DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology and Systems

Matlab Manual

September 2002

Contents

1 Matlab session

The way to start Matlab differs from computer to computer. You may type the command 'matlab' when you are working under DOS or UNIX. 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 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 this type of windows.

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 chapter 14) by means of the available text editor. 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.

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

$x(i) = i \qquad , i = 1, \ldots, 10,$
$y(i) = i^2 \qquad , i = 1, \ldots, 10,$
$z(i, j) = \sin(x(i) + y(j)) \qquad , i, j = 1, \ldots, 10.$

In most programming languages a computer implementation will be done using 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 = 1:10;  y = y.^2;
≫ z = sin(x+ y);
```

Both programming styles are possible in Matlab. In this manual we prefer the latter and all examples will be given in this style.

2 Lay-out

When you use Matlab's default configuration, the program distinguishes upper case and lower case characters.

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

≫ x = 2   %   Matlab prints the result

x =
2

≫ x = 2;   %   Matlab does not print the result

The symbol % 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 … 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	:	leave Matlab
help command name	:	gives information about a command
↑↓	:	retrieves preceding and following commands
pause	:	pauses execution, Matlab will proceed 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
cputime	:	determines the elapsed cpu time
demo	:	activates Matlab demonstrations

4 Numbers and strings

Numbers can be entered in Matlab in the usual way.

*≫ (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.

To manipulate text, Matlab uses strings.

*≫ **disp** ('give the spring constant a')*
give the spring constant a

5 Variables

A variable's name has to start with a letter, but may not contain more than 31 letters, digits, or underscores. 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
```

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 = c’ * c

modc2 =
        2.0000
```

An alternative way is:

```
≫ modc2 = abs(c) ^2

modc2 =
        2.0000
```

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
```

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 with a space or comma and columns with a semicolon.

```
≫ 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 and the third for the final value in the vector.

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

x  =
      0      0.2000      0.4000      0.6000      0.8000      1.0000
```

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 variables

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

 A =
      -1  2  -1
      -1  2  -1
      -1  2  -1
```

Matrices can also be constructed from smaller matrices

```
≫  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.

```
≫ v = 1 : 2 : 9                         %      generating vector v

v =
       1   3   5   7   9

≫ A = diag(v,-1) + eye(6) + 2*diag(v,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.

```
≫ 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.

The operators +, -, *, /, ^ can also be used for matrices. For every operation Matlab checks if the dimensions of the matrices involved are correct.

```
≫ v(1) = 1;  v(2) = 2;  v(3) = 3;  A = eye(3);
≫ w = A * v;

???  Error using → *
Inner matrix dimensions must agree.
```

Matlab assumes that the index of every new vector is a column index, i.e. unless explicitly specified otherwise (see example below), the new vector is a row vector. Hence in the example above v is a row vector, which results in an error message.

```
≫ v = zeros(3,1);%  v is now a column vector
≫ v(1) = 1;  v(2) = 2;  v(3) = 3;  A = eye (3);
≫ w = A * v;
```

If we use . (dot) as a prefix, we can do some of the operations element by element.

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

Furthermore the next functions and operations are often quite useful

’	:	returns the transpose of a matrix
size(A)	:	returns the size of the matrix A.
length(v)	:	returns the size of the vector v.
norm(v)	:	returns the Euclidian 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)).

```
≫ B = A’;              %    transpose A
≫ 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.

inprod =
        14

≫ normv2 = norm(v)
```

normv2 =

3.7417

≫ *normvinf = **norm**(v, inf)*

normvinf =

3

8 Elementary mathematical functions and constants

Some of the predefined constants in Matlab are:

pi	:	the constant pi
i	:	the imaginary unit

The following mathematical functions operate on each element of the variable separately:

abs	:	absolute value
sqrt	:	square root
exp	:	exponential function
log	:	natural logarithm
log10	:	logarithm with base 10
sin	:	sine
cos	:	cosine
tan	:	tangent
atan	:	arctangent
round	:	round off to the nearest integer
rem	:	remainder after division

