Matlab
Entrance Package

September 2017

Faculty of Electrical
Engineering, Mathematics
and Computer Science

TU Delft
Delft
University of
Technology
Artikelnummer 06934850001

Contents

Preface

The Matlab entrance package contains some basic material useful for students that need to program in Matlab. In particular, the Matlab manual will be useful in this context. This entrance package has been developed with the focus on students entering practicums associated with the basic Numerical Mathematics courses. For other students the package might be useful as well because it contains a rather compact Matlab manual containing examples.

DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

September 2016

0A2: MATLAB TUTORIAL, DESKTOP FACILITIES

Preface Gen0.1 (September 2016)

This tutorial aims to give a first survey of Matlab with emphasis on control. The document is written with the use of Matlab in courses on numerical mathematics and simulation in mind. The tutorial is optional. *You are advised to make this tutorial before the practicum starts in case you only have limited knowledge of the desktop facilities of Matlab.*

This tutorial consists of a number of actions that must be taken and questions directed to the observation of the result. *The elaboration of the tutorial will take about 1 hour*.

The tutorial is intended to illustrate the character of Matlab and to get acquainted with the desktop facilities. The tutorial is quite general and introductory. For the application of Matlab within specific educational activities additional material is often available, like the Matlab manual and the Matlab Introduction Exercise for courses in the area of Numerical Mathematics.

Operating system

This document is compatible with Matlab under Windows. For other operating systems small deviations may be present.

Preparation

Use the file manager (Explorer) to make a work folder with the name *Tutorial*. For this work folder choose an appropriate place that you wish to reserve for your activities in the context of the course.

Go to Blackboard, *Matlab folder* of the *NumPr_edu* site. Here, one may find the file *0A*.zip* with on the place of * a number corresponding to the code of the tutorial. Download the file to the work folder *Tutorial*. Unzip the file. The file contains the supporting files needed to conduct the tutorial.

Philosophy

Matlab can be used command line driven, i.e. commands are entered in the so-called Command Window and they are executed by using the return/enter button. The alternative is to use so-called scripts. Scripts are files that contain Matlab commands. A script can be executed. In this document we favor the approach using scripts.

Desktop settings

Start Matlab

Just below the upper toolstrip one may also find the location of the current directory, basically the directory in which Matlab is active. We need to make the work folder *Tutorial* the current folder in Matlab. For this it is needed to browse towards this folder. Browse actions are possible on the left of the echo print of the name of the current folder (go up + go down button) and also in the print of the name (black triangles). One way to go, is to click on the go up icon (maybe several times) and to browse to the work directory *Tutorial* in the current folder window.

Close Matlab. Start Matlab for a second time. Go to the folder *Tutorial* via the black triangle at the end of the 2nd row of the upper toolbar.

Remark: You can select the default current directory. Click, using the right button of the mouse, on the Matlab icon on the desktop. Choose Properties. Choose a directory after "Start in". □

Next, click on the Layout button in the upper toolstrip. Using the pull-down menu, delete the appearance of the Current Folder and, by repetition, delete also the appearance of the Command History. This results in desktop settings that are convenient for the present tutorial. Matlab stores these desktop settings and if Matlab is started a next time, it chooses these settings. If needed, one may alter the settings again in the future.

Workspace and Matlab Editor TD1 (September 2016)

Type the command *extd1* in the Command Window and give a return/enter. Look in the Workspace. Which variables have been created?

Go to the upper toolstrip and choose for 'Clear Workspace'.

Click on the 'Open' button in the upper toolstrip. Select *extd1.m* and open the file. The Matlab editor will be activated automatically and the contents of the file will be displayed. Click on the downward pointing arrow at the top of the editor window and choose for 'Undock' in the pull-down menu in order to display the Editor outside Matlab's main window. Click on "Run' in the upper toolstrip of the editor. Observe the contents of the Workspace in relation with the contents of the script file *extd1.m*.

Close the Matlab editor.

Go to the Command Window and type

```
text = 'try1';
```

Observe the result in the Workspace.

Clear the Workspace and the Command Window. For this type *clear, clc* in the Command Window. Alternatively, one may do this via the 'Clear Workspace' and 'Clear Commands' buttons in the upper toolstrip.

Array Editor TD2 (September 2016)

Open the file *extd2.m* and run the script. Go to the workspace and double click on the letter A. The Array Editor will be opened automatically and the contents of the variable A will be displayed.

You may undock the array editor via the downward pointing arrow on top. The Array Editor has several applications, but is at least useful to inspect variables. Observe the variable A in relation with the contents of the script.

Close the Array Editor and the Matlab editor. Next, clear both the Workspace as well as the Command Window.

Property editor TG1 (September 2012)

Open the file *extg1.m* and execute the script. You will see two graphical windows both containing a plot of a circle. Why did the author of the script include the command on the last line? Close the first graphical window and go to the second graphical window. Click on Edit, next choosing Axes Properties for access to the Property Editor. Use this editor to alter some of the properties of the axes, i.e.

Title	Circle
Xlabel	x
Xlimits	-1 and 1
Ylabel	y
Ylimits	-1 and 1

Click on the trajectory in the graphical window. This makes the trajectory the present graphical object and gives access to the property editor for altering properties of this object. Make the line thickness equal to 2.0 and the line color equal to black.

In this way the plot can be pimped up step by step.

Close the graphical window, the Matlab editor and clear the Workspace and the Command Window.

Graphical application TG3 (September 2012)

Open the file *extg3.m* and execute the script. A plot is shown. This plot presents the solution of the differential equations

$$\begin{cases} \dfrac{dx}{dt} = y \quad , \quad x(0) = 0 \\ \dfrac{dy}{dt} = -x \quad , \quad y(0) = 1. \end{cases}$$

where $x(t)$ and $y(t)$ are drawn in the (x, y)-plane or phase plane. Modify the file *extg3.m* such that a graph of x is plotted as a function of t in the (t, x)-plane.

It is interesting to represent the vector $[x\ y]$ as a function of t. To this end Matlab has the routine ***plot3*** available. Go to the Command Window and type ***help*** *plot3*, followed by a return/enter. Modify the file in order to make the new plot. The plot should show a spiraling movement on a cylinder.

Close the Matlab editor, the graphical window and clear the Command Window and the Workspace.

Advanced graphical application TG2 (September 2012)

Open the file *extg2.m* and execute the script. The first graphical window contains a so-called (computational) grid that is used for computing flow around an airfoil (e.g. the wing of an aircraft). How many flaps can you see?

In the second graphical window we see an image of a clown. Which color has the nose of the clown? Go to Edit in the upper toolbar of this window and choose for Colormap. This activates the Colormap Editor. Go to Tools and choose Standard Colormaps. Select *summer*. Which color has the nose now? Also, select *jet* as the colormap. Which color has the nose now? What colormap will be the default colormap in Matlab?

Of course, for building such an advanced image, the standard colormaps from Matlab will often be insufficient. However, the author of the script has build its own colormap. You can see this by consulting the Workspace. The author has created the variable map. Add the command ***colormap****(map)* as last line to *extg2.m* and run the adapted script. Which color has the nose now?

DELFT UNIVERSITY OF TECHNOLOGY September 2017
Faculty of Electrical Engineering, Mathematics and Computer Science

0B1: MATLAB INTRODUCTION EXERCISE

VARIOUS COURSES, NOT FOR WBMT2049, WI2031WBMT

GENERAL General Gen-1 (Gen11.1, September 2014)

This exercise will give you a brief introduction to the software package Matlab. In some courses, the exercise has to be conducted at the start of your practical work (in other courses the exercise is optional). Working out the Matlab introduction exercise will take about 4 hours. It is recommended (and expected) to work out this exercise, at least in part, before entering the first meeting (meeting 1). It is also recommended to work in groups of two persons; it is allowed to work individually. The students are free to form the groups themselves.

STUDENT DATA

Name :
Student ID :

Name :
Student ID :

ORGANISATION

Answers need to be written on the exercise form. Prints and plots can be shown directly on the computer screen or can be enclosed on separate sheets.

Show your work when consulting the lecturer. The lecturer will verify whether your answers, plots and prints are correct. If so, the lecturer will record that you have completed this part of the work.

When working in the computer room, it is advisable to consult the lecturers also intermediate for checking answers. Otherwise, all work needs to be done at the end of the session with long queues as a consequence.

Exercise Op-1 (Op11.1, September 2008) *(Matlab manual: sections 4, 5)*

Compute $5.1^{1.7}$

Answer: $5.1^{1.7} =$ *(in **format short e**)*

Exercise Op-2 (Op12.1, September 2017) *(Matlab manual: sections 5, 9)*

Compute $y = \dfrac{e^x + \sin(\pi x) + x^{10}}{10}$ for $x = 0.357$.

Answer: $y =$ *(in **format short e**)*

Exercise Op-3 (Op13.1, September 2008) *(Matlab manual: section 6)*

Given are the complex numbers

$$z_1 = 1 + i\,, \quad z_2 = 1 + 2\,i$$

Compute $z_1\, z_2\,, \quad |\, z_1\, z_2^{\,2}\,|$.

Answer: $z_1 z_2 =$ $|\, z_1\, z_2^{\,2}\,| =$

Intermezzo: General Gen-2 (Gen15.1, September 2016)

Go to the command windows and type

clear, close, clc

in order to clear the workspace, to close the graphical window and to clear the command window.

It is advisable to use script files. Go to the upper toolstrip and choose for 'New Script'. You enter the Matlab editor. Type

```
x=1; y=1;
plot(x, y,'+');
```

in the editing window. Next, choose 'Run' in the upper toolbar. A 'Save As' communication window pops up. Choose a convenient folder (directory) to store the script file. Give the file the name temp.m. Then, save the file in the selected directory by clicking the OK button, which also triggers running of the script file. Go to the central window and check the workspace to see that the variables x and y exist. Also, have a look in the command window and the graphical window. Next, go to the command window and type

clear; close all; clc;

in order to clear the workspace, to close the graphical window and to clear the command window. Use from now on the editor to build and run scripts, with the suggested names ex1.m, ex2.m, …. It is advised to start every script file with

clear; close all; clc;

Note that for running a script you first have to change the Matlab current directory (just above the Command Window) to your selected directory.

