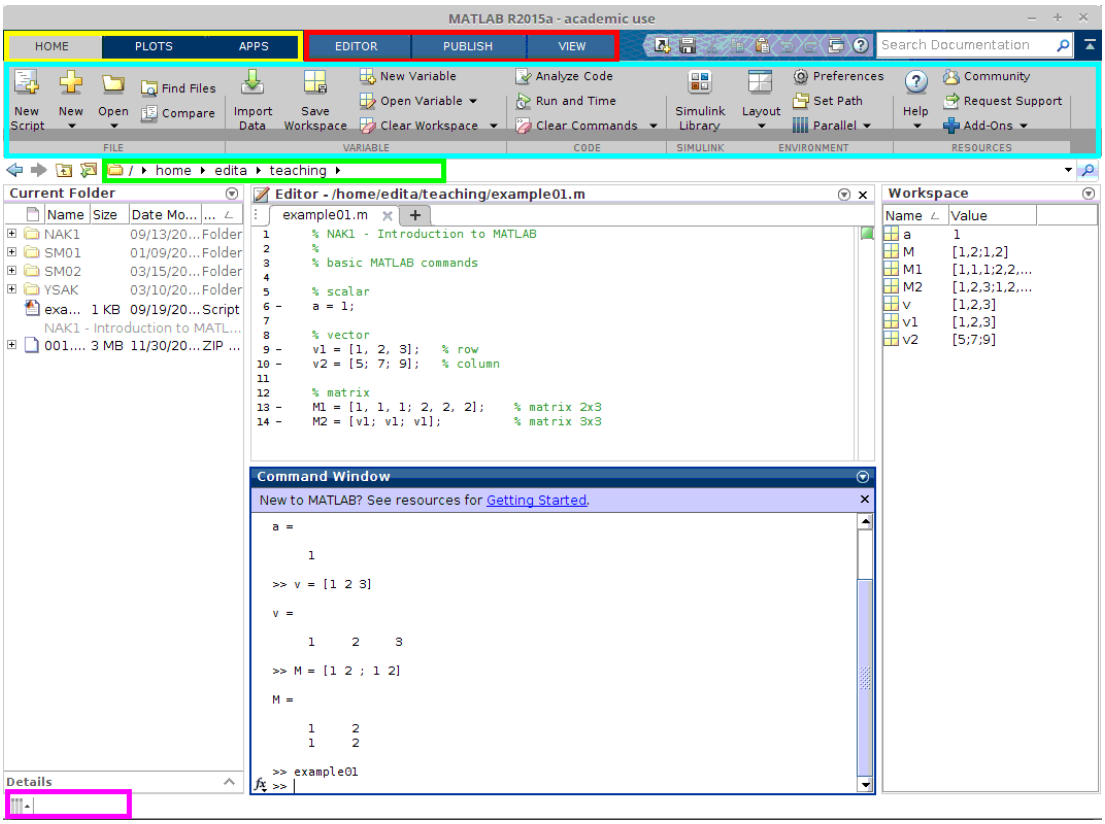
1 Seminar 1: Introduction to programming in Matlab/Octave

1.1 Matlab

1.1.1 Environment

Student's license for CTU students can be downloaded from https://download.cvut.cz/ (after login) Connection must be from the CTU domain.



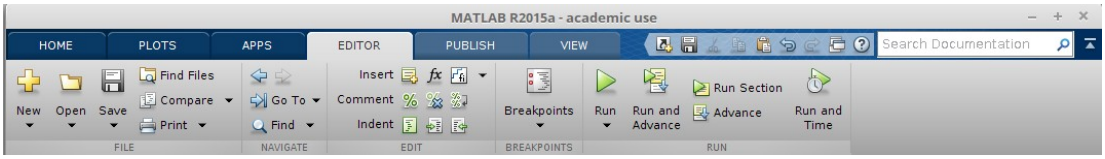
The default Matlab environment is in the figure. This main window can be controlled via three tabs (yellow box). Actually, the home tab is active with commands in the cyan box.

In the green box, there is a path to working directory.

The following windows are - Current Folder, Command Window, Editor, and Workspace.

In the Editor, the script example01.m is actually opened. All Matlab functions and scripts have the suffix *.m. You can edit, run and debug functions and scrips in the Editor.

