

# Jak pracovat s MATLABem

Jan Koláček  
Kateřina Konečná

# Obsah

Úvod	2
<b>1 Začínáme s MATLABem</b>	<b>4</b>
1.1 Popis prostředí . . . . .	4
1.2 Základní ovládání . . . . .	6
1.3 Náповěda . . . . .	8
<b>2 Jednoduché výpočty, proměnné a matice</b>	<b>11</b>
2.1 MATLAB jako kalkulátor . . . . .	12
2.2 Proměnné, matice a jejich definování . . . . .	13
2.3 Funkce pro tvorbu matic . . . . .	15
2.4 Některé speciální výrazy a funkce . . . . .	18
2.5 Obecná pravidla pro příkazy . . . . .	19
<b>3 Maticové operace</b>	<b>22</b>
3.1 Transpozice matic . . . . .	22
3.2 Sčítání a odčítání matic . . . . .	23
3.3 Maticové násobení . . . . .	24
3.4 Maticové dělení . . . . .	24
3.5 Umocňování matic . . . . .	26
3.6 Operace po složkách . . . . .	27
3.7 Logické operace . . . . .	29
3.8 Smíšené operace . . . . .	29
<b>4 Manipulace s maticemi</b>	<b>32</b>
4.1 Základní manipulace s maticemi . . . . .	32
4.2 Změna struktury matice . . . . .	34
4.3 Základní funkce lineární algebry . . . . .	36
4.4 Další funkce pro manipulaci s maticemi . . . . .	37

<b>5</b>	<b>Logické operace</b>	<b>43</b>
5.1	Relační operátory . . . . .	43
5.2	Logické operátory . . . . .	44
5.3	Logické funkce . . . . .	45
5.4	Funkce <code>find()</code> a <code>exist()</code> . . . . .	47
<b>6</b>	<b>Textové řetězce</b>	<b>50</b>
6.1	Vytváření řetězců . . . . .	50
6.2	Základní manipulace s řetězci . . . . .	52
6.3	Funkce pro manipulaci s řetězci . . . . .	53
6.4	Funkce <code>eval</code> a <code>feval</code> . . . . .	54
<b>7</b>	<b>Vyhodnocování výrazů</b>	<b>57</b>
7.1	Výraz jako textový řetězec . . . . .	57
7.2	Symbolický výraz . . . . .	59
7.3	Výraz jako <code>INLINE</code> funkce . . . . .	60
7.4	Polynomy . . . . .	60
<b>8</b>	<b>Práce se soubory</b>	<b>64</b>
8.1	Záznam práce . . . . .	64
8.2	Ukládání a načítání proměnných . . . . .	65
8.3	Soubory v systému <code>MATLAB</code> . . . . .	66
8.4	Cesta k souborům . . . . .	66
8.5	Další příkazy pro práci se soubory . . . . .	67
<b>9</b>	<b>Práce s grafikou</b>	<b>69</b>
9.1	Funkce <code>plot()</code> a její použití . . . . .	70
9.2	Vzhled grafu . . . . .	74
9.3	3D grafika . . . . .	77
9.4	Vlastnosti grafických objektů . . . . .	80
<b>10</b>	<b>Programování v <code>MATLABu</code></b>	<b>83</b>
10.1	Dávkové soubory (skripty) a funkce . . . . .	84
10.2	Lokální a globální proměnné . . . . .	86
10.3	Základní programové struktury . . . . .	86
10.3.1	Větvení programu . . . . .	86
10.3.2	Cykly . . . . .	88
10.4	Nástrahy při programování v <code>MATLABu</code> . . . . .	90
10.5	Ladění programu . . . . .	92
	<b>Seznam použité literatury</b>	<b>95</b>

# Úvod

MATLAB (z anglického MATrix LABoratory) je interaktivní programovací prostředí a skriptovací programovací jazyk pro operační systémy Windows, Linux i MacOS. Jedná se o systém vhodný pro vědecké a inženýrské výpočty, modelování, simulace, analýzu a vizualizaci dat, vývoj algoritmů a aplikací včetně tvorby grafického uživatelského rozhraní. Velké uplatnění má zejména v technických oborech a ekonomii. Možnosti MATLABu rozšiřují knihovny funkcí, tzv. toolboxy, soubory M-funkcí zaměřené na specifické účely (statistika, optimalizace, symbolické výpočty, neuronové sítě, zpracování signálů a obrazu, apod.).

Autor MATLABu, firma The MathWorks, Inc. (<http://www.mathworks.com/>), stejně jako její zástupce pro Českou republiku a Slovensko, firma HUMUSOFT (<http://www.humusoft.cz/>), poskytují svému prostředí značnou podporu. Uživatelé na výše zmíněných webových stránkách mohou získat informace o instalaci, nových produktech a rozšířeních, nabízí kurzy, školení apod. Licenční a instalační soubory jsou pro studenty a zaměstnance Masarykovy univerzity k dispozici na Intranetovém serveru MU. K systému MATLAB existují i volně šiřitelné alternativy jako Octave ([www.octave.org](http://www.octave.org)), Scilab ([www.scilab.org](http://www.scilab.org)) nebo FreeMat (<http://freemat.sourceforge.net/>), ty ovšem nedosahují takové kvality jako MATLAB.

Tento studijní text je určen především pro studenty předmětů "M4130 a M4130c Výpočetní matematické systémy", předpokládá pouze základní znalosti lineární algebry, práce na počítači a základy programování. Cílem textu je seznámit studenty se syntaxí MATLABu, maticí jako základním stavebním prvkem tohoto prostředí a maticovým uvažováním, závěr textu je věnován grafickému zobrazení dat a tvorbě vlastních skriptů a funkcí. Text je doplněn o množství jak demonstračních příkladů usnadňující pochopení příkazů, tak příkladů určených pro samostatné řešení.

Návod je psán pro verzi MATLABu 8.3. Přestože MATLAB pracuje od verze 6 s okenní strukturou, příkazová řádka i nadále zůstává základním komunikačním prostředkem a většina příkazů by měla být funkční i ve verzích nižších.

# Kapitola 1

## Začínáme s MATLABem

### Základní informace

Novější verze MATLABu pracují s tzv. okenní strukturou. Jedná se o interaktivní prostředí, ve kterém má velká část příkazů pro manipulaci a základní ovládání také své ekvivalenty v podobě klikacích ikoněk. V této kapitole se nejdříve naučíme v tomto prostředí orientovat – popíšeme jednotlivá okna, ze kterých se prostředí skládá, uvedeme jejich funkce a možnosti. Dále se zaměříme na základní syntaxi jazyka, nakonec se seznámíme s nápovědním systémem a všemi jeho možnostmi.

