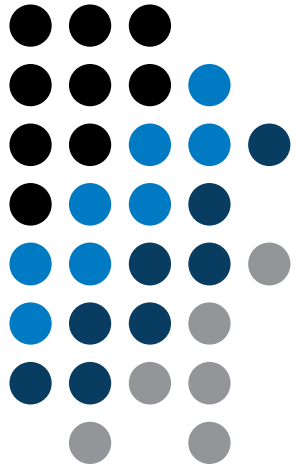
# A0B17MTB – Matlab

# Část #6


Miloslav Čapek
miloslav.capek@fel.cvut.cz

Filip Kozák, Viktor Adler

Katedra elektromagnetického pole
B2-626, Dejvice



# Cvičení #10 - řešení

# Cvičení #11 - řešení

# Cvičení #12 - řešení

# Cvičení #12 - řešení

- aproximace čísla $\pi$ s přesností $1 \cdot 10^{-6}$
- aproximace pomocí $\frac{\pi}{4} = \sum_{n=0}^{\infty} (-1)^n \frac{1}{2n+1} = x - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$
  - 1 000 001 cyklů, **64 ms**
- aproximace pomocí $\frac{\pi}{8} = \sum_{n=0}^{\infty} \frac{1}{(4n+1)(4n+3)} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \ldots$
  - 500 000 cyklů, **18 ms**
- aproximace pomocí $\frac{\pi}{4} = 6 \arctan\left(\frac{1}{8}\right) + 2 \arctan\left(\frac{1}{57}\right) + \arctan\left(\frac{1}{239}\right)$
  - 4 cykly, **0.0062 ms**

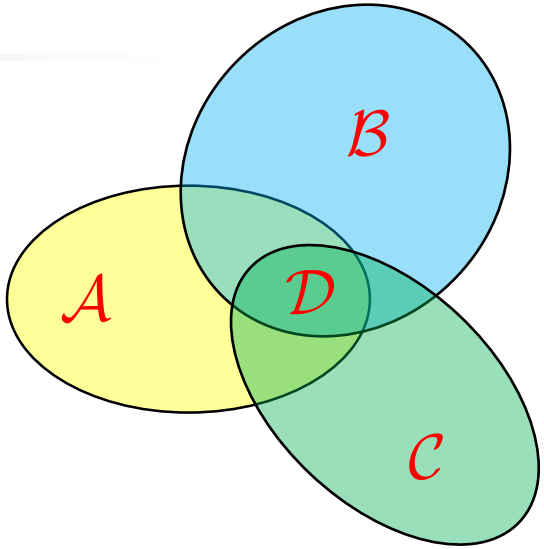
# Naučíte se …

**Množinové operace**

**Třídění prvků**

**Vyhledávání prvků**

**Funkce #1**




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

$$\mathcal{A} \cap \mathcal{B} = \{x : x \in \mathcal{A} \wedge x \in B\}$$

# Množinové operace

- v Matlabu známe následující operace (operátory) mezi prvky / poli
  - aritmetické (část #1)
  - relační (část #3)
  - logické (část #3)
  - množinové (část #4)
  - bitové (nápověda, `>> doc`)

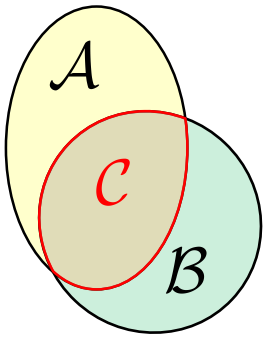
- množinové operace pracují s vektory, maticemi, poli, celly, řetězci, tabulkami
  - zpravidla nejsou důležité vzájemné velikost těchto struktur

| | |
|---|---|
| průnik množin | `intersect` |
| sjednocení množin | `union` |
| rozdíl množin | `setdiff` |
| exkluzivní disjunkce | `setxor` |
| jedinečné prvky množiny | `unique` |
| třídění, třídění řádků vcelku | `sort, sortrows` |
| je prvek členem množiny? | `ismember` |
| je množina setříděna? | `issorted` |

# Množinové operace #1

- průnik množin: `intersect`
  - příklad: průnik matice a vektoru:

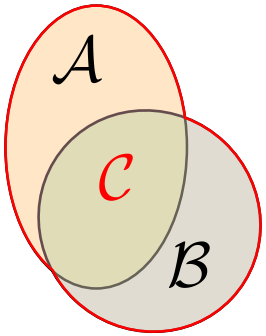
```
>> A = [1 -1; 3 4; 0 2];
>> b = [0 3 -1 5 7];
>> c = intersect(A, b)
% c = [-1; 0; 3]
```




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

- sjednocení množin: `union`
  - všechny množinové operace lze provádět „po řádcích“ (pak je nutné dodržet počet sloupců obou proměnných)

```
>> 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]
```

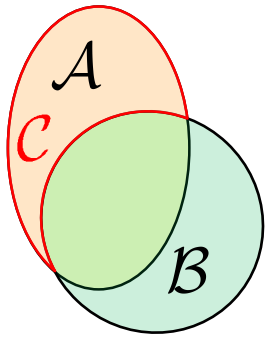



$\mathcal{C} = \mathcal{A} \cup \mathcal{B}$

**intersect**
**union**
setdiff
setxor
unique
sort, sortrows
ismember
issorted

# Množinové operace #2

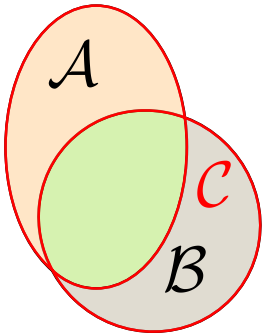
- průnik množiny s doplňkem jiné: `setdiff`
  - všechny množinové operace lze volat s více výstupními proměnnými – zjistíme i indexy dat




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

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

- exkluzivní průnik (XOR): `setxor`
  - všechny množinové operace lze provést jako „*stable*“ (beze změny pozice prvků) nebo jako „*sorted*“ (prvky jsou setříděny)




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

```
>> 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]
```

- `intersect`
- `union`
- **`setdiff`**
- **`setxor`**
- `unique`
- `sort, sortrows`
- `ismember`
- `issorted`

# Množinové operace #3

- výběr unikátních prvků pole: `unique`
  - množinové operace lze použít i pro pole které neobsahují (pouze) čísla

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

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

- intersect
- union
- setdiff
- setxor
- **unique**
- sort, sortrows
- ismember
- issorted

- lze kombinovat všechny výše uvedené techniky
  - např. vyhodnocení unikátních prvků nad maticí po řádcích, vč. indexů:

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

  - Vysvětlete funkci kódu výše? Je nastavení „`rows`“ nezbytné?

