A0B17MTB – Matlab

Part #6

Miloslav Čapek

miloslav.capek@fel.cvut.cz

Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field
B2-626, Prague

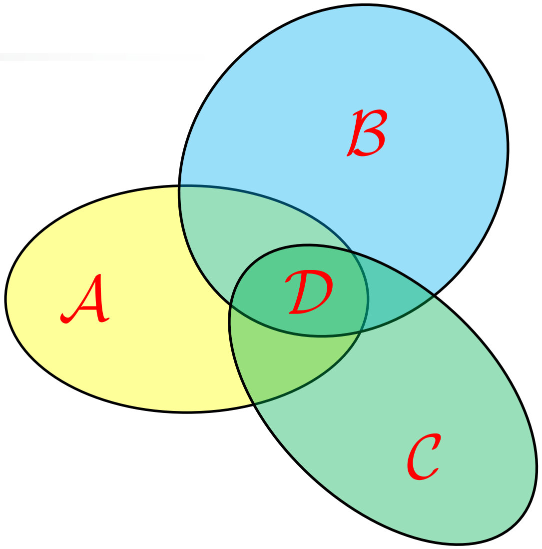
Learning how to …

Set operations

Sorting

Searching

Functions #1



$$\mathcal{D} = \mathcal{A} \bigcap \mathcal{B} \bigcap \mathcal{C}$$

$$\mathcal{A} \bigcap \mathcal{B} = \left\{ x : x \in \mathcal{A} \wedge x \in B \right\}$$

Set operations

- there exist following operations (operators) in Matlab applicable to arrays or individual elements
  - arithmetic (part #1)
  - relational (part #3)
  - logical (part #3)
  - set (part #4)
  - bit-wise (help, >> `doc`)

- set operations are applicable to vectors matrices, arrays, cells, strings and tables
  - mutual sizes of these structures are usually not important

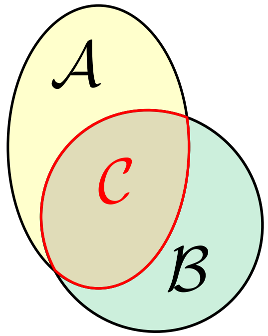
intersection of two sets	`intersect`
union of two sets	`union`
difference of two sets	`setdiff`
exclusive OR of two sets	`setxor`
unique values in a set	`unique`
sorting, row sorting	`sort`, `sortrows`
is the element member of a set?	`ismember`
is the set sorted?	`issorted`

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Set operations #1

- intersection of sets: `intersect`
  - example: intersection of a matrix and a vector:

```
>> A = [1 -1; 3 4; 0 2];
>> b = [0 3 -1 5 7];
>> c = intersect(A, b)
% c = [-1; 0; 3]
```

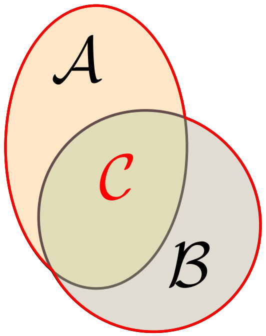


$$\mathcal{C} = \mathcal{A} \cap \mathcal{B}$$

intersect
union
setdiff
setxor
unique
sort, sortrows
ismember
issorted

- union of sets: `union`
  - all set operations can be carried out row-wise (in that case the number of columns has to be observed)

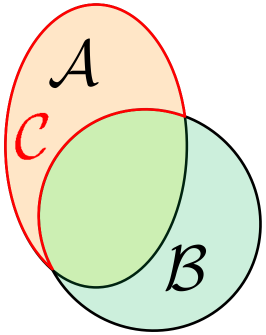
```
>> A = [1 2 3; 4 5 1; 1 7 1];
>> b = [4 5 1];
>> C = union(A, b, 'rows')
% C = [1 2 3; 1 7 1; 4 5 1]
```



$$\mathcal{C} = \mathcal{A} \cup \mathcal{B}$$

Set operations #2

- intersection of a set and complement of another set : `setdiff`
  - all set operations return more than one output parameter - we get the elements as well as the indexes

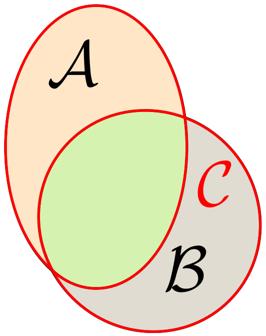


$$\mathcal{C} = \mathcal{A} \cap \mathcal{B}^{C} = \mathcal{A} \setminus \mathcal{B}$$

```
>> A = [1 1; 3 NaN];
>> B = [2 3; 0 1];
>> [C, ai] = setdiff(A,B)
% C = NaN, ai = 4
% i.e.: C = A(ai)
```

- exclusive intersection (XOR): `setxor`
  - all set operations can be carried out either as '*stable*' (not changing the order of elements) or as '*sorted*' (elements are sorted)

```
>> a = [5 1 0 4];
>> b = [1 3 5];
>> [C, ia, ib] = setxor(a, b, 'stable')
% C = [0 4 3], ia = [3; 4], ib = [2]
```



$$\mathcal{C} = \mathcal{A} \oplus \mathcal{B}$$

intersect
union
setdiff
setxor
unique
sort, sortrows
ismember
issorted

Set operations #3

$$\begin{pmatrix} c & b & a & c \\ a & c & b & a \\ c & c & d & b \end{pmatrix} \longrightarrow \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

- selection of unique elements of an array : `unique`
  - set operations are also applicable to arrays not (exclusively) containing numbers

```
>> A = {'Joe', 'Tom', 'Sam'};
>> B = {'Tom', 'John', 'Karl', 'Joe'};
>> C = unique([A B])
% C = {'John', 'Karl', 'Joe', 'Sam', 'Tom'}
```

- it is possible to combine all above mentioned techniques
  - e.g. row-wise listing of unique elements of a matrix including indexes :

```
>> D = round(rand(10, 3)).*repmat(mod((10:-1:1), 3)', [1 3])
>> [C, ai, bi] = unique(sum(D,2), 'rows', 'stable')
```

  - Interpret the meaning of the above code? Is the `'rows'` parameter necessary?

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Set operations #1

600 s ↑

- consider three vectors **a**, **b**, **c** containing natural numbers $x \in \mathbb{N}$ so that
  - vector **a** contains all primes up to (and including) 1000
  - vector **b** contains all even numbers up to (and including) 1000
  - vector **c** is complement of **b** in the same interval

- find vector **v** so that $\mathbf{v} = \mathbf{a} \cap (\mathbf{b} + \mathbf{c}),\quad \mathbf{b} + \mathbf{c} \equiv \{b_i + c_i\},\quad i \in \{1, 500\}$
  - what elements does **v** contain? $b_{i-1} < b_i < b_{i+1} \;\wedge\; c_{i-1} < c_i < c_{i+1},\; \forall i$

- how many elements are there in **v**?

```
v =

  Columns 1 through 24

     3     7    11    19    23    31    43    47    59    67    71    79

  Columns 25 through 48

   211   223   227   239   251   263   271   283   307   311   331   347

  Columns 49 through 72

   491   499   503   523   547   563   571   587   599   607   619   631

  Columns 73 through 87

   823   827   839   859   863   883   887   907   911   919   947   967
```

Set operations #2

500 s

- estimate the result of following operation (and verify using Matlab):

$$\mathbf{w} = \left(\mathbf{b} \cup \mathbf{c}\right) \setminus \mathbf{a}$$

  - what is specific about elements of the resulting vector **w**?