### Výstupy z výuky

Studenti

- se seznámí s okenní strukturou MATLABu a jejích využitím
- umí využít příkazové řádky
- ovládají základní syntaxi jazyka
- umí využívat nápovědního systému

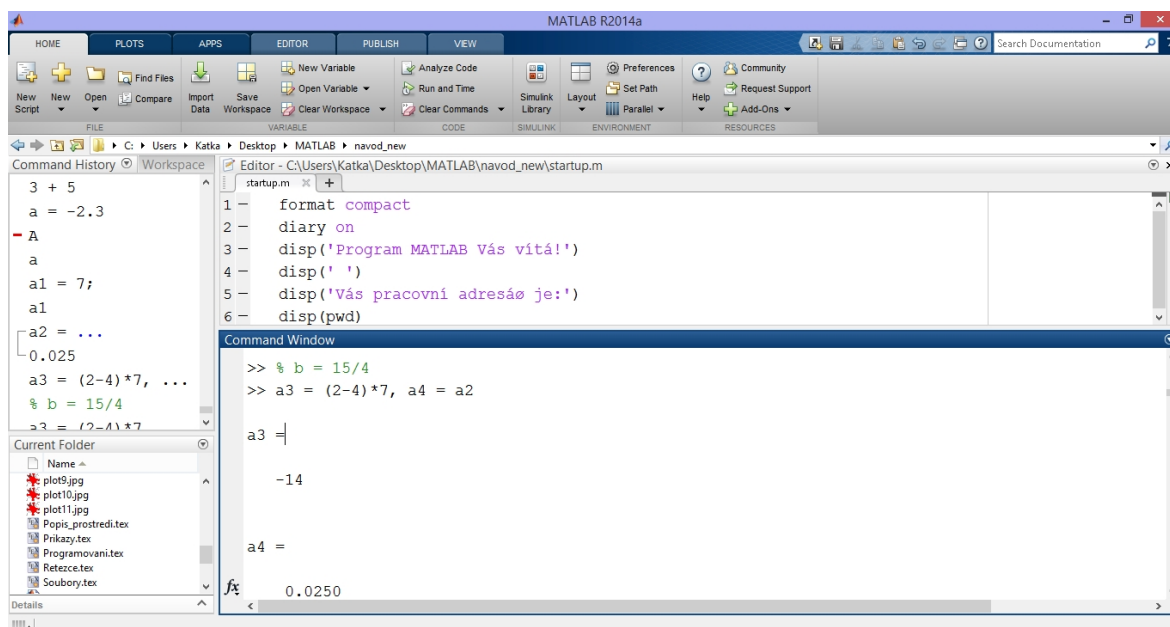
### 1.1 Popis prostředí

Spouštění MATLABu je závislé na použitém operačním systému. V Linuxu zpravidla MATLAB spouštíme zadáním příkazu `matlab` v příkazové řádce, ve Windows spouštíme kliknutím na příslušnou ikonu nebo program najdeme v menu. Dřívější verze MATLABu pracovaly výhradně s příkazovou řádkou, od verze MATLAB 6 pracují s okenní strukturou, která je uživatelsky přívětivější (Obr. 1.1).

## KAPITOLA 1. ZAČÍNÁME S MATLABEM

Základem grafického rozhraní MATLABu je *Command Window* (příkazové okno), dále *Current folder* (okno pro správu aktuální složky), *Workspace* (pracovní prostor) a *Command history* (okno historie příkazů). Jednotlivá okna lze uspořádat podle vlastního uvážení.

- *Command window* je tvořen příkazovým řádkem sloužícím ke spuštění jednotlivých příkazů, skriptů či funkcí a zobrazování jejich výstupů
- *Current folder* obsahuje informace o souborech v aktuální složce, lze měnit obsah adresáře, spouštět funkce
- *Workspace* obsahuje informace o všech definovaných proměnných (rozměr, typ, velikost v paměti), poklikáním na jejich název lze v editoru polí (*Editor*) zobrazit jejich obsah
- *Command history* je tvořen seznamem dříve použitých příkazů nejen od posledního spuštění, příkazy lze poklikáním přenést do příkazové řádky a znovu spustit nebo kliknutím pravým tlačítkem myši vybrat některou z položek kontextového menu



Obr. 1.1. Okenní struktura MATLABu.

V horní části hlavní pracovní plochy v nástrojové liště se nachází ikony pro nastavení a manipulaci s proměnnými, kódy a okny (Obr. 1.2).



Obr. 1.2. Nástrojová lišta s ikonami.

Záložka *Home* usnadňuje práci se soubory (skupina ikon *File*), obsahuje ikonky pro pohodlné otvírání již existujících souborů, vytváření nových souborů a jejich ukládání, import dat (*Import Data*), správu proměnných (skupina ikon *Variable*), dále ikonky vlastního uspořádání okenní struktury (*Layout*), nastavení vlastností (*Preferences*) či cesty k adresářům (*Set Path*).

Záložka *Plots* umožňuje grafické znázornění zvolených proměnných, záložka *Apps* obsahuje klikací prostředí pro prokládání křivek daty, optimalizaci, analýzu signálu apod. V případech, kdy je kód upravován (otevřen v okně *Editor*), se zobrazí další tři nové záložky: *Editor*, *Publish* a *View*. Tyto záložky obsahují funkce právě pro práci a úpravu kódu. Záložka *Editor* usnadňuje práci s kódy. Skupina ikon *File* umožňuje otvírání a ukládání existujících kódů, tvorbu nových kódů, prohledávání a porovnávání souborů. Skupina *Edit* obsahuje ikony pro vkládání předdefinovaných funkcí a komentářů, odsazování textu, skupina *Navigate* obsahuje ikony pro pohyb v souboru, *Breakpoints* ikony pro práci s 'breakpoints' a *Run* ikony pro spouštění kódu.

Záložka *Publish* obsahuje ikony pro grafickou úpravu kódu a *View* ikony pro členění okna editoru a vlastního kódu.

## 1.2 Základní ovládání

Jak již bylo řečeno v odstavci výše, MATLAB spustíme příkazem `matlab` v příkazové řádce nebo poklikáním na odpovídající ikonu. Program se ukončuje příkazem `exit` nebo zavřením okna, v němž program běží.

Po spuštění programu prostředí dominuje *Command window* s příkazovou řádkou, do které jsou zadávány veškeré příkazy. Příkazy je možné zadávat, pokud je aktivní prompt `>>` na začátku řádku, příkazy spouštíme klávesou ENTER. Po spuštění příkazu se na následujících řádcích objevují výstupy.

```
>> 3 + 5   příkaz s výpisem výstupní hodnoty, výstup je dočasně uchován v proměnné
ans       příkaz s výpisem výstupní hodnoty, výstup je dočasně uchován v proměnné
ans       příkaz s výpisem výstupní hodnoty, výstup je dočasně uchován v proměnné
ans =
8
```

Přiřazení příkazu do proměnné se realizuje pomocí `=`, obsah této proměnné může být kdykoliv zobrazen zadáním názvu proměnné do příkazové řádky. Při zadávání názvů proměnným bychom měli dodržovat určitá pravidla:

## KAPITOLA 1. ZAČÍNÁME S MATLABEM

---

- názvy proměnných se mohou skládat z písmen (a–z, A–Z), číslic (0–9) a podtržítka, žádné jiné symboly nejsou povoleny
- proměnné musí začínat písmenem
- proměnné mohou mít až 31 znaků
- rozlišují se malá a velká písmena (case sensitive)
- pro desetinnou čárku se používá desetinná tečka
- měli bychom se vyvarovat používání názvů vestavěných funkcí a proměnných

```
>> a = -2.3   přiřazení hodnoty do proměnné a
a =
    -2.3
>> A         MATLAB je case sensitive, proměnná A nebyla definována. Chybová hlášení
jsou zvýrazňována červenou barvou. Současně je na výstupu nabídnuta možná alter-
nativa (a), a to pouze v případě, kdy je definována proměnná podobného názvu
Undefined function or variable 'A'.
Did you mean:
>> a
```