# Množinové operace #1

600 s ↑

- uvažujte tři vektory **a**, **b**, **c** obsahující přirozená čísla $x \in \mathbb{N}$ tak, že
  - vektor **a** obsahuje všechna prvočísla do 1000 (včetně)
  - vektor **b** obsahuje všechna sudá čísla do 1000 (včetně)
  - vektor **c** je doplňkem vektoru **b** na stejném intervalu

- nalezněte vektor **v** tak, že $\mathbf{v} = \mathbf{a} \cap (\mathbf{b} + \mathbf{c})$,
  - jaké prvky vektor **v** obsahuje?

$$\mathbf{b} + \mathbf{c} \equiv \{b_i + c_i\}, \quad i \in \{1, 500\}$$

$$b_{i-1} < b_i < b_{i+1} \ \wedge \ c_{i-1} < c_i < c_{i+1}, \ \forall i$$

- kolik prvků obsahuje vektor **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
```

# Množinové operace #2

500 s ↑

- odhadněte a ověřte výpočtem v Matlabu následující operaci:

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

  - čím se vyznačují prvky výsledného vektoru **w**?

- pomocí logického indexování a matematických funkcí určete
  - kolik prvků vektoru **w** je dělitelných 3?

# Množinové operace #3

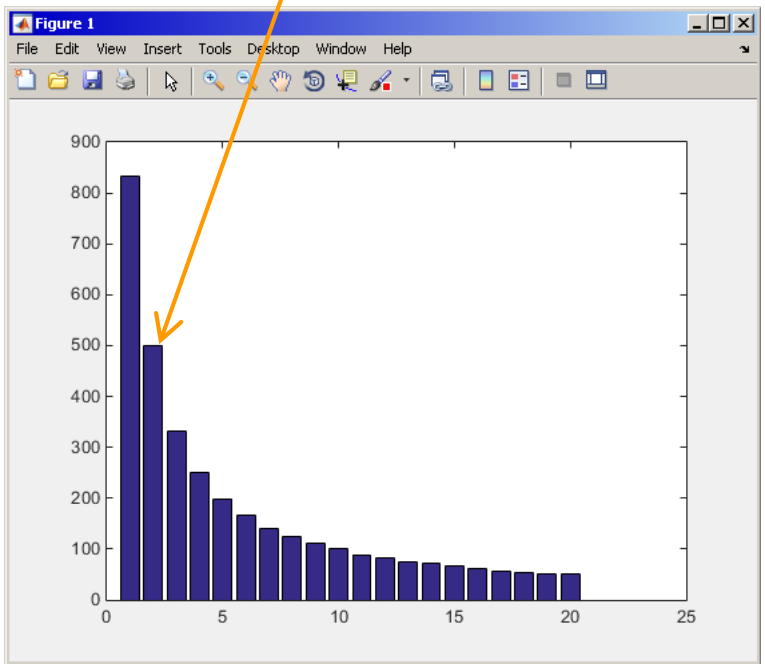
500 s ↑

- předchozí příklad zpracujte jako skript:

- skript dále modifikujte a vypočtěte kolik čísel z vektoru **w** je dělitelných čísly 1 až 20
  - pro výpočet využijte např. `for` cyklu
  - výsledky vykreslete ve formě grafu `bar`

# Množinové operace #4

kupř. čísel dělitelných 2,
které nejsou prvočísla je
v intervalu 1 až 1000 přesně 499



# Množinové operace #5

600 s ↑

- Radioreléové pojítko pracuje na frekvenci 80 GHz přes vzdálenost 20 km s modulací 64-QAM
  - pro dostatečně malou chybovost přenosu bez použití synchronizace a kódování je potřeba fázová stálost přijatého signálu ±0.5°
  - to odpovídá změně vzdálenosti mezi anténami o ±5 μm
  - statistika vzdáleností spoje s normálním rozdělením s $1 \cdot 10^6$ členy se dá vygenerovat jako:

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

  - Kolikrát je ve vektoru `distances` obsažena přesná vzdálenost `L`?
  - Kolik má vektor `distances` unikátních členů?
  - Dá se rozdělení považovat za spojité?

# Třídění prvků pole #1

- setřídí prvky daného pole
  - podél sloupců, vzestupně:

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

  - podél řádek, vzestupně:

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

  - sestupně:

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

  - sestupně, podél řádek:

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

- zkuste např. pro:

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

intersect
union
setdiff
setxor
unique
**sort**, sortrows
ismember
issorted

# Třídění prvků pole #2



- příkaz `sortrows` setřídí řádky matice
  - prvky v řádcích nejsou prohazovány – řádky jsou tříděny jako celek

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

---

`SORT:`

$$\left(\begin{array}{c|c|c} 3 & 1 & 2 \\ 4 & 5 & 6 \\ 8 & 9 & 7 \end{array}\right)$$

---

`SORTROWS:`

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

# Funkce `is*` související s množinami

intersect
union
setdiff
setxor
unique
sort, sortrows
**ismember**
**issorted**

- funkce `issorted` vrátí hodnotu `true`, je-li pole setříděno

- funkce `ismember(A,B)` testuje, zda je některý z prvků pole `B` rovněž prvkem pole `A`

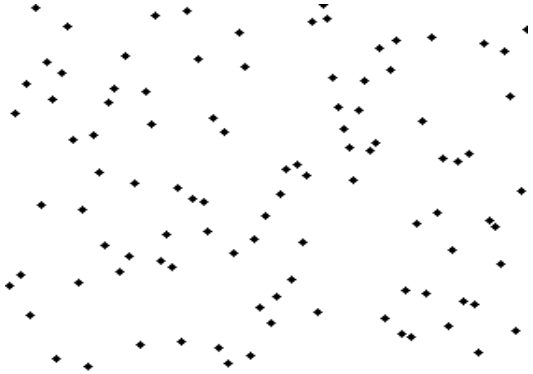
```
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
```

