*t)+7/5*C1*exp(3*t)-2/5*C2*exp(3*t)+2/5*C2*exp(-2*t)
x2 =
    7/5*C1*exp(3*t)-7/5*C1*exp(-2*t)+7/5*C2*exp(-2*t)-2/5*C2*exp(3*t)
```

gives the general solution of equation (5.2). Note that this general solution depends on arbitrary constants C_1 and C_2. If you want to specify initial conditions, you can do this in the following way:

```
>> [x1,x2] = dsolve('Dx1=5*x1-2*x2','Dx2=7*x1-4*x2','x1(0)=0','x2(0)=5')

x1 =
    2*exp(-2*t)-2*exp(3*t)
x2 =
    -2*exp(3*t)+7*exp(-2*t)
```

If you want to plot the functions x_1 and x_2, you can do this with the commands `ezplot(x1)` and `ezplot(x2)`. If you want to plot the state [x_1,x_2] in the phase plane, you first need to calculate x_1 and x_2 for a range of values of t. For example:

```
>> t = [0:0.1:1];
>> X = subs(x1,t);
>> Y = subs(x2,t);
>> plot(X,Y)
```

5.3.3 Higher order differential equations

MATLAB also enables you to solve higher order differential equations symbolically without first having to rewrite them as a set of first order differential equations. For example, the differential equation:

$$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = 0, \tag{5.9}$$

with initial conditions $y(0) = 1$, $\dot{y}(0) = 0$ can be solved with the command:

```
>> dsolve('m*D2y+c*Dy+k*y=0','y(0)=1','Dy(0)=0')
```

Chapter 6

Programming in MATLAB

6.1 Programming

This chapter introduces programming in MATLAB. The programming language (i.e., the structure of the commands and the set of commands) in MATLAB is ideally suited to learn the basic principles of programming, and after that to solve all kinds of problems. In this course, the emphasis is on learning to read existing programs, and learning to write or change programs. For this, it is important that you understand the basic principles of programming and that you are able to apply these principles yourself.

A computer program is nothing else but a series of consecutive commands that are executed by the computer. The 'order' and the number of commands does not need to be decided beforehand, but can depend upon initial values or the value of intermediate results. This is achieved by imposing a route (flow) through the commands. This makes a program as flexible as possible.

The basic components of programming are:

- variables or unknowns
- assignment statements, with which a value can be assigned to a variable
- comparison statements, with which two variables can be compared
- flow constructs like e.g. if... then ... and for... do loops, with which the order and the number of operations can be controlled
- functions and subroutines, built-in functions

These basic components will be treated extensively in this chapter.

MATLAB is a high level programming language
This means that the commands you program are very far away from the series of commands the microprocessor executes (these commands are expressed in assembly language). This

high level considerably facilitates programming, but also makes certain things impossible. A big restriction is that MATLAB programs only work within MATLAB itself. Essentially, a MATLAB program is nothing else but a series of MATLAB commands that are automatically executed consecutively. For many applications this offers a lot of advantages, like the facts that you can calculate with matrices in MATLAB and that you can use all built-in MATLAB functions. Moreover, all kinds of input and output facilities (for example the presentation of results in files or graphs) are available.

MATLAB versus other programming languages
Also in other respects, MATLAB is different from standard programming languages like C, C++ and Java. In the first place, MATLAB is a so called interpreter language, which means that MATLAB programs are read, interpreted and executed line by line. This takes a relatively long computation time, but it is very convenient for finding errors and making prototypes or test programs. Other programming languages (e.g. C, C++, Fortran and Java) are compiler languages. The program is translated into commands for the processor in a so called executable file (with extension .exe or .dll) once. After this, these files can be executed very quickly as many times as you want. However, it is more difficult to find errors in your program during the translation or execution.

Secondly, variables used in a MATLAB program do not need to be declared beforehand, and you do not need to state beforehand to which type or class a variable belongs. In MATLAB every variable is either of the type matrix (numerical) or of the type string (text), while in standard programming languages you need to declare beforehand what the type and size of every variable is. Standard types are real (numerical), integer, boolean (true or false), char (letter) etc. This makes matrix calculations in C cumbersome, while in MATLAB it goes automatically.

Another distinction is that MATLAB does not know pointers, while in most other programming languages a distinction is made between the value of a variable and its memory address. Also, MATLAB is not object oriented.

MATLAB is the ideal environment for learning to program, solve computationally simple problems and write prototype programs. If speed is really important, you can consider using another language. It is often possible as well to call pieces of compiled C code within MATLAB to improve computation speed. However, this will not be considered in this introductory course. Finally, MATLAB is not the appropriate tool for some applications, in particular large scale programs, Windows programming or Web-based programming. However, if you know the basic principles of programming, it is relatively easy to learn a new programming language.

Approach to programming in MATLAB
The reason that we want to write a computer program is that we want to solve a problem. Programming by itself does not solve the problem! We can only use a computer to (automatically) execute a certain solution strategy, saving a lot of work and time. However, we have to come up with the solution strategy ourselves and make the strategy clear to the computer! Therefore, programming is 'the translation of a technical problem into computer instructions'. A standard procedure for solving a technical problem using programming in MATLAB is the following:

1. analyse the problem and determine a solution strategy (on paper)
2. work out a formulation (on paper)
3. write an M-file in the MATLAB Editor/Debugger
4. testing and debugging
5. solve the problem

In this course, we will mainly consider points 3 and 4, and you will learn important elements and constructs from programming languages that you need to write a good program.

6.2 Some remarks about variables

First we recall some parts of Chapter 1. Variables in MATLAB are stored as arrays. You can access the contents of these arrays elementwise. A colon (:) indicates a whole row or column.

```
>> A = [1 2;3 4]

A =
    1     2
    3     4

>> A(2,1)

ans =
    3

>> A(:,2)

ans =
    2
    4

>> A(1,:)

ans =
    1     2
```

Variables in MATLAB have the following properties:

- case-sensitive (so X is not the same as x)
- maximum length of 31 characters
- start with a letter (so this is allowed: X51483 while this is not allowed: 1abcd)

- the names: `ans`, `pi`, `eps`, `inf`, `NaN`, `i`, `j`, `nargin`, `nargout`, `realmin`, `realmax` have a special meaning in MATLAB. Avoid using these names for your own variables!

You cannot give your own files an arbitrary name. MATLAB commands like `for` or `end`, but also e.g. `plot`, `prod` and `eig` can not be chosen. Sometimes you do not get a real error message, but just false results. Be careful with this, give your program a unique name.

6.3 Writing programs

In MATLAB you can write two kinds of programs. Since both kinds have the extension .m, they are called m-files. They have been briefly discussed in Chapter 1. One distinguishes the script m-file and the function m-file, whose difference has been discussed in Chapter 1. Unless stated otherwise, in the rest of this chapter script files are used.

In general, m-files are:

- not readable for others,
- not readable for yourself after about two weeks,

..... unless you have provided comments extensively (with a %-sign at the beginning of the line):

```
% these are comment line in an M-file
% and hence they are not executed as commands.
```

6.4 Programming language constructs

In this section you will be introduced to two special constructs that are often used in computer programs: the repeating construct and the conditional construct.

6.4.1 For-loop

MATLAB knows two repeating constructs. The first is the for-loop, and the second one is the while-loop.
The for-loop construct offers you the possibility to execute certain pieces of a program several times, while the value of certain variables is changing. A so-called loop variable keeps track of how often the program lines have to be repeated.

The general form of a for-loop is:

```
for loopvariable = start:step:end
    command lines
end
```

When step is omitted, a step size of 1 is used by default.

Example 6.1: a simple for-loop

```
% definition of constant n
n = 10;
% producing an array with n rows
% and 1 column, filled with zeroes
A = zeros(n,1);
% fill this column with values from 1 up to
% n in a FOR loop
for k = 1:n
    A(k,1) = k;
end
A % print the result on the screen.
```

This loop works as follows: after the for command, the loop variable k is given the value 1, and the program lines until the end are executed. Thus, A(1,1) is given the value 1. After the end command, MATLAB jumps back to the for line, and the loop variable k is given the value 2. After this, the program lines are executed again: A(2,1) is given the value 2. This is repeated until the last step, where the loop variable k is given the value 10. After this, the program continues with the lines after end, where A is printed on the screen.

`for k = 1:n` means 'for k is 1 up to n with steps of 1'. You can also use larger steps to go from 1 to n, or start with another value: for example, `for k = 2:2:10` means 'for k is 2 up to 10 with steps of 2' (i.e., 2,4,6,8,10). Of course you are free to choose other loop variables like, for instance, a and/or b.

Example 6.2

```
% initialisation of the variable p
p = 0;
% run loop n times
for k = 1:n
    p = p+1; % increase counter variable p
end
p
```

In this loop the value of p is raised by one every time the program goes through the loop. When the program goes through the loop for the first time, the new value of p becomes equal to the old value of p (which was 0) plus 1, so p becomes 1. **So this is an assignment statement and not an equation**. This kind of assignment construction is convenient to use to keep track of a counter or a summation within a loop construction.

It is also possible to nest one or more for loops. By using this, you can easily fill an $n \times m$ matrix by going through the matrix element by element:

Example 6.3: Give each element of a matrix a random value between 0 and 1

```
% definition of the size of the matrix
nr = 4;     % number of rows
nc = 3;     % number of columns
% producing an n x m matrix with nr rows
% and nc columns, which is filled with zeroes.
% it is not really necessary to make the matrix
% beforehand and fill it with zeroes, but it is
% neater, it prevents errors, and sometimes it
% may be useful.
A = zeros(nr,nc);
% fill the matrix with random values between 0 and 1
% in a nested double FOR loop
for r = 1:nr
    for c = 1:nc
        A(r,c) = rand(1);
    end
end
A
```

6.4.2 If statements

One of the most commonly used programming language constructs is the if-construct. With this construct it is possible to decide whether or not to execute certain program lines, based on a relational test of logical variables. The general form of this construct is:

```
if logical expression
    program lines
elseif logical expression
    program lines
else
    program lines
end
```

The `elseif` and `else` statements are optional, so they can also be omitted. A logical expression is either true or false. In MATLAB, the value 1 is given to a true expression, and the value 0 is given to a false expression. When evaluating logical expressions, we use relational and logical operators given in Table 6.1: Examples:

- `1 < 2` is **true**,
- `1 == 1` is **true**,
- `1 == 2` is **false**,
- `1 ~= 2` is **true**

Example 6.4: Write a MATLAB program that simulates the sign function. Use the MATLAB help to see what this function means.

Relational	
<	lower than
<=	lower than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

Logical	
&	and
|	or
~	not

Table 6.1: Relational operators

```
% make a row with time steps
t = -10:0.1:10

% determine the number of time steps in array t with the command size
n = max(size(t))

% initialise the values of f at zero
f = zeros(1,n)

% determine the function values in a FOR loop
for k = 1:n
    if t(k) < 0
        f(k) = -1;
    elseif t(k) == 0
         f(k) = 0;
    elseif t(k) > 0
        f(k) = 1;
    end
end

% plot the function
plot(t,f)
```

Check that, due to the fact that we have initialised f at 0, this if-loop can be simplified.

Example 6.5: Using a for-loop and conditional tests to calculate the inner product of two vectors.

To this end, we write a new function m-file, i.e., we create our own MATLAB command to calculate the inner product. We call our new command `inprod`. To realise this, we write a new function m-file 'inprod.m'. We will call the two vectors we want to multiply in this file a and b. We will call the output argument result.

```
function result = inprod(a,b)

% test whether the vectors a and b have the sam length
%(otherwise it is not possible to determine the inner product.)

[ra ca] = size(a); % the command size gives the size of the matrix a
                   % by assigning the number of rows of a to ra
                   % and the number of columns of a to ca
[rb cb] = size(b);

if ca~=1 % Note, that '~' represents 'not'
    error('first argument is not a vector')
end

if cb~=1
    error('second argument is not a vector')
end

if ra~=rb
    error('vectors cannot be multiplied: they do not have the same length')
end

% initialisation of result (a number)
result = 0;
for p = 1:ra
    result = result+a(p,1)*b(p,1);
end
```

6.4.3 While-loop

The second repeating construct in MATLAB is the while-loop. With the for-loop you have to specify beforehand how many times you want to execute the loop. With the while-loop this number of executions is not specified beforehand, but rather depends on whether or not a certain expression is **true** or **false**. The general form of a while-loop is:

```
while logical expressions
    command lines
end
```

Example 6.6: A 'fair' lottery. Here we use the while-loop. Write a script m-file with the name 'lottery.m' and the following contents:

```
% let MATLAB determine a random value between 1 and 10
n = round(rand(1)*10+0.5);
% initialisation of stopping variable
stop = 0;
while stop == 0
```

```
    choice = input('Enter an integer between 1 and 10')
    if choice == n
        stop = 1;
        disp('*** ***')
        disp('You have won!!!!')
        disp('*** ***')
    else
        disp('Sorry, incorrect: next player please')
    end
end
```

6.5 Creating programs

All programs that have a certain goal to achieve. For example, the goal of a script file may be, to plot the function $f(x)$ in the interval $0 \leq x \leq 10$.

Usually, some intermediate steps have to be taken to achieve the goal. In our example, the following steps are needed:

1. Distinct points x should be determined.
2. For each point x, the function values $f(x)$ should be computed.
3. The obtained function values should be plotted.
4. The figure should be given appropriate title and axes labels.

In addition, it is wise to clear all variables at the beginning of your program. Hereto, the command `clear all` can be added.

Of course, the number of steps will increase when obtaining the goal is more difficult.

When you want to write a program achieving a certain goal, it is advisable first to determine all steps that are needed. This will be done in a list of steps, that are needed. When you know your sequence of steps will achieve the goal required, you can start writing your m-file according to your designed sequence of steps.

To make your Matlab files understandable to others, and to make it easier for yourself to use them later, you should add comments (with a % character in front of it) in your file, that clarify the step you are programming in a certain part of your code.

In our example, the program design given above will yield the following m-file to plot the function $f(x) = arctan(2x^3)$ on 101 points. When a smoother plot is required, more points should be used.

```
%clear existing variables:
clear all;

% Determine distinct points x.
x=0:.1:10;
```

```
% For each point x, compute f(x) and store in an array F. Hereto,
% first create an array F:
F=zeros(length(x),1);
% compute f(x) for each x
for k=1:length(x)
    F(k)=atan(2*x(k)^3);
end

% plot F versus x
plot(x,F)
% labeling
title('function arctan(2x^3)');
xlabel('x')
ylabel('y')
```

Sometimes one has to figure out, what is the design of an existing program. In that case, from an existing m-file a sequence of steps should be derived. This may be a hard job when little or no comments are added.

Cell mode
The most important way to make an program understandable to others is the use of comments. In addition to these, one can subdivide a MATLAB program into cells. These are similar to sections in a normal texts. To start a cell, one should type `%% cellheader` in front of it, where cellheader can be replaced with a caption. For example, one can divide a program in a cell Initialisation, a cell Calculations and a cell Output Processing. When you are editing an m-file, you can run the active cell (i.e. the cell containing your cursor) by using Ctrl+Enter.

6.6 Debugging

After you have written a program, you can execute it. In most cases this is done by typing the name of the (script) m-file in the Command Window, pressing the F5 key or selecting the symbol  . Sometimes you will then get error messages, i.e. red lines in the command prompt, warnings, or something you do not expect happens.

In most cases, a MATLAB error message is clear enough to enable you to improve your program. This is especially the case with syntax errors, a forgotten `end` command or operations with matrices that do not have the correct size (e.g., you have forgotten to transpose). When you do not understand the error or if the program does run but gives unexpected results, you will have to debug the program.

The simplest way of debugging is to look at intermediate results. You can have these printed on your screen by omitting the semicolon at the ends of the corresponding lines. For example, this enables you to keep track of what happens to a certain variable in a loop.

Another, somewhat more intricate option, is to use the MATLAB Debugger. This is built into the Editor, and allows you to go through the program step by step, so that you can see where the error occurs. You can also indicate so-called break points. The program then runs from break point to break point and pauses at these break points. Type `doc debug` for more info, the link under "GUI Alternatives" may be helpful as well.

Example 6.7: Suppose you want to create an 10×8 array B, where each element b_{ij} is chosen randomly, with uniform distribution in [0,1]. Afterwards, create an array C, such that $c_{ij} = 1$ if $b_{ij} < \sqrt{\bar{B}}$, and $c_{ij} = 0$ otherwise, where $\bar{B}$ represents the mean of the squared elements of B, i.e. $\bar{B} = \dfrac{1}{mn} \displaystyle\sum_{i=1}^{m} \sum_{j=1} b_{ij}^2$.

Hereto, first the following program is written, where the first two characters of each line represent the line numbering:

```
 1 %Program to create B with random elements, and C according to the text
 2
 3 %% give size of matrices, initialise matrices C
 4 sh=8;    %horizontal size
 5 sv=10;   %vertical size
 6
 7 B=zeros(sv,sh);
 8 C=zeros(sv,sh);
 9
10 %% compute random elements of B
11 B=rand(sv,sh)
12
13 %% Show outputs of B in command window
14 B
15 Bbar=mean(B.^2)
16
17 %% Create C
18 for m=1:sv;
19     for n=1:sh
20         if B(n,m)<sqrt(Bbar);
21             C(n,m)=1;
22         else
23             C(n,m)=0;
24         end
25     end
26 end
27
28 %% Show C
29 C
```

However, running this file in MATLAB yields outputs for B and $Bbar$ and the following error message:

```
??? Attempted to access B(1,9); index out of bounds because size(B)=[10,8].

Error in ==> Untitled at 20
        if B(n,m)<sqrt(Bbar);
```

Apparently, the element $B_{1,9}$ is not defined, although MATLAB tries to read it. Since B was supposed to be a 10×8 matrix, this element $B_{1,9}$ should not be attempted to read. Thus, in the above m-file the indices n and m are interchanged in line 20,21 and 23. To repair this bug, these lines are changed to:

```
20          if B(m,n)<sqrt(Bbar);
21              C(m,n)=1;
23              C(m,n)=0;
```

Now, MATLAB indeed returns a B, $Bbar$ and C. This does not mean this answer is correct! Checking the intermediate outputs, $Bbar$ is not correct, since:

```
Bbar =
    0.2960    0.1812    0.4881    0.3324    0.3646...
```

is not a scalar, though an row vector with 8 elements. With `help mean` we find:

```
.. For matrices, MEAN(X) is a row vector containing the mean
value of each column...
```

Thus, we have to change line 15 to:

```
15 Bbar=mean(mean(B.^2)')
```

Note, that this command takes the mean of elements of the previously found vector `mean(B.^2)`. Hereto, this row vector is first transposed with `'`. Now, the m-file is working properly.

6.6.1 Structure variables

Structure variables are variables that have many fields of information. They are a very powerful and compact way to organize information in MATLAB.

For instance, an easy way to think about structures is in the context of storing the values of metal properties. Each metal has characteristic properties such as density, specific heat constant, melting point, Young's modulus, etc. These properties are listed in Table 6.2. For each of these properties an associated field in the structure variable can be created.

We can create this table of metal properties in MATLAB by typing the following at the command line:

	Metal	Density	Specific heat	Melting point	Young's modulus
1.	Steel	7.85 g/cm^3	486 J/kgK	371 K	209 GPa
2.	Gold	19.32 g/cm^3	130 J/kgK	1357 K	74,5 GPa
...	...	...	...	...	...

Table 6.2: Metal properties

```
>> metals(1).name='Steel';
>> metals(1).density=7.85;
>> metals(1).specific_heat=486;
>> metals(1).melting_point=371;
>> metals(1).youngs_modulus=209;

>> metals(2).name='Gold';
>> metals(2).density=19.32;
>> metals(2).specific_heat=130;
>> metals(2).melting_point=1357;
>> metals(2).youngs_modulus=74.5;
```

In the above example, fields were created by using the dot '.' after the main variable name. To see the field names of the structure variable, just type the variable name, or alternatively, use the fieldnames function, e.g.

```
>> fieldnames(metals)
```

To access the value of only one field, type the variable name, index, and field name, e.g.

```
>> metals(1).name
```

To see the actual content of all fields for one index of the structure (i.e., all properties of one metal), type the variable name with the appropriate index, e.g.