Vložíme-li za příkaz středník (;), zamezíme vypisování výstupu na obrazovku. Na jeden řádek je možné zadávat i více příkazů, pak je oddělujeme čárkou nebo středníkem. Pokud chceme příkaz rozdělit na více řádků, můžeme použít tři tečky (...) jako pokračovací znaky.

```
>> a1 = 7;   příkaz s potlačením výpisu výstupu
>> a1       zobrazení proměnné a1
a1 =
     7
>> a2 = ...  příkaz na více řádků
0.025
a2 =
     0.025
>> a3 = (2-4)*7, a4 = a2   více příkazů na řádku s výpisem výstupů
a3 =
    -14
a4 =
     0.025
```

Provedené příkazy nelze upravit přímo, úprava je možná jen editací v dalším příkazu. K listování v historii příkazů slouží klávesy  $\uparrow$  a  $\downarrow$ . Klávesy  $\rightarrow$  a  $\leftarrow$  slouží k pohybu kurzoru v aktuální příkazové řádce. Při použití části příkazu následovaného klávesou



↑ budou nabízeny především příkazy se shodným začátkem.

Pro přerušení výpočtu prováděného příkazu lze použít klávesovou zkratku CTRL+C, smazání příkazového řádku se provádí klávesou ESC.

Symbol % se používá jako komentář, veškerý text za tímto znakem není vyhodnocován.

```
>> % b = 15/4   komentář (zobrazuje se zeleně)
>> a↑          doplní příkaz posledním definovaným příkazem začínajícím výrazem a, v našem případě a3 = (2-4)*7, a4 = a2
```

## 1.3 Nápověda

Velmi důležitou součástí softwaru MATLAB je nápovědní systém, který obsahuje kompletní dokumentaci k celým knihovnám funkcí.

Jedním ze základních příkazů pro vyvolání nápovědy je samostatný příkaz `help`, který v *Command Window* zobrazuje všechny základní tématické oblasti spolu s cestou a jednoduchým popisem.

```
>> help   výpis části seznamu tématických oblastí
HELP topics:

Documents\MATLAB           - (No table of contents file)
matlab\testframework       - (No table of contents file)
matlab\demos                - Examples.
matlab\graph2d             - Two dimensional graphs.
matlab\graph3d             - Three dimensional graphs.
matlab\graphics            - Handle Graphics.
matlab\plottools          - Graphical plot editing tools
:
```

Pro zobrazení seznamu funkcí s jednoduchým popisem pro konkrétní tématickou oblast slouží příkaz `help nizev_tematicke_oblasti`.

```
>> help graph2d   výpis části seznamu funkcí tématické oblasti graph2d
Two dimensional graphs.

Elementary X-Y graphs.
plot           - Linear plot.
loglog        - Log-log scale plot.
semilogx     - Semi-log scale plot.
semilogy     - Semi-log scale plot.
```

```
polar          - Polar coordinate plot.  
plotyy        - Graphs with y tick labels on the left and right.  
:
```

Pro zobrazení nápovědy konkrétní funkce slouží příkaz `help nazev_funkce`. Nápověda obsahuje název funkce, její základní popis, způsob použití, odkaz na hypertextovou variantu nápovědy, funkce s touto funkcí související, popř. funkce stejného názvu obsažené v jiných knihovnách.


```
>> help sin    nápověda k funkci sin (sinus)  
sin - Sine of argument in radians  
  
This MATLAB function returns the sine of the elements of X.  
  
Y = sin(X)  
  
Reference page for sin  
  
See also asin, asind, sind, sinh  
  
Other functions named sin  
fixedpoint/sin, symbolic/sin
```

Stejný způsob použití a výstupu jako funkce `help` má i funkce `helpwin` s jediným rozdílem – zobrazení výstupu v samostatném okně. Práce s nápovědním systémem se tak může stát pro uživatele přehlednější a pohodlnější.

Hypertextová nápověda zobrazuje mnohem podrobnější dokumentaci v samostatném okně. Nápovědu můžeme vyvolat samostatným příkazem `doc`, který zobrazuje seznam všech nainstalovaných toolboxů, postupným rozklikáváním jednotlivých témat se můžeme dostat k nápovědě konkrétní funkce. K ní se můžeme dostat rovněž zadáním příkazu `doc nazev_funkce`. Nápověda obsahuje kromě detailnějšího popisu samotné funkce také popis vstupních a výstupních parametrů, mnohdy obsahuje rozbalovací menu (obsahující např. grafické znázornění, příklady použití apod.).

Hypertextová nápověda umožňuje vyhledávání v knihovnách (Refine by Product), podle kategorie (Refine by Category) nebo podle typu (Refine by Type).

```
>> doc sin
```

Alternativou k příkazu `doc` je nápověda v menu *Home* → *Resources* → *ikona otazníku*  nebo *Home* → *Resources* → *Help* → *Documentation* nebo klávesová zkratka F1.


Příkazem `demo` vyvoláme tematicky rozlišený seznam demonstračních programů obsahující příklady použití či návodná videa. Alternativou je vyvolání z menu *Home* → *Resources* → *Help* → *Examples*.

Informace o aktuální verzi MATLABu a nainstalovaných knihovnách získáme použitím příkazu `ver`.

### Příklady k procvičení

1. Definujte následující proměnné:
  - proměnnou `k` s hodnotou 5
  - proměnnou `l` s hodnotou -8.3645
2. Spočítejte obsah `S` trojúhelníka, znáte-li délku strany  $a = 5$  cm (proměnná `a`) a výšku na stranu  $a$   $v_a = 3$  cm (proměnná `va`).
3. Zobrazte několika způsoby nápovědu k funkci `pow2()` a stručně ji popište.

*Řešení.*

1. `k = 5, l = -8.3645`
2. `a = 5, va = 3, S = (a * va)/2`
3. `help pow2` nebo `doc pow2` nebo záložka *Home* → *Resources* → *Help* → *Documentation*/klávesová zkratka F1/ikona otazníku  + do *Search Documentation* napsat `pow2`. Příkaz `pow2(n)` spočítá  $2^n$ .

# Kapitola 2

## Jednoduché výpočty, proměnné a matice

### Základní informace

Systém MATLAB je využíván jako nástroj pro jednoduché matematické výpočty stejně jako složitější maticové počty a algoritmy z nich vycházející. Za základní stavební kámen systému lze považovat matici, v této kapitole se proto zaměříme především na to, jak matice vytvářet, jak generovat posloupnosti či náhodné hodnoty. Na samotném závěru kapitoly se seznámíme s některými speciálními výrazy a různými způsoby výpisu hodnot.

