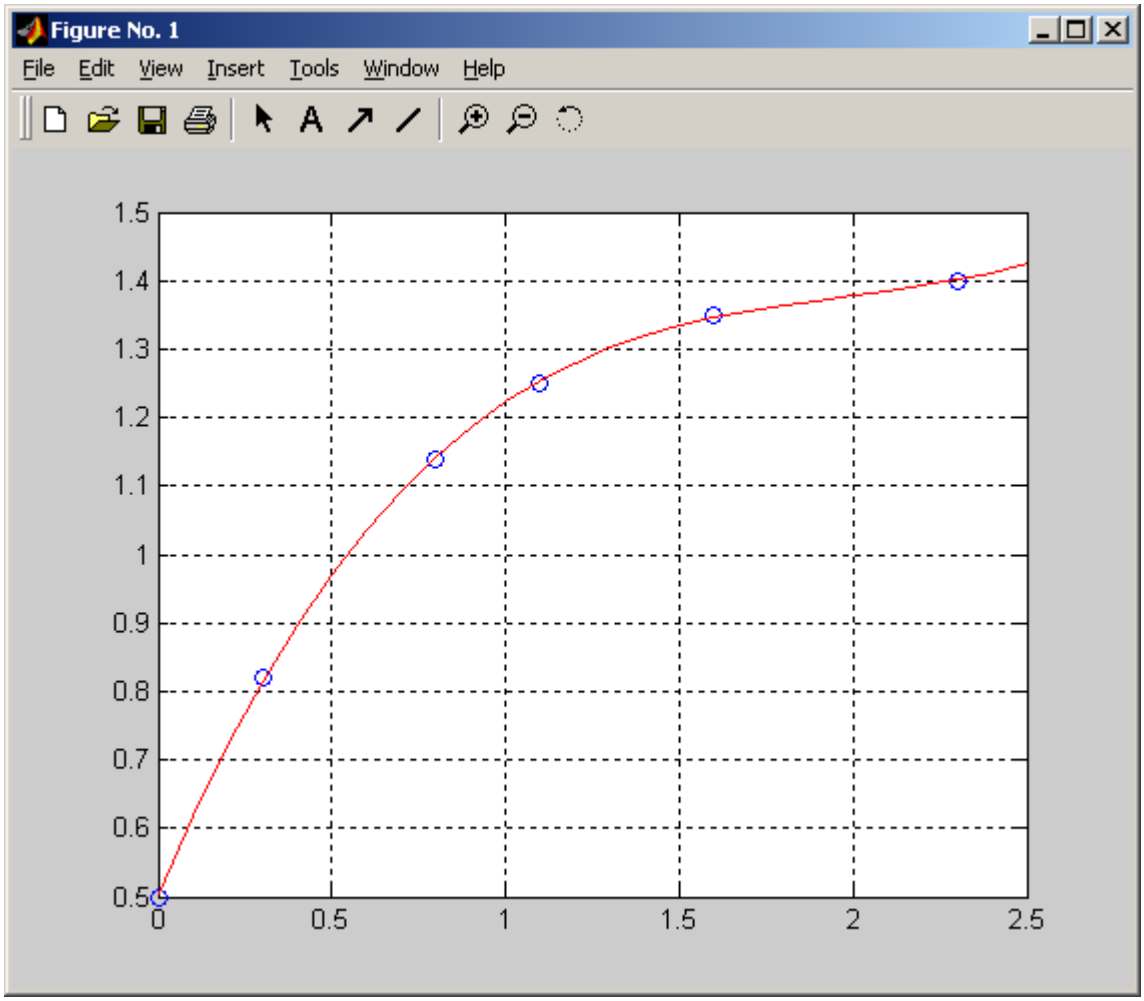
Monday – part 2

Petr Kropík

Polynomial functions and regression

```
%Polynomial functions
%Part of visualisation of input data
%---------------------------------
t=[0,.3,.8,1.1,1.6,2.3]';
y=[0.5,0.82,1.14,1.25,1.35,1.40]';
plot(t,y,'o')
grid on
hold on
%Part computerisation of regression data
%--------------------------------------
X=[ones(size(t)),t,t.^2,t.^3]
a=X\y
%Polynomial regression
%-------------------------------------
T=(0:0.1:2.5)';
Y=[ones(size(T)),T,T.^2,T.^3]*a;
plot(T,Y,'r-')
```

Output window



Polynomial regression of 3 degree

Polynomials

$p(x) = 4x^5 + 3.1x^3 – 7x^2 + 11$

$q(x) = –x^4 + x^3 –x$

In Matlab like a vektors:

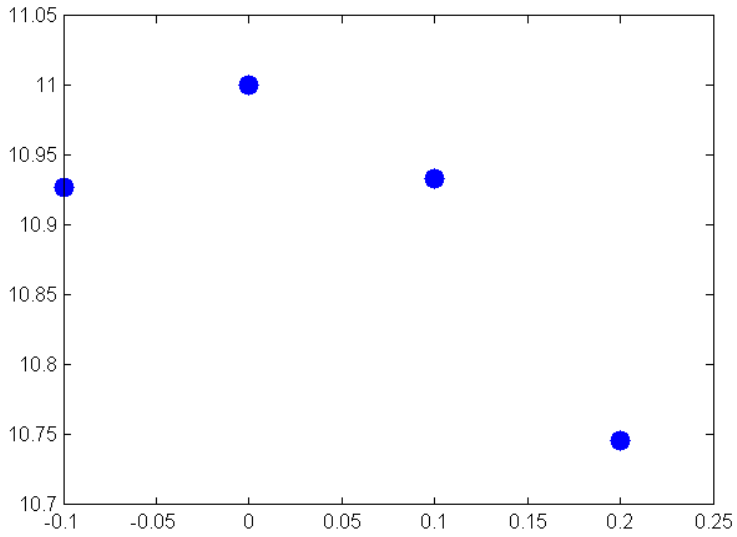
```
>> p = [4, 0, 3.1, -7, 0, 11]
p =
   4.0000         0    3.1000   -7.0000         0   11.0000

>> q = [-1, 1, 0, -1, 0]
q =
   -1     1     0    -1     0
```

How does MATLAB know, what does it p and q (vectors or polynomials)?
MATLAB doesn't know it.
p and q are vectors, and interpretation is depend on method of usage.

polyval(p, x) – value of polynomial p depend on x

```
>> polyval(p, 2)
ans =
  135.8000
>> x = -.1:0.1:0.2
x =
   -0.1000         0    0.1000    0.2000
>> polyval(p, x)
ans =
   10.9269   11.0000   10.9331   10.7461
```

```
plot(x, polyval(p, x), '.')
```



conv(p, q) – convolution vectors p and q

x = roots(p) – returns a column vector whose elements are the roots of the polynomial p (find x where p has a value 0)

p = polyfit(x, y, n) – finds the coefficients of a polynomial p(x) of degree n that fits the data (x, y) in a least squares sense. The result p is a row vector of length n+1 containing the polynomial coefficients in descending powers

Flow control, graphs and m-file editor

Flow control statements – loops (for, while), conditions (if, switch-case).

– constructs like in "standard" programming laguages

`for`

Example:

```
x = [];
for i = 1:n
      x=[x, i^2];
end
```

– this example creates a vector size n.

Example:

```
x = [];
for i = 1:-1:n
      x=[x, i^2];
end
```

– the same vector, but in reverse order (loop step is set to –1)

Example:

```
for i = 1:m
      for j = 1:n
            H(i, j) = 1/(i+j-1);
      end
end
H
```

– this example creates a Hilbert's matrix size m x n.

`while`

Example:

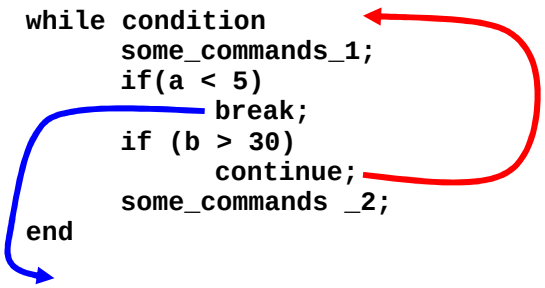
```
n = 0;
while (2^n < a)
      n = n + 1;
end
n
```

– this example for given number `a` computes and shows smallest non-negative number `n`, where `2^n>= a`

Loop control

`break;` – terminate execution of for or while loop

`continue;` – The continue statement passes control to the next iteration of the for or while loop in which it appears, skipping any remaining statements in the body of the loop



try and catch

The general form:

```
try
      some_commands_1;
catch
      some_commands _2;
end
```

In this sequence, the statements between try and catch are executed until an error occurs. The statements between catch and end are then executed. Use **lasterr** to see the cause of the error. If an error occurs between catch and end

if

Example:

```
if n < 0
      parity = 0;
elseif (rem(n,2) == 0)
      parity = 2;
else
      parity = 1;
end
```

switch

Example:

– conversion of cm to the unit defined in variable **units**

```
x = 2.7;
units = 'm'
switch units   % convert x to centimeters
      case { 'inch', 'in' }
            y = x * 2.54;
      case { 'feet', 'ft' }
            y = x * 2.54 * 12;
      case { 'meter', 'm' }
            y = x / 100;
      case { 'millimeter', 'mm' }
            y = x * 10;
      case { 'centimeter', 'cm' }
            y = x;
      otherwise
            disp(['Unknown Units: ', units])
            y = NaN;
end
```

Comparison operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
= =	equal to
~=	not equal to

Attention

– character = means assignment of a value to a variable
– character == means equal to
– relations can be joined with help of logical operators

&	and
|	or

~ not

Attention
– result of relation between scalars is a scalar (value 1 – means true or value 0 – means false)
– result of relation between matrixes (with same size) is a relation matrix of 0 and 1 as the result of comparison of elements at the same coordinates
– result of relation between two matrixes is whole true, if all elements of relation matrix are 1 (true)