For vectors are available:

max	:	maximal element
min	:	minimal element
sum	:	sum of the elements
norm	:	the norm ‖ v ‖

≫ ***max(abs(****v****))*** % *computing the largest element of v, irrespective of its sign*

For matrices we mention

eig	:	determines the eigenvalues of a matrix
lu	:	determines the LU-factorization of a matrix (using permutations if necessary).
chol	:	determines the Cholesky factorization of a symmetric matrix

9 Conditional statements

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

Furthermore there are three logical operators:

&	:	and
|	:	or
~	:	not

An example of a logical expression is

x > 1 **&** *(y <= 5 | z = 1)*

Conditional statements are, for example:

Syntax conditional statement:

```
if condition
    commands
end
```

Syntax single conditional statement:

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

Syntax multiple conditional statement:

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

The condition has to be a logical expression.

An example:

```
≫ if (x >= 1 & y >= 1)
      a = 1;
  elseif ~(x >= 1) & ~(y >= 1)
         a = -1;
     else
         a = 0;
  end
```

10 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 * .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 logical expression.

```
≫ x = 0;
≫ while  (x <= 1)
      x = x + .1;
  end
```

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 then 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:

```
≫ x

x =
      1.1000
```

11 Output

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** we can define how a number is displayed (see section 4).

<u>Remark:</u> Advanced control mechanisms for printing are available with the command **fprintf.** (see the example program or use **help fprintf**)

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.
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.

```
≫ diary output            %   open the file output as output file
≫ disp ('x   y')          %   display the header of the table
≫ for  j = 1 : 10
     disp ([x(j) , y(j)])  %   display the values
  end
≫ diary off
```

<u>Remark:</u> The **diary** command easily gives an unnecessarily amount of output. Therefore: Use the **diary** command in a selective way and only at places where output is needed.

<u>Remark:</u> Files can be printed using the printer command of the original operating system, see section 1.

<u>Remark:</u> Depending on the operating system other printer commands might be available. Be careful in using these commands, because they can cause an unnecessarily long waiting time for other people who want to use the printer. Using the printer command under MS-windows (in the file menu) you can print the whole Matlab session.

12 Input

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 Graphical Output

General commands in favour of the graphical screen are:

clf	:	clear graphical window
shg	:	show graphical screen (bring graphical screen to the foreground)
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’

Commands in favour of 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).

The most important plot command in favour of data representation is plot:

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 length.
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)
plot ([xbegin xend],[ybegin yend])	:	draws a line from (xbegin, ybegin) to (xend, yend).
zoom	:	makes it possible to zoom in at a specific point in the graph by clicking at that point.
ginput(n)	:	gets the coordinates of n points in the graph, which are located by positioning the cursor and thereupon clicking the mouse.

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

```
≫ x = [0:20]/10;  y = sin (pi * x);
≫ plot (x,y);                          %   plot sin πx on [0,2], plot
                                       %   automatically assigns the right
                                       %   dimensions to the graphical screen.
≫ axis ( [0  2  -1  1] );              %   adjust the x-axis and the y-axis; it is
                                       %   preferable to call axes 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 = [0:100]/100;
≫ for j = 1 : 4
      omega = j * pi ;
      ygraph = sin (omega * xgraph);   %   plot sin(ωx) on [0,1] for ω = π, 2π,
      plot (xgraph,ygraph,type(j,:));  %   3π, 4π distinguish the curves visually
  end
≫ print figure                         %   store the graph in a file; use
                                       %   help print to see a list of available
                                       %   formats.
```

The commands in favour of 3 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 mesh.
rotate3d	:	enables the user to rotate the plot by mouse.

```
≫ t = [0:500] * pi / 50;
≫ x = sin(t); y = cos(t);
≫ plot3(x,y,t)                         %   plots a spiral on the cylinder
                                       %   x^2 + y^2 = 1.
≫ x = [0:10] / 10;
≫ y = [0:20] / 10;
≫ for j = 1:21                         %   z = sin (x + y), note that
      z(j, :) = sin( x + y(j));        %   x runs over columns and
  end                                  %   y over rows.
≫ mesh(x,y,z);                         %   plot sin(x+y) for x = 0 : 0.1 : 1
                                       %   and y = 0 : 0.1 : 2
```

14 Script files, function 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. program.m. Every Matlab command is closed off with <return>. This also holds for the last line of the sript file. The commands in the script file are executed after the filename is typed on the command line (without extension .m, i.e. in the example above you need to type 'program'). It is also possible to run a script file from Matlab programs (see chapter 17).