### Výstupy z výuky

Studenti

- jsou schopni systém používat jako inteligentní kalkulačku
- umí určit funkční hodnoty elementárních matematických funkcí
- dokáží definovat proměnné, vektory a matice
- umí vytvořit nulovou, jedničkovou a jednotkovou matici, umí generovat posloupnosti a náhodné prvky
- dokáží vyjmenovat některé speciální výrazy, umí měnit způsob výpisu hodnot
- znají obecná pravidla pro definování příkazů, rozlišují vstupní a výstupní parametry

## 2.1 MATLAB jako kalkulátor

V MATLABu lze provádět libovolné výpočty, které lze běžně počítat na kalkulačce. Jednotlivé výrazy můžeme seskupovat pomocí běžných operátorů (+, -, \*, /, ^), které jsou podrobněji rozebrány v Kapitole Maticové operace. Priorita operátorů je řízena umístěním kulatých závorek, ty zároveň slouží i pro argumenty vestavěných či vlastních funkcí.

Mezi nejpoužívanější vestavěné funkce patří:

- goniometrické funkce a funkce k nim inverzní: `sin()`, `cos()`, `tan()`, `cot()`, `asin()`, `acos()`, `atan()`, `acot()`,
- hyperbolické funkce a funkce k nim inverzní: `sinh()`, `cosh()`, `tanh()`, `coth()`, `asinh()`, `acosh()`, `atanh()`, `acoth()`,
- `exp()` (exponenciální funkce), `log()` (přirozený logaritmus), `log10()` (dekadický logaritmus), `log2()` (logaritmus se základem 2), `pow2(n)` ( $n$ -tá mocnina 2), `pow2(n, m)` ( $n * 2^m$ ), `sqrt()` (2. odmocnina)
- `abs()` (absolutní hodnota)

MATLAB má ve své paměti zabudovány i některé konstanty: `pi` (Ludolfovo číslo), `i`, `j` (imaginární jednotky), `eps`, `realmin`, `realmax`.

```
>> pi    Ludolfovo číslo
ans =
    3.1416
>> 3*(15.8+3^8) - sin(pi/12) + sqrt(4.8)
ans =
    1.9732e+04    výstup značí hodnotu 1.9732 * 104
>> abs(sin(3*pi/2))
ans =
    1
```

Pro celočíselné zaokrouhlování se používají následující funkce:

`round` zaokrouhlení k nejbližšímu celému číslu  
`floor` zaokrouhlení směrem k minus nekonečnu  
`ceil` zaokrouhlení směrem k plus nekonečnu  
`fix` zaokrouhlení směrem k nule

```
>> r = round(-3.46)
r =
    -3
```

```
>> f1 = floor(-3.46)
f1 =
    -4
>> c = ceil(-3.46)
c =
    -3
>> f = fix(-3.46)
f =
    -3
```

*Poznámka.* Zaokrouhlování čísla 5: Kladné prvky s hodnotou 5 za desetinnou čárkou jsou zaokrouhovány směrem nahoru (k nejbližšímu většímu přirozenému číslu), záporné prvky s hodnotou 5 za desetinnou čárkou jsou zaokrouhovány směrem dolů (k nejbližšímu menšímu celému číslu).

```
>> round(11.5)
ans =
    12
>> round(-5.5)
ans =
    -6
```

## 2.2 Proměnné, matice a jejich definování

Výstup jakékoliv operace může být uložen (pomocí znaku =) v proměnné, kterou lze použít při dalších výpočtech. Pro připomenutí, název proměnné je libovolná posloupnost písmen, číslic a znaku `_` začínající písmenem.

```
>> x = 12*sin(pi/2)
x =
    12
>> x8 = 8*x    definování nové proměnné x8 pomocí proměnné x
x8 =
    96
```

Proměnné mohou být číselnými hodnotami, vektory či maticemi. Numerické hodnoty, resp. vektory jsou považovány za matice typu  $1 \times 1$ , resp.  $1 \times n$  či  $n \times 1$  (pro řádkový či sloupcový vektor), kde  $n$  je délka vektoru. Matice můžeme definovat výčtem jejích prvků v hranatých závorkách, přičemž jednotlivé prvky na řádku oddělujeme mezerou nebo čárkou, jednotlivé řádky matice oddělujeme středníkem.

```
>> u = [1 2 3 4]   řádkový vektor
u =
     1     2     3     4
>> v = [-1; -2; -3]   sloupcový vektor
v =
    -1
    -2
    -3
>> w = [1 3 -2]'   sloupcový vektor můžeme vytvořit i transpozicí (operátor ') řádkového vektoru
w =
     1
     3
    -2
>> A = [1 -1 2 -3; 3 0 4 5; 3.2, 5, -6 12]   matice, kvůli jednomu reálnému prvku jsou i ostatní celočíselné prvky matice zobrazovány jako desetinná čísla
A =
     1.0000    -1.0000     2.0000    -3.0000
     3.0000         0     4.0000     5.0000
     3.2000     5.0000    -6.0000    12.0000
```

Prázdnou matici lze vytvořit příkazem `[]`.

```
>> o = []
o =
     []
```

Matice lze vytvářet pomocí už definovaných proměnných, je ovšem potřeba kontrolovat, aby souhlasily typy jednotlivých proměnných.

```
>> y = [x, 2*x, 3*x]
y =
    12    24    36
>> B = [A; u]
B =
     1.0000    -1.0000     2.0000    -3.0000
     3.0000         0     4.0000     5.0000
     3.2000     5.0000    -6.0000    12.0000
     1.0000     2.0000     3.0000     4.0000
```

```
>> C = [A, v]
C =
    1.0000  -1.0000   2.0000  -3.0000  -1.0000
    3.0000         0   4.0000   5.0000  -2.0000
    3.2000   5.0000  -6.0000  12.0000  -3.0000
>> D = [A; v]   příkaz nelze provést kvůli nevyhovujícím rozměrům
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

Seznam definovaných proměnných lze získat příkazem `who`. Příkaz `whos` zobrazí seznam proměnných i s jejich typy a dalšími informacemi. Smazat definované proměnné je možno příkazem `clear nazev_promennych` (názvy proměnných jsou odděleny čárkami), příkazem `clear all` smažeme všechny definované proměnné.

```
>> who
Your variables are:
A  B  C  a  a1  a2  a3  a4  ans  c  f  fl  o  r  u
v  x  x8  y
>> clear C a3 a4 c o r y   smazání proměnných C, a3, a4, c, o, r, y
>> whos
Name  Size  Bytes  Class  Attributes

A     3x4    96  double
B     4x4   128  double
a     1x1     8  double
a1    1x1     8  double
a2    1x1     8  double
ans   1x1     8  double
f     1x1     8  double
fl    1x1     8  double
u     1x4    32  double
v     3x1    24  double
x     1x1     8  double
x8    1x1     8  double
```

## 2.3 Funkce pro tvorbu matic

Pro vytvoření matic větších rozměrů (obecně rozměrů  $m \times n$ ,  $m$  řádků,  $n$  sloupců) je možné použít některých funkcí MATLABu:



`zeros(m, n)` nulová matice (na všech pozicích jsou nulové hodnoty)  
`ones(m, n)` jedničková matice (na všech pozicích jsou hodnoty rovny jedné)  
`eye(m, n)` jednotková matice (na hlavní diagonále jsou jedničky, jinde nuly)  
`rand(m, n)` matice s náhodnými prvky mezi 0 a 1  
`randn(m, n)` matice s prvky mající standardizované normální rozdělení

Všechny výše uvedené funkce je možné zadávat pouze s jedním parametrem – v takovém případě je vytvořena čtvercová matice příslušného řádu.