Exercise Ma-1 (Ma11.1, September 2008) *(Matlab manual: sections 7, 8)*

Given is the matrix

$$A = \begin{bmatrix} 5 & -10 & 1 \\ -10 & 5 & -3 \\ 1 & -4 & 2 \end{bmatrix}.$$

Let v be the first column of A and let w be the second row of A. Compute vw and wv. Assume $B = A^3$. Determine B_{23}.

Hint: vw and wv will have different sizes! □

Answer: $vw =$ $wv =$ $B_{23} =$

Exercise Ma-2 (Ma12.1, September 2013) *(Matlab manual: sections 8, 9)*

Given is the matrix

$$A = \begin{bmatrix} 5 & -10 & 1 \\ -10 & 5 & -3 \\ 1 & -4 & 2 \end{bmatrix}.$$

We denote the entries of A with a_{kl}, i.e. $A = (a_{kl})$. The matrix $C = (c_{kl})$ is formed by setting

$$c_{kl} = (a_{kl})^3 .$$

Determine $\det(C)$.

Hint: Use operations in array sense to form C and next calculate the determinant. □

Answer: $\det(C) =$

Exercise Ma-3 (Ma13.1, September 2017) *(Matlab manual: sections 7, 9)*

Given is a grid consisting of 51 grid points covering the interval $[1,2]$, mathematically given by

$$x_k = 1 + k\,h\,, \quad k = 0, \cdots, 50\,, \quad h = \frac{2-1}{50}\,.$$

Let us collect the grid values in the row vector x. What is the size of the vector x? Present the Matlab command that generates the vector x. Do not use a loop statement, use one call of **linspace** instead in which x is generated directly without the necessity to rescale x (thus using a single Matlab command instead of multiple).

Matlab command(s) read:

We need to compute

$$N = \sum_{k=0}^{50} \sin(x_k)\,.$$

Present both the answer as well as the Matlab commands for this (do not use a loop statement).

Matlab command(s) read:

Answer: $N =$

Exercise Ma-4 (Ma14.1, September 2012) *(Matlab manual: section 7)*

You have to generate the row vector **y** with the following components

$$y_n = \begin{cases} 1 \;, & 1 \le n \le 10 \\ 5 \;, & 11 \le n \le 20. \end{cases}$$

Generate this vector by concatenating two subvectors.

Hint: Avoid loops, use **ones** instead. □

Matlab commands to generate **y** *are:*

Exercise IO-1 (IO11.1, September 2016) *(Matlab manual: section 12)*

Using the **disp** command, make a table in the command window with two columns v containing values of x, respectively y, where

$$y = \sin(x)\,,\; x = k\frac{\pi}{10} \text{ and } k = 0, 1, \cdots, 10\,.$$

Present the numbers in **format short**. Try to add a top line with text for headings.

Matlab commands read:

Exercise Ma-5 (Ma15.1, September 2016) *(Matlab manual: sections 1.2, 8, 14)*

With $N = 50$ and $h = \frac{2}{N}$, the vectors x, y, z are given by

$$x(n) = nh,$$
$$y(n) = \sqrt{x(n)},$$
$$z(n) = (x(n))^2, \quad n = 0, \cdots, N.$$

Note that x presents a grid of 51 grid points covering the interval $[0, 2]$.

a. Generate the vector x using the Matlab function **linspace**.

b. Create the vectors y and z without using loop statements.

Hint: Use array operations to form z from x. □

c. Display the functions $\sqrt{x}, x^2$ on the interval [0,2] as graphs. Put them in one figure and make use of the vectors mentioned above. Both graphs must be black. Keeping them black, make the two graphs visually different (use the Matlab command **help plot** to find out how you can do this). Let your plot be checked by the lecturer.

Hint: Use **hold on** to display two plots in one figure. □

a. *Matlab commands read:*

b. *Matlab commands read:*

c. *Matlab commands read:*

Exercise Lo-1 (Lo11.1, September 2008) (*Matlab manual : section 11)*

Compute the numbers $x = (x_n)_{n=1}^{50}$ from

$$x_1 = 0.1$$
$$x_{n+1} = 3.5\ x_n\ (1 - x_n), \quad n = 1, ..., 49.$$

Matlab commands read:

Next, investigate the Matlab command **plot**(x,':o'). What is displayed along the horizontal and vertical axes? A periodic behavior in this 'chaotic' system can be observed. What is the period?

Exercise Ma-6 (Ma16.1, September 2008) *(Matlab manual: section 16)*

Given are the matrix A and the vector b

$$A = \begin{bmatrix} 5 & -10 & 1 \\ -10 & 5 & -3 \\ 1 & -3 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Determine the solution x of $Ax = b$.

Matlab command(s) used:

Answer: $x =$

Exercise Ma-7 (Ma17.1, September 2015) *(Matlab manual: sections 7, 16, 21)*

Given are the $n \times n$-matrix A and n-dimensional vector b, with $n = 20$,

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & 0 & \\ & . & . & . & . \\ & 0 & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}.$$

Present the Matlab commands creating A and b:

Determine the solution x of $Ax = b$. Is it favorable to use **sparse**?

Matlab commands used:

Answer: $x(10) =$

Exercise Fu-1 (Fu11.1, September 2008) *(Matlab manual: section 15.2)*

Given is the function

$$f(x) = \frac{\sin(x)}{x}.$$

Make a function file called fu11.m for this function. *The function file fu11.m reads:*

Next, go to the command window and type

» *clear; y=fu11(x)*

Result: $y =$

Also try

» *clear; x=0.1; y=fu11(x)*

Result:

Exercise Fu-2 (Fu3.5, September 2017) *(Matlab manual: sections 7, 15.2)*

We consider the vector function $\mathbf{f} : \mathbb{R}\, x\, \mathbb{R}^2 \to \mathbb{R}^2$. This function has two components

$$\mathbf{f}(t, \mathbf{x}) = \begin{bmatrix} f_1(t, \mathbf{x}) \\ f_2(t, \mathbf{x}) \end{bmatrix} = \begin{bmatrix} f_1(t, x_1, x_2) \\ f_2(t, x_1, x_2) \end{bmatrix}.$$

In fact, the function is given by

$$f_1 = 2\, x_1 x_2 + \frac{1}{2} x_1 ,$$
$$f_2 = x_1 x_2 + x_2 - \cos(\pi t) .$$

Write a function file with input t, $\mathbf{x}$ and with output the function value **fval**. Both $\mathbf{x}$ and **fval** must appear as column vectors in this function file. The function file reads:

Via a function call, determine

$\mathbf{f}_a = \mathbf{f}(1, \begin{bmatrix} 1 \\ 1 \end{bmatrix}) =$ $\mathbf{f}_b = \mathbf{f}(1, \begin{bmatrix} 1 \\ 0 \end{bmatrix}) =$

Exercise Gr-1 (Gr3.5, September 2012) *(Matlab manual: sections 6, 14)*

It is given that $\lambda = -0.4 + i$. Consider the complex function $Q : \mathbb{R} \to \mathbb{C}$ given by

$$Q(h) = 1 + \lambda h,$$

the amplification factor of the Euler time stepping method. Our interest is towards its modulus $MQ = |Q|$. Make a plot of MQ as a function of h, where $0 \le h \le 2$. Let your plot be checked by the lecturer. The first positive value of h for which $|Q| = 1$ is denoted by $h_{\max}$. Determine $h_{\max}$ with an accuracy of 0.01. Do this graphically by adding gridlines to the graph using the command **grid** (and if necessary, use the Matlab zoom option in the graphical window).

Hint: h must be along the horizontal axis of the plot. □

Matlab commands used:

Answer: $h_{\max} =$

Exercise Gr-2 (Gr11.1, September 2012) *(Matlab manual: section 14)*

Given is the function $f : \mathbb{R}^2 \to \mathbb{R}$

$$f(x, y) = x^2 \sin(y).$$

Make a plot of the surface of f on the region $0 \le x \le 1, 0 \le y \le 3$. Show the plot to the lecturer.

Hint: First, using **linspace**, create two vectors x and y containing one-dimensional grids. Next, define a product grid [X,Y] = x ⊗ y on this region, using **meshgrid**. Then create a two-dimensional array f containing the values of the function on this grid (do not use loops). Note that the number of rows of f is equal to the size of y and the number of columns of f is equal to the size of x. □

Matlab commands used:

DELFT UNIVERSITY OF TECHNOLOGY September 2017
Faculty of Electrical Engineering, Mathematics and Computer Science

0B21: MATLAB INTRODUCTION EXERCISE

COURSES WBMT2049, WI2031WBMT ONLY

GENERAL General Gen-1 (Gen21.1, September 2015)

This exercise will give you a brief introduction to some Matlab concepts often used in numerical programming. The exercise has to be conducted at the start of the practicum. Working out the Matlab introduction exercise will take about 2 hours. It is recommended (and expected) to work out this exercise, at least in part, before entering the first meeting (meeting 1). It is also recommended to work in groups of two persons; it is allowed to work individually. The students are free to form the groups themselves.

STUDENT DATA

Name :
Student ID :

Name :
Student ID :

ORGANISATION

Answers need to be written on the exercise form. Prints and plots can be shown directly on the computer screen or can be enclosed on separate sheets.

Show your work when consulting the lecturer. The lecturer will verify whether your answers, plots and prints are correct. If so, the lecturer will record that you have completed this part of the work.

When working in the computer room, it is advisable to consult the lecturers also intermediate for checking answers. Otherwise, all work needs to be done at the end of the session with long queues as a consequence.

Exercise Ma-1 (Ma13.1, September 2017) *(Matlab manual: sections 7, 9)*

Given is a grid consisting of 51 grid points covering the interval $[1,2]$, mathematically given by

$$x_k = 1 + k\,h\,, \quad k = 0,\cdots,50\,, \quad h = \frac{2-1}{50}\,.$$

Let us collect the grid values in the row vector x. What is the size of the vector x? Present the Matlab command that generates the vector x. Do not use a loop statement, use one call of

linspace instead in which x is generated directly without the necessity to rescale x (thus using a single Matlab command instead of multiple).

