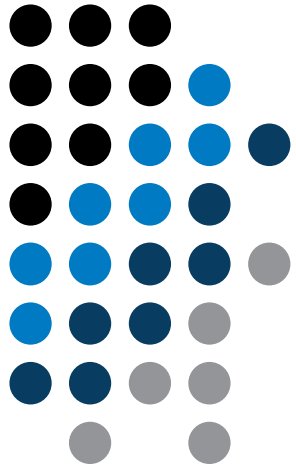
AE0B17MTB – Matlab

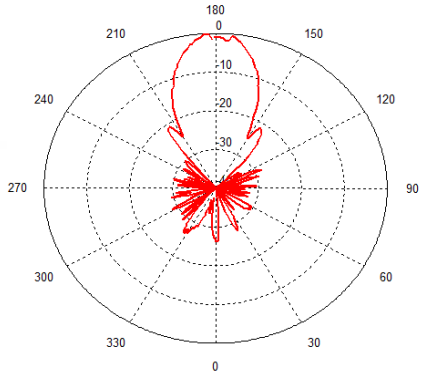
Part #12

Miloslav Čapek
miloslav.capek@fel.cvut.cz
Filip Kozák, Viktor Adler, Pavel Valtr

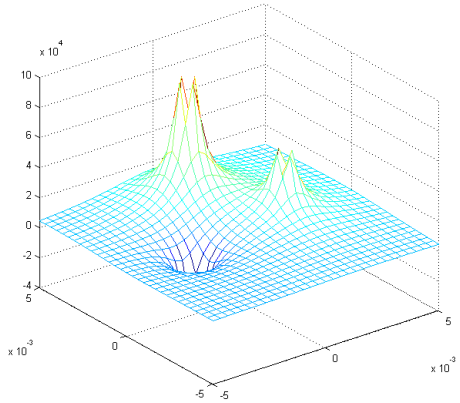
Department of Electromagnetic Field
B2-626, Prague



Learning how to …

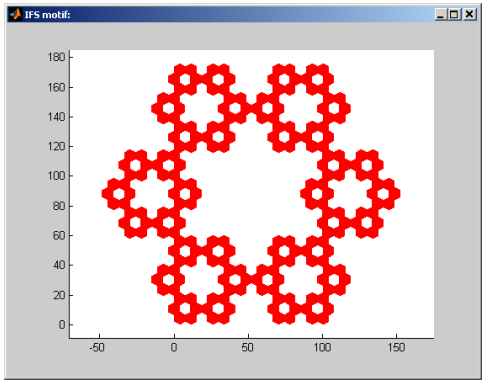


Exercise #1 – radiation pattern of an antenna



Exercise #2 – displaying field of point charges

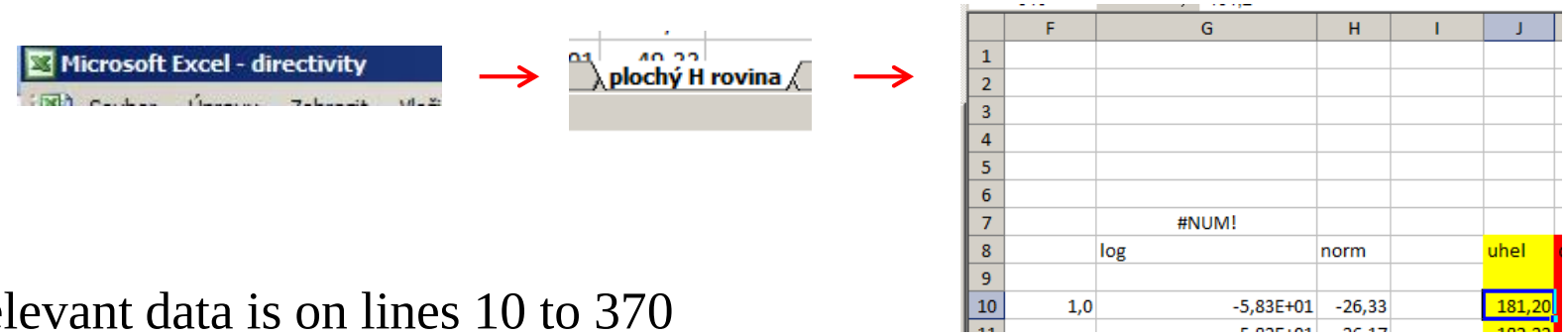
Exercise #3 – generation of IFS fractal structures



Exercise #1/01

- read data from file `'directivity.xls'`, sheet `'plochý H rovina'` columns `'uhel'` and `'dBm'`

```
>> num = xlsread('directivity.xls','plochý H rovina','J10:K370');
```



- relevant data is on lines 10 to 370
  → we should get 361 values

```
>> size(num) % = [361 2]
```

- store angle values in matrix U, measured values of power in D

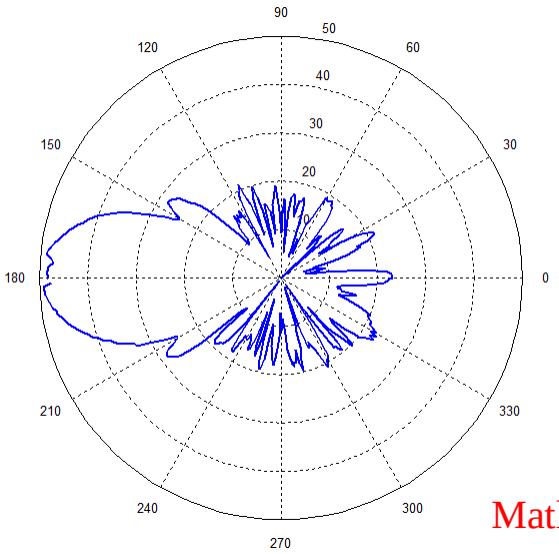
```
>> U = num(:,1);
>> D = num(:,2);
```

Exercise #1/02

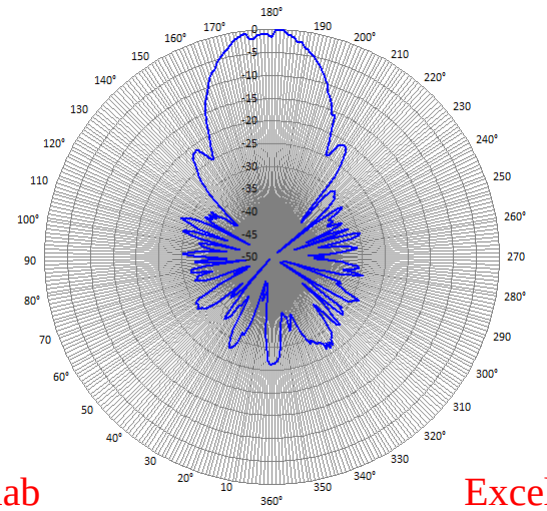
- in the next steps data can be processed arbitrarily
  - for instance Excel-like modification, with the same output:

```
>> Ur = pi*U/180;          % polar plots angular dependence (0:2*pi)
>> Dr = D + max(abs(D));   % polar plots values as
>>                         % a distance from origin (= positive number!)
>> figure; polar(Ur,Dr);   % plotting polar graph
```

as can be seen there exists different axis description in Matlab (wrong in the context of antenna technique → we possibly have to use another type of graph (A.P.P.)



Matlab



Excel

Exercise #1/03

- `polar` plot is still not quite suitable
  - we want the main lobe to be in the direction of *y*-axis
  - properly calibrated is the Excell graph axis

- options:
  - new function for polar plot (difficult, but not impossible)
  - we search for suitable alternative:

`http://www.mathworks.com/matlabcentral/fileexchange/`

  - type for instance „`polar plot`“ in browser
  - and arrange by rating (the best first)

Exercise #1/04

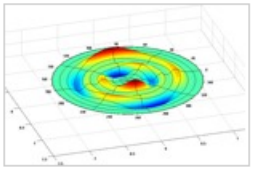
updated 4 years ago

Bullseye Polar Data Plot by Daniel Ennis

This function creates smooth patches of polar data in "bullseye" plot. (bullseye, polar, plotting)

fx [h,t]=bullseye(data,varargin);

5 Ratings
6 Comments
10 Downloads (30 Days)



updated 6 days ago

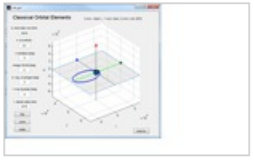
3D Polar Plot by Ken Garrard

Plots 3d polar data with polar axis and polar grid (polar, plot, specialized)

fx polarplot3d(Zp,varargin)

polarplot3d_demo.m

5 Ratings
10 Comments
141 Downloads (30 Days)



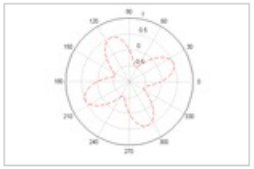
updated 1 year ago

Classical Orbital Elements GUI by Anders Edfors Vannevik

Visualize Classical Orbital Elements (aerospace, aeronautics, aerodef)

coe_gui(varargin)