```
>> Z = zeros(2,5)   nulová matice o 2 řádcích a 5 sloupcích
Z =
     0     0     0     0     0
     0     0     0     0     0
>> O = ones(3,4)   jedničková matice o 3 řádcích a 4 sloupcích
O =
     1     1     1     1
     1     1     1     1
     1     1     1     1
>> I = eye(5,8)   jednotková matice o 5 řádcích a 8 sloupcích
I =
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
>> o = ones(0,5)   prázdná matice
o =
Empty matrix: 0-by-5
>> O1 = ones(3)   čtvercová jedničková matice
O1 =
     1     1     1
     1     1     1
     1     1     1
>> R1 = rand(3,5)   náhodná matice o 3 řádcích a 5 sloupcích
R1 =
    0.8147    0.9134    0.2785    0.9649    0.9572
    0.9058    0.6324    0.5469    0.1576    0.4854
    0.1270    0.0975    0.9575    0.9706    0.8003
>> R2 = randn(4)   čtvercová náhodná matice s prvky z normálního rozložení
R2 =
    -0.2050    1.4172    1.6302    -0.3034
    -0.1241    0.6715    0.4889    0.2939
     1.4897   -1.2075    1.0347   -0.7873
     1.4090    0.7172    0.7269    0.8884
```

Jelikož jsou vektory považovány za matice s jedním řádkem nebo jedním sloupcem, pomocí výše uvedených funkcí a nastavením jednoho z argumentů na hodnotu 1 můžeme vytvářet i vektory. Vektor, jehož prvky tvoří aritmetickou posloupnost (tzv. „ekvidistantní“ vektor), je možné vytvořit pomocí operátoru `:` (dvojtečka). Příkaz `a : b` vygeneruje aritmetickou posloupnost prvků od `a` do `b` s krokem 1. Pro jiný krok mezi jednotlivými prvky lze použít příkaz `a : d : b`, kde `d` je délka kroku, je možno použít i záporný krok.

```
>> x = 1:10    vytvoření posloupnosti s krokem 1
x =
     1     2     3     4     5     6     7     8     9    10
>> y = 0:2:12  vytvoření posloupnosti sudých čísel
y =
     0     2     4     6     8    10    12
>> z = 0.5 : 0.1 : 1.1
z =
    0.5    0.6    0.7    0.8    0.9     1    1.1
>> x1 = 15:-2:7 vytvoření posloupnosti se záporným krokem
x1 =
    15    13    11     9     7
>> x2 = 2:3:15
x2 =
     2     5     8    11    14
```

Příkazy `a : b`, popř. `a : d : b`, lze nahradit příkazy `colon(a, b)`, popř. `colon(a, d, b)`.

```
>> x3 = colon(2,3,15)  analogie k 2:3:15
x3 =
     2     5     8    11    14
```

Další alternativou pro generování ekvidistantních vektorů jsou příkazy `linspace()` a `logspace()`:

`x = linspace(a, b, n)` generuje aritmetickou posloupnost prvků `x` od `a` do `b`. Třetí parametr `n` je volitelný, udává počet prvků posloupnosti `x`, jeho implicitní nastavení na `n=100` lze libovolně měnit.

`x = logspace(a, b, n)` generuje vektor `x` s desítkovou logaritmickou škálou prvků od  $10^a$  do  $10^b$ . Třetí parametr délky posloupnosti `n` je volitelný, jeho implicitní nastavení je `n=50`. Příkaz je ekvivalentní příkazu `10.^linspace(a, b, n)`.

```
>> x = linspace(-1, 4, 8)
x =
   -1.0000   -0.2857    0.4286    1.1429    1.8571    2.5714    3.2857
   4.0000
```

```
>> x = logspace(0, 1, 5)   ekvivalentní příkazu 10.^linspace(0, 1, 5)
x =
    1.0000    1.7783    3.1623    5.6234   10.0000
>> x = 10.^linspace(0, 1, 5)
x =
    1.0000    1.7783    3.1623    5.6234   10.0000
```

## 2.4 Některé speciální výrazy a funkce

Zvláštní postavení mezi proměnnými má proměnná `ans` (z anglického *answer*), do které se automaticky přiřazují hodnoty, jež nebyly přiřazeny explicitně do proměnné.

```
>> pi
ans =
    3.1416
```

Dále `i` a `j` jsou imaginární jednotky pro práci s komplexními čísly. Další speciální hodnotou je `eps` – je to rozdíl mezi 1 a nejbližším vyšším zobrazitelným číslem. Příkazem `realmin` a `realmax` zjistíme hodnotu nejmenšího, resp. největšího zobrazitelného čísla podle normy IEEE. Hodnoty `Inf` a `-Inf` vznikají např. při dělení nulou a symbolizují nekonečné hodnoty ( $+\infty$  nebo  $-\infty$ ). Hodnotu `NaN` (Not a Number) dostaneme jako výsledek neurčitého výrazu – např. `0/0`.

Způsob výpisu hodnot můžeme ovlivnit příkazem `format`:

<code>format long</code>	výpis na plný počet desetinných míst
<code>format short</code>	výpis na omezený počet desetinných míst
<code>format bank</code>	výpis na 2 desetinná místa (implicitní)
<code>format hex</code>	hexadecimální výpis
<code>format rat</code>	hodnoty jsou aproximovány zlomky
<code>format compact</code>	potlačí se vynechávání řádku při výpisu
<code>format loose</code>	zapne se vynechávání řádku při výpisu

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
    3.141592653589793
```

```
>> format rat
>> pi
ans =
    355/113
>> abs(1+i)    absolutní hodnota komplexního čísla  $a+i\cdot b$  je definována jako  $\sqrt{a^2 + b^2}$ 
ans =
    1.4142
```

## 2.5 Obecná pravidla pro příkazy

Na jednu řádku můžeme zadat několik příkazů oddělených čárkou nebo středníkem, který potlačuje výstup na obrazovku:

příkaz1, příkaz2, příkaz3, atd.,

přičemž za posledním příkazem na řádce čárka být nemusí. Pokud je příkaz příliš dlouhý, můžeme ukončit řádku třemi tečkami (...), příkaz pak pokračuje na další řádce.

Jednotlivé příkazy mohou být jednoduché příkazy MATLABu (`who`, `clear`, `dir`), příkazy definující proměnné (`A = [1 2; 3 4];`), volání MATLABovských programů – tzv. skriptů, nebo volání MATLABovských funkcí. Toto volání má v obecném případě tvar:

```
[v1, v2, ..., vm] = nazev_funkce(p1, p2, ..., pn)
```

`v1, ..., vm` jsou výstupní parametry, `p1, ..., pn` jsou parametry vstupní. Pokud je výstupní parametr jen jeden, nemusí být uzavřen v hranatých závorkách. Funkce také nemusí mít žádné vstupní nebo výstupní parametry, také je možné zadávat různý počet vstupních nebo výstupních parametrů pro tutéž funkci.

```
>> A = rand(3);
>> s = size(A)
s =
     3     3
>> [x, y] = size(A)
x =
     3
y =
     3
>> s1 = size(A, 2)
s1 =
     3
```

Vykřičník na začátku příkazu provede příkaz operačního systému, např. příkazem `!netscape` spustíme známý internetový prohlížeč.