Matlab command(s) read:

We need to compute

$$N = \sum_{k=0}^{50} \sin(x_k) .$$

Present both the answer as well as the Matlab commands for this (do not use a loop statement).

Matlab command(s) read:

Answer: $N =$

Exercise IO-1 (IO11.1, September 2016) *(Matlab manual: section 12)*

Using the **disp** command, make a table in the command window with two columns v containing values of x, respectively y, where

$$y = \sin(x), \; x = k\frac{\pi}{10} \text{ and } k = 0, 1, \cdots, 10 .$$

Present the numbers in **format short**. Try to add a top line with text for headings.

Matlab commands read:

Exercise Ma-2 (Ma15.1, September 2016) *(Matlab manual: sections 1.2, 8, 14)*

With $N = 50$ and $h = \frac{2}{N}$, the vectors x, y, z are given by

$$x(n) = nh ,$$
$$y(n) = \sqrt{x(n)} ,$$
$$z(n) = (x(n))^2 , \quad n = 0, \cdots, N .$$

Note that x presents a grid of 51 grid points covering the interval $[0, 2]$.

a. Generate the vector x using the Matlab function **linspace**.

b. Create the vectors y and z without using loop statements.

Hint: Use array operations to form z from x. □

c. Display the functions $\sqrt{x}$, x^2 on the interval [0,2] as graphs. Put them in one figure and make use of the vectors mentioned above. Both graphs must be black. Keeping them black, make the two graphs visually different (use the Matlab command **help plot** to find out how you can do this). Let your plot be checked by the lecturer.

Hint: Use **hold on** to display two plots in one figure. □

a. *Matlab commands read:*

b. *Matlab commands read:*

c. *Matlab commands read:*

Exercise Lo-1 (Lo11.1, September 2008) (*Matlab manual : section 11)*

Compute the numbers $x = (x_n)_{n=1}^{50}$ from

$$x_1 = 0.1$$
$$x_{n+1} = 3.5\ x_n\ (1 - x_n), \quad n = 1, ..., 49.$$

Matlab commands read:

Next, investigate the Matlab command **plot**(x,':o'). What is displayed along the horizontal and vertical axes? A periodic behavior in this 'chaotic' system can be observed. What is the period?

Exercise Fu-1 (Fu11.1, September 2008) *(Matlab manual: section 15.2)*

Given is the function

$$f(x) = \frac{\sin(x)}{x}.$$

Make a function file called fu11.m for this function. *The function file fu11.m reads:*

Next, go to the command window and type

» *clear; y=fu11(x)*

Result: $y =$

Also try

» *clear; x=0.1; y=fu11(x)*

Result:

Exercise Fu-2 (Fu3.5, September 2017) *(Matlab manual: sections 7, 15.2)*

We consider the vector function $\mathbf{f} : \mathbb{R}\, x\, \mathbb{R}^2 \to \mathbb{R}^2$. This function has two components

$$\mathbf{f}(t, \mathbf{x}) = \begin{bmatrix} f_1(t, \mathbf{x}) \\ f_2(t, \mathbf{x}) \end{bmatrix} = \begin{bmatrix} f_1(t, x_1, x_2) \\ f_2(t, x_1, x_2) \end{bmatrix}.$$

In fact, the function is given by

$$f_1 = 2\, x_1 x_2 + \frac{1}{2} x_1 ,$$
$$f_2 = x_1 x_2 + x_2 - \cos(\pi t) .$$

Write a function file with input t, $\mathbf{x}$ and with output the function value **fval**. Both $\mathbf{x}$ and **fval** must appear as column vectors in this function file. The function file reads:

Via a function call, determine

$$\mathbf{f}_a = \mathbf{f}(1, \begin{bmatrix} 1 \\ 1 \end{bmatrix}) = \qquad \mathbf{f}_b = \mathbf{f}(1, \begin{bmatrix} 1 \\ 0 \end{bmatrix}) =$$

Exercise Gr-1 (Gr3.5, September 2012) *(Matlab manual: sections 6, 14)*

It is given that $\lambda = -0.4 + i$. Consider the complex function $Q : \mathbb{R} \to \mathbb{C}$ given by

$$Q(h) = 1 + \lambda h ,$$

the amplification factor of the Euler time stepping method. Our interest is towards its modulus $MQ = |Q|$. Make a plot of MQ as a function of h, where $0 \le h \le 2$. Let your plot be checked by the lecturer. The first positive value of h for which $|Q| = 1$ is denoted by $h_{\max}$. Determine $h_{\max}$ with an accuracy of 0.01. Do this graphically by adding gridlines to the graph using the command **grid** (and if necessary, use the Matlab zoom option in the graphical window).

Hint: h must be along the horizontal axis of the plot. □

Matlab commands used:

Answer: $h_{\max} =$

DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

September 2016

Matlab Manual

Contents

1 Matlab session

The way to start Matlab differs from computer to computer. You may type the command 'matlab' in a command window of the operating system. Often, though, you will have to click on a specific icon in order to run the program.

1.1 Getting started with Matlab

Once you have started Matlab a Matlab command window will appear, showing the command prompt:

```
»                    %    The Matlab command prompt.
```

The line after the prompt is called the command line. On this line you can give Matlab commands. After you have pressed <return>, Matlab will execute the command.

```
» pause(5)           %    Wait 5 seconds before showing the plot.
» plot (x,y)         %    Plot vector y versus vector x.
```

Besides the command window Matlab has graphical windows. Output of plot commands is directed to the graphical window.

The **quit** command enables you to leave Matlab. To terminate a running Matlab command you may use **[Ctrl]+[c]** (Press both the Ctrl button and the c button simultaneously).

By using the **!** symbol you can use the original operating system

```
» ! printer command  %    Execute the printer command belonging to the
                     %    original operating system.
```

Only for short computations it is useful to execute Matlab straightaway from the command line. In general the next procedure is much more practical:

1. Make a script file (see section 15) by means of your favorite text editor or the Matlab Editor/Debugger (see Section 1.3). A script file consists of a sequence of Matlab commands. In general script files will contain the main program and subprograms.
2. If necessary make the additional function files, using the same editor. Through these files we are able to define the functions which play a role in script files.
3. Matlab executes the commands in the script file after you have typed the name of the script file on the command line. Note, however, that the script file should be in the current (working) directory, indicated in the box above the command window.

From the command line background information can be obtained using

1. **help**

*» **help plot*** *%* *gives information on the Matlab command **plot.***

2. **demo**

» **demo** % presents multiple examples of the usage of Matlab.

1.2 Matlab and matrices, a general remark

Suppose that we define vectors x, y and a matrix z by

$x(i) = i$, i = 1, …, 10,
$y(i) = i^2$, i = 1, …, 10,
$z(i, j) = \sin(x(i) * y(j))$, i, j = 1, …, 10.

In most programming languages a computer implementation will use nested loops:

```
» for i = 1:10
      x(i) = i; y(i) = i^2;
  end
» for i = 1:10
      for j = 1:10
            z(i, j) = sin (x(i) * y(j));
      end
  end
```

In Matlab this can be done quite differently, because matrices are basic objects:

```
» x=1:10; y=x.^2; z=sin(x'*y);      % matrix form, the apostrophe stands
                                    % for taking the transpose
```

Both programming styles are possible in Matlab. However, the latter is far more efficient. Therefore, we prefer the latter and all examples will be given in this style.

1.3 The Matlab Editor

It is advantageous to use the Matlab Editor when creating or editing script files. You invoke this editor by typing **edit** at the command prompt or via the "New' or 'Open' button in the upper toolstrip. The Matlab editor has various features to aid in editing script files so that most typing errors can be recognized. For example, text strings, reserved words (if, else, for, end, …) and expressions are all shown in different colours. Saving and running the script is easily done using the Debug menu.

In the Matlab Editor the so-called M-lint code checker is integrated. M-lint basically checks the syntax of the code. In the right vertical bar M-lint displays colored rectangles with syntax messages. The contents of the message will pop up if the colored rectangle is touched with the mouse cursor.

1.4 The Workspace Browser

The Workspace browser is invoked via the upper toolstrip, button 'Layout', and gives a list of current variables (scalars, vectors, matrices), similar to what **whos** (Section 3) does. By double clicking on a variable in the Workspace window the values of this variable are shown, in a separate window (the array editor), enabling inspection and interactive adaptations.

1.5 The property editor

Matlab directs graphical output to the graphical window. In this window the so-called property editor is available. Access is possible via the Edit or View button. Each graphical window contains several graphical objects such as axes and lines. One can select the different objects by clicking on them. Next, using the property editor one may inspect, make changes or add objects. This is in particular convenient in the final stage when it is needed to prepare the plot for inclusion in a report. It is then easy to add a title, label, etc.

2 Lay-out

When you use Matlab's default configuration, the program distinguishes upper case and lower case characters. One says that Matlab is *case sensitive*.

If the default configuration is used, Matlab will also print the result after every command. Typing ; (a semicolon) after the command will suppress this.

```
» x = 2      %     Matlab prints the result

x =
     2

» x = 2;     %     Matlab does not print the result
```

The symbol % (*comment*) is used to give comments.

```
» x = 2      %     gives the value 2 to x and prints the result
             %     printing the result can be suppressed with ;
```

The symbol … (*continuation*) denotes that the command continues on the next line

```
» x =  1 +  2 +  3 +  4 +  5 +  6 +  7 +  8 +  9 + 10 . . .
    + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20;
             %     this command does not fit on one line
```

3 Common commands

quit	:	exit from Matlab
help command name	:	gives information about a command
arrow up / down	:	retrieves preceding and following commands
pause	:	pauses execution, Matlab will continu after <return>
whos	:	gives a list of Matlab variables stored in the memory
clear	:	clears the memory
clc	:	clears the command window
clf	:	clears the graphical window
shg	:	brings the graphical window to the foreground
close	:	closes the graphical window
cputime	:	determines the elapsed cpu time
tic, toc	:	stopwatch timers for the elapsed real time
demo	:	activates Matlab demonstrations

4 Numbers and strings

Numbers can be entered in Matlab in the usual way; however, spaces inside a number should be avoided.

```
» (52/4 -0 .01) * 1 e−3

ans =
     1.2990 e−02
```

Matlab automatically assigns a type for every number you enter. Depending on the type, Matlab chooses an internal representation for these numbers and the corresponding computations. The external representation of a number (e.g. on the screen) can be altered with the format command.

format long e	:	16 digits, (exponential) floating point 3.141592653589793e−02	
format short e	:	5 digits, (exponential) floating point	3.1416e−02
format short	:	5 digits, fixed point	0.0314
format long	:	15 digits, fixed point	0.03141592653590
format	:	returns to the default configuration	

The default configuration is ‘format short’. It might be possible that the local system manager has changed this into short e.