The most important tab is EDITOR:



1.1.2 Methods of work

Two ways of work in Matlab:

- Work in Command Window

  – commands written directly in Command Window,
  Matlab evaluates commands in the order of entry

  – suitable for short calculations

- Work in Editor

  – a batch of commands are written in a script – program, algorithm
  – the program (script) is run as a whole, outputs are in Command Window
  – comments can be inserted
  – you can debug the script
  – you can write the whole clear and transparent program

1.1.3 Good to know

- Key F1

  – F1 runs the "Help"

- Font size

  – HOME → Preferences → Fonts

- Keyboard shortcuts

  – HOME → Preferences → Keyboard → Shortcuts

- Semicolons

  – semicolon indicates end of statement. If you want to suppress and hide
  the MATLAB output for an expression, add a semicolon after the expression.

  – If you are interested in the values of variables, the command line will be without a semicolon.
  Values will be printed in Command Window.

- Comments

  – comments are written after %

  – the beginning of each script or function is recommended to comment – whats the purpose,
  whats are the inputs and outputs, etc.

- Three dots …

  – for the sake of clarity, the three dots can be used for line breaks, e.g., in the case of long equations

- working Matlab

  – the running script is signalized by “Busy” in the bottom left corner
  (the ping box in the first figure)

  – after finishing the last command, the “Busy” disappear

- Matlab is still busy – something is wrong

  – running complicated programs can be time-consuming.
  If the solving process lasts too long, it can be caused by an error, e.g., endless cycle, function, etc.

  – Ctrl+C interrupts (kills) the process

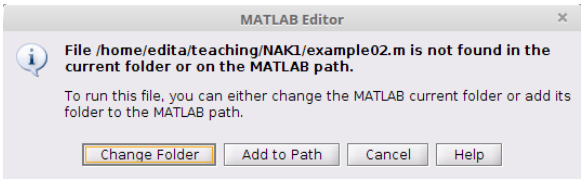
- Error reports

  – in the case of syntax mistakes or undefined operations,
  Matlab writes error reports in Command Window

  – links to lines with errors in the scripts are described in messages,
  the type of the error is included

- File is not found . . .

  – If you want to run a script in different directory than working directory this window opens.



  – you can choose from two options

    * Change Folder – it changes the actual directory to the directory of the opened files (script), the other work will continue in this directory

    * Add to Path – it adds the path to the running script, but the working directory is unchanged

- Results with Inf/NaN
  - Matlab is unable to return the solution of some operation as a value

    - Inf (infinity) – represents infinity, e. g., in the case of dividing by zero 0
    - NaN (not a number) – represents values, which are not real or complex numbers