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 ouput variables it is possible to communicate with other files. The filename of a function file, consisting of at the most 8 digits with standard extension .m, is used to execute the function.

Globally a function file has the following form:

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

The word function in the first line indicates it is not a script file.

<u>Remark</u>: At function_name you need to fill in the file name without the extension .m .

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

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

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

For the use of function files we refer to the example program (see section 17).

Variables used in a function file are local (i.e. only accessible inside the function), and therefore they are not stored in the Matlab memory.

```
function y = comp(x);
        y = a * x;              %   the variable a is defined locally and does not
                                %   have a value

function y = comp(x);
        a = 2;                  %   the variable a has the value 2, but only within this
                                %   function
        y = a * x;
```

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

```
function y = comp(x,a);
    y = a * x;          %   the variable a gets its value outside the function and is
                        %   given 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 use the same declaration.

```
function y = comp (x);
    global a;
    y = a * x;          %   the variable a is also defined globally at another
                        %   location 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.

An example of a situation in which the occurrence of **global** is unavoidable is the usage of a library routine with a predefined function, where it is needed to supply the function with an additional parameter for a specific application.

```
quad(f,0,1);            %   library routine to compute the integral of the function
                        %   f on [0,1]; the argument f refers to a user-supplied
                        %   function file
```

```
function y = f (x);
    global m;
    y = x ^ m;
```

15 Solving a system of equations

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 without the need to go into details, we may use the black box option:

```
≫ x = A\b;
```

Often it is necessary to go into more details to obtain an efficient program. As an example we mention:

ii) If more than one system needs to be solved, and A is the same but the right-hand side is always different:

```
≫ [L,U] = lu(A);             %  Make an LU-decomposition of A
≫ y = L\b;                   %  Solve lower triangular system
≫ x = U\y;                   %  Solve upper triangular system
≫ y2 = L\b2; x2 = U\y2;      %  Solution for other right-hand side
```

iii) If the matrix is symmetric and positive definite, we may use the Cholesky decomposition:

```
≫ R = chol(A);          %  Make a Cholesky decomposition of A: R' R = A
≫ y = R'\b;             %  Solve lower triangular system
≫ x = R\y;              %  Solve upper triangular system
```

In all cases the command A = **sparse**(A) can save a lot of runtime and memory space when A is a band matrix. As an example we give the number of floating-point operations for the case where A is a symmetric tridiagonal matrix of size 100 x 100.

	without **sparse**(A)	with **sparse**(A)
method i	378350	2149
method ii	35250	1889
method iii	358750	1393

16 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 re-calculate it manually. Some useful commands are:

echo on	:	gives an echo-print of all Matlab commands that are executed. With this it is possible to locate the exact moment on which the program fails.
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.