Example:
Is matrix A equal to matrix B ?

```
if A == B
      command;
end
```

Is matrix A non-equal to matrix B ?

```
if any(any(A ~= B)) % because we must test
      command;
end
```

or more simply:

```
if A == B
      ;
else
      command;
end
```

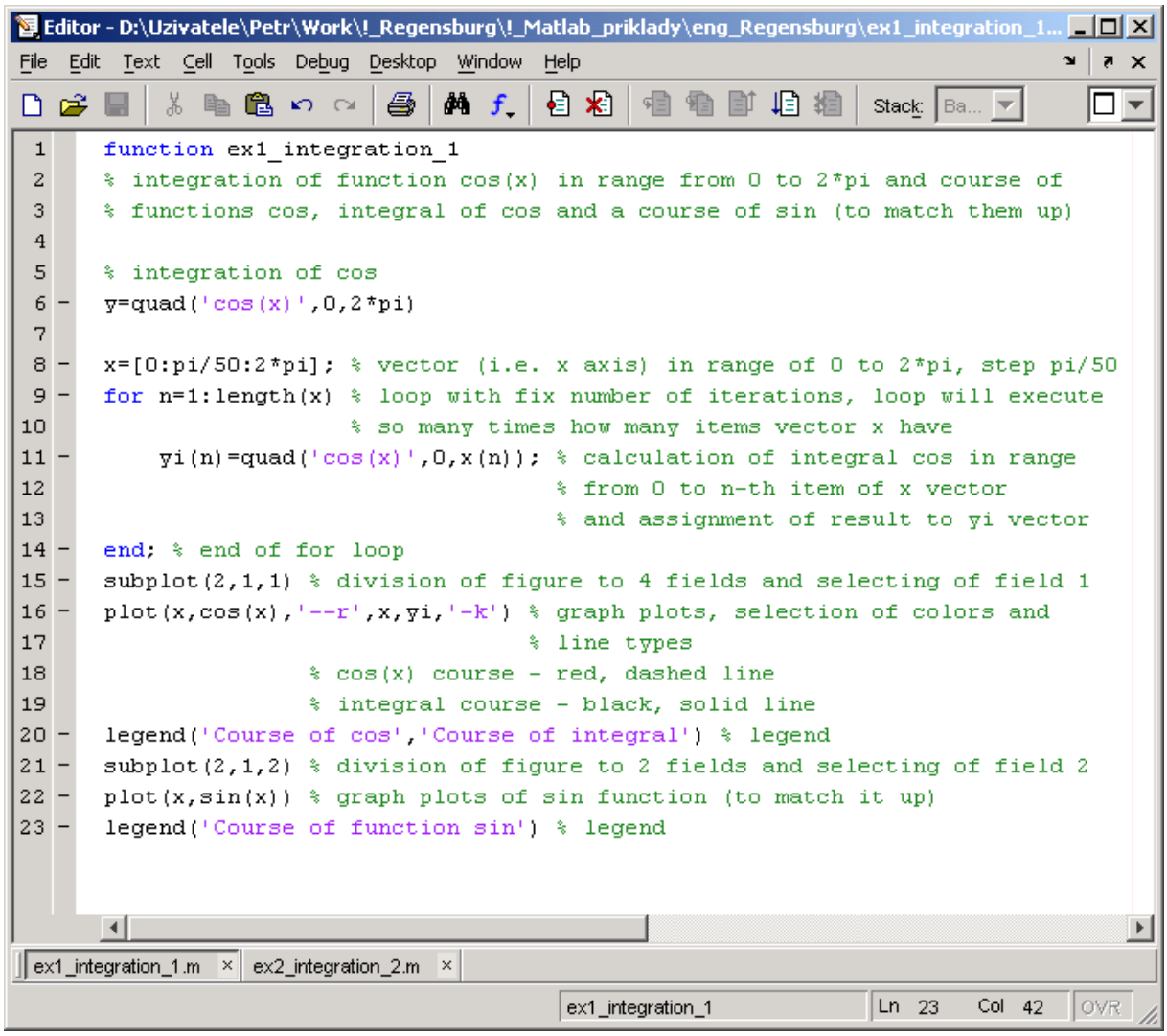
Command

```
if A ~= B
      command;
end
```

doesn't work properly, because **`command;`** will be executed only in case of whole relation matrix contains zeros

Useful operators:
`any` at least one non-zero element
`all` all non-zero elements

Ad 21. Editing m-files – creating functions (examples – integral and equation solving, user data input)



– variables in functions are usually local (compare with situation in scripts, please)

Example:

```
function c = randint(m,n)
      % randint Randomly generated integer matrix.
      % randint(m,n) returns an m-by-n such matrix with entries
      % between 0 and 9.
      c = floor(10*rand(m,n));
```