```
>> metals(1)
```

Chapter 7

Simulink

7.1 Introduction

Simulink is a toolbox of MATLAB that can be used for modeling, analyzing and simulating dynamical systems. Here, we focus on the use of Simulink on the use with ordinary differential equations.

Simulating the dynamical behavior of a system is important in engineering. Many systems can be written as a differential equation, such as:

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = u(t), \tag{7.1}$$

describing the movement of a single mass, on which an actuation force u is applied. For some simple differential equations, trajectories $x(t)$ of the system can be computed analytically. Many of these analytical solutions have been implemented in MATLAB and can be obtained with the command *dsolve*.

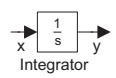
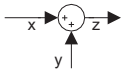
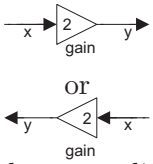
However, many systems found in engineering or physics are described by more complex differential equations, for which no analytical solution is known or even exists. Trajectories of these systems can be approximated numerically. Hereto, solvers are developed that approximate the change of the state variables over a small time step. Herewith, step by step, an approximation of the trajectory is found. For example, the MATLAB functions *ode23* and *ode45* are numerical solvers. Simulink is an other tool in MATLAB using numerical solvers.

Use of simulink has some advantages over the use of the functions *ode23* and *ode45*. Simulink has a graphical interface, such that the structure of the system can be clearly visible. Inputs, such as a discontinuous signal $u(t)$ for system (7.1) can be easily applied. Furthermore, real time applications are possible. Examples of real time applications are a control scheme to control a printer head motion, or a control scheme to control the flow of liquid in a heart simulator.

Before we start using Simulink, we first show how to represent a system such as (7.1) in a block diagram.

7.2 Creating a block diagram

Most dynamical systems can be represented in a block diagram. Thereto, a differential equation is splitted in simple blocks. Arrows connecting blocks are signals. The most used blocks are given in the following table:

Integrator		$y(t) = \int_0^t x(s)ds$
Sum		$z = x + y$
Gain		$y = 2x$

With these blocks, most differential equations can be described. This is done in four steps:

1. The highest state derivatives are identified;
2. These states are integrated once or more, to obtain all states;
3. The highest states are computed with a summation;
4. The signals entering the summation are computed by means of gains, multiplying known signals or are inputs. In block diagrams, inputs are represented by arrows pointing towards the blocks and outputs as arrows pointing out of the blocks.

In modeling differential equations, one should prefer the use of integrator blocks instead of differentiation blocks.

Remark: This constructive approach can also be used for more complex systems, for example systems in the form:

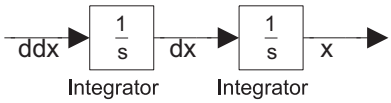
$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} f(x, y, t) \\ g(x, y, t) \end{pmatrix} \tag{7.2}$$

7.2.1 Example of a block diagram

To clarify the steps, a block diagram is constructed for the system (7.1), i.e.

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = u(t),$$

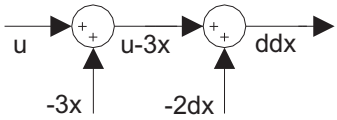
1. The highest state derivatives is $\ddot{x}$.
2. Integrating this state once, we obtain $\dot{x}$. Integration of $\dot{x}$ yields x. This is shown in the figure below. In this figure, $\ddot{x}$ is represented by ddx and $\dot{x}$ by dx.



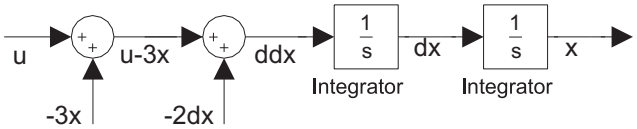
3. Substitution of $m = 1$, $b = 2$ and $k = 3$ and rewriting (7.1), yields an expression for $\ddot{x}$:

$$\ddot{x} = u - 2\dot{x} - 3x. \tag{7.3}$$

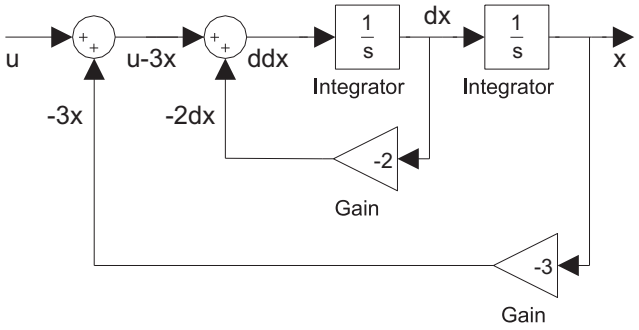
To transform (7.3) into a block diagram, we use two summation blocks:



This block is coupled to the blocks obtained in step 2:



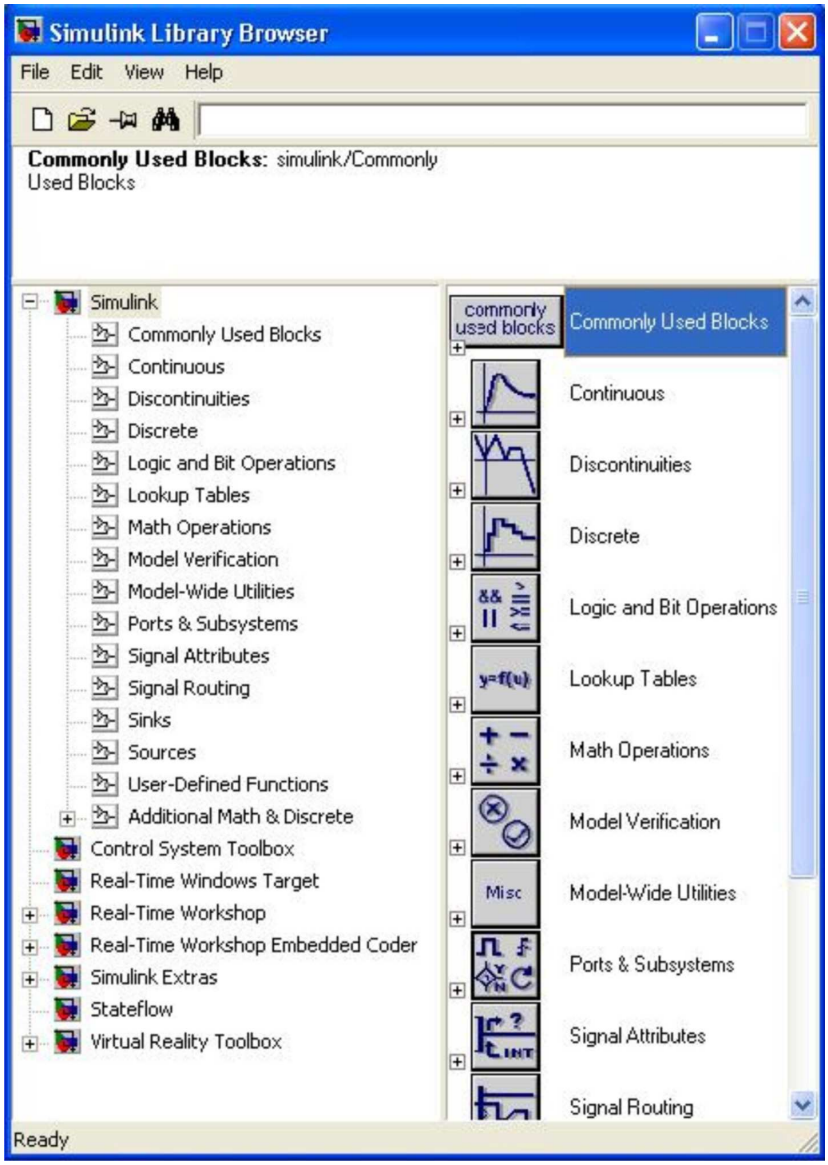
4. With the use of gain blocks, we can compute $-2x$ and $-3x$, such that a full block diagram is obtained:



Note, that we consider x to be an output and u to be an input.

7.3 Constructing a Simulink model

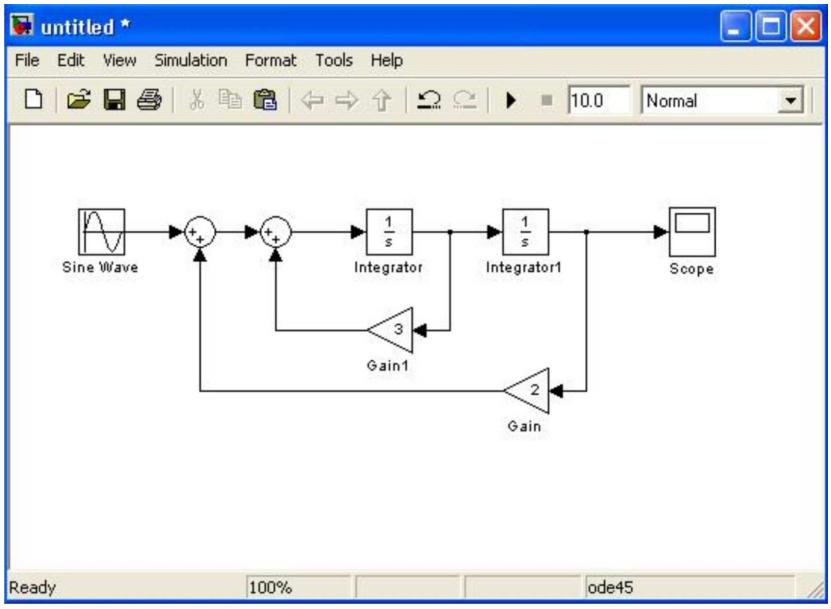
To compute trajectories with MATLAB by means of block diagrams, the tool *Simulink* should be used. To open *Simulink*, type 'simulink' in your command window or press the icon  in your MATLAB window. Now, the Simulink Library Browser should appear. A new file can be opened by selecting File>New>Model or selecting the icon on the toolbar. In the window appearing, one should build the model. Basically, you will have to draw the block diagram.



Hereto, blocks from the Library Browser should be dragged towards the model while holding your left mouse button.

In the library browser, you will find the integrator block in the category "Continuous", the gain and sum block are part of the category "Math Operations". When a block is inserted in the model, one can flip the block (press the right mouse button on block, select Format>Flip). To connect two blocks, click on the >> exiting the first block, hold your right-mouse-button and drag a line towards the >> entering the second block. You can give a name to a signal by double-clicking on the connection.

Sometimes, you will need to split a signal, for example the signal dx in the example of the previous section. Hereto, first make one of the connections, click on the >> entering the third block you want to connect, hold your left-mouse-button and drag a line towards the already existing connection.



Blocks and arrows can be moved in your model by using either the arrows on your keyboard or by dragging them with your mouse.

Many blocks in Simulink have certain parameters. To edit these parameters, double click on the block and change the value. The particular gain used is a parameter of the gain block. The initial value of a signal exiting an integrator block is a parameter of this block. In the parameters of the add-block, one can make the block subtract signals instead of add them, by changing the "list of signs" from "$|++$" to "$|+-$".

The integrator block has its initial condition as parameter. Here, one specifies the initial output of the integrator block. By changing these initial conditions, a Simulink model can be used to compute trajectories from predefined initial conditions.

Inputs in Simulink have to be defined as a function of time, for example $u(t) = 3\sin(2t)$. You will find many possible inputs in the Sources-category of your Library browser. Most used are the sine wave, step or constant blocks. A sine wave block has the properties amplitude, frequency and phase, the latter describing the initial phase of the signal. A step block has parameters initial time, initial value and final value. A constant source has the constant value as a parameter.

To handle outputs in Simulink, one needs to use the blocks in the category Sinks in the Library browser. Most used are the "Scope", "To File", and "To Workspace blocks". A scope will visualize the signals entering it. When you have finished a simulation, double-click on the scope to see the results. The "To File" block saves the entering signal to a .mat file. When you select the block properties, you can change the file name, and the name of the stored variable. The block "To Workspace" saves the signals to the workspace, such that one can use them later in the command window. In this block's properties, one can change the variable name, and choose whether the data should be saved as structure, structure with time or array.

It is allowed to call variables that are present in your workspace in block parameters. For

example, a gain-block with gain "k" will work properly, when a scalar "k" is available in the MATLAB workspace.

To save your model, press the "save" button in the toolbar of your model window. Simulink models will be saved with the extension .mdl. Later, you can open these models by selecting the file saved.

Please note, that the Library browser contains many more blocks as mentioned in this section. When you double click on a certain block in the library browser, you will see the purpose and parameters of the block.

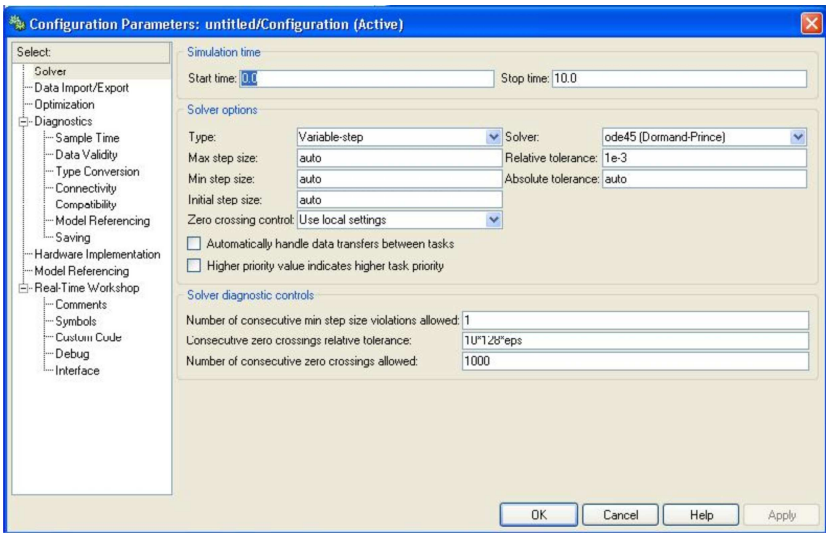
7.4 Running a simulation

To run a simulation of your Simulink model, you can select the button "Start Simulation" on the toolbar of the model. Take care, that the correct initial conditions are used for the integrator blocks. The simulation will finish when the maximum simulation time is reached. If this takes to long, you can terminate your simulation in the meantime by pressing the "Stop Simulation" button on the toolbar.

When the simulation is stopped, you can double-click on a scope to see the signal over time. Signals send to "To Workspace" or "To file" blocks are created in your workspace, or in the file in the current directory, respectively.

Simulation parameters

To edit the parameters used by Simulink to simulate the behavior of the system, click on Simulation>>Configuration Parameters. A screen similar to the one below will open:



Simulation time

In this window, you will be able to set a start and stop time for your simulation. Note, that these times represent the simulated time, which usually differs from the computation time.

Type of solver Many solver options can be changed. The most important one is the solver type. The solver type determines the time steps Simulink is taking. Though very small time steps are usually more accurate, they require longer computation time.

One can choose Variable-Step or Fixed-Step solvers. In a fixed step solver, the solver takes fixed time steps and computes at each time step, how the states should be updated. The most important option for this solver is the fixed-step size.

More complex solvers are variable-step solvers. They try to take their step size as large as possible, but will reduce the step size when the computation becomes less accurate. Usually, variable-step solvers are faster. By default, one should select the ode45 solver. However, for certain problems, ode45 will need very long computation times. In that case, one can choose a different solver, such as ode15s.

Important parameters that should be chosen for a variable step solver are the Relative Tolerance and the Absolute Tolerance. The relative tolerance measures the error relative to the size of each state. The relative tolerance represents a percentage of the state's value. The default, 1e-3, means that the computed state will be accurate to within 0.1%. The Absolute Tolerance is a threshold error value. This tolerance represents the acceptable error as the value of the measured state approaches zero. A variable step solver chooses its time steps such, that both absolute and relative tolerance are satisfied.

7.5 Example: Neuron model

A model of a single neuron cell is given by the differential equation:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} -ax^3 + bx^2 + \varphi_1 x + \varphi_2 + g_y y - g_z z + \alpha \mathrm{Inp} \\ -c - dx^2 - \varphi_3 x - \beta y \\ r[s(x + x_0) - z] \end{pmatrix}. \tag{7.4}$$

In this model, the state $x(t)$ represents the electric potential the neuron produces. Parameters $a, b, c, d, r, s, x_0, g_y, g_z, \alpha, \beta, \varphi_1, \varphi_2$ and φ_3 are constants. Inp is the input of the model.

Suppose we want to simulate this system for 1000 ms with initial conditions $x_0 = 0$, $y = 0$ and $z = 0$. Hereto, a Simulink scheme is created. To compute the signals x^2 and x^3, a new block is used: Fcn. In the Simulink library, this block is found under "User-Defined Functions". In the parameters of this block, one can enter the expression this function calculates, where u is the input of the block.

In addition, a somewhat different appearance is chosen for the Sum block. This block has the option "Icon shape". The default value "round" is used in the previous sections. Now, we use "Rectangular". To add or substract more signals, the option "List of signs" can be changed. For example, adding the first two signals and substracting a third one is achieved by the list of signs "++-".

The neuron model is implemented in Simulink scheme in Figure 7.1:

When simulating this model, the variables a, b, c, d, r, s, x_0, gy, gz, $Alpha$, $Beta$, $Phi1$, $Phi2$, $Phi3$ should be defined in Matlab his Workspace. For certain parameters, the x-trajectory depicted in Figure 7.2 is obtained for the constant input Inp = 3.3:

Since this model is *stiff*, the solver type ode15s is used.

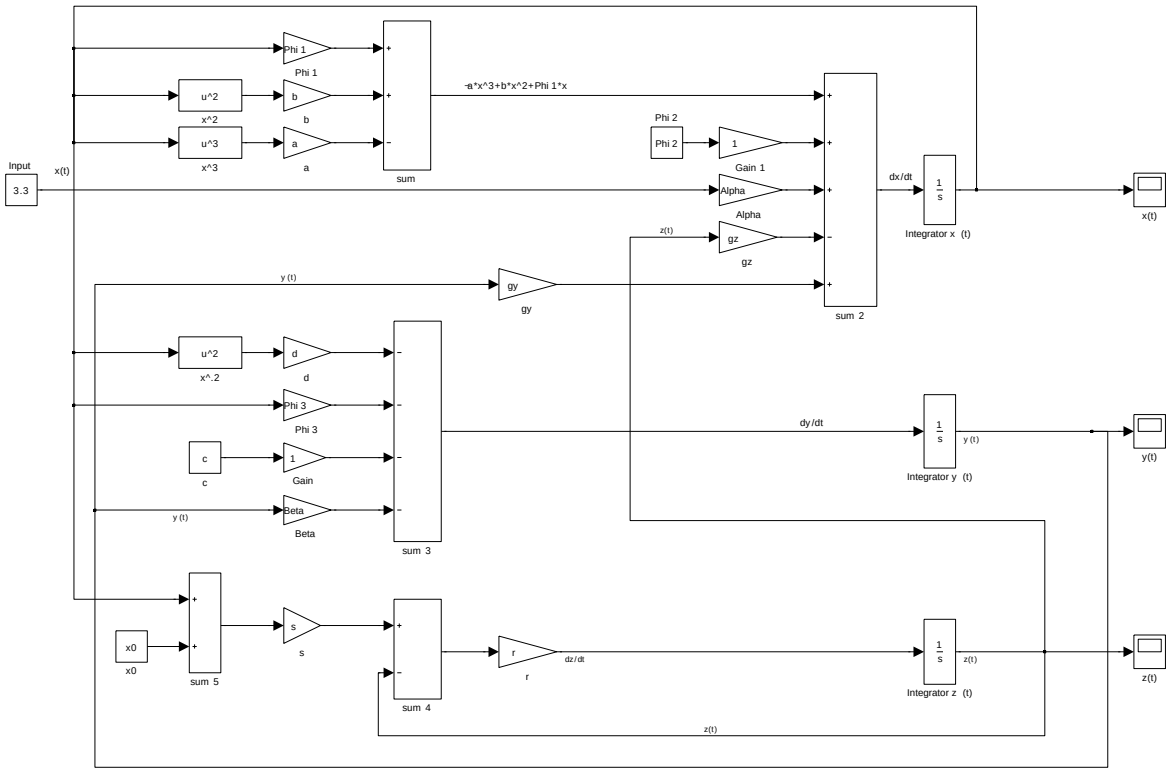


Figure 7.1: Simulink scheme of neuron model

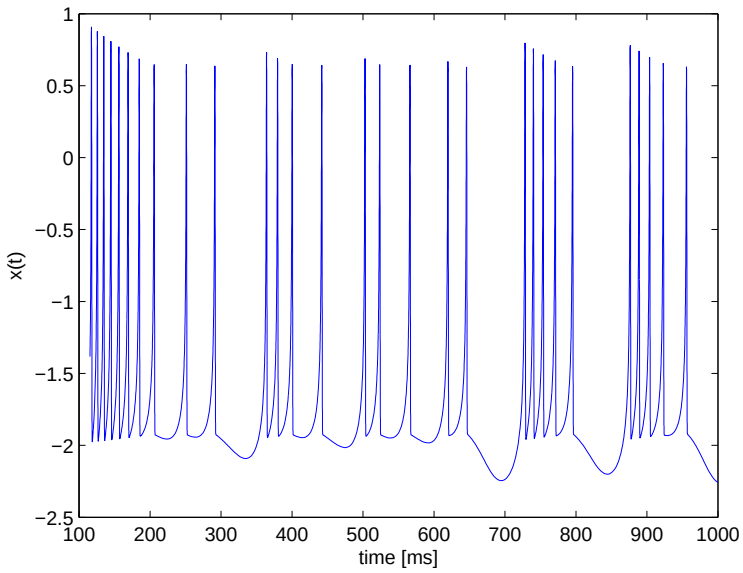


Figure 7.2: Time trajectory of neuron model

Appendix A

Exercises

A.1 Exercises chapter 1

Exercise 1.1: Entering commands

You are supposed to perform the following exercises in the specified order. Before entering each of the commands, think about the output on the screen and the result of each exercise.

a) `>> a = 1+2+3+4+5+6`

b) `>> b = 1+2+3+4+5+6;`

c) `>> b+3`

d) `>> 1+2+3+4+5+6`

e) `>> c = 10*ans`

Exercise 1.2: Interruption of calculations or output

Enter the array 0:900000 and make sure the output appears in the MATLAB Command Window.
Perform the exercise again and and use the key combination 'ctrl-c' quickly. What is the effect?

Exercise 1.3: Incomplete commands

By entering $a = [2, 3, 4]$ the array $(2, 3, 4)$ is specified to the variable a.

a) Enter `>> a = [2,3,4` and use the 'return' key. Finish the exercise.

b) Enter `>> a = [2,3,4` and use the 'return' key. Interrupt the input of the exercise.

Exercise 1.4: Removal of variables

Define the variables $u = [2\ 3\ 4]$ and $v = [1\ 5]$ and then delete them. Check if the variables are actually deleted.

Exercise 1.5: Saving data files

Perform the following two exercises:

```
>> f = 2
```

and

```
>> g = 3 + f
```

Save the variables f and g in the file 'try.mat'.
Close MATLAB and start it again. Load the file 'try.mat' and see if the variables f and g are known. Where is the file 'try.mat' placed?
Delete the file 'try.mat'.

Exercise 1.6: Calculation of mathematical expressions

Calculate the following expressions in MATLAB:

a) $\frac{3+2^2}{16+5^2}$

b) $4^{2/3}$

c) $\sin(\frac{\pi}{4})$

d) $\log(e)$

Exercise 1.7: Calculation of mathematical expressions with one variable

Give the variable x the value 2. In MATLAB enter the following expressions and calculate them.

a) $\frac{x^3}{6}$

b) e^{1+x^2}

c) $\frac{x}{\sqrt{1+x^2}}$

d) $x^3\sin(x^2)$

e) $2^{1/3}$

f) $\frac{\arctan(x)}{1+x^2}$

Exercise 1.8: Making a row vector

Make a row vector with:

- Initial value: -1
- Final value: 9
- Step size: 0.5

Exercise 1.9: Making a row vector

Make a row vector with:

- Initial value: 9
- Final value: 0
- Step size: -1

Exercise 1.10: Error messages with array operations

Consider the function: $f(x) = \frac{x^3}{1+x^2}$

We want to declare the row vector $[f(0), f(0.2), \ldots, f(1.8), f(2)]$ to the variable f. To perform the exercises below you first have to enter the array x with arguments $(0, 0.2, \ldots, 1.8, 2.0)$. Thus; `>> x = 0:0.2:2`

a) Perform the exercise: `>> f = x.^3/(1+x.^2)`.
MATLAB gives a result! But why is it not an array with function values?

REMARK:
IF YOU GET A RESULT, ALWAYS CHECK THAT RESULT!

b) Perform the exercise: `>> f = x^3./(1+x.^2)`.
What does the MATLAB error mean?

Exercise 1.11: Array operations

Make for the following functions f the array $[f(0), f(0.1), \ldots, f(2)]$.

a) $f(x) = x^2 + 2x + 1$

b) $f(x) = \frac{3^x}{1+3^x}$

c) $f(x) = \frac{x}{1+\sqrt{x}}$

Exercise 1.12: Relational array operations

Perform the exercise: `>> a = rand(5,5)`. This command generates an array with 5 rows and 5 columns where each element is chosen arbitrary between 0 and 1. Make an array with all elements of a which are smaller than 0.3.

HINT: use the MATLAB-function `find` in this exercise.

Exercise 1.13: Array operations

Consider the function:

$f(t) = \frac{t}{1+\sqrt{t}}$

Make the row vector $[f(0), f(0.1), \ldots, f(1)]$ by using array operations and the function `sqrt`.

Exercise 1.14: Array operations

Give an one line command that, for an array a, determines the number of elements in a which are greater than 5.

Test your command with the following arrays:

a) `a = [1 2 3; 4 5 6; 7 8 9]`

b) `a = 1:10`

c) `a = 10*rand(6,6)`

PAY ATTENTION: The same command must work in all three cases!

Exercise 1.15: Array operations

Give an one line command that, for an array a, calculates the mean value of the elements of a.

Test you command with the following arrays:

a) `a = [1 2 3; 4 5 6; 7 8 9]`

b) `a = 1:10`

c) `a = 10*rand(6,6)`

PAY ATTENTION: The same command must work in all three cases!

Exercise 1.16: Drawing a graph

Plot the functions $\sin(t)$ and $\cos(t)$ on the interval $[0, 2\pi]$ in one figure.

Exercise 1.17: Drawing a graph

Plot the function f below on the interval $[0, \pi]$. Add names to the axes in the figure.

$f(t) = \sqrt(t)\sin(2t)$

Exercise 1.18: Drawing a graph

Plot the function f below on the interval $[-2, 3]$.

$f(t) = e^{-t^2}$

Exercise 1.19: Script file

Why can one better save the commands to plot a figure than to save the figure itself?

Exercise 1.20: Script file

Open via the menu "File\New\M-file" the "MATLAB Editor/Debugger".
Save the lines below under the name 'plate.m' and describe the meaning of the lines in this file.

```
x = 0:0.1:2
y = abs(x.*sin(2*pi*x))
figure
plot(x,y)
title('f(x) = |x*sin(2*pi*x)| on the interval [0,2]')
axis([0 2 0 2])
shg
```

Exercise 1.21: User defined function

Open the "MATLAB Editor/Debugger" via the menu "File\New\M-file" and enter the following function f:

```
function y=f(x)
y = x.^2+exp(x);
```

Save this function under the name 'f.m'.
Where is the file 'f.m' placed?

Exercise 1.22: User defined function

Enter the following function f:

```
function y=f(x)
y = x.^2+exp(x);
```

and calculate some values.

In the definition of the function f, with respect to the previous exercise, the semicolon (;) is left out. What is the effect?

Exercise 1.23: User defined function

Consider the function:

$$g(x) = \begin{cases} 0, & x \leq 0 \\ x, & x \geq 0 \end{cases} \tag{A.1}$$

Make your own function g in MATLAB. For this, use the MATLAB functions `max` and `zeros`.

Check your own function with the commands:

```
>> x = -1:0.2:1;
>> g(x)
```

WARNING: The names of the function and the variables may NOT BE the SAME!

Exercise 1.24: User defined function

Make your own function with the name h for the function $h(x) = x^2$ that is capable for row-operations.

Perform the following exercises in the correct order:

a) `>> h([1,2,3,4])`

b) `>> h = 2.5`

c) `>> h([1,2,3,4])`

d) `>> h(1)`

e) `>> h(1.4)`

f) `>> clear h`

Questions:

- Is h, during the parts c) till e), be seen as a function or a variable?

- Why does part d) not give an error message?

- Does MATLAB still know the function h?

Exercise 1.25: Help command

Ask for information about the sine function and the function `load`. For this, use the command `>> help`.

Exercise 1.27: Help browser

Search for information about the sine function by means of the MATLAB Help Browser.

HINT: You can call the MATLAB Help Browser by clicking on the yellow question mark in the MATLAB menu bar.

Exercise 1.28

Make a row vector that consists of the fifth power of the first 40 natural numbers.

Exercise 1.29

In MATLAB calculate the product $1 \cdot 2 \cdots 24 \cdot 25$.

Exercise 1.30

In MATLAB calculate the exact sum of:

$1 + 2^2 + 3^2 + 4^2 + \cdots + 199^2 + 200^2$

Exercise 1.31

Draw the graph of the function $f(x)$:

$f(x) = \frac{x}{1+x}$

on the interval $[0, 0.01, \ldots, 3.99, 4]$. Create an anonymous function!

Exercise 1.32

Draw a circle with radius 1 and center (0,0) in a square with $-2 < x < 2$ and $-2 < y < 2$. Make sure that the circle looks round!

HINTS:
First make a list of x-coordinates and a list of y-coordinates. Draw the circle. Change the shape of the figure by using the command `>> axis` in different combinations.

Exercise 1.33

The periodic function f with period 2 is given by:

$$f(t) = \begin{cases} t^2 & , 0 \leq t \leq 2 \\ f(t-2) & , 2 \leq t \\ f(t+2) & , t < 0 \end{cases} \tag{A.2}$$

Define this function in MATLAB. Make use of the command `>> mod`. For a number r the command `>> mod(r,2)` returns a number s in such a way that $0 < s < 2$ and s and r differ are a multiple of 2 from each other. Thus $f(r) = f(s) = s^2$.

Make sure that the function is also capable to work with arrays.

A.2 Exercises chapter 2

Exercise 2.1: User defined function

Make your own function f with $f(x) = x^3 - x^2 - 3\arctan(x) + 1$.

Exercise 2.2: Numerical approximation of a zero

Approximate the zero of the function:

$f(x) = x^3 - x^2 - 3\arctan(x) + 1$

using the command:
`>> fzero('f(x)',[-2,-1])`.

Show that the approximation of the zero introduces an maximum error of $1 \cdot 10^{-4}$.

Exercise 2.3: Numerical approximation of a zero

Given is the same function as in the preceding exercise:

$f(x) = x^3 - x^2 - 3\arctan(x) + 1$.

Now, determine all zeroes of this function numerically.

Exercise 2.4: Drawing a function

Given the function f:

$f(x) = x^3 - x^2 - 3\arctan(x) + 1$.

Draw the graph of the function $-f(x)$ on the interval [-2,3]. Do not change the file 'f.m'!

Exercise 2.5: Numerical determination of a minimum

Given the function f:

$f(x) = x^3 - x^2 - 3\arctan(x) + 1$.

Make sure that the numbers appear on the screen with 15 digits.

a) Enter the command

```
>> fminbnd('f(x)',0,2,optimset('TolX',10^n))
```

four times. Each time, replace the exponent n with -4, -6, -8 and -10 respectively.

b) Examine which digits in the different solutions remain the same.

Exercise 2.6: Numerical determination of a zero

The function $f(x) = (x^3 - 2)^2$ has exactly one zero.

a) Determine this zero numerically.

b) Compare the value of the zero you solved numerically with the exact zero.

HINT: Draw the graph of the function f on the interval [0,2].

Exercise 2.7: Numerical determination of zeroes and extrema

Determine the zeroes and extrema of the function $f(x) = e^{2x} - 4e^x + 2$.

Hereby, choose a suitable interval to draw the function f with MATLAB. A suitable interval should give a clear view all zeroes and extrema of the function.

Exercise 2.8: Numerical integration

Determine the following integral numerically using the command `quad`:

$$\int_1^3 (x^3 + x^2 + \frac{1}{x^2})dx \tag{A.3}$$

Evaluate the integral for different levels of accuracy. Compare your results with the exact value of the integral: 88/3.

When you are done, make sure that MATLAB shows 5 digits on the screen again.

Exercise 2.9

Consider the function:

$$f(x) = x^2 e^{(-x)}. \tag{A.4}$$

Answer questions a,b and c without using the derivative f'.

a) Why does the function $f(x)$ has a minimum at $x = 0$?

b) Why does the function $f(x)$ has a maximum on the interval [0,∞]?

c) Determine the maximum on the interval [0,∞] with the command `fminbnd`.

d) What is the error between the result found in question c and the real value of the maximum?

Exercise 2.10

Consider the function:

$$f(x) = x \sin^2(x). \tag{A.5}$$

a) Draw the graph of the function $f(x)$ on the interval [0,2π].

b) Determine the position and the value of the maxima of the function $f(x)$ numerically.

c) Numerically, determine the following integral:

$$\int_0^{2\pi} f(x)dx \tag{A.6}$$

Exercise 2.11

Compute the values of

$$y(x) = x^5 - x^2 + 8x + 10 \tag{A.7}$$

for $x = -3, -1.2, 1, 1.1$. Make use of *polyval*.

Exercise 2.12

Compute $y = 1 - e^{-x}$ for $x = 0, 0.01, 0.02, \ldots, 2.98, 2.99, 3$. Draw the curve $y(x)$. Fit the curve by a third order polynomial. What are the coefficients of the polynomial? Draw also the fit (in the same figure).

A.3 Exercises chapter 3

Exercise 3.1: Removing variables

Delete all variables from the workspace.

Exercise 3.2: Numerical determination of an integral

In MATLAB calculate the following integral:

$$\int_1^3 (\cos(x-1) + \frac{1}{x})dx \tag{A.8}$$

What is numerical value of this integral?

Exercise 3.3: Symbolic determination of extrema

Determine the extrema of the function

$$f(x) = 2xe^{-x} - \sin(x) \tag{A.9}$$

on the interval [0,π].

The boundaries of this interval are not taken into account in this exercise.

Exercise 3.4

Plot the following functions on the interval $[0, 2]$

a) $f(x) = \frac{\arctan(x)}{1+x^2}$

b) $f(x) = \frac{1+\sin(x)}{(1+\cos(x))^2}$

If necessary, symbolise the symbols needed with the command `syms`.

Exercise 3.5

Enter the following exercises in order and give an explanation for the output.

a) `>> x^1/3`

b) `>> x^(1/3)`

Exercise 3.6

In MATLAB substitute in the following expression

$$x^2y^3 + \sin(\pi xy) \tag{A.10}$$

the variables x and y for 2 and 3 respectively.

Exercise 3.7

Determine by means of MATLAB the indefinite integral

$$\int \frac{1}{1+x^4} dx \tag{A.11}$$

Show (use MATLAB) that this answer is correct.

HINT: Differentiate the result. Apply the MATLAB function `numden` on the difference between the result and $1/(1+x^4)$.

Exercise 3.8

In MATLAB determine the following integrals:

a) $\int_0^{\pi} \frac{x}{2\sin(x)} dx$

b) $\int_0^{\infty} e^{-2x}\cos(3x) dx$

c) $\int_0^1 {}^{10}\log(x) dx$

d) $\int_0^4 |(x^2-2)^3| dx$

Call up the numerical values of the result.
Calculate this integral by dividing the integration-interval in two parts and get rid of the absolute values.
Do the results correspond with each other?

Exercise 3.9

In MATLAB determine the following two integrals and interpret the results.

a) $\int_0^2 \sin(t^2)\sqrt{1+t^3} dt$

b) $\int e^{-t^2} dt$

Exercise 3.10

In MATLAB plot the function $f(x) = 3 + \sin(x)$ on the interval $[-1,\ 2]$.
Make sure that the origin (0,0) is visible.

Exercise 3.11

Look at the result of the following MATLAB commands

