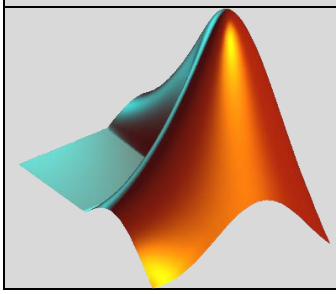
Numerical methods using Matlab



Numerical methods using Matlab

1. Introduction
2. Working modes
3. Operations and use of numbers, vectors and matrices
4. Functions
5. Arrays
6. Loops and script branching
7. 2D graphics and objects
8. 3D graphics
9. Data files
10. Symbolic mathematics toolbox
11. Numerical solution of the nonlinear equations

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

1. Introduction

MATLAB (short for MAtrix LABoratory) is the numerical analysis and calculation software of The Mathworks, Inc. Over time, it has evolved into a programming environment with which a wide range of mathematical operations, technical measurements, statistical analyzes, graphics and the like can be performed. With a number of additional libraries, called toolboxes in Matlab, its capabilities extend to virtually all possible technical applications. The user can acquire, analyze, optimize, visualize data and model events in various fields, not only in technical fields.
This part of the material does not aim to fully describe all functions of Matlab. We will only describe those functions that are important for other parts of this work. For a full description of Matlab's capabilities, I would refer the reader to the literature used.
The most important thing to successfully master Matlab is to accept the idea that everything is a matrix of complex numbers that consists of rows and columns. If a matrix has only one row, we call it a vector. A scalar is also a matrix that has exactly one row and exactly one column.

2. Working modes in Matlab

Matlab consists of a large number of functions available in libraries. This means that common functions for various tasks (eg solving a system of linear equations) can be called by a simple function. There is extensive graphics support that allows results to be visualized using just a few commands. Each Matlab numeric object is treated as a double-precision array. The orientation in the programs written in Matlab, which is called m-file, is very simple and clear. The most commonly used modes of work in Matlab are dialog, program and graphical.

Examples of the commands:

a=2:25;
5+6
*b=0:pi/180:2*pi;*
c=sin((pi/2)-0.2)
D=[1,2;3 4]
log(2)/(2^3)
E=D'
who
whos
clear
clear all
help
help elfun
clc
pi
eps
i; j;
sin(x)
abs(cos(x))
inv(D)
*h=inv(D')*3.14*
f=sqrt(tan(x))
ans

Dialog mode

- is accessible in the command window
- The commands entered here are executed immediately
- the assignment statement is executed with the = sign
- The results of the calculation are automatically saved in the ans variable, unless otherwise named
- If you want to suppress the printing of results (especially for vectors that have a large number of elements), type a semicolon after the command
- In this mode, Matlab works much like a smart calculator
- function arguments are enclosed in parentheses
- vectors and matrices can be generated, rows are separated by a semicolon, columns by a comma or space, if we work with matrix elements, we use square brackets
- for elementary operations the following characters are used:
- +, -, *, /, ^ (exponentiation)
- we use an apostrophe to transpose matrices and vectors
- used variables are kept in memory (we see them in the workspace window), they can be printed by the who command, with their properties by the whos command
- variables can be deleted either individually with clear variable_name or all with clear all

	• Matlab has a very sophisticated help system by typing help with a space and topic name

	Program mode • works with a special editor called debugger • when writing a script, it is advisable to use comments that are marked with the% character, anything that is found after the% character (percent) to the end of the line is considered a comment and is ignored by Matlab • Commands can be separated by commas or semicolons • Start the program mode as follows: • Click the icon in the menu • menu command New / Script • by double-clicking the mouse and opening an existing m-file • the file must be saved to disk and named according to the usual rules • m-file can be started from the Command window by entering the file name • it is useful to set the current path in the Path Browser row
Examples of the commands: *figure(1)* *close* *close all* *plot(x,y)* *plot(a,b,'lz',a,c,'k')* *plot(x,x.^2)* *help plot* *grid on* *grid off* *hold on* *hold off* *axis* *axis equal* *axis square* *axis tight* *help axis* *xlabel* *ylabel* *title* *legend*	**Graphic mode** • uses a separate graphical window - Figure • use figure to open a new window • The close command closes the last active graphics window • At the beginning of m-file, it is advisable to close all previously used graphical windows with the command close all • we use the plot command (a, b) where a represents the domain of the domain and b represents the domain of the value domain • The plot command can also be used in the form of a fence (a, b, 'lz') where l represents the color of the line and z represents its type • for more information about the plot command, see help plot by entering help plot • You can use the command to display the auxiliary scale • grid on resp. grid off • displaying more curves into one graph can be realized by the command hold on, resp. disable it with hold off • use the axis command ([xmin, xmax, ymin, ymax]) to change the definition and value ranges • axis labels can be completed with xlabel ('text'), ylabel ('text') • the title of the graph can be completed with the title ('text') command • the legend of the graph can be added with the command legend ('text')

MATLAB

3. Data types, variables, operations and functions of numbers, vectors and matrices

The most commonly used data types or classes in Matlab are double, char, and logic. All these types are considered as arrays in Matlab. Numeric objects belong to the double class, which represents an array with double precision. Numbers are treated as a 1x1 field. Char elements are an array of strings (a sequence of letters). Logical types contain arrays that contain either 0 (false) or 1 (true).

Another important class, unique in Matlab (when comparing other programming environments), is function_handle. It contains the information required to find and execute the function. The function_handle name consists of an @ character followed by a parenthesis with the specified input arguments and a function expression; for example @ (x) 2 * cos (2 * x + 0.5). Function_handle is used as an input argument for a function call. For example, suppose we have to draw the function 2cos (2x + 0,5) in the interval from 0 to 2π in Matlab. For example, the script might look like this:

```
F = @ (x) 2 * cos (2 * x + 0.5); % function_handle
    x = -2 * pi: 0.01: 2 * pi; % domain definition declaration
    plot (x, F (x), 'r'). % function call
```

Other types of data we can work with in Matlab are sparse (sparse matrices), inline (objects), and struct (structured array). The user can also define his own data types. The object data type can be printed using the class command.

The name of the variable in Matlab must start with a letter (Matlab is case sensitive) and we can use both letters and numbers. The length of the name is unlimited, but Matlab only considers the first 63 characters. Local variables that Matlab recognizes automatically are not accessible in other parts of the program. Global variables must be declared by the user using the global command and can be shared with all parts of the program.

Matlab operates with several implemented constants and variables. The most important are: ans (automatic result name), eps (smallest number for: 1 + eps> 1), inf (infinity), NaN (undefined value), i or j (√ (-1)), pi (π), realmin (the lowest applicable positive number) and realmax (the largest applicable positive number).

To create an array, enter its elements in square brackets. Separate elements with spaces and separate lines with a semicolon or the Enter key. Matlab distinguishes row and column vector. For example, b = [1 2 3] is a line vector, b = [1; 2; 3;] is a column vector and b = [1 2 3] 'is a transposed row vector. An apostrophe is an operator for transposing variables.

Field elements are accessed using parentheses. For example, we define the matrix A = [8 1 6; 3 5 7; 4 9 2], then A (2,3) is 7, A (:, 2) is the second column, A (2: 3,2: 3) is a 2x2 submatrix [5 7; 9 2] and A (7) is 6 (counts down columns).

An array of cells is a sequence of arbitrary objects. It can be created by enumerating objects in curly brackets. For example, using the command C = {[1 2 3], 'text', 2-3i}. List it with the celldisp (C) command. For example, we can access each item as follows: C {1} or C {1} (2).