- with the help of logical indexing and mathematical functions determine how many elements of **w** are divisible by 3

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Set operations #3

500 s

- write previous exercise as a script:

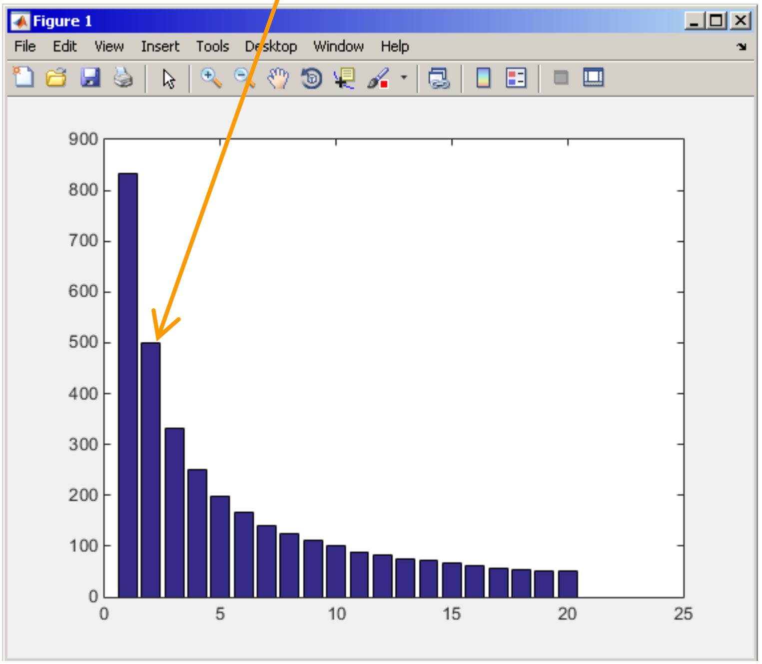
```
%% script depicts number of integers from 1 to 1000 in %
dependence on division remainders
clear; clc;

a = primes(1e3);
b = 2:2:1e3;
c = setdiff(1:1000, b);
w = setdiff(union(b, c), a);
% ...
    m = sum(not(mod(w, 3)));
% ...
```

- modify the script in the way to calculate how many elements of **w** are divisible by numbers 1 to 20
  - use for instance `for` loop to get the result
  - plot the results using `bar` function

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Set operations #4

for instance the amount of numbers in the interval from 1 to 1000 that are divisible by 2 and are not primes is 499



Set opeartions #5

600 s ↑

- Radio relay link operates at frequency of 80 GHz at 20 km distance with 64-QAM modulation
  - phase stability of ±0.5° is required for sufficiently low bit error rate without using synchronization and coding
  - that corresponds to the change of distance between antennas equal to ±5 μm
  - the statistics of link distance with normal distribution containing $1 \cdot 10^6$ elements can be generated as:

```
L = 20e3; % length of path
deviation = 5e-6; % standard deviation
N = 1e6; % number of trials
% random distances
distances = L + randn(1, N)*deviation;
```

- How many times is the distance `L` contained in the vector `distances`?
- How many unique elements are there in `distances`?
- Can the distribution be considered continuous?

Array sorting #1

- sort array elements
  - column-wise, in ascending order:

```
>> sort(A)
```

  - row-wise, in ascending order :

```
>> sort(A, 2)
```

  - in descending order:

```
>> sort(A, 'descend')
```

  - in descending order, row-wise:

```
>> sort(A, 2, 'descend')
```

- apply the sorting function, to following matrices (for instance):

```
>> A = reshape([magic(3)  magic(3)'], [3 3 2])
>> B = 'for that purpose';
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Array sorting #2

- function `sortrows` sorts rows of a matrix
  - elements of the rows are not swapped - rows are sorted as blocks

$$\begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

SORT:

$$\begin{pmatrix} 3 & 1 & 2 \\ 4 & 5 & 6 \\ 8 & 9 & 7 \end{pmatrix}$$

SORTROWS:

$$\begin{pmatrix} 3 & 5 & 7 \\ 4 & 9 & 2 \\ 8 & 1 & 6 \end{pmatrix}$$

is* functions related to sets

- function `issorted` returns `true` if array is sorted

- function `ismember(A,B)` tests whether an element of array `B` is also an element of array `A`

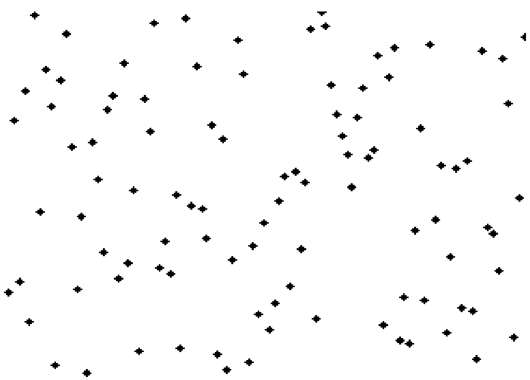
```
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
```

```
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])

ans =

     1     1     0
     1     0     0
     0     0     0
```

Array sorting

600 s ↑

- try to write your own sorting algorithm `bubbleSort.m`
  - use the *bubble sort* algorithm
  - use the function `issorted` to test whether the resulting array is sorted



wikipedia.org

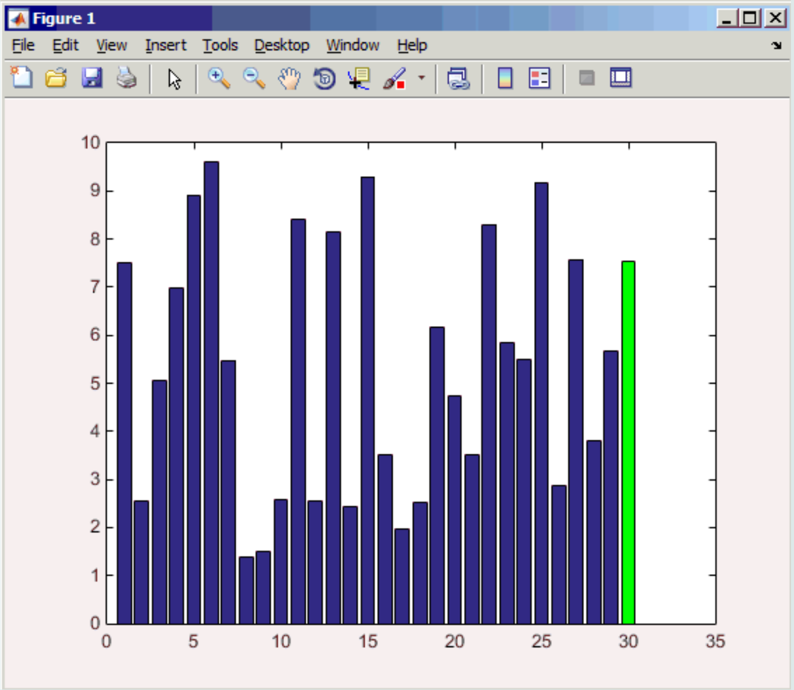
if you wish, you can use the following code inside loops :