```
>> syms x
>> ezplot(asin(sqrt(x)),[-5,5])
```

Why is the graph only given on the interval $0 \leq x \leq 1$?
Make a better drawing of the graph in MATLAB.

Exercise 3.12

After entering the command `>> syms x n`, calculate the integral

$$\int x^n dx \tag{A.12}$$

a) For which n is this answer not correct?

b) Ascribe the value of question a) to n and calculate the integral again. Is the answer correct this time?

Exercise 3.13

Explain the result of the command `>> sqrt(x^2)`

Exercise 3.14

Consider the function $f(x) = \frac{\log(1+x)+\frac{x^2}{2}}{(1+x)^2}$.
Enter the following MATLAB statements:

```
>> syms x y
>> y = (log(1+x)+x^2/2)/(1+x)^2
```

Answer the following questions:

a) Calculate $f'(x)$.

b) Calculate $f'(1)$.

c) Plot the graphs of $f(x)$ and $f'(x)$ in a figure.

Exercise 3.15

The following function is given $f(x) = e^{2x} - 5e^x + 6$.

a) Plot the graph of f on a suited interval.

b) Numerically determine the location of the minimum of f by using `fminbnd`.

c) Calculate the function value and the value of the derivative in the previously obtained point.

d) How many zeroes does f have?

Exercise 3.16

Consider the function $f(x) = x - 2 + 8\sqrt[3]{(x-1)^2}$

a) Plot the graph of f (first by hand).

b) Determine the zeroes and extrema of f.

c) Show that you have found all zeroes and extrema.

Exercise 3.17

Consider the function $f(x) = x\sin(x)$

a) Determine the extrema of the function f that lie inside the interval $[0, 2\pi]$.

b) Plot the graphs of $f(x)$ and $\sin(x)$, for $0 < x < 2\pi$, in a figure.

c) Explain the location of the extrema of f in comparison with those of the sine function.

Exercise 3.18

Consider the equation $\sqrt{x}\sin(\frac{1}{x}) = \frac{1}{4}, \quad x > 0$

a) Sketch the function $\sqrt{x}\sin(\frac{1}{x})$ and the constant $\frac{1}{4}$ in one figure.

b) Why does the equation not have any solutions for x close to 0.

c) How many solutions do there exist?

d) Approach at least one solution.

Exercise 3.19

The equation $e^x = ax(1-x)$,with an unknown x and a positive value for a, has exactly one solution between 0 and 1. Give an estimation of this value for a by means of graphs.

REMARK: The number a can be calculated exactly.

Exercise 3.20

The function $f(x) = e^{-x^2}(x + \sqrt{x})$ has a maximum in the neighborhood of $x = 0.6$.

a) Determine the location x_0 of the maximum.

b) Calculate $f'(x_0)$

Exercise 3.21

In MATLAB determine the 10^{th} derivative of the function f, i.e. $f^{(10)}(x), \quad x \neq 1$.

$f(x) = \sin(\frac{x^2+2x-3}{x-1})$

The answer can be given in one line! Explain your result.

A.4 Exercises chapter 4

Exercise 4.1

Enter the following 4×4 matrix:

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

Exercise 4.2

Enter the following 2×3 matrix:

$$A = \begin{bmatrix} 1.5 & 2 & \frac{1}{7} \\ \sqrt{3} & 2 & 0.625 \end{bmatrix}$$

Exercise 4.3

Enter the following matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}$$

Now, make a matrix D, which has matrices A and B above each other as first two columns, and matrix C as 3^{rd} and 4^{th} columns.

Exercise 4.4

Enter the following 2×3 matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{13} \end{bmatrix}$$

Exercise 4.5

Make a row vector with initial value -1, final value 9 and step size 0.5.

Exercise 4.6

Make a column vector initial value 9, final value 0 and step size 1.

Exercise 4.7

Make a column vector initial value a, final value $a + 20$ and step size 2.

Note that when computing the transpose, MATLAB assumes that the symbolic variables may have complex values. One can solve this by using the command:

```
>> x.'
```

Using the dot '.' implies that the operations is performed element-wise on every entry of the vector.

Another possibility is to declare the variable a as a 'real'. Look up how you should do this using the help function. Type `help sym` or `help syms`

Exercise 4.8

Enter the following matrices using commands like `eye` and `zeros`:

a) The 6×6 unity matrix

b) The 5×10 zero matrix

c) The 5×15 matrix in which all entries have value 1

d) A random 5×5 matrix

e) A diagonal matrix with the numbers 1 up to 5 on the diagonal.

Exercise 4.9

Enter the following 4×4 matrix.

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

and perform the commands:

```
>> A(3,2) = 3
>> A(3,[2 4])
>> A(1:3,[2 4])
>> A(2,:)-7*A(1,:)
>> A(2,:) = A(2,:)-7*A(1,:)
```

Exercise 4.10

Enter the following 4×4 matrix.

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

Investigate the effect of the command: `>> A([1 3],[1 3]) = 10*ones(2)`

Exercise 4.11

Enter the following 4×4 matrix.

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

a) Determine the transpose A^T of the matrix A by means of the command `B = A'`.

b) Calculate AA^T by typing `C = A*B`. Check whether C is symmetric.

c) Calculate A^TA by typing `D = B*A`. Compare your answer with $C = A * B$.

d) Calculate $3A + 5B^3$ by typing `E = 3*A+5*B^3`

Exercise 4.12

Enter matrix A and the vector $v = [1\ 2\ 3\ 4]$.

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

a) Calculate vA and Av^T. What happens with the rows and columns of the matrix when performing these multiplications?

b) Make a diagonal matrix D which has the elements of v on the diagonal. Calculate DA and AD. What happens with the rows and columns of the matrix when performing these multiplications?

Exercise 4.13

Enter the vector $v = [1\ 2\ 3\ 4]$.

Calculate vv^T and v^Tv and check whether it is a regular matrix multiplication.

Exercise 4.14

Enter the matrix A:

$$A = \begin{bmatrix} 1 & 2 & a \\ b & -3 & 2 \\ 3 & 1 & c \end{bmatrix}$$

a) Determine the transpose A^T of the matrix A by means of the command `B = A'`.

b) Calculate AA^T by typing `C = A*B`. Check whether C is symmetric.

c) Calculate A^TA by typing `D = B*A`. Compare your answer with $C = A * B$.

d) Calculate $3A + 5B^3$ by typing `E = 3*A+5*B^3`

Exercise 4.15

Consider matrix A, vector b and the system of linear equations $Ax = b$:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 14 \\ -2 \end{bmatrix}$$

Solve the linear system using the command `A\b` and using the `rref` command.

Exercise 4.16

Consider matrix A, vector b and the system of linear equations $Ax = b$:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} -4 \\ 4 \end{bmatrix}$$

a) Solve the system of linear equations using the command `A\b`. Is there a unique solution?

b) Determine all possible solutions using the command **rref**

c) Make the symbolic matrices `AA = sym(A)` and `bb = sym(b)` and solve the linear system symbolically using the command `AA\bb`.

Exercise 4.17

Consider matrix A, vector b and the system of linear equations $Ax = b$:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 10 \\ 16 \\ 17 \end{bmatrix}$$

a) Investigate by means of the command `rref([A,b])` whether this is an overdetermined system.

b) What is the result of the command `A\b`? Is this a solution of the linear system?

c) Make the symbolic matrices `AA = sym(A)` and `bb = sym(b)` and solve the linear system symbolically using the command `AA\bb`.

Exercise 4.18

Consider matrix A, vector b and the system of linear equations $Ax = b$:

$$A = \begin{bmatrix} 1 & 2 & -3 \\ 2 & 1 & -3 \end{bmatrix}, \quad b = \begin{bmatrix} -4 \\ 4 \end{bmatrix}$$

a) Solve this linear system using the command `solve`.

b) What is the general solution in vector notation?

Exercise 4.19

Solve the system of linear equations $Ax = b$ by means of symbolic matrices, that are a dependent on the parameter a. Is this solution defined for all a, when:

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & -1 \\ 1 & 1 & (a^2 - 5) \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 3 \\ a \end{bmatrix} ?$$

Compute the value of a for which the solution is not defined. Hereto, use the command for substitution `>>subs`

Exercise 4.20

Solve the system of linear equations $Ax = b$ by means of symbolic matrices, that are a dependent on the parameter a. Is this solution defined for all a, when:

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & -1 \\ 1 & 1 & (a^2 - 5) \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 3 \\ a \end{bmatrix} ?$$

Check whether solutions exist for $a = \sqrt{6}$ by use of the commands:

- `>>rref` and
- `>>A\b`

Exercise 4.21

Consider the system:

$$A = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

a) Compute A^2 and A^3 without use of MATLAB.

b) Could you give an analytical expression for A^n?

c) Compute A^{10} with MATLAB. Is this answer correct?

When an answer from MATLAB is required with more digits, this can be done by use of the command `>>format long`.

d) Use the command `>>format long` and again try to compute A^{10}. Compare this answer with the one of question c).

e) Could you give an analytical expression for A^n?

f) Use `>>format` to return to the representation with 5 digits.

Apparently, rounding errors may be visible in the final results. Use `>> help format` for more information about possible MATLAB outputs. The influence of rounding errors in the final results of computations can usually be reduced by means of designing a good algorithm.

Exercise 4.22

Consider two systems of equations:

$$\begin{cases} 10x_1 + 7x_2 + 8x_3 + 7x_4 = 32 \\ 7x_1 + 5x_2 + 6x_3 + 5x_4 = 23 \\ 7x_1 + 6x_2 + 10x_3 + 9x_4 = 33 \\ 7x_1 + 5x_2 + 9x_3 + 10x_4 = 31 \end{cases}$$

and

$$\begin{cases} 2y_1 + y_2 + 5y_3 + y_4 = 9 \\ y_1 + y_2 - 3y_3 - y_4 = -5 \\ 3y_1 + 6y_2 - 2y_3 + y_4 = 8 \\ 2y_1 + 2y_2 + 2y_3 - 3y_4 = 3 \end{cases}$$

a) Solve both systems of equations

By changing the right-hand-side of both systems of equations, we can obtain an impression of the sensitivity of the matrices for small changes.

b) Solve the first system of equations with the righthandsides $(32.1, 22.9, 32.9, 31.1)$ and $(32.01, 22.99, 32.99, 31.01)$. Solve the second system of equations with right hand-sides $(9.1, -5.1, 7.9, 3.1)$ and $(9.01, -5.01, 7.99, 3.01)$. What is the effect of these minor changes?

By slightly changing the coefficients of the equations and studying the effect of these changes to the solutions, one can get an impression of the sensitivity for small data errors.

c) Change the elements of the original matrices by adding a matrix `0.1*rand(4)` to both of them, such that for each element, a random number between 0 and 0.1 is added. Solve the systems of equations with the original right-hand sides.What is the effect of the changes?

d) Systems of equations that are sensitive for small data changes have *ill-conditioned* matrices of coefficients. Which of the two systems has the most well-conditioned matrix of coefficients?

Exercise 4.23

With the MATLAB command `>>A=hilb(n)` the $n \times n$ Hilbert matrix is created, whose elements are given by $A_{ij} = \frac{1}{i+j-1}$.

a) Determine the 5×5 Hilbert matrix and call it A

b) Solve $Ax_1 = b_1 = (2.2833, 1.4500, 1.0929, 0.8845, 0.7456)^T$

c) Solve $Ax_2 = b_2 = (2.2834, 1.4501, 1.0928, 0.8844, 0.7457)^T$

d) Compare b_1 with b_2 and x_1 with x_2.

Exercise 4.24

Determine all 2×2 matrices that commute with:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix},$$

In other words, find all matrices B such that $AB = BA$.

Hint: use the Symbolic toolbox.

Exercise 4.25

Determine all 2×2 matrices that commute with:

$$A = \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix}$$

Exercise 4.26

Consider n particles in a plane, with position vectors $r_1, r_2, \ldots, r_n$ and masses $m_1, m_2, \ldots, m_n$. The position vector of the center of mass is given by:

$$r_{CM} = \frac{r_1 m_1 + r_2 m_2 + \ldots + r_n m_n}{m_1 + m_2 + \ldots m_n}$$

Consider a triangular plate with corner points $(1, 2)$, $(5, 1)$ and $(4, 4)$, whose mass is concentrated at the three corner points. How should we distribute a total mass of 1kg over the corner points, such that the center of mass is positioned at $(3, 3)$?
Now, consider a plate with four corner points. Hereto, use the previous positions for three of the points, choose a fourth position yourself. What mass distributions are possible?

Exercise 4.27

The linear momentum P of a system with n particles with masses $m_1, m_2, \ldots, m_n$ and velocities $v_1, v_2, \ldots, v_n$ is given by:

$$P = m_1 v_1 + m_2 v_2 + \ldots + m_n v_n$$

Consider two particles with velocities $v_1 = (2, 2, 2)^T$ and $v_2 = (5, 7, 9)^T$. These particles will make a collision. After this collision, their velocities are $w_1 = (4, 6, 4)^T$ and $w_2 = (4, 5, 8)^T$. Assuming preservation of linear momentum during impact, what is known about the masses of both particles?

Exercise 4.28

Draw in a rectangular area $-6 \leq x \leq 6, \quad -6 \leq y \leq 6$ the triangle with corner points $p_1 = (1, 2)^T$, $p_2 = (5, 1)^T$ and $p_3 = (4, 3)^T$. Let D be the matrix:

$$D = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

a) Draw the triangle with corner points $b_i = Dp_i$.

b) Draw the triangles with corner points $D^2 p_i$ and $D^3 p_i$ in the same graph.

c) What is the effect of multiplication with D?

d) What matrix should be used, such that multiplication with this matrix yields an image, mirrored in the x-axis?

A.5 Exercises chapter 5

Exercise 5.1

a) Using `ode23` calculate the solution of the following differential equation:

$\dot{x} = 1 - 2tx$

with initial condition $x(0) = 1$ on the interval $[0, 3]$.

b) What is the result when one leaves out the specification $[t, x]$?

Exercise 5.2

Consider the initial value problem:

$\dot{x} = \frac{tx}{(1+x^2)}, \quad x(0) = 3$

a) Using `ode23` calculate $x(t)$ on the interval $[0, 2]$. How many steps are used to obtain this result? Plot the graph of $x(t)$.

b) What is de approximation for $x(2)$ in part a)?

Exercise 5.3

Draw the direction field for the differential equation of the following system

$x(1)' = 5x(1) - 2x(2)$
$x(2)' = 7x(1) - 4x(2)$

on the grid given by the vectors:

x = [-1:0.1:1] and y = [-1:0.1:1].

Exercise 5.4

Check if the differential equations

$$\dot{x} = 1 - tx, \text{ and } \dot{x} = \frac{tx}{1 + x^2}, \tag{A.13}$$

can be solved symbolically. Notice that new functions are introduced.
Check with 'Help' what kind of functions this are?

Exercise 5.5

a) Work out the example from paragraph 5.3.2.

b) Plot the solution curves in the phase plane (in 1 figure) with initial conditions: $(x_1(0), x_2(0))$ and (2,8), (2,7), (0.1,0.1), (-2,-8), (-2,-7), (-2,-6), (-0.1,-0.1) respectively for $t \in [0, 1]$.

HINT: It is recommended to fix the axes with the command `axis`, e.g. x_1 and x_2 between -10 and 10.

Perform this exercise both numerically and symbolically.

Exercise 5.6

Consider the differential equation:

$\ddot{y}(t) + 2k\dot{y}(t) + 25y(t) = 0, \quad$ with initial conditions: $y(0) = 1$ and $\dot{y}(0) = 1$.

a)

1) Determine the solution of this equation for $k = 0$.
   The solution describes a vibration. You can visualise this by plotting the solution.
2) Is the vibration damped?
3) Is the vibration periodic?
4) What is the frequency of this solution?

b)

1) Determine the solution of this equation for $k = 1$.
2) What is the eigenfrequency of the system?
   The solution describes a vibration. You can visualise this by plotting the solution.
3) Is the vibration damped?
4) Is the vibration periodic?
5) What is the frequency of the solution?

   Plot the solution and the function e^{-kt} in the same figure.

c)

1) Determine the solution of the equation:

   $\ddot{y}(t) + 2\dot{y}(t) + 25y(t) = \cos(5t), \quad$ with $y(0) = 1$ and $\dot{y}(0) = 1$.

   The solution describes a vibration.You can visualise this by plotting the solution.

2) Is the vibration damped?

3) Is the vibration periodic?

4) What is eventually the frequency and amplitude of the solution?

Exercise 5.7

Consider the differential equation of a harmonic oscillator given by a damped mass-spring-damper system (eq. 4, Chapter 3):

$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = 0$

Suppose we have a spring that stretches 20 cm in length caused by a weight of 98 Newton. Thus $k = 490$ N/m and $m = 10$ kg. The damping c is chosen such that $c^2 < 4mk$ holds, that is sub-critically damped.

$10\ddot{y}(t) + 50\dot{y}(t) + 490y(t) = 0$

This differential equation is preferably written as set of first order differential equations. Suppose $x_1(t) = y(t)$ and $x_2(t) = \dot{y}(t)$, then we get the following equations:

$\dot{x}_1 = x_2$
$\dot{x}_2 = -5x_2 - 49x_1$

a) Plot the solution $y(t)$ with initial conditions $y(t) = 0.2$ and $\dot{y}(t) = 0$ on a time-interval that is large enough and plot this solution in the phase plane as well.
What happens if you choose other initial conditions?

b) Plot for the super-critically damped system, $c > 140$, the solution. Use the same initial conditions.

c) See if you can find initial conditions in the case of critical damping, $c = 140$, for which the solution at least possesses one zero (at least a crossing of the equilibrium position).

Exercise 5.8

Consider the Lotka-Volterra equations for a prey-predator model (more information can be found in [5])

$\frac{dp}{dt} = (a_1 - b_1 r)p \qquad \frac{dr}{dt} = -(a_2 - b_2 p)r$

with a_1, a_2, b_1, $b_2 > 0$. Here $p(t)$ denotes the number of preys and $r(t)$ the number of predators. The preys have an inexhaustible food source and there exist no competition among themselves. When there are no predators, the population of the preys grows exponentially. For the predators the preys are the only food source. When no preys are left the predator population dies out.

One can show that $p(t)$ and $r(t)$ satisfy the following relation:

$a_2\log(p) - b_2p + a_1\log(r) - b_1r = \text{constant}$

This equation gives a closed curve in the (p, r)-plane as result for every constant. The population develops itself thus according to a periodic pattern.

a) Take the values $a_1 = 1.00$, $b_1 = 0.10$, $a_2 = 0.50$ and $b_2 = 0.02$.

   Plot the solutions $p(t)$, $r(t)$ and the corresponding curve in the phase plane for some self chosen initial conditions and time-interval. Notice that a stationary point (constant solution) exist in the phase plane at $p = \frac{a_1}{b_1} = 10$ and $r = \frac{a_2}{b_2} = 25$.

REMARK: It could be that the plotted phase curve is not exactly a closed curve due to numerical reasons in MATLAB but a spiral curve. This only occurs when plotting on large i-intervals. `ode45` gives the best result in this case.

b) The parameters a_1 and a_2 determine the development when there is no interaction.
   The parameters b_1 and b_2 determine the interaction between predator and prey.
   See how the population develops itself for different combinations of the parameters.
   Notice that only the first quadrant of the (p, r)-plane is of importance.

Exercise 5.9

Consider a driven mass-spring-damper system given by the equation:

$m\ddot{y} + c\dot{y} + ky = c\dot{x} + kx$

Here x denotes input of the drive and y the response of the output.
$m = 870$ kg, $k = 70000$ N/m, $c = 5000$ N/ms (one can think of the suspension of a car). For the input we suppose $x(t) = \sin(t)$.

Calculate and plot the output.
(Choose your own initial conditions, hence not (0,0), and choose an interval on which the periodic behaviour is visible.

Exercise 5.10

As like the previous exercise consider a driven mass-spring-damper system given by the equation:

$m\ddot{y} + c\dot{y} + ky = k * \text{sign}(\sin(2\pi t))$

Here $m = 870$ kg, $k = 70000$ N/m, $c = 5000$ N/ms.

Investigate what the right-hand side of the equation means.
Notice that the MATLAB function `sign` does not accept symbolic arguments.

Calculate and plot the output (choose your own initial conditions, hence not (0,0), and choose an interval on which the periodic behaviour is visible.

A.6 Exercises chapter 6

Exercise 6.1: Script file

a) Make a script file 'test1.m' in which MATLAB successively performs the following:

   - Delete all variables from the memory.
   - Define a row vector x =1:0.1:10.
   - Calculate $y = \sin(x) - \cos(x)$.
   - Plot y as function of x.

   For later use, add comments to your script file so you can understand why you made the file.

   Save this script file in the MATLAB work directory.

b) Test the script file and, if necessary, debug the file.

c) Call up the commentary of the file with `>> help test1`

Exercise 6.2: Function file

a) Make a function file 'test2.m' in which MATLAB for an arbitrary input x the output y calculates and displays on the screen.

   - Calculate $y = \sin(x) - \cos(x)$.
   - Plot y as function of x.

   For later use, add comments to your script file so you can understand why you made the file.

   Save this script file in the MATLAB work directory.

   Pay attention! that the function name is the same as the corresponding filename.

b) Test the function file with a = 1:0.1:10 and, if necessary, debug the file.

c) Call up the commentary of the file with `>> help test2`

d) Explain the difference between a script file and a function file.

Exercise 6.3: FOR loop

Make a function file of the following form:

```
function b = kwad(r)
```

in which a row vector is squared element-wise. Do this without using the command `>> .^2`.

(HINT: Use the command `>> for`)

a) Work out the problem on paper.

b) Write the m-file in MATLAB.

c) Test the function file with the row a = [1 2 3 4 5] and, if necessary, debug the file.

Exercise 6.4: FOR loops

Make a function file of the following form:

```
function B = kwadm(A)
```

in which a matrix is squared element-wise. Do this without using the command `>> .^2`.

(HINT: Use the function file from the previous exercise as starting point)

a) Work out the problem on paper.

b) Write the m-file in MATLAB.

c) Test the function file with the matri A = [0 1;2 3] and, if necessary, debug the file.

Exercise 6.5: Input

Make a script file 'date.m' with which the user can call up de desired date.

How to solve the problem

- Ask the user (with the command `>> input` if he/she wants to know the date (1=yes, 2=no).
- If the answer is 'yes' the date must be displayed (`>> date`).
- If the answer is 'no' display on the screen that the user does not want to know the date (`>> disp`)

a) Write the m-file in MATLAB.

b) Test the script file and, if necessary, debug the file.

Exercise 6.6: IF loop

Make a function file of the following form:

```
function b = replace(a)
```

in which all elements of a column greater than 5 are replaced by 0. Do this by without using the command `find`.

(HINT: Use the commands `>> if` and `>> for`)

a) Work out the problem on paper.

b) Write the m-file in MATLAB.

c) Test the function file with the column [1;2;3;4;5;6;7] and, if necessary, debug the file,

Exercise 6.7: IF loops

Make a function file of the following form:

```
function B = replace2(A)
```

in which all elements of a matrix greater than 5 are replaced by 0. Do this without using the command `>> find`.

(HINT 1: Use the commands `>> if` and `>> for`)
(HINT 2: Take the function file from the previous exercise as starting point.)

a) Work out the problem on paper.

b) Write the m-file in MATLAB.

c) Test the function file with the matrix [2 12;5 8] and, if necessary, debug the file.

Exercise 6.8: WHILE loop

Make a function file 'divide.m'.

This function must divide a number x by 2 till the new obtained number is smaller than 2.

Input of the function file:

- x: An arbitrary number.

Output of the function file:

- p: A number that represents how often the number x is divided by 2.
- q: The value that is left when the number x is divided by 2, p times.

a) Work out the problem on paper.

b) Write the m-file in MATLAB.

Exercise 6.9: Debugging

Take over the function file 'errors.m' below and save the file.

Test and debug the file with x = [1 2 3;4 5 6;7 8 9;10 11 12].

(HINT: 3 errors are present in the function file)

```
function [c,d] = errors(x)

% This function file generates a matrix d of which the elements
% are the elements of x minus 1. At the same time this function
% file calculates how often (c) passes through the 2nd loop.