## Příklady k procvičení

1. Spočítejte velikost úhlu  $\alpha$  při vrcholu  $A$  trojúhelníku  $ABC$ , znáte-li délku protilehlé strany  $a = 5$  cm a přilehlé strany  $b = 3$  cm.
2. Vypište hodnotu Ludolfova čísla, pak:
  - a) jej zaokrouhlete k nejbližšímu celému číslu,
  - b) jej zaokrouhlete k plus nekonečnu,
  - c) jej zobrazte jako zlomek,
  - d) jej zobrazte na plný počet desetinných míst.
3. Třemi způsoby vygenerujte vektor  $u1$  délky 10 s počáteční hodnotou 2 a krokem 0.2.
4. Vygenerujte náhodný vektor délky 5
  - a) s názvem  $u2$  a prvky z intervalu  $[0, 1]$ ,
  - b) s názvem  $u3$  a prvky z intervalu  $[-2, 4]$ ,
  - c) s názvem  $u4$  a celočíselnými prvky z intervalu  $[-8, 1]$ .
5. Vygenerujte matici s rozměry  $5 \times 3$ 
  - a) s názvem  $A1$  a prvky z intervalu  $[-12, 5]$ ,
  - b) s názvem  $A2$  a celočíselnými prvky z intervalu  $[1, 7]$ .
6. Vytvořte matici  $B$  rozměrů  $4 \times 5$  sestavenou z vektoru  $u4$  a matice  $A2$ .
7. Vytvořte matici rozměrů  $4 \times 9$ 
  - a) s názvem  $C1$ , maticí  $B$  v levém bloku a jednotkovou maticí v pravém bloku,
  - b) s názvem  $C2$ , maticí  $B$  v levém bloku a maticí obsahující hodnoty 0.5 v pravém bloku.

*Řešení.*

1.  $a = 5, b = 3$   
 $\alpha = \tan(a/b)$
2. `exp(1)`
  - a) `round(exp(1))`
  - b) `ceiling(exp(1))`
  - c) `format rat, exp(1)`
  - d) `format long, exp(1)`

3. 1. způsob: `u1 = 2:0.2:3.8`,  
2. způsob: `u1 = colon(2, 0.2, 3.8)`,  
3. způsob: `u1 = linspace(2, 3.8, 10)`
4. a) `u2 = rand(1, 5)`  
b) `u3 = 6*rand(1, 5) - 2`  
c) `u4 = round(9*rand(1, 5) - 8)`
5. a) `A1 = rand(17*rand(5, 3) - 12)`  
b) `A2 = round(6*rand(5, 3)+1)`
6. `B = [u4; A2']`
7. a) `C1 = [B, eye(4)]`  
b) `C2 = [B, 0.5 + zeros(4)]` nebo  
`C2 = [B, 0.5 + zeros(4, 4)]` nebo  
`C2 = [B, 0.5 + ones(4)]` nebo  
`C2 = [B, 0.5 + ones(4, 4)]`

# Kapitola 3

## Maticové operace

### Základní informace

Již ze samotného názvu systému MATLAB, který pochází z anglického MATrix LABoratory, je zřejmé, že zde hlavní roli při práci bude hrát matice. Proto je velmi důležité se nejen naučit porozumět maticovým operacím, ale také je umět výhodně využívat. Cílem následující kapitoly je rozšířit již známé maticové operace o operace méně známé a demonstrovat práci s nimi na jednoduchých příkladech.

### Výstupy z výuky

Studenti

- seznámí se s transpozicí reálných i komplexních matic
- ovládají sčítání a odčítání matic
- umí správně použít maticové násobení a umocňování matic
- umí vysvětlit rozdíl mezi pravostranným a levostranným násobením a dělením matic
- dokáží definovat operace po složkách

### 3.1 Transpozice matic

Transpozici matic označuje jednoduchý apostrof ('), případně jednoduchý apostrof s tečkou (.'). V případě reálných matic mezi těmito operátory není rozdíl, oba definují transponovanou matici. Pokud ovšem má matice  $A$  komplexní prvky, příkazem  $X=A'$

získáme matici  $X$ , která vznikne transpozicí matice  $A$ , ale také její prvky budou komplexně sdružené k prvkům matice  $A$ , tj.  $x_{ij} = \overline{a_{ji}}$ .

Pokud bychom chtěli získat pouze transponovanou matici  $X$  bez komplexně sdružených prvků, tj.  $x_{ij} = a_{ji}$ , je třeba použít jednoduchý apostrof s tečkou (`.`').

```
>> A = [1 1+i; 2-3*i i]
A =
    1.0000 + 0.0000i    1.0000 + 1.0000i
    2.0000 - 3.0000i    0.0000 + 1.0000i
>> X = A'    transpozice s komplexně sdruženými prvky
X =
    1.0000 + 0.0000i    2.0000 + 3.0000i
    1.0000 - 1.0000i    0.0000 + 1.0000i
>> X = A.'    transpozice bez komplexně sdružených prvků
X =
    1.0000 + 0.0000i    2.0000 - 3.0000i
    1.0000 + 1.0000i    0.0000 + 1.0000i
```

## 3.2 Sčítání a odčítání matic

Symbole  $+$  a  $-$  označují sčítání a odčítání matic. Matice musí mít shodné dimenze. Operace jsou prováděny po složkách, tj.  $x_{ij} = a_{ij} \pm b_{ij}$ . Přičítání a odčítání konstanty k matici se realizuje po složkách, tj. konstanta je přičtena, popř. odečtena, ke každému prvku matice.

```
>> A = [1 5; 2 3];
>> B = [0 1; -3 2];
>> X = A + B
X =
     1     6
    -1     5
>> c = 2;
>> X = A + c    odpovídá  $x_{ij} = a_{ij} + c$ 
X =
     3     7
     4     5
>> X = c - B    odpovídá  $x_{ij} = c - b_{ij}$ 
X =
     2     1
     5     0
```



### 3.3 Maticové násobení

Symbol  $*$  označuje maticové násobení. Operace je definována, pokud jsou vnitřní rozměry dvou operandů stejné, tj. pokud je počet sloupců prvního činitele shodný s počtem řádků druhého činitele. Nechť  $A$  je typu  $n \times k$  a  $B$  typu  $k \times m$ , výsledná matice  $X$  bude typu  $n \times m$  a platí  $x_{ij} = \sum_{r=1}^k a_{ir} b_{rj}$ .