3 Ratings
2 Comments
28 Downloads (30 Days)



updated 2 years ago

Polar 2 by Daniel Armyr

An update to Matlabs built-in polar.m (graph types, plotting, graphics)

fx polar2(varargin)

2 Ratings
12 Comments
63 Downloads (30 Days)

updated 2 years ago

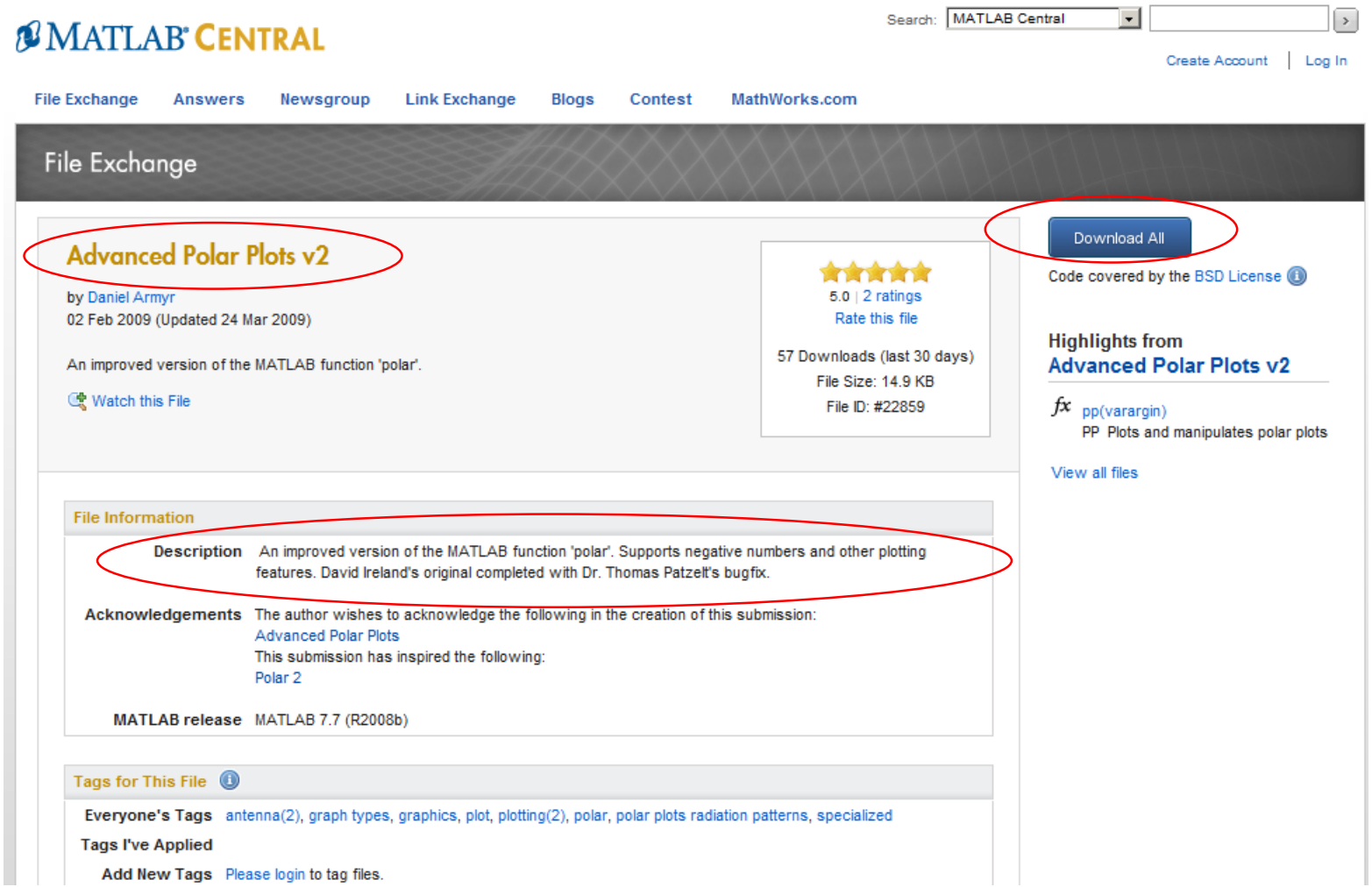
Advanced Polar Plots v2 by Daniel Armyr

An improved version of the MATLAB function 'polar'. (antenna, plotting, graphics)

fx pp(varargin)

2 Ratings
18 Comments
57 Downloads (30 Days)

Exercise #1/05

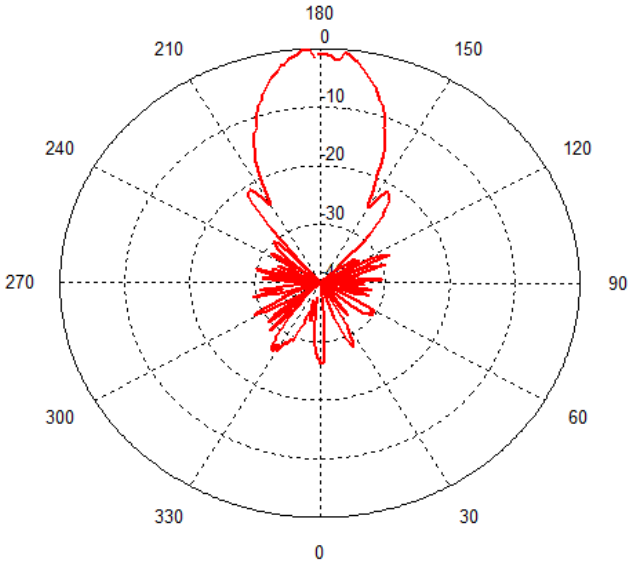


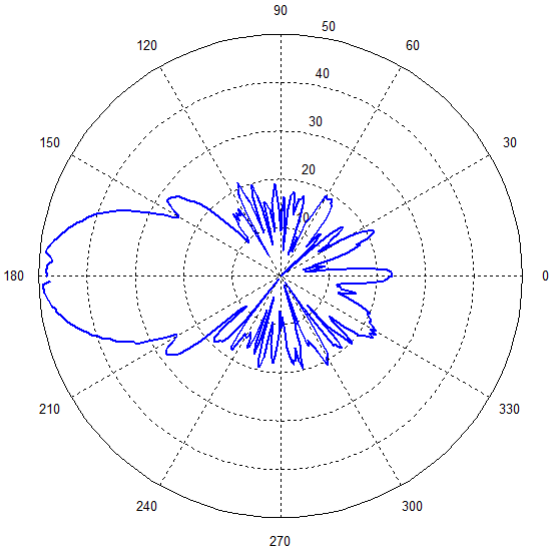
Exercise #1/06

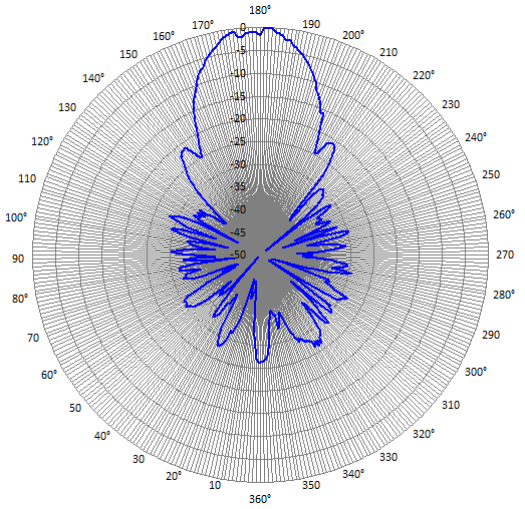
- download file
- browse help

```
>> help pp
```

```
>> pp(Ur,D,[-40 0],'MagMarkAngle',90,'ThetaStartAngle',270, ...
     'LineWidth',2,'LineColor','r','FigureBackgroundColor','w');
```