[p,q] = size(x)
for a = 1:q
   for b = 1:p
   d(a,b) = x(a,b)-1
   c = c+1
end
```

Exercise 6.10

Make a function file of the following form:

```
function D = matmult(A,B)
```

that multiplies matrices A and B of arbitrary dimensions. Do this by using only the scalar multiplying operator '*'. In other words, write a matrix multiplication such that only (multiple times after each other) two normal numbers are multiplied.

a) Analyse the problem (HINT: Investigate how the product of a matrix and a column can be obtained element after element. Pay attention that the multiplication $A * B$ is not the same as $B * A$. First of all, for simplicity, consider a 2x2 or a 3x3 matrix to notice how things work. The final programme has to work for all kinds of dimensions.)

b) The function must perform successively the following steps:

   1) Determine the dimensions of the input arguments `mat` and `col`.
   2) Investigate if these dimensions are suitable for multiplication. If not, generate an error message on the screen. (HINT: Use the commands `>> error('text')` and `>> disp('text')`)
   3) Determine the dimensions of the resulting matrix based on the dimensions of the input arguments. Initialise this matrix with zeroes for its elements. (HINT: Use the command `>> zeros`)

4) ,5) Fill the resulting matrix, element after element, with a for-loop. The value of each element is a summation of different product terms (an inner-product). For this a new for-loop needs to be implemented in the first for-loop with the following assignment instruction:

   `sum = sum + change` (the new value of the sum becomes the previous value of the sum plus the value of the change)

   First calculate the inner-product for a row-column multiplication and later on extend this to a matrix-column multiplication.

6) Make sure that the resulting matrix is the output argument of the function.

c) Test the function for all kinds of matrices A and columns B. Check the results with MATLAB's own matrix multiplication.

Exercise 6.11

Extend the function file 'matmult.m' from exercise 6.10 to a multiplication of two matrices. Save this file under the name 'matmult2.m'.

P.S. This exercise is meant for the die-hard MATLAB guru's (which you will all be at the end of this course!)

A.7 Exercises chapter 7

Exercise 7.1: Source and Sink

Generate a sinusoidal wave in Simulink. Visualise this signal by means of a scope. Hereto:

1. open simulink and a new .mdl file,
2. drag the block "sinewave" from the library brower, category "Sources", to the .mdl file,
3. drag the block "scope" from the library brower, category "Sinks", to the .mdl file,
4. connect both blocks,
5. save the model as exercise71.mdl, and
6. run the model

Exercise 7.2: Block parameters

1. Adapt the Simulink model of the previous excercise, such that the sinus has amplitude 10 and frequency 0.5Hz. Rerun the file. *Hint: Double click on the "sine-wave" block"*
2. Change the simulation time to 40 seconds. Rerun the file. *Hint: Simulation - Simulation parameters"*

Exercise 7.3: Dynamical system

1. Extend the Simulink model in file exercise71.mdl, such that the sine wave is integrated. Visualise the integrated signal in a scope. Run the file.
2. Use -1 as initial value of the integrated signal. Rerun the model.

Exercise 7.4: Save data

1. Model a cosine in Simulink. Visualise the signal for 10 seconds in a scope.
2. Save the output of the cosine in a .mat-file. Hereto, use the "To File" block from the library category Sinks. Replace the "Variable name" to "cosine". Run the Simulink model.
3. Load the .mat file in Simulink with the `load` command. Plot the second row of it's data versus the first row. This should yield the same result as the scope.

Exercise 7.5: Equation

Model with Simulink the equation

$$A = \frac{3}{4}B + 5.$$

Hint: B can be any input. However, to run the file and check the result, use an arbitrary source, such as a ramp.

In this exercise, proceed as follows:

1. choose necessary blocks,
2. drag these blocks to a new model window,
3. adapt the parameters,
4. connect the blocks,
5. save the .mdl file, and
6. run the file.

Exercise 7.6: differential equation

1. Simulate the trajectory $x(t)$ of the following differential equation for 10 seconds:

$$\dot{x} = Ax + Bu, \quad x(t=0) = 4, \tag{A.14}$$

   where $A = -0.5$, $B = 3$ and $u(t) = 1$. *Hint: First construct a block diagram. Build this diagram in Simulink, add sinks and sources.*

2. Change the input to $u(t) = \sin(2t)$ and rerun the simulation

Exercise 7.7: differential equation

Simulate the trajectory $x(t)$ of the following differential equation for 10 seconds:

$$\begin{aligned} \dot{x} &= -x - xy \\ \dot{y} &= x^2 - y + u, \end{aligned}$$

with $x(t=0) = 1, y(t=0) = -1$. First, assume the input $u(t) = 0$.

1. Run a simulation and save both $x(t)$ and $y(t)$ to the workspace.
2. Plot the trajectory in phase plane.
3. Now, use $u(t) = 0.01\sin(t)$, rerun the model and plot the new trajectory in phase plane

Hint: Use the "Product" block from "Math operations".

Appendix B

Answers

B.1 Answers chapter 1

Answer 1.1

a) `>> a = 21`
The result of the sum $1 + 2 + 3 + 4 + 5 + 6$ is ascribed to the variable a. De output $a = 21$ appears in the MATLAB Command Window.

b) `>> ()`
The result of the sum $1 + 2 + 3 + 4 + 5 + 6$ is ascribed to the variable b. The output is not visible in the MATLAB Command Window because the semicolon sign (;) is used. However, it is possible to call up the variable b.

c) `>> ans = 24`
The output of the sum $b + 3$ appears in the Matlab Command Window as $ans = 24$. No variable is ascribed to this calculation. In such a case the result is automatically ascribed to the variable ans.

d) `>> ans = 21`
Also in this exercise no variable is ascribed to the calculation. The output is thus as follows: $ans = 21$.

e) `>> c = 210`
The result of this exercise depends on the previous answer because the variable ans is used in this calculation. Assume that $ans = 21$ (obtained from exercise d), then the answer on this question becomes: $c = 210$.

Answer 1.2

By using the key combination 'ctrl-c' you can interrupt a calculation or output in MATLAB. The effect of the key combination 'ctrl-c' in this exercise is that the output is early canceled, considering the fact that you apply the key combination in time.

OUTPUT SUMMARY:
`>> 0:90000`
key combination: ctrl-c

Answer 1.3

a) `>>  a = [2 3 4`
The exercise $a = [2, 3, 4$ can be finished, after entering the 'return' key, by closing the array yet with a bracket] and again enter the 'Return' key. The array [2 3 4] is now ascribed to the variable a.

b) `>> ctrl-c ()`
The exercise $a = [2, 3, 4$ can be interrupted by using the command 'ctrl-c'. The calculation is then stopped and nothing is ascribed to the variable a.

Answer 1.4

```
>> u = 2:4; v = [1 5];
>> who
>> clear u v
>> who
```

Answer 1.5

Existing variables can be saved in a mat-file by using the command `>> save filename.mat`. The file is written to the current active directory (see: 'Current directory' in the MATLAB taskbar). Later on it is possible to call up the variables by loading the mat-file (pay attention that you are working in the directory in which MATLAB has written the mat-file) by using the command `>> load filename`. Entering the command `>> who` gives a list with used variables. Entering the command `>> what` gives a list with existing mat-files in that current directory. The mat-file can be deleted by entering the command `>> delete filename.mat`.

INPUT SUMMARY:

```
>> f = 2
>> g = 3 + f
>> save try.mat f g
% Exit Matlab and start it again
>> who
>> what
>> delete try.mat
```

Answer 1.6

a) 0.1707

b) 2.5198

c) 0.7071

d) 1

Answer 1.7

First you have to enter `>> x = 2`. Then it is possible to enter the expressions by using the input below.

Answer 1.8

```
a =
```

```
Columns 1 through 7

 -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000

Columns 8 through 14

  2.5000    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000

Columns 15 through 21

  6.0000    6.5000    7.0000    7.5000    8.0000    8.5000    9.0000
```

Answer 1.9

You can define a row vector as follows
Vector name = initial value : step size : final value

INPUT SUMMARY:
`>> a = 9:-1:0`

Answer 1.10

a) The error in this formulation is captured in the division (/). In this way a matrix-wise division is used (which has a number as result) instead of an element-wise division (which has an array as result).

b) The error message means that you used the power sign (^) wrong. When you want to calculate a matrix to the power n the matrix has to be square. The first term `x^3` must therefore be replaced with the term `x.^3`.

Answer 1.11

Enter the array x:
`>> x = 0:0.1:2`

Enter the functions f. Pay attention that you make use of element-wise calculations (thus with dots!).

a) `>> f=x.^2+2*x+1` or `>> f=x.^2+2.*x+1`

b) `>> f=(3.^x)./(1+3.^x)`

c) `>> f=x./(1+sqrt(x))` or `>> f=x./(1+x.^(1/2))`

Entering '`f =`' is not necessary. Although it is useful when you want to separate different equations.

Answer 1.12

```
>> nr = find(a<0.3)
```

Then the result can be obtained as follows: `>> a(nr)`

Answer 1.13

```
>> clear all
>> t = 0:0.1:1;
>> f = t./(1+sqrt(t))
```

Answer 1.14

```
>> prod(size(find(a>5)))
```

or

```
>> length(find(a>5))
```

a)
```
ans =

     4
```

b)
```
ans =

     5
```

c) Different output possible due to the command **rand**

Answer 1.15

The one line command is given by:

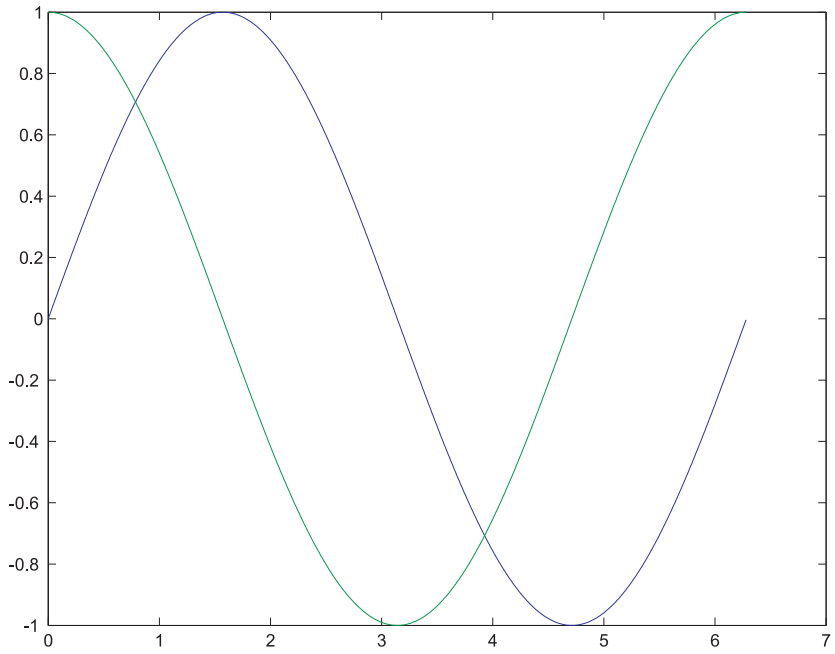
```
>> meanvalue = sum(sum(a))/(prod(size(a)))
```

The mean value of the array is calculated by:
(sum of elements)/(number of elements)

You can calculate the sum of the elements by using the command `sum`. This command gives you the sum of the elements in each row. To calculate the sum of the elements of the matrices it is necessary to use this command twice.
You can obtain the number of elements in an array by using the command **size**. This command returns the dimensions of the array (2 numbers; number of rows, number of columns). By multiplying these 2 numbers you get the number of elements of the array a.

PAY ATTENTION: You do not have to use a dot before the division sign (/) because this time a division of 2 numbers is concerned.

Answer 1.16



Answer 1.17

Plotting a figure in MATLAB can be done in several ways. In this answer a plot is made by typing several commands.

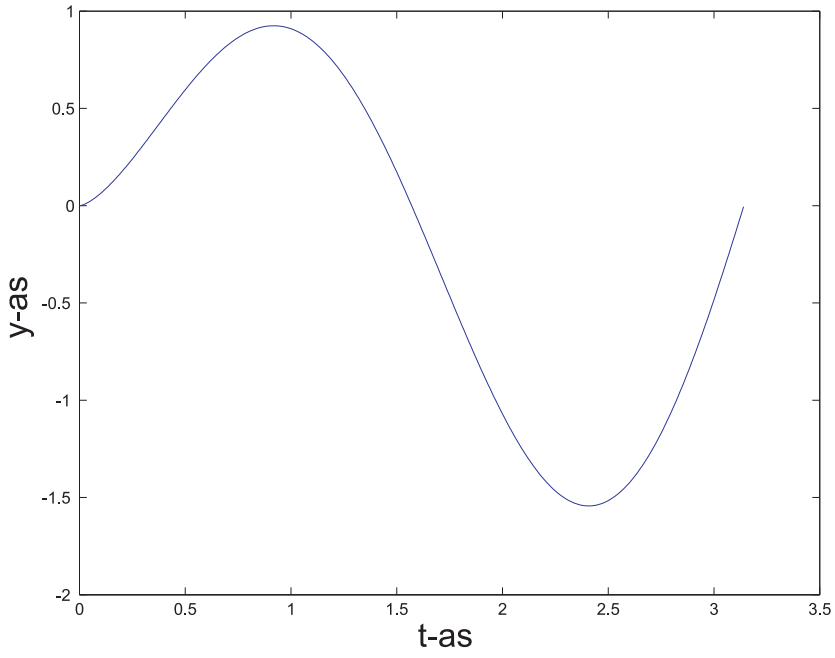
INPUT SUMMARY:

```
>> clear all
>> t = 0:0.01:pi;
>> y = sqrt(t).*sin(2t);
>> figure
>> plot(t,y)
>> xlabel('t-as')
>> ylabel('y-as')
```

Answer 1.18

Answer 1.19

The reason that you can better save the commands for plotting a figure than the figure itself is simply because then you can see how you obtained that figure. If you want to adjust the



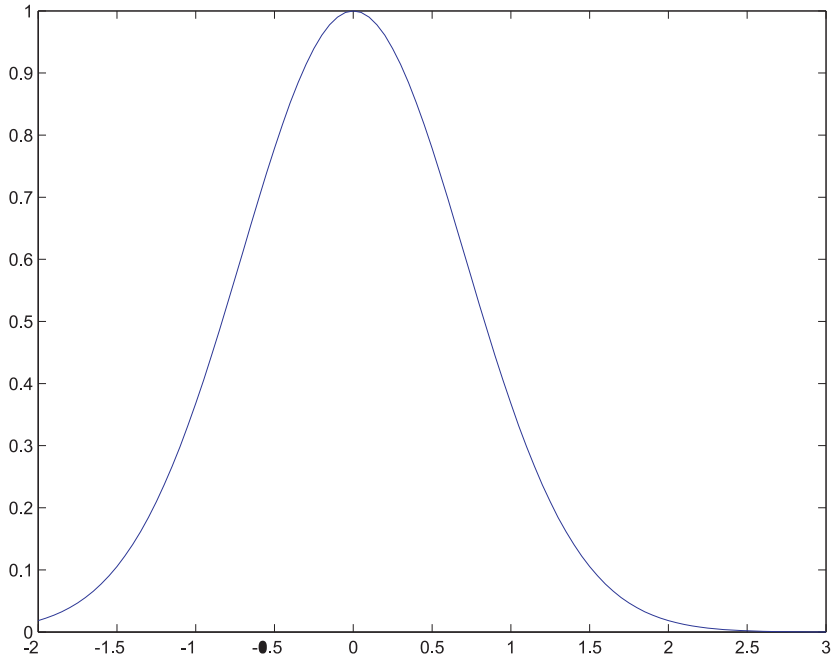


figure in a later stadium then it is not necessary anymore to enter the commands again.

Answer 1.20

```
x = 0:0.1:2;
% Enter the array x with starting value 0 and final value 2
```

```
% and step size 0.1, without reproduction in the Matlab
% Command Window (;)
y = abs(x.*sin(2*pi*x));
% Enter the function: f(x) = |x*sin(2*pi*x)|
% also without reproduction in the MATLAB Command Window.
figure(1)
% Producing of figure 1
plot(x,y)
% Depicting the function y, with x on the x-axis and y on the y-axis.
title('f(x) = |x*sin(2*pi*x)| on the interval [0,2]')
% The title between the brackets is ascribed to the figure.
axis([0 2 0 2])
%% The axes of the figure are scaled by [xmin xmax ymin ymax]
shg
% The figure is presented on the foreground.
```

Answer 1.21

The file is saved in the current directory.On top of the MATLAB window you can find the name of the current directory.

With the command `>> dir` you can call up the content of the current directory.

If you want to call up the function value of f for $x = 1$, it can be done as follows:

```
>> f(1)

ans =
    3.7183
```

Answer 1.22

```
>> f(1)

y =
    3.37183
ans =
    3.7183
```

The output is not displayed on the screen when the semicolon (;) is used while the opposite is true when the semicolon is left out.

Answer 1.23

In the following you will find a possible formulation for your own made function $g(x)$ in MATLAB.

```
function y = g(x)

a = zeros(size(x));
y = max(x,a);
```

The array a consists of zeroes (made with the MATLAB function `zeros` with the same dimensions as the array x.

With the command `>> max` you can compare the arrays x and a element-wise and that gives you an array y that element-wise consists of the maximal values.

Thus:
if $x(1) > a(1)$ then $y(1) = x(1)$.
if $x(1) < a(1)$ then $y(1) = a(1)$.

Answer 1.24

- In the parts c) till e) 'h' is considered as a variable.

- In part d) no error message arises because you call up the first element of array h. The array h at that moment consists of one element with the value 2.5.

- Even if you enter a variable with the same name as in the previous defined function, the function still exists.

Answer 1.25

In the MATLAB Command Window it is directly possible to obtain information about different functions by using the command `help 'keyword'`. The specification that appears in the MATLAB Command Window gives you specific information about the function which is asked for.
This is an useful approach if you know which function you have to use. Hence, if you do not know this you can better use the MATLAB Help Browser and then the header "Search".

INPUT SUMMARY:
```
>> help sin
>> help load
```

Answer 1.26

```
>> helpwin
```

then clicking:

→ Matlab
→ In Alphabetical Order
→ atan

Answer 1.27

This exercise can be solved in several ways. It is important that you find the way that is most suited for you. In the following a possible solution is given.

INPUT SUMMARY:
→ You know that 'sin' is the MATLAB function name for the sine function, thus you are going to use the MATLAB "Help Browser" and the option "Search".
→ Click on the 'yellow question mark' in the MATLAB taskbar.
→ Select the header "Search" in the "Help Navigator"
→ Enter by "Search for" the following: 'sin'
→ ENTER
→ Now 'sin' appears in the the window below.
→ Choose for 'sin' and obtain the help information by double clicking.

Answer 1.28

```
ans =

  Columns 1 through 5

           1           32          243         1024         3125

  Columns 6 through 10

        7776        16807        32768        59049       100000

  Columns 11 through 15

      161051       248832       371293       537824       759375

  Columns 16 through 20

     1048576      1419857      1889568      2476099      3200000

  Columns 21 through 25

     4084101      5153632      6436343      7962624      9765625

  Columns 26 through 30

    11881376     14348907     17210368     20511149     24300000
```

```
Columns 31 through 35

  28629151    33554432    39135393    45435424    52521875

Columns 36 through 40

  60466176    69343957    79235168    90224199   102400000
```

Answer 1.29

```
>> 1.5511e+025
```

This question can be answered in several ways:

1. The easiest approach is by typing `>> 1*2*3*4*...*25*25` in the MATLAB Command Window.
2. A more elegant approach, which is also easy to expand for numbers greater than 25, is by writing a MATLAB script. First you need to define a row vector with the numbers 1 till 25. Then multiplying these elements with each other by means of the MATLAB command `prod`, you get the desired answer.

   INPUT SUMMARY (Option 2):

   ```
   >> clear all
   >> a = 1:25;
   >> k = prod(a)
   ```

Answer 1.30

```
>> 2686700
```

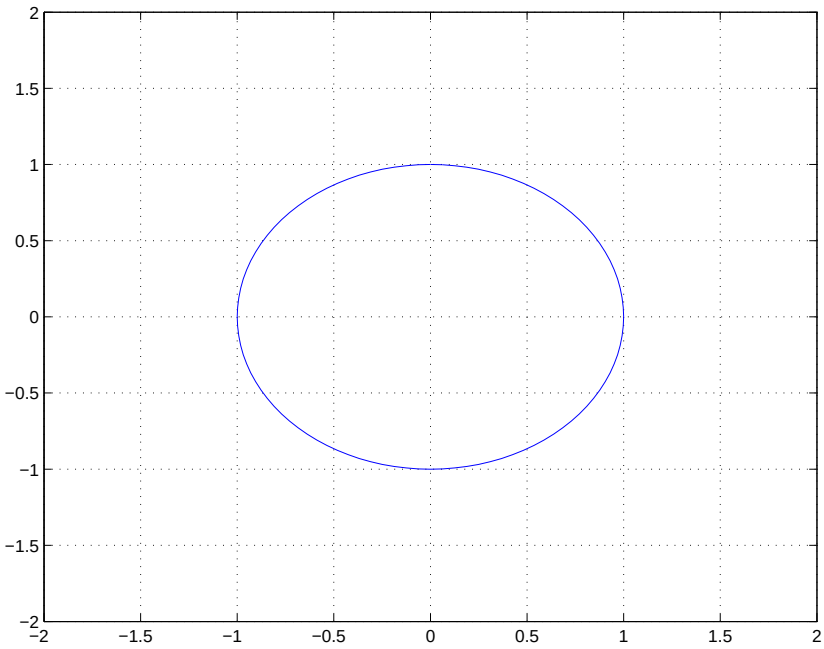
Answer 1.31

It is the best approach to write an m-file when you want to display a function in a figure. Before you are going to calculate vectors or matrices, it is usual that you delete all current defined variables with the MATLAB command `clear`.
To plot the function $f(x) = x/(1+x)$ on the domain $[0, 4]$ you first have to define a row vector with elements. Then you need to calculate $f(x)$ by means of element-wise calculations. Then you can plot the vectors x and y by using the command `plot`.
If you want to embellish your figure you can make use of commands such as; `title`, `xlabel`, `ylabel` and `grid`.

INPUT SUMMARY:

```
>> clear all
>> f = @(x)(x./(1+x));
>> x = 0:0.01:4;
>> figure
>> plot(x,f(x))
>> xlabel('x'); ylabel('f(x)')
>> grid
```

Answer 1.32



Answer 1.33

You can define a new MATLAB function by opening a new m-file and the type
`>> function [output] = function_name(input)`.
The function in this exercise must be able to handle with arrays so therefore you need to multiply element-wise (thus with a dot!).

INPUT SUMMARY:

```
>> function y = fun(t)
>> s = mod(t,2)
>> y = s.*s
```

or

```
>> function y = f(x)
>> y = mod(x,2).^2
```

	Input	Output
a)	`(x^3)/6`	1.3333
b)	`x/(sqrt(1+x^2))`	0.8944
c)	`2^(1/3)`	1.2599
d)	`exp(1+x^2)`	148.4132
e)	`x^3*sin(x^2)`	-6.0544
f)	`atan(x)/(1+x^2)`	0.2214

B.2 Answers chapter 2

Answer 2.1: User defined function

The Matlab function $f(x) = x^3 - x^2 - 3\arctan(x) + 1$ can be made in the following way:

```
function y = f(x)
y = x.^3-x.^2-3.*atan(x)+1;
```

The word 'function' gets the colour blue, which means that MATLAB recognizes this word as a MATLAB function to be defined.
The dots have to placed at the particular positions in order to get the function operational for arrays and matrices as x-input.

The ';' is not really necessary, but is extremely helpful to prevent undesired screenfilling with formulas, variables and so on.

The m-file has to be saved with the same name as the function. So, the name of this m-file will be 'f.m'. (where '.m' is the extension of a m-file.)

INPUT SUMMARY:
```
function y = f(x)
y = x^3-x^2-3*atan(x)+1;
```

Answer 2.2: Numerical approximation of a zero

```
ans =

    -1.27798143534050
```

Answer 2.3: Numerical approximation of a zero

The other zeroes of the graph of f can be determined in the same way. However, in this case one should choose the whole interval where the function f is defined.

This means that we have to search for intervals which contain the remaining zeroes. The function `fzero` will search these intervals for a possible zero. Mind that the bounds of the intervals should have opposite sign.

Examine the graph which is plotted in demo 3.1.
From that graph you can extract three intervals where the graph passes through the x-axis. Take, for instance, the following intervals:

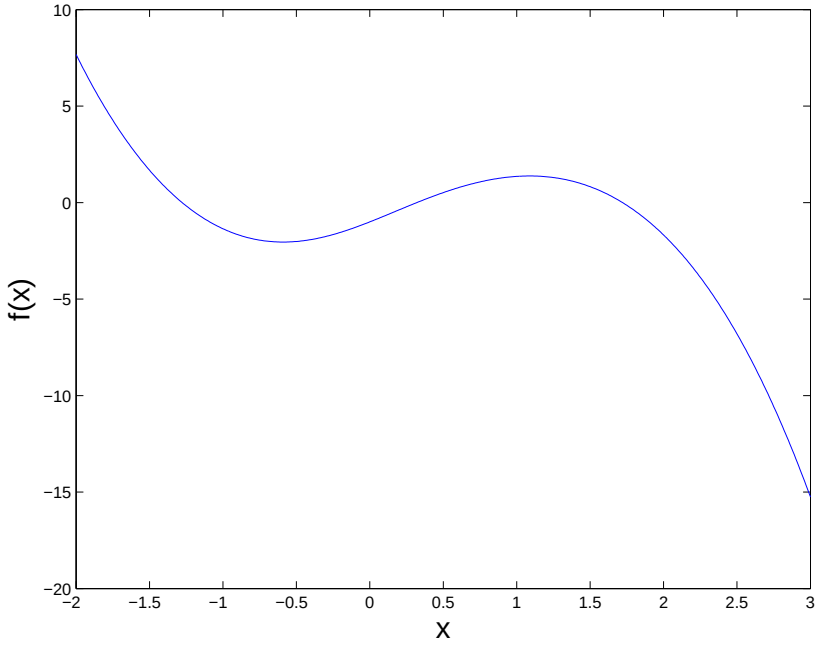
1. $x \in [-2, -1]$
2. $x \in [0, 0.5]$

3. $x \in [1.5, 2.5]$

By means of `fzero` you can approximate the zeroes in these intervals.

1. `np = fzero('f',[-2,-1])` gives $np = -1.2780$.
   Substitution of np in the function f: $f(np) = 4.4409e^{-016}$.
2. `np = fzero('f',[0,0.5])` gives $np = 0.3204$.
   Substitution of np in the function f: $f(np) = -2.2204e^{-016}$.
3. `np = fzero('f',[1.5,2.5])` gives $np = 1.7205$.
   Substitution of np in the function f: $f(np) = 0$.

Answer 2.4: Drawing of a function



Answer 2.5: Numerical determination of a minimum

Use `fminbnd` in this exercise.
In order to adjust the accuracy, you have to call the option: 'optimset' with the right arguments.

```
>> minimum = fminbnd('f',0,2,optimset('TolX',10^n))
```

The minimum is searched in the interval [0,2].
The option 'TolX' states which tolerance is used in the x-direction. This tolerance is 10^n, where n should be taken as -4, -6, -8 and -10 respectively.

Then, the following values should be visible on your screen:

```
1.08776796550467
1.08775494081871
1.08775500139164
1.08775500139164
```

For more information about `fminbnd`: `>> help fminbnd`

Answer 2.6: Numerical determination of a zero

a) `x0 =`

```
1.259931887274389
```

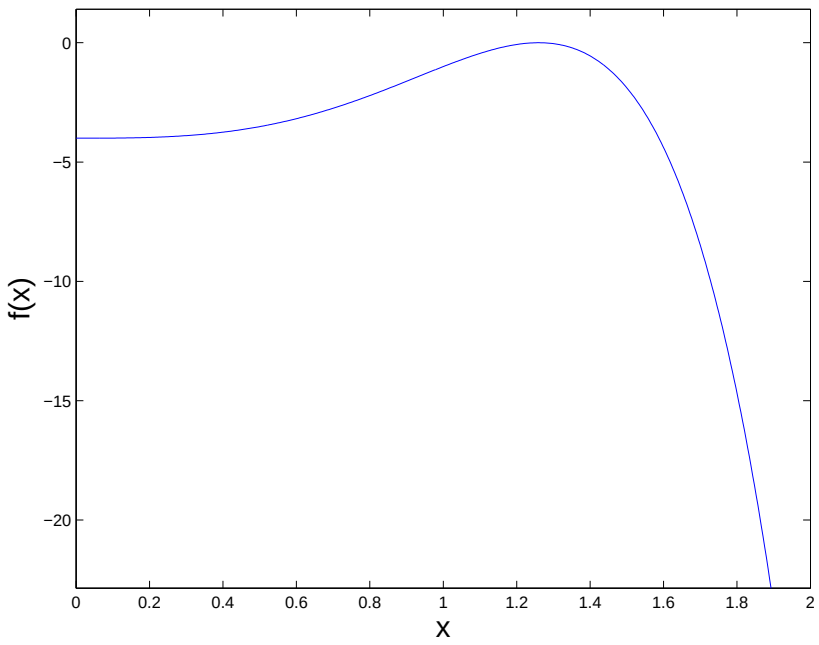
b) Exact zero:

$x_0 = 2^{\frac{1}{3}}$

Comparison between numerically obtained zero and the exact zero:

`Delta_x0 =`

```
1.083737951534936e-005
```



Answer 2.7: Numerical determination of zeroes and extrema

a) A possible suitable interval to depict the extrema and the zeroes of the function is: [-10,2].

```
>> ezplot('f2_7',[-10 2]); grid on;
```

b) An interval is suitable when it contains all extrema and zeroes. With this interval this can be checked by extending the left and right bound of the interval. (HINT: Make sure that enough digits are displayed).