A string is a sequence of characters created by writing them between apostrophes. We can also work with them according to the following examples.

```
r1 = 'good'; r2 = 'day';
r3 = strcat (r1, r2) (result: r3 = good day)
r4 = r3 (7: 9) (result: r4 = day)
```

Examples of the commands:	**Operators**
A=[1 2 3;4 3 2]; *B=[4 5 6;2 1 0];* *A+B* *ans =* *5 7 9* *6 4 2* *A*B'* *ans =* *32 4* *43 11* *A.*B* *ans =* *4 10 18* *8 3 0* *A>B* *ans =* *0 0 0* *1 1 1* *(A>B)|(B>7)* *ans =* *0 0 0* *1 1 1*	Operators are divided into arithmetic, relational and logical. • Arithmetic ○ Matrix +(addition),-(subtraction),*(multiplication),/ division from right), \ (division from left), ^ (exponentiation), ‘(transposition) ○ Vector - done by element . * (multiplication),. / (division),. ^ (exponentiation) • Relational - the result is either 1 (true) or 0 (false) <(less),> (greater), <= (less than or equal to),> = (greater than or equal to), == (is equal to), ~ = (not equal to) • Logical & (and at the same time), / (or), ~ (negation)

MATLAB	4. Functions
Examples of the commands: *z1=exp(-2*t)* *z2=log(t)* *z3=log10(t)* *z4=cosh(t)* *z5=acos(t)* *z6=acoth(t)* *det(A)* *inv(A)* *sum(A)* *min(min(A))* *x=inv(A)*b* *real(b)* *round(c)* *size(A)* *length(B)*	The basic groups of functions in Matlab are scalar, vector, matrix and user. • **Scalar** these functions apply to every element of the matrix, this includes common functions available from help elfun, eg. sin(x) • **Vector** these predominantly statistical functions are applied to each column of the matrix, such as the sum (A) function • **Matrix** these functions apply to the entire matrix, including common functions available from help matfun, elmat, eg inv (A) • **User created** viz. the following table row

Examples of the commands: *Function* *F=soucet(a,b,N)* *h=(b-a)/N;* *f=cos(a:h:b);* *F=sum(f(1:N))*h;* *a=0; b=10;N=100;* *F=soucet(a,b,N)*	**User-created functions** A function is actually a named sequence of commands. Its syntax is: function [output_variable] = name (input_variable) • **Name** unique name, under this name we have to save the function as a file to the current directory with the extension * .m, then the function is accessible also in other parts of the program • **input_ variables** a comma-separated list of input parameters in parentheses • **output_ variables** list of output parameters in square brackets separated by commas The body itself may begin with a comment. The comment starts with the% character and ends with the end of the line, it can be called with the help function_name. • **Inline function** another way to declare a function in Matlab is to use an inline object from a string that expresses the function. The command might look like this: function_name = inline ('expression', 'argument1', 'argument2',…)

MATLAB	5. Array, matrices
Examples of the commands: *P=[0 1.5 2.5 4];* *Q=0:0.01:1;* *R=linspace(0,1,9);* *S=logspace(0,2,10);* *T=zeros(m,n);* *U=ones(m,n);* *V=eye(m);* *W=rand(2,3);*	For basic work with the field we use the following commands: • field entered by enumerating all elements in square brackets eg. P = [0 1.5 2.5 4] • Equidistant array of elements using a colon Q = first_ element: increment: last_ element eg. Q = 0: 0.01: 1; • an equidistant array of n elements using the linspace function R = linspace (first_ element, last_ element, n) eg. R = linspace (0,1,9) • logarithmic array of n elements using the logspace function S = logspace (first_exponent, last_exponent, n) eg. S = logspace (0,2,10) • zero array of m rows and n columns eg. T = zeros (m, n) • array of all ones m rows and n columns eg. U = ones (m, n) • a unit matrix of m rows eg. V = eye (m) • matrix of m rows and n columns of random numbers from 0 to 1 eg. W = rand (2,3)

Examples of the commands: *n=length(v);* *[m,n]=size(M);* *M=reshape(v,m,n);* *sk=dot(u,v);* *s=sum(v);* *ss=prod(v);* *sm=sum(M);* *ssm=prod(M);* *vs=cross(u,v);*	**Examples of matrix functions** Matlab contains a large number of functions for working with matrices. • the number of elements n of the vector v can be determined by the length function eg. *n = length (v);* • the order of the matrix M [m, n] can be determined by the size function eg. *[m, n] = size (M);* • sorting the elements of vector v into a matrix M of the order [m, n] by size *eg. M = reshape (v, m, n);* • the scalar product sk of two vectors u, v is calculated by the function dot eg. *sk = dot (u, v);* • the sum s and the product ss of the vector elements v can calculated using the sum and / or sum functions resp. prod eg. *s = sum (v); ss = prod (v);* • the sum of sm and the product of ssm of the elements M of the matrix M can be calculated by the function sum or prod • the result is a column sum vector resp. products eg. *sm = sum (M); ssm = prod (M);* • vector product vs vectors u, v is calculated by the cross function eg. *vs = cross (u, v);*

MATLAB	6. Branching, decision making and loops
Examples of the commands: *if A>10* *B=A+1;* *else* *B=A+2;* *end*	**Branching and decision making in script** If we do not use branching in the script, each command is executed exactly once in the given order, if so, each command inside the branching is executed at most once. The branch always starts with an *if* statement, followed by a logical condition and ends with an end statement. The part of the script between *if* and *end* statements is called the body, which can also contain *else* statements. The script body is called positive if the else statement is missing between *if* and *end* statements, and negative if the statements are between the *else* and *end* statements. The return command can jump out of the branch prematurely. • The complete condition syntax looks like this: *if* condition % positive part commands *else* % negative part commands *end* • Incomplete condition syntax looks like this: *if* condition % positive part commands *end*

Examples of the commands: *while a~=b* *if a >b* *a= a+b;* *else* *b=a-b;* *end* *end*	Loop while The *while* loop is a loop with a condition at the beginning. If the condition is met, the commands are executed inside the cycle. The cycle is started by the *while* statement followed by a logical condition. This is followed by the cycle body and ends with the end statement. With the return statement we can jump out of the cycle prematurely. • The syntax of the while loop looks like this: *while* logical condition % body cycle *end*
Examples of the commands: *for n=1:10* *a= a+5;* *end*	**Loop for** The *for* loop executes the statements for all elements of the line vector v, that is, for values from 1 to *length (v).* Unfortunately in Matlab it is not possible to index from 0, we always have to start with 1. The cycle start is realized by the for statement with the name of the vector type variable, followed by the cycle body whose commands are executed for all vector elements and finally the end statement. With the return statement we can jump out of the cycle prematurely. • The for loop syntax looks like this: *for* variable = 1: number of loop repetitions % body cycle *end*

MATLAB	7. 2D graphic and objects
Examples of the commands: *figure;* set (gcf,'Units', 'Normal','Position', [0.1,0.2,0.3,0.4]); *s = axes('Position',* *[*0.1,0.2,0.3,0.4*])* set (s,'Color', [1 0 1]) *subplot (4,2,2);* *fplot(@fce,[1 2])*	The general command in Matlab to work with a given object is *name = command (property, value)*. The object property can be set using the *set* command *(object, property, value)* and multiple properties can be set at the same time. The command to open the graphics window is *figure*. The units in use are in *Normal* mode, which is the interval 〈0; 1〉 x 〈0; 1〉, where position 〈0; 0〉 is the lower left corner of the screen and 〈1; 1〉 its upper right corner. To set the position and size of the current graphics window, use the following command: *set (gcf, 'Units', 'Normal', 'Position', [x1, y1, x2, y2])*, where x1, y1 is the lower left corner of the graphics window from the lower left corner of the screen and x2, y2 width and height graphical window in the same units. You can place a coordinate system in the graphics window by using the *coordinate = axes('Position', [x1, y1, x2, y2])* command. Existing object properties with their current values are displayed by Matlab after the *get (object)* command. We set the colors as a combination of RGB in the range 〈0; 1〉. The most commonly used commands for 2D graphics have been listed in section 1.2. Modes of work in Matlab in graphics mode. Nevertheless, we will add some, especially the *subplot (parameter1, parameter2, parameter3)* command, which can divide the graphical

	window into the number of parts given by the number of parameter3, into rows whose number is given by parameter1 and into columns whose number is given by parameter2.