```
figure(1);
plot(R,'*','LineWidth',2);
pause(0.01);
```

Array sorting

600 s

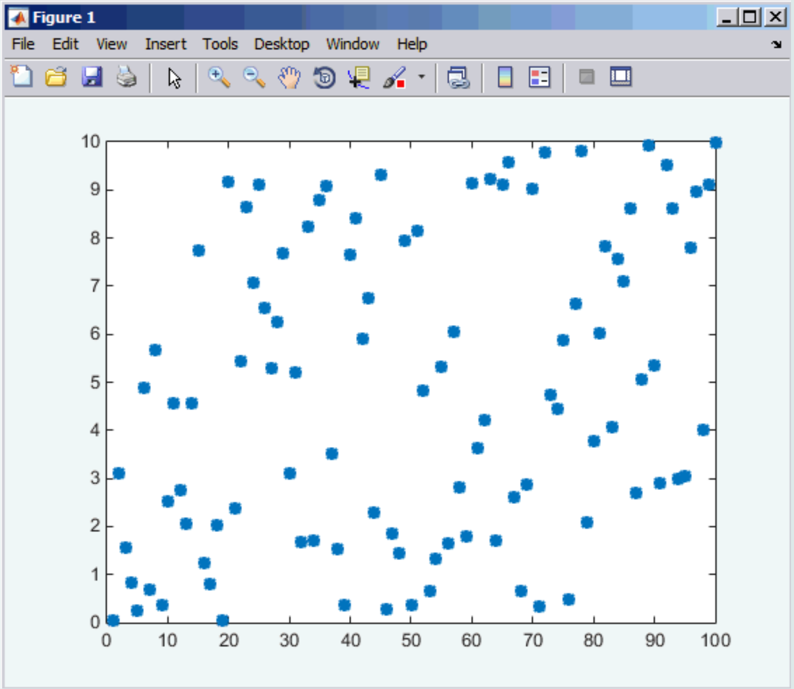
- try to get plot as in the figure using `bar` function:



Array sorting – shaker sort

600 s

- try to write your own sorting algorithm `shakerSort.m`
  - use the *shaker sort* algorithm



Searching in an array – `find`

- `find` function is a very useful one!!
- returns positions of non-zero (logical true) elements of a matrix
  - useful for searching in an array of logical values
  - example: find positions of those elements of vector $\mathbf{A} = \begin{pmatrix} \frac{\pi}{2} & \pi & \frac{3}{2}\pi & 2\pi \end{pmatrix}$ fulfilling the condition $\mathbf{A} > \pi$

```
>> A = pi/2*(1:4)
>> find(A > pi)
```

  - compare the above command with `A > pi`. What is the difference?
- function `find` can also search a square matrix etc.
- to find first / last `k` non-zero elements of `X`:

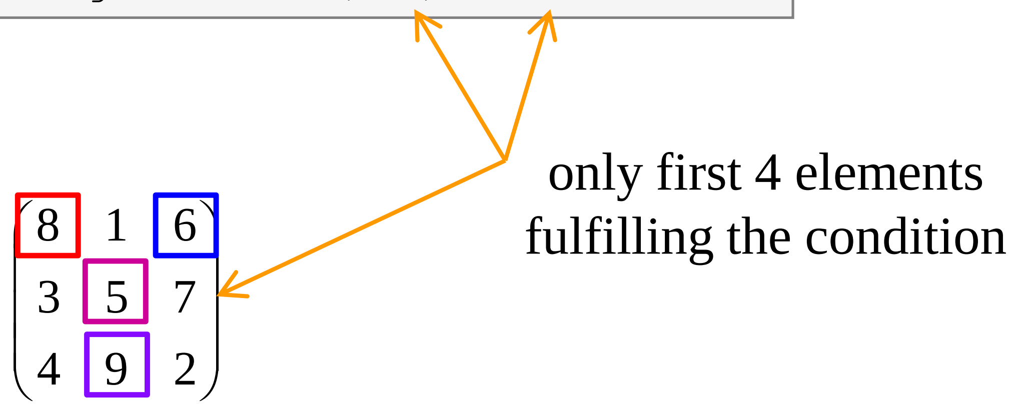
```
>> ind = find(X, k, 'first')
>> ind = find(X, k, 'last')
```

- for more see `>> doc find`

Advanced application of `find` function

- can be called with more output parameters as well, which can often prove useful!

```
>> [rw,cl] = find(magic(3) > 4, 4, 'first')
```




```
rw =

     1
     2
     3
     1
```

```
cl =

     1
     2
     2
     3
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Array searching #1

420 s

- sort the vector $\mathbf{v} = \begin{pmatrix} 16 & 2 & 3 & 13 & 5 & 11 & 10 & 8 & 9 & 7 & 6 & 12 & 4 & 14 & 15 & 1 \end{pmatrix}$ in descending order and find the elements of the vector (and their respective positions within the vector) that are divisible by three and at the same time are greater than 10

```
>> v  = reshape(magic(4)', [1 numel(magic(4))])
```

```
v =

    16     2     3    13     5    11    10     8     9     7     6    12     4    14    15     1


v1 =

     0     1     0     0     1     0     0     0     0     0     0     0     0     0     0     0


ans =

    15    12


ans =

     2     5
```

Array searching #2

300 s

- in matrix **w**

```
>> w = (8:-1:2)'*(1:1/2:4).*magic(7)
```

find last 3 values that are smaller than 50

- find out the column and row positions of the values

```
w =

  240.0000  468.0000  768.0000   20.0000  240.0000  532.0000  896.0000
  266.0000  493.5000   98.0000  157.5000  378.0000  661.5000  812.0000
  276.0000   54.0000   96.0000  255.0000  468.0000  735.0000  888.0000
   25.0000  105.0000  160.0000  312.5000  510.0000  630.0000  900.0000
   52.0000   90.0000  192.0000  330.0000  504.0000  616.0000   64.0000
   63.0000  103.5000  192.0000  307.5000  387.0000   31.5000  144.0000
   44.0000   93.0000  160.0000  245.0000   12.0000   77.0000  160.0000
```

Application of the `find` function

600 s ↑

- Samples of demodulated signal of a radio receiver can be approximated as :

```
w = 0.6833; t = 1:10; % time
samples = 2.7 + 0.5*(cos(w*t) - sin(w*t) - cos(2*w*t) + sin(2*w*t) ...
   - cos(3*w*t) + 3*sin(3*w*t) + 2*cos(4*w*t) + 4*sin(4*w*t));
plot(samples, '*')
```

- Voltage corresponding to characters are within ±0.5 V tolerance
- Decipher the message!