Example:

```
function c = randint2(m,n,a,b)
      % randint2 Randomly generated integer matrix.
      % randint(m,n,a,b) returns an m-by-n such matrix with entries
      % between a and b.
      % rand(m,n,a,b) return entries between integers a and b.
      % nargin represents number of input arguments
      if nargin < 4, a = 0, b = 9; end
      c = floor((b-a+1)*rand(m,n)) + a;
```

Example: one function calls another

File sumvar.m

```
function c=sumvar(a,b)
      c = a + b;
```

File mulvar.m

```
function mulvar
      a = 5;
      b = 3;
      c = sumvar(a, b);
      d = a * c;
```

Example:

Function with more output arguments

```
function [mean, stdev] = stat(x)
      % STAT Mean and standard deviation
      % For a vector x, stat(x) returns the
      % mean and standard deviation of x.
      % For a matrix x, stat(x) returns two row vectors containing,
      % respectively, the mean and standard deviation of each column.
      [m n] = size(x);
      if m == 1
           m = n; % handle case of a row vector
      end
      mean = sum(x)/m;
      stdev = sqrt(sum(x.^2)/m - mean.^2);
```

Function is saved in file `stat.m`

Run (on command line):

```
      [xm, xd] = stat(x)
```

Example:

```
function ex1_integration_1
      % integration of function cos(x) in range from 0 to 2*pi and course of
      % functions cos, integral of cos and a course of sin (to match them up)

      % integration of cos
      y=quad('cos(x)',0,2*pi)

      x=[0:pi/50:2*pi]; % vector (i.e. x axis) in range of 0 to 2*pi, step pi/50
      for n=1:length(x) % loop with fix number of iterations, loop will execute
                        % so many times how many elements vector x have
            yi(n)=quad('cos(x)',0,x(n)); % calculation of integral cos in range
                                  % from 0 to n-th element of x vector
                                  % and assignment of result to yi vector
      end; % end of for loop
      subplot(2,1,1) % division of figure to 4 fields and selecting of field 1
      plot(x,cos(x),'--r',x,yi,'-k') % graph plots, selection of colors and
                                     % line types
                  % cos(x) course - red, dashed line
                  % integral course - black, solid line
      legend('Course of cos','Course of integral') % legend
      subplot(2,1,2) % division of figure to 2 fields and selecting of field 2
      plot(x,sin(x)) % graph plots of sin function (to match it up)
      legend('Course of function sin') % legend
```

Example:

```
function ex2_integration_2
      % integration of function sin(x) in range from 0 to 2*pi and course of
      % functions sin(x), integral of sin(x) and a course of -cos(x) (to match them up)

      % integration of sin
      y=quad('sin(x)',0,2*pi)

      x=[0:pi/50:2*pi]; % vector (i.e. x axis) in range of 0 to 2*pi, step pi/50
      for n=1:length(x) % loop with fix number of iterations, loop will execute
                        % so many times how many elements vector x have
          yi(n)=quad('sin(x)',0,x(n)); % calculation of integral sin in range
                                       % from 0 to n-th element of x vector
                                       % and assignment of result to yi vector
      end; % end of for loop
      subplot(2,1,1) % division of figure to 4 fields and selecting of field 1
      plot(x,sin(x),':g',x,yi,'-k') % graph plots, selection of colors and
                                    % line types
                     % sin(x) course - green, dotted line
                     % integral course - black, solid line
      legend('Course of sin','Course of integral') % legend
      subplot(2,1,2)  % division of figure to 2 fields and selecting of field 2
      plot(x,-cos(x)) % graph plots of -cos(x)(to match it up)
      legend('Course of function -cos') % graph plots of -cos function (to match it up)
      axis([0,7,-2,1]) % axis range redefinition
```

MATLAB – differential equations

1. ODE-functions – solving of first order ordinary differential equations

Functions:

```
ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb
```

Syntax:

```
[time, solution]=ode45(@name_of_m-file, [interval of solution], [a vector of
initial conditions]);
```

Example:

```
[t,y]=ode45(@diff_rov,[0,3],[1,0,0]);
```

(detailed example see in chapter 4)

2. System of differential equations.

```
function dy=sys_diff_eq(t,y)
% solving of the system of differential equations
% dy1/dt=-4*y1+10*y2*y3
% dy2/dt=4*y1-10*y2*y3-30*y2^2
% dy3/dt=30*y2^2

dy=zeros(3,1);
dy(1)=(-4*y(1))+(10*y(2)*y(3));
dy(2)=(4*y(1))-(10*y(2)*y(3))-(30*y(2)^2);
dy(3)=30*y(2)^2;

function solving
% solving of system of differential equations

[t,y]=ode45(@sys_diff_eq,[0,3],[1,0,0]); % solving of system of differential equations
                                         % defined in m-file sys_diff_eq
                                         % interval of integration from 0
                                         % to 3 and vector of initial conditions
                                         % y1(0)=1,y2(0)=0,y3(0)=0
figure % graphical window creating
subplot(3,1,1) % dividing graph to three parts (subplots), active is 1st
plot(t,y(:,1)) % 2D plot drawing
xlabel('t')    % x axis label
ylabel('y_1')  % y axis label
title('Reseni pomoci ode45') % graph's title
subplot(3,1,2) % dividing graph to three parts (subplots), active is 2nd
plot(t,y(:,2)) % 2D plot drawing
xlabel('t')    % x axis label
ylabel('y_2')  % y axis label
subplot(3,1,3) % dividing graph to three parts (subplots), active is 3rd
plot(t,y(:,3)) % 2D plot drawing
xlabel('t')    % x axis label
ylabel('y_3')  % y axis label

figure % new graphical window creating (second)
plot(t,y) % 2D plot drawing
legend('y_1','y_2','y_3') % legend
```