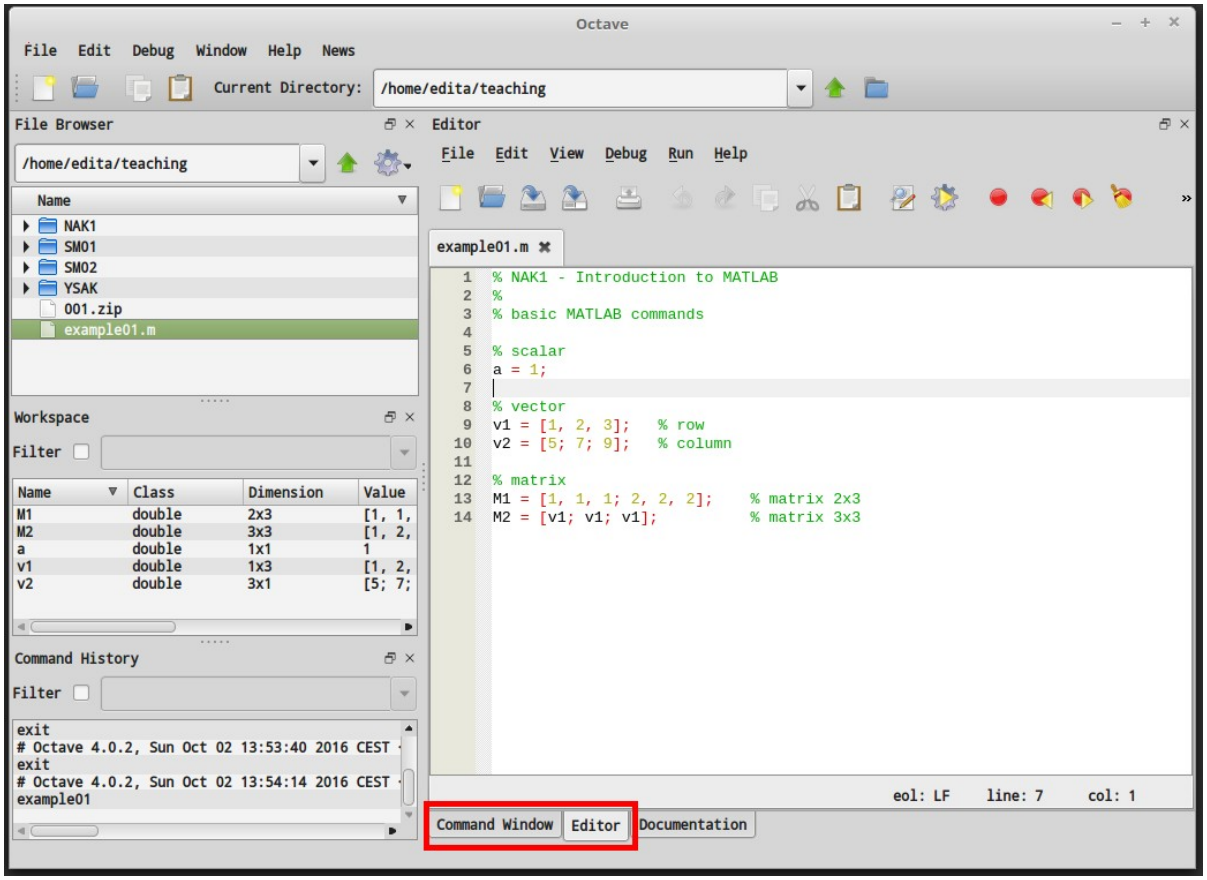
1.2 Octave

1.2.1 Environment

Octave is free software available on the website https://octave.org.

Octave can be controlled from a command line or via a graphical environment similar to Matlab.

Most of the commands and scripts are compatible with Matlab.



In the figure, the Octave window is presented with the red box switching between Editor and Command Window. Work in this environment is similar to Matlab.

1.3 Basis of programming in Matlab/Octave

1.3.1 Scalars, vectors, matrices

Matlab ("Matrix laboratory") prompts that the software is suitable for work with matrices, which are the fundamental structure in this environment. Scalar (1x1) and vector (1xn) or (nx1) are supposed to be in the general format as matrices (nxn).

```
In [1]: % scalar
        a = 1

        % vector 1x3
        v = [1, 2, 3]

        % matrix 2x3
        M = [1, 2, 3;
             2, 3, 4]
```

```
a =  1
v =

   1   2   3

M =

   1   2   3
   2   3   4
```

Matrices are written in square brackets, commas separate columns and semicolons separate rows

```
In [2]: vrow = [1, 2, 3]   % row vector
        vrow = [1 2 3]     % row vector
        vcol = [4; 5; 6]   % column vector
```

```
vrow =

   1   2   3

vrow =

   1   2   3

vcol =

   4
   5
   6
```

Matrix transpose with apostrophe

```
In [3]: vt = vcol'    % vcol = [4; 5; 6]
        Mt = M'       % m = [1, 2, 3; 2, 3, 4]

vt =

   4   5   6

Mt =

   1   2
   2   3
   3   4
```

The access to individual matrix components is through round brackets with row and column indices (r,c). We can assign values of components with an overall index via operator :

We can choose the whole row and column or the range of indices.

The numbering in Matlab starts from 1 (C and C++ start numbering from 0)

```
In [4]: a = vt(3)
        b = Mt(3,2)
        c = Mt(6)
        Mcol2 = Mt(:,2)
        Mrow12 = Mt(1:2,:)

        Mt(3,2) = 1;
        Mt

        Mt(:,1) = [0 0 0]';
        Mt

a =  6
b =  4
c =  4
Mcol2 =

   2
   3
   4
```

```
Mrow12 =

   1   2
   2   3

Mt =

   1   2
   2   3
   3   1

Mt =

   0   2
   0   3
   0   1
```

Operator : can be used for a vector definition, too. The default step value 1 is possible to change by a value between the first and the last components

```
In [5]: v1 = 1:5
        v2 = 3:-0.25:2

v1 =

   1   2   3   4   5

v2 =

    3.0000    2.7500    2.5000    2.2500    2.0000
```