Voltage [V]	Character
1	a
2	c
3	d
4	g
5	m
6	r
7	s

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

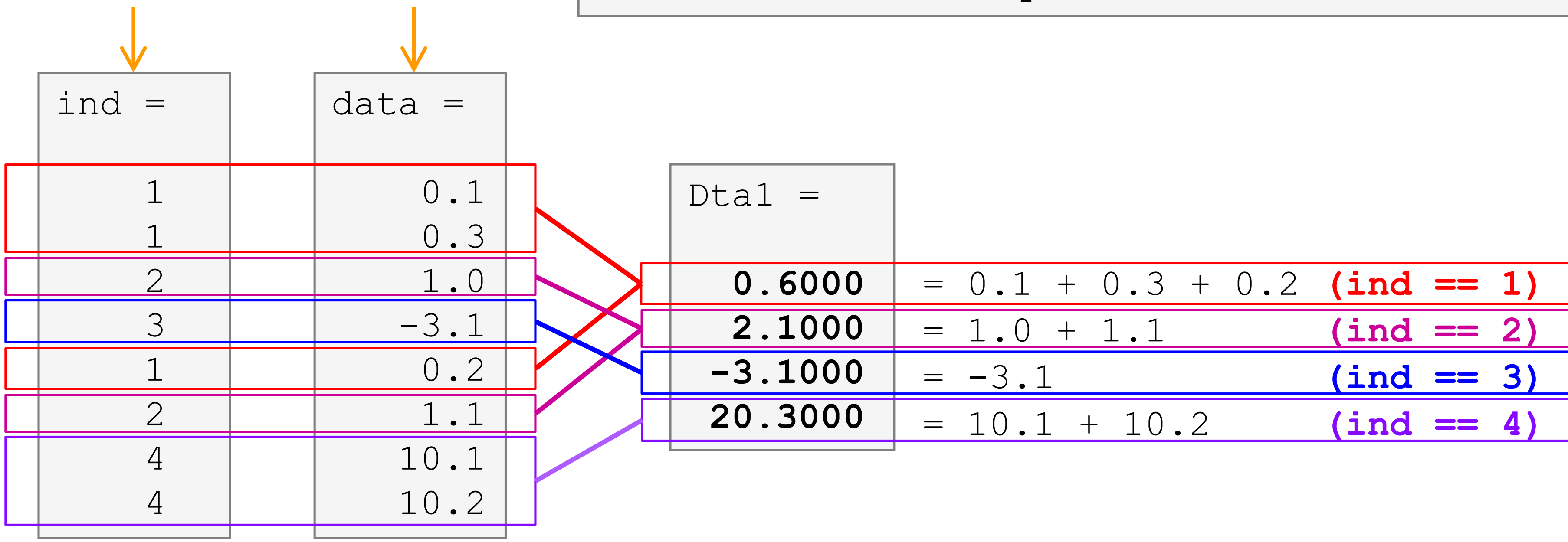
Function `accumarray` #1

- the function `accumarray` is able to group data with the same index
  - not a very well known function, but an exceptionally useful one
- quite often we deal with a dataset that is organised in the following way:

indexes (e.g. measurement number)

values (measured)

```
>> ind  = [1 1 2 3 1 2 4 4]';
>> data = [.1 .3 1 -3.1 .2 1.1 10.1 10.2]';
>> Dta1 = accumarray(ind, data)
```



Function `accumarray` #2

- basic operation applicable to data from one 'box' (data with the same index) is summation
- any other function can be applied, however
  - e.g. maximum of a set of elements with the same index
  - we use the `max` function

```
>> Dta2 = accumarray(ind, data, [], @max)
```

```
Dta2 =
   0.3000
   1.1000
  -3.1000
  10.2000
```

  - e.g. listing of all elements with the same index
  - we use so called handle function and `cell` data type

```
>> Dta3 = accumarray(ind, data, [], @(x) {x})
```

```
Dta3 =
  [3x1 double]
  [2x1 double]
  [   -3.1000]
  [2x1 double]
```

Function `accumarray` #3

- the function has a wide variety of other features
- it is possible, for instance, to use 2D indexation of results
    - the results are not put in a 1D set of 'boxes' but to a 2D array instead

```
>> ind = [1 1;2 2;1 2;1 3;1 1;3 1];
>> data = [10 22 12 13 1 pi];
>> Dta4 = accumarray(ind, data)
```

```
ind =

   1   1
   2   2
   1   2
   1   3
   1   1
   3   1
```

```
data =

   10
   22
   12
   13
    1
   pi
```

ind == [1 1] 10 + 1 = **11**	ind == [1 2] **12**	ind == [1 3] **13**
ind == [2 1] **0**	ind == [2 2] **22**	ind == [2 3] **0**
ind == [3 1] **pi**	ind == [3 2] **0**	ind == [3 3] **0**

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Function accumarray

300 s

- account transfers in CZK, EUR a USD are as follows
  - (CZK ~ 1, EUR ~ 2, USD ~ 3)
- find out account balance in each currency
  - the exchange rate is 28 CZK = 1€, 21 CZK = 1$, find out total balance

$$\begin{pmatrix} 1 & -110 \\ 1 & -140 \\ 2 & -22 \\ 3 & -2 \\ 2 & -34 \\ 1 & -1300 \\ 2 & -15 \\ 1 & -730 \\ 3 & 24 \end{pmatrix}$$

```
>> dta = [1 -110; 1 -140; 2 -22; 3 -2; ...
          2 -34; 1 -1300; 2 -15; 1 -730; 3 24]
>> K   = [1 28 21]
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Functions in Matlab

- more efficient, more transparent and faster than scripts
- defined input and output, comments → function header is necessary
- can be called from Command Window or from other function (in both cases the function has to be accessible)
- each function has its own work space created upon the function's call and terminated with the last line of the function

Function types by origin

- built-in functions
  - not accessible for editing by the user, available for execution
  - optimized and stored in core
  - usually frequently used (elementary) functions
- Matlab library functions ([`toolbox`] directory)
  - subject-grouped functions
  - some of them are available for editing (not recommended!)
- user-created functions
  - fully accessible and editable, functionality not guaranteed
  - obligatory parts: function header
  - mandatory parts of the function: function description, characterization of inputs and outputs, date of last editing, function version, comments are recommended

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

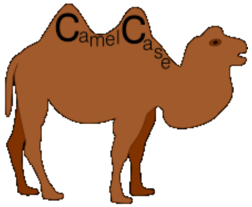
Function header

- has to be the first line of a standalone file!
  - function can't be placed for instance at the end of a script
- function header has the following syntax:

```
function [out1, out2, ...] = functionName(in1, in2, ...)
```

keyword — function's output parameters — function's name — function's input parameters

  - `functionName` has to follow the same rules as a variable's name
  - `functionName` can't be identical to any of its parameters' name
  - `functionName` is usually typed as `lowerCamelCase` or using underscore character (`my_function`)



Function header – examples

```
function functA
%FUNCTA - unusual, but possible, without input and output
```

```
function functB(parIn1)
%FUNCTB - e.g. function with GUI output, print etc.
```

```
function parOut1 = functC
%FUNCTC - data preparation, pseudorandom data etc.
```

