



TECHNICKÁ UNIVERZITA V LIBERCI
**Fakulta mechatroniky, informatiky
a mezioborových studií**



MATLB: přednáška 2

Datové typy a struktury

Jaroslav Čmejla

Část I

Vyhledávání a logické indexování

Logické indexování

- Výsledkem logické operace je proměnná typu `logical` (obdoba typu `boolean`)
- Logickou proměnnou lze použít k indexování. Např.

```
>> s=[1 2 3 4]
s =
     1     2     3     4
>> a=s>2
a =
     0     0     1     1
>> s(a)
ans =
     3     4
```

- Složené výrazy: efektní a krátký kód. Např.

```
>> s(s>2)=5
s =
     1     2     5     5
```

Logické indexování

- Výsledkem logické operace je proměnná typu `logical` (obdoba typu `boolean`)
- Logickou proměnnou lze použít k indexování. Např.

```
>> s=[1 2 3 4]
s =
     1     2     3     4
>> a=s>2
a =
     0     0     1     1
>> s(a)
ans =
     3     4
```

- Složené výrazy: efektní a krátký kód. Např.

```
>> s(s>2)=5
s =
     1     2     5     5
```

Logické indexování

- Výsledkem logické operace je proměnná typu `logical` (obdoba typu `boolean`)
- Logickou proměnnou lze použít k indexování. Např.

```
>> s=[1 2 3 4]
s =
     1     2     3     4
>> a=s>2
a =
     0     0     1     1
>> s(a)
ans =
     3     4
```

- Složené výrazy: efektní a krátký kód. Např.

```
>> s(s>2)=5
s =
     1     2     5     5
```

Logické indexování

- Při logickém indexování, jako je výraz

$s(s > 2)$,

zadáme v podstatě jen jeden indexující parametr.

- Platí tedy pravidla pro výsledek indexování jediným parametrem. Výsledkem je sloupcový vektor, ve kterém jsou výsledné prvky seřazené podle původního pořadí v paměti (výjimkou jsou řádkové vektory). Například

```
>> A=randn(3);
```

```
>> A(A>0.5)
```

```
ans =
```

```
1.0115
```

```
0.5542
```

```
0.7589
```

```
1.1449
```

Logické indexování

- Při logickém indexování, jako je výraz

$s(s > 2)$,

zadááme v podstatě jen jeden indexující parametr.

- Platí tedy pravidla pro výsledek indexování jediným parametrem. Výsledkem je sloupcový vektor, ve kterém jsou výsledné prvky seřazené podle původního pořadí v paměti (výjimkou jsou řádkové vektory). Například

```
>> A=randn(3);
```

```
>> A(A>0.5)
```

```
ans =
```

```
1.0115
```

```
0.5542
```

```
0.7589
```

```
1.1449
```

Logické indexování - další příklady

■ Umocnění záporných prvků na druhou

```
>> s=randn(1,5)
```

```
s =
```

```
1.0347    0.7269   -0.3034    0.2939   -0.7873
```

```
>> s(s<0)=s(s<0).^2 % pravá strana musí mít stejný rozměr  
% jako levá
```

```
s =
```

```
1.0347    0.7269    0.0921    0.2939    0.6198
```

■ Složitější logický výraz: vrací prvky $s \in (-1, 3) \cup (9, 16)$

```
>> s(((s>-1)&(s<3))|(((s>9)&(s<16))))
```


Logické indexování - další příklady

- Umocnění záporných prvků na druhou

```
>> s=randn(1,5)
```

```
s =
```

```
1.0347    0.7269   -0.3034    0.2939   -0.7873
```

```
>> s(s<0)=s(s<0).^2 % pravá strana musí mít stejný rozměr  
% jako levá
```

```
s =
```

```
1.0347    0.7269    0.0921    0.2939    0.6198
```

- Složitější logický výraz: vrací prvky $s \in (-1, 3) \cup (9, 16)$

```
>> s(((s>-1)&(s<3))|(((s>9)&(s<16))))
```

Logické indexování - další příklady

■ Co nefunguje:

```
>> s([1 0 0 1 1]) % ručně zadané [1 0 0 1 1] není  
                    % logickou hodnotou nýbrž double  
??? Subscript indices must either be real positive integers  
    or logicals.
```

```
>> s([1 0 0 1 1]==1) % takto to lze obejít  
ans =  
    1.0347    0.2939    0.6198
```

```
>> s(logical([1 0 0 1 1])) % nebo přetypováním  
ans =  
    1.0347    0.2939    0.6198
```

Vyhledávání prvků pomocí `find` a indexování

■ Příkaz `find` vrací indexy nenulových prvků

```
>> s=[1 0 10 0];  
>> find(s)
```

```
ans =  
     1     3  
>> find(s==10)
```

```
ans =  
     3
```

■ Indexování pomocí `find`

```
>> s(find(s==10)) % lze nahradit logickým  
                  % indexováním s(s==10)  
  
ans =  
    10
```

Vyhledávání prvků pomocí `find` a indexování

- Příkaz `find` vrací indexy nenulových prvků

```
>> s=[1 0 10 0];  
>> find(s)
```

```
ans =  
     1     3  
>> find(s==10)
```

```
ans =  
     3
```

- Indexování pomocí `find`

```
>> s(find(s==10)) % lze nahradit logickým  
                  % indexováním s(s==10)
```

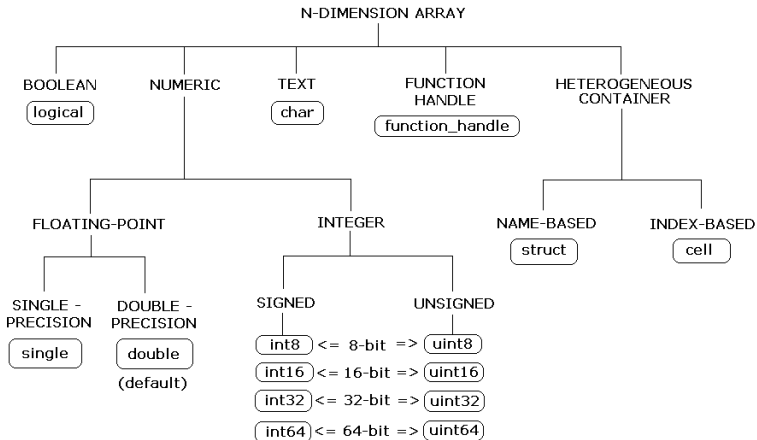
```
ans =  
    10
```

Část II

Datové typy a struktury

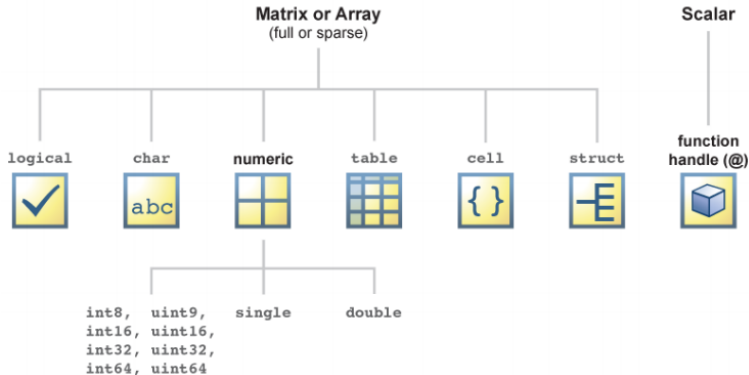
Datové typy a struktury

Přehled typů: Matlab R2011



Datové typy a struktury

Přehled typů: Matlab R2013b



Numerické a logické datové typy

■ Numerické: `int8`, `uint8`, `int16`, `single`, `double` ...

■ Přetypování pomocí příkazů stejného názvu. Např.

```
>> a=randn(1,5)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> a=single(a)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x5	20	single	

■ Logické typy: výstupy logických výrazů, typ `logical`

Numerické a logické datové typy

■ Numerické: int8, uint8, int16, single, double ...

■ Přetypování pomocí příkazů stejného názvu. Např.

```
>> a=randn(1,5)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> a=single(a)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x5	20	single	

■ Logické typy: výstupy logických výrazů, typ logical

Numerické a logické datové typy

- Numerické: int8, uint8, int16, single, double ...
- Přetypování pomocí příkazů stejného názvu. Např.

```
>> a=randn(1,5)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> a=single(a)
```

```
a =
```

```
    -1.3499    3.0349    0.7254   -0.0631    0.7147
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x5	20	single	

- Logické typy: výstupy logických výrazů, typ logical

Vícerozměrná pole

■ Vícerozměrné pole typu double

```
>> A=randn(3,4,5,2);
```

```
>> A(2,3, :, 1) % indexování jako u matic
```

```
>> size(A(2,3, :, :))
```

```
ans =
```

```
1      1      5      2
```

Vícerozměrná pole

■ Příkazy sum, mean, prod

```
>> sum(v) % součet 1D pole = vektoru  
           % je jedno jestli je řádkový nebo sloupcový
```

```
>> sum(A) % součet sloupců
```

```
>> sum(A,n) % součet přes n-tý rozměr
```

```
>> size(sum(A,3)) % n-tý rozměr výsledku je potom 1
```

```
ans =
```

```
      3      4      1      2
```

```
>> sum(A(:)) % součet všech prvků
```

Vícerozměrná pole

■ Příkaz squeeze odstraní 1-dimenzionální (singleton) rozměry

```
>> size(sum(sum(A,3),2))  
ans =  
      3      1      1      2  
>> squeeze(sum(sum(A,3),2))  
ans =  
  
    1.3769    -4.0844  
    0.8480    -0.8979  
    2.9229     2.0747
```

■ Příkaz reshape mění rozměry pole

```
>> reshape(A,12,10) % změna rozměrů  
    % počet prvků nového pole musí být stejný  
    % pořadí prvků v paměti je zachováno
```

Vícerozměrná pole

- Příkaz `squeeze` odstraní 1-dimenzionální (singleton) rozměry

```
>> size(sum(sum(A,3),2))  
ans =  
      3      1      1      2  
  
>> squeeze(sum(sum(A,3),2))  
ans =  
  
    1.3769    -4.0844  
    0.8480    -0.8979  
    2.9229     2.0747
```

- Příkaz `reshape` mění rozměry pole

```
>> reshape(A,12,10) % změna rozměrů  
    % počet prvků nového pole musí být stejný  
    % pořadí prvků v paměti je zachováno
```

Vícerozměrná pole

■ Příkaz permute - výměna pořadí rozměrů

```
>> B=randn(2)
```

```
B =
```

```
    0.0335    1.1275  
   -1.3337    0.3502
```

```
>> permute(B,[2 1]) % de facto transpozice
```

```
ans =
```

```
    0.0335   -1.3337  
    1.1275    0.3502
```

```
>> permute(A,[3 4 1 2])
```

■ Příkaz repmat - dlaždicové rozšíření pole

```
>> repmat(B,1,2)
```

```
ans =
```

```
    0.0335    1.1275    0.0335    1.1275  
   -1.3337    0.3502   -1.3337    0.3502
```

Vícerozměrná pole

■ Příkaz permute - výměna pořadí rozměrů

```
>> B=randn(2)
```

```
B =
```

```
    0.0335    1.1275  
   -1.3337    0.3502
```

```
>> permute(B,[2 1]) % de facto transpozice
```

```
ans =
```

```
    0.0335   -1.3337  
    1.1275    0.3502
```

```
>> permute(A,[3 4 1 2])
```

■ Příkaz repmat - dlaždicové rozšíření pole

```
>> repmat(B,1,2)
```

```
ans =
```

```
    0.0335    1.1275    0.0335    1.1275  
   -1.3337    0.3502   -1.3337    0.3502
```


Vícerozměrná pole

■ Příkaz `arrayfun` aplikuje funkci na každý element pole

```
>> C=arrayfun(@sin,A); % zde ve skutečnosti arrayfun  
                        % nepotřebujeme  
                        % stejné jako sin(A)
```

```
>> files=dir;
```

```
>> length(files.name) % délky názvů souborů  
                        % takto to nejde
```

```
Error using length  
Too many input arguments.
```

```
>> arrayfun(@(x)length(x.name),files) % délky názvů souborů  
                                        % takto ano
```

Vícerozměrná pole

- Příkaz `bsxfun` - aplikuje po prvcích zadanou binární operaci. Je-li některý z rozměrů jednotkový (singleton), rozšíří se rozměr na stejný rozměr jako má druhé pole (jako pomocí funkce `repmat`).

```
>> B=[1 1 1; 1 2 3; -1 -2 -3]
```

```
B =
```

1	1	1
1	2	3
-1	-2	-3

```
>> bsxfun(@minus,B,mean(B,2)) % odečte z každého řádku  
                             % jeho průměrnou hodnotu
```

```
ans =
```

0	0	0
-1	0	1
1	0	-1

Znaky a řetězce

■ Znak je typu char

■ Řetězec je:

- pole typů char pracujeme s ním tedy podobně jako s maticemi (spojování, indexování, atd.)
- pole typu string (od R2016)

```
>> s='Ahoj'
```

```
s =
```

```
Ahoj
```

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x4	8	char	

```
>> s(2)
```

```
ans =
```

```
h
```

Znaky a řetězce

■ Znak je typu char

■ Řetězec je:

- pole typů char pracujeme s ním tedy podobně jako s maticemi (spojování, indexování, atd.)
- pole typu string (od R2016)

```
>> s='Ahoj'
```

```
s =
```

```
Ahoj
```

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x4	8	char	

```
>> s(2)
```

```
ans =
```

```
h
```

Znaky a řetězce

- Znak je typu char
- Řetězec je:
 - pole typů char pracujeme s ním tedy podobně jako s maticemi (spojování, indexování, atd.)
 - pole typu string (od R2016)

```
>> s='Ahoj'
```

```
s =
```

```
Ahoj
```

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x4	8	char	

```
>> s(2)
```

```
ans =
```

```
h
```

Znaky a řetězce

- Znak je typu char
- Řetězec je:
 - pole typů char pracujeme s ním tedy podobně jako s maticemi (spojování, indexování, atd.)
 - pole typu string (od R2016)

```
>> s='Ahoj'
```

```
s =
```

```
Ahoj
```

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x4	8	char	

```
>> s(2)
```

```
ans =
```

```
h
```

Řetězce

```
>> s=[s ' Honzo'] % spojování do řádku
```

```
s =
```

```
Ahoj Honzo
```

```
>> a=['Ahoj';'Honzo'] % spojování pod sebe: chyba - řádky matice  
nejsou stejně velké
```

```
??? Error using ==> vertcat
```

```
CAT arguments dimensions are not consistent.
```

```
>> a=['Ahoj ','Honzo'] % spojování pod sebe: zde správně
```

```
a =
```

```
Ahoj
```

```
Honzo
```

Standardní příkazy pro práci s řetězci

■ strcmp - porovnávání

```
>> strcmp(a(1,:), 'Ahoj ') % porovnání řetězců  
ans =  
1
```

- strfind, regexp - vyhledávání, regulární výrazy, parsování
- fprintf, sprintf - formátované výrazy
- upper, lower - velká/malá písmena
- eval - vyhodnocení výrazu

Standardní příkazy pro práci s řetězci

■ strcmp - porovnávání

```
>> strcmp(a(1,:), 'Ahoj ') % porovnání řetězců  
ans =  
1
```

■ strfind, regexp - vyhledávání, regulární výrazy, parsování

■ fprintf, sprintf - formátované výrazy

■ upper, lower - velká/malá písmena

■ eval - vyhodnocení výrazu

Standardní příkazy pro práci s řetězci

- strcmp - porovnávání

```
>> strcmp(a(1,:), 'Ahoj ') % porovnání řetězců  
ans =  
    1
```

- strfind, regexp - vyhledávání, regulární výrazy, parsování

- fprintf, sprintf - formátované výrazy

- upper, lower - velká/malá písmena

- eval - vyhodnocení výrazu

Standardní příkazy pro práci s řetězci

- strcmp - porovnávání

```
>> strcmp(a(1,:), 'Ahoj ') % porovnání řetězců  
ans =  
1
```

- strfind, regexp - vyhledávání, regulární výrazy, parsování
- fprintf, sprintf - formátované výrazy
- upper, lower - velká/malá písmena
- eval - vyhodnocení výrazu

Standardní příkazy pro práci s řetězci

- strcmp - porovnávání

```
>> strcmp(a(1,:), 'Ahoj ') % porovnání řetězců  
ans =  
1
```

- strfind, regexp - vyhledávání, regulární výrazy, parsování
- fprintf, sprintf - formátované výrazy
- upper, lower - velká/malá písmena
- eval - vyhodnocení výrazu

Záznamy

■ Datový typ struct

■ Automatická definice položek

```
>> s.jmeno='Tomas';  
>> s.adresa='Praha';
```

% nebo

```
>> s=struct('jmeno','Tomas','adresa','Praha')  
s =  
    jmeno: 'Tomas'  
    adresa: 'Praha'
```

■ Každý prvek je zároveň pole

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x1	268	struct	

Záznamy

■ Datový typ struct

■ Automatická definice položek

```
>> s.jmeno='Tomas';  
>> s.adresa='Praha';
```

% nebo

```
>> s=struct('jmeno','Tomas','adresa','Praha')  
s =  
    jmeno: 'Tomas'  
    adresa: 'Praha'
```

■ Každý prvek je zároveň pole

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x1	268	struct	

Záznamy

- Datový typ struct
- Automatická definice položek

```
>> s.jmeno='Tomas';  
>> s.adresa='Praha';
```

% nebo

```
>> s=struct('jmeno','Tomas','adresa','Praha')  
s =  
    jmeno: 'Tomas'  
    adresa: 'Praha'
```

- Každý prvek je zároveň pole

```
>> whos s
```

Name	Size	Bytes	Class	Attributes
s	1x1	268	struct	

Pole záznamů

```
>> s(2).jmeno='Ales'  
s =  
1x2 struct array with fields:  
    jmeno  
    adresa  
  
>> s(2)  
  
ans =  
    jmeno: 'Ales'  
    adresa: []
```


Záznamy - automatické doplňování

```
>> s(2).vaha=70 % nový záznam
```

```
s =
```

```
1x2 struct array with fields:
```

```
    jmeno
```

```
    adresa
```

```
    vaha
```

```
>> s(1).vaha % v prvním prvku byl záznam automaticky vytvořen
```

```
ans =
```

```
[]
```

```
>> s(1).vaha='velka';
```

```
>> s.vaha % záznam vaha může mít v každém prvku jiný typ
```

```
ans =
```

```
velka
```

```
ans =
```

```
3
```

Příklad: výstup příkazu `dir`

```
>> files=dir('*.m')  
files =  
6x1 struct array with fields:  
    name  
    date  
    bytes  
    isdir  
    datenum  
  
>> files(1)  
ans =  
    name: 'dft.m'  
    date: '05-XI-2009 10:59:17'  
    bytes: 143  
    isdir: 0  
    datenum: 7.3408e+005
```

Příklad: výstup příkazu `dir`

```
>> files(1:3).name
```

```
ans =
```

```
dft.m
```

```
ans =
```

```
funkcef.m
```

```
ans =
```

```
pokus.m
```

```
>> [files(1:3).name]
```

```
ans =
```

```
dft.mfunkcef.mpokus.m
```

```
>> strvcat(files(1:3).name)
```

```
ans =
```

```
dft.m
```

```
funkcef.m
```

```
pokus.m
```

Příklad: výstup příkazu `dir`

```
>> files=dir  
files =  
131x1 struct array with fields:  
    name  
    date  
    bytes  
    isdir  
    datenum
```

```
>> directory=files([files.isdir]) % pouze adresáře  
directory =  
7x1 struct array with fields:  
    name  
    date  
    bytes  
    isdir  
    datenum
```

Pole paměťových buněk - Cell Arrays

- Paměťová buňka - `cell`: obsahuje libovolnou datovou strukturu libovolné velikosti
- Pole paměťových buněk: nejobecnější možné indexované pole
- Indexování pomocí složených závorek `{, }`

```
>> c{1,1}=randn(3,4);  
>> c{1,2}=struct('jmeno','Ales');  
>> c{1,3}=c
```

c =

[3x4 double] [1x1 struct] {1x2 cell}

Pole paměťových buněk - Cell Arrays

- Paměťová buňka - `cell`: obsahuje libovolnou datovou strukturu libovolné velikosti
- Pole paměťových buněk: nejobecnější možné indexované pole
- Indexování pomocí složených závorek `{, }`

```
>> c{1,1}=randn(3,4);  
>> c{1,2}=struct('jmeno','Ales');  
>> c{1,3}=c
```

c =

[3x4 double]

[1x1 struct]

{1x2 cell}

Pole paměťových buněk - Cell Arrays

- Paměťová buňka - `cell`: obsahuje libovolnou datovou strukturu libovolné velikosti
- Pole paměťových buněk: nejobecnější možné indexované pole
- Indexování pomocí složených závorek `{, }`

```
>> c{1,1}=randn(3,4);  
>> c{1,2}=struct('jmeno','Ales');  
>> c{1,3}=c
```

`c =`

`[3x4 double] [1x1 struct] {1x2 cell}`

Pole paměťových buněk - Cell Arrays

■ Indexování pomocí {, }

```
>> c(2) % vrací paměťové pole 1x1 na 2. pozici v c  
>> c{2} % vrací obsah 2. prvku v c  
>> c{1}(2,3) % (2,3)-prvek v 1. prvku c  
>> c{3}{1,2}.jmeno(3) % no comment
```

```
ans =  
e
```

■ Spojování: {, } zachovávají úroveň vnoření - rozdíl od [,]

```
>> {{2} {3 4}}
```

```
ans =  
      {1x1 cell}      {1x2 cell}
```

```
>> [[2] [3 4]]
```

```
ans =  
      2      3      4
```


Pole paměťových buněk - Cell Arrays

■ Indexování pomocí {, }

```
>> c(2) % vrací paměťové pole 1x1 na 2. pozici v c  
>> c{2} % vrací obsah 2. prvku v c  
>> c{1}(2,3) % (2,3)-prvek v 1. prvku c  
>> c{3}{1,2}.jmeno(3) % no comment
```

```
ans =  
e
```

■ Spojování: {, } zachovávají úroveň vnoření - rozdíl od [,]

```
>> {{2} {3 4}}  
ans =  
    {1x1 cell}    {1x2 cell}
```

```
>> [[2] [3 4]]  
ans =  
    2        3        4
```

Tabulky

- Tabulka (`table`) je datová struktura skládající se ze sloupců a řádků. V každém sloupci je libovolný typ (v rámci sloupce však stejný). Všechny sloupce mají stejný počet řádků.

```
>> a=table(randn(2,2), [3;5], randn(2,2))
```

```
a =
```

	Var1		Var2		Var3
	0.27607	0.44342	3	-1.2507	-0.74111
	-0.26116	0.39189	5	-0.94796	-0.50782

```
>> a(2:3,4:6) % podtabulka tabulky a
```

```
>> a{1,2} % obsah prvku tabulky
```

- Tabulka je ve skutečnosti třída (`class`) - vrátíme se k tématu až budeme zmiňovat možnosti objektově orientovaného programování (OOP) v Matlabu

Tabulky

- Tabulka (table) je datová struktura skládající se ze sloupců a řádků. V každém sloupci je libovolný typ (v rámci sloupce však stejný). Všechny sloupce mají stejný počet řádků.

```
>> a=table(randn(2,2), [3;5], randn(2,2))
```

```
a =
```

	Var1		Var2		Var3
	0.27607	0.44342	3	-1.2507	-0.74111
	-0.26116	0.39189	5	-0.94796	-0.50782

```
>> a(2:3,4:6) % podtabulka tabulky a
```

```
>> a{1,2} % obsah prvku tabulky
```

- Tabulka je ve skutečnosti třída (class) - vrátíme se k tématu až budeme zmiňovat možnosti objektově orientovaného programování (OOP) v Matlabu

Výčtové typy

- Typ `categorical` - výčtový typ nabývající hodnot z konečné množiny

- Příklad

```
>> colors = categorical({'r' 'b' 'g'; 'g' 'r' 'b';...  
                        'b' 'r' 'g'}, {'r' 'g' 'b'}, {'red' 'green' 'blue'})
```

```
colors =  
      red      blue      green  
      green    red      blue  
      blue     red      green
```

```
>> colors == 'red'  
ans =  
      1      0      0  
      0      1      0  
      0      1      0
```

Výčtové typy

- Typ categorical - výčtový typ nabývající hodnot z konečné množiny

- **Příklad**

```
>> colors = categorical({'r' 'b' 'g'; 'g' 'r' 'b';...  
                        'b' 'r' 'g'}, {'r' 'g' 'b'}, {'red' 'green' 'blue'})
```

```
colors =  
      red      blue      green  
      green    red      blue  
      blue     red      green
```

```
>> colors == 'red'
```

```
ans =  
      1      0      0  
      0      1      0  
      0      1      0
```

Část III

Vektorizace a skládání výrazů a příkazů

Vektorizace a skládání výrazů a příkazů

- V Matlabu se obecně snažíme vyhnout cyklům. Jsou pomalé, vytváří zbytečně dlouhý kód, může nastat problém s doalokováním. Např.

```
y=[]; % prázdné pole  
for x=1:10000  
    y=[y x]; % na konec pole y přidáme prvek x  
end
```

- V cyklu for často zpracováváme postupně všechny prvky pole, tzv. po prvcích. To lze často řešit vektorizovaným výrazem.
- Již známý příklad z minulé přednášky: vyhledávání prvků pomocí logického indexování.

Vektorizace a skládání výrazů a příkazů

- V Matlabu se obecně snažíme vyhnout cyklům. Jsou pomalé, vytváří zbytečně dlouhý kód, může nastat problém s doalokováním. Např.

```
y=[]; % prázdné pole  
for x=1:10000  
    y=[y x]; % na konec pole y přidáme prvek x  
end
```

- V cyklu `for` často zpracováváme postupně všechny prvky pole, tzv. po prvcích. To lze často řešit vektorizovaným výrazem.
- Již známý příklad z minulé přednášky: vyhledávání prvků pomocí logického indexování.

Vektorizace a skládání výrazů a příkazů

- V Matlabu se obecně snažíme vyhnout cyklům. Jsou pomalé, vytváří zbytečně dlouhý kód, může nastat problém s doalokováním. Např.

```
y=[]; % prázdné pole  
for x=1:10000  
    y=[y x]; % na konec pole y přidáme prvek x  
end
```

- V cyklu `for` často zpracováváme postupně všechny prvky pole, tzv. po prvcích. To lze často řešit vektorizovaným výrazem.
- Již známý příklad z minulé přednášky: vyhledávání prvků pomocí logického indexování.

Vektorizace a skládání výrazů a příkazů

Příklad: Výpočet $\log_{10}(x)$ na intervalu $[0.01, 10]$ s krokem 0.01

```
index=0;  
for x=0.01:0.01:10  
    index=index+1;  
    y(index)=log10(x);  
end
```

versus

```
x=0.01:0.01:10;  
y=log10(x);
```

versus

```
y=log10(0.01:0.01:10);
```

Vektorizace a skládání výrazů a příkazů

Příklad: Výpočet $\sin^2(x) \cos(x)$ na intervalu $[1,20]$ s krokem 0.1

```
index=0;  
for x=1:0.1:20  
    index=index+1;  
    y(index)=sin(x)^2*cos(x);  
end
```

versus

```
x=1:0.1:20;  
  
y=sin(x)^2*cos(x); % toto je špatně!  
  
y=sin(x).^2.*cos(x);
```

Vektorizace a skládání výrazů a příkazů

Příklad: Odečtení řádkových průměrů z řádků matice A

```
for i=1:size(A,1)
    for j=1:size(A,2)
        prumer=0;
        for k=1:size(A,2)
            prumer=prumer+A(i,k);
        end
        prumer=prumer/size(A,2);
        A(i,j)=A(i,j)-prumer;
    end
end
% tohle je katastrofa: zápočet nikdy!
```

versus

```
for k=1:size(A,1)
    A(k,:)=A(k,:)-mean(A(k,:));
end
```

Vektorizace a skládání výrazů a příkazů

Příklad: Odečtení řádkových průměrů z řádků matice A

```
A = A - repmat(mean(A,2),1,size(A,2));  
% repmat vytváří pole "dlaždicováním" první proměnné
```

versus

```
A = A - mean(A,2)*ones(1,size(A,2));  
% častý trik s využitím maticového násobení
```

versus

```
A = bsxfun(@minus,A,mean(A,2));  
% použití speciální funkce bsxfun
```

Novinka od verze jádra 9.0 (R2016b)

```
A = A - repmat(mean(A,2),1,size(A,2));  
% repmat vytváří pole "dlaždicováním" první proměnné
```

Ize provést jednoduše pomocí

```
A = A - mean(A,2);
```

Operátory tedy implicitně realizují funkci `bsxfun`.

Tento materiál vznikl v rámci projektu ESF CZ.1.07/2.2.00/28.0050

Modernizace didaktických metod a inovace výuky technických předmětů,
který je spolufinancován Evropským sociálním fondem a státním rozpočtem ČR.