Při násobení matice konstantou je operace provedena po složkách.

```
>> A = [1 5 0; -1 2 3]
A =
     1     5     0
    -1     2     3
>> B = [0 1 1; 1 -3 2; -1 4 2; 1 0 3]
B =
     0     1     1
     1    -3     2
    -1     4     2
     1     0     3
>> X = A * B    součin se neprovede kvůli nesouhlasným rozměrům činitelů
Error using *
Inner matrix dimensions must agree.
>> X = A * B'
X =
     5    -14    19     1
     5     -1    15     8
>> c = 2;
>> X = A * c    odpovídá  $x_{ij} = a_{ij} * c$ 
X =
     2    10     0
    -2     4     6
>> X=c*B    odpovídá  $x_{ij} = c * b_{ij}$ 
X =
     0     2     2
     2    -6     4
    -2     8     4
     2     0     6
```

### 3.4 Maticové dělení

V MATLABu existují dva způsoby dělení matic – levostranné  $\backslash$  a pravostranné  $/$  dělení. Obecně platí

## KAPITOLA 3. MATICOVÉ OPERACE

---

$$\begin{aligned} X = A \setminus B & \quad \text{je řešením } A * X = B, \\ X = B / A & \quad \text{je řešením } X * A = B. \end{aligned}$$

Je-li  $A$  regulární čtvercová matice, potom  $X = A \setminus B$ , resp.  $X = B/A$ , formálně odpovídají levostrannému, resp. pravostrannému, násobení matice  $B$  maticí inverzní k matici  $A$ ; tj.  $\text{inv}(A)*B$  resp.  $B*\text{inv}(A)$ . Funkce `inv()` slouží pro výpočet inverzní matice, výpočet pomocí levostranného, resp. pravostranného, dělení je získán přímo, bez výpočtu inverze.

Je-li matice  $A$  obecně typu  $\underline{k} \times n$ ,  $B$  musí být typu  $\underline{k} \times m$ , výsledná matice  $X = A \setminus B$  bude typu  $n \times m$ . Pravostranné dělení  $X = B/A$  je definováno pomocí levostranného dělení jako  $X = B/A$ , což je ekvivalentní  $(A' \setminus B')$ .

Při dělení matice konstantou se dělení provádí po složkách, výsledek pravostranného i levostranného dělení je stejný.

```
>> A = [1 5 0; -1 2 3; 1 2 1];
>> b = [1 3 2]';
>> x = A \ b    můžeme se přesvědčit, že tento příkaz je ekvivalentní příkazu x =
inv(A) * b
x =
    0.6875
    0.0625
    1.1875
>> x = inv(A) * b
x =
    0.6875
    0.0625
    1.1875
>> c = 2;
>> x = c \ A
x =
    0.5000    2.5000         0
   -0.5000    1.0000    1.5000
    0.5000    1.0000    0.5000
>> x = A / c
x =
    0.5000    2.5000         0
   -0.5000    1.0000    1.5000
    0.5000    1.0000    0.5000
```

### 3.5 Umocňování matic

Pro maticové umocňování se používá symbol  $\wedge$ . Předpokládejme, že  $A$  je čtvercová matice a  $p$  je skalár. Při umocňování mohou nastat tyto případy:

1.  $X = A \wedge p$

(a)  $p > 0$  celé číslo  $\Rightarrow X = \underbrace{A * A * \dots * A}_p$

(b)  $p = 0 \Rightarrow X = I \dots$  jednotková matice (lze vytvořit příkazem `eye(size(A))`)

(c)  $p < 0$  celé číslo  $\Rightarrow X = \underbrace{A^{-1} * A^{-1} * \dots * A^{-1}}_p$

(d)  $p$  není celé číslo  $\Rightarrow$  uvažujme diagonální matici  $D = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$ , kde

$\lambda_1, \dots, \lambda_n$  jsou vlastní čísla matice  $A$ , a matici  $V$ , jejíž sloupce jsou vlastní vektory příslušné postupně těmto vlastním číslům (`[V,D] = eig(A)`). Platí tedy vztah  $A * V = V * D$ , což je v tomto případě jakási motivace pro výpočet

mocniny. Lze ukázat, že platí  $A^p * V = V * D^p$ , kde  $D^p = \begin{pmatrix} \lambda_1^p & & 0 \\ & \ddots & \\ 0 & & \lambda_n^p \end{pmatrix}$ .

Odtud dostáváme vztah pro výpočet  $X = V * D^p / V$ .

2.  $X = p \wedge A$

Při výpočtu opět vycházíme ze vztahu  $A * V = V * D$ , kde  $V$  a  $D$  jsou výše

popsané matice. Odtud je  $p^A * V = V * p^D$ , kde  $p^D = \begin{pmatrix} p^{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & p^{\lambda_n} \end{pmatrix}$ . Odtud

opět dostáváme vztah pro výpočet  $X = V * p^D / V$ .

Pokud matice  $A$  není čtvercová nebo pokud  $p$  není skalár, výpočet skončí chybovým hlášením.

```
>> A = [1 3; 3 2];
>> p = 0.5;
```

```

>> [V, D] = eig(A)   výpočet vlastních čísel (diagonální prvky matice D) a vlastních
vektorů (sloupce matice V)
V =
    -0.7630    0.6464
     0.6464    0.7630
D =
    -1.5414         0
         0    4.5414
>> X = A^p
X =
    0.8904 + 0.7228i    1.0510 - 0.6123i
    1.0510 - 0.6123i    1.2407 + 0.5187i
>> X = V*D^p/V
X =
    0.8904 + 0.7228i    1.0510 - 0.6123i
    1.0510 - 0.6123i    1.2407 + 0.5187i
>> Y = p^A
Y =
     1.7126    -1.4144
    -1.4144     1.2411
>> Y = V*p^D/V
Y =
     1.7126    -1.4144
    -1.4144     1.2411

```

### 3.6 Operace po složkách

V MATLABu lze též provádět tzv. „operace po složkách“. Tyto operace se definují tak, že před operaci, kterou chceme provádět po složkách, zadáme tečku (.). U operací, které se automaticky provádí po složkách (sčítání a odčítání), přidání tečky nemá smysl a výpočet končí chybovým hlášením. Uvažujme tedy libovolný maticový operátor  $\circ \in \{*, /, \setminus, ^\}$ . Pro provádění operací po složkách musí být obě matice shodné dimenze, výstup bude mít tutéž dimenzi. Obecně tedy pro prvky matice  $X=A \circ B$  platí  $x_{ij} = a_{ij} \circ b_{ij}$ .

*Poznámka.* Mezi operátory, u nichž má význam operace po složkách patří také transpozice, která je blíže popsána v odstavci Transpozice matic.

```

>> A = [1 2; 3 4]
A =
     1     2
     3     4

```

## KAPITOLA 3. MATICOVÉ OPERACE

---

```
>> B = [2 4;6 8]
A =
     2     4
     6     8
>> X = A*B
X =
    14    20
    30    44
>> X = A.*B
X =
     2     8
    18    32
>> X = B/A
X =
     2     0
     0     2
>> X = B./A
X =
     2     2
     2     2
>> X = B^A   neexistuje
Error using ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
>> X = B.^A
X =
     2     16
    216   4096
```

Operace po složkách lze také provádět tehdy, pokud jedna z matic je skalár. V tomto případě má tečka smysl pouze u operace umocňování, neboť ostatní operace se provedou po složkách automaticky.

```
>> c = 2;
>> X = A/c   dělení konstantou s operátorem / vrací stejný výsledek jako s operátorem ./
X =
    0.5000    1.0000
    1.5000    2.0000
>> X = A./c
X =
    0.5000    1.0000
    1.5000    2.0000
```

```
>> X = A^c   při umocňování je tečka podstatná

X =
     7  10
    15  22
>> X = A.^c
X =
     1   4
     9  16
>> X = c^A
X =
    10.4827  14.1519
    21.2278  31.7106
>> X = c.^A
X =
     2   4
     8  16
```

### 3.7 Logické operace

Mezi tyto operace patří relační operátory a logické funkce. Více se o nich dozvíme v Kapitole 5.