The following two scripts with explanatory commentary and the resulting graphs are shown as examples for working with 2D graphics, see Fig.1 and Fig.2.*Příklad 1:*

```
clc;close;clear;                          % clear Command Window, all graphical windows
and all variables
x=-4*pi:0.001:4*pi;                       % the domain of the functions
y1=sin(x)./(x);                           % the value range of the first function
y2=sin(2*x)./(2*x);                       % the value range of the second function
y3=sin(3*x)./(3*x);                       % the value range of the third function
y4=sin(4*x)./(4*x);                       % the value range of the fourth function

subplot(2,2,1)                            % figure 1
grid on; hold on;                         % turn on the grid and draw
   plot(x,y1)                             % the graph of the first function
   xlabel('x');ylabel('y')                % captions of the axes
   title('y=sin(x)/x')                    % title of the graph 1
   osy=[-4*pi,4*pi,-0.3,1.1];             % axes limit
   axis(osy)
   plot([-4*pi,4*pi],[0,0],'r')           % axis x as abscissa
   plot([0,0],[-0.5,1.1],'r')             % axis y as abscissa
```

```
subplot(2,2,2)
grid on;hold on;
   plot(x,y2)
   xlabel('x');ylabel('y')
   title('y=sin(2x)/2x')
   osy=[-4*pi,4*pi,-0.3,1.1];
   axis(osy)
   plot([-4*pi,4*pi],[0,0],'r')
   plot([0,0],[-0.5,1.1],'r')
```

```
subplot(2,2,3)
grid on;hold on;
   plot(x,y3)
   xlabel('x');ylabel('y')
   title('y=sin(3x)/3x')
   osy=[-4*pi,4*pi,-0.3,1.1];
   axis(osy)
   plot([-4*pi,4*pi],[0,0],'r')
   plot([0,0],[-0.5,1.1],'r')
```

```
subplot(2,2,4)
grid on;hold on;
   plot(x,y4)
   xlabel('x');ylabel('y')
   title('y=sin(4x)/4x')
   osy=[-4*pi,4*pi,-0.3,1.1];
   axis(osy)
   plot([-4*pi,4*pi],[0,0],'r')
   plot([0,0],[-0.5,1.1],'r')
```

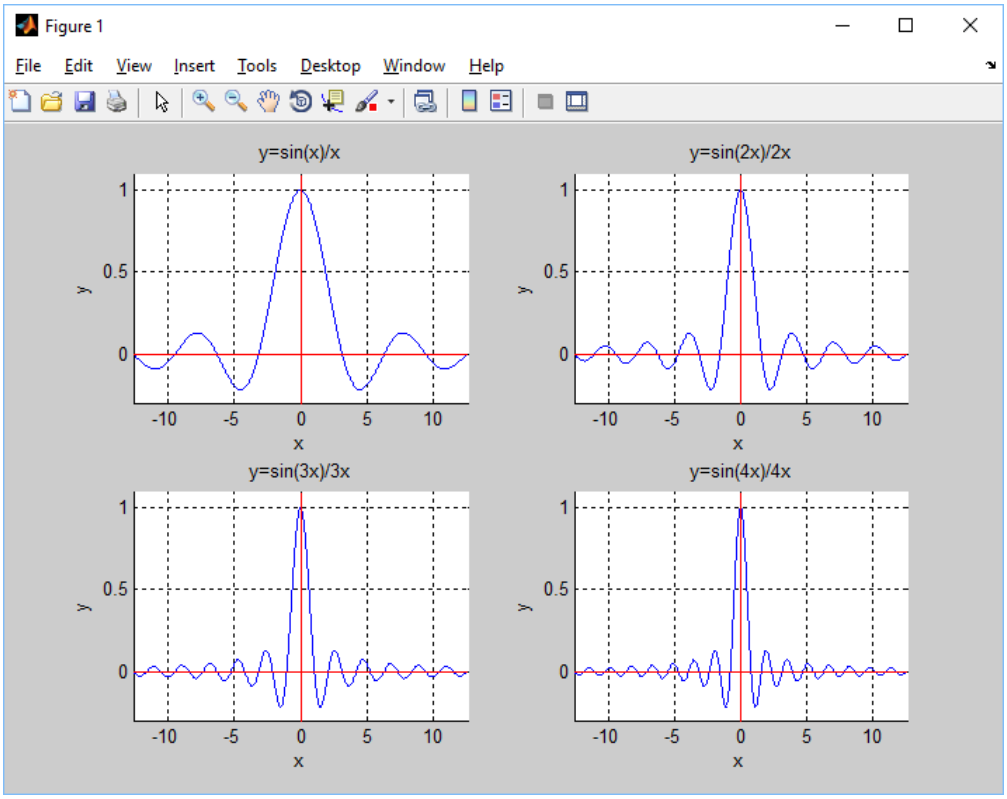


Fig. 1

Exampled 2:

```
function y=fce(x)                    % saved function
y=(x).*cos(x.^2)+1./(x.^2);

fplot(@fce,[pi 10*pi])               %graph of the saved function on the domain
fromd π to 10 π
grid on
xlabel('x');ylabel('y')
title('y=x*cos(x^2)+1/(x^2)')
```

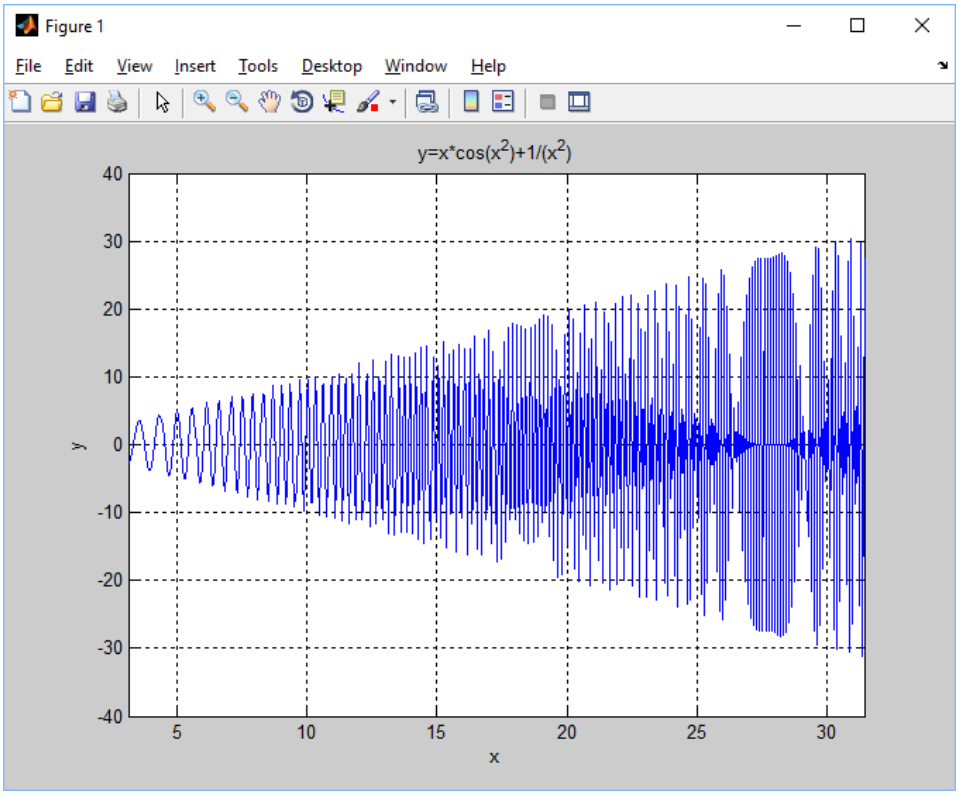


Fig. 2

MATLAB	8. 3D graphic
Examples of the commands: *[x,y]=meshgrid (-4:0.1:4);* *surf(x,y,z);* *mesh(x,y,z);* *contour(x,y,z)* *colormap(M)* *colorbar*	The 3D graphics mode is used to display the functions of two variables *z = f (x, z)*. The definition range of each point is generated as a grid using the *meshgrid* command. Several commands can be used to display the function - for example *plot3, mesh, meshc, surf, surfc, surfl, surface*. It is the *surface* command that creates a surface as an object, which we can then work on, as with other objects, such as changing its properties. You can use the *contour* command to create contour lines of a given surface. We can also modify colors, use the RGB model in the range rozsahu 0; 1〉 , using the colormap command, and display the scale using the colorbar command.