3. Higher orders differential equations.

Example:
Solution of the Van der Pol differential equation

$$\frac{d^2 y_1}{dt^2} - \mu\left(1 - y_1^2\right)\frac{dy_1}{dt} + y_1 = 0$$

It's a second order diff. equation. We must modify it to the system of first order differential equations, using substitution.

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = \mu\left(1 - y_1^2\right) y_2 - y_1$$

$$\frac{dy_2}{dt} - \mu(1 - y_1^{\,2}) y_2 + y_1 = 0$$

In MATLAB:

There are two ways to define system of diff. equations:

1st way:

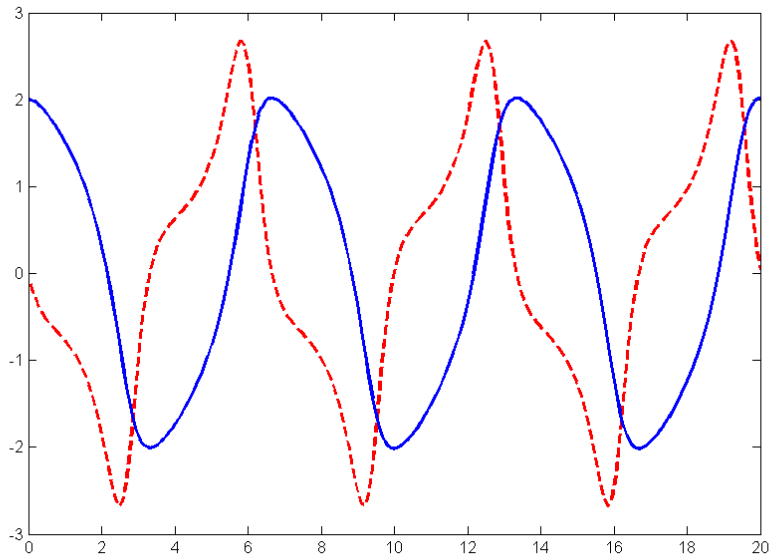
```
function dy=fce_vdp(t,y)
mi=1;
dy=[y(2);mi*(1-y(1)^2)*y(2)-y(1)];
```

or

2nd way:

```
function dy=fce_vdp(t,y)
mi=1;
dy(1,1)= y(2);
dy(2,1)= mi*(1-y(1)^2)*y(2)-y(1);
```

```
function difrov
% solving of the system of first order differential equations
[t,y]=ode45(@fce_vdp,[0 2*pi],[0 1]);

% solution printing
for cykl=1:length(t)
    fprintf('t = %8.4f\ty1 = %8.4f\ty2 = %8.4f\n',t(cykl),y(cykl,1),y(cykl,2));
end

% graphs
plot(t,y(:,1),'-b','LineWidth',2);
hold on
plot(t,y(:,2),'-r','LineWidth',2);

title('Solution of the VDP');
xlabel('t');
ylabel('y_{1,2}');
legend('y_1','y_2');
```



Strings, interactive input / output commands, user defined error messages, commentary (functions input, error)

Text strings, error messages, user input

– text strings are enclosed by apostrophes (single quote marks)

Example:
```
s = 'This is a test';
```

Example:
```
disp('This message is hereby displayed');
```
or
```
s = 'This message is hereby displayed';
disp(s);
```

Function `error`
– if this function is used in m-file, it terminate execution of it

Example:
```
error('Sorry, the matrix must be symmetric');
```

Function `input`
– it is used for interactive input of values from user

Example:
```
iter = input('Enter the number of iterations: ');
```

Example:
Greatest common divisor of two integer numbers (Euclides algorithm)

```
function a = mygcd(a,b)
      % GCD Greatest common divisor
      % mygcd(a,b) is the greatest common divisor of
      % the integers a and b, not both zero.
      a = input('a = ');
      b = input('b = ');
      a = round(abs(a));
      b = round(abs(b));
      if a == 0 & b == 0
            error(' Greatest common divisor is not defined, because both numbers has zero
      value.')
      else
            while b ~= 0
                  r = rem(a,b);
                  a = b;
                  b = r;
            end
      end
```