```
function parOut1 = functD(parIn1)
%FUNCTD - „proper“ function
```

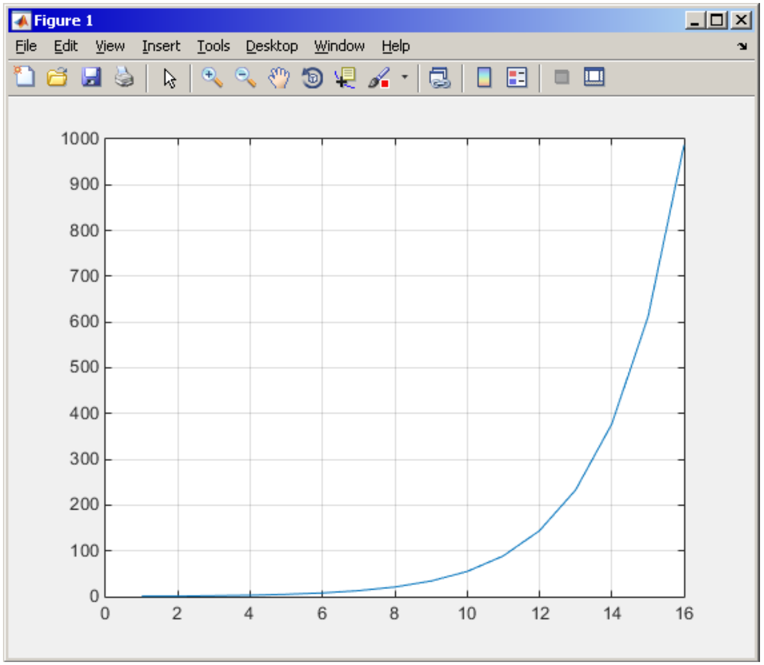
```
function parOut1 = functE(parIn1, parIn2)
%FUNCTE - proper function, square brackets [] not necessary
```

```
function [parOut1, parOut2] = functF(parIn1, parIn2)
%FUNCTF - proper function with more parameters
```

Calling Matlab function

```
>> f = fibonacci(1000);  % calling from command prompt
>> plot(f); grid on;
```

```
function f = fibonacci(limit)
%% Fibonacci sequence
f = [1 1]; pos = 1;
while f(pos) + f(pos+1) < limit
    f(pos+2) = f(pos) + f(pos+1);
    pos = pos + 1;
end
```



- Matlab carries out commands sequentially
  - input parameter: `limit`
  - output variable: Fibonacci series `f`
  - drawbacks:
    - input is not treated (any input can be entered)
    - matrix `f` is not allocated, i.e. matrix keeps growing (slow)

Simple example of a function

- any function in Matlab can be called with less input parameters than stated in the header
- any function in Matlab can be called with less output parameters than stated in the header
  - for instance, consider following function:

```
function [parOut1, parOut2, parOut3] = functG(parIn1, parIn2, parIn3)
%FUNCTG - 3 inputs, 3 outputs
```

  - all following calling syntaxes are correct

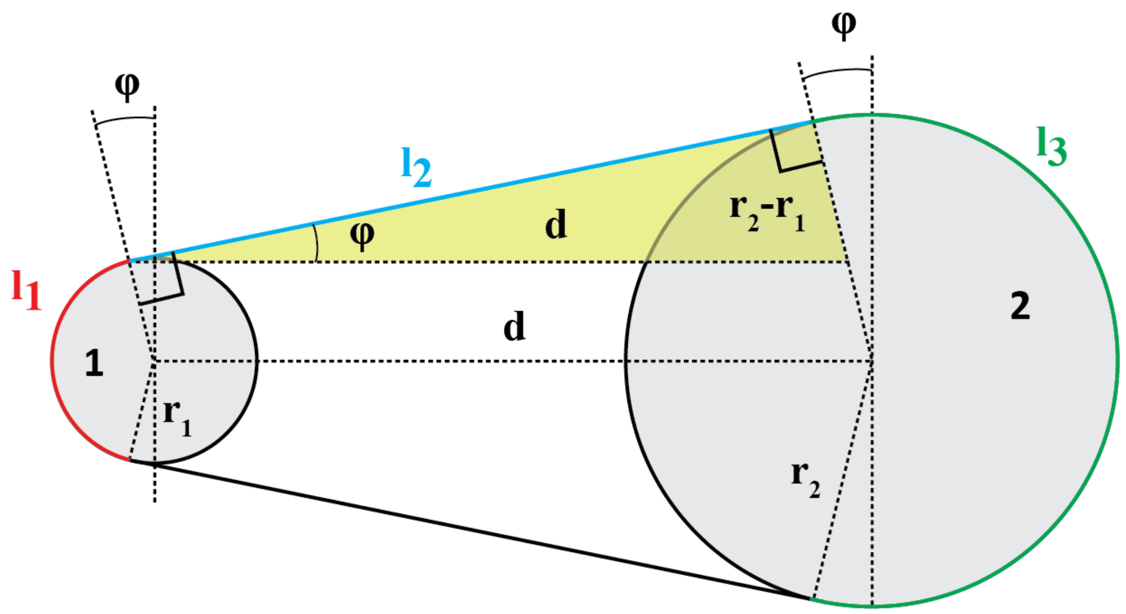
```
>> [parO1, parO2]          = functG(pIn1, pIn2, pIn3)
>> [parO1, parO2, parO3] = functG(pIn1)
>> functG(pIn1,pIn2,pIn3)
>> [parO1, parO2, par03] = functG(pIn1, pIn2, pIn3)
>> [parO1, ~, par03] = functG(pIn1, [], pIn3)
>> [~, ~, par03] = functG(pIn1, [], [])
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Simple example of a function

100 s

- propose a function to calculate length of a belt between two wheels
  - diameters of both wheels are known as well as their distance (= function's inputs)
  - sketch a draft, analyze the situation and find out what you need to calculate
  - test the function for some scenarios and verify results
  - comment the function, apply commands `lookfor`, `help`

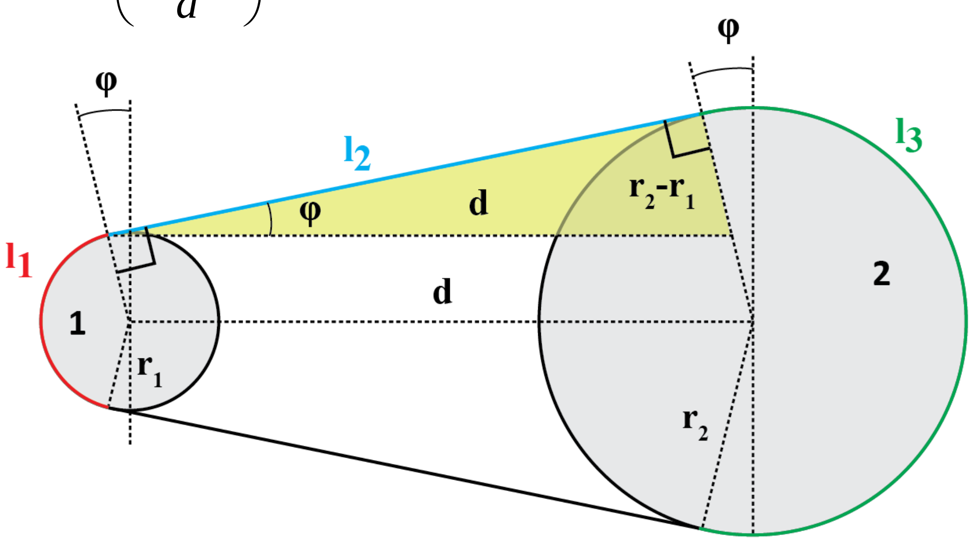


Simple example of a function

500 s ↑

- total length is $l = l_1 + 2l_2 + l_3$
- known diameters ⟶ recalculate to radiuses $r_1 = d_1 / 2,\ r_2 = d_2 / 2$
- l_2 to be determided using Pythagorean theorem : $l_2 = \sqrt{d^2 - (r_2 - r_1)^2}$
- Analogically for φ: $\varphi = \mathrm{asin}\left(\dfrac{r_2 - r_1}{d}\right)$
- and finally : $l_1 = (\pi - 2\varphi) r_1$