Matrices and vectors can be joined in case of the same dimension

```
In [6]: M1 = [v1;v2]
        M2 = [v1 v1]

        M3 = [M1;M1;v2]
        M4 = [M3,v1']

M1 =

   1.0000   2.0000   3.0000   4.0000   5.0000
   3.0000   2.7500   2.5000   2.2500   2.0000

M2 =
```

```
   1   2   3   4   5   1   2   3   4   5

M3 =

   1.0000   2.0000   3.0000   4.0000   5.0000
   3.0000   2.7500   2.5000   2.2500   2.0000
   1.0000   2.0000   3.0000   4.0000   5.0000
   3.0000   2.7500   2.5000   2.2500   2.0000
   3.0000   2.7500   2.5000   2.2500   2.0000

M4 =

   1.0000   2.0000   3.0000   4.0000   5.0000   1.0000
   3.0000   2.7500   2.5000   2.2500   2.0000   2.0000
   1.0000   2.0000   3.0000   4.0000   5.0000   3.0000
   3.0000   2.7500   2.5000   2.2500   2.0000   4.0000
   3.0000   2.7500   2.5000   2.2500   2.0000   5.0000
```

The dimension of a matrix is find out by a command “size”, for vectors is “length”

```
In [7]: sM4 = size(M4)
        rM4 = size(M4,1)
        cM4 = size(M4,2)

        cM2 = size(M2,2)
        sv1 = length(v1)

sM4 =

   5   6

rM4 =  5
cM4 =  6
cM2 =  10
sv1 =  5
```

A row/column is deleted by an assign of empty vector []

```
In [8]: M4(:,2:4) = []

M4 =

   1   5   1
   3   2   2
```

```
   1   5   3
   3   2   4
   3   2   5
```

Matlab is able to create some special matrices – null (zeros), filled with ones (ones), unit diagonal (eye)

```
In [9]: Mzeros = zeros(2,3)
        Mones = ones(2,3)
        Meye = eye(2,3)

Mzeros =

   0   0   0
   0   0   0

Mones =

   1   1   1
   1   1   1

Meye =

Diagonal Matrix

   1   0   0
   0   1   0
```

Matrix operations – adding, subtracting, multiplying; dividing and multiplying by a scalar; watch out for correct dimensions!

```
In [10]: M1 = [1 2; 3 4];
         M2 = [3 4; 5 6];

         M1 + M2
         M1 - M2
         M1 * M2

ans =

    4    6
    8   10

ans =
```

```
  -2  -2
  -2  -2

ans =

   13   16
   29   36
```

Dot operator is used for operations (adding, subtracting, multiplying) by components

```
In [11]: M1 .* M2

ans =

    3    8
   15   24
```

Dividing and multiplying by a scalar

```
In [12]: M3 = ones(2,3) * 6
         M4 = M1 / 3

M3 =

   6   6   6
   6   6   6

M4 =

   0.33333   0.66667
   1.00000   1.33333
```

Eigen values and eigen vectors

```
In [14]: eigvals = eig(M1)   % eigenvalues
         [vecs, vals] = eig(M1)   % eigenvectors (in columns) and eigenvalues
                                  % stored in a diagonal matrix M1
eigvals =

  -0.37228
   5.37228

vecs =
```

```
  -0.82456  -0.41597
   0.56577  -0.90938

vals =

Diagonal Matrix

  -0.37228         0
         0   5.37228
```

Invers matrix inv().

```
In [13]: iM1 = inv(M1)
         iM1 * M1     % check

iM1 =

  -2.00000   1.00000
   1.50000  -0.50000

ans =

   1.00000   0.00000
   0.00000   1.00000
```

1.3.2 Solution of equation systems

System of linear equations $Ax = b$

```
In [20]: A = [1 2 3; -1 0 2; 1 3 1]
         b = [1 0 0]'

A =

   1   2   3
  -1   0   2
   1   3   1

b =

   1
   0
   0
```

The inverse matrix can be used for the solution $x = A^{-1}b$. A preferable solution is to use the operator </code>, which is using the Gauss elimination.

```
In [21]: % inverse
         x = inv(A)*b

         % Gauss elimination
         x = A\b

         % check
         A*x-b
```

```
x =

   0.66667
  -0.33333
   0.33333

x =

   0.66667
  -0.33333
   0.33333

ans =

   0.0000e+00
   1.1102e-16
   5.5511e-17
```

1.3.3 Cycles and conditions

Cycles and conditions are the important commands for the program control. Cycles ensure to run parts of the script repeatedly, e.g., the stiffness matrix assembly for all elements).