Example
```
num_err = 0;
while(1)
    bottom = input('Zadejte pocatecni hodnotu: ');
    top = input('Zadejte koncovou hodnotu: ');
    if (top > bottom)
        break;
    end;
    num_err = num_err + 1;
    switch num_err
        case 0
            disp('Program error, zero mistakes, it's impossible ?!?!');
```

```
            case 1
                disp('First mistake...');
                disp('Bottom value can't be lower then top value...');
            case 2
                disp('Second mistake...');
                disp('Bottom value can't be lower then top value...');
            case 3
                disp('Third mistake...');
                disp('Bottom value can't be lower then top value...');
            case {4, 5, 6, 7, 8, 9}
                disp('4. – 9. mistake...');
                disp('Bottom value can't be lower then top value...');
            case 10
                disp('Jubileum! Bingo! 10. mistake!');
                disp('Bottom value can't be lower then top value...');
            otherwise
                disp('We are not on speaking terms, crazy human...');
                return;
        end
    end;
```

Simple text output

```
disp(variable);
disp('text string');
```

- there is no way to format output

Formatted text output

fprintf

- print values in form: 8 chars, add spaces on left handed side

```
fprintf('%8d %8d\n', a, b);
      56       34
```

- print values in form: 8 chars, add zeros on left handed side at first time and add spaces on left handed side at second time

```
fprintf('%08d %8d\n', a, b);
00000056       34
```

- typed value 8 means "at least 8 chars".
It mustn't crop the longer integer.

- in case of decimal numbers – we want at whole 15 chars including decimal point and including 2 decimal places

```
>> fprintf('%15.2f\n', x);
         123.35
```

- if we want to print a sign every time (+ and -):

```
>> fprintf('%+d %+d\n', a, b);
+56 -34
```

Important special characters:

\n – new line
\t – tabulator
\r – return to the same line beginning (it's operating system depend)
\b – in some operating systems – beep (loudspeaker).
**** – print char \

%% – print char %

%d – print integer decadic number (sign)
%i – print integer decadic number (sign)
%o – print octal scale number
%u – print decimal (decadic) number (unsign)
%x or **%X** – print hexadecimal number
%f – print decimal number – fixed decimal point
%e or **%E** – print decimal number – exponential form
%g or **%G** – like **%f** and **%e** resp. **%E** – exponential form is used, if it is needed (if the number is to big). High order zeros are omitted.
%c – print character (stored in variable)
%s – print text string (stored in variable)

Example:

```
a =
   234
>> fprintf('Decadic scale: %d\nOctal scale:
%o\n', a, a);
Decadic scale: 234
Octal scale: 352
>> fprintf('Hexadecimal scale: %x\n', a);
Hexadecimal scale: ea
```

```
>> fprintf('Hexadecimal scale: %X\n', a);
Hexadecimal scale: EA

x =
  123.3456

>> fprintf('Fixed decimal point: %f\n', x);
Fixed decimal point: 123.345600

>> fprintf('exponencial: %e\n', x);
exponencial: 1.233456e+002

>> fprintf('EXPONENCIAL: %E\n', x);
EXPONENCIAL: 1.233456E+002

>> fprintf('"Smart" style g: %g\n', x);
"Smart" style g: 123.346

>> fprintf('"Smart" style g: %g\n', x * 100000000);
"Smart" style g: 1.23346e+010

>> fprintf('"SMART" STYLE G: %G\n', x);
"SMART" STYLE G: 123.346
x =
  1.2344e+034

>> fprintf('"SMART" STYLE G: %G\n', x * 100000000);
"SMART" STYLE G: 1.23346E+010

in_char = input('Enter a character: ', 's');
fprintf('Entered: %c\n', zadany_znak);
Was entered: n

in_text = input('Your text: ', 's')
>> fprintf('Was entered: %s\n', in_char);
Was entered: Wie geht es Ihnen ?
```

Files – basics

```
fd = fopen('C:\\file_path\\file.txt', 'mode');
```

where
fd – variable, so-called file descriptor
mode should be:
'r' – read
'w' – write, overwrite (re-write), create new file (if it is possible)
'a' – append at the end of the existing file
'r+' – read and write
'w+' – read and write, overwrite (re-write), create new file (if it is possible)
'a+' – read and write, overwrite (re-write), create new file (if it is possible) and append at the end of the existing file

– binary files is available too, but it is more complicated.

Write to text file:
fprintf – like on the screen, only one difference – we must type the name of variable where is a file descriptor is stored.

```
my_file = fopen('C:\\file_path\\file.txt', 'w');
fprintf(my_file, parameters like on screen)
```