```
f2_7(-10) = 1.99981840234210
f2_7(-100) = 2
f2_7(10) = 4.850770915466111e+008
f2_7(100) = 7.225973768125749e+086
```

At the left bound the function approaches 2 asymptotically, while at the right bound it reaches to infinity. The extrema are in between, which is clear when using above mentioned interval.

c) The position of the zeroes can be determined by choosing the right intervals. Next, the command `fzero` can be used:

```
>> np1 = fzero('f2_7',[-2,0])
np1 = -0.53479999673957

>> np2 = fzero('f2_7',[0,2])
np2 = 1.22794717729952
```

From the graph, it can be seen that the extremum is a minimum at:

```
>> position_minimum = fminbnd('-f2_7',0,2)
```
with value
```
>> minimum = f2_7(position_mimimum)
```

Answer 2.8: Numerical integration

```
>> quad('x.^3+x.^2+(1./(x.^2))',1,3,1e-6)


ans =

  29.333333386482213
```

When choosing a smaller value for the tolerance the calculation will be slower but more accurate. With that the comparison between the result and the exact solution ($\frac{88}{3}$) will be better. When increasing the tolerance the opposite is true.

Answer 2.9

a) $f(x) \geq 0$ for x close to 0 and $f(0) = 0$. So, $f(x)$ has a minimum at $x = 0$.

b) $f(0) = 0$ and $f(x) \to 0$ when $x \to \infty$. Moreover, $f(x) \neq 0$ on the interval $[0,\infty]$. So, $f(x)$ must have a maximum somewhere between 0 and ∞.

c) The maximum can be found by using `fminbnd`. This is done in the same way as in Demo 2.3. For representing infinity take a high number, for instance:

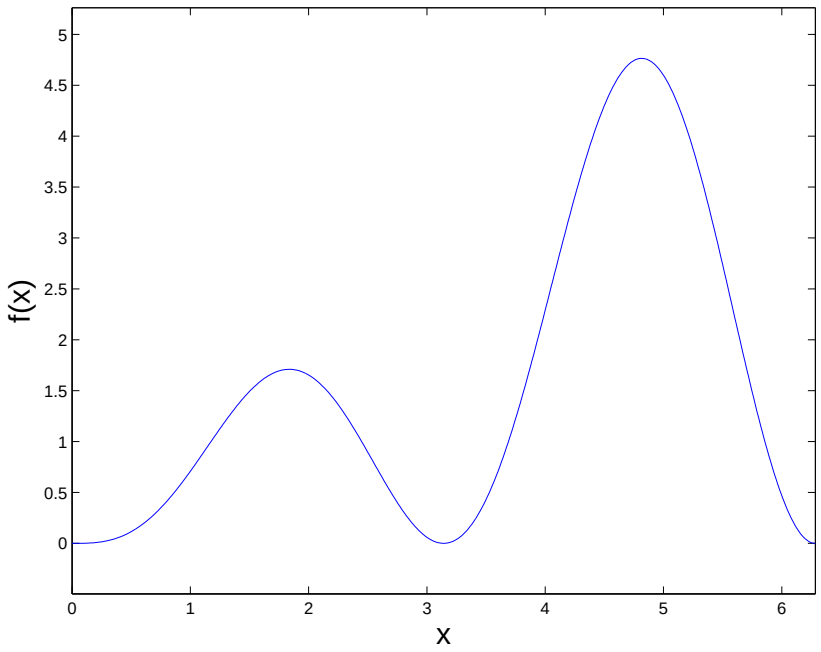
```
>> positionmax = fminbnd('f2_9(x)',0,1000)
>> maximum = f2_9(positionmax)
```

The following values should appear:
positionmax = 2, with corresponding value:
maximum = 0.5413.

d) Determine the real maximum by calculating the derivative f' and then solving $f' = 0$. Next, substitute the x-value you have found in the function f. This will give the value of the real maximum. Compare this value with your answer at question c by using more digits.

Answer 2.10

a)



b)

```
>> positionmax1 = fminbnd('f2_10(x)',0,2)
>> maximum1 = f2_10(positionmax1)
```

```
>> positionmax2 = fminbnd('f2_10(x)',pi,2*pi)
>> maximum2 = f2_10(positionmax2)
```

c) `>> quad('x.*sin(x).^2',[0 2*pi])`

Answer 2.11

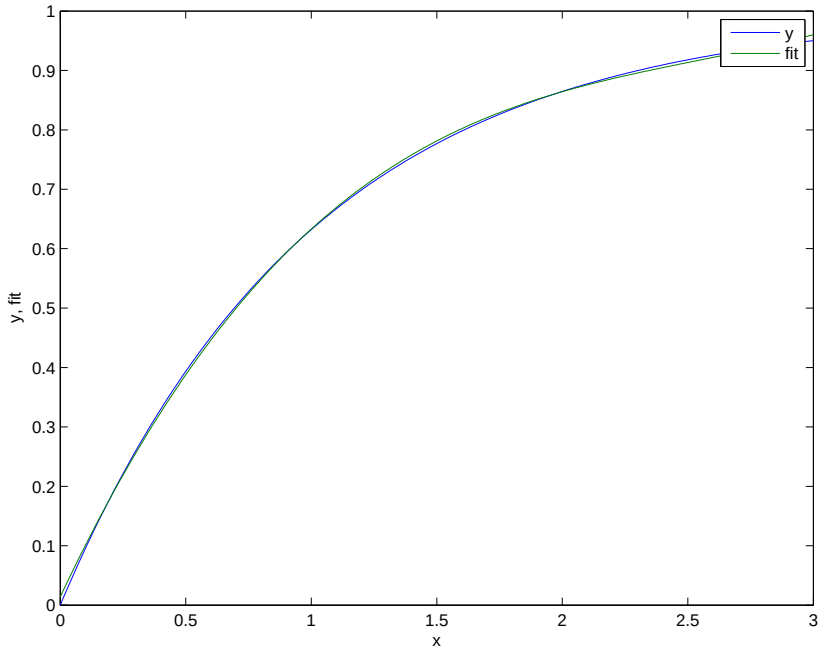
First define the coefficients of the polynomial and the values x.

```
>> p=[1 0 0 -1 8 10];
>> x = [-3, -1.2, 1, 1.1];
```

Then use polyval to evaluate:

```
>> polyval(p,x)

ans =

 -266.0000   -3.5283   18.0000   19.2005
```

Answer 2.12



Coefficients:

```
0.0421   -0.3203    0.8973    0.0139
```

B.3 Answers chapter 3

Answer 3.1: Removing variables

You can delete all variables in the workspace with the command `clear`.
This command can be used in several ways which are explained below.

1. `clear` removes all variables from the workspace.
2. `clear variables` does the same thing.
3. `clear all` removes all variables, globals, functions and MEX links.

Answer 3.2: Numerical determination of an integral

```
ans =

    2.0079
```

Answer 3.3: Symbolic determination of extrema

The extrema of the function f on the interval $[0, \pi]$ can be calculated with the help of a figure `ezplot` and the MATLAB command `fminbnd`. INPUT SUMMARY:

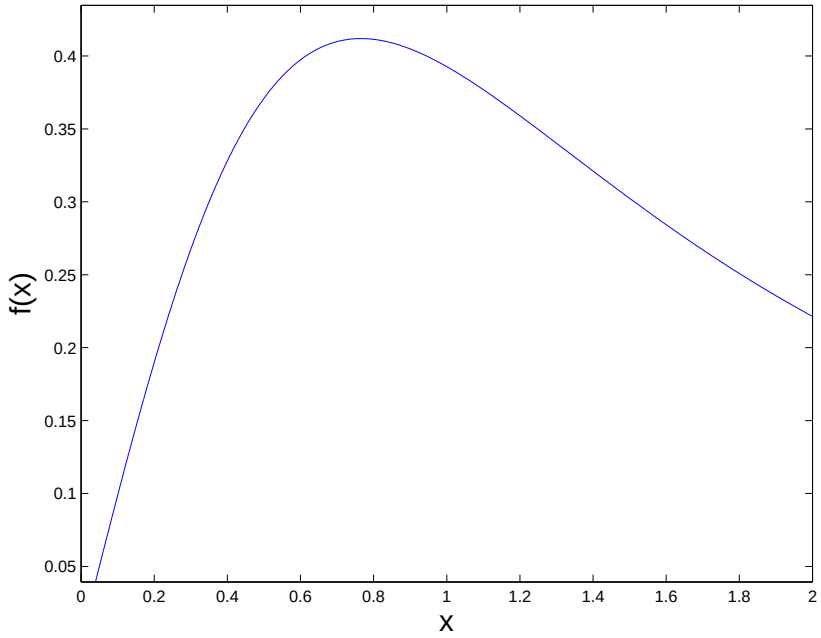
```
>> syms x
>> y = 2*x*exp(-x)-sin(x)
>> figure
>> ezplot(y,[0 pi])

>> xmin = fminbnd(char(y),1.5,2)
>> ymin = subs(y,x,xmin)
>> xmax = fminbnd(char(-y),0,0.5)
>> ymax = subs(y,x,xmax)
>> xnp1 = fzero(char(y),[0.5,1])
>> ynp1 = subs(y,x,xnp1)
>> xnp2 = fzero(char(y),[2.5,3])
>> ynp2 = subs(y,x,xnp1)
```

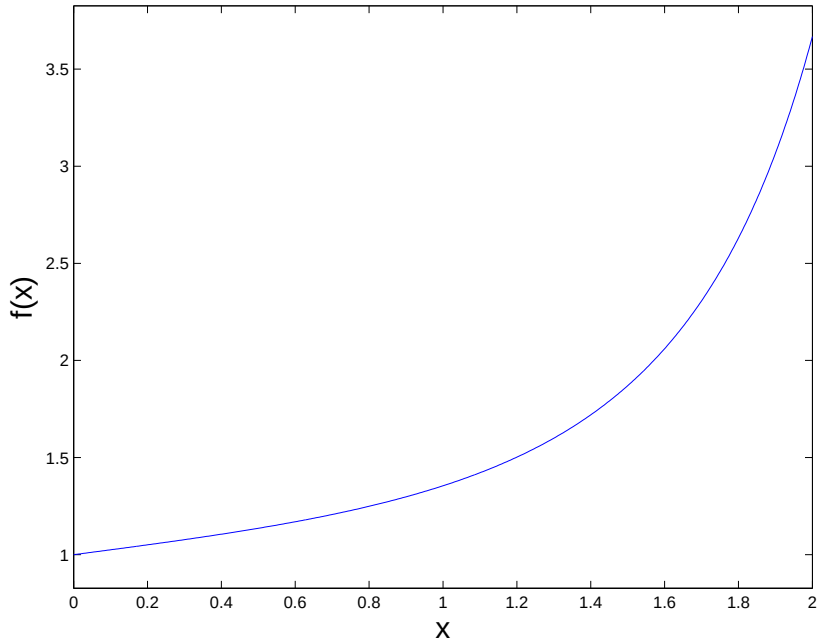
WARNING: The values of xnp and ynp can depend on the properties of your MATLAB version.

Answer 3.4

a)

b)





Answer 3.5

The difference in output between the commands `x^1/3` and `x^(1/3)` is caused by the absence of brackets in the first case and the math rules that MATLAB operates with.

The order of math rules that MATLAB operates with is as follows:

The exercise `>> x^1/3` is calculated as: $(x\text{^}1) * 1/3$
The exercise `>> x^(1/3)` is calculated as: $x\text{^}(1/3)$

Answer 3.6

```
>> syms x y
>> q = x^2y^3+sin(pi*x*y)
>> q = subs(q,x,2)
>> q = subs(q,y,3)
```

Answer 3.7

The aim of this exercise is to show you that sometimes it can occur that MATLAB presents a difficult formula on the screen while in reality the function is very simple.

The correctness of the integral can be examined by differentiating the output of the integral ('`diff`'). This output is divided in two parts; a numerator and denominator ('`numden`'). Canceling out the common terms and a further working out of the denominator ('`expand`') makes clear that b actually equals $\frac{1}{(1+x^4)}$.

INPUT SUMMARY:

```
>> syms x
>> y = int(1/(1+x^4),x)
```

Differentiating cancels integrating

```
>> b = diff(y,x)
>> [t,n] = numden(b-1/(1+x^4))

t = 0
n = 16*(x^2+x2^(1/2)+1)^3*(x^2-x*2^(1/2)+1)^4
```

Conclusion: Differentiating of the integral actually results in $\frac{1}{(1+x\text{^}4)}$.

Answer 3.8

a) `inta =`

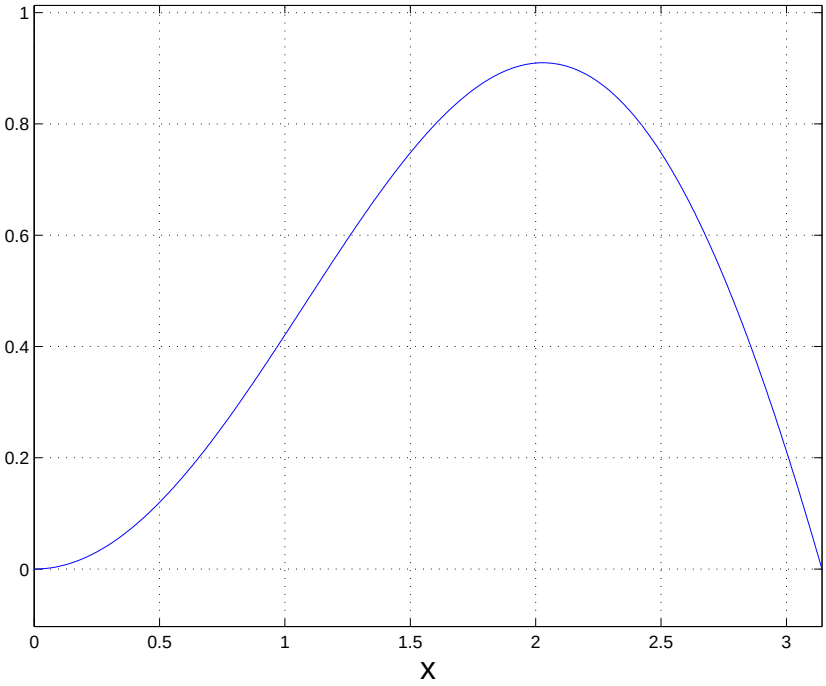
```
    1.5708
```

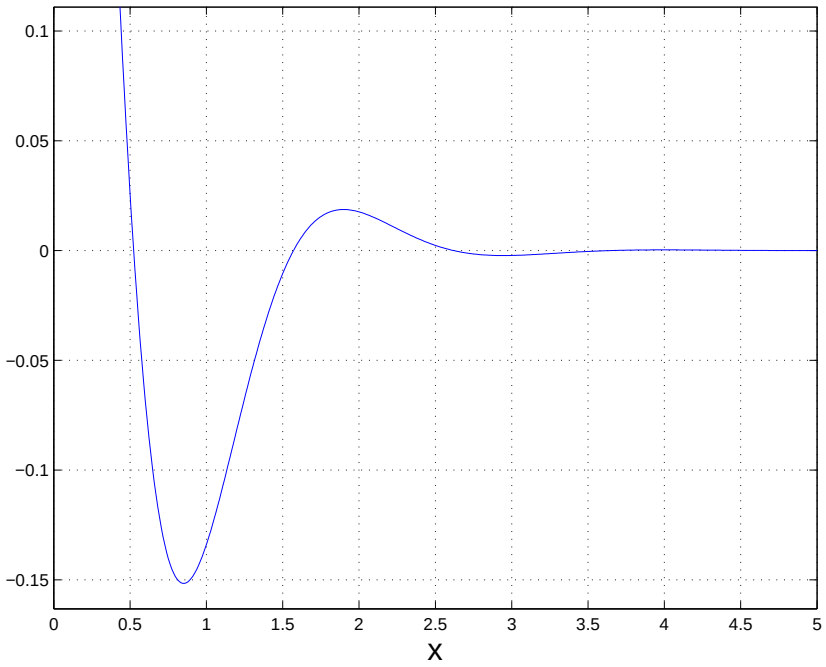
b) `intb =`

```
    0.1538
```

c) `intc =`

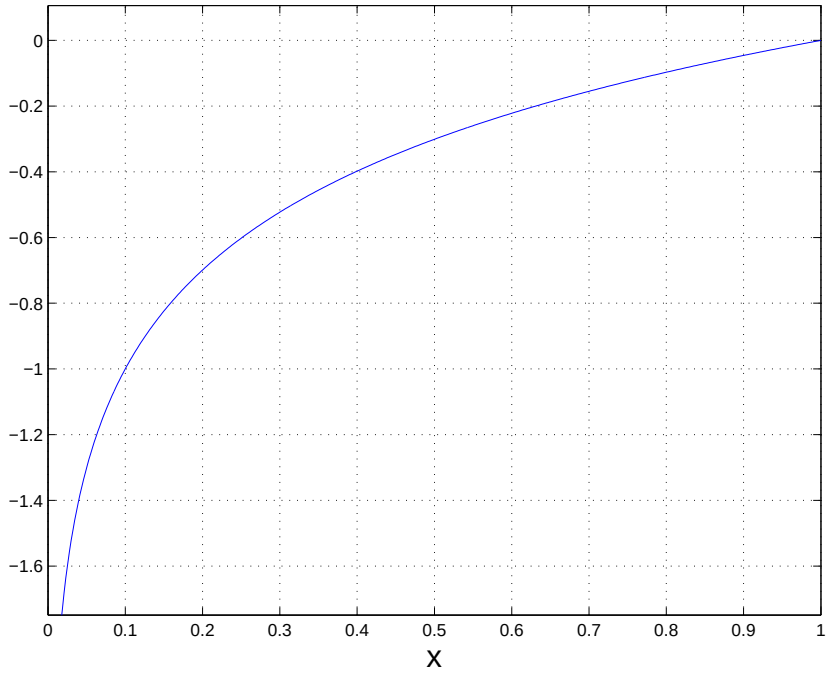
```
   -0.4343
```

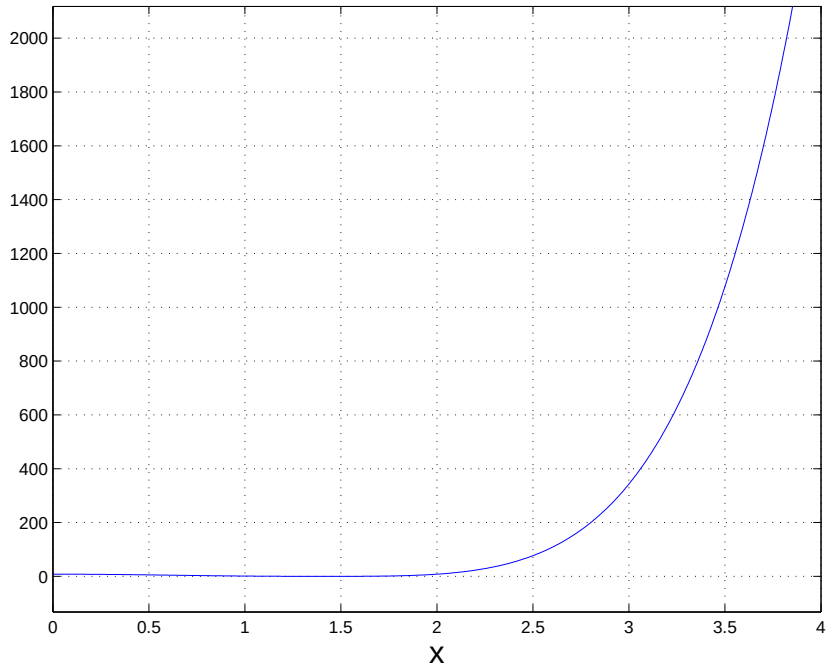




d) `intd =`

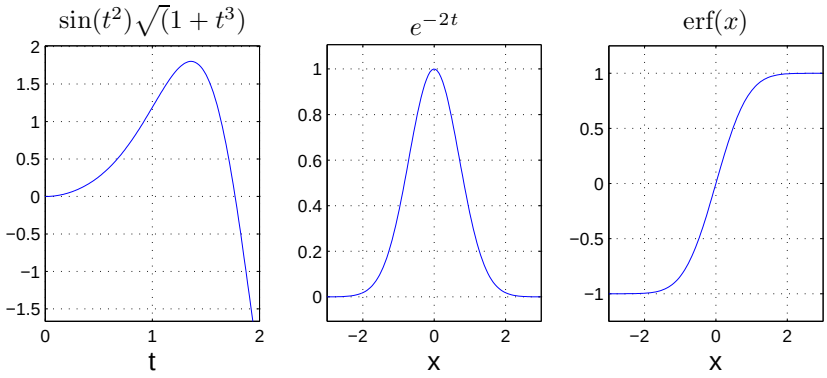
`1.3461e+003`





Answer 3.9

The integrals of part a) and b) can be calculated by the following commands. The erf-function which is present in answer b) is given in the 3rd figure.



a) INPUT SUMMARY:

```
>> syms t
>> ya = sin(t^2)*sqrt(1+t^3);
>> inta = int(ya,t,0,2);
>> figure
>> ezplot(ya,[0 2])
```

MATLAB returns the command, because this integral can not be determined exact.

b) INPUT SUMMARY:

```
>> syms x
>> yb = exp(-x^2);
>> intb = int(yb,x);
>> figure
>> ezplot(yb,[-3 3])
```

MATLAB gives a result including an error function. Since this function can not be expressed in elementary function, the integral of part b) can not be calculated by MATLAB.

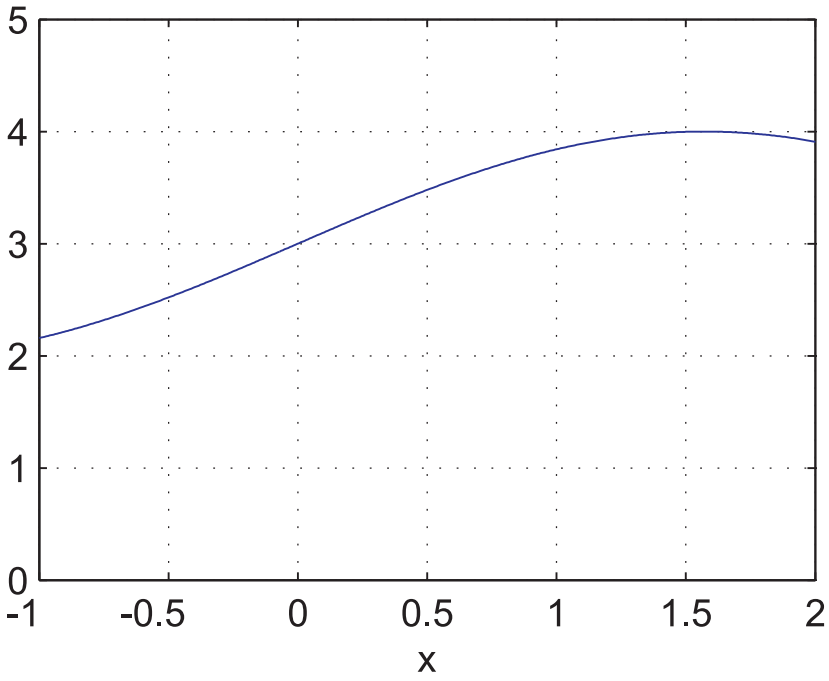
Display of the error function: `>> yc = erf(x);`

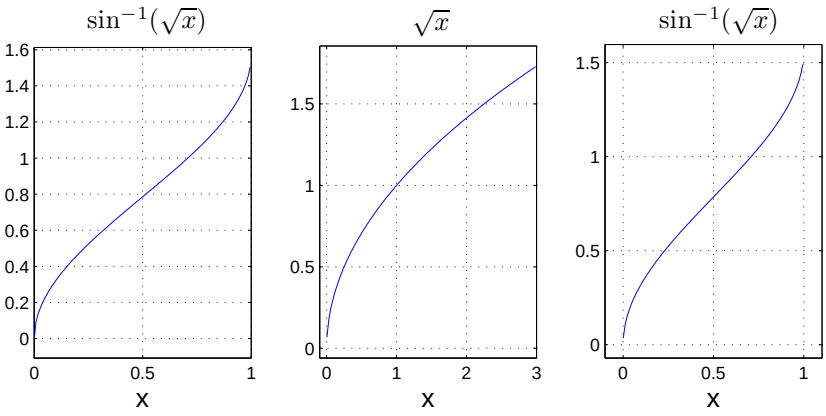
```
>> figure;
>> ezplot(yc,[-3 3])
```

Answer 3.10

Answer 3.11

The results of the particular MATLAB commands is shown in the first figure. De graph is only given on the interval $0 \leq x \leq 1$, because the function asin(x) is defined for $-1 \leq x \leq 1$ and the function $\sqrt{x}$ for $x \geq 0$. A better drawing of the function can be made by adjusting





the domain of x into [-0.1, 1.1] in stead of [0, 1].

INPUT SUMMARY:

```
>> syms x
>> figure
>> ezplot(asin(sqrt(x)),[0 1])

>> ezplot(asin(x),[-1.1 1.1])
>> ezplot(sqrt(x),[-0.1 3])

>> ezplot(asin(sqrt(x)),[-0.1 1.1])
```

Answer 3.12

a) The result is not valid for $n = -1$.

b) `ans =`

```
log(x)
```

This answer is not correct. It should be `log(abs(x))`. For positive x the answer is correct.

Answer 3.13

The 'symbolic toolbox' uses a standard notation. It denotes a square root as ^(1/2). Hence, sqrt(x^2) is rewritten as (x^2)^(1/2).

Notice that, in general, sqrt(x^2) equals abs(x) and not just x. Therefore, MATLAB holds on to the square root.

INPUT SUMMARY:

```
>> syms x
>> sqrt(x^2)

ans =
    (x^2)^(1/2)
```

Answer 3.14

a) `dy =`

```
(1/(1+x)+x)/(1+x)^2-2*(log(1+x)+1/2*x^2)/(1+x)^3
```

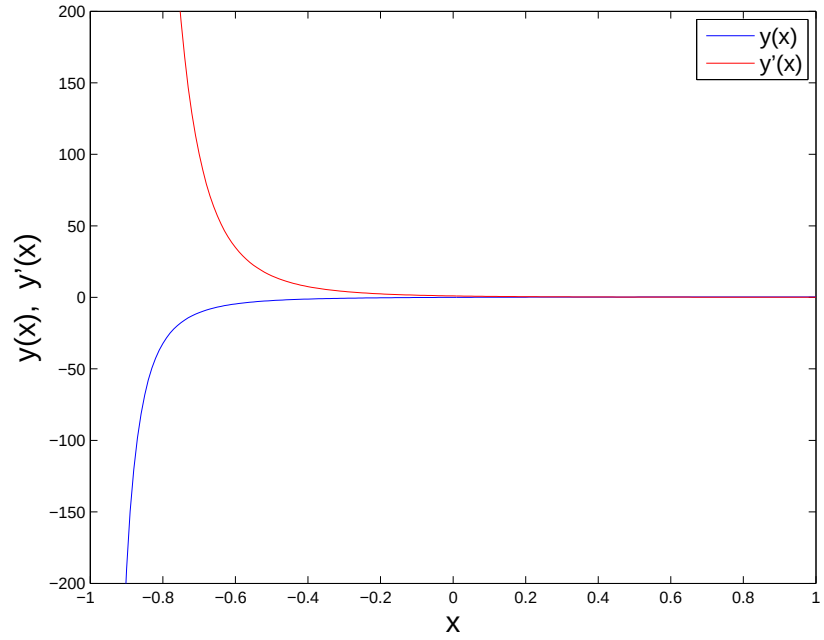
b) `ans =`

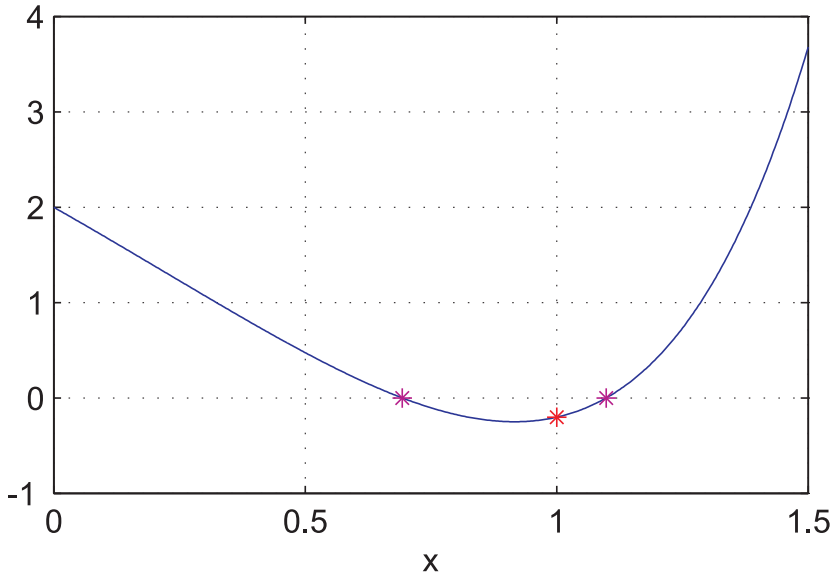
```
    0.0767
```

c)

Answer 3.15

Using the command `ezplot` you can draw symbolic functions. The minimum of the function can be determined with the command `fminbnd`. The position, function value and value of the derivative are 1.001, -0.23023 and 1.1876 respectively. The function f has two zeroes (see figure). The coordinates of the zeroes are: (1.0986,0) and (0.6931,0).





a) The graph of the function on the interval [0, 1.5] gives all information. Notice that $f(x) \to 6$ if $x \to -\infty$ and that $f(x) \to \infty$ if $x \to \infty$.

```
>> syms x
>> y = exp(2*x)-5*exp(x)+6
>> figure
>> ezplot(y,x,[0 1.5])
```

b) The minimum has its position at $x = 0.9163$ and has the value -0.2500.