```
>> 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
```

# Třídění polí

600 s ↑

- zkuste si naprogramovat vlastní algoritmus třídění `bubbleSort.m`
  - využijte třídícího algoritmu *bubble sort*
  - otestuje zda je výsledné pole setříděné pomocí funkce `issorted`



wikipedia.org

chcete-li, využijte uvnitř cyklů následující kód:

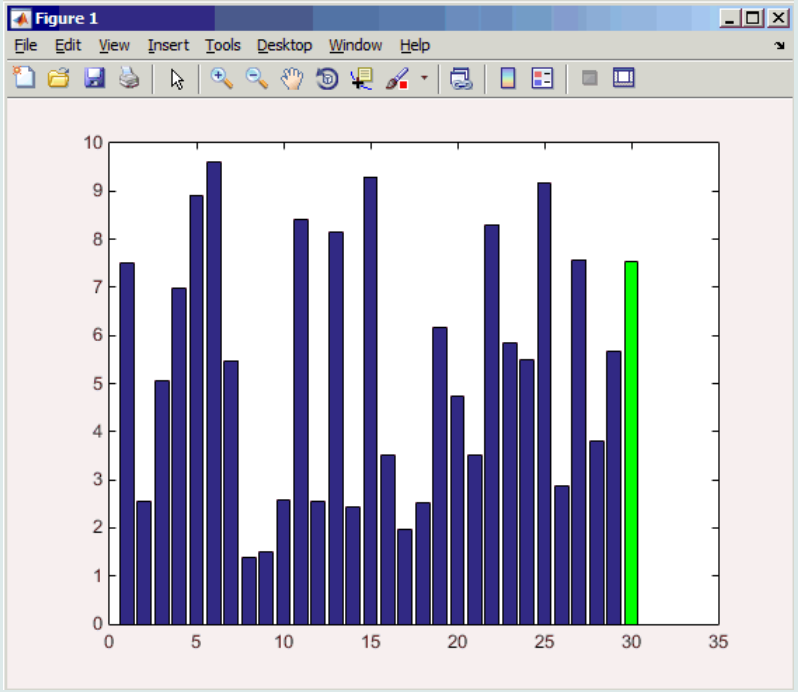
```
figure(1);
plot(R,'*','LineWidth',2);
pause(0.01);
```

```
sort(R)
```

# Třídění polí

600 s ↑

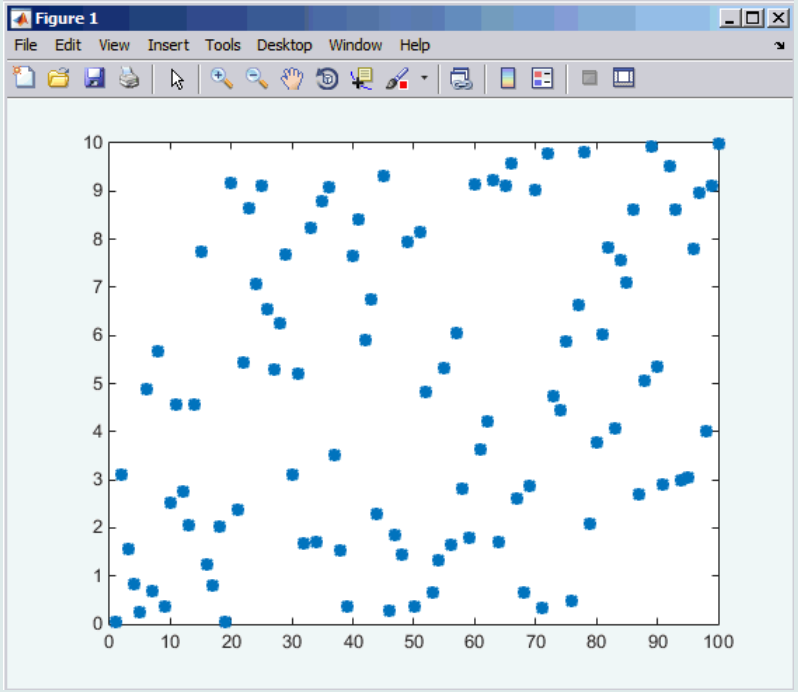
- pomocí příkazu `bar` se pokuste dosáhnout vykreslení jako na obrázku:



# Třídění polí – shaker sort

600 s ↑

- zkuste si naprogramovat vlastní algoritmus třídění `shakerSort.m`
  - využijte třídícího algoritmu *shaker sort*



# Vyhledávání v poli – `find`

- funkce `find` – velice užitečná funkce!!
- vrací pozice ve zkoumané matici, na kterých jsou uložené prvky různé od nuly (hodnoty jsou `true`)
  - užitečné pro hledání v poli logických hodnot
  - příklad: ve vektoru $\mathbf{A} = \left( \frac{\pi}{2} \quad \pi \quad \frac{3}{2}\pi \quad 2\pi \right)$ určete pozice prvků $\mathbf{A} > \pi$

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

  - srovnejte tento příkaz s příkazem `A > pi`. V čem je rozdíl?
- funkce `find` dokáže hledat i ve čtvercové matici atp.
- pro nalezení `k` prvních a posledních nenulových prvků `X`:

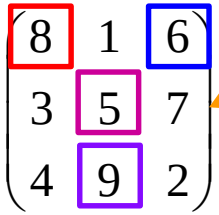
```
>> ind = find(X, k, 'first')
>> ind = find(X, k, 'last')
```

- více viz `>> doc find`

# Pokročilé využití funkce `find`

- lze volat i s více výstupními parametry, což se nám často hodí!

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




pouze první 4 prvky, které splní podmínku

| rw = | cl = |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |
| 1 | 3 |

# Vyhledávání prvků v poli #1

420 s ↑

- vektor $\mathbf{v} = (16 \quad 2 \quad 3 \quad 13 \quad 5 \quad 11 \quad 10 \quad 8 \quad 9 \quad 7 \quad 6 \quad 12 \quad 4 \quad 14 \quad 15 \quad 1)$ setřiďte od největších po nejmenší čísla a najděte, které hodnoty a na kterých pozicích jsou dělitelné 3 a zároveň větší než 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
```

# Vyhledávání prvků v poli #2

300 s ↑

- najděte v matici **w**

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

  poslední 3 hodnoty, které jsou menší než 50
  - určete v jakých sloupcích a řádcích jsou tyto hodnoty

```
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
```

# Použití funkce `find`

600 s ↑

- Vzorky demodulovaného signálu z radiového přijímače se dají popsat touto aproximací:

```
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, '*')
```