As an example for working with 3D graphics, we will mention the following script with an explanatory comment and the resulting graph, see Figure 3.

```
[x,y]=meshgrid(-4:0.1:4);                    % the definition of the grid for the domain
z=x.^2.*exp(-x.^2-y.^2);                     % the value range of the function
surf(x,y,z);                                 % the graph of the function
title('f(x,y)=x^2*exp(-x^2-y^2)');xlabel('x');ylabel('y');   % title, label of the axes
M=[0 1 0;0 0 1;1 0 0];
colormap(M)                                  % the definition of the colors
colorbar                                     % the scale in defined colors
```

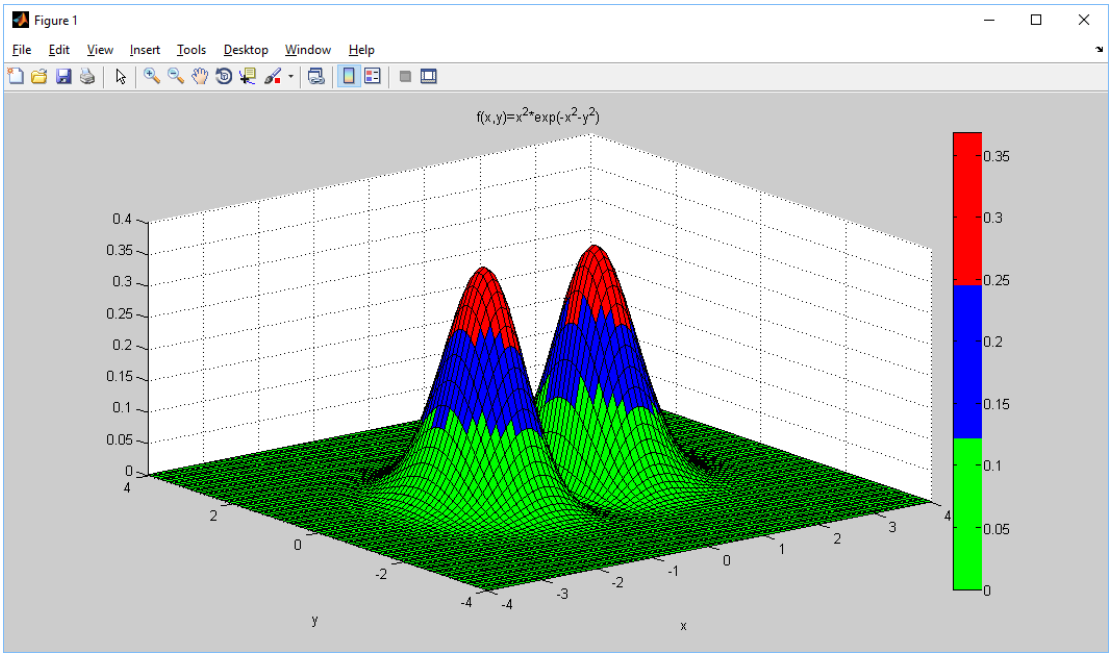


Fig. 3

MATLAB	9. The data files
Examples of the commands: *figure;* *A=[0 1 2]; b=6;* *save data1;* *save data2 A;* *clear all;* *load data1;* *who*	In Matlab we can work with data from other programs. We always have a choice of several options, the best method of import and export always depends on the volume of processed data, their format, etc. The most commonly used activities in this area are saving to a file, loading from a file and importing data from Exel. • **Save to file** Use the save command to save the data to a file. A possible command syntax is: *save filename* - saves the entire workspace to a file named *filename.mat* *save file_name prom1 prom2… promn* - the variable named *prom1, prom2,…, promn* is saved in the file named *filename.mat*

	• **Read from file** To load variables from files with the * .mat extension, use the *load* command • **Import data from Excel** The imported data file must be in the current directory. Open the file, select the appropriate area, set the variable names and use the import button to import the data. Matlab can also generate a script to import the relevant data.

MATLAB	10. Toolbox of the symbolic mathematic
Examples of the commands: *help symbolic* *syms a b t x y;* *V=sin(x)^2+cos(x)^2* *simple(V)* *pretty(V)* *limit(sin(x)/x)* *limit(1/x,x,0,'left')* *limit(1/x,x,0,'right')* *limit(1/x+5,x,inf)* *ezplot(y,[-2 2])* *d=(a+b)^2;* *subs(d,{a,b},{1 2})* *symsum* *(1/n)^2,1,inf)* *DE='t^2*Dy+t*y=1';* *y=dsolve* *(DE,'y(1)=2')* *ezplot(y,0.01,4)* *f1=double* *(subs(y,t,2.5))*	The symbolic mathematics toolbox serves as a tool for general calculations such as derivatives, primitive functions, solving differential equations, etc. Detailed description of all options and demo programs is described in help, after entering the *help symbolic* command. If we want to work with the symbolic mathematics toolbox, we must always introduce symbolic variables, either by the *sym ('prom')* or *syms prom1 prom2…* command. Expressions can be simplified with the *simple (prom)* command, or modified with *pretty(prom)*. The result of the symbolic calculation can be converted to variables in the Matlab computational environment using the *double(symbolic_variable)* command. For symbolic calculation of limits we use the limit command in several alternatives. The *limit(expression)* command calculates the limit when the symbolic variable approaches to zero, the *limit(expression, x, a)* calculates the limit when the symbolic variable approaches a, and the *limit(expression, x, a, 'left')* limit, when the symbolic variable approaches the value and from the left and the *limit(expression, x, a, 'right')* we calculate the limit, when the symbolic variable approaches the value and from the right. The graphs of the functions are displayed by the command *ezplot(prom, interval)*. The substitution into the expression after the given variable is realized by the *subs(expression, variable, value)* command. Using the *symsum* command, we can add infinite convergent series. Furthermore, we can simply differentiate functions with the *diff(f1)* command, integrate functions with the *int(f2)* command, and solve differential equations with the *dsolve('DifR', 'variable')* command. For the notation of the differential expressions $y' \left(\frac{dy}{dx}\right)$ and $y'' \left(\frac{d^2y}{dx^2}\right)$ we use the designation *Dy* and *D2y*. To symbolically solve the differential equation, we use the *dsolve* command, which has the parameters given by the differential equation and initial conditions. To calculate the value of the resulting function for a given parameter, we use the *double* command.

The symbolic solution of differential equations in Matlab is a very important topic for this work. Therefore, we will show here some typical examples from various technical fields. Each example is devoted to one basic type of ordinary first-order and second-order differential equations.

Example 1.

Symbolically solve the initial problem for the first order differential equation: $t^2y' + ty = 1$, with the initial condition $y(1) = 2$. Plot the graph of the particular solution in the interval from 0.01 to 4. Determine y(2.5).

Script:

```
syms y t;                          % introduction of the symbolic variables y, t
DE='t^2*Dy+t*y=1';                 % determination of the differential equation
y=dsolve(DE,'y(1)=2')              % the solution of the initial problem for the differential equation DE
ezplot(y,0.01,4)                   % the plot of the particular solution
hold on; grid on;                  % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,2.5))           % calculating the value of the particular solution for t=2,5
plot(2.5,f1,'*r','MarkerSize',10)  % plotting the value of the particular solution for t=2,5 to the graph
```

The Matlab result:

```
y =
log(t)/t + 2/t
f1 =
   1.1665
```

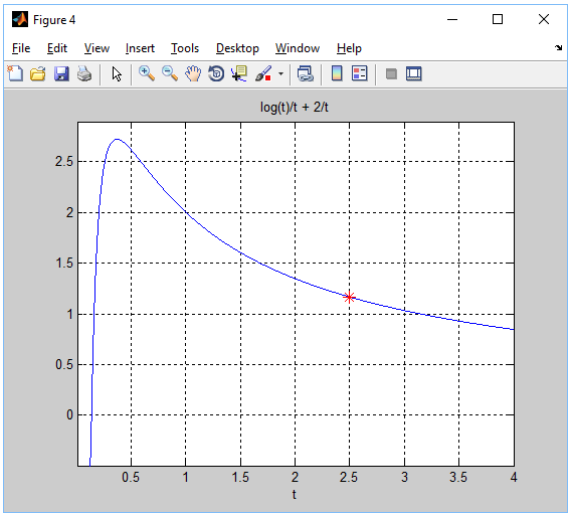


Fig. 4
Particular solution from example 1

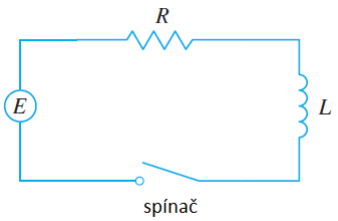
Example 2.