```
>> xmin = fminbnd(char(y),1,2)
```

c) The function value and the value of the derivative can be found with:

```
>> ymin = subs(y,x,xmin)
>> dy = diff(y,x)
>> dymin = subs(dy,x,xmin)
```

d) The function f has two zeroes:

```
>> syms x
>> y = exp(2*x)-5*exp(x)+6
>> x1 = fzero(char(y),[xmin,xmin+1])
>> x2 = fzero(char(y),[xmin-1,xmin])
```

The number of zeroes can also be determined by means of the command `solve`:

```
>> zeroes = solve(y)
```
or
```
>> zeroes = solve(exp(2*x)-5*exp(x)+6=0)
```

Answer 3.16

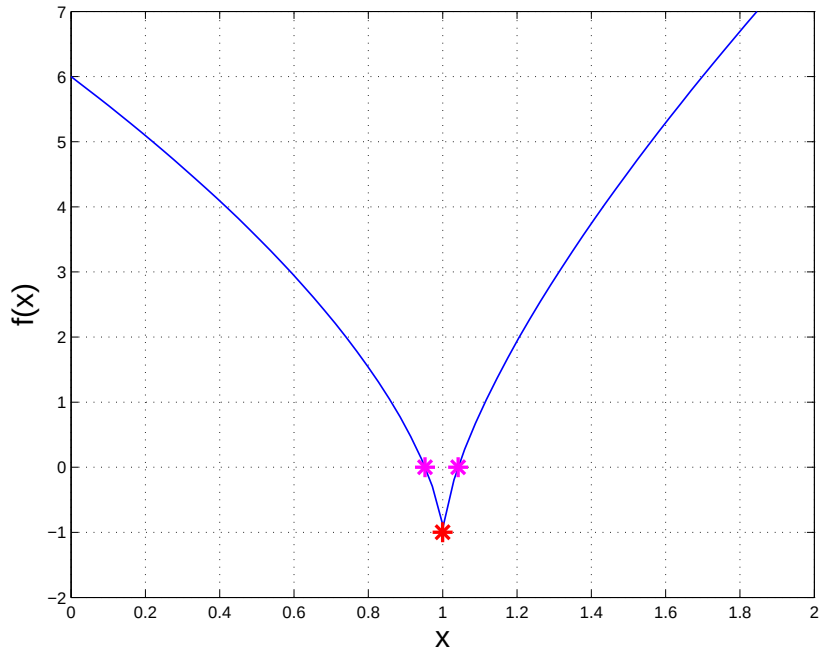
a) No answer

b)
```
xmin =

    1.0000

ymin =

   -0.9993

zero_1 =

    1.0415

zero_2 =

    0.9526
```

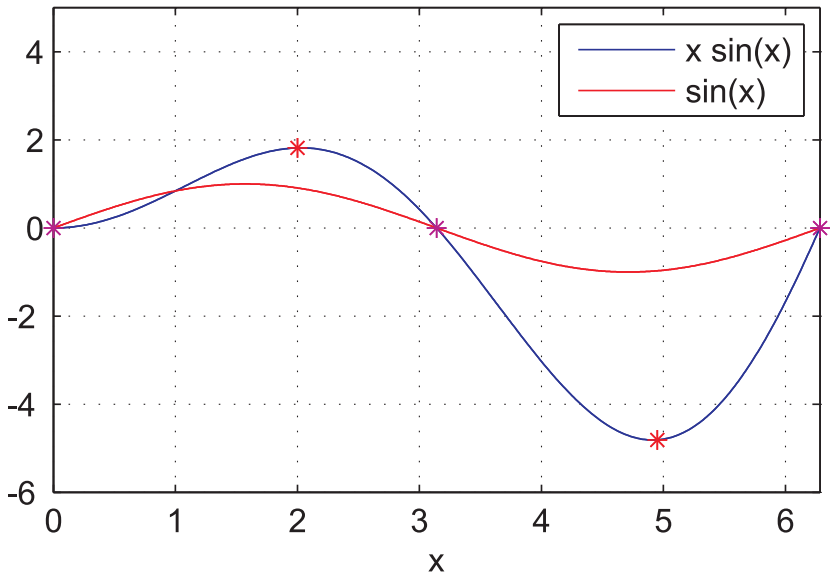
Answer 3.17

Before you are going to determine the minimum, maximum and zeroes, it is wise to draw the function first using `ezplot`. See the figure.
The x-coordinate of the minimum can be calculated with the command `fminbnd`. You can

c)



obtain the y-coordinate by substituting the x-coordinate in the function using the command `subs`.
The x-coordinate of the maximum is calculated in the same way as the minimum, but you are supposed to replace y for $-y$. The y-coordinate is again obtained by substitution.
The x-coordinates of the zeroes are obtained using `fzero`.
You can add the function $\sin(x)$ to the former figure using `plot` and `hold on`. The extrema of f are placed somewhat to the right compared to $\sin(x)$. This is because, for instance, around $x = \frac{\pi}{2}$, $\sin(x)$ hardly changes, but $x\sin(x)$ increases.



SUMMARY OF INPUT:

```
>> clear all
>> syms x
>> y = x*sin(x)
>> figure
>> ezplot(y,[0 2*pi]); grid;

>> xmin = fminbnd(char(y),2,2.05)
>> ymin = subs(y,x,xmin)
>> xmax = fminbnd(char(-y),4.9,4.95)
>> ymax = subs(y,x,xmax)

>> xzero1 = fzero(char(y),[0,0.2])
>> xzero2 = fzero(char(y),[pi-1,pi+1])
>> xzero3 = fzero(char(y),[2*pi-1,2*pi+1])

>> hold on
>> plot(xmin,ymin,'r*')
>> plot(xmax,ymax,'r*')
>> plot(xzero1,0,'m*',xzero3,0,'m*',xzero3,0,'m*')

>> xs = 0:0.01:2*pi;
>> ys = sin(xs)
>> hold on
>> plot(xs,ys,'r')
```

Answer 3.18

a)

b) The function and the line do not intersect each other there, because $|\sqrt{x}\sin(\frac{1}{x})| < \sqrt{x} < \frac{1}{4}$ when $x < 1/16$.

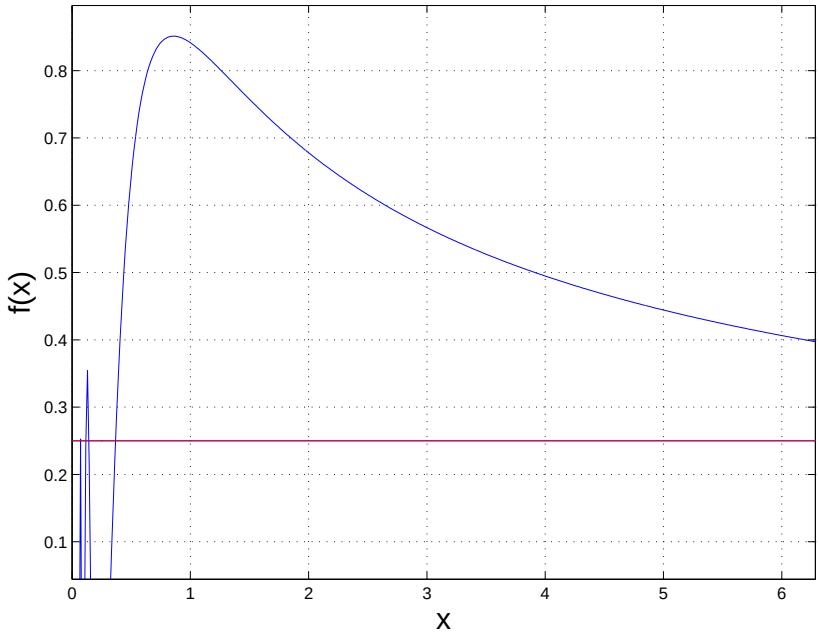
c) There are 5 solutions.

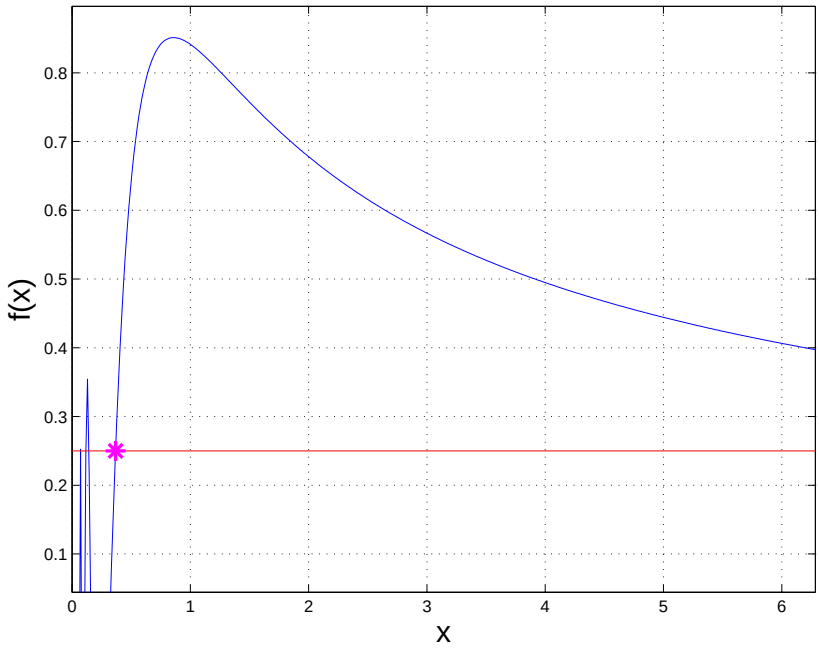
d)
```
intersection =

    0.3681    0.2500
```

Answer 3.19

The positive constant a, for which the equation $e^x = ax(1-x)$ has exactly one solution between 0 and 1, is in this answer determined graphically. The following approach is used:

1. Draw the function e^x.
2. Draw the function $ax(1-x)$ for several values of a.





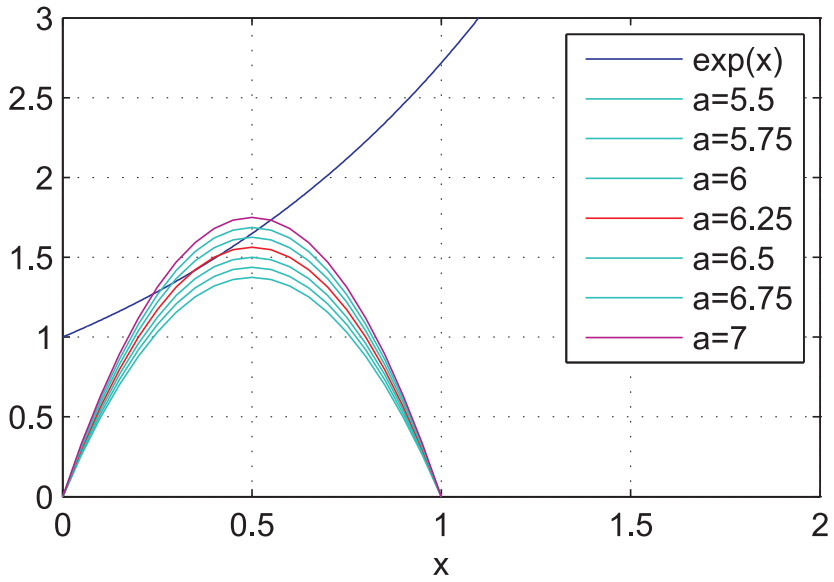
3. Search for the value of a with just one tangent point.

The final approximation for a is $a = 6.25$.

SUMMARY OF INPUT:

```
>> clear all
```

```
>> syms x
>> y1 = exp(x);
>> x2 = 0:0.05:1;
>> figure
>> ezplot(y1,[0,1])

>> for a=5.5:0.25:7
>>    y2 = a*x2.*(1-x2);
>>    hold on
>>    if a==7
>>        plot(x2,y2,'m')
>>        legend('exp(x)','a=5.5','a=5.75','a=6',...
                'a=6.25','a=6.5','a=7')
 >>   elseif a==6.25
>>        plot(x2,y2,'r')
>>    else
>>        plot(x2,y2,'c')
>>    end
>>end
```



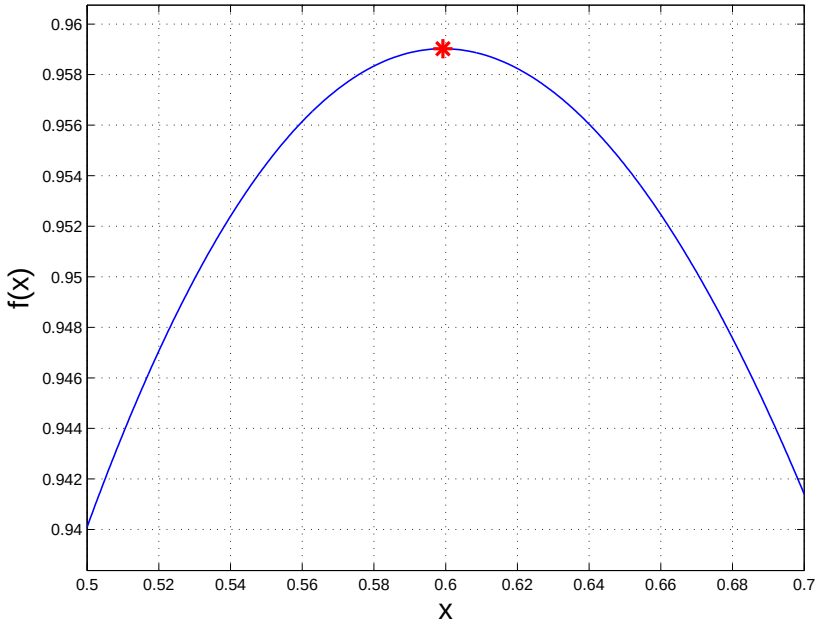
Answer 3.20

a) `x0 =`

`0.5992`

b) `ans =`

`-5.8133e-006`



Answer 3.21

The derivative of a function can be obtained using the command `diff`.
When you are using the 'symbolic toolbox', sometimes it is useful to simplify functions. You can do this with the command `simplify`.
In this way you can simplify the function $\sin(\frac{x^2+2x-3}{x-1})$ to a much more easier expression $\sin(x+3)$.
Using this expression, it is easy to determine the 10^{th} derivative, which is $-\sin(x+3)$.

SUMMARY OF INPUT:

```
>> clear all
>> close all
>> syms x y
>> y = sin((x^2+2*x-3)/(x-1))
>> y = simplify(y)
>> dyten = diff(y,x,10)
```

	x	y
Minimum	1.8409	-0.3795
Maximum	0.3384	0.1505
Zero 1	0.8030	0
Zero 2	2.7923	0

1. ()	Between brackets
2. ^	Raising to power
3. *	Multiplying
4. /	Dividing
5. sqrt	Extracting roots
6. +	Addition
7. −	Subtraction

B.4 Answers chapter 4

Answer 4.1

You can enter the matrix A in different ways. Three possibilities are as follows:

```
>> A = [1 2 -1 3; 7 2 0 1; 3 -2 -1 -1; 0 1 4 8]

>> A = [1,2,-1,3;7,2,0,1;3,-2,-1,-1;0,1,4,8]

>> A = [1 2 -1 3
        7 2 0 1
        -3 -2 -1 -1]
```

Answers 4.2

A matrix element can consist of all MATLAB elements. A good example of a MATLAB element is the square root of 3. The square root of 3 can be entered as `sqrt(3)` or as `3^(1/2)`.
INPUT SUMMARY:

```
>> B = [1.5 2 (1/7); sqrt(3) 2 0.625]
```

Answer 4.3

Matrices can be concatenated in two different ways:

1. Concatenate as column: `d = [A;B]`
   The matrix d is composed out of the matrix A and the addition of the matrix B as 3$^{\text{rd}}$ and 4$^{\text{th}}$ row. Hence, `d = [A;B] = [1 2; 3 4; 5 6; 7 8]`.

2. Concatenate as row: `d = [A B]`
   The matrix d is composed out of the matrix A and the addition of the matrix B as 3$^{\text{rd}}$ and 4$^{\text{th}}$ column. Hence, `d = [A B] = [1 2 5 6; 3 4 7 8]`.

INPUT SUMMARY:

```
>> A = [1 2; 3 4]
>> B = [5 6; 7 8]
>> C = [1 2; 1 2; 1 2; 1 2]
>> d = [A; B]       % concatenate as column
>> D = [d C]        % concatenate as row
% or concatenate in one command:
>> D = [[A;B] C]
```

Answer 4.4

Symbolic objects can be entered as matrix elements. In order to enter matrix elements a_{11}, a_{12}, a_{13}, a_{21}, a_{22} and a_{23} you should declare them as symbolic objects first using the command `>> syms`.

INPUT SUMMARY:

```
>> syms a11 a12 a13 a21 a22 a23
A = [a11 a12 a13; a21 a22 a23]
```

Answer 4.5

In principle, a row vector is a matrix with one row. Vectors are entered in the same way as matrices. However, when the step between successive elements of the row vector are the same, you can generate such a row vector easily by entering:

'variable' = 'initial value':'step size':'final value'

INPUT SUMMARY:

```
>> a = -1:0.5:9
```

or between brackets, since we are making a vector:

```
>> a = [-1:0.5:9]
```

Answer 4.6

In principle, a row vector is a matrix with one row. Vectors are entered in the same way as matrices. However, when the step between successive elements of the row vector are the same, you can generate such a row vector easily by entering:

'variable' = 'initial value':'step size':'final value'

Note that the step size should be a negative number for the row vector in this exercise.

INPUT SUMMARY:

```
>> a = [9:-1:0]'
```

Answer 4.7

Since MATLAB assumes that symbolic objects can have a complex value, MATLAB computes the conjugated values when taking the transpose. When you declare these symbolic objects as real numbers, MATLAB knows that these variables can only have a real value. Therefore, there are no problems anymore when taking the transpose and other matrix transformations.

INPUT SUMMARY:

```
>> syms a
```

```
>> v = a:2:a+20;

% taking the transpose
>> vtransp1 = v'    % results in complex numbers
>> vtransp2 = v.'   % results in real numbers only

>> sym
>> a = sym('a','real')
>> v = a:2:a+20;
>> vtransp3 = v'    % results in real numbers only
```

Answer 4.8

Some special matrices can be entered quite efficient in MATLAB by means of commands like `eye`, `zeros`, `ones`, `rand`, and `diag`.

INPUT SUMMARY:

```
>> a = eye(6)
>> b = zeros(5,10)
>> c = ones(5,15)
>> d = rand(5)

>> e0 = 1:5
>> e = diag(e0)     % puts the elements of the vector e0
                    % on the diagonal of the matrix e
```

Answer 4.9

INPUT SUMMARY:

```
>> A = [1 2 -1 3; 7 2 0 1; 3 -2 -1 -1; 0 1 4 8]
>> A(3,2)=3;
>> A(3,[2 4])
ans =
    -2    -1

>> A([1 2 3],[2 4])
ans =
     2     3
     2     1
    -2    -1

>> A(1:3,[2 4])
ans =
     2     3
```

```
     2     1
    -2    -1

>> A(2,:)-7*A(1,:)
ans =
     0   -12     7   -20

>> A(2,:) = A(2,:)-7*A(1,:)
A =
     1     2    -1     3
     0   -12     7   -20
     3    -2    -1    -1
     0     1     4     8
```

Answer 4.10

Within the matrix A, the 2×2 submatrix, formed by the elements having indices (1,1), (1,3), (3,1) and (3,3), is replaced by the 2×2 matrix which has all elements valued 10.

INPUT SUMMARY:

```
>> A = [1 2 -1 3; 7 2 0 1; 3 -2 -1 -1; 0 1 4 8]
A =
    1     2    -1     3
    7     2     0     1
    3    -2    -1    -1
    0     1     4     8

>> A([1 3],[1 3]) = 10*ones(2)
A =
    10     2    10     3
     7     2     0     1
    10    -2    10    -1
     0     1     4     8
```

Answer 4.11

INPUT SUMMARY:

```
>> A = [1 2 -1 3; 7 2 0 1; 3 -2 -1 -1; 0 1 4 8]
A =
    1     2    -1     3
    7     2     0     1
    3    -2    -1    -1
    0     1     4     8
```

```
>> B = A'
B =
    1     7     3     0     % b11 = a11, b12 = a21, b13 = a31, ....
    2     2    -2     1
   -1     0    -1     4
    3     1    -1     8

>> C = A*B
C =
    15    14    -3    22    % c11 = a11*b11+a12*b21+a13*b31+a14*b41
    14    54    16    10
    -3    16    15   -14
    22    10   -14    81

>> D = B*A
D =
    59    10    -4     7    % d11 = b11*a11+b12*a21+b13*a31+b14*a41
    10    13     4    18    % so A*B is not the same as B*A
    -4     4    18    30
     7    18    30    75

>> E = 3*A+5*B^3
E =
    628         731         -58         594
    281         591         -50         238
    489         524          97         982
    1375        1668        -153        2619
```

Answer 4.12

INPUT SUMMARY:

```
>> v = [1 2 3 4]
>> A = [1 2 -1 3; 7 2 0 1; 3 -2 -1 -1; 0 1 4 8]
```

a)
```
>> A1 = v*A     % a1_11 = v1*a11+v2*a21+v3*a31+v4*a41
A1 =
   24     4    12    34

>> A2 = A*v'    % a2_11 = a11*v1+a12*v2+a13*v3+a14*v4
A2 =
   14
   15
   -8
   46
```

b)
```
>> D = diag(v);
```

```
>> B1 = D*A
B1 =
     1     2    -1     3    % b1_11 = d11*a11+0*a21+0*a31+0*a41
    14     4     0     2    % b1_21 = 0*a11+b22*a21+0*a31+0*a41
     9    -6    -3    -3
     0     4    16    32

>> B2 = A*D
B2 =
     1     4    -3    12    % b2_11 = a11*d11+a12*0+a13*0+a14*0
     7     4     0     4    % b2_21 = a21*0+a22*b22+a23*0+a24*0
     3    -4    -3    -4
     0     2    12    32
```

Answer 4.13

The multiplications in this exercise are regular matrix multiplications.

INPUT SUMMARY:

```
>> v = [1 2 3 4]
>> a = v*v'          % a = v1*v1+v2*v2+v3*v3+v4*v4
a =
    30

>> b = v'*v          % b11 = v1*v1, b21 = v2*v1, ...
b =
    1     2     3     4
    2     4     6     8
    3     6     9    12
    4     8    12    16
```

Answer 4.14

INPUT SUMMARY:

```
>> a = sym('a','real');
>> b = sym('b','real');
>> c = sym('c','real');
>> A = [1 2 a; b -3 2; 3 1 c]
A =
    [  1,  2,  a]
    [  b, -3,  2]
    [  3,  1,  c]
```

a) `>> B = A'`

```
B =
[  1,  b,  3]                      % b11 = a11, b12 = a21, b13 = a31, ...
[  2, -3,  1]
[  a,  2,  c]
```

b)
```
>> C = A*B
C =
[     5+a^2,   b-6+2*a,     5+a*c] % c11 = a11*b11+a12*b21+a13*b31+a14*b41
[   b-6+2*a,    b^2+13, 3*b-3+2*c]
[     5+a*c, 3*b-3+2*c,    10+c^2]
```

c)
```
>> D = B*A
D =
[    10+b^2,     5-3*b, a+2*b+3*c] % d11 = b11*a11+b12*a21+b13*a31+b14*a41
[     5-3*b,        14,   2*a-6+c] % so A*B is not the same as B*A
[ a+2*b+3*c,   2*a-6+c, a^2+4+c^2]
```

d)
```
>> E = 3*A+5*B^3
E =
    e11 = 68+10*b+30*a+5*b*(-4+a)+15*a*c
    e12 = -54-10*b+5*b*(2*b+11)+15*a*b+30*c
    e13 = ...
```

Answer 4.15

This exercise is a nice example of a linear system $Ax = b$ with a unique solution x. Producing the reduced row echelon form of the matrix [A b] using the Gauss Jordan algorithm results in a matrix which consists of two parts:

1. a diagonal matrix with the same size as matrix A,
2. a vector with the same size as vector b which is the solution of the linear system.

INPUT SUMMARY:

```
>> A = [1 2 3; 2 -3 2; 3 1 -1];
>> b = [6 14 -2]';
>> x = A\b
x =
    1.0000
   -2.0000
    3.0000

>> A*x          % Check your solution. The result of A*x equals the vector b,
                % so x is a solution of the linear system Ax = b.

>> rref([A,b])  % The solution is unique, since the reduced row echelon form
                % consists of a unity matrix and the solution vector.
```

Answer 4.16

This exercise is an example of a linear system with multiple solutions.

INPUT SUMMARY:

```
>> clear all
>> A = [1 2 3; 2 1 -3];
>> b = [-4 4]';
>> x = A\b
x =
         0
   -1.5385
   -0.3077

>> A*x          % Check your solution. It turns out that x is a valid solution.

>> rref([A,b])  % The linear system Ax = b has multiple solutions. This follows
                % the presence of free variables and the fact that the linear
                % system has more free variables than equations.