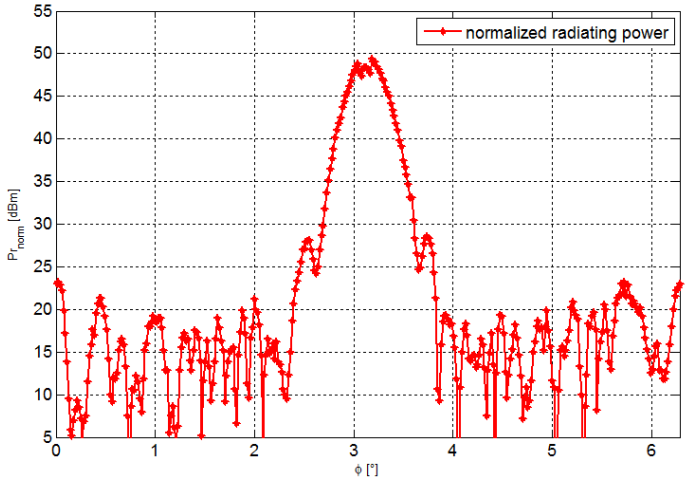






Exercise #1/07

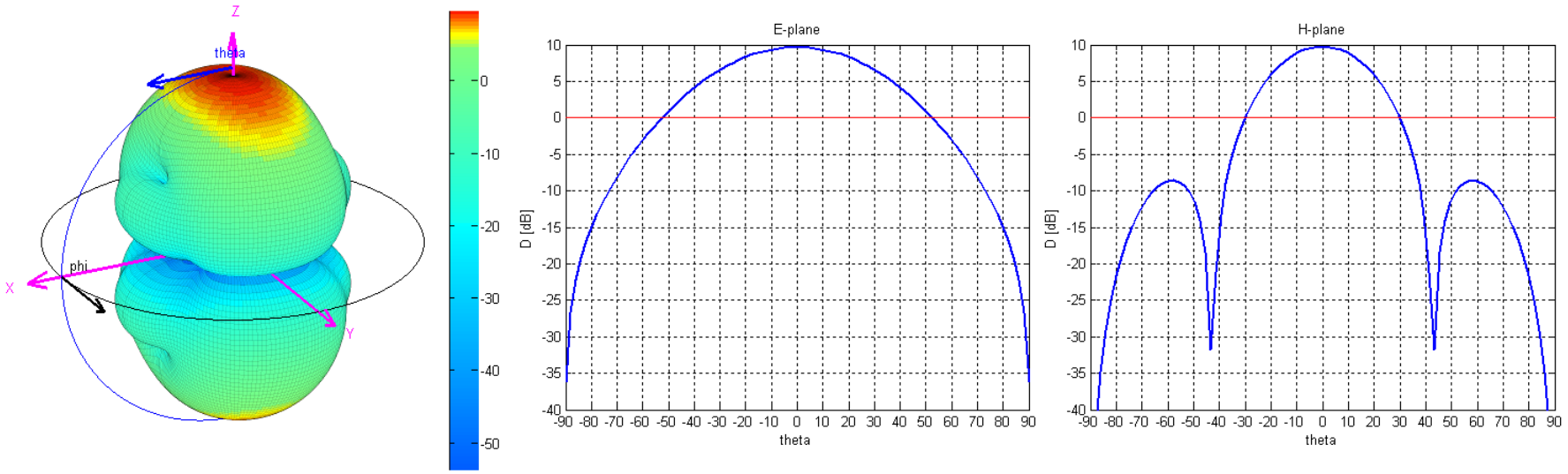
- we demonstrate advanced graphics modifications
- making use of 'classical' *x*-*y* plot




```
>> figure('Color','w','Position',[100 100 850 550],'MenuBar','none');
>> plot(Ur,PrdB,'*-r','LineWidth',2);
>> grid on;
>> xlim([0 2*pi]);
>> ylim([5 55]);
>> xlabel('\phi [°]');      % Matlab accepts TeX input as well
>> ylabel('Pr_{norm} [dBm]');
>> legend('normalized radiating power');
>> set(gca,'FontSize',13);
```

Exercise #1/08

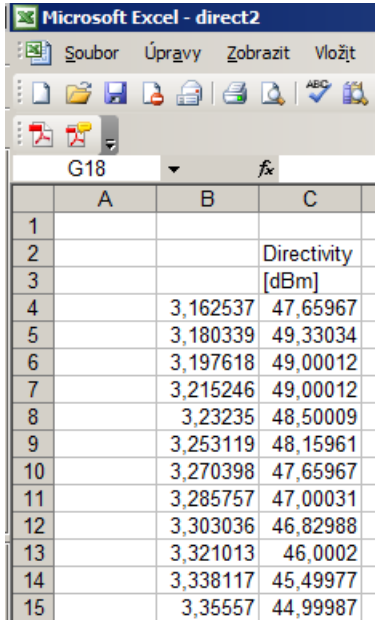
- our own solution developed at the Department of Electromagnetic Field:
  - (Jan Eichler)



Exercise #1/09

- the other way round, we can also write into (new as well) `xls` file
  - write selected data one by one

```
>> d = {'Directivity';'[dBm]'};
>> xlswrite('direct2.xls',d,2,'C2');
>> xlswrite('direct2.xls',Ur,2,'B4');
>> xlswrite('direct2.xls',Dr,2,'C4');
```



	A	B	C
1			
2			Directivity
3			[dBm]
4		3,162537	47,65967
5		3,180339	49,33034
6		3,197618	49,00012
7		3,215246	49,00012
8		3,23235	48,50009
9		3,253119	48,15961
10		3,270398	47,65967
11		3,285757	47,00031
12		3,303036	46,82988
13		3,321013	46,0002
14		3,338117	45,49977
15		3,35557	44,99987

Exercise #1/10

- use function `importdata` to read data
  - an alternative is to use previously mentioned function `uiimport`

```
>> TXT = importdata(...
         '39364,7134632176.txt');
```

file name (visible by Matlab)

```
>> TXT
```

```
>> TXT = importdata(...
         '39364,7134632176.txt');
>> TXT

TXT =

          data: [361x3 double]
      textdata: {3x3 cell}
    colheaders: {'uhel[°]'  'vykon[W]'  'faze[°]'}

>>
```

39364,7134632176 – Poznámkový blok

Soubor Úpravy Formát Zobrazení Nápověda

```
WinEman 2.4.1

uhel[°]          vykon[W]          faze[°]

0.59      1.469000000000000E-0009      0.000000000000000E+0000
1.21      1.524000000000000E-0009      0.000000000000000E+0000
2.21      1.413000000000000E-0009      0.000000000000000E+0000
3.39      1.211000000000000E-0009      0.000000000000000E+0000
4.39      7.079000000000000E-0010      0.000000000000000E+0000
5.37      3.828000000000000E-0010      0.000000000000000E+0000
6.37      1.778000000000000E-0010      0.000000000000000E+0000
7.41      6.561000000000000E-0011      0.000000000000000E+0000
8.24      2.818000000000000E-0011      0.000000000000000E+0000
9.22      2.415000000000000E-0011      0.000000000000000E+0000
10.27      3.690000000000000E-0011      0.000000000000000E+0000
11.27      4.819000000000000E-0011      0.000000000000000E+0000
12.27      6.310000000000000E-0011      0.000000000000000E+0000
13.26      5.212000000000000E-0011      0.000000000000000E+0000
14.26      3.828000000000000E-0011      0.000000000000000E+0000
15.24      2.239000000000000E-0011      0.000000000000000E+0000
16.30      3.548000000000000E-0011      0.000000000000000E+0000
17.36      4.140000000000000E-0011      0.000000000000000E+0000
18.25      1.040000000000000E-0010      0.000000000000000E+0000
19.23      2.075000000000000E-0010      0.000000000000000E+0000
20.25      2.818000000000000E-0010      0.000000000000000E+0000
21.29      4.295000000000000E-0010      0.000000000000000E+0000
22.27      3.690000000000000E-0010      0.000000000000000E+0000
23.30      6.561000000000000E-0010      0.000000000000000E+0000
24.28      8.570000000000000E-0010      0.000000000000000E+0000
25.27      1.000000000000000E-0009      0.000000000000000E+0000
26.25      1.000000000000000E-0009      0.000000000000000E+0000
27.28      7.943000000000000E-0010      0.000000000000000E+0000
28.32      6.067000000000000E-0010      0.000000000000000E+0000
29.30      4.295000000000000E-0010      0.000000000000000E+0000
30.34      1.919000000000000E-0010      0.000000000000000E+0000
31.32      7.362000000000000E-0011      0.000000000000000E+0000
32.34      6.067000000000000E-0011      0.000000000000000E+0000
33.32      1.167000000000000E-0010      0.000000000000000E+0000
```

Exercise #1/11

- data assignment followed by identical operations like in Excel

```
>> U    = TXT.data(:,1);
>> P    = TXT.data(:,2);
>> P(10:20)

>> Ur   = pi*U/180;
>> PdB  = 10*log10(P*1e3);
>> PrdB = PdB + max(abs(PdB));
>> figure; polar(Ur,PrdB);
```

Exercise #2/01

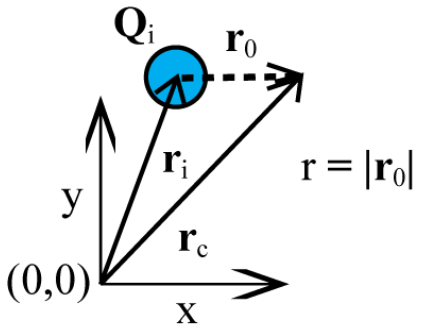
- create a script (function) to display electric field intensity of electric charges
  - it is necessary to find suitable equations to calculate intensity (of electromagnetic field)
  - consider physical background of the problem