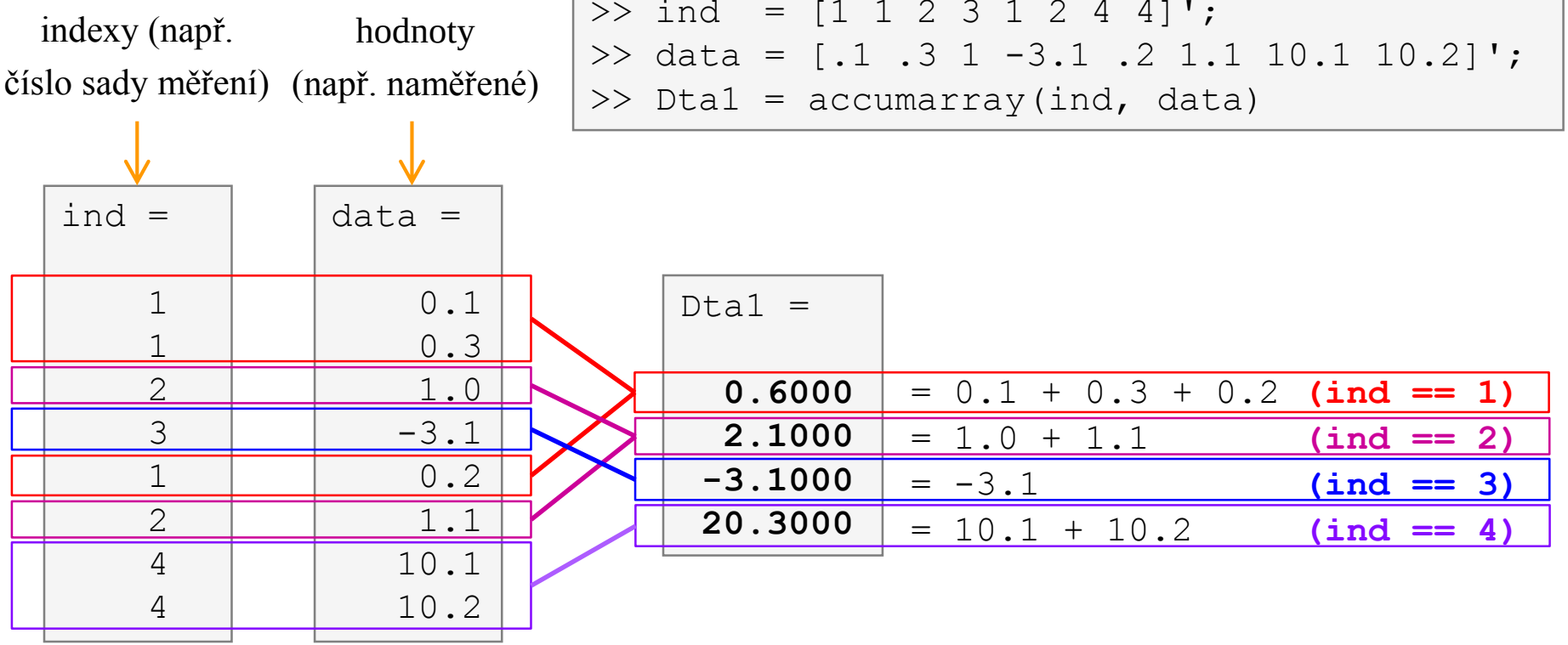
- Zpráva nesená signálem je dána převodní tabulkou:
- Napětí pro znaky platí s tolerancí ±0.5 V
- Dešifrujte zprávu!

| Napětí [V] | Znak |
|---|---|
| 1 | a |
| 2 | c |
| 3 | d |
| 4 | g |
| 5 | m |
| 6 | r |
| 7 | s |

# Funkce accumarray #1

- funkce `accumarray` umí seskupit data se shodným indexem
  - relativně málo známá, ale mimořádně užitečná funkce
- často máme sadu dat (*dataset*), který je organizován např. následovně:

```
>> 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)
```



# Funkce accumarray #2

- základní operací nad všemi prvky z jedné „krabičky“ (data se stejným indexem) je jejich součet
- lze však uvést libovolnou jinou funkci
  - např. maximální prvek z množiny všech prvků se stejným indexem
  - využijeme funkce `max`

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

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

  - např. vyjmenování všech prvků se stejným indexem
  - využijeme tzv. handle funkce a datového typu `cell` (probereme později)

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

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

# Funkce accumarray #3

- funkce má celou řadu dalších možností
- kupř. je možné použít 2D indexace výsledků
  - potom výsledky neřadíme do 1D řady „krabiček“, ale máme jich 2D pole

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

# Funkce accumarray

300 s ↑

- pohyby na účtu v CZK, EUR a USD jsou následující
  - (CZK ~ 1, EUR ~ 2, USD ~ 3)

- určete bilanci v jednotlivých měnách
  - je-li aktuální kurz 28Kč = 1€, 21Kč = 1$, určete celkovou bilanci

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

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

# Funkce v Matlabu

- efektivnější, přehlednější a rychlejší než skripty
- definovaný vstup a výstup, dostupnost komentáře → nezbytná je hlavička funkce
- mohou být volány z pracovního okna, nebo z prostoru jiné funkce (v obou případech musí být funkce dostupná)
- každá funkce má vlastní pracovní prostor, vzniká při zavolání funkce a zaniká s poslední řádkou kódu funkce

# Typy funkcí z hlediska původu

- vestavěné (tzv. built-in)
  - nejsou uživateli přístupné k editaci, lze je pouze volat k výpočtům
  - optimalizované a uložené v jádře
  - zpravidla se jedná o často využívané (= elementární) funkce
- funkce v knihovnách Matlabu (zejm. adresáři [`toolbox`])
  - „tematické“ (problémově zaměřené) funkce
  - některé je možné editovat (nedoporučuje se!)
- funkce vytvořené uživatelem
  - plně přístupné a editovatelné, funkčnost není garantována
  - povinné části: hlavička funkce
  - doporučené součásti funkce: popis funkce, vstupů a výstupů, datum poslední editace, verze, vhodné jsou komentáře