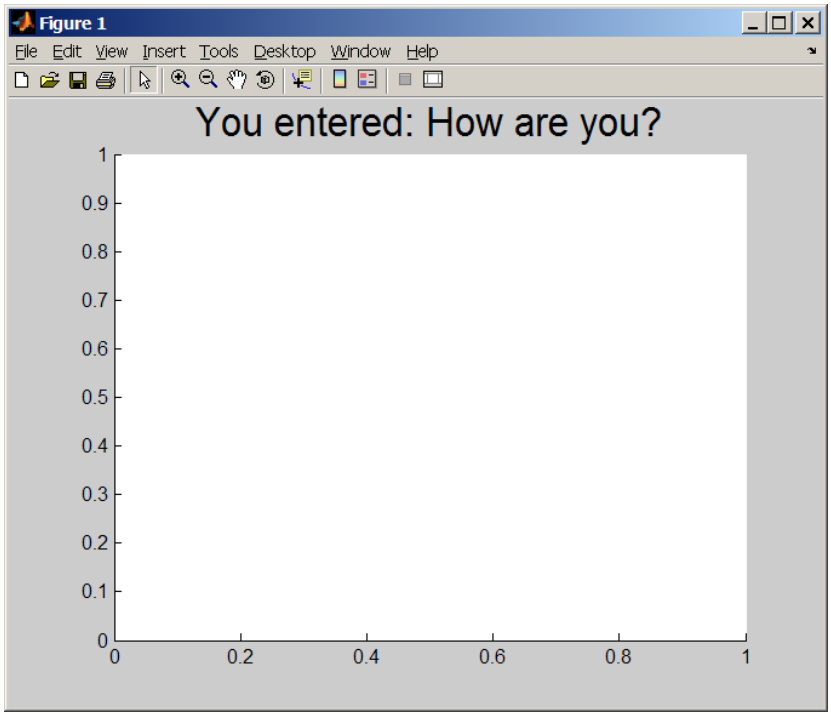
At the end we must close our file:

```
fclose(my_file);
```

```
>> my_file = fopen('datei.txt', 'w');
>> fprintf(my_file,'Decadic scale: %d\n', a);
>> fprintf(my_file,'Octal scale: %o\n', a);
>> fprintf(my_file,'Hexadecimal scale: %X\n', a);
>> fclose(my_file)
```

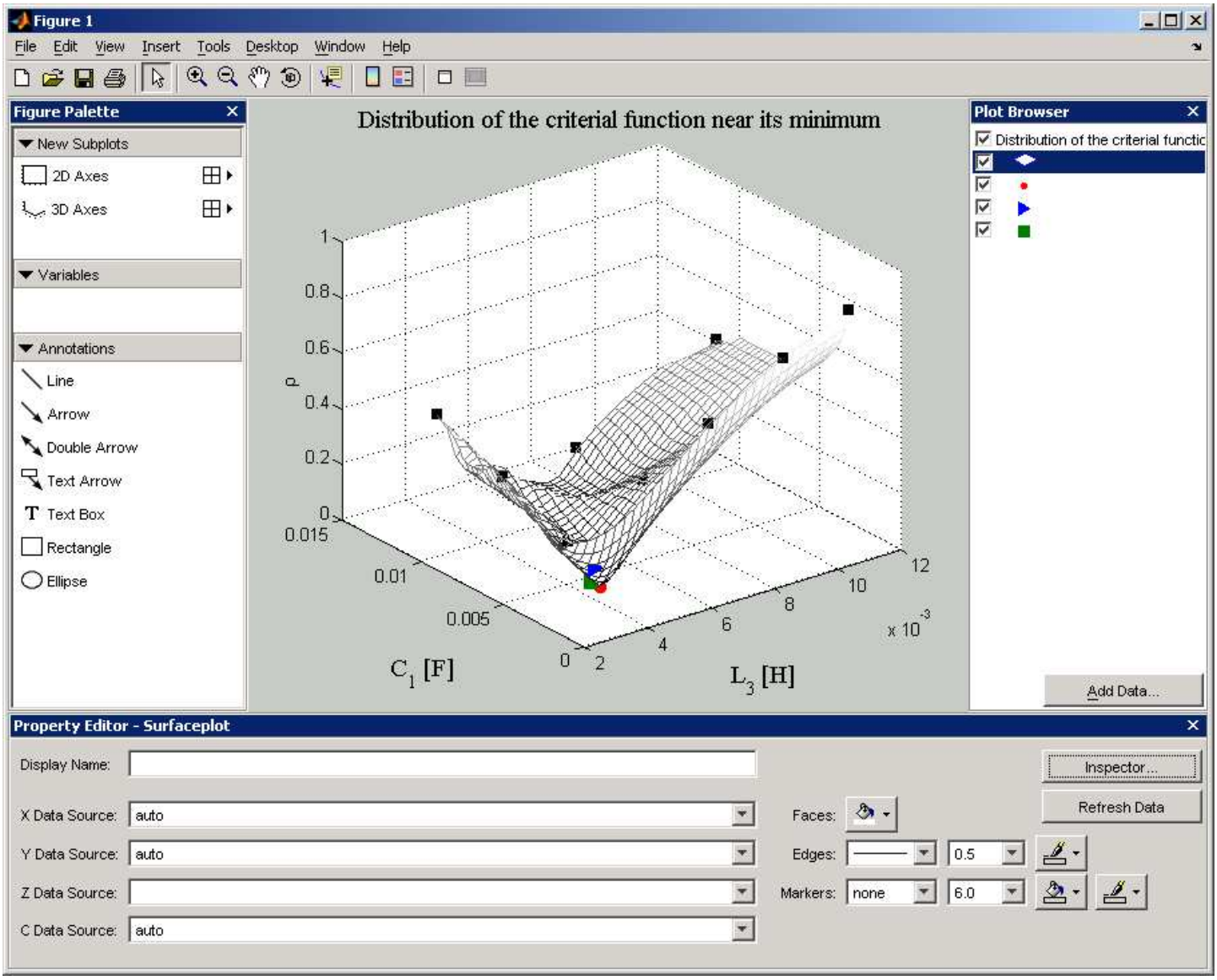
Write to string variable:

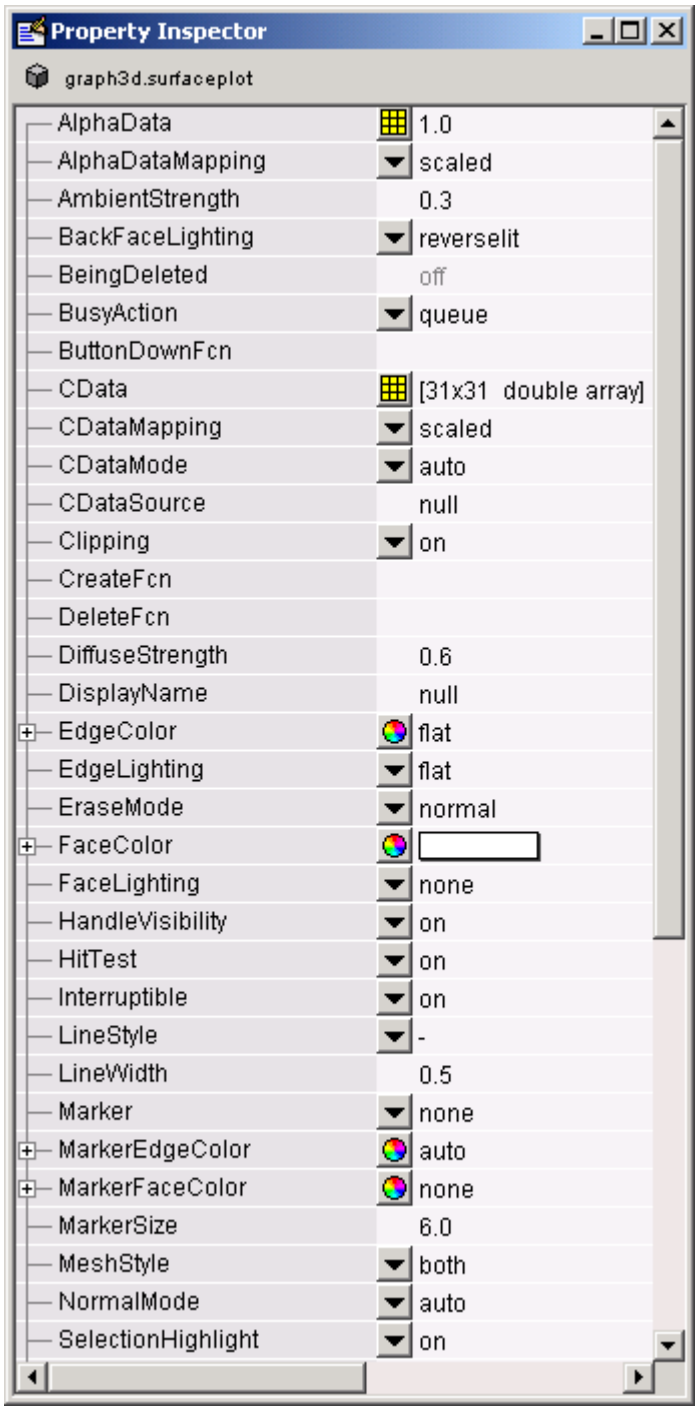
```
res_string = sprintf(parameters like on screen);

mess = sprintf('You entered: %s\n', in_text);

in_text = input('Enter text: ', 's');
Enter text: How are you?

mess = sprintf('You entered: %s\n', in_text);
figure     % empty figure
title(mess)   % figure's title
```




```
msgbox(mess,'Problem...','error');
```

Ad 4. Graphs and figures adjustment – manually




Property Inspector
graph3d.surfaceplot
AlphaData 1.0
AlphaDataMapping scaled
AmbientStrength 0.3
BackFaceLighting reverselit
BeingDeleted off
BusyAction queue
ButtonDownFcn
CData [31x31 double array]
CDataMapping scaled
CDataMode auto
CDataSource null
Clipping on
CreateFcn
DeleteFcn
DiffuseStrength 0.6
DisplayName null
EdgeColor flat
EdgeLighting flat
EraseMode normal
FaceColor
FaceLighting none
HandleVisibility on
HitTest on
Interruptible on
LineStyle -
LineWidth 0.5
Marker none
MarkerEdgeColor auto
MarkerFaceColor none
MarkerSize 6.0
MeshStyle both
NormalMode auto
SelectionHighlight on