- solution
- application of Coulomb's law for a point charge:

$$\mathbf{E}(r) = \frac{Q}{4\pi\varepsilon} \cdot \frac{1}{r^2} \cdot \mathbf{r_0} \qquad \text{where distance} \qquad r = |\mathbf{r_c} - \mathbf{r}_i|$$

- calculation using scalar potential is easier because

$$\mathbf{E} = -\nabla\varphi \qquad \varphi(r) = \frac{Q}{4\pi\varepsilon} \cdot \frac{1}{r} + K$$



Exercise #2/02

- for more (n) charges, use principle of superposition:

$$\mathbf{E}(\mathbf{r}) = \sum_{i=1}^{n} \mathbf{E}_i(\mathbf{r}) \qquad \text{and thus} \qquad \varphi(r) = \sum_{i=1}^{n} \frac{Q_i}{4\pi\varepsilon} \cdot \frac{1}{|\mathbf{r_c} - \mathbf{r}_i|}$$

- for two charges (Q_1, Q_2):

$$\varphi(r) = \frac{Q_1}{4\pi\varepsilon} \cdot \frac{1}{|\mathbf{r_c} - \mathbf{r}_1|} + \frac{Q_2}{4\pi\varepsilon} \cdot \frac{1}{|\mathbf{r_c} - \mathbf{r}_2|}$$

- for known positions of charges $[x_1,y_1]$ and $[x_2,y_2]$ placed in vacuum the potential can be written as:

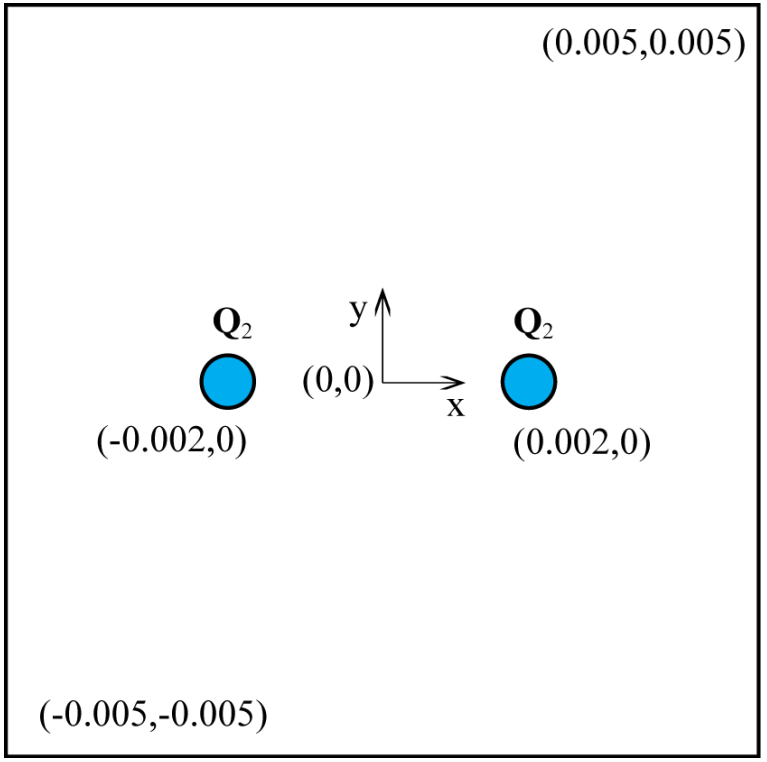
$$\boxed{\varphi(r) = \frac{1}{4\pi\varepsilon_0} \cdot \left(\frac{Q_1}{\sqrt{(x_c - x_1)^2 + (y_c - y_1)^2}} + \frac{Q_2}{\sqrt{(x_c - x_2)^2 + (y_c - y_2)^2}} \right)}$$

Cvičení #2/03

- what else do we not (and need to) know?
  - permittivity of vacuum: $\varepsilon_0 = 8.854 \cdot 10^{-12}\,\mathrm{Fm}^{-1}$
  - value of elementary charge: $e = 1.6022 \cdot 10^{-19}\,\mathrm{C}$
  -
  - (values of the charges concerned are kept equal to $Q_1 = +e$; $Q_2 = \pm e$)
  - position of charge 1: $x_1 = 2\,\mathrm{mm}$; $y_1 = 0\,\mathrm{mm}$;
  - position of charge 2: $x_2 = -2\,\mathrm{mm}$; $y_2 = 0\,\mathrm{mm}$;
  - grid size: $\langle -5, 5\rangle \times \langle -5, 5\rangle$ mm with step 1/3 mm

Exercise #2/04

- problem outline:



Note.: Because of dividing numbers that are very different in magnitude, it is necessary to pay attention to mantissa (omission of this fact can lead to uncontrolled spread of a numerical error in calculation).

$$\mathbf{E}(x_c, y_c) = -\frac{1}{4\pi\varepsilon_0} \cdot \nabla \left(\frac{Q_1}{\sqrt{(x_c - x_1)^2 + (y_c - y_1)^2}} + \frac{Q_2}{\sqrt{(x_c - x_2)^2 + (y_c - y_2)^2}} \right)$$

Exercise #2/05

- Matlab code (compare with previous exercise)
  - open new script
  - clear screen, definition of constants

```
clear, clc, close all;
x1 = 2e-3;  y1 = 0;
x2 = -2e-3; y2 = 0;

e0   = 1.6022e-19;
q1   = +e0;
q2   = -e0;
eps0 = 8.854e-12;
```

  - create grid for chart calculation

```
t = (-5:1/3:5).*1e-3;
[x,y] = meshgrid(t);
```

Exercise #2/06

- superposition principle:

```
z = q1./(eps0*4*pi*sqrt((x-x1).^2+(y-y1).^2)) + ... % pole 1. náboje
    q2./(eps0*4*pi*sqrt((x-x2).^2+(y-y2).^2));      % pole 2. náboje
```

- calculation of electric field intensity **E**

```
[gx gy] = gradient(z);
```

Exercise #2/07

- treatment of non-physical dependence in the vicinity of a point charge: notice that the resulting gradient is infinite which is caused by following singularity (the problem does not occur in reality)

```
gx(gx == +Inf) = NaN;
gy(gy == +Inf) = NaN;
gx(gx == -Inf) = NaN;
gy(gy == -Inf) = NaN;
```

$$\lim_{\substack{x_1 \to x_c \\ y_1 \to y_c}} \frac{1}{\sqrt{(x_c - x_1)^2 + (y_c - y_1)^2}} \approx \frac{1}{0} \to \infty$$

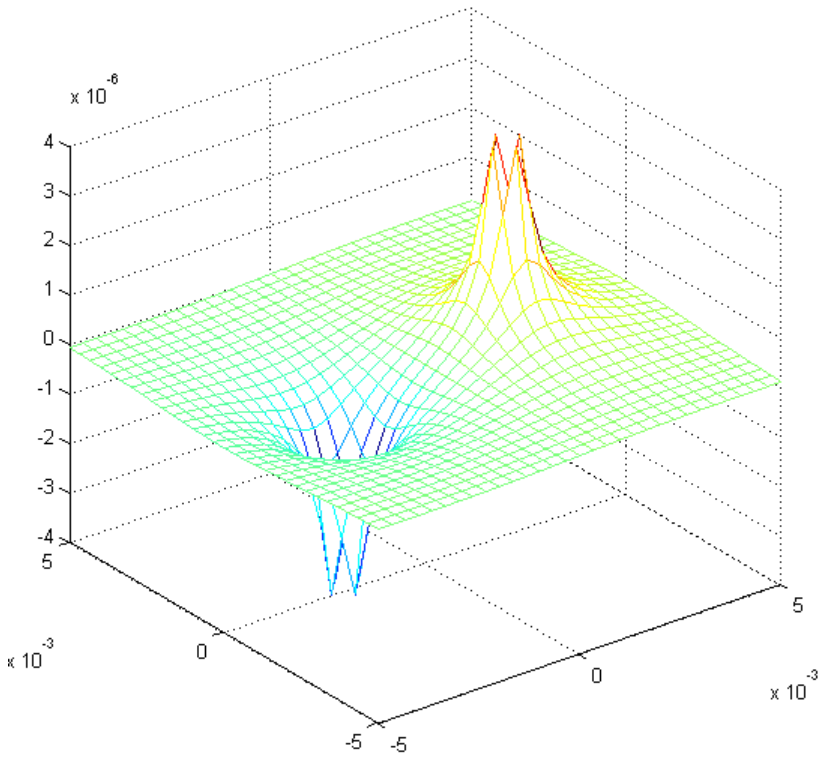
- plotting the potential and electric field intensity:

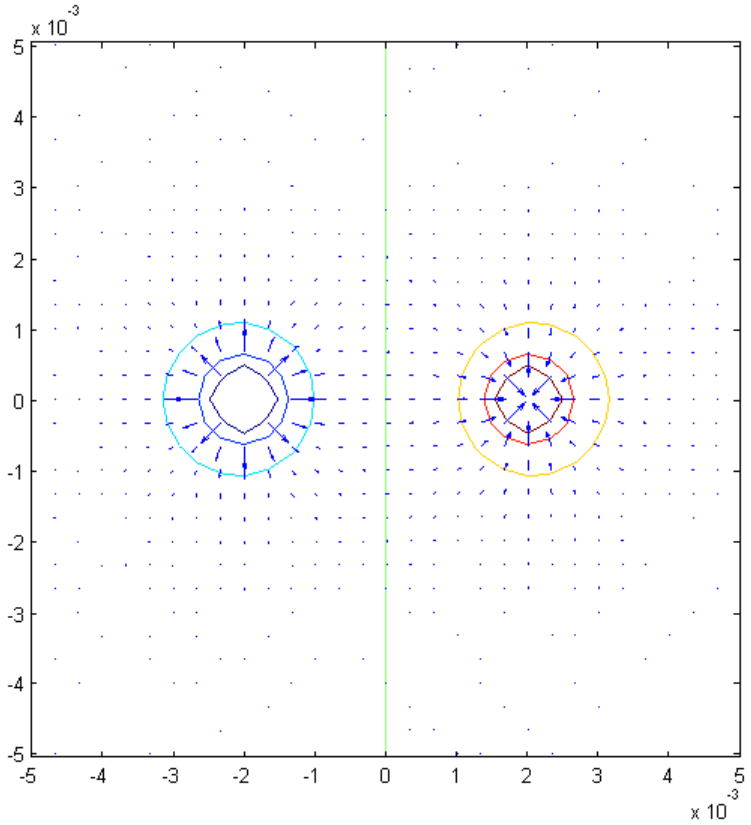
```
figure('pos',[50 50 1300 700]);
subplot(1,2,1);
mesh(x,y,z);
```

```
subplot(1,2,2);
contour(x,y,z);
hold on;
quiver(t,t,gx,gy);
```

Exercise #2/08

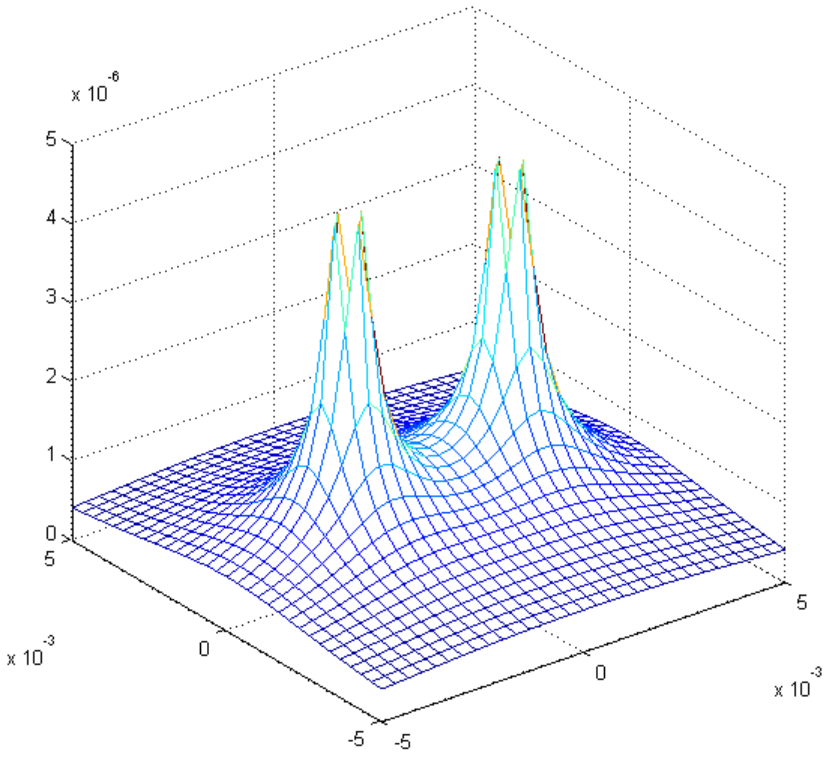
- opposite polarity of charges:

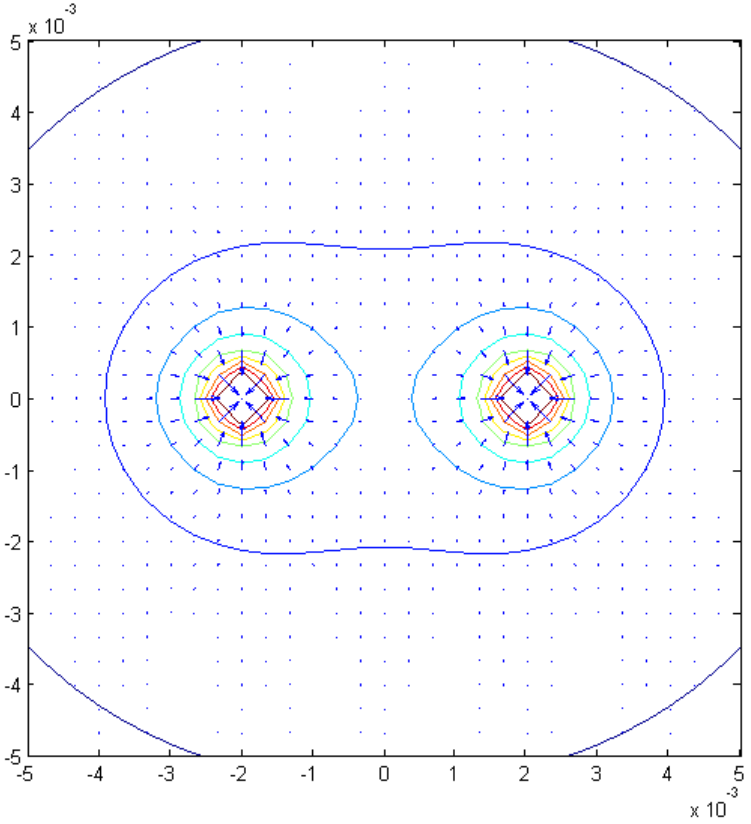




Exercise #2/09

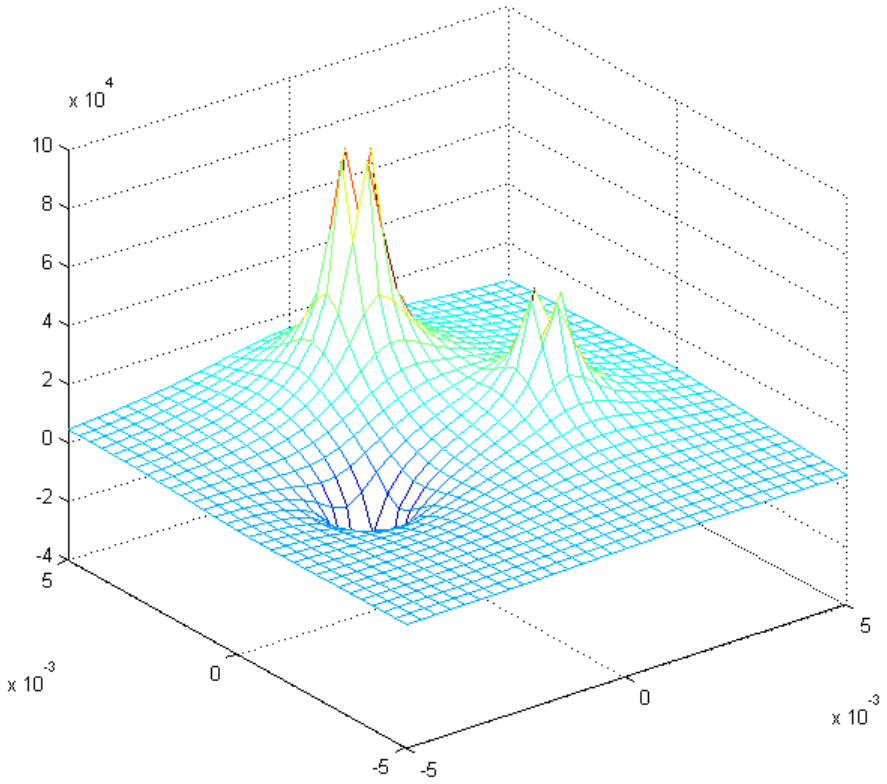
- identical polarity of charges :