Remark: The **format** command influences only the external representation of real numbers. This command has no influence on the internal representation used by Matlab to process the program and its computations.

Remark: The **format** command is not able to alter the external representation of integer numbers.This can result for example in jumps in tables.

The command **vpa** from the Symbolic Toolbox is helpful for displaying numbers and variables:

```
» x = pi/10000; d = 8; disp(vpa(x,d))
   .31415927e−3
```

In the above the variable d in the call of **vpa** refers to the number of digits to be displayed.

To manipulate text, Matlab uses strings.

```
» disp ('give the spring constant a')
give the spring constant a
```

5 Variables and simple operations

A variable's name has to start with a letter, but may not contain more than 31 letters, digits, or underscores. Matlab is *case sensitive* in its default settings. This means that j and J do not have the same meaning. Matlab automatically reserves space in the computer's memory to store the variable. Variables do not need to be declared before use; Matlab derives the type of the variables by looking at the stored data. So it is possible that the type of a variable changes while a session is in progress.

The basic element of Matlab is the matrix. Depending on the size of the matrix we distinguish scalars (1 x 1 - matrix), vectors (1 x m -, or m x 1 – matrix), etc. Depending on the context Matlab also assigns the type of the variables in the matrix, e.g. real or complex.

The operators **+, −, *, /, ^** can be used for all Matlab variables (x ^ y = x to the power of y). In the scalar case these operations will reduce to the usual computations. At every operation step Matlab checks if the dimensions of the matrices involved are correct.

```
» a = 1; c = 1 + i; v(1) = 1; v(2) = 2; word ='text';
```

The command **whos** (see section 3) gives:

Name	Size	Class
a	1 x 1	double array
c	1 x 1	double array (complex)
v	1 x 2	double array
word	1 x 4	char array

Multiplying a vector v with itself is not possible. If we try this anyhow we get:

```
» w = v * v;

??? Error using ==» *
Inner matrix dimensions must agree.
```

6 Complex variables

A complex number can be defined using the imaginary number i.

```
» c = 1 + i

c =
   1.0000 + 1.0000 i
```

Imaginary and real parts of a variable can be obtained with the functions **real** and **imag**:

```
» a = real (c)

a =
      1.0000
» b = imag (c)

b =
      1.0000
```

The operators **+, −, *, /, ^** also work for complex numbers. With the symbol ’ we conjugate a complex variable:

```
» cgec = c’

cgec =
      1.0000 − 1.0000 i
```

The square of the modulus of c can be computed in the following way:

```
» modc2 = abs(c) ^2

modc2 =
         2.0000
```

An alternative method is:

```
» modc2 = c’ * c

modc2 =
         2.0000
```

7 Matrices and vectors

Matlab stores its variables in matrices of size n x m. If m = 1, we are dealing with a column vector, and if n = 1, with a row vector. When n = m = 1 the matrix represents a scalar. The size of a matrix does not have to be given; Matlab determines the size from the data given by the user. Matlab does not recognize a more general array structure, for example v (-10:100); the lower bound in Matlab is always equal to 1.

We can define matrices in different ways, e.g. by giving values to every element separately. We separate row entries by a space or comma and column entries by a semicolon or a <return>.

```
» A = [1 2 3;4,5,6;7 8 9]          % generating matrix A

  A =
    1   2   3
    4   5   6
    7   8   9

» A = [1 2 . . .
       3 4;
       5 6 7 8]                    % generating matrix A; . . . means continuation

  A =
    1  2  3  4
    5  6  7  8
```

Vectors can also be made using the colon symbol : . Here the first value stands for the initial value, the second for the step size.

```
» x  =  0 : 0.2 : 1                % generating row vector x

x  =
      0      0.2000      0.4000      0.6000      0.8000      1.0000
```

A very convenient alternative to generate the present row vector x is to use the Matlab function **linspace**:

```
» x  =  linspace(0,1,6)            % generating row vector x

x  =
      0      0.2000      0.4000      0.6000      0.8000      1.0000
```

The default vector in Matlab is a row vector as can be seen here. If a column vector is needed, extra measures are needed (see later on).

Sometimes it is good to use one of the following Matlab functions:

zeros(n,m) : gives an n x m matrix with zeros
ones(n,m) : gives an n x m matrix with ones
eye(n) : gives the n x n identity matrix

```
» A = ones(3,3) + 2*eye(3)          % generating matrix A

  A =
     3  1  1
     1  3  1
     1  1  3
```

Matrices can also be built from smaller variables

```
» v = ones(3,1); A = [−v 2*v −v]          % generating matrix A

 A =
    −1  2  −1
    −1  2  −1
    −1  2  −1
```

Sometimes it is useful to contruct matrices by concatenation.

```
» v1= [1 2]; v2= [3  4];
» v = [v1  v2]                  %  or  v = cat (v1, v2)
v =
      1   2   3   4

» E = eye(3); C = zeros(3,2); D = [E C]

D =
     1 0 0 0 0
     0 1 0 0 0
     0 0 1 0 0
```

Large diagonal band matrices can be made by using the function

diag(v,k) : returns a square matrix with on the k-th upper diagonal the entries of the vector v.

```
» vm = ones(6,1); vs = linspace(1,9,5)          % generating vectors vm and vs

vs =
      1   3   5   7   9

» A = diag(vs,-1) + diag(vm,0) + 2*diag(vs,1)
```

```
A =
     1     2     0     0     0     0
     1     1     6     0     0     0
     0     3     1    10     0     0
     0     0     5     1    14     0
     0     0     0     7     1    18
     0     0     0     0     9     1
```

Matrix elements can be used separately or in groups:

A(i , j)	=	A_{ij}
A(: , j)	=	j^{th} column of A
A(i , :)	=	i^{th} row of A
A(i , j1:j2)	=	vector existing of the entries, from column j1 to j2, of row i
A(i1:i2 , j1:j2)	=	matrix existing of the entries, from column j1 to j2, of the rows i1 to i2.

```
» x = A(2,3)
   x =
         6
```

```
» plot ( x(75 : 125) , y(325 : 375) );
```

In case one of the dimensions equals one, we are dealing with a vector. We can refer to this vector with just one index. This index is either a row index or a column index, depending on the type of the vector.

Remark: Matrix elements can also be addressed using one index only. This might sometimes be handy. One-dimensional references are taken column wise by Matlab:

```
» A = [1 2 3; 4 5 6; 7 8 9];
» x = A(6)
   x =
         8
```

However, in many cases such a reference is the result of a programming error, and then one must be aware of the fact that no error message will be given.

8 Matrix and array operations

Arithmetic operations on matrices and vectors come in two distinct forms. ***Matrix sense operations*** are based on the normal rules of linear algebra and are obtained with the usual symbols **+, −-, *, /, ^.** ***Array sense operations*** are defined to act elementwise and are obtained by preceding the symbol with a dot and therefore also often referred to as ***dot arithmetic.***

```
» v = v1.*v2;          %   multiply v1 and v2 element by element
                       %   v(1)=v1(1)*v2(1),....,v(n)=v1(n)*v2(n).

» y = v.^2;            %   y(1)=v(1)^2,...,y(n)=v(n)^2.
```

A very useful matrix operation is presented by taking the transpose

```
» B=A';                %   transpose A
» B=transpose(A);      %   alternatively
```

In the case of matrix operations linear algebra introduces restrictions with respect to the dimensions of the matrices involved. Of course, Matlab checks for this.

```
» v(1) = 1;  v(2) = 2;  v(3) = 3;  A = eye(3);   %      A is the 3x3 identity matrix.
» w = A * v;

??? Error using ==» *
Inner matrix dimensions must agree.
```

Matlab always assumes that the index of a new vector is a column index unless explicitly specified otherwise (see example below). This means that the new vector v above is a row vector and an error message results.

```
» v = zeros(3,1);                          %  v is forced to be  a column vector
» v(1) = 1;  v(2) = 2;  v(3) = 3;  A = eye (3);
» w = A * v;
```

Alternatively, the row vector can be changed into a column vector by taking the transpose.

```
» v(1) = 1;  v(2) = 2;  v(3) = 3;  A = eye(3);
» v = v';                                  %     change  v into a column vector
» w = A * v;
» inprod = v' * v                          %     inner product if v is a column vector.
```

9 Elementary mathematical functions and constants

Some of the predefined constants in Matlab are:

pi	:	the constant pi
i	:	the imaginary unit
inf	:	infinity

The following mathematical functions operate in array sense, i.e. on each element of the variable separately:

abs	:	absolute value or modulus (complex case)
sqrt	:	square root
exp	:	exponential function
log	:	natural logarithm
log10	:	logarithm with base 10
sin	:	sine
cos	:	cosine
tan	:	tangent
atan	:	arctangent (inverse tangent)
round	:	rounds towards the nearest integer
rem	:	remainder after division

For vectors are available:

max(v)	:	maximal element of the vector v
min(v)	:	minimal element of the vector v
sum(v)	:	sum of the elements of the vector v
length(v)	:	returns the size of the v
norm(v)	:	returns the Euclidean norm of the vector v, i.e. **norm**(v)=**sqrt**(**sum**(v.^2)).
norm(v,inf)	:	returns the infinity norm of the vector v, i.e. **norm**(v,inf)=**max**(**abs**(v)).