  $l_3 = (\pi + 2\varphi) r_2$

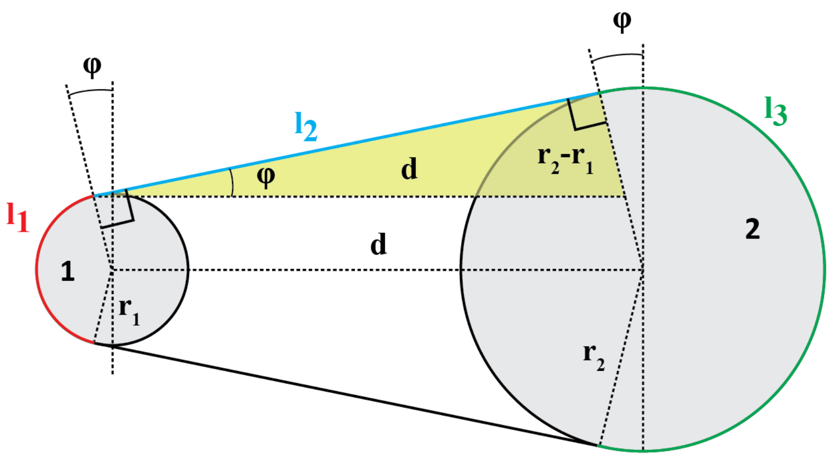


- verify your results using

$$d_1 = 2,\ d_2 = 2,\ d = 5$$
$$L = \pi + 2 \cdot 5 + \pi \approx 16.2832$$

Simple example of a function

```
>> help band_wheel,
>> type band_wheel,
>> lookfor band_wheel,
```



Comments inside a function

function help, displayed upon:
`>> help myFcn1`

```
function [dataOut, idx] = myFcn1(dataIn, method)
%MYFCN1: Calculates...
% syntax, description of input, output,
% expamples of function's call, author, version
% other similar functions, other parts of help

matX = dataIn(:, 1);
sumX = sum(matX); % sumation
%% displaying the result:
disp(num2str(sumX));
```

1st line (so called H1 line), this line is searched for by `lookfor`. Usually contains function's name in capital characters and a brief description of the purpose of the function.

```
function pdetool(action, flag)
%PDETOOL PDE Toolbox graphical user interface (GUI).
%   PDETOOL provides the graphical user ...
```

```
DO COMMENT!
% Comments significantly improve
% transparency of functions' code !!!
```

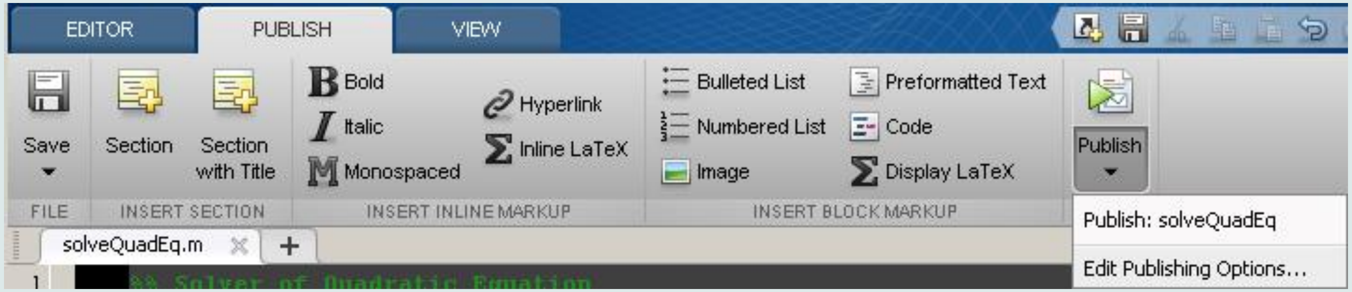

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Function documentation – example

```
function Z = impFcn(f,MeshStruct,Zprecision)
%% impFcn: Calculates the impedance matrix
% -solver-
%
%  Syntax:
%    Z = impFcn(f,MeshStruct,Zprecision)
%
% impFcn version history:
%    ver. 1.0a
%    ver. 1.0b (8.8.2011)
%         default option (if nargin == 2) is Zprecision = true
%
%    Last update: 8.8.2013
%
% Notes:
% A) (contains rwg3.m): Calculates the impedance matrix (includes infinite
%                       groud plane)
% B)
%   RHO_P(3,9,edgTotal)
%   RHO_M(3,9,edgTotal)
%
%   Temporary variables:
%   RP(3,9,EdgesTotal)
%
% C) See: [1] Sergey N. Makarov: Antenna and EM Modeling with MATLAB
%    Copyright 2002 AEMM. Revision 2002/03/05 and ČVUT-FEL 2007-2010
%
% D) This function is used by preTCM software!
%
% Author(s): Sergey N. Makarov, Copyright 2002 AEMM. Revision 2002/03/05
%            Miloslav Čapek, capekm6@fel.cvut.cz, 2010-2013
%
% See also impBsxFcn, impGndFcn, preTCM, prepTCMinput, TCM_RUN_solver
```

Function `publish`

- serves to create script, function or class documentation
- provides several output formats (html, doc, ppt, LaTeX, ...)
- help creation (`>> doc my_fun`) directly in the code cpmments!
  - provides wide scale of formatting properties (titles, numbered lists, equations, graphics insertion, references, ...)
- enables to insert print screens into documentation
  - documented code is implicitly launched on publishing
- supports documentation creation directly from editor menu :



Function `publish` - example

```matlab
%% Solver of Quadratic Equation
% Function *solveQuadEq* solves quadratic equation.
%% Theory
% A quadratic equation is any equation having the form
% $ax^2+bx+c=0$
% where |x| represents an unknown, and |a|, |b|, and |c|
% represent known numbers such that |a| is not equal to 0.
%% Head of function
% All input arguments are mandatory!
function x = solveQuadEq(a, b, c)
%%
% Input arguments are:
%%
% * |a| - _qudratic coefficient_
% * |b| - _linear coefficient_
% * |c| - _free term_
%% Discriminant computation
% Gives us information about the nature of roots.
D = b^2 - 4*a*c;
%% Roots computation
% The quadratic formula for the roots of the general
% quadratic equation:
%
% $$x_{1,2} = \frac{ - b \pm \sqrt D }{2a}.$$
%
% Matlab code:
%%
x(1) = (-b + sqrt(D))/(2*a);
x(2) = (-b - sqrt(D))/(2*a);
%%
% For more information visit <http://elmag.org/matlab>.
```

publish →

Solver of Quadratic Equation

Function **solveQuadEq** solves quadratic equation.