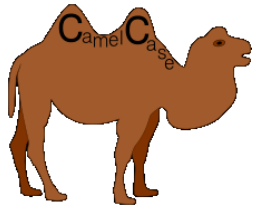
# Hlavička funkce

- musí být první funkční řádek samostatného souboru!
  - funkci nelze psát např. na konec skriptu
- hlavička funkce má následující syntaxi:

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

klíčové slovo — výstupy funkce — název funkce — vstupní parametry funkce

- `jmenoFunkce` musí splňovat stejná pravidla jako jméno proměnné
- `jmenoFunkce` se nesmí shodovat s žádným názvem její proměnné
- `jmenoFunkce` zpravidla píšeme pomocí `lowerCamelCase` nebo s podtržítkem (`my_function`)



# Hlavička funkce – příklady

```
function functA
%FUNCTA - neobvykle, byť možné, bez vstupu a výstupu
```

```
function functB(parIn1)
%FUNCTB - např. funkce s GUI výstupem, tiskem apod.
```

```
function parOut1 = functC
%FUNCTC - příprava dat, pseudonáhodných dat atd.
```

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

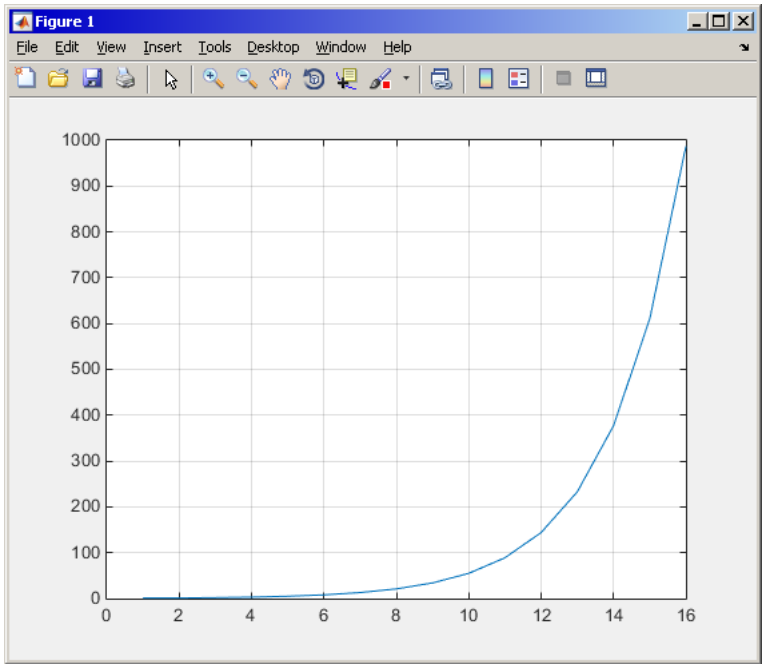
```
function parOut1 = functE(parIn1, parIn2)
%FUNCTE - plnohodnotná funkce, závorky [] být nemusí
```

```
function [parOut1, parOut2] = functF(parIn1, parIn2)
%FUNCTF - plnohodnotná funkce s více parametry
```