» ***min(abs(v))*** *% computing the in absolute sense smallest element of v*

For matrices we mention

'	:	returns the transpose of a matrix
size(A)	:	returns the size of a matrix
det(A)	:	returns the determinant of a matrix
transpose(A)	:	returns the transpose of a matrix
eig(A)	:	determines the eigenvalues (and eigenvectors) of the matrix A
lu(A)	:	determines the LU-factorization of the matrix A (using permutations if necessary).

10 Logical programming

10.1 Relational and logical operators

In Matlab six relational operators can be used to compare variables or evaluate expressions

<	:	smaller than
<=	:	smaller than, or equal to
>	:	greater than
>=	:	greater than, or equal to
==	:	equal to
~=	:	not equal to

```
» x=5; y=10; x<y
      1      %     thus true
```

Relational operators can be applied to vectors and then they work elementwise

```
» v = 0:0.1:1; v > 0.5
      0  0  0  0  0  0  1  1  1  1  1
```

The first six elements of *v* are smaller than or equal to 0.5 and the remaining five are larger.

Furthermore there are three basic logical operators:

&	:	and
|	:	or
~	:	not

```
» x=5; y=10; x>1 & y<=5
      0      %     thus false
```

Several built-in logical functions are available. We mention only the **find** function for vector input.

find(v) : returns the indices of the nonzero elements of the vector v

```
» v = 0:0.1:1; find(abs(v−0.5) > 0.2)
      1  2  3  9  10  11
```

At the first three positions and last three positions the elements of v are further away from 0.5 than 0.2.

10.2 Conditional statements

A conditional statement is a command that allows Matlab to take a decision whether to

execute a group of commands that follow the conditional statement, or to skip these commands. In a conditional statement a conditional expression or condition is stated. If the condition is true, a group of commands that follow the statement is executed.

The if-end structure

```
if condition
        commands
end
```

The if-else-end structure

```
if condition
        commands [1]
else
        commands [2]
end
```

The if-elseif-else-end structure

```
if condition
    commands [1]
elseif condition
    commands [2]
else
    commands [3]
end
```

The condition needs to be a relational or logical expression taking the value true or false, in Matlab 1 (true) or 0 (false).

```
» if (x >= 1 & y >= 1)              % both x and y larger than or equal to 1
      a = 1;
  elseif ~(x >= 1) & ~(y >= 1)
      a = −1;                       %     both x and y less than 1
  else
      a = 0;                        %     either x or y less than 1
  end
```

11 Loop statements

FOR - loop

Syntax:

for variable = start value : increment : end value
commands
end

The increment can be positive as well as negative. If not defined, Matlab will use increment 1 as a default.

```
» for j = 0 : 10
    v(j + 1) = j * 0.1;  %     the row vector v needs to start at index one!
  end
```

WHILE - loop

Syntax:

while condition
commands
end

The condition needs to be a relational or logical expression.

```
» x = 1;
» while  (x >10^-6)
     x = x /2;
  end
```

Forced exit from loops

A loop can be terminated with the **break** (which passes control to the first statement after the corresponding end) or the **error** statement (which halts the program). This is in particular convenient to avoid infinite while-loops, because sometimes it is not certain that the condition will be met at a certain moment.

```
tol = 10^-6;                                  %    specify tolerance
criterium = inf;                              %    initialize to a large number
m = 0;                                        %    initialize counter to zero
while criterium > tol
  compute a new approximation x and update the value of the criterium for this x
  m = m+1;                                    %    count
  if m > 1000, break, end                     %    jump out if too many
  % or if m > 1000, error('no convergence'), end   %    halt program if too many
end
```

This example presents a typical implementation of a numerical iterative procedure to obtain a

numerical approximation within a specified tolerance taking into account that the numerical procedure might not converge.

Influence of round-off

```
» x = 0;
» while (x <1)
      x = x+0.1;
  end
» x

x =
      1.1000
```

Note that this is not a proper method when we want to execute the loop statement exactly ten times. In Matlab 10 times 0.1 is just a bit less than 1, because Matlab makes round-off errors. This means that the loop will be executed 11 times in stead of 10 times, and therefore the final value of x will be too big. In such a case it is better to rely upon integer computation.

```
» x = 0;
» for j = 1:10
      x = x+0.1;
  end
```

In order to force integer computation the command **round** (section 9) is often useful.

12 Output

One must be careful in using general print commands. For example, using the print command in the file menu you can print the whole Matlab session. This may lead to unnecessarily long waiting time for other people who want to use the printer.

12.1 The disp command

Output can be directed to the screen with the command disp.

disp (x)	:	print the value of the variable x
disp ('text')	:	print the string 'text'
disp (A(:,5))	:	print the 5-th column of matrix A

With the command **format** or **vpa** we can define how a number is displayed (see section 4).

12.2 The diary command

diary output	:	makes a diary of everything which happens during the session and writes this to the file output. If necessary this file can be printed.
diary off	:	stops monitoring the session.

The **diary** command easily gives an unnecessary amount of output. Therefore use the diary command in a selective way and only at places where output is needed.

```
» diary output      % open the file output as output file
» disp (table);     % display a table containing x and y
» diary off         % the file output contains table
```

12.3 Generating a table using the disp command

An example of a classical approach using loops is:

```
» x = 0:0.1:1; y=exp(x);      % x and y are row vectors
» format compact              % suppress empty lines in output
» disp ('x   y')              % display the header of the table
» for j = 1 : length(x)
      disp([x(j) , y(j)])     % display the values
  end
```

Using Matlab matrix features one may condense this to:

```
» table=[x' y'];
```

» ***disp****(table);* *% display a table containing x and y*

With the **format** command one may influence the appearance of the output a little. Helpful as well is the **vpa** function from the symbolic toolbox:

» ***disp (vpa****(table,8));* *% display a table containing x and y presenting 8 digits*

Even more advanced features are available using the **fprintf** command.

12.4 The fprintf command

Advanced control mechanisms for printing are available with the command **fprintf** (see the example program in section 20 or use **help fprintf**).

12.5 The save command

We can write data to a file.

save data varlist	:	the variables in varlist are written to the file data.mat in binary format. This file can be used in a subsequent session.
save output varlist - ascii	:	the variables in varlist are written legible (8 digits, floating point) to the file output. If necessary this file can be printed.

13 Input

13.1 Command line driven

Variables can be entered in Matlab in different ways. The first method is using the command line. We already explained how to do this. Another way to enter it is:

» *x =* ***input*** *('x-coordinate =');*

This command writes the text from the string to the screen, waits for input (until a <return> is given) and assigns the value given by the user to the variable x.

The following command:

» ***load*** *data*

fetches the binary file data.mat. The file data.mat needs to be produced beforehand in Matlab and besides a list of variables it also contains the values of these variables.

The command

» ***load*** *data.txt*

fetches an ASCII file with the name *data.txt*. The content is stored in a double precision array with the name data. The file *data.txt* is only allowed to contain numbers, separated by blanks. Each line of the file corresponds with a row in the array data. Each number on a line corresponds with a column in the array data. The number of columns must be equal on each line.

13.2 The import wizard

Data can also be imported using a graphical user interface, called the Import Wizard. Choose in upper toolstrip for the option Import Data to start the Import Wizard. Then choose a file in the new window from which data should be read, preview the data, and if you are satisfied, use the next and finish buttons to import the variables in the Matlab workspace. You may check the imported variables afterwards, using **whos** (Section 3) or the Workspace Browser (Section 1.4).

14 Graphical issues

General commands affecting the graphical screen are:

clf	:	clear graphical window
shg	:	show graphical screen (bring graphical screen to the foreground)
close	:	close the graphical window
hold on	:	keep the contents of the graphical screen during new plot commands
hold off	:	erase the contents of the graphical screen before executing new plot commands
print	:	direct the graph to the printer
print filename	:	store the graph in the file 'filename'
figure(n)	:	opens (or jumps to) a graphical window with title 'figure n'

Commands affecting the layout of the graph are:

axis ([xmin xmax ymin ymax])	:	sets the axes according to the given values
grid	:	adds a grid to the graph
title('text ')	:	writes text above the plot
xlabel('text ')	:	writes text underneath de x-axis
ylabel('text ')	:	writes text next to the y-axis
text(x, y, 'text ')	:	writes text at the point (x,y) of the graphical screen
t = **num2str**(var)	:	makes text of the value of var, which can for example be written to the screen with **text**(x,y,t).

Some basic commands are:

linspace(x1,x2,n)	:	generates a grid of n points on [x1,x2]
plot (y)	:	plots the vector y versus the indices 1, ..., n.
plot (x,y)	:	plots the vector y versus the vector x; x and y must be of the same size.
plot (x,y, 'symbol')	:	plots the vector y versus the vector x by using a symbol or color, e.g. ':' (dotted), '--' (dashed), 'o' (circles), 'x ' (crosses), 'y ' (yellow line), 'r' (red line), 'g' (green line)
fplot (@f,[xbegin xend])	:	plots the function f on the interval [xbegin, xend].
plot ([xbegin xend],[ybegin yend])	:	draws a line from (xbegin, ybegin) to (xend, yend).
subplot(m,n,p)	:	creates a matrix of plot areas in the graphical window, jumping to the p-th member (row wise counting).