17 Example program, time integration

```
% This program computes the numerical solution of dy/dt=f(t,y) with Heun's method.

% Filename          : exprog.m
% Function-file     : exf.m
% Script-files      : exheun.m, extabel.m             ex stands for example

% Screen management:
    clear;          % Clear work memory
    clc;            % Clear text window
    clf;            % Clear graphical screen

y0=[1; 0];                                  % Initial condition
t0=0;                                       % Start time
tend=5;                                     % End time
nrsteps=input('Give the number of steps:');
h=(tend-t0)/nrsteps;                        % Time step

exheun          % Call for the script file containing Heun's method
extabel         % Call for the script file designed to print the results
                % The commands from a script file are substituted directly into this program.

pause(5);       % The program pauses for a while before it shows the graph
hold on;
plot(tgraph,ygraph(1,:),'-',tgraph,ygraph(2,:),'--');   % components of y are distinguished by
                                                        % plotting the 2nd component with a broken line
axis([t0 tend -5 5]);                                   % it is preferable to call axis after plot
title('Initial value problem: Name compulsory!');
xlabel(['Solution with Heun with h=' num2str(h)]);
hold off;
```

```
% Heun's method. Script file with filename exheun.m.

% Assuming (t0, y0) "nrsteps" steps are being executed
% The derivative of the diff. eq. is given in the function file exf.m.
% The results are put in tgraph and ygraph

tgraph=[t0];
ygraph=[y0];                % start saving results

y=y0; t=t0;
for j=1:nrsteps

   k1=h*exf(t,y);
   k2=h*exf(t+h,y+k1);
   ynew =y+(k1+k2)/2;
   tnew =t+h;

   tgraph=[tgraph tnew];    % Paste the new results
   ygraph=[ygraph ynew];    % after the old results

   t=tnew;
   y=ynew;

end;
```

```
% Vectorfunction f. Function file with filename exf.m

function      yacc = exf(t,y);

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

```
% Print result. Script file with file name extabel.m.

% The results which are stored in tgraph and ygraph are
% printed in an 8-digit floating-point format in a table.

% In order to make a hardcopy of the tables you need to remove
% the first and the last two comment symbols (%).

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

%    %5.1f         means: fixed-point format with 1 decimal and 5 positions.
%    %15.7e        means: exponential format and floating-point with 7 decimals and 15 positions.
%    \n            means: continue printing on the next line

% After the format string the variables which (possibly) need to be printed follow.

% the actual file starts now:
% diary output

fprintf('Heun's method, h=%5.3f \n',h);
fprintf('step    t        y(1)          y(2)\n');
for k=0 :  nrsteps/10 : nrsteps                                          % print every 10th result
     fprintf(' %4.0f %5.1f %15.7e %15.7e\n',k,tgraph(k+1),ygraph(1:2,k+1));    % note that nrsteps needs to a
end;                                                                     % multiple of 10.

% diary off
% !lpr output     % Make a hard copy of the file output
```

```
% Results of the example program

Heun's method, h=0.100
step    t        y(1)              y(2)
  0    0.0    1.0000000e+00     0.0000000e+00
  5    0.5    5.2536330e-01    -2.9586400e-01
 10    1.0    5.7914644e-01    -5.2647651e-01
 15    1.5    8.4164231e-01    -8.2955459e-01
 20    2.0    1.2051207e+00    -1.2023466e+00
 25    2.5    1.6240001e+00    -1.6233635e+00
 30    3.0    2.0751573e+00    -2.0750112e+00
 35    3.5    2.5455986e+00    -2.5455650e+00
 40    4.0    3.0276755e+00    -3.0276678e+00
 45    4.5    3.5167996e+00    -3.5167979e+00
 50    5.0    4.0101983e+00    -4.0101979e+00
```

18 Example program, filling a penta-diagonal matrix

```
% We will give two methods in this example in order to construct the N x N dimensional pentadiagonal matrix A:
%
%                 [   5  -4-b    1     0     0     0                              ]
%                 [ -4+b    6  -4-b    1     0     0                              ]
%                 [   1  -4+b    6  -4-b     1     0                              ]
%                 [   0     1  -4+b    6  -4-b     1                              ]
%                 [              .     .     .     .     .                        ]
%       A = c     [                    .     .     .     .     .                  ] ,
%                 [                    0     1  -4+b     6  -4-b     1     0      ]
%                 [                    0     0     1  -4+b     6  -4-b     1      ]
%                 [                    0     0     0     1  -4+b     6  -4+b      ]
%                 [                    0     0     0     0     1  -4+b     5      ]
%
% where c and b are given constants. Note that the matrix A is symmetric for b = 0.

% Method 1: Construction using a loop.
% In order to make the commands in the loop also valid for the first and the last column we first add some extra columns.

 B=zeros(N,N+4);          % Creates a  N x (N+4) matrix, consisting of zeros

 for j=1:N
          B(j,j:j+4)=c*[1 -4+b 6 -4-b 1];
 end;                     % Assigns values to the coefficients B(j,j), B(j,j+1), B(j,j+2), B(j,j+3) and B(j,j+4)

 A=B(:,3:N+2);            % Copies B, without the first two and last two columns
 clear B;                 % Removes temporary matrix B

 A(1,1)=5*c;
 A(N,N)=5*c;              % Changes the upper-left and the lower-right coefficient of A

 A=sparse(A);             % Only the non-zero elements of A are stored into the memory.
                          % When N is large it is better to use the non-zero elements of A only.
                          % For an LU-decomposition this saves computing time as well as memory
                          % space.

% Method 2: Construction on the basis of the diagonals of A
% Firstly we make 5 vectors, all containing the elements of a diagonal.

 vm2      = c * ones(N-2,1);     % This diagonal contains (N-2) elements, all equal to c
 vp2      = c * ones(N-2,1);
 vm1      = c*(b-4)*ones(N-1,1);
 vp1      = c*(-b-4)*ones(N-1,1);

 v  =c*[5; 6*ones(N-2,1); 5];    % The first and last element of the main diagonal have different values.
                                 % We construct a column vector consisting of sixes, by 6 * ones(N-2,1),
                                 % and add the fives at the beginning and the end. Note that we use
                                 % the semicolons because we are making a column vector.

 A=diag(vm2,-2)+diag(vm1,-1)+diag(v,0)+diag(vp1,1)+diag(vp2,2);
 clear vm2 vm1 v vp1 vp2

 A=sparse(A);                    % See above
```

19 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

Comand	***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	*6*
,	symbol that separates between row elements in a matrix.	*4*
;	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 one line	*6*
%	All text on a line after the symbol % will be regarded as comment	*4*
!	used to insert operating system commands	*4*
:	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	*13*
.*	v .* w : multiplication of two vectors by multiplying element by element	
\	A\b gives the solution of Ax = b	
^	x ^ y = x to the power y	
&	logical and	*16*
|	logical or	*16*
~	logical not	*16*
abs	**abs**(x) is the absolute value of (the elements of) x	*10,15*
atan	**atan**(x) is the arctangent of (the elements of) x	*15*
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.	*21*
chol	**chol**(A) yields the Cholesky decomposition of the matrix A.	*15*
clc	**clc** clears the command window and moves the cursor to the upper left corner	*7*

clear	**clear** clears the variables from the Matlab memory. **clear** x {y} removes the variable x {and y}	*7*
clf	**clf** clears the graphical window	*7,21*
cos	**cos**(x) is the cosine of (the elements of) x	*15*
cputime	**cputime** determines the cpu time	*7*
demo	**demo** starts a demonstration	*4,7*
diag	**diag** (v,k) returns a square matrix with the elements of the vector v on the k^{th} upper diagonal	*12*
diary	**diary** *filename* puts all following commands and results in a file with the name *filename*. This stops after **diary off** is entered	*19*
disp	**disp**('*text*') writes *text* on the screen. **disp**(x) displays the value of the variable x, without variable name	*8,19, 26*
echo	**echo on** lets Matlab display every executed command when a script file runs. **echo off** switches it off	*26*
eig	**eig**(A) computes the eigenvalues of A and returns a vector containing these eigenvalues	*15*
else, elseif	see **if**	
end	**end** is the command that closes off loop statements and conditional statements	
exp	**exp**(x) is the exponent of (the elements of) x with base e	*15*
eye	**eye**(n) is the nxn identity matrix. **eye**(m,n) is an mxn matrix with ones on the diagonal and zeros elsewhere	*11*
for	loop statement: **for** variable = start value : increment : end value, commands **end**	*5,18*
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	*8*
fprintf	C-command for advanced formatting of output	*19*
function	user defined function: **function** outputvar = *functionname* (inputvars) commands **end** Function files have to have the name *functionname.m*. The name *functionname* may contain up to 8 letters.	*23*
ginput(n)	gets the coordinates of n points in the graph, which are located by positioning the cursor and thereupon clicking the mouse	*21*
global	**global** x changes x into a global variable	*24*
grid	adds a grid (a lattice of horizontal and vertical lines) to a graph	*21*

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	*4,7*
hold	**hold on** keeps the last plot in the graphical screen. Therefore a graph will be plotted on top of the existing one. **Hold off** restores the default settings. In this case, when a plot command has been given, the graphical screen will be cleared before the new graph will be plotted	*21*
i	the imaginairy unit	*15*
if	conditional statement: **if** statement commands **else** commands **end** **if** statement commands **elseif** statement commands **else** commands **end**	*16*
imag	**imag**(c) returns the imaginary part of the complex number c	*10*
input	**input**('*text*') displays *text* on the screen and waits for input by the user. Can be assigned to a variable	*20*
length	**length**(v) returns the number of elements of the vector v	*13*
load	**load** *filename* loads the variables of the file *filename* into the memory. The file is of type .mat or of type .txt.	*20*
log	**log**(x) is the natural logarithm of (the elements of) x	*15*
log10	**log10**(x) is the logarithm with base 10 of (the elements of) x	*15*
lu	**lu**(A) computes the **lu** decomposition of the matrix A	*15*
max	**max**(v) returns the largest element of the vector v	*13,15*
mesh	**mesh**(x,y,Z) plots the matrix Z versus the vectors y and x, creating a mesh	*22*
min	**min**(v) returns the smallest element of the vector v	*15*
norm	**norm**(v) computes the Euclidian norm of v and **norm**(v, inf) computes the infinity norm	*13*
num2str	**num2str**(*var*) converts the number *var* to a text string	*21*
ones	**ones**(n) is an nxn matrix filled with ones, **ones**(m,n) is an mxn matrix	*11*
pause	**pause** pauses the running programme and waits until the user presses any key. **Pause**(n) pauses during n seconds	*4,7*
pi	**pi** is the machine's representation of π	*15*
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	*4,12, 21*
plot3	**plot3**(x,y,z) plots a line through the coordinates of vectors x, y, z	*22*
print	direct the graph to the printer. **Print** *filename* stores the graph in the file *filename*	*21*

quad	library routine: **quad**(*f,0,1*) computes the integral of the function *f* on [0,1].	*24*
quit	logout from Matlab	*7*
real	**real** (c) returns the real part of the complex vector c	*10*
rem	**rem**(m,n) returns the remainder after division of m by n	*15*
rotate3d	enables the user to rotate the plot by mouse	*22*
round	**round**(x) rounds the elements of x to the nearest integer	*15*
save	**save** *filename* x {y} saves the variable x {and y} into the file *filename*.mat	*19*
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.	*23f*
shg	**shg** shows the most recent graphical screen	*7,21*
sin	**sin**(x) is the sine of (the elements of) x	*5,15*
size	[m,n] = **size**(A) returns the size of the matrix A	*13,26*
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 (see section 15).	*25*
sqrt	**sqrt**(x) is the square root of (the elements of) x	*13,15*
sum	**sum**(v) is the sum of the elements of the vector of v	*15*
surf	**surf**(x,y,Z) plots the matrix Z versus the vectors y and x, creating a surface	*22*
surfc	same as **surf**, but adds a contour plot beneath the surface	*22*
tan	**tan**(x) is the tangent of (the elements of) x	*15*
text	**text**(m,n,'*text*') writes *text* at position (m,n) of the graphical screen	*21*
title	**title**('*text*') writes *text* as a title above the graph in the graphical screen	*21*
while	conditional loop statement: **while** statement commands **end**	*18*
whos	**whos** shows the name, size and type of the variables in the Matlab memory	*7,9, 26*
xlabel	**xlabel**('*text*') places *text* underneath the x-axis of the graphical screen	*21*
ylabel	**ylabel**('*text*') places *text* next to the y-axis of the graphical screen	*21*
zeros	**zeros**(n) returns an nxn matrix with zeros; **zeros**(m,n) returns an mxn matrix	*11*
zoom	**zoom** makes it possible to zoom in at a specific point in the graph by clicking the mouse at that point	*21*