We will use “for”, and “while” cycles. “for” performes the known number of iterations, “while” performs cycles until a prescribed condition.

Conditions branch out the program and run the individual parts of the code.
Conditions use logical operators or relation operators:

>, <, >=, <=, == (is equal), ~= (is not equal), || (or), && (and).

Common mistake:

Do not change the relation operator == for the assignment operator =

```
In [1]: % number of cycles
        n = 5;

        % allocation of empty vector of size n
        v = zeros(n, 1);

        for i = 1:n
            v(i) = 2*i;
        end

        % save transposition of vector v into v_transpose
        v_transpose = v'

        j = 1;
        sum_j = 0;
        while (j<10)
            sum_j  = sum_j + j;
            j = j*2;
        end

        sum_j
```

```
v_transpose =

    2    4    6    8   10

sum_j =  15
```

if (elseif) – if the condition is valid, then do following commands

```
In [2]: a = 1;
        b = 2;
        c = 3;
        d = 1;

        % Condition 1
        disp('C1:');
        if (a == 1)
            disp('a is equal 1'); % function disp('...') prints string
                                  % in the argument to the Command Window
        end

        % Condition 2
        disp('C2:');
        if (a > d)
            disp('a is greater than d');
        elseif (a < d)
            disp('a is less than d');
```

```
        else
            disp('a is equal to d');
        end

        % Condition 3
        disp('C3:');
        if (a ~= 1 || c <=3)
            disp('a is not equal to 1 OR c is less or equal to 3')
        end

        % Condition 4
        disp('C4:');
        if (a == 1 && c <=3)
            disp('a is equal to 1 AND c is less or equal to 3')
        end
```

```
C1:
a is equal 1
C2:
a is equal to d
C3:
a is not equal to 1 OR c is less or equal to 3
C4:
a is equal to 1 AND c is less or equal to 3
```

switch – analogy to if , suitable for more cases of the variable
(to avoid the multiple use of elseif)

```
In [17]: type = 'beam'; % string

         switch type
             case 'truss'
                 disp('Type of structure: truss')
             case 'beam'
                 disp('Type of structure: beam')
             otherwise
                 disp('Unknown type of structure.')
         end
```

```
Type of structure: beam
```

break, continue – end the actual cycle or actual iteration and continuing to the next iteration (continue)

```
In [18]: % break
         disp ('Use of break in a cycle.');
         i = 0;
```

```
while 1
    i = i+1;
    if i == 4
        break;
    end
    disp(i);
end

% continue
disp ('Use of continue in a cycle.');
i = 0;
while i < 5
    i = i+1;
    if i == 4
        continue;
    end
    fprintf('%d  ',i);    % another way how to print the results
end
```

```
Use of break in a cycle.
 1
 2
 3
Use of continue in a cycle.
1  2  3  5
```

1.3.4 Localization

In the solution by finite element method, we will need to organize (localize) individual stiffness matrices into the overall stiffness matrix of the structure. This process is called localization. In our tutorials, we will use the localization system with the help of the localization matrix (loc), where code numbers of degrees of freedom of elements are written onto each row

```
In [3]: loc =  [1 2 3 4;
                3 4 5 6;
                5 6 7 8]

        K = zeros(8,8);

        for i = 1:3
            k = ones(4,4) * i;    % element stiffness matrix
            K(loc(i,:),loc(i,:)) = K(loc(i,:),loc(i,:)) + k;% global
                                                            % stiffness matrix
        end
        K
```

```
loc =

   1   2   3   4
   3   4   5   6
   5   6   7   8

K =

   1   1   1   1   0   0   0   0
   1   1   1   1   0   0   0   0
   1   1   3   3   2   2   0   0
   1   1   3   3   2   2   0   0
   0   0   2   2   5   5   3   3
   0   0   2   2   5   5   3   3
   0   0   0   0   3   3   3   3
   0   0   0   0   3   3   3   3
```

1.3.5 Functions

Functions are useful for large programs and codes. The code is then more transparent. Its parts can be called repeatedly. Functions are stored into the individual files (*.m).

The names of the file and of the function must be the same!