Remark: **help plot** shows all the possible arguments with **plot**.

```
» x = linspace(0,2,21);  y = sin(pi*x);
» plot(x,y);                          %   plot sin πx on [0,2], plot
                                      %   automatically assigns the right limits
                                      %   to the axes in the graphical screen.
» axis( [0  2  -1  1] );              %   adjust the x-axis and the y-axis; it is
                                      %   preferable to call axis after plot
» xlabel('x');  ylabel('sin x');
» print                               %   send the graph to the printer.

» clf
» hold on
» type = ['- ' ; ': ' ; '- .' ; '- -']   %   spaces to equalize the length of the
                                          %   character strings
»  xgraph = linspace(0,1,101);
» for j = 1 : 4
      omega = j * pi ;                %   plot sin(ωx) on [0,1]
      ygraph = sin(omega*xgraph);     %   for ω = π, 2π, 3π, 4π,
      plot(xgraph,ygraph,type(j,:));  %   distinguish the curves visually
  end
» print figure                        %   store the graph in a file named figure;
                                      %   use help print to see a list of
                                      %   available formats.
```

Some commands for three dimensional graphs are:

plot3(x,y,z)	:	plots a line through the coordinates of vectors x, y, z.
surf(X,Y,Z)	:	plots the matrix Z versus the vectors Y and X, creating a surface.
surfc(X,Y,Z)	:	same as surf, but adds a contour plot beneath the surface.
mesh(X,Y,Z)	:	plots the matrix Z versus the vectors Y and X, creating a surface together with a mesh on it.
meshgrid(x,y)	:	creates matrices for a 2D product grid from two 1D vectors.

```
» t = linspace(0,10*pi,501);
» x = sin(t); y = cos(t);
» plot3(x,y,t)                        %   plots a spiral on the cylinder
                                      %   x^2 + y^2 = 1.
» x = linspace(0,1,11);               %   or x = 0 : 0.1 : 1
» y = linspace(0,2,21);               %   or y = 0 : 0.1 : 2
» [X,Y]=meshgrid(x,y)                 %   build a 2D product grid by
                                      %   forming x ⊗ y
» Z=X.*sin(X+Y);                      %   compute function values for the 231 points
                                      %   of the 2D product grid. Note the usage of
                                      %   dot arithmetic.
» mesh(X,Y,Z);   %    or surf(X,Y,Z)  %   plot x*sin(x+y) on the domain
                                      %   0≤x≤1, 0≤y≤2 using 231 grid points.
```

Moreover, Matlab has some easy to use plotting routines with names starting with **ez**.

Examples are:

ezplot	:	plots a function f(x) on an interval
ezsurf	:	presents a surface of a function f(x,y)

As has been mentioned in section 1.5, the *property editor* can be used conveniently to manipulate the graphical window.

15 Script files, function files

15.1 Script files

A script file is a file that consists of a sequence of Matlab commands: a Matlab program. We make such a file with the help of the Editor. The standard extension for these files is .m, e.g. for a script plotting the sinus function an appropriate name is *plotsin.m*.

```
% plotsin.m:      The script creates a plot of the function sin(x) on the interval [0, π]
clear; clc; clf;                % clear

x = linspace(0, pi, 41);        % create a grid with spacing π/40 on [0, π].
y = sin(x);
plot(x,y);
axis([0 pi -1 1]);              % set limits along axes
title('sin(x)');                % put text above the plot
```

The commands in the script file are executed after the filename without the .m extension, i.e. *plotsin*, is typed on the command line. It is also possible to run a script file from another Matlab program by including the command *plotsin*.

15.2 Function files

By means of function files we can add new functions to Matlab. A function file contains a sequence of commands, like a script file, but by means of input and output variables it is possible to communicate with other files. The filename of a function file is used to execute the function.

A function file has the following form:

```
function    output_variable = function_name(input_variable)
  commands
```

The word **function** on the first line implies that it is a function file and not a script file.

Remark: At the position of function_name you need to fill in your chosen name for the function. The filename of the file containing this function must be the same as this name with the standard extention .m. Similar to names of variables, the function name is not allowed to start with a number.

Both the input variable as well as the output variable may be matrices or vectors.

```
function y = average(v);
% This function computes the average of the elements of vector v.
% The function is stored in the file average.m
```

```
    n = length(v);
    y = sum(v)/n;
```

Having defined the function average, stored in the file average.m, this function is available for scripts and other functions.

```
% examplescript.m:     script file to compute average temperature

T = [17.0 18.5 17.8 17.9 18.3];
averT = average(T);
disp(averT);
```

15.3 Collecting multiple Matlab files

Matlab does not allow for a mixture of scripts and functions in a single file. However, functions and subfunctions can be collected in a single file. As an example we present a single file, called examplescript.m, containing both files from subsection 15.2.

```
function examplescript
% EXAMPLESCRIPT     This function executes the commands as contained in the
%                   script examplescript.m (previous subsection). No input
%                   and output arguments are used.

   T = [17.0 18.5 17.8 17.9 18.3];
   averT = average(T);
   disp(averT);

function y = average(v)
% This function computes the average of the elements of vector v.

   n = length(v);
   y = sum(v)/n;
```

Typing *examplescript* on the command line, excutes the script. However, it is a disadvantage that the workspace can not be accessed easily using the Workspace Browser because functions have local workspaces.

15.4 Parameters and Functions

Functions and subfunctions have local memories and, as a consequence, variables used in the function file are local (i.e. only accessible inside the function), and therefore they are not stored in and/or taken from the central Matlab memory.

```
function y = comp(x);
   y = a * x;                  % the variable a is defined locally and does not
                               % have a value irrespective of the outside world,
                               % i.e. an error message follows.
```

```
function y = comp(x);
   a = 2;                  % the variable a has the value 2, only within this
   y = a * x;              % function and this value is not know elsewhere.
```

Often it is neccessary to use a parameter in a function that gets its value outside the function. You can show Matlab you want to use such a parameter by adding it to the list of that function. In the following example we add the variable a to the list of the function comp:

```
function y = comp(x,a);
y = a * x;                 % the variable a gets its value outside the function
                           % and is passed to the function
```

This is not always possible. It may sometimes be necessary to define a variable globally, by using **global**. If you do so the variable is defined in all program parts that contain the same global declaration.

```
function y = comp (x);
global a;
   y = a * x;              % the variable a must also be defined globally at other
                           % locations, i.e. specifically where a gets its value.
```

Remark: The declaration with **global** should only be used when there is no other possibility. If you use it, it is wise to add some comments. For an example of the use of **global** see subsection 19.1.

Remark: In the context of the courses for which this manual is written a convenient way to communicate parameters, avoiding the global command, is to use nested functions, see subsection 19.2.

.

16 Solving a system of equations

Consider the system of equations $A\,x = b$. A straightforward approach is to set $x = A^{-1}b$, implemented in Matlab as *x = **inv**(A)*b* or *x = A^−1*b*. However, **this approach is not recommended** at all, among others because of the high computational costs.

To solve systems of equations Ax = b we can use several methods:

i) If a small system with a full matrix needs to be solved only once, we may use the black-box backslash operator:

```
» x = A\b;          % black-box option, let Matlab decide what to do
```

Often it is necessary to be more specific to obtain an efficient program. As an example, we mention:

ii) If more than one system needs to be solved with the same matrix A, but different right-hand sides:

```
» [L,U] = lu(A);          % Make an LU-decomposition of A
» y1 = L\b1;              % Solve lower triangular system, forward substitution
» x1 = U\y1;              % Upper triangular system, backward substitution

» y2 = L\b2; x2 = U\y2;   % Solution for a second right-hand side
```

The implementation *x1 = A\b1; x2 = A\b2;* is about twice as expensive.

A *sparse matrix* (Nederlands: ijle matrix) is a matrix containing many zero elements (or few nonzero elements). Sparse matrices often occur in engineering sciences. The simplest measure is to use A = **sparse**(A) in order to declare the matrix as being sparse as soon as solution procedures become active. As an example, we give the number of floating-point operations in case A is a tridiagonal matrix of size 100 x 100.

	without A = **sparse**(A)	with A = **sparse**(A)
method i	378350	2149
method ii	35250	1889

17 Tracking down errors

Matlab does not give specific directions for debugging programs. In any way it is useful to generate (intermediate) output, and if necessary to recalculate it manually. Some useful commands are:

whos	:	gives a table of the variables that are stored in memory. With this command you can check the size.
size(A)	:	gives the size of a matrix A.
disp('labelj')	:	directs the labeled text to the screen; can be used to find out at which line the program does not work properly.
disp(var)	:	prints the actual value of the variable var on the screen. Can be used to check values of intermediate results.

The Workspace Browser (Section 1.4) provides a very simple, but often effective, way to check names, types and sizes of variables. Moreover, variables can be opened in the array editor and then it is easy to check the values of the variables in use.

A more advanced feature can be found in the Matlab editor. After opening a Matlab file one can use the Debug menu, and, among others, set breakpoints.

18 Symbolic computing

Matlab offers the Symbolic Math Toolbox, which is based upon the Maple kernel from Waterloo Maple, inc. This Toolbox allows symbolic manipulation of variables in a way very much similar to Maple, so it might be helpful to consult your Maple Manual. To obtain an overview of functions in the toolbox type **help symbolic** in the Matlab window. A short survey is given below.

sym x	:	declare x to be a symbolic variable
syms x y	:	declare x and y to be symbolic
subs(y,x,value)	:	substitute the 'value' for x into the symbolic expression y

As an example, observe the effect of following commands

```
» syms x  y                    %   declare x, y to be symbolic
» y=sqrt(x)
» y1=subs(y,x,2)               %   substitute x = 2 into y = √x
» y2=subs(y,x,sym(2))          %   y2 will be symbolic
» yv=subs(y,x,1:10)            %   yv will be a vector
```

Expressions may contain several symbolic objects which can be substituted by a call to **subs** with lists (so-called cell arrays in Matlab) as second and third argument. Symbolic integration and differentiation is performed through the **int** and **diff** functions, as in Maple. The resulting expression sometimes appears to be quite complicated; then **simplify**, **factor** or conversion to numeric form, using **double** might help.

diff(f,x)	:	differentiate f with respect to x
int(f,x,a,b)	:	integrate f with respect to x from a to b
simplify(y)	:	simplify the symbolic expression y
factor(y)	:	factorize the symbolic expression y
double(y)	:	convert the symbolic expression into a floating point number

```
» syms x  y
» f=sqrt(1+x^2+y^2)
» f12=subs(f,{x,y},{1,2})      %   substitute x = 1 and y = 2 into f
» f12num=double(f12)           %   convert f12 to floating point
» fx=diff(f,x)                 %   differentiate w.r.t. x
» fxsim=simplify(fx)           %   simplify
```

Expressions can be solved for a variable using **solve**, whereas **dsolve** tries to solve differential equations symbolically. The derivatives are denoted by D, D2, etc., and initial conditions can be passed as additional arguments.

solve (f,x)	:	solve f = 0 for x
dsolve(deq,init)	:	solve the differential equation given by deq for the initial value init

```
» syms x  a  c
» f = a*x^2+c
» x0=solve(f,x)                      %   find the zeros of f
» x0=solve('a*x+c=0')                %   solve ax+c = 0
» y=dsolve('D2x+x=0')                %   solve the differential equation x''+x = 0
» y=dsolve('D2x+x=0','x(0)=1')       %   solve the differential equation with initial
                                     %    value x(0) = 1
```

19 Functions, some advanced issues

Function files, often userdefined, form the heart of Matlab applications. In this section we pay attention to some advanced features related to communication.