Consider an electric circuit with a switch, 12 Ω resistance and an inductance of 4 H. The battery gives a constant voltage of 60 V. We turn the switch on at 0 s. The current at 0 s is 0 A.

a) find a function that expresses the current versus time I (t),

b) current value at 1 s,

c) from the graph, estimate the steady-state current limit value (for going to infinity).



We describe this situation using the first order ordinary differential equation: $LI' + RI = R(t)$. after substitution and modification we get the initial problem for the differential equation $y' + 3y = 15, y(0) = 0$, which we can solve by the following script:

Script:

```
syms y t;                                  % introduction of the symbolic variables y, t
DE='Dy+3*y=15';                            % determination of the differential equation
y=dsolve(DE,'y(0)=0')                      % the solution of the initial problem for the differential
equation DE
ezplot(y,0,3)                              % the plot of the particular solution
hold on; grid on;                          % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,1))                     % calculating the value of the particular solution for t=2,5
plot(1,f1,'*r','MarkerSize',10)            % plotting the value of the particular solution for t=2,5 to
the graph
```

The Matlab result:

```
y =
 5 - 5*exp(-3*t)

f1 =
   4.7511
```

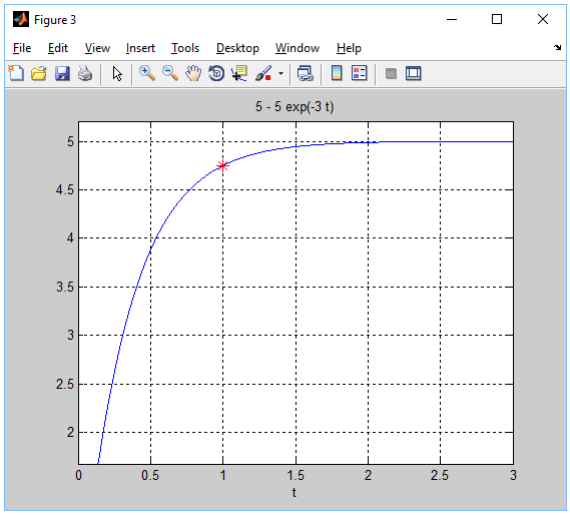


Fig. 5
The particular solutions of the ex. 2, the steady state current will be 5A

Example 3.

Consider the electric circuit of Example 2. However, we change the current source to a generator that generates current according to function E (t) = 60sin (30t). The current at 0 s is 0 A.

a) find the dependence of current on time I (t),

b) current at 2 s,

c) draw the current waveform in the interval from 0 to 3 seconds.

We adjust the initial problem for the differential equation from example 2 to $4y' + 12y = 15\sin(30t),\ y(0) = 0$, which we can solve with the following script:

Script:

```
syms y t;                          % the introduction of the symbolic variables y, t
DE='4*Dy+12*y=60*sin(30*t)';       % the determination of the differential equation
y=dsolve(DE,'y(0)=0')              % the solution of the initial problem for the differential equation DE
ezplot(y,0,3)                      % the plot of the particular solution
hold on; grid on;                  % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,2))             % calculating the value of the particular solution for t=2
plot(2,f1,'*r','MarkerSize',10)    % plotting the value of the particular solution for t=2 to the graph
```

The Matlab result:

```
y =
(50*exp(-3*t))/101 -
(50*cos(30*t))/101+ (5*sin(30*t))/101

f1 =
   0.4576
```

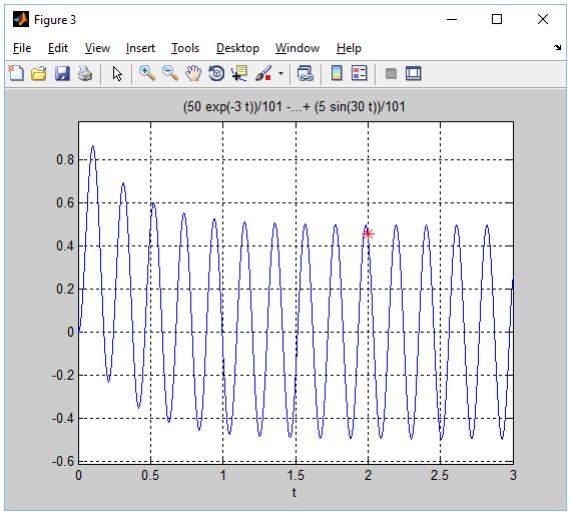
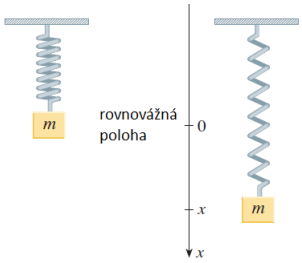


Fig. 6
The particular solution of the example 3

Example 4.

A 2 kg weight is suspended on the spring. The length of the spring at rest is 0.5 m. To stretch the spring to a length of 0.7 m, a force of 25.6 N is required. Find the position of the weight at time t=4 s.



We describe this situation using second order ordinary differential equation in the form $mx'' + cx' + kx = F(t)$. After substitution and adjustment we get the initial problem for the differential equation $y'' + 64y = 0, y(0) = 0{,}2,\ y'(0) = 0$, which we will solve using the following script:

```
syms y t;                                % the introduction of the symbolic variables y, t
DE='D2y+64*y=0';                         % the determination of the differential equation
y=dsolve(DE,'y(0)=0.2', 'Dy(0)=0')       % the solution of the initial problem for the differential equation DE
ezplot(y,0,5)                            % the plot of the particular solution
hold on; grid on;                        % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,4))                   % calculating the value of the particular solution for t=4
plot(1,f1,'*r','MarkerSize',10)          % plotting the value of the particular solution for t=4 to the graph
```

The Matlab result:

```
y =
cos(8*t)/5

f1 =
   0.1668
```

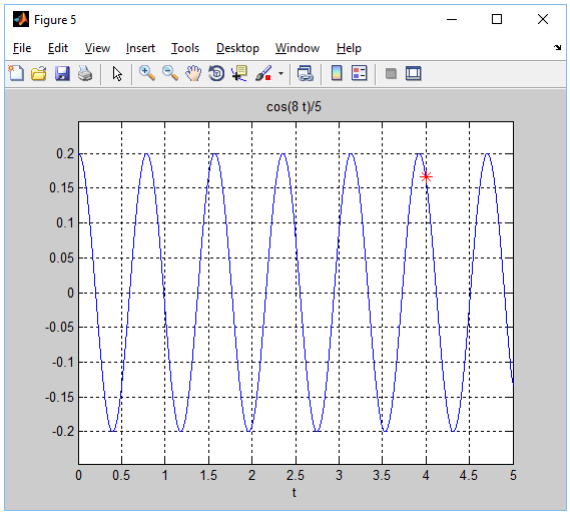


Fig. 7
The particular solution of the ex. 4

Example 5.

We give the spring of example 4 in a liquid with a damping constant c = 40. Find the position of the weight at time t if it starts with an initial speed of 0.6 m / s. Draw the graph of the displacement versus time in the interval from 0 to 2 s. Find the position of the weight at time t = 0.5 s.

We describe this situation using second order ordinary differential equation in the form $mx'' + cx' + kx = F(t)$. After substitution and adjustment we get the initial problem for the differential equation $y'' + 20y' + 64y = 0$, $y(0) = 0$, $y'(0) = 0{,}6$, which we will solve using the following script:

Script:

```
syms y t;                                % the introduction of the symbolic variables y, t
DE='D2y+20*Dy+64*y=0';                   % the determination of the differential equation

y=dsolve(DE,'y(0)=0', 'Dy(0)=0.6')       % the solution of the initial problem for the differential equation DE
ezplot(y,0,5)                            % the plot of the particular solution
hold on; grid on;                        % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,0,5))                 % calculating the value of the particular solution for t=0,5
plot(0,5,f1,'*r','MarkerSize',10)        % plotting the value of the particular solution for t=0,5 to the graph
```

The Matlab result:

```
y =
exp(-4*t)/20 - exp(-16*t)/20
f1 =
   0.0067
```

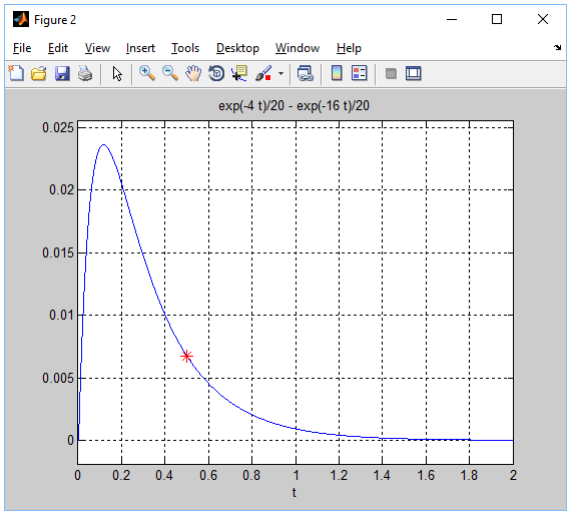


Fig. 8
The particular solution of ex.5

Example 6.

The spring of example 5 will be excited by force F (t) = 1000cos (2πt). Find the position of the weight at time t if it starts at an initial speed of 0.6 m / s. Draw a graph of displacement versus time between 0 and 5 s. Find the weight position at t = 2.5 s.

We describe this situation using second order ordinary differential equation in the form $mx'' + cx' + kx = F(t)$. After substitution and adjustment we get the initial problem for the differential equation $y'' + 20y' + 64y = 1000\cos(2\pi t), y(0) = 0,\ y'(0) = 0{,}6$, which we will solve using the following script:

Script:

```
syms y t;                                  % the introduction of the symbolic variables y, t
DE='D2y+20*Dy+64*y=1000*cos(2*pi*t)';      % the determination of the differential equation
y=dsolve(DE,'y(0)=0', 'Dy(0)=0.6')         % the solution of the initial problem for the differential equation DE
ezplot(y,0,5)                              % the plot of the particular solution
hold on; grid on;                          % add-drawing to the graph, turn on of the grid
f1=double(subs(y,t,2,5))                   % calculating the value of the particular solution for t=2,5
plot(2,5,f1,'*r','MarkerSize',10)          % plotting the value of the particular solution for t=2,5 to the graph
```

The Matlabu result:

```
y =
exp(-4*t)*(16000/(9*(4*pi^2 + 256)) -
4000/(9*(4*pi^2 + 16)) -
(250*pi^2)/(9*(4*pi^2 + 16)) +
(250*pi^2)/(9*(4*pi^2 + 256)) + 1/20) -
exp(-16*t)*(4000/(9*(4*pi^2 + 256)) -
1000/(9*(4*pi^2 + 16)) -
(250*pi^2)/(9*(4*pi^2 + 16)) +
(250*pi^2)/(9*(4*pi^2 + 256)) + 1/20) +
(250*(4*cos(2*pi*t) +
2*pi*sin(2*pi*t)))/(3*(4*pi^2 + 16)) -
(250*(16*cos(2*pi*t) +
2*pi*sin(2*pi*t)))/(3*(4*pi^2 + 256))

f1 =
   -1.4962
```

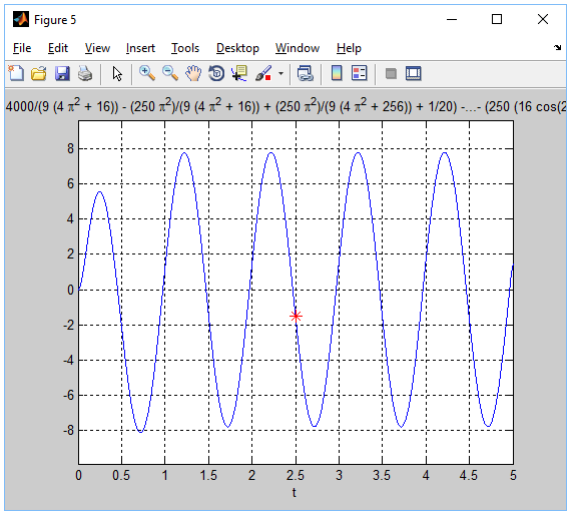
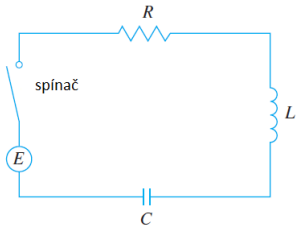


Fig. 9
The particular solution of example 6

Example 7.

Find the charge and current at time t in an electric circuit if R = 40 Ω, L = 1 H, C = 0.0016 F, E (t) = 100cos (10t) and the initial charge and current are both 0. Find the numerical value of the charge i The current and charge waveforms should be plotted in the interval from 0 to 3 s.



We describe this situation using second order ordinary differential equation in the form $LQ'' + RQ' + \left(\frac{1}{C}\right)Q = E(t)$. After substitution and adjustment we get the initial problem for the differential equation $y'' + 40y' + 625y = 100\cos(10t), y(0) = 0,\ y'(0) = 0$, which we will solve using the following script:

```
syms y t                                % the introduction of the symbolic variables y, t
DE='D2y+40*Dy+625*y=100*cos(10*t)';      % the determination of the differential equation
y=dsolve(DE,'y(0)=0','Dy(0)=0')         % the solution of the initial problem for the differential equation DE
I=diff(y)                               % derivation of the  y (derivation of the charge is the current)
f1=double(subs(y,t,2.5))                % charge in the time 2,5 s
f2=double(subs(I,t,2.5))                % current in the time 2,5 s
subplot(2,1,1)                          % the plot of the particular solution
   ezplot(y,[0,3])                      % the plot of the particular solution of the charge
   hold on; grid on;                    % add-drawing to the graph, turn on of the grid
   plot(2.5,f1,'*r','MarkerSize',10)    % plotting the value of the charge for t=2,5 to the graph
```

```
subplot(2,1,2)                    % the plot of the particular solution
  ezplot(I,[0,3])                 % the plot of the particular solution of the current
  hold on; grid on;               % add-drawing to the graph, turn on of the grid
  plot(2.5,f2,'*r','MarkerSize',10)   % plotting the value of the current for t=2,5 to the graph
```

The Matlab result:

```
y =
cos(15*t)*((2*cos(5*t))/51 + (10*cos(25*t))/123 - (8*sin(5*t))/51 - (8*sin(25*t))/123) +
sin(15*t)*((8*cos(5*t))/51 + (8*cos(25*t))/123 + (2*sin(5*t))/51 + (10*sin(25*t))/123) - (84*cos(15*t)*exp(-
20*t))/697 - (464*sin(15*t)*exp(-20*t))/2091


I =
 15*cos(15*t)*((8*cos(5*t))/51 + (8*cos(25*t))/123 + (2*sin(5*t))/51 + (10*sin(25*t))/123) -
cos(15*t)*((40*cos(5*t))/51 + (200*cos(25*t))/123 + (10*sin(5*t))/51 + (250*sin(25*t))/123) -
15*sin(15*t)*((2*cos(5*t))/51 + (10*cos(25*t))/123 - (8*sin(5*t))/51 - (8*sin(25*t))/123) +
sin(15*t)*((10*cos(5*t))/51 + (250*cos(25*t))/123 - (40*sin(5*t))/51 - (200*sin(25*t))/123) -
(640*cos(15*t)*exp(-20*t))/697 + (13060*sin(15*t)*exp(-20*t))/2091


f1 =
   0.1073
f2 =
   1.0696
```

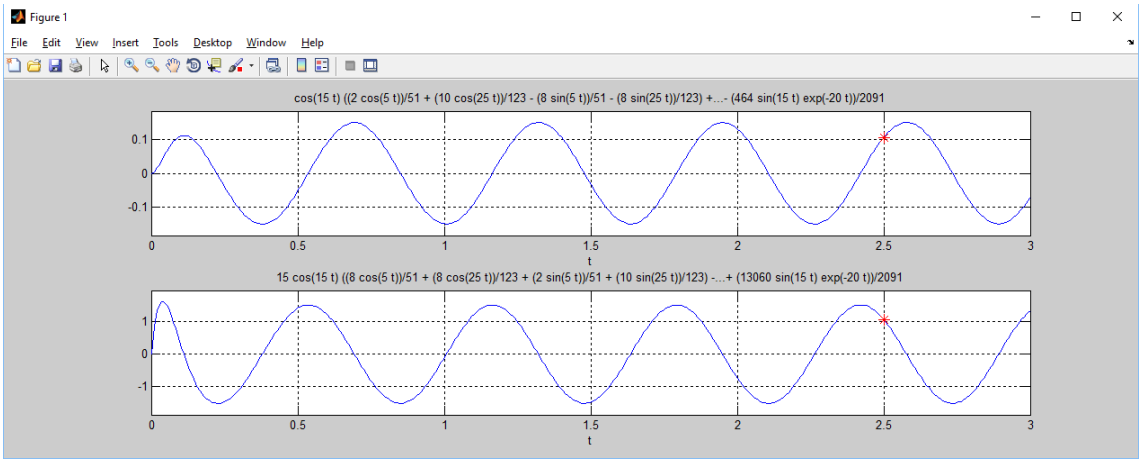


Fig. 10
The particular solution of the ex. 7

MATLAB	11. The solving of the nonlinear equations in Matlab
Examples of the commands: *syms a b c x* *rce='x^2-2*x-4';* *k=solve(rce);*	In this chapter, we present selected methods for solving nonlinear equations in Matlab. Equations can be solved in Matlab using implemented functions. In the symbolic toolbox we use the *solve* command. This function looks for solutions of equations and systems of equations. If it is unable to find a symbolic solution, it returns a numeric solution. The *solve(equation)* command prints a solution to an equation

kd=double(k) *kvrce=* *'a*x^2+b*x+c=0'* *solve(kvrce,x)* *kp=roots([1 -2 -4])*	that is specified by either a string or a symbolic expression that automatically equals zero. If there are more symbolic variables in a symbolic expression, we must specify for which of them we are looking for a solution using the *solve (equation, x)* command. Among other possibilities how to solve equations in Matlab, let's mention for example the function *roots*, which finds all roots of the polynomial, which is given by the vector of coefficients of individual powers unknown in descending order. For example, the vector *v = [3 -4 0 2]* gives a polynomial $3x^3 - 4x^2 + 2$. We have to create our own functions for numerical methods of solving equations. We divide these methods into closed ones, where we have to locate the roots precisely (root separation) and opened, where only a good initial root estimate is sufficient. The group of closed methods includes, for example, the bisection method, the falsi regulation method, etc. The opened method group includes the Newton method, the simple iteration method, the mowing method and others. These methods are very important in technical practice, so here are some examples of scripts with explanatory comments.

Example 1.

Using the bisection method find the root of the equation $10\cos(x-1) - x^2 + 2x - 1 = 0$, which is in the interval $\langle 2{,}3;\, 2{,}4\rangle$ with accurancy $0{,}01\%$.

Script of the function:

```
function [fv,k,re,i]=bisection(fce,xd,xh,me,mi)
% bisection: Method for finding the root using the two initial conditions; closed method
% Input:
% fce: Function its root we are looking for
% xd: initial down approximation
% xh: initial up approximation
% me: maximal relative mistake (preset = 0.0001%)
% mi: maximal number of the iteration (preset = 100)
% Output:
% fv: the root value of the function
% k: root
% re: the relative mistake of the approximation (%)
% i: the number of the iterations

   if nargin<3,error('minimum 3 input arguments'),end
   if nargin<4, maxm_err=0.005;end
   if nargin<5, max_iter=100;end
   if fce(xd)*fce(xh)>0,error('no sign change'),end
% first step
   i = 0;
   while(1)
      xn =(xd + xh)/2;
      re=abs((xn-xh)/xn);
```

```
    if((re<me) || mi<i),break;end
    i = i + 1;
    test = fce(xd)*fce(xn);
    if test<0
       xh = xn;
    elseif test>0
       xd = xn;
    else
       re = 0;
    end
  end
    k=xn;
    fv =fce(k);
end
```

Script of the solution:

```
clc; close; clear;                                  %   clear of the Command window, graph windows
and variables
fce=@(x) 10*cos(x-1)-x^2+2*x-1;                     %   defined function
format long                                         %   results in 15 decimal digits
[fv,k,re,i]=bisection(fce,2.3,2.4,0.0001,100)       %   calling of the function with real parameters
```

The Matlab result:

```
fv =  -0.001604652167318          %   the value of the function in the approximation
k =   2.379492187500000           %   approximation of the root with given accuracy
re =     8.208158909949030e-05    %   relative mistake of the approximation
i =    8                          %   the number of the iterations to achive thegiven accurancy
```

Example 2.

Using the Regula-Falsi method find the root of the equation $10\cos(x-1)-x^2+2x-1=0$, in the interval $\langle 2{,}3;\ 2{,}4\rangle$ with accuracy $0{,}01\%$.

Script of the function:

```
function [fv,k,re,i]=regulafalsi(fce,xd,xh,me,mi)
% regulafalsi: Method for finding the root using the two initial conditions; closed method
% Input
% fce: Function its root we are looking for
% xd: initial down approximation
% xh: initial up approximation
% me: maximal relative mistake (preset = 0.0001%)
% mi: maximal number of the iteration (preset = 100)
% output:
% fv: the root value of the function
```

```
% k: root
% re: the relative mistake of the approximation (%)
% i: the number of the iterations

   if nargin<3,error('minimum 3 input parameters'),end
   if nargin<4, me=0.005;mi=10000;end
   if nargin<5, mi=10000;end
   if fce(xd)*fce(xh)>0,error(without the sign change '),end
   % first step
   i = 0;
   while(1)
      t1=fce(xd);
      t2=fce(xh);
      if t1==t2,error('choose the other interval, in this is the method divergent');end
      xn =(xd*t2-xh*t1)/(t2-t1);
      re=abs((xn-xh)/xn);
      if((re<me) || mi<i),break;end
      i = i + 1;
      test=t1*t2;
      if test<0,
         xh = xn;
      elseif test>0
         xd = xn;
      else
         re = 0;
      end
   end
      k=xn;
      fv =fce(k);
end
```

Script of solving of the ex. 2:

```
clc; close; clear;                          % clear of the Command window, graph windows and variables
fce=@(x) 10*cos(x-1)-x^2+2*x-1;             % defined function
format long                                 % results in 15 decimal digits
[fv,k,re,i]=regulafalsi(fce,2.3,2.4,0.0001,100)        % calling of the function with real parameters
```

The Matlabu result:

```
fv =    3.552713678800501e-15       % the value of the function in the approximation

k =   2.379364594222031             % approximation of the root with given accuracy

re =    6.290953031559411e-11       % relative mistake of the approximation

i =    3                            % the number of the iterations to achieve the given accuracy
```

Example 3.

Find the root of the equation by a simple iteration method $10\cos(x-1)-x^2+2x-1=0$, in the interval $\langle 2{,}3;\ 2{,}4\rangle$ with accuracy $0{,}01\%$.

For this method, we have to note that we must first derive a contractual mapping (Banach's theorem) $x=\varphi(x)$, which generates a sequence that converges to the root. We always have more options. In this case, for example:
fce1=@(x) sqrt(10*cos(x-1)+2*x-1);
fce2=@(x) -1/2*(10*cos(x-1)-x^2-1);
fce3=@(x) 1+acos(1/10*(x^2-2*x+1));
Of course you can prove which impression is contractive, but Matlab is faster for testing. According to the results we can see that the contractive mapping is fce3=@(x) 1+acos(1/10*(x^2-2*x+1)).
Script of the function:

```
function [k,re,i]=mpi(fce,x1,me,mi)
% simple iteration method: looking for the root with initial estimate
% Input
% fce: Function its root we are looking for
% x1: initial estimate
% me: maximal relative mistake (preset = 0.0001%)
% mi: maximal number of the iteration (preset = 100)
% output:
% k: root
% re: the relative mistake of the approximation (%)
% i: the number of the iterations
   if nargin<2,error('min 2 inputi parameters'),end
   if nargin<3, me=0.005;mi=100;end
   if nargin<4, mi=100;end
% first step
   i = 0;
   xn=x1;
   while(1)
      xb=xn;
      xn=fce(x1);
      re=abs((xn-xb)/xn);
      if((re<me) || mi<=i),break;end
      x1=xn;
      i=i + 1;
   end
      k=xn;
end
```

Script of the solution of the ex. 2:

```
clc; close; clear;                          % clear of the Command window, graph windows
and variables
%fce=@(x) 10*cos(x-1)-x^2+2*x-1;            % defined function
fce1=@(x) sqrt(10*cos(x-1)+2*x-1);          % 1st mapping
fce2=@(x) -1/2*(10*cos(x-1)-x^2-1);         % 2nd mapping
```

```
fce3=@(x) 1+acos(1/10*(x^2-2*x+1));        % 3rd mapping
format long                                % results in 15 decimal digits
[k,re,i]=mpi(fce3,2.3,0.0001)              % calling of the function mpi with real parameters
```

The Matlab result for mappings fce1 a fce2:

```
k =   NaN                                  % approximation of the root, we can see mapping is no contractive
re =   NaN                                 % relative mistake of the root
i =   100                                  % the number of the iterations
```

The Matlab result for mappings fce3:

```
k =   2.379326468875754                    % approximation of the root, we can see mapping is contractive
re =     7.304277791300346e-05             % relative mistake of the root
i =     5                                  % the number of the iterations
```

Example 4.

Find the root of the equation using Newton method $10\cos(x-1)-x^2+2x-1=0$, in the interval $\langle 2{,}3;\ 2{,}4\rangle$ with accuracy $0{,}01\%$.

Script of the function:

```
function [fv,k,re,i]=newton(fce,fceder,x1,me,mi)
% newton: looking for the root with initial estimate
% open method
% Input
% fce: Function its root we are looking for
% fceder: the derivation of the function its root we are looking for
% x1: initial estimation
% me: maximal relative mistake (preset = 0.0001%)
% mi: maximal number of the iteration (preset = 100)
% Output
% k: root
% fv: the value of the function in the approximation
% re: the relative mistake of the approximation (%)
% i: the number of the iteration

    if nargin<3,error('min 3 input parameters'),end
   if nargin<4, me=0.005;end
   if nargin<5, mi=100;end
% first step
   i = 0;
   xn=x1;
   if fceder(xn)==0,error('bad estimate, divergent'),end
   while(1)
            xb=xn;
      xn=xn-fce(xn)/fceder(xn);
      i = i + 1;
      if xn==0,xn=0.001;warning('Newton method can be divergent, try another method');end
```

```
    re=abs(xn-xb)/xn;
    if((re<=me) || mi<i),break;end
  end
    k=xn;
    fv =fce(k);
end
```

Script of solving of ex. 4:

```
clc; close; clear;                    % clear of the Command window, graph windows and variables
fce=@(x) 10*cos(x-1)-x^2+2*x-1;       % defined function
fceder=@(x) -10*sin(x-1)-2*x+2;       % derivation of the defined function
format long                           % results in 15 decimal digits
[fv,k,re,i]=newton(fce,fceder,2.3,0.0001,50)          % calling of the function newton with real parameters
```

Výsledek Matlabu:

```
fv =    -1.0152003682151l9e-09   % the value of the function in the approximation

k =   2.379364594302756          % approximation of the root with given accuracy

re =    9.586114182151763e-06    % relative mistake of the approximation

i =    3                         % the number of the iteration
```

Example 5.

Find the root of the equation using secant method $10\cos(x-1)-x^2+2x-1=0$, in interval $\langle 2{,}3;\ 2{,}4\rangle$ with accuracy $0{,}01\%$.

Script of the function:

```
function [k,re,i]=secna(fce,x1,x2,me,mi)
% secant method: looking for the root using 2 initial estimate
% Open method
% Input
% fce: the funkce its root we are looking for
% x1: initial estimation 1
% x2: initial estimation 2
% me: maximal relative mistake (preset = 0.0001%)
% mi: maximal number of the iteration (preset = 100)
% Output:
% k: root
% re: the relative mistake of the approximation (%)
% i: the number of the iterations

  if nargin<3,error('min 3 input parameters'),end
  if nargin<4, me=0.005;mi=100;end
  if nargin<5, mi=100;end
% first step
  i = 0;
  while(1)
```

```
    t1=fce(x1);
    t2=fce(x2);
    if t1==t2,error('bad estimate, divergent');end
    x3=(x1*t2-x2*t1)/(t2-t1);
    i=i + 1;
    re=abs((x3-x2)/x3);
    if((re<me) || mi<i),break;end
    x1=x2;
    x2=x3;
  end
    k=x3;
end
```

Script of the solution of the ex. 5:

```
clc; close; clear;                    % clear of the Command window, graph windows and variables
fce=@(x) 10*cos(x-1)-x^2+2*x-1;       % defined function
format long                           % results in 15 decimal digits
[k,re,i]=secna(fce,2.3,2.4,0.0001,100)        % calling of the function secna with real parameters
```

Výsledek Matlabu:

```
k =   2.379364594257317          % approximation of the root with given accuracy

re =    3.549720839167417e-07    % relative mistake of the aproximation
i =    3                         % the number of the iteration
```

Example 6.

Solve the system of the nonlinear equations using the Newton method $\begin{pmatrix} 1-4x+2x^2-2y^3 \\ -4+x^4+4y+4y^4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ with initial approximation $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0{,}1 \\ 0{,}7 \end{pmatrix}$ with accuracy $0{,}0000001$.

The solution using Newton's iteration method is described in four steps. In the first step we read the input parameters and define the accuracy of the solution: maximum number of iterations (mi), precision (eps) and number of equations of the system. In the second one we retrieve all the system equations, which we define as an anonymous function and a vector of initial estimates. In the third step, we define an iterative process that will repeat until the required accuracy is met. Here we have to calculate the value of given functions in estimates, group the matrix of numerical derivatives of functions into Jacobian, solve the system of equations, calculate the following estimate and test whether the convergence criterion for given accuracy is met. Finally, we write estimates that meet the required accuracy and number of iterations.

Script solution of example 6:

```
clc; close; clear;
%  Solver of systems of nonlinear equations:
%  this script solves the system of nonlinear equations using Newton's iteration method
%  steps:
```

```
%  1) loading of input and solution parameters:
%  - maximal number of the iterations: mi
%  - the accuracy of convergence: eps
%  - the number of the equations: n
%  2) loading of the equations
%  - loading of the all equations of the system
%  - loading of the vector of the initial estimations
%  3) Iteration (repetition until desired accuracy is achieved):
%     3.1) the grouping the matrix of the derivations of the functions
%     3.2) calculate new values of the variables
%     3.3) comparing the difference between the new and old estimates and the convergence criterion (eps)
%  4) Presentation and saving of calculated values
% 1 loading of input and solution parameters:
mi=1000;
eps=0.0000001;
n=2;
% 2 loading of the equations
%F1 = @(x(1),x(2)) 1-4*x(1)+2*x(1)^2-2*x(2)^3;
% equations are defined as anonymous functions;
%  must be numbered gradually: F1,F2,... ,Fn;
%  must be grouped to the array of the functions: F={F1,F2,... Fn}
%  variables must be numbered: x(1),x(2),... ,x(n);
%1 equation: (F1)
F1 = @(x) 1-4*x(1)+2*x(1)^2-2*x(2)^3;
%2 equation: (F2)
F2 = @(x) -4+x(1)^4+4*x(2)+4*x(2)^4;
% the array of the functions
F={F1,F2};
%  initial estimations:
x(1)=0.1;
x(2)=0.7;
% ... x(n)=a;
% 3) Iteration
ni=0;
nc=0;
while ni<mi
   % grouping of the derivation matrix
   dFidxj=zeros(n,n);
   for i=1:1:n
      for j=1:1:n
         derivace;
      end
   end
   % calculate vector of function values in points
   for i=1:1:n
     j=F{i};
     D(i,1)=j(x);
   end
   % computing of the mistake
```

```
   A=inv(dFidxj);
   error=A*D;
   dx=error';
   % adding an error to x gives a new estimate
   x=x-dx;
   % Parameters for the ending of the while loop
   ni=ni+1;
   nc=nc+1;
% convergence criterion (error < eps)
   maxdx=max(abs(error));
      if maxdx<eps
         ni=mi;
      end
end
% 4) Presentation and saving of calculated values
x
nc
```

Script for calculating of the numerical derivatives in a file derivace.m:

```
% calculate the system derivative
fx=F{i};
xj=x;
%function behind the point
xj(j)=x(j)+eps/2;
y1=fx(xj);
% function in front of the point
xj(j)=x(j)-eps/2;
y0=fx(xj);
%Derivation in the point
dFidxj(i,j)=(y1-y0)/eps;
```

The Matlab result:

```
x =    0.0618    0.7245
nc =     4
```