Contents

Theory

A quadratic equation is any equation having the form $ax^2+bx+c=0$ where x represents an unknown, and a, b, and c represent known numbers such that a is not equal to 0.

Head of function

All input arguments are mandatory!

```matlab
function x = solveQuadEq(a, b, c)
```

Input arguments are:

- a - *qudratic coefficient*
- b - *linear coefficient*
- c - *free term*

Discriminant computation

Gives us information about the nature of roots.

```matlab
D = b^2 - 4*a*c;
```

Roots computation

The quadratic formula for the roots of the general quadratic equation:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}.$$

Matlab code:

```matlab
x(1) = (-b + sqrt(D))/(2*a);
x(2) = (-b - sqrt(D))/(2*a);
```

For more information visit http://elmag.org/matlab.

Discussed functions

`intersect`	intersection of sets (vectors / matrices)	
`union`	intersection of sets (vectors / matrices)	
`setdiff`	Subtraction of sets (intersection of a set and complement of another set)	
`setxor`	exclusive intersection	
`unique`	selection of unique elements of an array	
`sort`	sort vector/matrix elements	
`sortrows`	sorts rows of a matrix as a whole	
`accumarray`	group data	●
`ismember`	is given element is member of array?	
`issorted`	is array sorted?	
`find`	find elements fulfilling given condition	●
`function`	function header	●

Exercise #1

600 s

- expand exponential function using Taylor series:
  - in this case it is in fact McLaurin series (expansion about 0)

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$$

  - compare with result obtained using `exp(x)`
  - find out the deviation in [%] (what is the base, i.e. 100% ?)
  - find out the order of expansion for deviation to be lower than 1%

  - implement the code as a script, enter :
    x (function argument)
    N (order of the series)

Exercise #2

600 s

```
>> x   = 1.15;
>> n   = 10;
>> N   = 0:n;
>> f   = (x.^N)./factorial(N);
>> f1  = sum(f),
>> f2  = exp(x),
>> err = 100*abs(1 - f1/f2)
```

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$$

- implement as a function
  - choose appropriate name for the function
  - input parameters of the function are `x` and `n`
  - Output parameters are values `f1`, `f2` and `err`
  - add appropriate comment to the function (H1 line, inputs, outputs)

  - test the function

Exercise #3

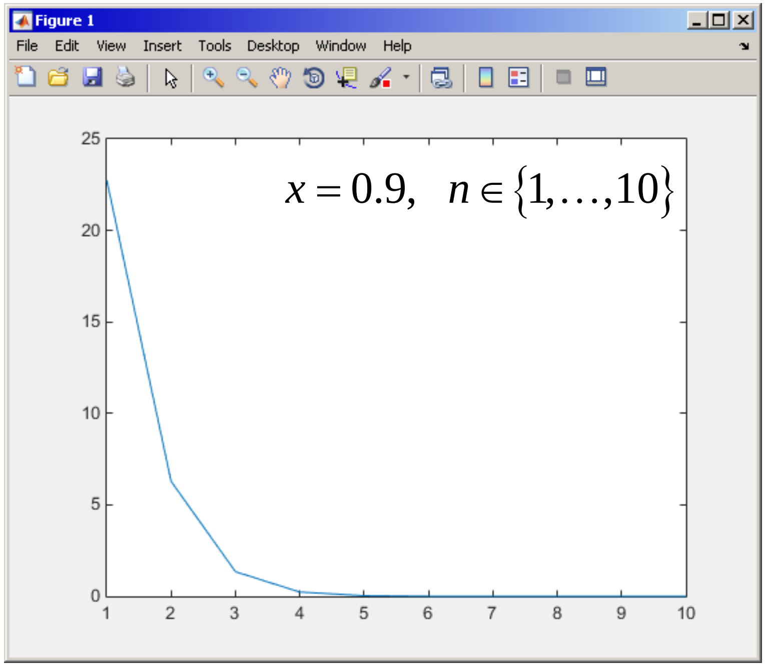
600 s

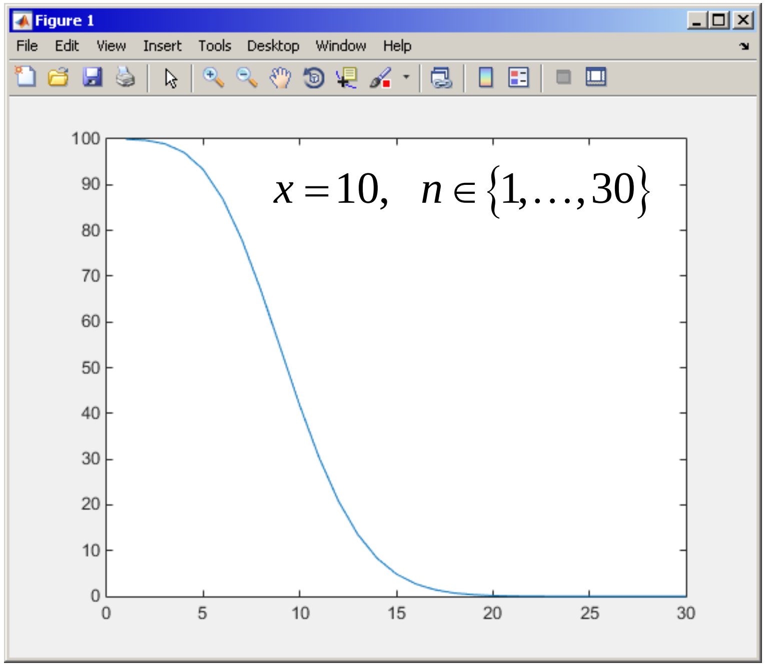
```
function [f1, f2, err] = exp_approx(x, n)
%% exp_approx: calculates an aproximation of exp(x)
% [f1, f2, err] = exp_approx(x,n)
% x  ... function argument, n  ... order of approximation
% f1 ... calculated value
% f2 ... „exact" value (given by built-in function)
% err ... relativ. error
% example: [fa, fp] = exp_approx(4, 30);

    N   = 0:n;
    f   = (x.^N)./factorial(N);    % it is vectorized
    f1  = sum(f);                  % approximative value
    f2  = exp(x);                  % exact value
    err = 100*abs(1 - f1/f2);         % absolute error (in [%])
```

- create a script to call the above function (with various `n`)
  - find out accuracy of the approximation for $x = 0.9, \quad n \in \{1, \ldots, 10\}$
  - plot the resulting progress of the accuracy (error as a function of n)

Exercise #4

```
Order   = 1:10;
x       = 0.9;
Error   = zeros(1, length(Order));
for mErr = 1:length(Order)
    [~,~,Error(mErr)] = exp_approx(x, Order(mErr));
end
plot(Order, Error);
```





Exercise #5

- measurement of temperature was carried out in the course of 5 days every second clock hour. data was measured at 3 different sites (A, B, C)

- find out average daily temperature in given week for all 3 sites
  - i.e. get mean value of measurement at the same hour on the same site

- generate the data using `temperature_measurement.m`
  - see the script on the following slide
  - see the variables required