19.1 Passing a function as an argument

A function can be passed as an argument in several ways (we mention function handles, inline objects, strings). Following Mathworks, we recommend using function handles.

```
%     program.m :      script file to plot the function f(x) =x^2;
%                      uses the function file f.m

domain = [1 2];
fplot(@f,domain);                %     @ indicates the function handle
```

The function f is given in the following function file:

```
%     f.m          :      Function file presenting the function x^m:
function fvalues = f (x);
   fvalues =  x.^2;              %.Note the usage of array operations
                                 %.The function f is called with an array as input
                                 % variable. This array contains multiple x-values.
```

For short functions it is also possible to prevent the occurrence of a separate function file by using anonymous command line functions.

```
fhandle = @(x) x.^2;             % fhandle is a function handle referring to a
                                 % command line function
fplot(fhandle,domain);
```

or alternatively

```
fplot(@(x) x.^2,domain);
```

19.2 Communicating parameters

In case of anonymous command line functions communicating parameters is quite straightforward. As an example let us plot the function $f(x) = x^m$ with the parameter m obtaining its value outside the function. The function shares the workspace of the program in which the command line function is present. Thus the parameter m is automatically known to the function because m is in the workspace of the script program.m.

```
%     program.m :      script file to plot the function f(x) =x^m;
```

```
m = 2; domain = [1 2];
fplot(@(x) x.^m,domain);
```

In the context of the courses for which this manual is written it is often needed to vary parameters which are present in low-level routines. We can communicate parameters using the **global** command (see subsection 15.4). However, this is not recommended. The alternatives are:

- use nested functions
- use function handles with full lists

The first method is advocated in some text books. The second method is frequently mentioned in the Matlab help.

In order to focus, we consider the initial value problem

$$\begin{cases} \dfrac{dy}{dt} = -a\,y + \sin t\,, \\ y(0) = 1\,. \end{cases}$$

The parameter *a* needs to get its value outside the function file presenting the right-hand side of the differential equation. In the examples below we use the Matlab routine **ode45** for the numerical integration of the differential equation. If needed, it is easy to substitute your own solver, either as a stand-alone routine or as a nested function.

Nested functions

The underlying mechanism for communication of parameters is that nested functions share the same workspace. This enables easy communication of parameters. If functions are nested, then it is obligatory to use **end** on the last line of the function

```
function solvede                        %  level 0
% SOLVEDE solves a simple DE            %  top level

  a = 2;                                %  give a its value at the top level
  y0 = 1 ; tspan = [0 1];
  [t,y] = ode45(@ownrhs,tspan,y0);      %  function handle
  plot(t,y);

   function fvalue = ownrhs(t,y)        % level 1, 1st nested level
   % OWNRHS contains the user-supplied rhs of the DE, ownrhs shares the
   % workspace of solvede and thus it is possible to use the parameter a freely

      fvalue = - a*y+sin(t);

   end              % end of ownrhs

end         % end of solvede
```

Function handles with full lists

The above example can be done on basis of a script file solvede and a separate function file ownrhs.

```
% solvede.m : script file to solve a simple DE

a = 2;
y0 = 1 ; tspan = [0 1];
fhandle = @(t,y) ownrhs(t,y,a);        %  function handle referring to the user
                                       % defined function ownrhs and presenting
                                       % both a list of the basic variables as well as
                                       % a list of the basic variables + parameters
[t, y] = ode45(fhandle, tspan, y0);
plot(t,y)
```

By comparing the two lists Matlab knows how to separate basic variables and parameters. Moreover, the function handle has access to the workspace of solvede, i.e. the value of the parameter a is known to fhandle. The function file accompanying the script file reads:

```
function fvalue = ownrhs(t,y,a)
% the parameter a needs to be in the input list

   fvalue = – a*y+sin(t);
```

Usage of global

For completeness, the option using the global command is presented as well.

```
% solvede.m : script file to solve a simple DE
global a

a = 2;
y0 = 1 ; tspan = [0 1];
[t, y] = ode45(@ownrhs, tspan, y0);
plot(t,y)

function fvalue = ownrhs(t,y)
% global a

   fvalue = – a*y+sin(t);
```

20 Example program, initial value problem

Consider the initial value problem

$$\begin{cases} y_1' = 2y_1 + y_2 + t\,, & y_1(0) = 1\,, \\ y_2' = y_1 + 2y_2 - t\,, & y_2(0) = 0\,. \end{cases}$$

The present system of differential equations is a linear system and of the form

$$y' = \mathbf{f}(t,\mathbf{y})\,, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \mathbf{f}(t,\mathbf{y}) = A\mathbf{y} + \mathbf{g}(t)\,, \quad A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} t \\ -t \end{bmatrix}.$$

For obtaining a numerical approximation of the initial value problem, we apply the Euler's forward method, leading to the basic formula

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mathbf{f}(t_n, \mathbf{w}_n)\,.$$

The first thing we need is a function file implementing the right-hand side of the system:

```
function yacc = rhs(t,y)
% matrix*vector product written out explicitly

   yacc = [2*y(1)+y(2)+t; y(1)+2*y(2)−t];
```

or

```
function yacc = rhs(t,y)
% full matrix-vector formulation

   A = [2 1; 1 2]; g = [t; −t];
   yacc = A*y+g;
```

Next, a loop is implemented doing the time stepping:

```
% set time span, initial value and time step h; determine the time grid.

t_0 = 0; t_end = 2; y_0 = [1; 0];                % y_0 is a column vector
h = 0.1;
nsteps = round((t_end – t_0)/h);                 % take care of integer values
% check for; does the physical t_end coincides with the numerical one (t_0 + nsteps*h)?
ngrid = nsteps + 1;
t = linspace(t_0,t_end,ngrid);                   % t is a row vector

% the approximate solution will be stored in a 2-column row vector with each 2-column
% containing the approximation at the corresponding time level.

w = zeros(2,ngrid); w(:,1) = y_0;                % preallocate for storage, start storage
```

```
% do the time stepping with forward Euler

for n = 1:nsteps
  w(:,n+1) = w(:,n) + h*rhs(t(n),w(:,n));
end
```

Finally, plotting and printing is done:

```
plot(t,w(1,:),t,w(2,:),':');    % second component of solution with a dotted line

fprintf('   t          y(1)              y(2)\n');                % print header
for n = 1:ngrid
  fprintf('%6.1f  %17.7e  %17.7e  \n', t(n), w(:,n));   % print solution per line
end
```

fprintf *is actually a C-command (included in Matlab)*
In the format string of fprintf (the part between quotation marks), text which needs to be printed is given, and for every number which needs to be printed, the format is given.

%6.1f	*means: fixed-point format with 1 decimal and 6 positions (field width).*
%17.7e	*means: floating point (exponential) format with 7 decimals and 17 positions.*
\n	*means: continue printing on the next line*

After the format string, the variables which need to be printed follow.

The field width is chosen in such a way that extra blanks take care that everything looks nice.

21 Example program, boundary value problem

Consider the boundary value problem

$$\begin{cases} -y'' + 2y = \cos(x), \quad x_{left} \le x \le x_{right}, \quad x_{left} = 0, x_{right} = 1, \\ y(0) = y_{left}, y(1) = y_{right}, \quad y_{left} = 3, y_{right} = 1. \end{cases}$$

For obtaining a numerical approximation of the solution, we apply the finite difference method with mesh size h, leading to the basic formula

$$-\frac{w_{j-1} - 2w_j + w_{j+1}}{h^2} + 2w_j = \cos(x_j), \; j = 1, \cdots, n \quad (x_j = x_{left} + j\,h).$$

Here, n stands for the number of interior nodal or grid points. Let N denote the number of unknown values of the solution on the grid. Because of the two Dirichlet conditions it holds that $N = n$ and no extra efforts are needed. The boundary values are just filled in, via $w_0 = y_{left}$, $w_{n+1} = y_{right}$, and the resulting terms are transported to the right-hand side. This leads to the system $Aw = b$ with the matrix A of size $N\,x\,N$.

```
% set interval, boundary values and spatial grid size h
% determine grid, number of unknowns

xleft = 0; xright = 1; yleft = 3; yright = 1;
h = 0.1;
n_interval = round((xright − xleft)/h);          % take care of integer values
% check for; does the physical xright coincides with the numerical one (xleft + n_interval*h)?
n = n_interval − 1; n_grid = n_interval + 1;
x = linspace(xleft,xright,n_grid);               % x is a row vector
N = n;                                           % number of unknowns

% build the right-hand side of the system Ax = b

b = cos(x(2:N+1)); b = b';          % x(1) contains xleft, b must be a column vector
b(1) =b(1) + yleft/h^2; b(N) = b(N) + yright/h^2;      % account for boundary terms

% build the tridiagonal matrix A, either using loops or using diag

A = zeros(N,N);                                  % create a N x N matrix filled with zeros
for j = 1:N; A(j,j) = 2/h^2 + 2; end;                          % fill main diagonal
for j = 1:N-1; A(j,j+1) = −1/h^2; A(j+1,j) = −1/h^2; end;      % fill subdiagonals
```

or

```
v = (2/h^2 + 2)*ones(N,1);                       % main diagonal of length N
vp1 = -1/h^2*ones(N-1,1); vm1 = vp1;             % subdiagonals of length N-1
A = diag(vm1,-1) + diag(v,0) + diag(vp1,1);
```

% solve the system Ax = b

w = A\b; *% use the black-box backslash operator*
or

[L,U] = ***lu****(A);s = L\b; w = U\s;* *% use LU-decomposition*

% plot the solution

w = [yleft; w; yright]; ***plot****(x,w);* *% include the boundary*

The matrix A is a sparse matrix and using sparse options is favorable.
A first possibility is to include the command A = sparse(A) after having build A as above.
Even better is to use sparse techniques right from the start. In this case only nonzero elements of A are involved while building A and in the remainder of the matrix automatically zeros are placed. Thus initialization of the zeros is not needed. Next, one may use

*v = (2/h^2 + 2)*ones(N,1); vp1 = -1/h^2*ones(N,1); vm1 = vp1;* *% the vectors containing subdiagonals now also have length N*
A = ***spdiags****(vm1,-1,N,N) +* ***spdiags****(v,0,N,N) +* ***spdiags****(vp1,1,N,N);* *% shorter: A =* ***spdiags****([vm1 v vp1],-1:1,N,N)*

22 Reference and index

In this section the following notation holds:

n,m - scalar
A - matrix
v,w,b - vector
x,y - arbitrary
f - user supplied function file

Command	*Explanation*	*Page*
[ ]	are used for creating matrices and vectors	
( )	are used to : - indicate the order of operations - embrace arguments of functions - embrace indices of matrices and vectors	
…	Three or more dots at the end of a line indicate that the line continues on the next line	*8*
,	symbol that separates between row elements in a matrix.	*13*
;	symbol that separates between distinct rows in a matrix. We can also use the semicolon to suppress the display of computations on the screen and to separate different commands on a line	*8, 13*
%	All text on a line after the symbol % will be regarded as comment	*8*
!	used to insert operating system commands	*8*
:	used for the generation of variables in **for**-loops and used to select matrix elements: A(:,n) is the n^{th} column of A, A(m,:) the m^{th} row	
'	transposes matrices and vectors	*16*
.*	v .* w : multiplication of two vectors by multiplying element by element (operation in array sense)	*6, 16*
\	A\b gives the solution of Ax = b	*31*
^	x ^ y = x to the power y	*11*
&	logical and	*18*
|	logical or	*18*
~	logical not	*18*
abs	**abs**(x) is the absolute value of (the elements of) x	*12, 17*
atan	**atan**(x) is the arctangent of (the elements of) x	*17*
axis	**axis**(v), with v = [xmin xmax ymin ymax] replaces the automatical scaling of a graph's axes by the configuration given by the vector v. **axis** ('square') switches over from a rectangular graphical screen to a square-shaped graphical screen, **axis** ('normal') switches back to the rectangular screen.	*25*
break	jumps out of a loop	*20*
clc	**clc** clears the command window and moves the cursor to the upper left corner	*9*
clear	**clear** clears the variables from the Matlab memory. **clear** x {y} removes the variable x {and y}	*9*

clf	**clf** clears the graphical window	*9, 25*
close	**close** closes the graphical window	*9, 25*
cos	**cos**(x) is the cosine of (the elements of) x	*17*
cputime	**cputime** determines the cpu time	*9*
demo	**demo** starts a demonstration	*6, 9*
det	**det**(A) determinant of A	*17*
diag	**diag** (v,k) returns a square matrix with the elements of the vector v on the k^{th} upper diagonal	*14*
diary	**diary** *filename* puts all following commands and results in a file with the name *filename*. This stops after **diary off** is entered	*22*
diff	**diff**(f,x) differentiates f w.r.t. x	*33*
disp	**disp**('*text*') writes *text* on the screen. **disp**(x) displays the value of the variable x, without variable name	*10, 22*
double	**double**(expr) converts the symbolic expression to a number	*33*
dsolve	**dsolve**(deq) solves the differential equation deq	*33, 34*
eig	**eig**(A) computes the eigenvalues and eigenvectors of A	*17*
else, elseif	see **if**	*19*
end	**end** is the command that ends loop statements, conditional statements and nested functions	*19, 20 36*
exp	**exp**(x) is the exponent of (the elements of) x with base e	*17*
eye	**eye**(n) is the nxn identity matrix. **eye**(m,n) is an mxn matrix with ones on the diagonal and zeros elsewhere	*13*
factor	**factor**(expr) factors the symbolic expression expr	*33*
figure	**figure**(n) opens a new graphical window with title 'figure n'. If the window already exists it jumps to it.	*25*
find	**find**(v) returns the indices of the nonzero elements of v	*18*
for	loop statement: **for** variable = start value : increment : end value, commands **end**	*6, 20*
format	formats the output: **format short** - 5 digits, fixed point **format short e** - 5 digits, floating point **format long** - 15 digits, fixed point **format long e** - 15 digits, floating point standard configuration is **short**. **format compact** suppresses extra empty lines in the output **format loose** adds empty lines	*10*
fplot	**fplot**(@f,[a,b]) plots the function f on [a,b]	*25, 35*
fprintf	C-command for advanced formatting of output	*23, 39*
function	user defined function: **function** outputvar = *functionname* (inputvars) commands Function files have to have the name *functionname.m*.	*28*
global	**global** x changes x into a global variable	*30, 37*
grid	adds a grid (a lattice of horizontal and vertical lines) to a graph	*25*

help	**help** shows the functions you can get information about. **help** *functionname* shows this information on the screen. **helpwin** generates an extra window with helptopics	*5, 9*
hold	**hold on** keeps the last plot in the graphical screen. A new graph will be plotted on top of the existing one. **hold off** restores the default settings. In that case, when a plot command has been given, the graphical screen will be cleared before the new graph is be plotted	*25*
i	the imaginary unit	*17*
if	conditional statement: **if** statement commands **else** commands **end** **if** statement commands **elseif** statement commands **else** commands **end**	*19*
imag	**imag**(c) returns the imaginary part of the complex number c	*12*
inf	infinity	*17*
input	**input**('*text*') displays *text* on the screen and waits for input by the user. Can be assigned to a variable	*24*
int	**int**(f,x,a,b) integrates f w.r.t. x from a to b	*33*
length	**length**(v) returns the number of elements of the vector v	*17*
linspace	**linspace**(x1,x2,n) generates a n-point grid on [x1,x2]; i.e. grid with spacing (x2-x1)/(n-1)	*13, 25*
load	**load** *filename* loads the variables of the file *filename* into the memory. The file is of type .mat or of type .txt.	*24*
log	**log**(x) is the natural logarithm of (the elements of) x	*17*
log10	**log10**(x) is the logarithm with base 10 of (the elements of) x	*17*
lu	**lu**(A) computes the **lu** factorization of the matrix A	*17, 31*
max	**max**(v) returns the largest element of the vector v	*17*
mesh	**mesh**(x,y,Z) plots the matrix Z versus the vectors y and x, creating a mesh	*26*
meshgrid	**meshgrid**(x,y) creates a 2D grid from the vectors x,y	*26*
min	**min**(v) returns the smallest element of the vector v	*17*
norm	**norm**(v) computes the Euclidean norm of v and **norm**(v, inf) computes the infinity norm	*17*
num2str	**num2str**(*var*) converts the number *var* to a text string	*25*
ones	**ones**(n) is an nxn matrix filled with ones, **ones**(m,n) is an mxn matrix	*13*
pause	**pause** pauses the running programme and waits until the user presses any key. **pause**(n) pauses during n seconds	*5, 9*
pi	**pi** is the machine's representation of π	*17*
plot	Drawing a graph: **plot**(v) - plot the vector v versus its indices **plot**(v,w) - plot the vector w versus the vector v **plot**(m,n,'symbol') - put a symbol at position (m,n) in the graph. The following symbols can be used: +, *, o and x	*5, 25*

plot3	**plot3**(x,y,z) plots a line through the coordinates of vectors x, y, z	*26*
print	direct the graph to the printer. **print** *filename* stores the graph in the file *filename*	*25*
quad	library routine: **quad**(*f,0,1*) computes the integral of the function *f* on [0,1].	
quit	logout from Matlab	*5, 9*
real	**real** (c) returns the real part of the complex vector c	*12*
rem	**rem**(m,n) returns the remainder after division of m by n	*17*
round	**round**(x) rounds the elements of x to the nearest integer	*17*
save	**save** *filename* x {y} saves the variable x {and y} into the file *filename*.mat	*23*
script file	a script file is a file consisting of a sequence of Matlab commands. After you have typed the file name at the command prompt these commands will be executed.	*28*
shg	**shg** shows the most recent graphical screen	*9, 25*
simplify	**simplify**(expr) simplifies the symbolic expression expr	*33*
sin	**sin**(x) is the sine of (the elements of) x	*17*
size	[m,n] = **size**(A) returns the size of the matrix A	*17, 32*
solve	**solve**(expr,x) solve expr=0 for x	*33, 34*
sparse	**sparse**(A) saves computations as well as memory space. It can be used when A is a band matrix, and the computations only involve non-zero elements.	*31, 41*
sqrt	**sqrt**(x) is the square root of (the elements of) x	*17*
subplot	**subplot**(m,n,p) breaks the figure window into a m-by-n matrix of plot areas and selects the p-th member for the current plot, where counting is done row wise.	*25*
subs	**subs**(y,x,value) substitutes x=value into y	*33*
sum	**sum**(v) is the sum of the elements of the vector of v	*17*
surf	**surf**(x,y,Z) plots the matrix Z versus the vectors y and x, creating a surface	*26*
surfc	same as **surf**, but adds a contour plot beneath the surface	*26*
sym	**sym** x declares x to be symolic	*33*
syms	**syms** x y declares x and y to be symbolic	*33, 34*
tan	**tan**(x) is the tangent of (the elements of) x	*17*
text	**text**(m,n,'*text*') writes *text* at position (m,n) of the graphical screen	*25*
tic,toc	**tic and toc** are stopwatch timer for the elapsed time	*9*
title	**title**('*text*') writes *text* as a title above the graph in the graphical screen	*25*
vpa	**vpa**(x,d) takes x using d digits	*10, 23*
while	conditional loop statement: **while** statement commands **end**	*20*
whos	**whos** shows the name, size and type of the variables in the Matlab memory	*9, 11 32*
xlabel	**xlabel**('*text*') places *text* underneath the x-axis of the graphical screen	*25*
ylabel	**ylabel**('*text*') places *text* next to the y-axis of the graphical screen	*25*

zeros	**zeros**(n) returns an nxn matrix with zeros; **zeros**(m,n) returns an mxn matrix	*13*