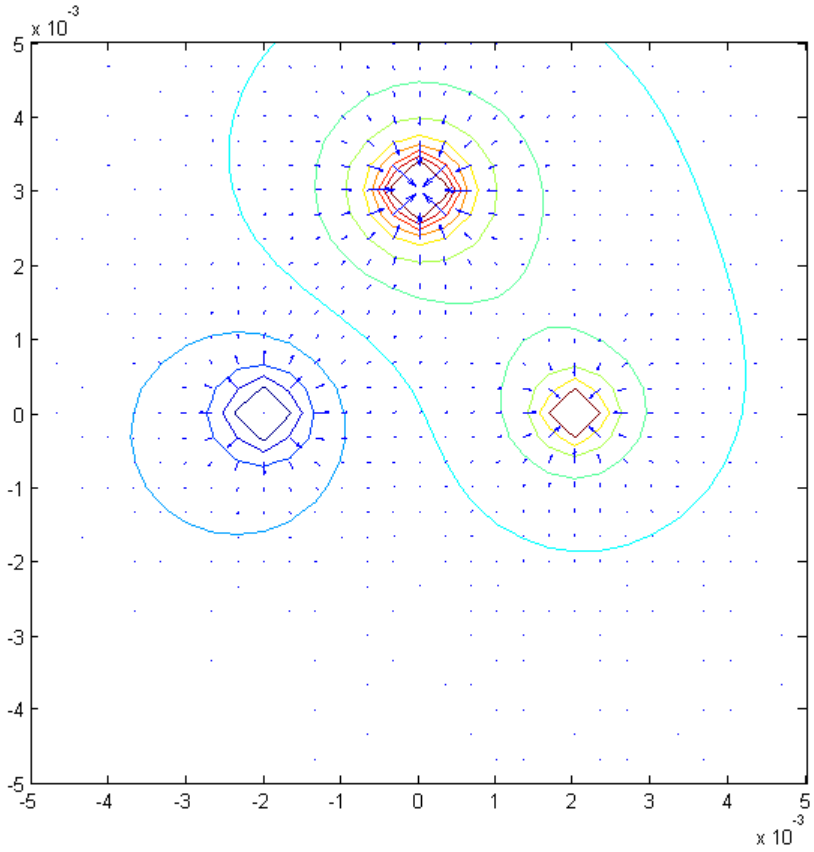




Exercise #2/10

- and finally the case of three charges (of different values) :





Exercise #3/01

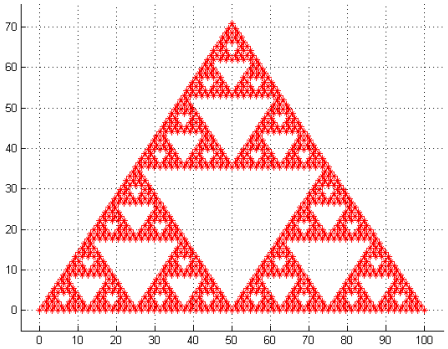
- create a IFS fractal generator
  - search internet for information on IFS fractals
  - use sets of points and transformation as an input for efficient ganaration

  - the whole problem is complex an unfeasible at first sight, let's try to split it into individual (meaning doable) phases!
    - what are your suggestions??

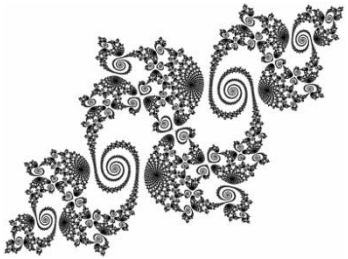
Exercise #3/02

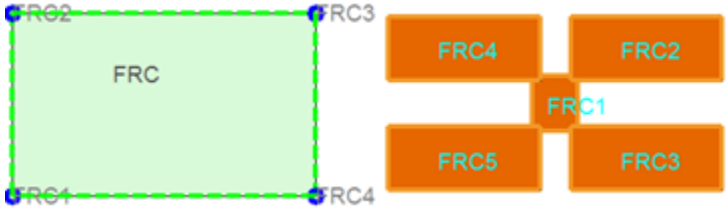
- 1) what is our goal – theoretical analysis
  - find out the meaning of following terms: fractal, polygon, iteration
  - what is IFS (*Iterated Function System*)
  - how to transform formal notation of IFS generation in the language of Matlab
- 2) what the code should look like
  - more separate parts?
  - what is the input and output data format?
  - what should function headers look like?
- 3) design and implementation
  - from simple to more complex
  - treatment of inputs
  - functions testing (functionality, speed, improper inputs)
- 4) further improvement of basic version

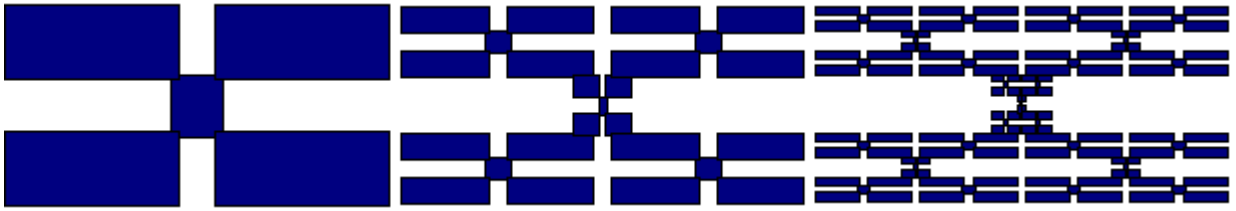
Exercise #3/03



- 1) our goal
  - find out what a fractal is (wiki, internet)
  - what is IFS?







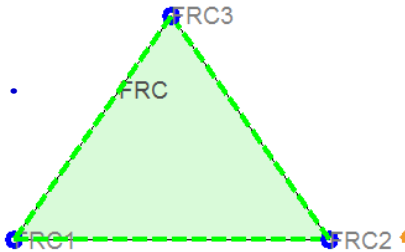
  - for the choice of notation see :
    `http://www.elmag.org/doku.php/wiki:user:capek:ifsmaker`

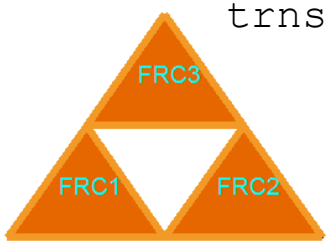
Exercise #3/04

- format of input variables (polygon, transformation):

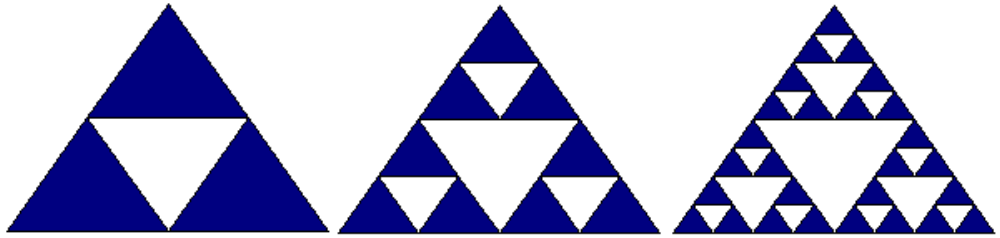
Variable	Size	Description
`pts`	(*n*,2)	array of points of the base shape (*n* is number of points)
`trns`	(*m*,6)	array affine transformations (*m* is number of transformations)
`iter`	(1,1)	number of iterations (positive integer)

```
pts = [0 0; ...
       100 0; ...
       50 √2*50]
```






```
trns = [0.5 0 0 0.5 0 0; ...
        0.5 0 0 0.5 50 0; ...
        0.5 0 0 0.5 25 √2*25]
```



`iter = 1;` `iter = 2;` `iter = 3;`

Exercise #3/05

- 2) what should the code look like?
  - more separate parts
    - part (A) will generate the fractal, the result will be a set of polygons
    - part (B) will draw the fractal
  - what is the input and output data format?
    - part (A) requires a set of input points (initial polygon), a set of affine transformations and number of iterations (variables `pts`, `trns`, `iter`)
    - Output of (A) will be 3D array of polygons, representing the resulting fractal collage, marked as `IFSfractal`
    - input of (B) will be the output of (A), i.e. `IFSfractal`
    - output of (B) will be handle to a figure in which the fractal will be drawn

Exercise #3/06

- 2) what will the intended code look like?
  - what will the function headers look like, shall the input be treated?

```
function IFSfractal = gen_ifs_fractal(pts,trns,iter)
%%GEN_IFS_FRACTAL: Generates iterated-function-system (IFS) motif
%
% code for IFS collage generation
```

```
function hndl = draw_fractal(IFSfractal)
%%DRAW_FRACTAL: Plots IFS collage (3D polygon array)
%
% code to draw generated IFS collage
```

Exercise #3/07

FUNCTION (A)