# Volání funkce

```
>> f = fibonacci(1000);  % volání funkce z příkazové řádky
>> 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 sekvenčně vykonává příkazy
  - vstupní parametr: `limit`
  - výstupní proměnná: Fibonacciho řada `f`
  - nedostatky:
    - není ošetřený vstup (lze zadat v principu cokoliv)
    - není alokována velikost matice `f`, tj. matice stále roste (pomalé)

# Jednoduchý příklad funkce

- jakoukoliv funkci v Matlabu lze volat s méně vstupními parametry než je celkový počet v hlavičce
- jakoukoliv funkci v Matlabu lze volat pro méně výstupních parametrů než je celkový počet v hlavičce
  - např. máme funkci s hlavičkou:

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

  - potom např. všechna následující volání jsou v pořádku

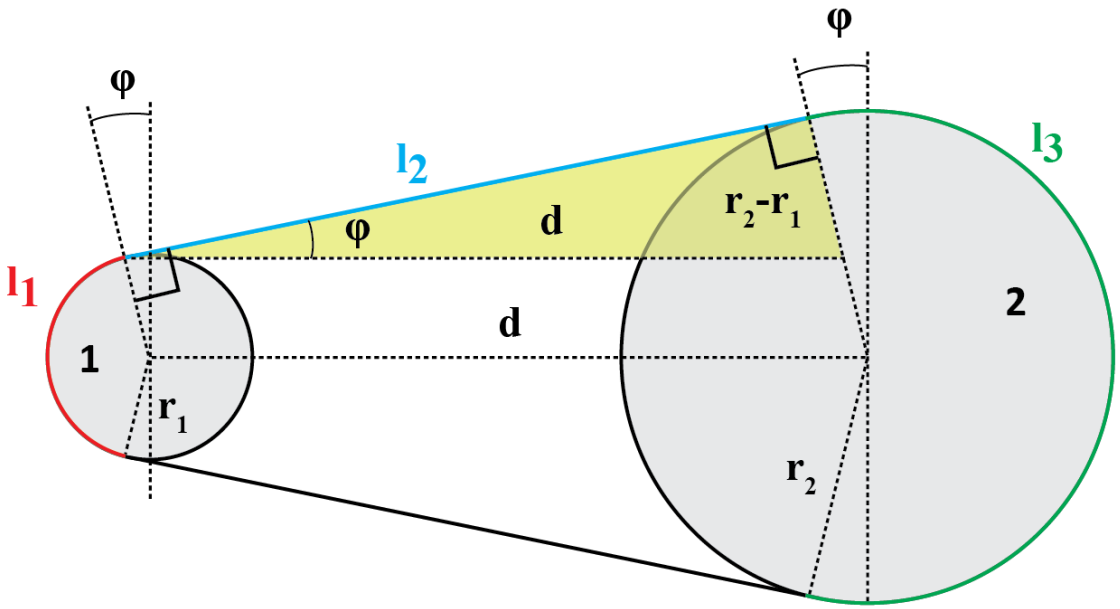
```
>> [parO1, parO2]          = functG(pIn1, pIn2, pIn3)
>> [parO1, parO2, parO3] = functG(pIn1)
>> functG(pIn1,pIn2,pIn3)
>> [parO1, parO2, par03] = functG(pIn1, pIn2, pIn3)
>> [parO1, ~, par03] = functG(pIn1, [], pIn3)
>> [~, ~, par03] = functG(pIn1, [], [])
```

# Jednoduchý příklad funkce

100 s ↑

- navrhněte funkci, která vypočte délku řemenice mezi dvěma koly
  - zadané jsou průměry obou kol a jejich vzdálenost (= vstupy funkce)

  - připravte si náčrtek, analyzujte situaci, co potřebujete spočítat?
  - otestujte funkci pro některé scénáře, ověřte výsledky
  - funkci řádně zdokumentujte, vyzkoušejte příkazy `lookfor`, `help`

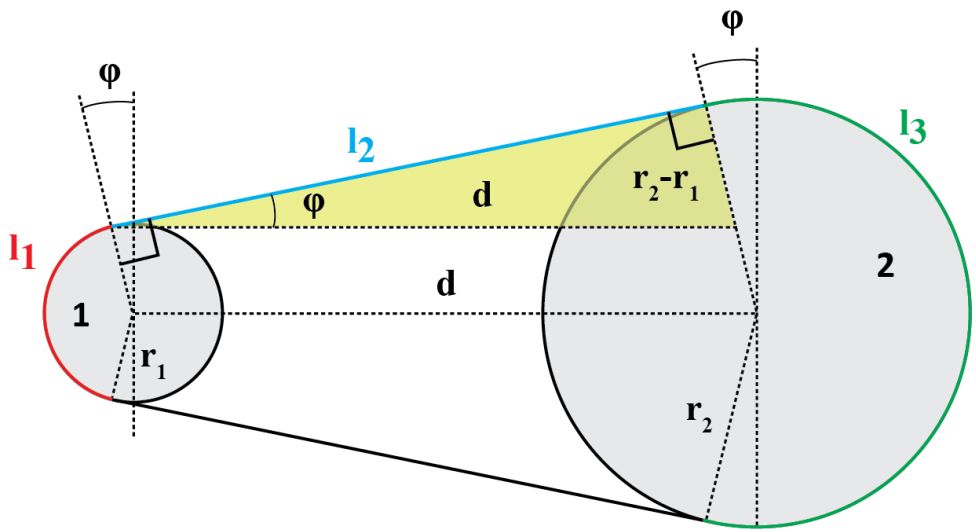


# Jednoduchý příklad funkce

500 s ↑

- celková délka je $l = l_1 + 2l_2 + l_3$
- zadané jsou průměry → přepočet na poloměry
- pro stanovené délky lze využít pravoúhlý trojúhelník:

- úhel φ lze vypočítat obdobně:
- a konečně:

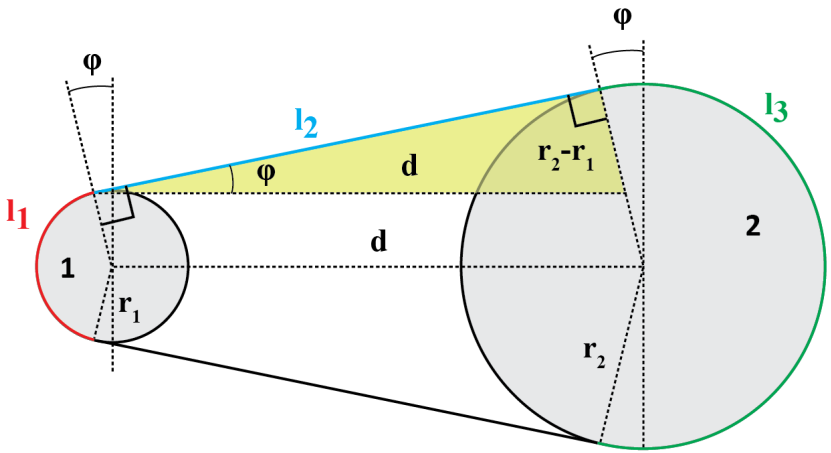



- řešení otestujte například pro:

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

# Jednoduchý příklad funkce

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



# Komentáře uvnitř funkce

nápověda pro funkci, zobrazena na dotaz:
`>> help myFcn1`

1. řádka (tzv. H1 řádka), tuto řádku vyhledává příkaz `lookfor`. Zpravidla obsahuje jméno funkce velkými písmeny a stručný popis účelu funkce.

```
function [dataOut, idx] = myFcn1(dataIn, method)
%MYFCN1: Umožňuje vypočítat cosi...
% syntax, popis vstupu, vystupu,
% priklady volani, autor, verze
% dalsi podobne funkce, dalsi casti napovedy

matX = dataIn(:, 1);
sumX = sum(matX); % sečtení členů
%% zde je výpis výsledku:
disp(num2str(sumX));
```

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

KOMENTUJTE!

% Komentáře zásadním způsobem

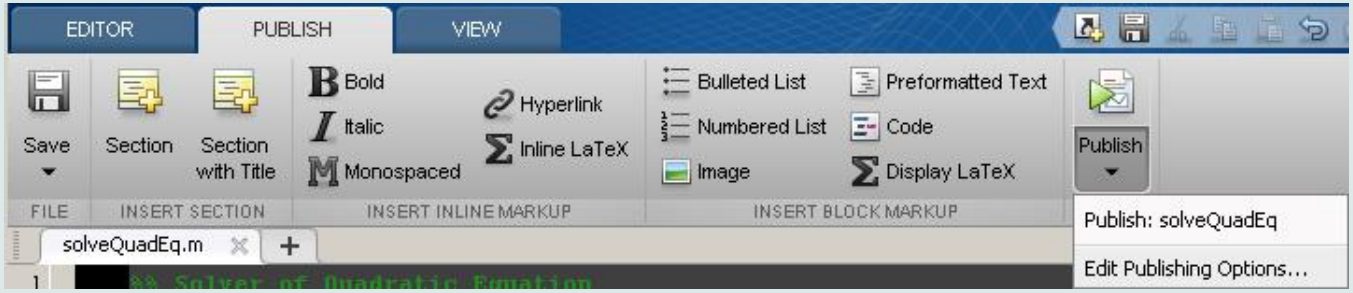
% zvyšují srozumitelnost funkce!!!

# Dokumentace funkce – příklad

```
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
```

# Funkce `publish`

- slouží k vytvoření dokumentace ke skriptu, funkci či třídě
- poskytuje několik výstupních formátů (html, doc, ppt, LaTeX, ...)
- vytváření nápovědy (`>> doc my_fun`) přímo komentáři v kódu!
  - umožňuje široké možnosti formátování (nadpisy, číslované seznamy, rovnice, vkládání obrázků, odkazy, ...)
- umožňuje vkládat printscreeny grafických oken do dokumentace
  - dokumentovaný kód se při publikování implicitně spustí
- podpora vytváření dokumentace přímo z menu editoru:



# Funkce `publish` - příklad