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Exercise #6

script for data generation

and the results …

```
clear; clc;
%% allocation
days = 5; hours = 12;
TimeA = zeros(days*hours,1);
TimeB = TimeA;
TimeC = TimeA;
%% creation of time data-set
for kDay = 1:days
    TimeA((hours*(kDay-1)+1):(hours*(kDay-1)+12),1)  = 2*(randperm(12)-1)';
    TimeB((hours*(kDay-1)+1):(hours*(kDay-1)+12),1)  = 2*(randperm(12)-1)';
    TimeC((hours*(kDay-1)+1):(hours*(kDay-1)+12),1)  = 2*(randperm(12)-1)';
end
%% place and tempreture data-sets
PlaceA = abs(abs(TimeA - 11) - 10) + 10 +  5.0*rand(size(TimeA,1),1);
PlaceB = abs(abs(TimeB - 12) - 10) +  5 + 10.0*rand(size(TimeB,1),1);
PlaceC = abs(abs(TimeC - 11) - 11) +  5 +  7.5*rand(size(TimeC,1),1);

%% generating final variables for the example
TimeAndPlace = [TimeA/2+1   ones(size(TimeA,1),1);...
                TimeB/2+1 2*ones(size(TimeA,1),1);...
                TimeC/2+1 3*ones(size(TimeA,1),1)];
MeasuredData  = [PlaceA; PlaceB; PlaceC];

%% plot final data-set
plot(TimeA,PlaceA,'LineWidth',1,'LineStyle','none','Marker','x',...
    'MarkerSize',15); hold on;
plot(TimeB,PlaceB,'LineWidth',1,'LineStyle','none','Marker','*',...
    'MarkerSize',15,'Color','r');
plot(TimeC,PlaceC,'LineWidth',2,'LineStyle','none','Marker','o',...
    'MarkerSize',10,'Color','g');
set(gcf,'Color','w','pos',[50 50 1000 600]); set(gca,'FontSize',15);
xlabel('time','FontSize',15); ylabel('Temperature','FontSize',15);
title('Measured Data'); grid on; legend('Place A','Place B','Place C');
```

Exercise #7

- all the data are contained in 2 matrices:
  - `TimeAndPlace (5×3×12,2) = (180,2)`
  - `MeasuredData (5×3×12,1) = (180,1)`

number of days | number of measurement sites | number of measurements per day

- unfortunately, data in `TimeAndPlace` are intentionally unsorted

INDEXES:	`TimeAndPlace =`		`MeasuredData =`	**DATA:**
tindex = 10, Place = 1	10	1	15.0797	T(10,1) = 15.0797 °C
	4	1	18.9739	
	7	1	19.3836	
	...	...	...	
	12	2	9.9506	
tindex = 6, Place = 2	6	2	19.7588	T(6,2) = 19.7588 °C
	...	...	...	

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Exercise #8

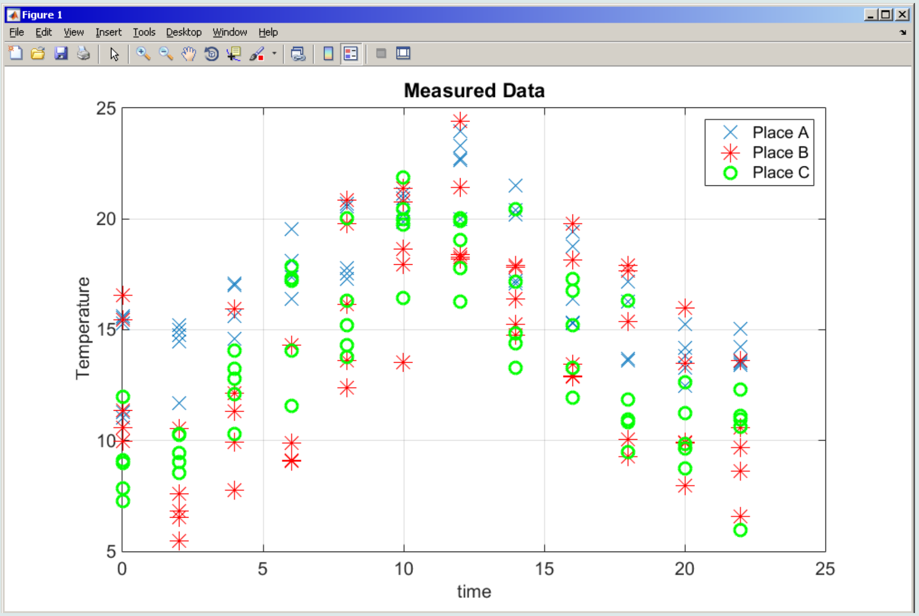
600 s

- following holds true
  - Place1 ~ measurement site A
  - Place2 ~ measurement site B
  - Place3 ~ measurement site C
  - measurement hour = 2*(tindex-1)

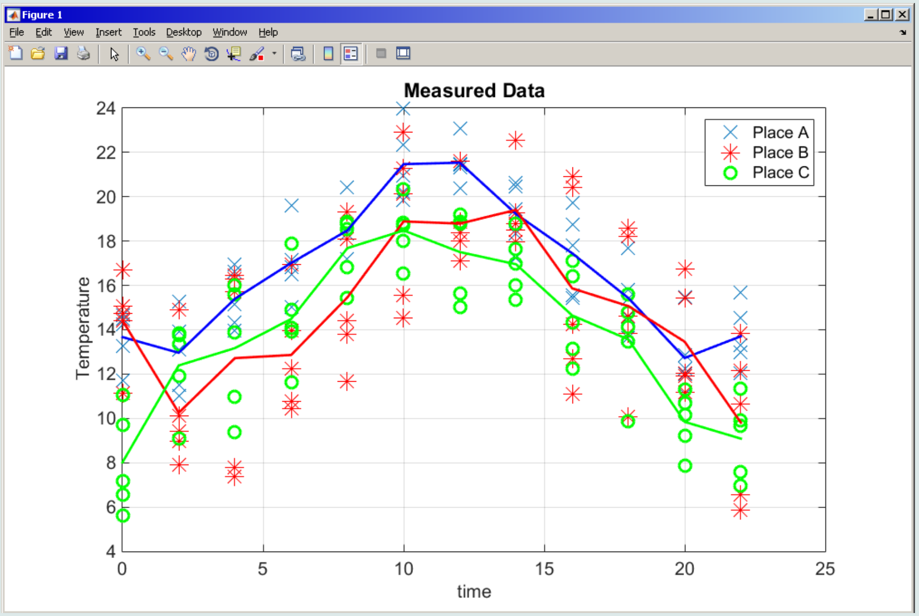
- now try to place your cone in the script to carry out the averaging and plot the data in the existing figure

```
%% PLACE YOUR CODE HERE
%===========================================================================

% ...
% dataA = ...
% dataB = ...
% dataC = ...
%===========================================================================

%% plot the averaged data
plot(0:2:22,dataA,'LineWidth',2,'Color','b','LineStyle','-');
plot(0:2:22,dataB,'LineWidth',2,'Color','r','LineStyle','-');
plot(0:2:22,dataC,'LineWidth',2,'Color','g','LineStyle','-');
```

Exercise #9



measured data



measured and averaged data

Thank you!

ver. 4.3 (12/11/2015)

Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz

Document created as part of A0B17MTB course.