The first line declares the function and comments usually follow on the next lines. These lines are suppose to be the help of the function and it can be displayed by "help and name_of_the_function".

```
In [22]: function [m1, m2] = magicNumbers(num1, num2)
         %
         % This function returns two "magic numbers" calculated
         % from two input numbers.

             m1 = floor((num1*10 - num2 * 3) / 6) - 3; % function floor()
                                                       % rounds down the argument
             m2 = ceil((num1*0.1 + num2 / 2) * 4) + 8;  % function ceil()
                                                        % rounds up the argument
         end
```

The function is then recalled from the script (or from the other function)

```
In [23]: [magic1, magic2] = magicNumbers(8, 5)

         a = -8;
         b = 0;
         [magic3, magic4] = magicNumbers(a, b)

magic1 =  7
magic2 =  22
magic3 = -17
magic4 =  5
```

1.3.6 Graphic output

Matlab quickly draws graphs with the command "plot." An arbitrary number of functions can be plotted into one graph. Command "hold on" ensures to keep the previous function in the graph.

Optional arguments are "color," "pattern" of points, and lines.

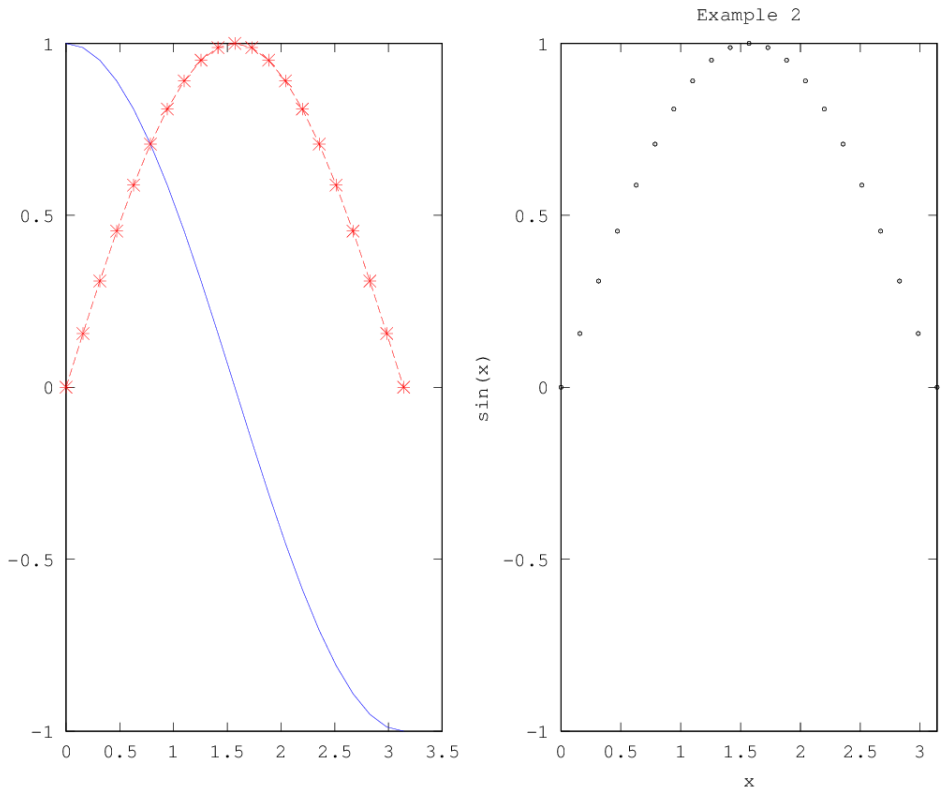
"Scatter" prints only points.

The graphic output is into separate windows (figures).
The new windows can be opened by command "figure." "close all" closes all windows.
The graph window can be split into sub-windows by "subplot."
"axis" sets up the values range . "xlabel," "ylabel" names axes.
The title of the graph is written by the command "title."

```
In [24]: subplot(1, 2, 1)
         x = [0:0.05:1] * pi;
         y = cos(x);
         plot(x,y);

         hold on;
         y = sin(x);
         plot(x,y, '--r*');

         subplot(1, 2, 2)
         scatter(x,y, 'k');
         axis([0 pi -1 1]);
         xlabel('x'); ylabel('sin(x)');
         title('Example 2')
```



There are other functions suitable for 2D (patch) and 3D (mesh, surf), which will be shown in the following seminars during the semester.

Reference: Czech course of “Numerická analýza konstrukcí” (Numerical analysis of structures) by B. Patzák (borek.patzak@fsv.cvut.cz)