>> AA = sym(A)
>> bb = sym(b)
>> xx = AA\bb
xx =
     5          % This is another solution to the linear system.
     0
    -3
```

Answer 4.17

This exercise is a nice example of an overdetermined system. Producing the reduced row echelon form shows this. The command `A\b` results in a vector x, but this vector is not a valid solution. This can be checked by entering `A*x`.

INPUT SUMMARY:

```
>> clear all
>> A = [1 0; 1 2; 1 4];
>> b = [10 16 17]';
>> rref([A,b])
>> x = A\b
x =
   10.8333
    1.7500

>> A*x          % Check your solution. The vector x is not a valid solution,
```

```
                % because the result of A*x does not equal b

>> AA = sym(A)
>> bb = sym(b)
>> xx = AA\bb

Warning: System is inconsistent. Solution does not exist.
> In sym.mldivide at 34

xx =
    Inf
    Inf
```

Answer 4.18

This exercise illustrates how you can solve a system of linear equations which has multiple solutions. You should use the command `solve` from the Symbolic Math Toolbox.
The general solution in MATLAB vector notation looks like: `x = [x1; -5*x1+16; -3*x1+12]`

INPUT SUMMARY:

```
>> [x1,x2,x3] = solve('x1+2*x2-3*x3=-4','2*x1+x2-3*x3=4','x1','x2','x3')

Warning: 2 equations in 3 variables.
> In solve at 113
x1 =
    4+x3
x2 =
    -4+x3
x3 =
    x3
```

Answer 4.19

The solution of the system $Ax = b$ is not defined for $a = -2$. After substitution of $a = -2$ in MATLAB, solving $Ax = b$ yields $x = [\infty, \infty, \infty]^T$.

INPUT SUMMARY:

```
>> A=[1 1 -1; 1 2 1; 1 1 (a^2-5)];
>> b=[2; 3 ; a];
>> Asub=subs(A,a,-2);
>> bsub=subs(b,a,-2);
>> xsub=Asub\bsb
xsub =
    Inf
```

```
Inf
Inf
```

Answer 4.20

Studying the system of equations $Ax = b$ with $a = \sqrt{6}$ yields the solution $x = [1.6742, 0.5505, 0.2247]^T$.

INPUT SUMMARY:

```
>> Asub=subs(A,a,sqrt(6));
>> bsub=subs(b,a,sqrt(6));
>> xsub=Asub\bsb;
>> D=rref([A,b]);
>> xsol=D(:,4);
```

Answer 4.21

a) $A^2 = \begin{bmatrix} 81 & 0 \\ 0 & 1 \end{bmatrix}$, $A^3 = \begin{bmatrix} 729 & 0 \\ 0 & 1 \end{bmatrix}$.

b) $A^n = \begin{bmatrix} 9^n & 0 \\ 0 & 1 \end{bmatrix}$,

c)-e)
```
>> A=[9 0;0 1];
>> A^10
ans =
  1.0e+009 *
    3.4868         0
         0    0.0000
>> format long
>> A^10
ans =
  1.0e+009 *
   3.486784401000000                   0
                   0   0.000000001000000
>> format
```

Answer 4.22

a) $x = [-0.1111, 2.8889, 0.4444, 1.3333]^T$ and $y = [1, 1, 1, 1]^T$

b) $x = [6.0, -7.2, 2.9 - 0.1]^T$ and $x = [1.50.181.190.89]^T$, respectively.
$y = [0.9, 1.0558, 1.0542, 0.9733$ and $y = [0.99, 1.0056, 1.0054, 0.9973]$, respectively.

c) Try yourself.

d) The second system is better conditioned.

Answer 4.23

Vectors x_1 and x_2 are quite different. Apparently, the 5×5 Hilbert matrix is ill-conditioned.

INPUT SUMMARY:

```
>> A=hilb(5)
A =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
>> b1=[2.2833;1.45;1.0929;0.8845;0.7456];
>> x1=A\b1
x1 =
    1.0555
    0.0000
    5.1870
   -5.1800
    3.9690
>> b2=[2.2834;1.4501;1.0928;0.8844;0.7457];
>> x2=A\b2
x2 =
    1.1260
   -1.6080
   12.8940
  -17.6120
   10.3320
```

Answer 4.24

All 2×2 diagonal matrices commute with A.

Answer 4.25

Using the symbolic toolbox, you can check whether two matrices commute. Hereto, start with a symbolic B and derive expressions that should be satisfied for commutivity.

B is commuting with A if and only if $b_{21=0}$ and $3b_{11} + b_{12} - 3b_{22} = 0$.

INPUT SUMMARY:

```
>> syms b11 b12 b21 b22;
>> B=[b11 b12;b21 b22];
>> A=[1 3;0 2];
>> A*B-B*A
```

```
ans=
    [3*b21,   -b12+3*b22-3*b11]
    [b21,     -3*b21          ]
```

Answer 4.26

Three points: $[m_1, m_2, m_3]^T = [0.3636, 0.5455, 0.0909]^T$
Four points: Infinite number of solutions.

Answer 4.27

Conservation of linear momentum yields:

$$m_1 v_1 + m_2 v_2 = m_1 w_1 + m_2 w_2$$

Therefore, we find:

$$m_1 v_1 + m_2 v_2 - m_1 w_1 - m_2 w_2 = 0$$

This yields $m_2 = 2m_1$.

INPUT SUMMARY:

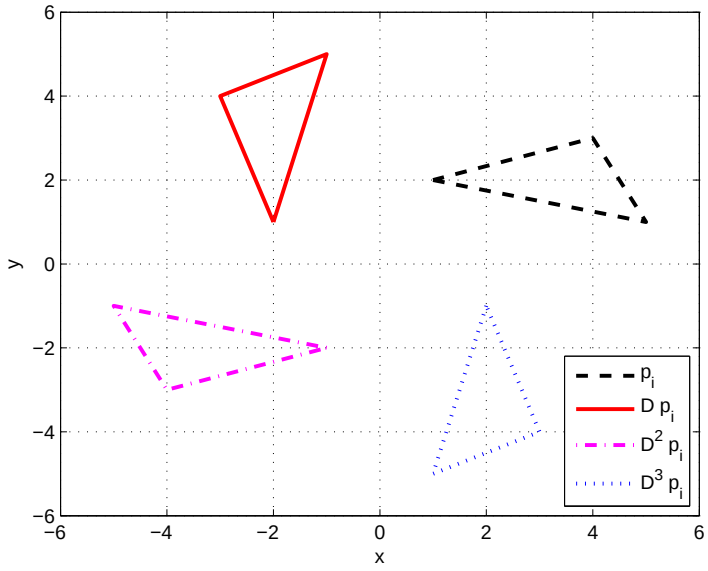
```
>> syms m1 m2;
>> v1=[2 2 2]';
>> v2=[5 7 9]';
>> w1=[4 6 4]';
>> w2=[4 5 8]';
>> m1*v1+ m2*v2- m1*w1- m2*w2
ans =
[ -2*m1+m2  ]
[ -4*m1+2*m2]
[ -2*m1+m2  ]
```

Answer 4.28

The requested figure:

Multiplication with D yields a rotation of $\frac{\pi}{2}$ radians with respect to the origin. To mirror the image with respect to the x-axis, one should premultiply a vector with the matrix $M = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

INPUT SUMMARY:




```
p1=[1;2];
p2=[5;1];
p3=[4;3];
figure(1);
plot([p1(1), p2(1), p3(1), p1(1)],[p1(2), p2(2), p3(2), p1(2)],'k--')
axis([-6 6 -6 6]);
hold on;

D=[0 -1;1 0];
Dp1=D*p1;
Dp2=D*p2;
Dp3=D*p3;
plot([Dp1(1), Dp2(1), Dp3(1), Dp1(1)],[Dp1(2), Dp2(2), Dp3(2), Dp1(2)],'r-')
% etc...
```

B.5 Answers chapter 5

Answer 5.1

a) To obtain a solution of the differential equation you first have to make a function file in the MATLAB Editor/Debugger with the following content:

```
function xdot = filename(t,x)
xdot = 1-2*t*x;
```

Then you have to save the file under the name 'filename'.

This DE can be solved as follows:

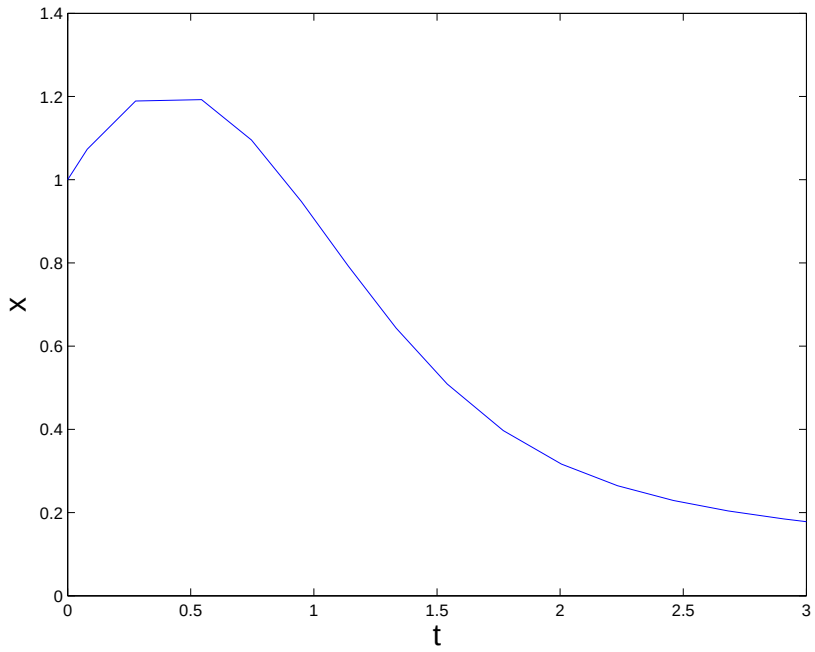
```
>> [t,x] = ode23('filename',[0,3],1);
```

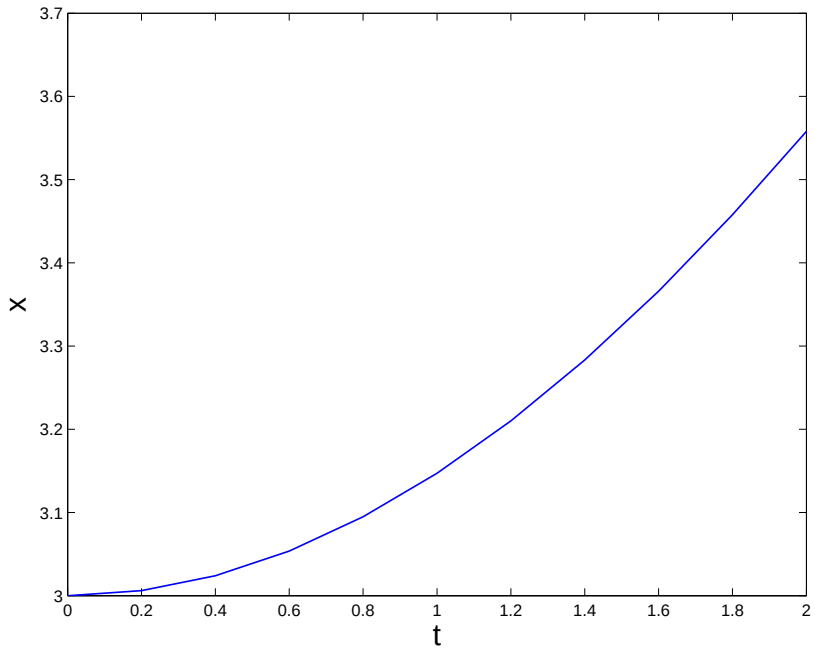
after which the matrices t and x are stored in the MATLAB memory.

The solution can be plotted with the command:

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

b) If you leave out the specification `[t,x]` the solution is directly plotted in a figure. The values of t and x corresponding to this solution are not stored in the MATLAB memory. In this case we can speak of a "real-time" solution.





Answer 5.2

Thus 11 steps are taken and the approximation for $x(2) = 3.5578$.

Answer 5.3

The direction field given by the equations

$\dot{x}_1 = 5x_1 - 2x_2$
$\dot{x}_2 = 7x_1 - 4x_2$

is plotted on the grid, given by the vectors x = [-1:0.1:1] and y = [-1:0.1:1], as follows:

- First define the grid:

  ```
  [x,y] = meshgrid(-1:0.1:1,-1:0.1:1)
  ```

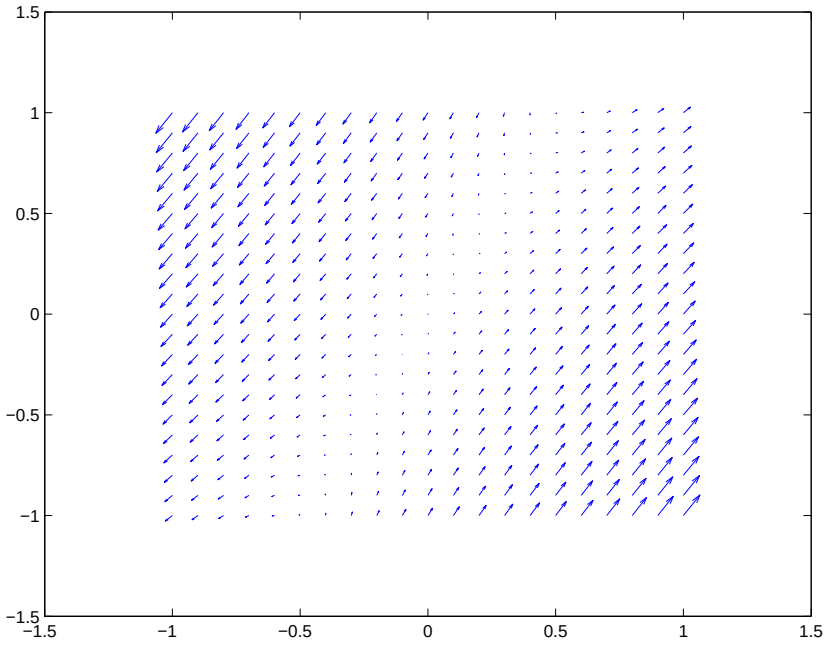
- Then calculate the direction vectors at every point in the grid:

  ```
  >> u = 5*x - 2*y;
  >> v = 7*x - 4*y;
  ```

  The ';' are practical if you do not want to display the complete matrix of u and v on the screen twice.

- Finally the direction field can be plotted:

  ```
  >> quiver(x,y,u,v)
  ```



Answer 5.4

Ex 6.1) `sol = -1/2*(i*pi^(1/2)*2^(1/2)*erf(1/2*i*2^(1/2)*t)-2*C1)*exp(-1/2*t^2)`

Ex 5.4) `sol = exp(-1/2*lambertw(exp(t^2+2*C1))+1/2*t^2+C1)`

Answer 5.5

$\dot{x}_1 = 5x_1 - 2x_2$
$\dot{x}_2 = 7x_1 - 4x_2$

Both the numerical and symbolic solution is given below for just one initial condition, namely $x_1(0) = 2$ and $x_2(0) = 8$. The procedure is analogous for any other initial condition.

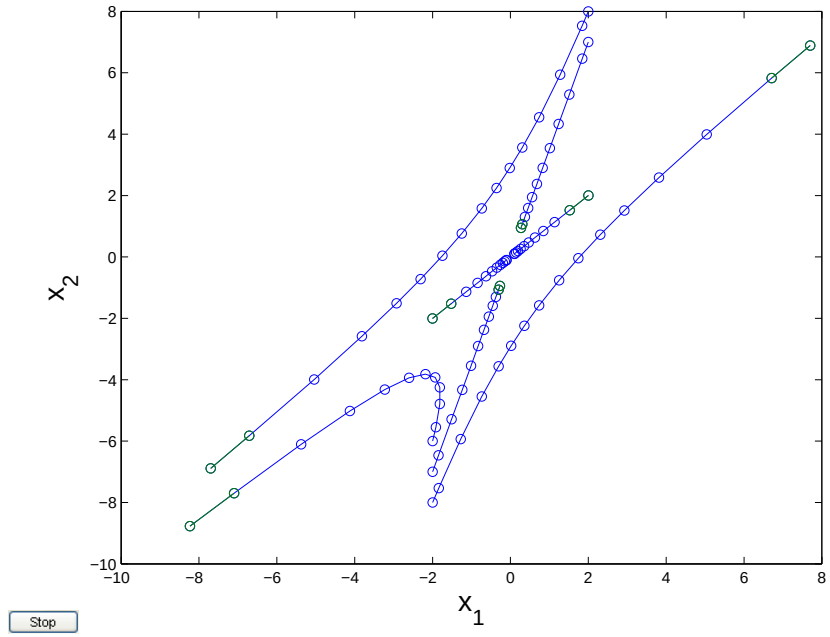
NUMERICAL SOLUTION:

Plotting the solutions for all initial conditions in the phase plane can be done numerically with `ode23` as follows:

```
>> ode23('fn5_7',[0,1],[2,8],odeset('outputfcn','odephas2'))
>> hold on
```

- 'diffvgl' is the function file that describes the differential equation.
- [0,1] represents the time interval in which the solution must be calculated.
- [2,8] represents the initial conditions $x_1(0) = 2$, $x_2(0) = 8$.

The first line can be repeated subsequently for the other initial conditions. The solutions can be plotted in the same figure by using the command `hold on`. SYMBOLIC SOLUTION:



Solving the problem symbolically is possible as well with `dsolve`:

```
>> [x1,x2] = dsolve('Dx1=5*x1-2*x2','Dx2=7*x1-4*x2','x1(0)=2','x2(0)=8')
>> t = [0:0.1:1];
>> x1 = subs(x1,t);
>> x2 = subs(x2,t);
>> figure, plot(t,x1,t,x2)
>> figure, plot(x1,x2)
```

The first graph is the time-plot at which the solutions of both equations of the system are plotted against the time. The second graph is the phase-plot at which the solutions of both equations of the system are plotted against each other.
Both graphs can be made for all initial conditions at which also the solutions for all initial conditions in one figure are plotted.

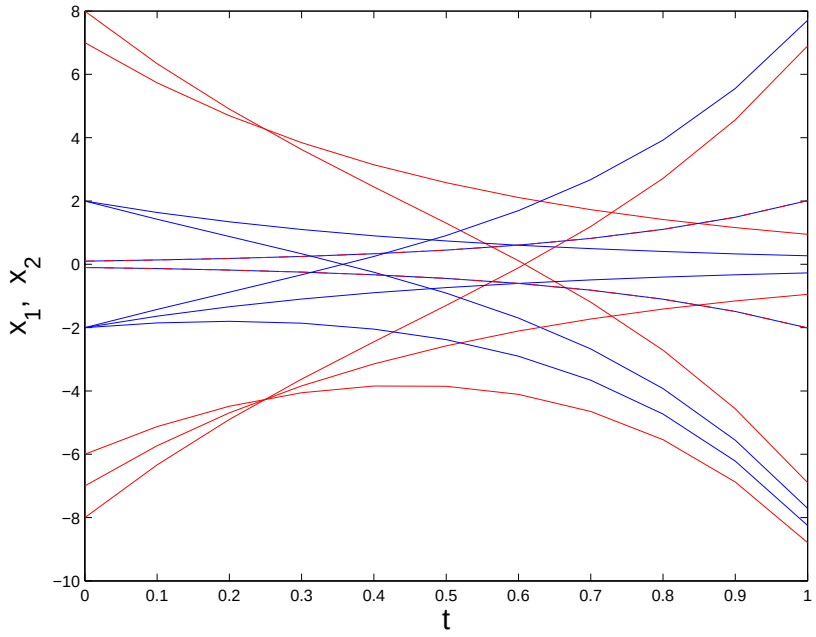
Answer 5.6

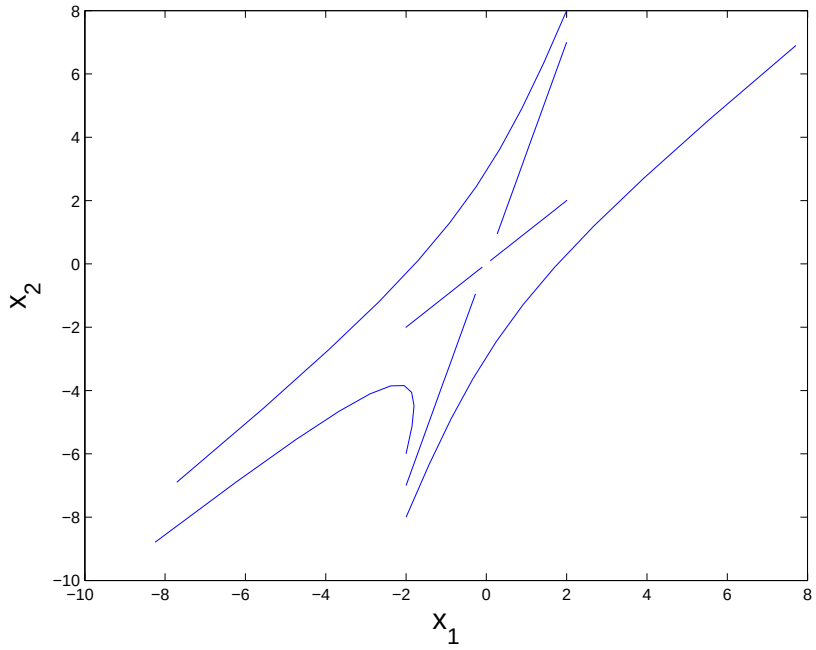
a) 1) `>> y = 1/5*sin(5*t)+cos(5*t)`

2) The vibration is undamped because the oscillation and the amplitude remain equal.

3) and 4) The vibration is periodic with period $\frac{2}{5}\pi$ and a frequency of $\frac{5}{2\pi}$ Hz.

b) 1) `y = 1/6*6^(1/2)*exp(-t)*sin(2*6^(1/2)*t)+exp(-t)*cos(2*6^(1/2)*t)`



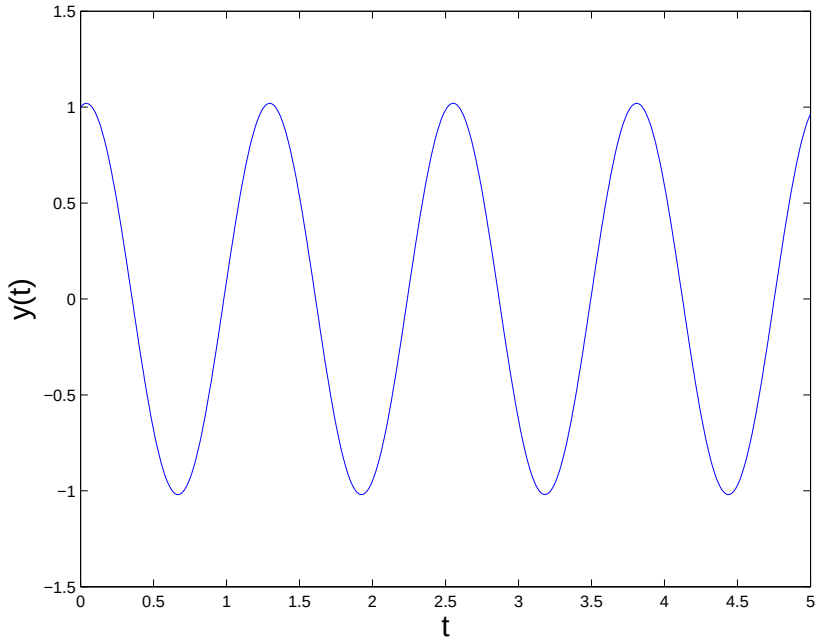


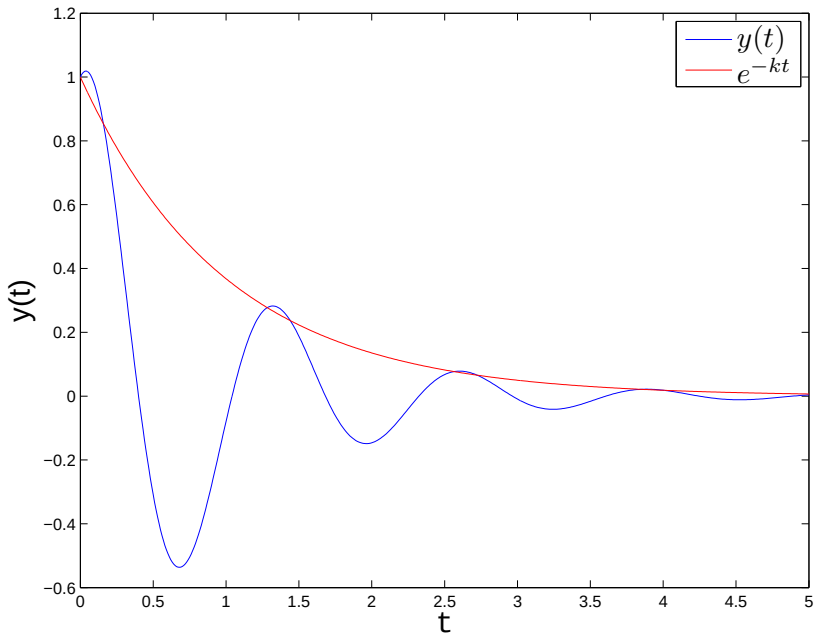
2) The eigenfrequency of the system is the solution of the homogenous problem which is $f_{\text{eig}} = 2\sqrt{6}$.

3) The vibration is damped.

4) and 5) The vibration is periodic with frequency $\frac{\sqrt{6}}{\pi}$ Hz.

c) 1) `y =`



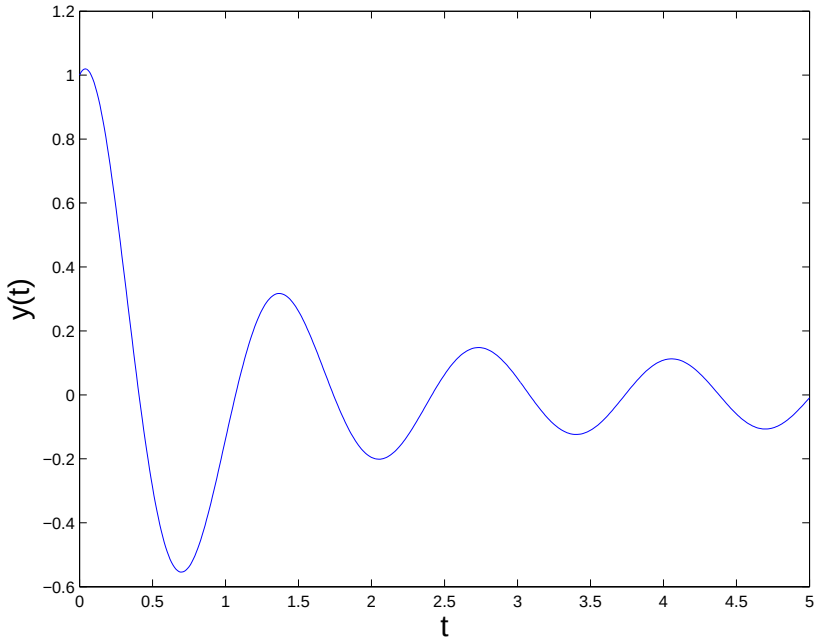



```
1/8*6^(1/2)*exp(-t)*sin(2*6^(1/2)*t)+exp(-t)*cos(2*6^(1/2)*t)+1/10*sin(5*t)
```

c

2) The vibration eventually looks to be undamped, but k is larger than 0. This can be noticed in the solution of the equation as well. Thus the vibration is actually damped.

3) and 4) The vibration is periodic with period $\frac{2}{5}\pi$ and a frequency of $\frac{5}{2\pi}$ Hz.



Answer 5.7

The relevant differential equation is:

$10\ddot{y}(t) + 50\dot{y}(t) + 490y(t) = 0, \quad$ with $y(0) = 0.2, \;\; \dot{y}(0) = 0.$

a1) Plotting this differential equation in a symbolic way can be done as follows:

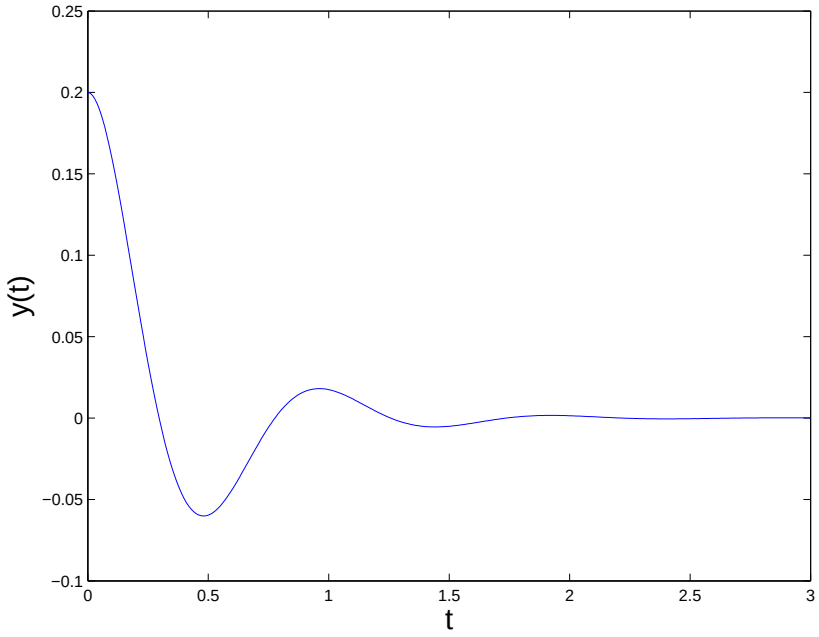
```
>> y = dsolve('10*D2y+50*Dy+490*y=0','y(0)=0.2','Dy(0)=0')
>> t = [0:0.01:3];
>> y = subs(y,t);
>> plot(t,y)
```

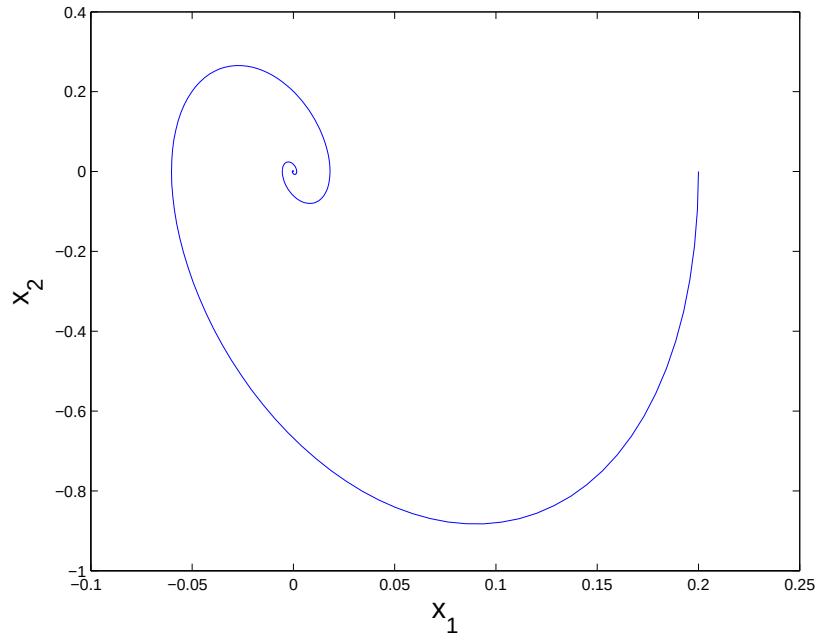
a2) Plotting of the solution in the phase plane is easiest by using separated equations like:

```
>> [x1,x2] = dsolve('Dx1=x2','Dx2=-5*x2-49*x1','x1(0)=0,2','x2(0)=0')
>> t = [0:0.01:3];
>> x1 = subs(x1,t);
>> x2 = subs(x2,t);
>> plot(x1,x2)
```

a3) Both methods mentioned above can be used to analyse the solutions for other initial conditions.

$y(0) = 0.2$ and $\dot{y}(0) = 0$ blue

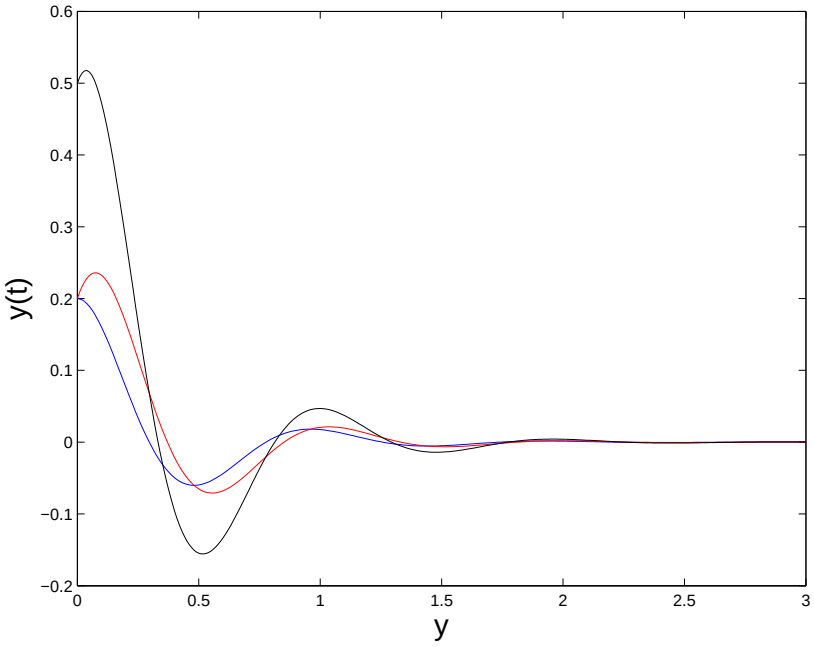


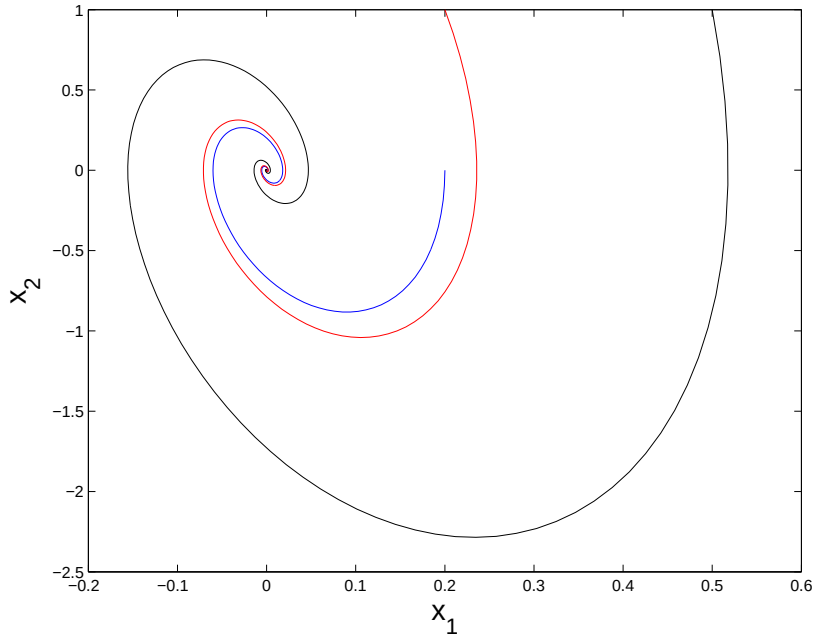


$y(0) = 0.2$ and $\dot{y}(0) = 1$ red
$y(0) = 0.5$ and $\dot{y}(0) = 1$ black

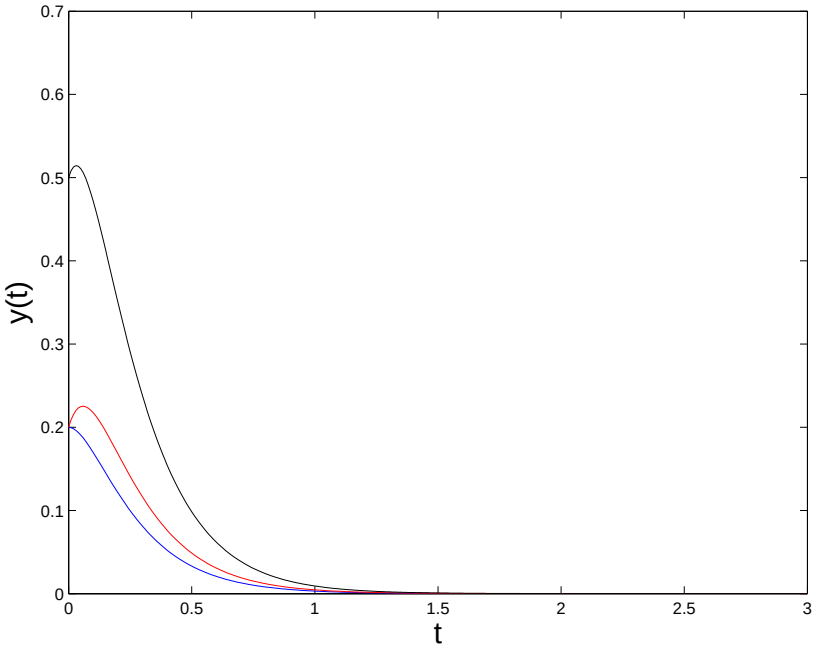
b) For super-critical damping the same method can be applied as in part a) only this time the damping constant has to be chosen correctly to achieve the desired super-critical damping: $k > 140$. In this case a damping constant of $k = 150$ is chosen.

c) If you let the spring loose above its equilibrium position, then, according to the super-

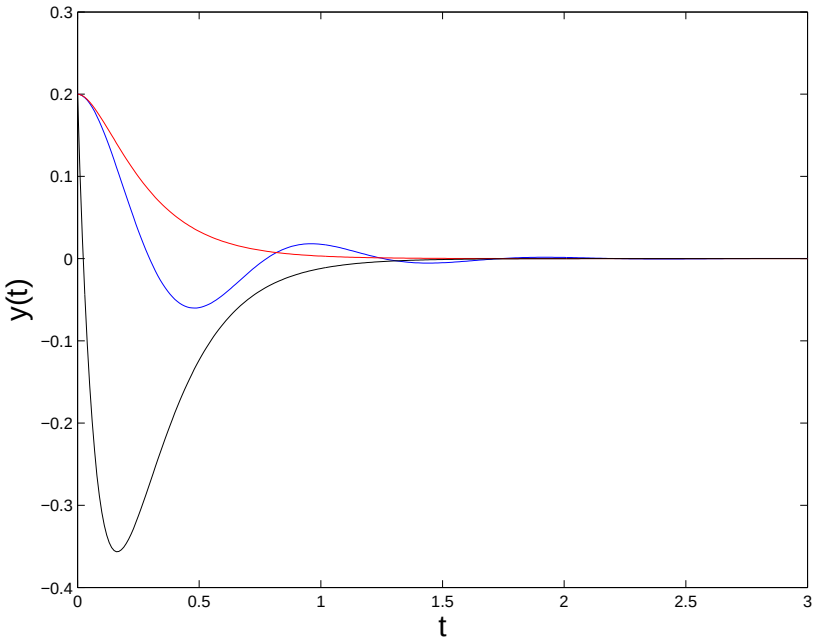




critical damping, it will go to its equilibrium position in one movement. To force this movement through this equilibrium position a higher initial velocity has to be taken. Thus choose $\dot{y}(0)$ high enough, for example $\dot{y}(0) = -25$. This effect is illustrated in the figure below. Critical damping occurs when the solution of the characteristic polynomial possesses 2 real solutions.

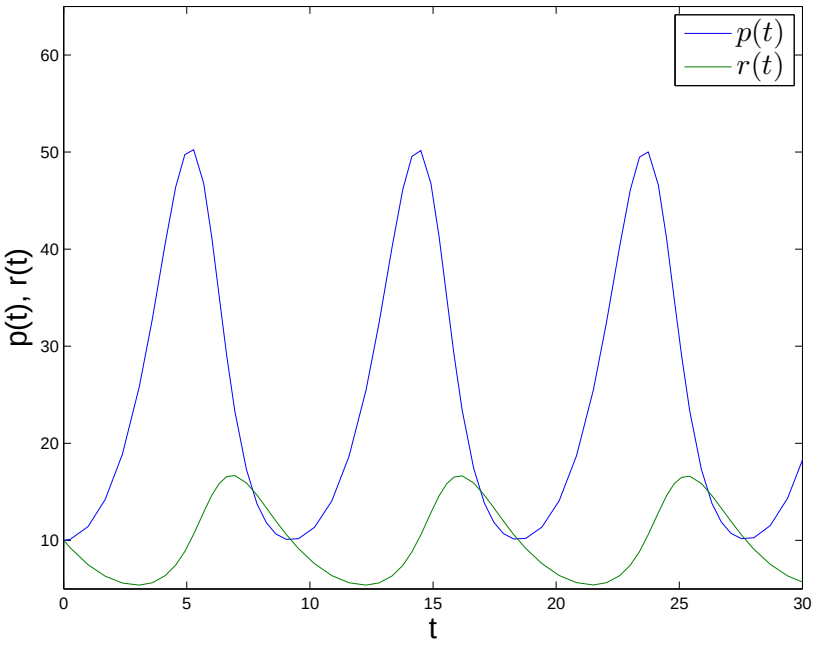


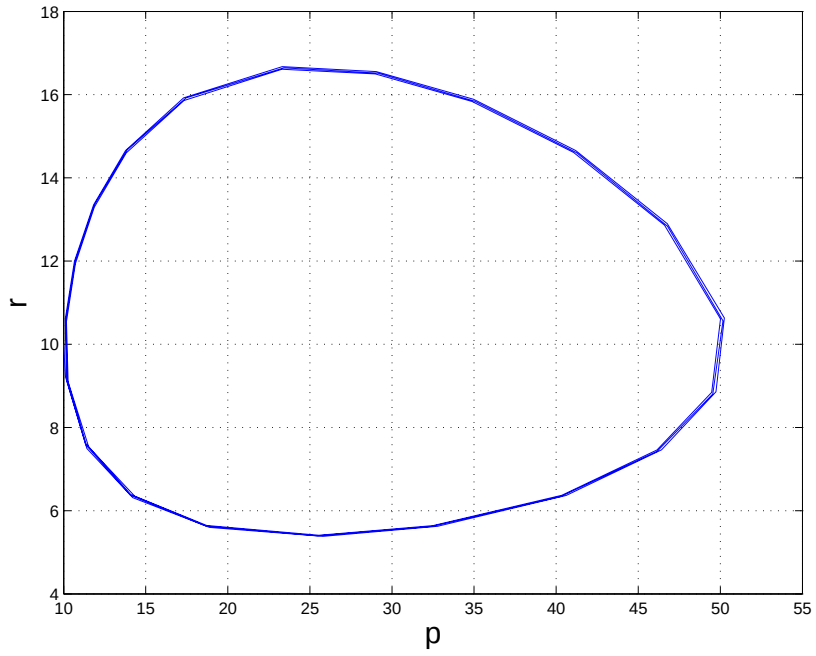
$c = 50, \quad y(0) = 0.2 \quad \text{and} \quad \dot{y}(0) = 0 \qquad \text{blue}$
$c = 150, \quad y(0) = 0.2 \quad \text{and} \quad \dot{y}(0) = 0 \qquad \text{red}$
$c = 150, \quad y(0) = 0.2 \quad \text{and} \quad \dot{y}(0) = -10 \qquad \text{black}$



Answer 5.8

a) For the initial conditions $y(0) = 10$ and $\dot{y}(0) = 10$ the results are the following:





b) No answer.

Answer 5.9

The driven mass-spring-damper system, given by the differential equations below, will be solved and visualised by the following approach:

$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = c\dot{x}(t) + kx(t)$

where $m = 870$ kg, $k = 70000$ N/m and $c = 5000$ N/ms.

For example take the initial conditions $y(0) = 1$ and $\dot{y}(0) = 2$ and a time interval from 0 till 10.

The solution is then given by:

```
>> y = dsolve('870*D2y+5e3*Dy+7e4*y=5e3*x2+7e4*x','y(0)=1','Dy(0)=2')
```

The definition for the input is now filled in:

```
>> syms t
>> x = sin(t); x2 = diff(x)
>> yx = subs(y,x);
>> yx2 = subs(yx,x2);
```

The solution is plotted by:

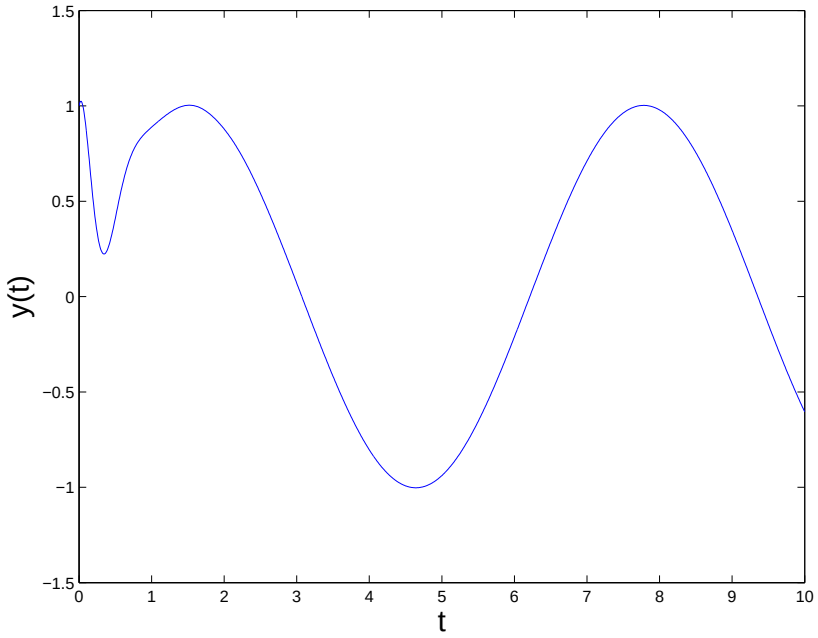
```
>> t = [0:0.01:10]
>> ynum = subs(yx2,t);
>> plot(t,ynum)
```

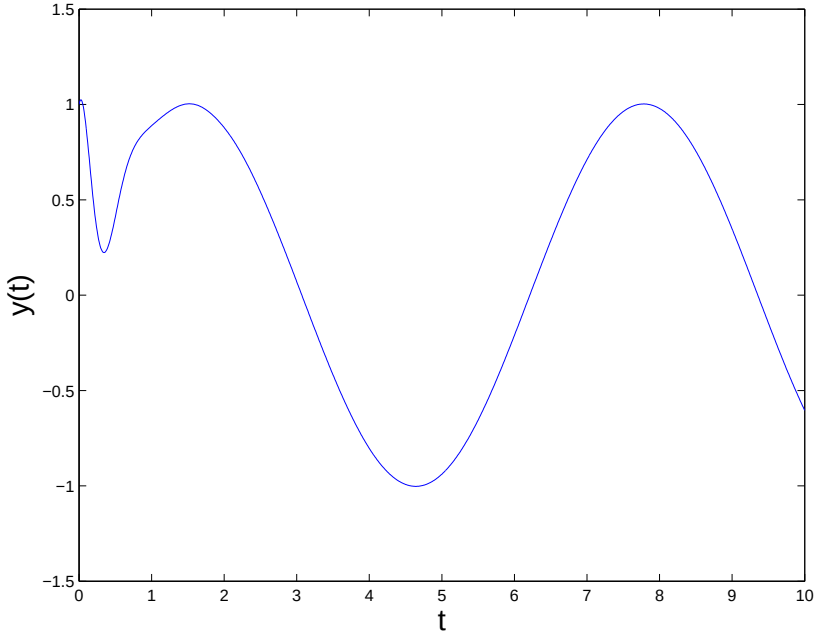
The periodic behavior is now clearly visible.

NB: The intermediate results can easily be checked with the command `pretty`. For instance try: `pretty(yx)`.
It is clear which variables are still present in the intermediate solution.

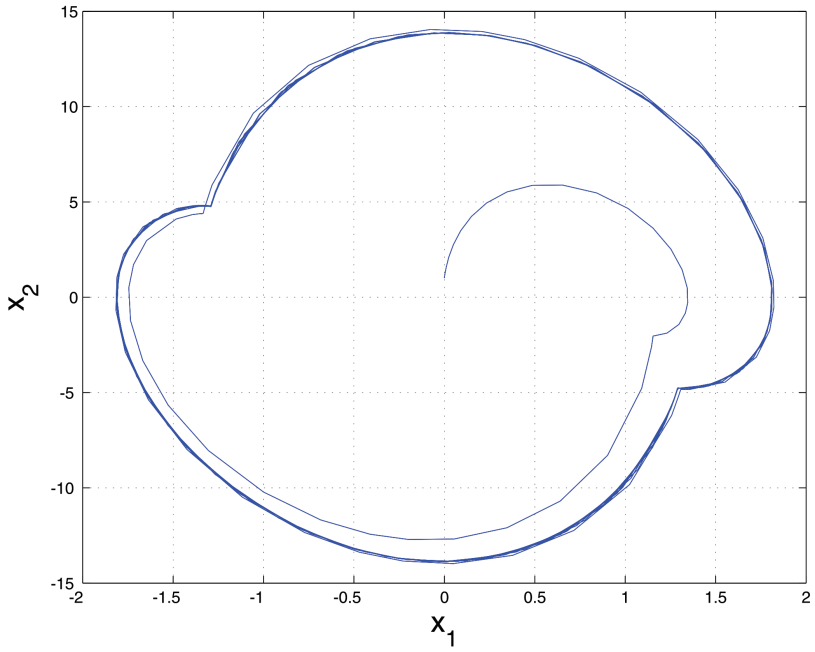
Answer 5.10

The same mass-spring-damper system is used as in the previous exercise with corresponding coefficients. Hence, this time another input function is used, namely an excitation function. The `sign(x)` function is the signum-function with argument x. This function returns a value 1 when the argument is greater than 0, 0 if the argument equals 0, and -1 if the argument is smaller than 0.


1.5
1
0.5
0
−0.5
−1
−1.5
y(t)
0
1
2
3
4
5
6
7
8
9
10
t


1.5
1
0.5
0
−0.5
−1
−1.5
y(t)
0
1
2
3
4
5
6
7
8
9
10
t

t	x
0.0000	3.0000
0.2000	3.0060
0.4000	3.0239
0.6000	3.0536
0.8000	3.0948
1.0000	3.1471
1.2000	3.2101
1.4000	3.2832
1.6000	3.3660
1.8000	3.4577
2.0000	3.5578

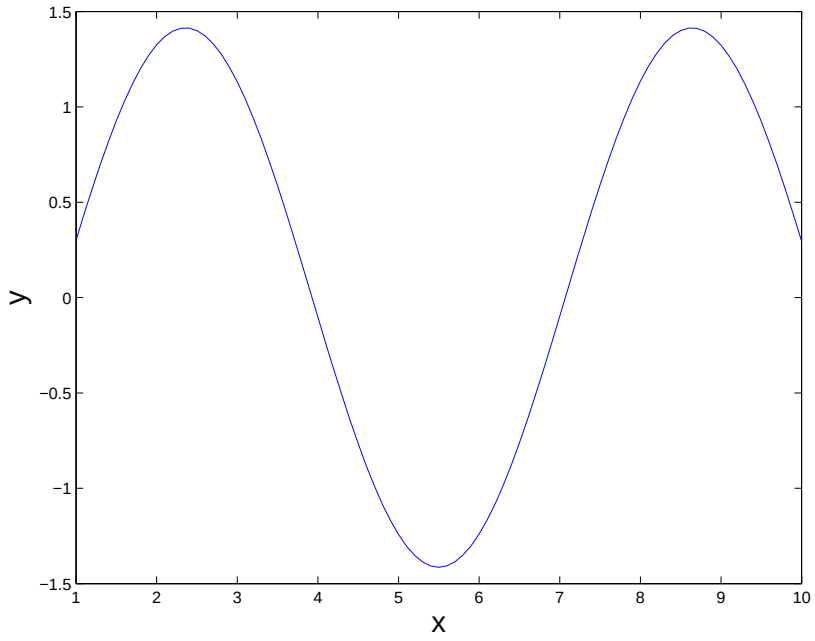


B.6 Answers chapter 6

Answer 6.1: Script file

a) The script file 'test1.m' is given by

```
% This script file calculates y = sin(x)-cos(x)
% and plots y as function of x.

clear all
x = 1:0.1:10;
y = sin(x)-cos(x);
figure;
plot(x,y)
```



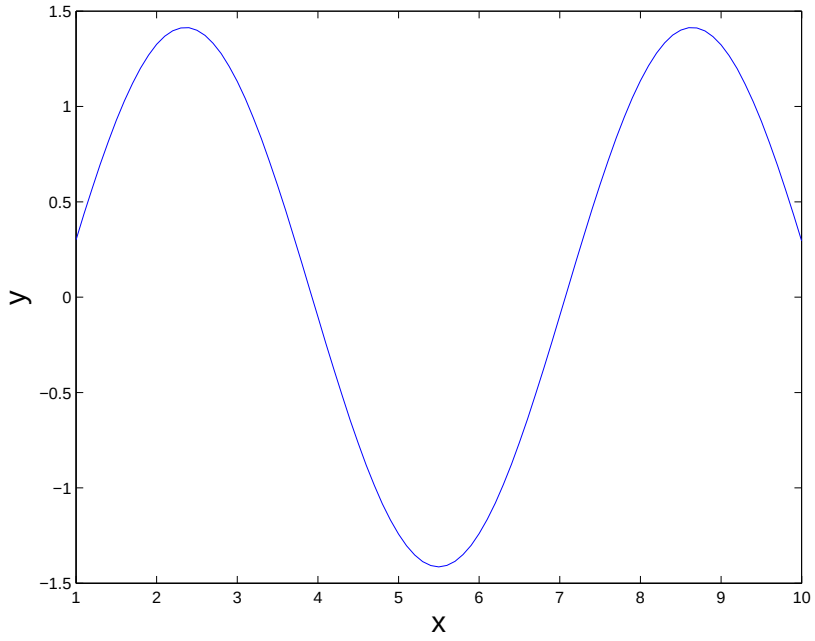
b) You can run the script file by typing `>> test1` in the MATLAB Command Window.

c) Calling up the commentary of the function file can be done with the command `>> help test1`. The result is the following:

```
> This script file calculates y = sin(x)-cos(x)
> and plots y as function of x.
```

Answer 6.2: Function file

a) No answer



b)

c)
```
> Function name: test 2   input: x   output: y
> This function calculates y = sin(x)-cos(x)
> and displays y as function of x.
```

d)

Scr: A script file is a programma that can work on its own and which can be called up every time. It contains a collection of entered commands. Furthermore all variables are accessible from the MATLAB Command Window.

Fun: A function file is a programma that cannot work on its own. Input variables are needed to run the file. The result are output variables. This time only the input- and output variables are accessible from the MATLAB Command Window. All other variables are only usable in the file itself. The first line of the file is always as follows:

```
function output = function_name(input)
```

Answer 6.3: FOR loop

a) The step-by-step approach is as follows:

- Define L as the length of r.
- Begin: For-loop $\rightarrow$ Perform for a (with the values $1, 2, \ldots, L$) the following commands:
  - Square the a^{th} element form the row vector r and store the result in the row vector b on the location of the a^{th} element.

- End: For-loop.

b) The function file 'kwad.m' is given by

```
function b = kwad(r)

% Function name: kwad   input: r   output: b
% This function file squares element-wise the elements of
% the row vector r and stores the result in the row vector b

L = length(r);
for a = 1:L
   b(a) = r(a)*r(a);
end
```

c) You can test the function file by typen the following text in the MATLAB Command Window:

```
>> a = [1 2 3 4 5];
>> kwad(a)


ans =

     1     4     9    16    25
```

Answer 6.4: FOR loops

a) No answer

b) No answer

c)
```
>> A= [0 1;2 3];
>> kwadm(A)

ans =

     0     1
     4     9
```

Answer 6.5: Input

a) The script file 'date.m' is given by

```
a = input('Do you want to know the date? Press 1 for 'yes' or 2 for 'no',
           followed bij Enter');
if a == 1
```

```
   today = date
else
   disp('You don't want to know the date')
end
```

b) You can test the function file in the MATLAB Command Window by typing:

```
>> date
```

Answer 6.6: IF loop

a) No answer

b) No answer

c)
```
>> a = [1;2;3;4;5;6;7];
>> replace(a)

ans =

     1
     2
     3
     4
     5
     0
     0
```

Answer 6.7: IF loops

a) The step-by-step approach is as follows:

- Define rA and cA as the number of rows of the matrix A and the number of columns of A respectively.
- Begin: For-loop1 $\rightarrow$ Perform for a (with the values $1, 2, \ldots, rA$) the following commands:
  - Begin: For-loop2 $\rightarrow$ Perform for b (with the values $1, 2, \ldots, cA$) the following commands:
    - Begin: If-loop. If the element in matrix A with coordinates (a, b) is greater than 5, then store a zero in matrix B on the location of the element with coordinates (a, b).
    - Else (if the element $A(a, b)$ is not greater than 5) store the value of the element $A(a, b)$ in $B(a, b)$.
    - End: If-loop.
  - End: For-loop2.
- End: For-loop1.

b) A possible function file 'replace2.m' is given by

```
function B = replace2(A)

% This function file replaces all elements of A greater than 5 with 0.
% Then it stores the new elements in a matrix B.

[rA cA] = size(A);
for a = 1:rA
   for b = 1:cA
       if A(a,b) > 5
         B(a,b) = 0;
       else
         B(a,b) = A(a,b);
       end
   end
end
```

c) You can test the function file by typing the following in the MATLAB Command Window:

```
>> A = [2 12; 5 40];
>> replace2(A)


ans =

     2     0
     5     0
```

Answer 6.8: WHILE loop

```
>> [p,q] = divide(27)

p =

     4


q =

    1.6875
```

Answer 6.9: Debugging

Error message 1:

```
??? Error: File: errors.m Line: 3 Column: 1
At least one END is missing: the statement may begin here.
```

The reason that this error message arises is because you make use of two for-loops, but only one of these loops is ended with the command `end`. The rule is that every for-loop needs to be ended with this command.

Error message 2:

```
??? Undefined function or variable "c".

Error in ==> errors at 6
       c = c+1
```

The reason that this error message arises is because the variable c needs to be initialised first. After this it is possible to use the variable.

Error message 3:

```
??? Attempted to access x(1,4); index out of bounds because size(x)=[4,3].

Error in ==> errors at 6
        d(a,b) = x(a,b)-1
```

The variables p and q are switched which causes the problem that elements of x are asked for that do not exist.

The correct function file is given by

```
function [c,d] = errors(x)

% This function file generates a matrix d of which the elements
% are the elements of x minus 1. Furthermore this file gives as
% result how often (c) the 2nd for-loop is passed through.

[p q] = size(x)
c = 0                    % error message 2
for a = 1:p              % error message 3
   for b = 1:q           % error message 3
   d(a,b) = x(a,b)-1
   c = c+1
   end
end                      % error message 1
```

Answer 6.10

A possible examples is given below

```
>> A = [3 4;6 1];
>> B = [1 3]';
>> D = matmult(A,B)

D =

    15
     9
```

Answer 6.11

Extending the function file 'matmult.m' to an matrix-matrix multiplication results in the function file 'matmult2.m' given below.

Pay attention that this function file is NOT the only solution for this problem! The programmer is responsible for the approach in solving the problem and thus for the result of the function file. Every programmer has its own style. This is one of the most important reasons to add comments to your script. Then it is possible for every other programmer to understand the programme very quickly.

```
function D = matmult2(A,B)

[rA cA] = size(A);
[rB cB] = size(B);
D = zeros(rA,cB);
for r = 1:cB
   for p = 1:cA
      for q = 1:cA
      c = A(p,q)*B(q,r);
      D(p,r) = D(p,r)+c;
      end
   end
end
```

B.7 Answers chapter 7

Answer 7.1

mdl file (zipped).

Answer 7.2

mdl file (zipped).

1. Change the maximal step size!

Answer 7.3

mdl file (zipped).

Answer 7.4

mdl file (zipped).

1. Use the sine block and introduce a phase shift to obtain the cosine!

Answer 7.5

mdl file (zipped).

Answer 7.6

mdl file (zipped).

Answer 7.7

mdl file (zipped).