Práce s maticemi

Vytváření matic

- přímým zápisem prvků v Command window nebo do M-souboru - viz 1. lekce (hranaté závorky, čárky (mezery) a středníky, případně s využitím operátoru dvojtečka, jsou-li částí matice aritmetické posloupnosti),
- pomocí příkazu `load` (viz 1. lekce),
- využitím importu dat z Excelu (viz 1. lekce),
- pomocí speciálních funkcí - viz níže,
- jako výsledek některých operací, např. sečtením již existujících matic (viz níže - operátory pro práci s maticemi),
- spojováním matic (viz níže; jde v podstatě o 1. způsob, ale uvnitř hranatých závorek jsou matice),
- výběrem prvků z již existujících matic nebo přepsáním některých prvků/částí matice (viz níže - přístup k prvkům matice)

Vytváření matic pomocí speciálních funkcí

Funkce	**Popis**
`eye(n)`	jednotková matice řádu *n*
`zeros(n)` `zeros(m,n)`	matice samých nul řádu *n* nebo typu *m*x*n*
`ones`	matice samých jedniček, volání jako u `zeros` - 1 nebo 2 vstupy
`rand`	matice náhodných čísel (rovnoměrné rozdělení), volání jako u `zeros` - 1 nebo 2 vstupy
`randn`	matice náhodných čísel (normální rozdělení), volání jako u `zeros` - 1 nebo 2 vstupy
další speciální matice	
`magic(n)`	magický čtverec (součty řádků, sloupců a hl./vedl. diagonály jsou shodné), funguje správně pro *n*>2
`pascal(n)`	Pascalova matice (symetrická pozitivně definitní matice, "čtvercová" část Pascalova trojúhelníku)
`hadamard(n)`	Hadamardova matice (ortogonální matice složená z prvků 1 a -1), $n=2^k$, $k=1,2,...$

`toeplitz(c,r)`
 	
`toeplitz(r)`
	Toeplitzova matice (má stejné "diagonály") vytvářená z 2 vektorů (co je ve sloupcích, co je v řádcích)
	symetrická Toeplitzova matice
	využívá se při zpracování signálů

`hankel`	Hankelova matice (podobná Toeplitzově, ale má stejné "vedlejší diagonály"; speciálním případem je Hilbertova matice)
`hilb`	Hilbertova matice (velmi špatně podmíněná čtvercová symetrická matice, jejíž první řádek je tvořen začátkem harmonické řady - převrácenými hodnotami přirozených čísel; další řádky matice vznikají posunem v harmonické řadě o jednu pozici vpravo)
`invhilb`	inverze Hilbertovy matice
vytvoření matic z jiných	
`rot90(A)`	otočení matice o 90 ° (proti směru hodinových ručiček)

`tril(A)`	dolní trojúh. matice vzniklá z `A` vynulováním prvků nad hl. diagonálou (bez provedení řádkových úprav!)
`triu(A)`	horní trojúh. matice vzniklá z `A` vynulováním prvků pod hl. diagonálou (bez provedení řádkových úprav!)
`diag(v)`	diagonální matice z vektoru `v` (je víc možností - viz `help diag`)
`blkdiag (A,B,...)`	blokově diagonální matice ze zadaných matic
`fliplr(A)`	převrácené pořadí sloupců matice
`flipud(A)`	převrácené pořadí řádků matice

Další funkce získáte v nápovědě - `help elmat`.

Příklady:
```
>> I=eye(4) % jednotková matice řádu 4
>> M=magic(5) % magický čtverec řádu 5
>> N=zeros(3,2) % nulová matice 3x2
>> x=a+(b-a)*rand(1,10) % 10 náh. čísel s rovnoměrným rozložením z intervalu [a,b]
>>  y=0.6+sqrt(0.1)*randn(4) % matice řádu 4, náh. čísla s normálním rozložením (stř. hodnota je 0,6 a rozptyl 0,1)
```

Spojování matic

Pokud dodržíme pravidlo, že výsledkem bude obdélníkové schéma, je možno matice a vektory spojovat do větších celků, jak ukazují následující příklady:
```
>> M1 = [2 4; -2 -1];
>> M2 = zeros(4); M3 = ones(2); M4 = 1:4;
>> M5 = eye(4); M6 = 0.1:0.1:0.4;
>> % spojujeme:
>> M_spoj1 = [M1 M3]
>> M_spoj2 = [M1; M3]
>> M_spoj3 = [[M1; M3] M4']
>> M_spoj4 = [[[M1 M3]; M2; M4; M6] [M6'; M6']]
```

Poznámka: se spojováním matic jsme se již setkali v případě řetězců (tj. jednořádkových matic obsahujících znaky):
```
>> disp(['Výsledkem výrazu 3+2^3 je ' num2str(3+2^3) '.'])
```

Poznámka: pro spojování matic (polí) lze použít také funkci **`cat`**.

Prázdná matice

Prázdná matice je každá matice, která má alespoň jednu dimenzi nulovou:
```
>> M = [] % obě dimenze nulové (matice)
>> M = zeros(3,0) % druhá dimenze nulová (vektor)
>> M = 1:-3 % první dimenze nulová (vektor)
```

Je matice (proměnná) prázdná?

Funkce **`isempty(p)`** vrací pravdu (1), pokud je proměnná `p` prázdnou maticí. Jinak vrací false (0):
```
>> isempty(M) % 1
>> isempty(A) % 0
```

Pozn.: pokud chcete testovat, zda proměnná vůbec existuje, použijte funkci **`exist(str)`**, kde `str` je název proměnné jako řetězec:

```
>> exist('f') % 0 = neexistuje
>> exist('M') % 1 = existuje a je to proměnná ve workspace
>> exist('sin') % 5 = existuje a je to built-in funkce
```

Funkce nabízí širší možnosti - viz nápověda.

Zjištění rozměrů matice

Víme, že délku vektoru lze zjistit pomocí funkce `length` nebo `size`. V případě matice můžeme využít:

- funkci **`size`** - vrací rozměr(y) matice podle způsobu volání:
  - `d = size(M)` ... vrátí nejméně dvouprvkový vektor se všemi rozměry matice (řádky, sloupce, 3. dimenze, 4. dimenze apod.),
  - `[m,n] = size(M)` ... vrátí počet řádků a sloupců matice,
  - `m = size(M,dim)` ... vrátí počet prvků matice v daném rozměru (`dim=1` pro řádky, `dim=2` pro sloupce),
  - `[d1,d2,d3,...,dn] = size(M)` ... vrátí počty prvků v požadovaných rozměrech matice (mělo by být `n=ndims(M)`).
- funkci **`ndims`** - vrací počet rozměrů/dimenzí matice (většinou máme jenom dvourozměrné matice)
- funkci **`numel`** - vrací celkový počet prvků matice ve všech rozměrech/dimenzích

Přístup k prvkům matice

!!! Matice jsou v MATLABu indexovány od jedničky (jinak chyba `Subscript indices must either be real positive integers or logicals.`) a index nesmí překročit velikost matice v daném rozměru/dimenzi (jinak chyba `Index exceeds matrix dimensions.`) !!!

Způsobů, jak k prvkům matice přistupovat, je hodně:
mějme matici `A = [1 1/4 0; 0 4 3.11; -2 0 7];`

- **přístup k jednomu prvku pomocí konkrétních indexů:**
  do kulatých závorek uvedeme konkrétní hodnoty indexů oddělené čárkami
  ```
  >> A(3,1)
  ```
- **přístup k jednomu prvku pomocí jediného indexu:**
  do kulatých závorek uvedeme jen jeden index - matice je pak chápána jako sloupcový vektor (složený z jejích sloupců - pod sebou):
  ```
  >> A(1) % první prvek v prvním řádku
  >> A(7) % prvek v 1. řádku a ve 3. sloupci
  ```
  Zkuste najít vztah pro převod indexu *k* na dva indexy *i*,*j*! (Nebo prostudujte funkce `sub2ind` a `ind2sub`.)
- **přístup ke koncovým prvkům pomocí `end`:**
  k indexaci posledního prvku v daném směru lze použít `end`:
  ```
  >> A(2,end) % výsledkem bude 3.11
  >> A(2,end-1) % výsledkem bude 4
  >> A(end,end) % výsledkem bude 7
  ```
- **přístup k celému řádku/sloupci:**
  místo indexu sloupce/řádku uvedeme dvojtečku (označuje "všechny prvky v daném směru")
  ```
  >> A(1,:) % celý první řádek matice A
  >> A(:,2) % celý druhý sloupec matice A
  ```
  Poznámka: dvojtečku lze také použít pro indexaci pomocí jediného čísla, tj. z matice je vyroben sloupcový

vektor:

```
>> v = A(:) % sloupce původní matice se poskládají pod sebe
```

- **přístup k submatici - indexace pomocí vektorů:**
  místo indexu řádku a/nebo sloupce uvedeme vektor indexů

```
>> A(3,[1 3]) % první a poslední prvek z třetího řádku
>> A([2 3],[1 3]) % všechny prvky z 2. a 3. řádku a zároveň 1. a 3. sloupce
>> A(2:3,[1 3]) % totéž
>> B=A([end:-1:1],:) % vytvoří matici B z A prohozením pořadí jejích řádků
>> C=rand(5,7); D=C(3:5,[1:3]) % D je "levá dolní" část matice C
```

- **přístup k více prvkům najednou - logická indexace:**
  jedná se o indexaci nesouvislého výběru prvků matice pomocí logického výrazu - vybrány budou ty prvky, u nichž je logický výraz pravdivý. Pokud se tento typ indexace použije pro vymazávání prvků, vznikne řádkový vektor.

```
>> A(A<1) % výsledkem bude proměnná 'ans' - sloupcový vektor prvků menších než 1
>> A(A>2) = 7 % matice, kde všechny prvky větší než 2 budou nastaveny na 7
>> g = A(A<1) % řádkový vektor obsahující všechny prvky <1
>> h = A; h(h<1) = [] % ŘÁDKOVÝ vektor obsahující všechny prvky >=1
>> h = A(:); h(h<1) = [] % SLOUPCOVÝ vektor obsahující všechny prvky větší nebo rovné
1
>> M = magic(5); M(isprime(M)) = 0 % v matici M se všechna prvočísla přepíší na nulu
>> M = magic(5); M(~isprime(M)) = 0 % v matici M se složená čísla přepíší na nulu
```

Vyhledávání indexů prvků podle podmínky

Jak jsme zjistili v předchozím příkladu, příkaz `A(A<1)` vrací sloupcový vektor hodnot matice **A**, které splňují danou podmínku. Co když ale potřebujeme znát jejich indexy? Pak můžeme použít funkci **`find`**: funkce vrací sloupcový vektor jednoduchých indexů (tj. chápe matici jako sloupcový vektor):

```
>> k = find(A==0) % vrátí sloupec s příslušnými indexy všech nul v matici A
>> B = zeros(size(A)); B(k) = 1 % využití vektoru 'k'
>> k = find(A==0)' % chceme-li řádkový vektor
>> B = A; B(find(B==0)) = 100 % náhrada nul za stovky (lze vyřešit i bez funkce find)
>> B = A; B(B==0) = 100 % totéž
```

Změna prvku/ů matice

Pomocí přiřazovacího příkazu, kde vlevo je indexovaná oblast dané matice a napravo matice stejného rozměru, jaký vytvoří indexovaná oblast (tato matice může být vytvořená výpočtem nebo nějakou funkcí):

```
>> A(2,2) = 13.7 % změna 1 prvku
>> B(:,2) = [1;1;1] % změna 2. sloupce matice B
>> B(:,2) = ones(3,1) % totéž
>> C(3,[1 3 5 7]) = zeros(1,4) % vynulování prvků ve 3. řádku a 1., 3., 5. a 7. sloupci
>> C(2:4,4:6) = ones(3) % "uprostřed" matice C budou jedničky
```

Odstranění řádku/sloupce matice

Pomocí přiřazovacího příkazu, kde vlevo je indexovaná oblast dané matice (nesmí to být jen jeden prvek!) a napravo prázdná matice. Po provedení příkazu musí zůstat opět obdélníkové schéma (matice menšího rozměru):

```
>> A(2,:) = [] % odstranění 2. řádku matice A
>> B(1,2) = [] % NELZE provést!!!
>> B(:,end) = [] % odstranění posledního sloupce matice B
>> C(3,[1 3 5 7]) = zeros(1,4) % vynulování prvků ve 3. řádku a 1., 3., 5. a 7. sloupci
>> C(2:4,4:6) = ones(3) % "uprostřed" matice C budou jedničky
```

Poznámka: jednotlivé prvky lze z matice odstranit pouze s použitím jediného indexu, výsledkem však už nebude matice, ale řádkový vektor:

```
>> E = A; E(5) = [] % vznikne řádkový vektor, srovnejte s příkazem E = A(:); E(5) = []
>> F = A; F([2 4 6]) = [] % vznikne řádkový vektor s 6 prvky
```

Pokud se pro vymazávání prvků použije logická indexace, tak také vznikne řádkový vektor:

```
>> A(A<3) = []
```

Základní maticové operace

- **součet matic**
  - provádí se po prvcích,
  - matice musí být stejného typu,
  - je komutativní (A+B = B+A)
- **součet matice a skaláru**
  - skalár se přičte ke každému prvku matice,
  - je komutativní (a+A = A+a)
- **rozdíl matic**
  - provádí se po prvcích (a není komutativní),
  - obě matice musí být stejného typu
- **rozdíl matice a skaláru**, resp. skaláru a matice
- **součin matic**
  - definován podle pravidel lineární algebry,
  - matice musí být typů $m \times n$ a $n \times k$,
  - obecně není komutativní (existují výjimky pro některé čtvercové matice)
- **součin matice a skaláru**
  - každý prvek matice se vynásobí daným číslem,
  - je komutativní (A.a = a.A)
- **transpozice matice**
  - funguje správně i pro komplexní matice (mění prvky na komplexně sdružené)

MATLAB nabízí také operátor "dělení matic" - je určen pro nalezení řešení soustavy rovnic. Podrobný přehled operátorů je uveden níže.

Operátory `>> help ops`

MATLAB je orientován na práci s maticemi komplexních čísel. Tomu je přizpůsobeno chování operátorů. Operátory lze používat nejen pro matice, ale i pro skaláry (viz lekce 1) nebo vektory (viz lekce 4). Nyní si uvedeme přehled operátorů vzhledem k maticím (případně s poznámkami o vektorech a skalárech).

Poznámka: všimněte si (`>> help ops`), že operátory jsou v MATLABu definovány i jako funkce.

Připravme si následující proměnné (viz také soubor `promenne.mat`):

```
>> A = [1 2 3; 1 2 3; 1 1 1]
>> B = [1 0 1 0; 2 1 2 1; 1 1 2 2]
>> C = [5 4 3; 1 -1 0]
>> D = [1+i 2-i 2+2i; 3 -4i 1-2i; 1 2 -5i]
>> E = [1 0 0; 0 2 0; 0 0 3] % nebo: E = diag([1 2 3])
```

```
>> F = [1 1 -1; 1 0 -1; -1 1 0]
>> u = [1+2i 3+i 4-2i -5i]
>> v = [1 5 2+2i 3]
>> w = [-12.4; pi/2; -3/4]
>> z = [2 7 -3]
```

V části "vyzkoušejte" jsou uvedeny ukázky, přičemž chybné výrazy jsou označené >> a výrazy generující varování jsou označené >>.

1) Aritmetické operátory

a) unární

název	syntaxe	popis	vyzkoušejte
unární plus	`+a1`	- výsledek je shodný s `a1`	`>> +D`
unární minus	`-a1`	- výsledek je téhož rozměru jako `a1` a obsahuje opačná čísla	`>> -D`

transpozice a adjungování	`a1'`	- výsledkem je adjungovaná matice (= transponovaná matice, kde navíc všechna komplexní čísla změní znaménko imaginární části na opačné)
		- u reálných matic funguje stejně jako `.'`
			`>> D'`
			`>> B'`
			`>> v'`

transpozice	`a1.'`	- výsledkem je transponovaná matice (oproti `'` nemění komplexní prvky znaménko imaginární části)	`>> D.'` `>> B.'`

b) binární

název	syntaxe	popis	vyzkoušejte

sčítání	`a1+a2`	- provádí se po prvcích
		- argumety musí mít SHODNÉ rozměry (nebo alespoň jeden z nich musí být skalár)
		- prvky výsledku jsou součtem/rozdílem odpovídajících si prvků proměnných `a1` a `a2` (skalár je automaticky "doplněn na rozměr" většího operandu)
		- sčítání je komutativní (tj. `a1+a2 = a2+a1`), odčítání ne
			`>> A+D`
			`>> 3+A % A+3`
			`>> u+A`

odčítání	`a1-a2`		`>> u-2`
			`>> 2-u`
			`>> A-D`
			`>> A-B`

umocnění	`a1^a2`	- jsou-li oba operandy skaláry, jde o `a2`-tou *mocninu čísla* `a1`. Platí:
		$a^0 = 1$
		$a^{-n} = \frac{1}{a^n}$ pro $n \in \mathbb{N}$
		$a^{m/n} = \sqrt[n]{a^m}$ pro $m \in \mathbb{Z},\ n \in \mathbb{N}$
		- umocnění komplexních čísel: dle pravidel (např. Moivreův vzorec + převod na algebr. tvar)
		- v případě odmocnin (tj. exponent je zlomek) ze záporného čísla je vrácena první komplexní odmocnina
			`>> 2^3 % 8`
			`>> 27^(1/3) % 3`
			`>> 2^(-1) % 0.5`
			`>> 27^(-2/3)`
			`% 1/9`
			`>> (3-2i)^3`
			`>> (-1)^.5 % i`

		- pokud je `a1` matice a `a2` CELÉ číslo (skalár), jde o *umocnění matice*:
		A. pro kladné `a2` platí: `a1^a2 = a1*a1*...*a1` (násobení matic!)
		B. pro `a2=0` je výsledkem jednotková matice
		C. pro záporné `a2` platí: `a1^a2 = A*A*...*A`, kde `A` je inverzní matice k `a1` (násobení matic!), `a1` musí být REGULÁRNÍ!
			`>> A^3 % A*A*A`
			`>> A^0`
			`>> A^(-1)%sing.`
			`>> B^2 % nečtv.`

		- `a1` vždy musí být čtvercová matice (viz násobení matic) - **poznámka**: pokud `a2` není celé číslo, používají se vlastní čísla a vlastní vektory matice `a1` (viz nápověda)	`>> E^(-1)` `>> F^(-2)`
		- pokud je `a1` skalár a `a2` matice, používají se vlastní čísla a vlastní vektory (viz nápověda) - pokud jsou oba operandy matice, Matlab ohlásí chybu	`>> A^D`
		- žádné umocnění (ani "maticové", ani "po prvcích") není komutativní	
umocnění po prvcích	`a1.^a2`	- jsou-li oba operandy skaláry, pak není rozdíl mezi `.^` a `^`, tj. pro skaláry je `a1.^a2=a1^a2`	`>> 3.^2 % = 3^2`

		- jsou-li oba operandy matice/vektory, pak je provedeno *umocnění po prvcích*. Oba operandy tedy musí mít SHODNÉ rozměry!
		- výsledek je téhož typu jako vstupy a platí (pro `C=A.^B`): $C_{i,j} = (A_{i,j})^{B_{i,j}}$
			`>> A.^A`
			`>> A.^E`
			`>> E.^A`
			`>> A.^B`

		- je-li právě jeden z operandů skalár (a druhý matice nebo vektor), potom je skalár "rozšířen" na rozměr druhého operandu a operace se chová tak, jakoby oba operandy byly matice/vektory téhož typu: pro `C=A.^a` je $C_{i,j} = (A_{i,j})^{a}$ a pro `C=a.^A` je $C_{i,j} = a^{A_{i,j}}$	`>> A.^3`
			`>> A.^0`
			`>> 2.^A`
			`>> 7.^B`
			`>> (1+2i).^C`

název	syntaxe	popis	vyzkoušejte
násobení matic	`a1*a2`	-jsou-li oba operandy skaláry, jedná se o *násobení čísel* - násobení čísel je komutativní (tj. `a1*a2=a2*a1`)	`>> 3*pi` `>> 4*(2+i)`
		- je-li jeden z argumetnů skalár a druhý matice, jedná se o *algebraické násobení matice/vektoru číslem*, přičemž $C_{i,j} = A_{i,j} \cdot a$ - je komutativní (tj. `C = A*a = a*A`)	`>> C*2 % 2*C` `>> (1+i)*u`

		- jsou-li oba operandy matice, jedná se o *algebraické násobení matic*, a proto počet sloupců `a1` musí být shodný s počtem řádků `a2` (tj. *mxp* a *pxn*). Pak výsledek `C=A*B` má rozměr *mxn* a každý jeho prvek je $C_{i,j} = \sum_{k=1}^{p} A_{i,k} \cdot B_{k,j}$
		- násobení matic NENÍ komutativní!
			`>> A*C`
			`>> C*A % jiné`
			`>> A*B`
			`>> B*A`
			`>> u*v`

násobení po prvcích	`a1.*a2`	- je-li některý z operandů skalár, pak není rozdíl mezi operátorem `.*` a `*` (tečku lze vynechat), tedy `a1.*a2=a1*a2`
		- je komutativní, tedy `a1.*a2=a2.*a1`
		- obecně je tedy (když `A` je matice/vektor a `a` je skalár): `A.*a=A*a=a*A=a.*A`
			`>> 3.*u % 3*u`
			`>> C.*(3+2i)`
			`% C*(3+2i)`

		- pokud oba operandy jsou matice (resp. vektory), jedná se o *násobení odpovídajících si prvků matic/vektorů*, přičemž $C_{i,j} = A_{i,j} \cdot B_{i,j} = B_{i,j} \cdot A_{i,j}$
		- oba operandy musí mít SHODNÉ rozměry!
		- je komutativní
			`>> A.*C`
			`>> C.*A % totéž`
			`>> A.*B`
			`>> B.*A`
			`>> u.*v`

název	syntaxe	popis	vyzkoušejte

dělení	`a1/a2`	- jsou-li oba operandy skaláry, jde o *dělení čísel* (v případě, že `a2=0`, Matlab vypíše varování a vrátí nekonečno `Inf` se znaménkem nebo "nečíslo" `NaN` = not a number)	`>> 117/13 % 9`
			`>> -5/0 % -Inf`
			`>> 0/0 % NaN`

		- je-li `a1` matice/vektor a `a2` je skalár, pak platí: `a1/a2 = a1*(1/a2)`, tj. jedná se o *násobení matice/vektoru* `a1` *převráceným číslem k* `a2`
		- v případě `a2=0` je vypsáno varování a vrácena "matice/vektor nekonečen"
			`>> A/5 % A*0.2`
			`>> A/0`
			`>> B/2`
			`>> u/-10`

		- je-li `a1` skalár a `a2` matice/vektor, pak operace skončí chybou "nesouhlasí rozměry"	`>> 7/A`
			`>> 0/B`
			`>> -5/u`

		- jsou-li oba operandy matice/vektory, pak platí: **`a1/a2 = (a2'\a1')'`** (viz dělení zleva)
		- je-li `a2` singulární matice, tak Matlab vypíše varování a vrací "matici/vektor nekonečen"
		- počet sloupců `a1` se musí rovnat počtu řádků `a2`!
			`>> F/F % F.F^-1`
			`>> A/A % sing.`
			`>> z/F % z.F^-1`
			`>> z/A % sing.!`
			`>> A/B % rozměr`
			`>> B/F % rozměr`
			`>> (B')/F %B'/F`

		- žádné dělění (ani "maticové", ani "po prvcích") není komutativní	
dělení po prvcích	`a1./a2`	- jsou-li oba operandy skaláry, jde o *dělení čísel* (a není rozdíl mezi `./` a `/`)	`>> 9./3 % = 9/3` `>> 7./0 % Inf`

		- je-li jeden z operandů matice/vektor a druhý skalár, pak se každý prvek matice/vektoru chápe jako dělenec (resp. dělitel) a skalár je chápán jako dělitel (resp. dělenec) - operace samozřejmě není komutativní!
		- výsledkem je matice/vektor stejného rozměru jako vstupní matice/vektor
		- v případě, kdy `a2` je skalár, není rozdíl mezi `./` a `/`
			`>> A./3 % = A/3`
			`>> 3./A`
			`>> 5./B`
			`>> A./0 % = A/0`
			`>> 0./A`

		- jsou-li oba operandy matice/vektory, pak jde o *dělení odpovídajících si prvků*
		- výsledek `C=A./B` obsahuje prvky $C_{i,j} = \frac{A_{i,j}}{B_{i,j}}$
		- oba operandy musí mít STEJNÉ rozměry!
			`>> A./A`
			`>> F./F % děl.0`
			`>> A./B %rozměr`
			`>> u./v`
			`>> A./E % děl.0`

název	syntaxe	popis	vyzkoušejte
dělení zleva	`a1\a2`	- jsou-li oba operandy skaláry, pak platí: `a1\a2 = (1/a1)*a2 = a2/a1`	`>> 3\2 % 2/3` `>> 0\10 % Inf`
		- je-li `a1` skalár a `a2` matice/vektor, pak se jedná o *násobení* `a2` *převráceným číslem k* `a1`	`>> 3\A % 3^-1.A` `>> 0\u % děl.0`
		- je-li `a1` matice/vektor a `a2` skalár, pak Matlab ohlásí chybu "nesouhlasí rozměry"	`>> A\2` `>> F\2`

		- jsou-li oba operandy matice/vektory, pak lze operaci ve většině případů chápat jako **`a1\a2 = inv(a1)*a2`** (kde `inv` vrací inverzní matici)
		- operace je v Matlabu zavedena především kvůli **řešení soustav lineárních rovnic**, kdy `a1=A` je matice řádu *n* a `a2=b` je vektor pravých stran (typu *n*x1): řešení soustavy `Ax=b` získáme příkazem `x=A\b` (používá se Gaussova eliminace)
			`>> F\w`
			`>> A\w % sing.`
			`>> F\z %rozměr`
			`>> F\z'`
			`>> B\z'`
			`% více pravých stran:`
			`>> F\B`

		• je-li `a1` regulární, pak je vráceno úplné (jediné) řešení soustavy • v případě, že `a1` je singulární, vypíše se varování • je-li `a1` typu *m*x*n*, kde *m*>*n* (resp. *m*<*n*), pak je vráceno řešení ve smyslu metody nejmenších čtverců. Nemusí to ale být partikulární řešení soustavy! ... viz např. soustava, která nemá řešení: `>> x=[1 2; 1 -1; 2 0]\[3 0 0]' % x = [0.3103 1.1379], ale není řešením` • VŽDY by se měl provést test řešitelnosti soustavy (Frobeniova věta) • jak získat VŠECHNA řešení soustavy (s neregulární maticí), si ukážeme na cvičení - *poznámka*: `X = A\B` vrací řešení maticové rovnice `AX = B`, kdežto `X = B/A` vrací řešení maticové rovnice `XA = B`	
		- žádné dělění zleva (ani "maticové", ani "po prvcích") není komutativní	
dělení zleva po prvcích	`a1.\a2`	- provádí se po prvcích a platí: **`a1.\a2 = a2./a1`**, takže výsledek `C=A.\B` obsahuje prvky $C_{i,j} = \frac{B_{i,j}}{A_{i,j}}$	`>> A.\E % = E./A`

!!! POZOR u čtvercových matic **!!!**
Pro čtvercové matice většina operátorů funguje, aniž by MATLAB hlásil chybné rozměry matic. Proto je potřeba dávat si u těchto matic velký pozor na to, zda chceme provést operaci "maticovou" (ve smyslu lin. algebry), anebo "po prvcích" (tj. matice chápeme jen jako "tabulku" s daty). Výsledky jsou obecně *odlišné*!
Porovnejte výsledky následujících výrazů (Matlab ohlásí nanejvýš varování při dělení nulou):

```
>> E*F    >> E.*F
>> E/F    >> E./F
>> F^4    >> F.^4
```

c) dvojtečka

název	syntaxe	popis	vyzkoušejte

vytváření aritmetických posloupností	`i:j`	- výsledkem je aritmetická posloupnost od *i* do *j* s krokem jedna, tj. `[i i+1 i+2k ... j]` (pokud *j* není celé číslo, je zaokrouhleno dolů)
		- je-li konec zvolen chybně (tzn. že se nelze od *i* dostat přičítáním jedničky k *j*), pak je výsledkem prázdná matice (tj. `[]`)
			`>> 1:10`
			`>> 3:14.2`
			`>> 5:-5 % []`
			`>> 10:1 % []`

	- všechny operandy musí být SKALÁRY		

	`i:k:j`	- výsledkem je aritmetická posloupnost od *i* do *j* s krokem *k*, tj. `[i i+k i+2k ... j]` (v případě, že (*j*-*i*)/*k* není celé číslo, tak poslední člen není přesně číslo *j*, ale číslo menší maximálně o *k*-1)
		- je-li krok nebo konec zvolen chybně (tzn. že se nelze od *i* dostat k *j* přičítáním *k*), pak je výsledkem prázdná matice `[]`
			`>> -2:0.1:2`
			`>> 2:2:9 % do 8`
			`>> 5:-1:-5`
			`>> -3:-1:3 % []`
			`>> 10:5:0 % []`

2) Relační operátory

název	syntaxe	popis	vyzkoušejte

menší	`a1<a2`	- operandy mohou být:
		A. dva skaláry → výsledek je 0 nebo 1
		B. skalár a matice/vektor (příp. matice/vektor a skalár)
		C. dvě matice (vektory) STEJNÉHO typu
		- v případě B. je se skalárem porovnáván každý prvek druhého operandu, v případě C. se porovnání provádí PO PRVCÍCH
		- výsledek má rozměr jako neskalární operand (jen v případě A. je výsledkem skalár) a obsahuje **pouze nuly** (nepravda, false) **a jedničky** (pravda, true) podle toho, zda je příslušná relace mezi prvky pravdivá
		- při porovnávání 2 komplexních čísel je u relací <, >, <= a >= zanedbána imaginární část!
			`%pravda:`
			`>> 2<3`
			`>> 1+i==1+i`
			`>> -2i>=2i`
			`%nepravda:`
			`>> 5<=-7`
			`>> 5i~=5i`
			`% matice:`
			`>> w<=2`
			`>> -0.5>F`
			`>> A>F`
			`>> A>B`

menší nebo rovno	`a1<=a2`		
větší	`a1>a2`		
větší nebo rovno	`a1>=a2`		
rovnost (je rovno?)	`a1==a2`		
nerovnost (je různé?)	`a1~=a2`		

!!! POZOR: je velký rozdíl mezi == a = **!!!**

- v případě `x==5` jde o výraz, jehož hodnotou je 0 nebo 1 (**porovnání** `x` s číslem 5) a `x` se nemění, zatímco
- v případě `x=5` jde o **přiřazení** (příkaz), které nevrací hodnotu, nýbrž změní obsah proměnné `x`!!!

3) Logické operátory

a) unární

název	syntaxe	popis	vyzkoušejte

negace	`~a1`	- aplikuje se na každý prvek `a1`
		- výsledek má rozměr jako `a1`
		- výsledek obsahuje pouze nuly a jedničky (viz tabulka negace)
		- operand musí být reálné číslo (ne tedy komplexní), resp. reálná matice/vektor!!!
			`>> ~3.125`
			`>> ~0`
			`>> ~-1.2`
			`>> ~B`
			`>> ~D`

b) binární

název	syntaxe	popis	vyzkoušejte

logický součin, AND	`a1&a2`	- provádějí se po prvcích
		- operandy musí mít STEJNÉ rozměry nebo alespoň jeden z nich je skalár
		- výsledek má rozměr jako neskalární operand (nebo je to skalár)
		- výsledek obsahuje pouze nuly a jedničky (viz tabulka logického součinu a součtu)
		- operandy musí být reálné!!!
			`>> 2&0`
			`>> A&F`
			`>> 1&2i % komplex.!`
			`>> w&z % rozměry`

logický součet, OR	`a1|a2`		`>> -3.2|0`
			`>> F|0`
			`>> D|1 % komplex.!`
			`>> C|u % rozměry`

Poznámka: kromě výše uvedených operátorů nabízí Matlab ještě
- funkci `xor(a1,a2)`, která realizuje binární operátor XOR (= exkluzivní OR, negace ekvivalence),
- operátory `&&` a `||`, které provedou logický součin a součet se zkráceným vyhodnocením (zde ale vstupy musí být pouze SKALÁRY!),
- funkci **`any(a1)`** pro zjištění, zda matice `a1` obsahuje alespoň jednu pravdivou hodnotu (vrací skalár: 1 nebo 0) a
- funkci **`all(a1)`** pro zjištění, zda všechny prvky matice jsou pravdivé (vrací skalár: 1 nebo 0).

Tabulka výsledků logických operátorů:

negace	
`a1`	**`~a1`**
nenulové	`0`
`0`	`1`

logický součin (AND) a součet (OR)				**XOR**
`a1`	**`a2`**	**`a1&a2`**	**`a1|a2`**	**`xor(a1,a2)`**
nenulové	nenulové	`1`	`1`	`0`
nenulové	`0`	`0`	`1`	`1`
0	nenulové	`0`	`1`	`1`
0	`0`	`0`	`0`	`0`

Priorita operátorů `>> help precedence`

Pořadí, v jakém se budou jednotlivé části výrazu vyhodnocovat, můžeme podle potřeby zadat tím, že použijeme **kulaté závorky** pro ohraničení všech potřebných částí výrazu. Protože každý z výše uvedených operátorů má pevně dané pořadí vyhodnocování, není někdy závorek potřeba. Nyní si uvedeme pořadí vyhodnocování (prioritu) všech operátorů. Operátory v tabulce jsou seřazeny shora dolů podle klesající priority (tj. ty nejnižší se vyhodnocují nejpozději).

	symbol	poznámka
1.	`()`	*závorky*
2.	`.' .^ ' ^`	*transpozice, umocnění, transpozice+konjugovanost, maticové umocnění*
3.	`+ - ~`	*unární plus, unární minus, negace*
4.	`.* ./ .\ * / \`	*násobení po prvcích, dělení po prvcích, dělení po prvcích zleva, násobení matic, dělení matic, dělení matic zleva*
5.	`+ -`	*sčítání, odčítání*
6.	`:`	*dvojtečka*
7.	`< <= > >= == ~=`	*relační operátory*
8.	`&`	*logický součin, AND*
9.	`|`	*logický součet, OR*
10.	`&&`	*logický součin se zkráceným vyhodnocením*
11.	`||`	*logický součet se zkráceným vyhodnocením*

MATLAB a lineární algebra

Následující skupiny funkcí jsou seřazeny podle kapitol skripta *Matematika III* od Evy Dontové.

Vektory se v MATLABu chápou jako sloupcové (pouze výjimečně je píšeme do řádků). Většinu z níže uvedených operací jsme již probírali v rámci práce se symbolickým toolboxem (lekce 3), nyní si uvedeme funkce a operace pro datový typ `double`.
POZOR: vzhledem k reprezentaci čísel s plovoucí řádovou čárkou vznikají ZAOKROUHLOVACÍ CHYBY!

1. Vektorové podprostory

- **výsledek lineární kombinace** spočítáme pomocí operátoru `*`, který vrátí výsledek LK vektorů zapsaných jako sloupce matice `X` s koeficienty zapsanými do sloupcového vektoru a `LK = X*a`;
- **lineární (ne)závislost vektorů** - vektory zapíšeme do matice `X` a použijeme test:

```
if rank(X) == size(X,jak)
  disp('LN');
else
  disp('LZ');
```

  přičemž `jak=2` pro vektory zapsané do sloupců, `jak=1` pro vektory zapsané do ŘÁDKŮ;
- **zjištění dimenze podprostoru** *P*: funkce `d = rank(P)`, kde generátory podprostoru *P* tvoří sloupce nebo řádky matice `P`;
- **nalezení báze podprostoru** *P*:
  - ortonormální bázi lze získat pomocí funkce `B = orth(P)`, kde generátory podprostoru *P* jsou ve sloupcích matice `P` nebo
    s využitím vztahu $P=(P^{\perp})^{\perp}$: `B = null(null(P)')`, kde generátory podprostoru *P* jsou v ŘÁDCÍCH matice `P`,
  - bázi v "normalizovaném" tvaru lze získat s využitím téhož vztahu pomocí `B = null(null(P)','r')`, kde generátory podprostoru *P* jsou v ŘÁDCÍCH matice `P`,
  - "normalizovanou" bázi lze vyčíst z výsledku funkce `Pnorm = rref(P)`, kde generátory podprostoru jsou zapsané ve SLOUPCÍCH matice `P`,
  - pro výběr báze ze zadaných generátorů MATLAB žádnou funkci nenabízí;
- **součet dvou podprostorů** *P*, *Q*:
  - **dimenze** součtu:
    `ds = rank([P Q])`, když jsou generátory podprostorů ve sloupcích matic `P` a `Q` nebo
    `ds = rank([P; Q])`, když jsou generátory podprostorů v ŘÁDCÍCH matic `P` a `Q`;
  - **báze** součtu:
    `Bs = null(null([P Q])')`, kde generátory podprostorů tvoří sloupce matic `P` a `Q` nebo
    `Bs = null(null([P; Q])')`, kde generátory podprostorů tvoří ŘÁDKY matic `P` a `Q`;
- **průnik dvou podprostorů** *P*, *Q*:
  - **dimenze** průniku:
    `dp = rank(P)+rank(Q)-rank([P Q])`, když jsou generátory podprostorů ve sloupcích matic `P` a `Q`,
    `dp = rank(P)+rank(Q)-rank([P; Q])`, když jsou generátory podprostorů v ŘÁDCÍCH matice `P` a matice `Q`;
  - **báze** průniku:
    `Bp = null([null(P)'; null(Q)'])`, kde generátory podprostorů tvoří ŘÁDKY matic `P` a `Q`;

- **souřadnice vektoru vzhledem k bázi** nalezneme pomocí operátoru `\`, přičemž zadáme sloupcový vektor `x` a bázi `B` (vektory opět ve sloupcích):
  obdržíme (sloupcový) souřadnicový vektor `a`: **`a = B\x`** (matice `B` musí být regulární, tj. `rank(B)==size(B,2)`)
  (kontrolu souřadnic provedeme pomocí LK, tedy `B*a % x`)

2. Lineární zobrazení a matice

Následující funkce pracují s maticí zobrazení *A*:

- **jádro** lineárního zobrazení - bázi jádra vrací funkce **`KerA = null(A)`**,
- **defekt** lineárního zobrazení - dimenzi jádra vrací funkce **`dA = rank(null(A))`**,
- **obraz** lineárního zobrazení - bázi obrazu vrací funkce **`ImA = null(null(A)')`**,
- **hodnost** lineárního zobrazení - dimenzi obrazu vrací funkce **`hA = rank(A)`**.

Funkce pro práci s maticemi:

- **maticové operace** byly shrnuty v části základní maticové operátory;
- **hodnost matice** vrací funkce **`h = rank(A)`**;
- **defekt matice** vrací funkce **`d = rank(null(A))`**;
- **prvky na hlavní diagonále** vrací funkce `diag`
  prvky vedlejší diagonály lze získat po "překlopení" matice podle svislé osy: `diag(fliplr(M))`;
- **součet prvků na diagonále** vrací funkce `trace`. Platí: `trace(A)` je shodné se `sum(diag(A))`;
- **inverzní matici** vrací funkce **`Ainv = inv(A)`**.

Poznámka: inverzní matice existuje jenom k regulárním maticím. Pokud potřebujeme nalézt "inverzní" matici k jinému typu matic, tak můžeme použít tzv. **Mooreovu-Penroseovu pseudoinverzi** **`B = pinv(A)`**, která vrací matici `B` splňující tyto podmínky:
`A*B*A = A` a
`B*A*B = B` a
`A*B` je hermiteovská matice[1] a
`B*A` je hermiteovská matice.

[1] Hermiteovská matice je taková matice, která po (komplexní) transpozici zůstane nezměněna.

3. Soustavy lineárních rovnic

Řešení homogenní soustavy `Ax=0` (podprostor *V*) vrací funkce `null`:
`V = null(A)` (ortonormální báze) nebo
`V = null(A,'r')` ("normalizovaná" báze).

Řešení nehomogenní soustavy `Ax=b` je množina (lineál) ve tvaru `M=y+V`, kde `V` je řešením příslušné homogenní soustavy a vektor `y` je nějaké partikulární řešení.
Partikulární řešení získáme pomocí operátoru `\` (při splnění Frobeniovy věty, tj. když `rank(A)==rank([A b])` pro sloupcový vektor `b`):
`y = A\b`.
Pozn.: v případě regulární matice `A` je partikulární řešení zároveň úplným řešením soustavy.

Hermiteův tvar matice vrací funkce **`H = rref(A)`**.

4. Determinanty

Determinant matice vrací funkce `D = det(A)`.

Transpozici matice provede operátor `AT = A'`.

5. Skalární součin a ortogonalita

- **skalární součin 2 vektorů** (ze stejného $\mathbf{R}^n$ nebo $\mathbf{C}^n$):
  - pro dva řádkové vektory: `s = u*v'`,
  - pro dva sloupcové vektory: `s = u'*v`,
  - pro dva vektory (řádkové nebo sloupcové): `s = dot(u,v)`;
- **norma vektoru**: `n = norm(v)` (existují i jiné normy než eukleidovská norma);
- **ortonomální báze podprostoru** *P*: `B = orth(P)` (generátory *P* jsou ve sloupcích matice `P`).
  POZOR: funkce neprovádí Grammův-Schmidtův ortogonalizační proces (používá jiný algoritmus)!
- **báze ortogonálního doplňku k podprostoru** *P*: funkce `null` s voláním ve tvaru `B_doplnek = null(P)` nebo `B_doplnek = null(P,'r')`.

6. Lineární geometrie

Převod parametrického vyjádření lineálu `L=a+S` (ve sloupcích matice `S` jsou směrové vektory, `a` je sloupcový vektor) na soustavu normálových rovnic:

```
N = null(S') % normálový podprostor lineálu
c = N*a % pravé strany rovnic; L: Nx=c.
```

Převod soustavy normálových rovnic `Nx=c` (v ŘÁDCÍCH matice `N` jsou normálové vektory) na parametrické vyjádření lineálu:

```
S = null(N) % zaměření lineálu
a = N\c % nějaký bod lineálu; L=a+S.
```

Převod afinního obalu bodů na parametrické vyjádření lineálu:

```
a = AO(:,1); % bod lineálu; AO má body lineálu ve sloupcích
S = [AO(:,2:end)' - meshgrid(a)]' % funkci meshgrid si vysvětlíme příště; L=a+S.
```

7. Vlastní čísla a vlastní vektory matice

Koeficienty **charakteristického polynomu**:
`p = poly(A)` ... vektor `p` je řazen od největší mocniny k absolutnímu členu.

Výpočet **vlastních čísel** a **vlastních vektorů**[2] čtvercové matice`A`:
`lambda = eig(A)` ... vlastní čísla matice,
`[X,D] = eig(A)` ... vlastní vektory (sloupce matice `X`) příslušné jednotlivým vlastním číslům matice (na diagonále matice `D`),

---

[2] Nechť **A** je čtvercová matice řádu *n*, pak nenulové řešení soustavy $\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ se nazývá vlastní vektor (eigenvector) $\boldsymbol{x}$ a číslo λ se nazývá vlastní číslo (eigenvalue) matice **A**.

Práce s polynomy `>> help polyfun`

Důležitou část algebry tvoří práce polynomy. V MATLABu lze polynomy reprezentovat pomocí vektoru koeficientů a existuje mnoho funkcí, které provádějí běžné operace s polynomy (např. hledají kořeny polynomu).

Vyjádření polynomu v MATLABu

Každý polynom (mnohočlen) *n*-tého stupně je v MATLABu reprezentován pomocí (*n*+1)-prvkového **vektoru jeho koeficientů**:
vektor `p=[p1 p2 ... pn pn+1]` vyjadřuje polynom

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

Můžeme snadno vytvářet i polynomy s komplexními koeficienty.

Příklad: vytvoření polynomu
```
>> polynom = [3 1 0 1] % 3x^3+x^2+1
```

Funkce pro práci s polynomy

Přehled následujících funkcí získáte příkazem `>> help polyfun`.

- ***poly* - vytvoření polynomu z jeho kořenů**
```
>> p1 = poly([-1 1 3]) % polynom 3. stupně
```

- ***roots* - výpočet kořenů polynomu**
```
>> x = roots(p1)
```
Každý polynom stupně *n* má v tělese komplexních čísel přesně *n* kořenů (některé mohou být vícenásobné).

- ***polyval* - vyčíslení hodnoty polynomu v daném bodě**
```
>> y0 = polyval(p1, 0)
>> y = polyval(p1, [-5 -2 -1 0 .5 1 2 3 10])
```

- ***polyfit* - proložení dat polynomem (aproximace)**
```
>> x = 0:pi/10:pi; y = cos(x); % máme nějaká data (část fce kosinus)
>> p = polyfit(x,y,2) % aproximujeme data parabolou
>> y_aprox = polyval(p,x); % vyhodnocení polynomu v zadaných bodech
>> plot(x,y,'.k', x,y_aprox,'r') % graf (bez popisků!)
```

- ***conv* - násobení polynomů**
```
>> p2 = [3 1 0 1] % 3x^3+x^2+1
>> p3 = [1 -2 0] % x^2-2x
>> p4 = conv(p2, p3)

>> p5 = conv([1 -4], [1 4]) % vyjde x^2-16
```

- ***deconv* - dělení polynomů (i se zbytkem)**
```
>> p = deconv(p4,p3) % vyjde p2
>> [p, zbytek] = deconv(p4, p1)
```

- ***polyder* - derivace polynomu**
  Derivací polynomu vznikne polynom (n-1). stupně.
  ```
  >> d = polyder(p1)
  ```

- ***polyint* - primitivní funkce k polynomu**
  Primitivní funkcí k polynomu je polynom (n+1). stupně.
  ```
  >> p = polyint(d) % vyjde p1
  ```

- ***residue* - rozklad na parciální zlomky (obecně výpočet reziduí)**
  Funkci lze využít dvěma způsoby:

  - `[a,b,k] = residue(p,q)` ... rozklad na parciální zlomky (`k` není prázdné, pokud čitatel má vyšší stupeň než jmenovatel).

    Příklady:
    ```
    >> p = [1 0 -7 -6] % čitatel; p = poly([-1 -2 3])
    >> q = [1 -12 49 -78 40] % jmenovatel; q = poly([1 2 4 5])
    >> [a,b,c] = residue(p,q) % a = [7 -5 -2 1], b = [5 4 2 1], c = []
    ```
    $$\frac{x^3-7x-6}{x^4-12x^3+49x^2-78x+40}=\frac{7}{x-5}-\frac{5}{x-4}-\frac{2}{x-2}+\frac{1}{x-1}$$

    ```
    >> q = [1 -2 -5 6 0] % jmenovatel má 2 společné kořeny s 'p'
    % q = poly([-2 1 0 3])
    >> [a,b,c] = residue(p,q) % a = [0 0 2 -1], b = [3 -2 1 0], c = []
    ```
    $$\frac{x^3-7x-6}{x^4-2x^3-5x^2+6x}=\frac{0}{x-3}-\frac{0}{x+2}+\frac{2}{x-1}-\frac{1}{x}$$

    ```
    >> q = [1 3 3 1 0] % jmenovatel má vícenásobné kořeny
    % q = poly([-1 -1 -1 0])
    >> [a,b,c] = residue(p,q) % a = [7 4 0 -6], b = [-1 -1 -1 0], c = []
    ```
    $$\frac{x^3-7x-6}{x^4+3x^3+3x^2+x}=\frac{7}{x+1}+\frac{4}{(x+1)^2}+\frac{0}{(x+1)^3}-\frac{6}{x}$$

    ```
    >> q = [1 -7 13 -1 -14 8] % jmenovatel má vícenás. kořeny
    % q = poly([-1 1 1 2 4])
    >> [a,b,c] = residue(p,q) % a = [0.3333 2 -0 -2.3333 -2], b = [4 2 -1 1 1], c =
    []
    ```
    $$\frac{x^3-7x-6}{x^5-7x^4+13x^3-x^2-14x+8}=\frac{1/3}{x-4}+\frac{2}{x-2}-\frac{0}{x+1}-\frac{7/3}{x-1}-\frac{2}{(x-1)^2}$$

  - `[p,q] = residue(a,b,k)` ... zpětná rekonstrukce polynomů ze zlomků

Speciální znaky - přehled

V následující tabulce nejsou zahrnuty ty znaky, které jsou jen a pouze součástí operátorů nebo názvů proměnných. V příkladech jsou použity některé knihovní funkce.

`[ ]` (hranaté závorky)

- slouží pro vytváření vektorů a matic (např. `[9/2,-6,9.64,sqrt(36)]` pro čtyřprvkový vektor)
- uzavírají výstupy funkcí (např. `[m,n]=size(A)`)
- lze je použít pro vymazání řádku nebo sloupce matice

`( )` (kulaté závorky)

- mění prioritu operátorů
- uzavírají vstupní argumenty funkcí (např. `M=ones(5,3)`)
- umožňují indexaci (přístup k prvkům) matice/vektoru (např. `M(2,3)` pro třetí prvek na druhém řádku matice `M`)

`{ }` (složené závorky)

- vytvářejí pole buněk (např. `pb1={2; 3+2i; [1 2; 3 0; 4 1]}` nebo `pb2={[2; 1] 3+2i; [1 2; -2 3] [1:10]}`)
- přístup k prvkům pole buněk (např. `pb1{3}` nebo `A=pb2{1,2}`)
  Pozn.: přístup k prvku `-2` v matici uvnitř pole buněk `pb2` ... `pb2{2,1}(2,1)`

Poznámka: `< >` neslouží jako lomené závorky, ale jen a pouze jako relační operátory

` ` (mezera)

- odděluje od sebe prvky na každém řádku matice (viz vytváření matic a vektorů, např. `[-1 2.1 8]`)
- odděluje od sebe parametry některých příkazů (`save`, `clear`)
- lze ji použít jako "bílé místo" pro zpřehledňování příkazů (např. `a = 1.8 + 2/5 - 1`)

`,` (čárka)

- odděluje od sebe prvky na každém řádku matice (viz vytváření matic a vektorů, např. `[-1,2.1,8]` pro tříprvkový vektor)
- odděluje indexy při přístupu k vícerozměrným polím (tj. maticím), např. `M(1,2)`
- odděluje vstupní argumenty funkcí (např. `ones(3,5)`)
- odděluje výstupy funkcí (např. `[m,n]=size(M)`)
- odděluje od sebe více příkazů na jednom řádku, přičemž výpis jejich výsledků není potlačen (např. `if a==8, a, text='a je rovno 8', end`)

`;` (středník)

- odděluje od sebe řádky matice (viz vytváření matic a vektorů, např. `[9/2;-6;9.64;sqrt(36)]`)

- odděluje od sebe více příkazů na jednom řádku, přičemž potlačuje výpis jejich výsledků (např. `if a==8; a; text='a je rovno 8'; end` - srovnejte s čárkou)

`.` (tečka)

- desetinná tečka (`15.218`)
- je součástí některých aritmetických operátorů (`.*`, `./`, `.\`, `.^`)
- slouží pro přístup k prvkům struktury (zatím neprobráno)
- dvě tečky: `cd ..` slouží pro změnu pracovního adresáře na adresář o úroveň výš
- tři tečky (pokračování): `...` umožňují rozdělit jeden příkaz na více řádků

`=` (rovnítko)

- přiřazení (přiřazovací příkaz, např. `a=8`)
- součást relačních operátorů (`<=`, `>=`, `==`, `~=`)

`'` (anglický apostrof)

- transpozice matice (např. `M'` nebo `M.'`)
- uzavírá řetězce (např. `'Roman Straka'`)

`%` (procento)

- komentář - vše až do konce řádku bude MATLABem ignorováno

`:` (dvojtečka)

- slouží pro vytváření aritmetických posloupností (např. `p=10:-1:0`)
- submatice (např. `M(2:4,1:2)=2.3` nebo `A=M(1:3,1:2)`)
- součást cyklu `for`