```
%% 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!

```
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.

# Probrané funkce

| | | |
|---|---|---|
| `intersect` | průnik prvků množin (vektorů / matic) | |
| `union` | sjednocení prvků množin (vektorů / matic) | |
| `setdiff` | odečet množin (resp. průnik množiny a doplňku druhé množiny) | |
| `setxor` | exkluzivní průnik | |
| `unique` | výběr unikátních prvků z polí | |
| `sort` | třídění prvků vektoru / matice | |
| `sortrows` | třídění matice, řádky jsou tříděny vcelku | |
| `accumarray` | shromáždí data | ● |
| `ismember` | je daný prvek členem pole? | |
| `issorted` | je dané pole setříděno? | |
| `find` | hledání prvků splňující danou podmínku | ● |
| `function` | hlavička funkce | ● |

# Cvičení #1

600 s ↑

- asymptoticky rozveďte exponenciálu do Taylorovy řady:
  - v tomto případě jde de facto o McLaurinovu řadu (rozvoj kolem $x = 0$)

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

  - porovnejte se skutečným výsledkem získaným pomocí `exp(x)`
  - stanovte odchylku v [%] (co bude základ, tedy 100% ?)
  - najděte takový řád aproximace, kdy je chyba menší než 1%

  - implementujte jako skript, zadejte:
    $x$ (argument funkce)
    $N$ (nejvyšší mocnina řady)

# Cvičení #2

600 s ↑

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

- přepište na funkci
  - zvolte adekvátní název funkce
  - vstupní parametry funkce jsou `x` a `n`
  - výstupním parametrem jsou hodnoty `f1`, `f2` a `err`
  - doplňte funkci vhodným komentářem (H1 řádka, vstupy, výstupy)

  - funkci otestujte

# Cvičení #3

600 s ↑

- k této funkci vytvořte skript, který ji několikrát volá (pro různá `n`)
  - stanovte přesnost aproximace pro $x = 0.9, \quad n \in \{1,\ldots,10\}$
  - výsledný průběh přesnosti vykreslete (chyba v závislosti na $n$)

# Cvičení #4

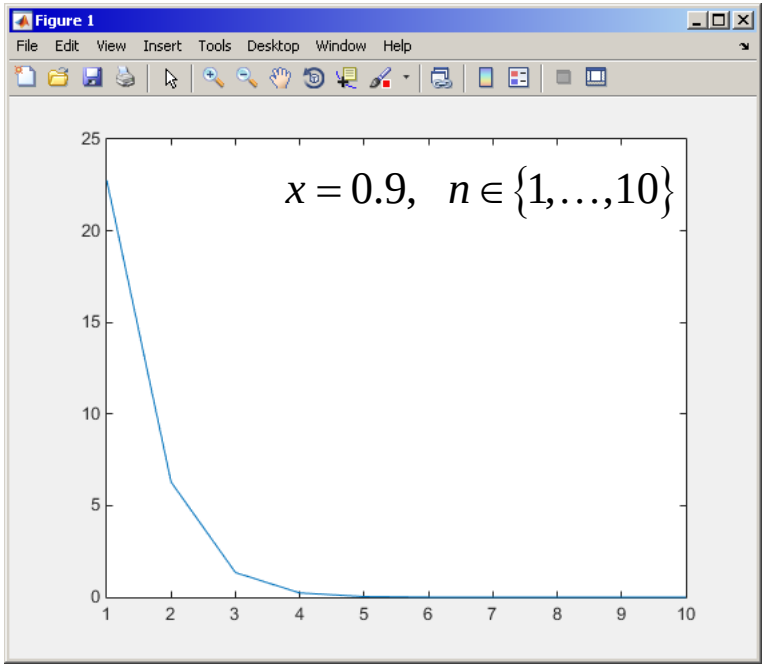


Figure 1
File Edit View Insert Tools Desktop Window Help
$x = 0.9, \quad n \in \{1,\ldots,10\}$
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10

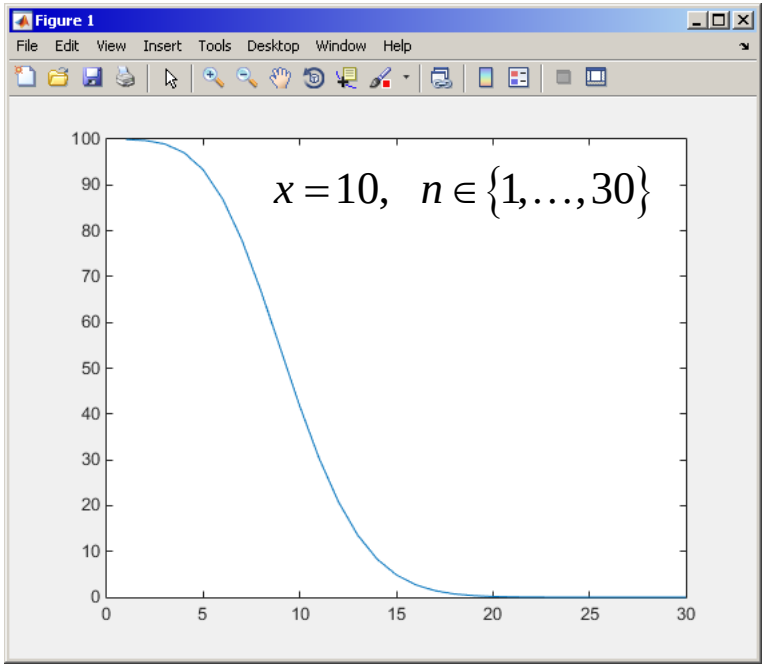


Figure 1
File Edit View Insert Tools Desktop Window Help
$x = 10, \quad n \in \{1,\ldots,30\}$
100
90
80
70
60
50
40
30
20
10
0
0 5 10 15 20 25 30

# Cvičení #5

- po dobu 5 dnů probíhalo měření teploty, a to vždy každou 2. celou hodinu, data byla měřena na 3 rozdílných místech (A, B, C)

- zjistěte průměrnou denní teplotu v daném týdnu pro všechna 3 místa
  - tj. udělejte aritmetický průměr pro měření ve stejnou hodinu na stejném místě

- data si vygenerujte skriptem `temperature_measurement.m`
  - skript je přiložen i na následujícím slajdu
  - dále jsou popsány potřebné proměnné

# Cvičení #6

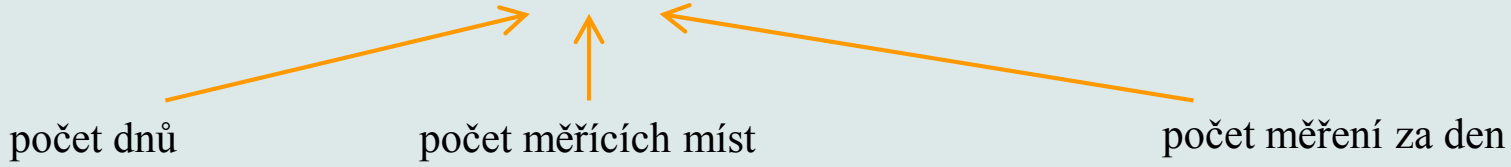
skript vygeneruje data