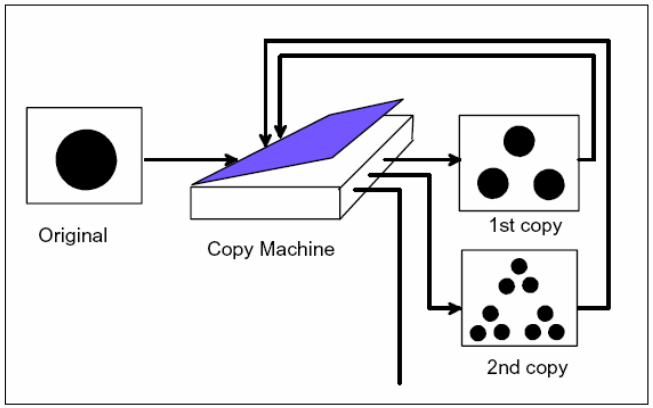
```
IFSfractal = gen_ifs_fractal(pts,trns,iter)
```

```
function IFSfractal = gen_ifs_fractal(pts,trns,iter)
%%GEN_IFS_FRACTAL: Generates iterated-function-system (IFS) motif
%
% code for IFS collage generation
```

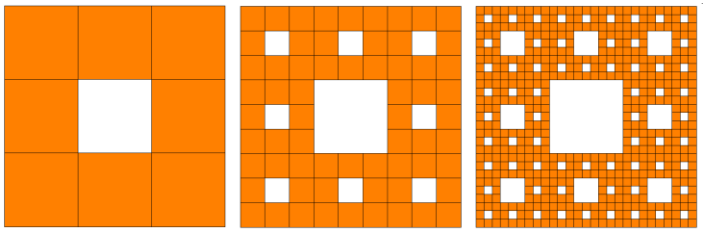
Exercise #3/08

- 3) draft and implementation of code: **IFS generation**
  - from the simple to more complex:
    PSEUDOCODE:

```
(01) for_all_iteration
(02)     for_each_polygon
(03)         for_each_point
(04)             generate_new_point
(05)         end
(06)     end
(07) end
```



$$w : \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$



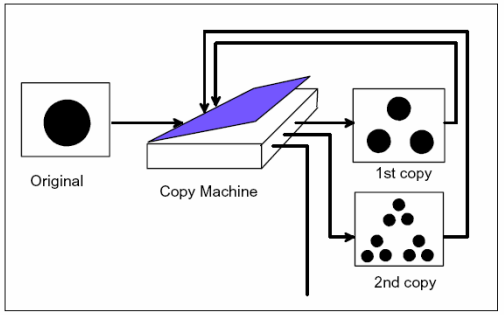
Exercise #3/09

- 3) draft and implementation of code: **IFS generation**

  IMPROVED PSEUDOCODE (analysis):

```
(01) inicialization IFSfractal
(02) for_all_iterations
(03)     select_polygons_of_the_last_iteration
(04)     for_each_polygon_of_the_last_iter
(05)         for_each_transformation
(06)             for_each_point_of_polygon
(07)                 generate_new_point
(08)                 /new_point/
(09)             end
(10)         end
(11)         /new_polygon/
(12)     end
(13)     /new_iteration/
(14)     store_it_at_the_end
(15) end
```

$$w : \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$



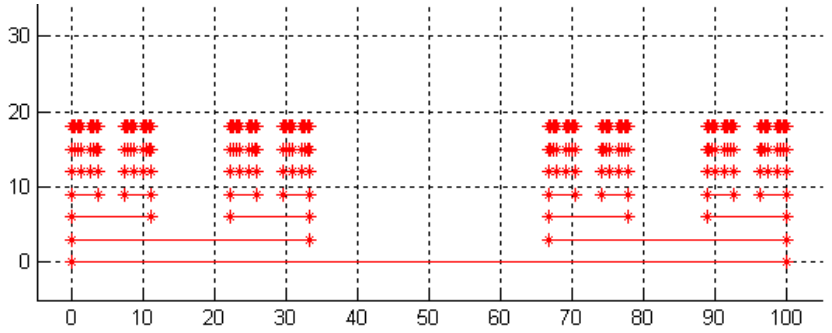
Exercise #3/10

- 3) draft and code implementation: **IFS generation**
- conclusions:
  - apart from `IFSfractal` we need one temporary variable to successively store (and retrieve) polygons of the iteration concerned
  - size of this variable must be initiated prior to beginning of generation and all polygons will be stored in it
  - this can be easily realized using cell data type; the respective variable will be denoted `Cell`
  - the last iteration will be stored in `IFSfractal` and in this way the required collage will be obtained

- the question is: how to outsmart `cell`?

Exercise #3/11

- 3) draft and code implementation: **IFS generation**
- now it is suitable to go through the code and use a simple case to verify if the code works properly
  - we choose Cantor's cloud, for instance (discovered by B. Mandelbrot; describes, among others, distribution of interference on a telephone line)
  - base object: 2 points (1 line), therefore:
    ```
    pts = [0 0; 100 0]
    ```
  - transformation: size reduction to 1/3 + shift, 2×6 coefficients, therefore:
    ```
    trns = [1/3 0 0 0 0 3; 1/3 0 0 0 200/3 3]
    ```
  - iteration:
    - `iter = 5`



Cvičení #3/12

```
Cell =

    [2x2   double]
    [2x2x2 double]
    []
    []
    []
```

- after the code execution and termination the variable `Cell` should fill-up like shown on left (5 iterations, `iter = 5`)

```
Cell =

    [2x2   double]
    [2x2x2 double]
    [2x2x4 double]
    []
    []
```

- `IFSfractal` then contains all polygons of tle last iteration, i.e.

```
IFSfractal = Cell{end};
```

```
>> size(IFSfractal)

ans =

     2     2    16
```

```
Cell =

    [2x2   double]
    [2x2x2 double]
    [2x2x4 double]
    [2x2x8 double]
    []
```

- the first item of variable `Cell` is the initial polygon (0-th iteration), serves for generation of the 1st iteration
  - its size is `iter+1` (all iterations + base object)

```
Cell =

    [2x2    double]
    [2x2x2  double]
    [2x2x4  double]
    [2x2x8  double]
    [2x2x16 double]
```

```
Cell    = cell(iter+1,1);
Cell{1} = pts;
```

Exercise #3/13

- 3) draft and code implementation: **IFS generation**

  CODE SEGMENTS IN MATLAB:

```
for n = 1:iter                                  % iteration-by-iteration
    thisIter  = Cell{n};                        % polygons from last iteration
    newPolygs = zeros(size(thisIter,1),2,size(thisIter,3)*trnsSize); % inicializing
    for m = 1:size(thisIter,3)                  % to be applied to all polygons
        thisPolyg(:,:) = thisIter(:,:,m);       % one-by-one
        for cur_tr = 1:trnsSize                 % for all transformations
            for cur_pt = 1:size(thisPolyg,1) % all points of polygon
                newPolygs(cur_pt,1,(m-1)*trnsSize+cur_tr) = ...
                    thisPolyg(cur_pt,1,:)*trns(cur_tr,1) + ...
                    thisPolyg(cur_pt,2,:)*trns(cur_tr,2) + trns(cur_tr,5);
                newPolygs(cur_pt,2,(m-1)*trnsSize+cur_tr) = ...
                    thisPolyg(cur_pt,1,:)*trns(cur_tr,3) + ...
                    thisPolyg(cur_pt,2,:)*trns(cur_tr,4) + trns(cur_tr,6);
            end                                 % matrix-wise solution also possible!
        end
    end
    Cell{1+n} = newPolygs(:,:,:);               % save all calculated iteration
end
IFSfractal = Cell{end};                         % of interest (IN THIS CASE!) is just
                                                  the last iteration
```

Exercise #3/14

- 3) draft and code implementation: **IFS generation**
  - add allocation of `Cell` structure and `trnsSize`
  - add header
  - add function's help
  - function: `gen_ifs_fractal.m`
  - treatment of inputs
    - treatment in compliance with IFS definition
  - fuction test
    - test of functionality using Cantor's cloud, 1st iteration

Exercise #3/15

FUNKCE (B)

```
hndl = draw_fractal(IFSfractal)
```

```
function draw_fractal(IFSfractal)
%%DRAW_FRACTAL: Plots IFS colage (3D polygon array)
%
% code for drawing generated IFS collage
```

Exercise #3/16

- 3) draft and code implementation: **IFS generation**
  PSEUDOCODE:

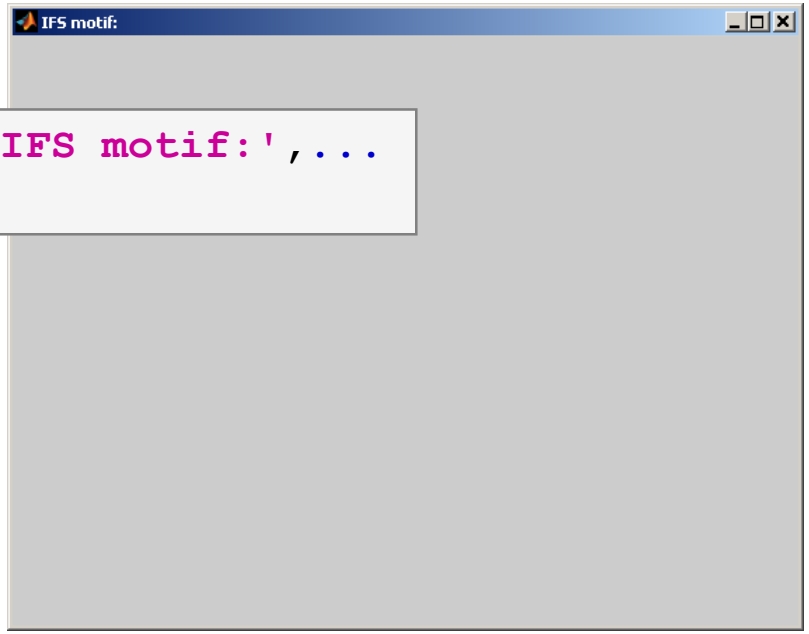
```
function draw_fractal(IFSfractal)
%%DRAW_FRACTAL: Plots IFS colage (3D polygon array)
%
% code for drawing generated IFS collage
```

```
(01) create figure
(02) axis
(03) hold on
(04) for_all_polygons
(05)     for_all_points
(06)         draw_line
(07)     end
(08) end
```

Exercise #3/17

- 3) draft and code implementation: **drawing IFS**
  - ensure menu is not displayed
  - ensure figure number is not displayed
  - figure label is „`IFS motif:`“

```
hndl = figure('Units','Pixels','Name','IFS motif:',...
    'NumberTitle','off','Menu','none');
```



Exercise #3/18

- 3) draft and code implementation: **drawing IFS**
  - we set the range of individual axes - could be skipped but in that case the axis are scaled!
    - that is why we fix the axis at the beginning
  - we start from known maximum size of polygons

```
A = min(min(IFSfractal(:,1,:)));
B = max(max(IFSfractal(:,1,:)));
C = min(min(IFSfractal(:,2,:)));
D = max(max(IFSfractal(:,2,:)));
```

  - and we set the axis range accordingly

```
axis([A(1)-0.05*B(1) B(1)+0.05*B(1) C(1)-0.05*D(1) D(1)+0.05*D(1)]);
```

Exercise #3/19

- 3) draft and code implementation: **drawing IFS**
  - on top of that we set the background color and place axis within a box
  - apply hold on (to fix canvas)

```
hold on
box on
```

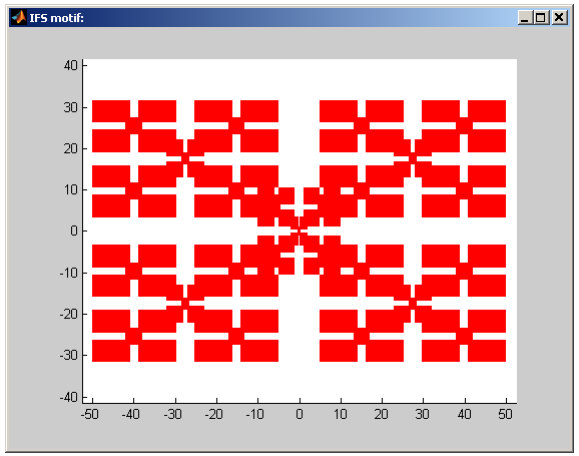
  - all polygons will be drawn iteration-by-iteration using function `patch`

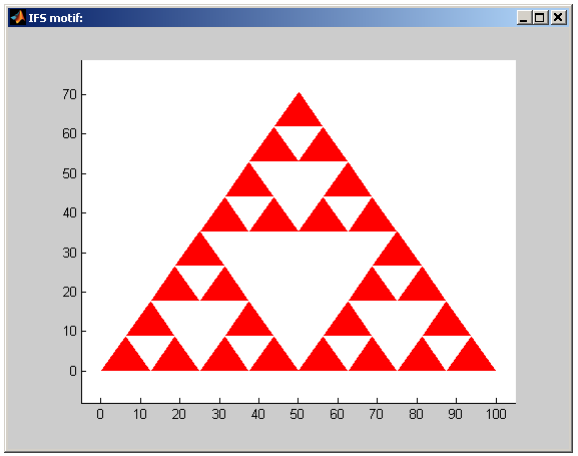
```
for o = 1:size(IFSfractal,3)
    X = IFSfractal(:, 1, o);
    Y = IFSfractal(:, 2, o);
    patch(X, Y, 'r', 'EdgeColor', 'none');
end
```

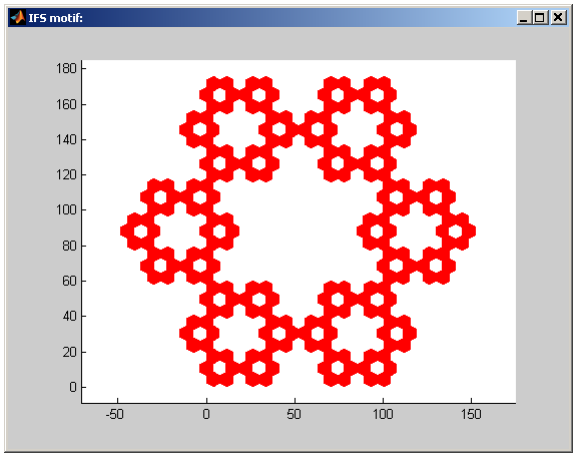
Exercise #3/20

- both functions are now available
  - (A): `IFSfractal = gen_ifs_fractal(pts,trns,iter)`
  - (B): `draw_fractal(IFSfractal)`

  - test using own fractals
    - prepared data of three collages are: `IFS1.m`, `IFS2.m` a `IFS3.m`







Exercise #3/21

- the appearance of the figure can be further modified
  - do you know, how to get screen resolution?
  - set the size of figure window to 0.3×0.4 multiple of screen resolution and place it 0.05 multiple of screen resolution from bottom left corner
  - choose your own window color

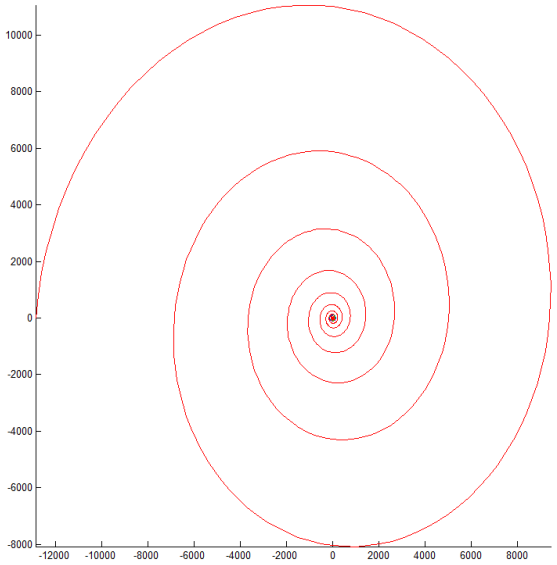
```
monitor = get(0,'Screensize');
hndl = figure('Units','Pixels','Name','IFS motif:',...
    'Position',[0.025*monitor(3) 0.05*monitor(4) 0.3*monitor(3)...
    0.4*monitor(4)],'Color',[0.9 0.9 0.9],...
    'NumberTitle','off','Menu','none');
```

Exercise #3/22

- other tasks:
  - try to treat input for entering larger/smaller matrices, strings etc.
  - try to increase the speed of algorithm (mainly the drawing part)
    - hint: using `profile` tool find out what takes most of time, do you need given parts?
  - think over whether it is not more user friendly to use just one structure for entire description of IFS collage
    - hint: have a look at data type `struct` (`>> doc struct`), what does it offer?

Exercise #4/01

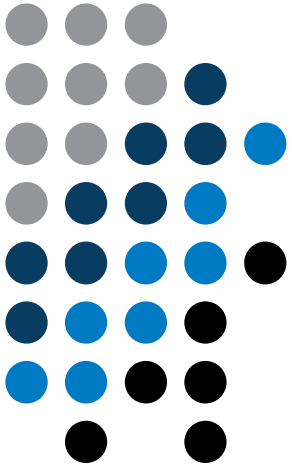
- using function `comet` display twisted logarithmic spiral




```
alpha   = 25*pi:-pi/50:0; % axis (angles)
a       = 5; % coefficients
b       = 0.1;
r       = a*exp(b*alpha);
[x,y,z] = sph2cart(alpha,0,r);

figure('Pos',[25 25 850 850],'Color','w');
comet3(x,y,z);
```

Thank you!



ver. 5.1 (10/05/2016)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz