1 Basics

Variable definitions are implemented using equal signs, such as

```
>> x = pi/2
x =
   1.5708e+00
>> sin(x)
ans =
     1
```

If the outputs are to be suppressed, a semicolon is placed at the end of the entry

```
>> x = pi/2;
>> sin(x)
ans =
     1
```

You can work with the command windows and direct inputs and outputs or with an *.m-file. In an *.m-file many commands can be listed, which are processed one after the other. The *.m-file is executed from the command window, and any output or error messages are also displayed. Furthermore an *.m-file can can also be defined as a routine (function), see later section 4.

2 Vectors and matrices

Use square brackets to define 1- and 2D-fields:

```
>> A = [1 2 3;4 5 6]
A =
     1 2 3
     4 5 6
>> b = [8;9]
b =
     8
     9
>> c = [pi exp(1)]
c =
    3.1416 2.7183
```

The multiplication of vectors and matrices is done by using *. If the dimensions are not correct, the system complains:

```
>> c*b
ans =
   49.5973
>> b*c
ans =
   25.1327 21.7463
   28.2743 24.4645
>> A*b
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of
columns in the first matrix matches the number of rows in the second matrix.
To perform elementwise multiplication, use '.*'.
```

To transpose a vector or a matrix use ':

```
>> A'
ans =
     1  4
     2  5
     3  6
>> A'*b
ans =
    44
    61
    78
```

3 Simple m-files

If several commands (possibly still under development) are processed one after the other, they can be combined in an *.m-file. This is executed and the commands contained in it are processed.

A simple example:

```
n = 50;
h = 1/(n+1);

A = zeros(n,n);
for i=1:n
    for j=1:n
        if i == j
            A(i,j) = -2;
        elseif abs(i-j) == 1
            A(i,j) = 1;
        end
    end
end
A = A/h^2;
b = ones(n,1);
x = A\b;

plot([0:h:1],[0; x; 0])
```

Or shortly:

```
n = 50;
h = 1/(n+1);

B = 1/h^2*(-2*diag(ones(n,1))+diag(ones(n-1,1),1)+diag(ones(n-1,1),-1));
b = ones(n,1);
x = B\b;

plot([0:h:1],[0; x; 0])
```

4 Functions

If a complex procedure is required, *.m-files are a good choice. The *.m-file contains the name of the function and the first line is
„function RETURN(N)=FUNCTIONNAME(INPUTVALUE(E))“.
Example:

```
function fac = factorial(n)
% Computes n!
% A recursive function is used
if n==1
    fac = 1;
else
    fac = n*fakultaet(n-1);
end
```

Comments are inserted with the help of % signs. The procedure is called in the command window

```
>> factorial(3)
ans =
     6
>> factorial(6)
ans =
   720
```

Another implementation:

```
function fac = factorial2(n)
% Computes n!
% Uses a for-loop
```

```
fac = 1;
for i=2:n
    fac = fac*i;
end
```

Therein if- and for-loops are used. Another option is to use while loops:

```
function fac = factorial3(n)
% Computes n!
% Uses a while-loop
fac = n;
i = n-1;
while i>1
    fac = fac*i;
    i = i-1;
end
```

5 Graphical representations

Graphical representations are implemented as polygon chains using vectors

```
>> x = linspace(0,2*pi,100);
>> plot(x,f(x))
```

First, a vector x was defined containing 100 equidistant entries between 0 and 2π. Plotted is the polygonal chain of x values and the values $f(x)$. Useful options are

```
>> plot(x,f(x),'-.')
>> plot(x,f(x),'x')
>> plot(x,f(x),'-x')
>> plot(x,f(x),'linewidth',1)
```

Two or more graphical representation are separated by commas

```
>> plot(x,f(x),x,cos(x),x,1./(x+1),'linewidth',1)
```

6 Solution of systems of linear equations

The backslash can be used to solve systems of linear equations

```
>> A = [1 2 3;4 5 6;7 8 1];
>> b = [1 3 9]';
>> x = A\b
x =
   -0.1667
    1.3333
   -0.5000
```

Linear least squares problems can also be solved using backslash:

```
>> A = [1 2;3 4;5 6]
A =
     1 2
     3 4
     5 6
>> b = [1 3 9]';
>> x = A\b
x =
    3.6667
   -1.6667
>> A*x-b
ans =
   -0.6667
    1.3333
   -0.6667
>> norm(ans,2)
ans =
    1.6330
```