```
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');
```

výsledky pro Vás…

# Cvičení #7

- všechna data jsou obsažena ve 2 maticích:
  - `TimeAndPlace (5×3×12,2) = (180,2)`
  - `MeasuredData (5×3×12,1) = (180,1)`




- bohužel, data v `TimeAndPlace` nejsou úmyslně setříděna

| **INDEXY:** | `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 |
| | ... ... | ... | |

# Cvičení #8

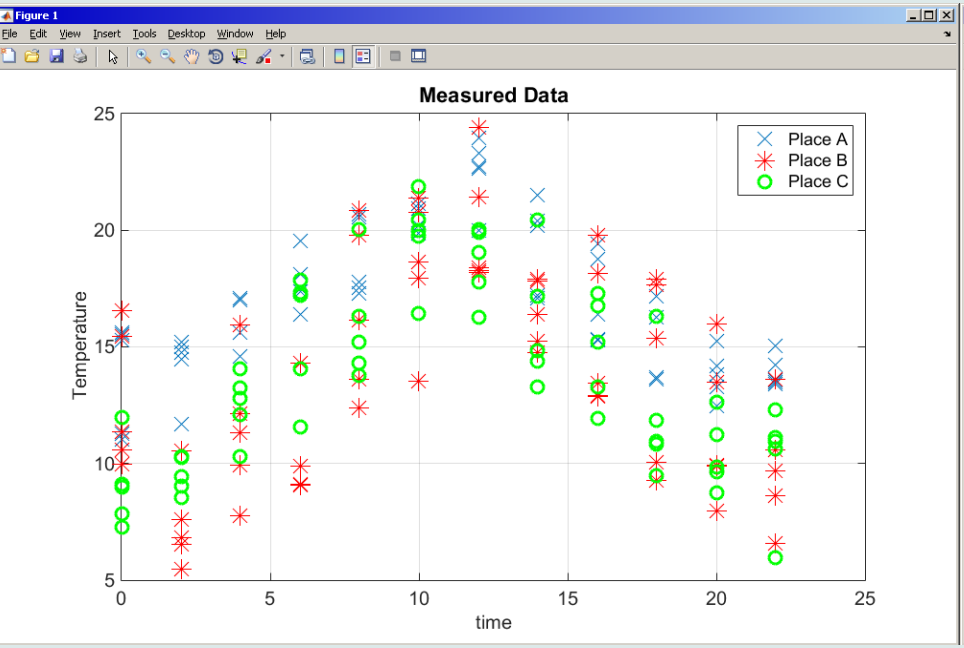
600 s ↑

- dále platí, že
  - Place1 ~ měřící stanoviště A,
  - Place2 ~ měřící stanoviště B
  - Place3 ~ měřící stanoviště C
  - hodina měření = 2*(tindex-1)

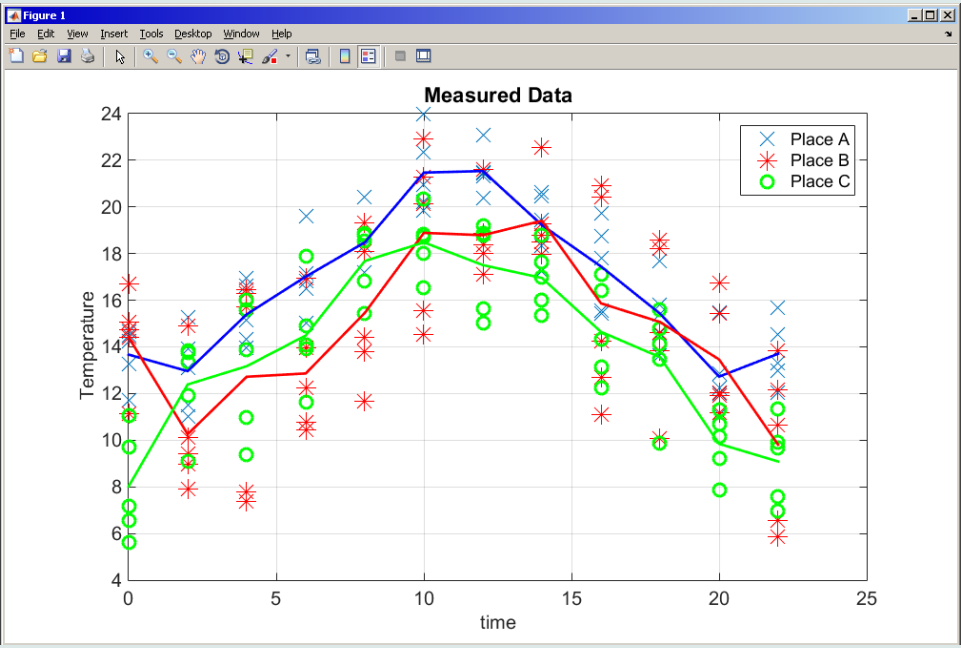
- nyní se pokuste do skriptu dopsat Váš kód který provede zprůměrování a zakreslete data do existujícího grafu

```
%% 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','-');
```

# Cvičení #9




naměřená data




naměřená data s průměry

# Děkuji!


ver. 3.7 (31/03/2014)
Miloslav Čapek
miloslav.capek@fel.cvut.cz