### 3.8 Smíšené operace

Jednotlivé operace (logické i aritmetické) je možné vzájemně efektivně kombinovat. Priorita operací je následující (od nejvyšší po nejnižší):

1. umocňování:  $\wedge$   $\cdot^{\wedge}$
2. násobení, dělení:  $*$   $\cdot^*$   $\backslash$   $\cdot\backslash$   $/$   $\cdot/$
3. sčítání, odčítání:  $+$   $-$
4. relační operátory:  $==$   $\sim=$   $<$   $<=$   $>$   $>=$
5. negace:  $\sim$
6. logické spojky 'a' a 'nebo':  $\&$   $|$

Prioritu výše uvedených operátorů můžeme řídit pomocí kulatých závorek.

Všechny operátory pro práci s maticemi uvedené v této kapitole mají své ekvivalenty:

funkce	význam	operátor
plus	plus	+
uplus	unární plus	+
minus	minus	-
uminus	unární minus	-
mtimes	maticové násobení	*
times	násobení po prvcích	.*
mpower	maticová mocnina	^
power	mocnina po prvcích	.^
mldivide	levé maticové dělení	\
mrdivide	pravé maticové dělení	/
ldivide	levé dělení po prvcích	.\
rdivide	pravé dělení po prvcích	./

## Příklady k procvičení

- Jsou dány matice  $A = [1 \ 0 \ -1; \ 2 \ 1 \ 1]$ ,  $B = [2 \ -2 \ 1; \ 0 \ 1 \ 2]$ ,  $C = [1 \ 0 \ 1; \ 0 \ 1 \ 1; \ 1 \ 0 \ 0]$  a  $D = [3 \ 5 \ -6; \ 1 \ 1 \ -2; \ 0 \ -2 \ 0]$ . Vyřešte následující rovnice:
  - $A + X1 = B$ ,
  - $(A - X2) * C = B$ ,
  - $A'*(X3 - B) = D$ .
- Vytvořte komplexní matici  $F$ , jejíž reálná část bude tvořena maticí  $X1$  a imaginární část maticí  $X2$ .
  - Vytvořte k ní transponovanou matici  $F1$ .
  - Vytvořte k ní transponovanou matici  $F2$ , která bude navíc obsahovat komplexně sdružené prvky.
- Je dána matice  $E = [1 \ 2; \ -1 \ 1]$ .
  - Vytvořte matici  $E1$ , která bude druhou mocninou matice  $E$ .
  - Vytvořte matici  $E2$ , která bude obsahovat druhé mocniny prvků matice  $E$ .
  - Vytvořte matici  $E3$  funkčních hodnot funkce kotangens v bodech  $E$ .

*Řešení.*

- $A = [1 \ 0 \ -1; \ 2 \ 1 \ 1]$ ,  $B = [2 \ -2 \ 1; \ 0 \ 1 \ 2]$ ,  $C = [1 \ 0 \ 1; \ 0 \ 1 \ 1; \ 1 \ 0 \ 0]$ ,  
 $D = [3 \ 5 \ -6; \ 1 \ 1 \ -2; \ 0 \ -2 \ 0]$ 
  - $X1 = B - A$
  - $X2 = A - B/C$
  - $X3 = A' \setminus D+B$
- $F = X1 + i*X2$ 
  - $F1 = F.'$
  - $F2 = F'$

3.  $E = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}$

a)  $E_1 = E^2$

b)  $E_2 = E.^2$

c)  $E_3 = \cot(E)$



# Kapitola 4

## Manipulace s maticemi

### Základní informace

Pro praktické řešení problémů je důležitá dobrá orientace v možnostech, které systém nabízí. Následující kapitola nabízí přehled a praktické ukázky použití nejdůležitějších a často používaných funkcí a příkazů pro manipulaci s maticemi.

### Výstupy z výuky

Studenti

- dokáží vytvářet submatice a zjistit rozměry matice
- seznámí se s maticovými operacemi - sčítání, odčítání, násobení, dělení
- umí použít funkci `reshape()` a znají funkce pro překlápění a otáčení matice
- ovládají základní funkce lineární algebry
- demonstrují práci s diagonálami matice
- aplikují funkce `min()`, `max()`, `sum()`, `prod()`

### 4.1 Základní manipulace s maticemi

Pro zjištění rozměrů matice slouží příkaz `size()`. Tento příkaz má i volitelný druhý argument – v případě hodnoty 1 vrací počet řádků matice, v případě hodnoty 2 vrací počet sloupců matice. Příkaz `size()` lze použít i pro zjištění rozměrů vektoru, alternativním příkazem pro počet prvků (řádkového i sloupcového) vektoru je příkaz `length()`. Příkaz `length()` lze použít i pro matici, v tomto případě vrací její největší

rozměr.

```
>> A = [1 -1 2 -3; 3 0 4 5; 3.2, 5, -6 12]
A =
    1.0000   -1.0000    2.0000   -3.0000
    3.0000         0    4.0000    5.0000
    3.2000    5.0000   -6.0000   12.0000
>> [r, sl] = size(A)   výstup funkce size() lze uložit do 2-prvkového vektoru,
první prvek udává počet řádků matice A, druhý počet sloupců
r =
     3
sl =
     4
>> r1 = size(A, 1)   počet řádků matice A
r1 =
     3
>> u = [3 -1 2]
u =
     3  -1  2
>> size(u)
ans =
     1     3
>> length(u)   počet prvků vektoru
ans =
     3
>> length(A)   největší rozměr matice A
ans =
     4
```

Na jednotlivé prvky matice je možné se odkazovat pomocí kulatých závorek – tj.  $A(r,sl)$  je prvek matice  $A$  na  $r$ -tém řádku a v  $s$ -tém sloupci. Toto vyjádření lze použít i obecněji, kdy první parametr je vektor obsahující indexy vybraných řádků a druhý parametr je vektor sloupcových indexů.

```
>> A(2,3)   prvek na 2. řádku a ve 3. sloupci matice A
ans =
     4
>> A([1 3], [4 2])   prvky na 1. a 3. řádku a 4. a 2. sloupci matice A
ans =
    -3  -1
    12   5
```

```
>> A([1 3],[4 2]) = eye(2) do takto vybraných prvků je možno také při zachování
správných rozměrů provádět přiřazení
A =
    1.0000         0    2.0000    1.0000
    3.0000         0    4.0000    5.0000
    3.2000    1.0000   -6.0000    0.0000
```

Symbol `:` (dvojtečka) slouží pro výběr celého řádku/sloupce matice.

```
>> u1 = A(:, 3)  výběr 3. sloupce matice A
u1 =
     2
     4
    -6
>> u2 = A(2,:)  výběr 2. řádku matice A
u2 =
     3     0     4     5
>> A(2,:) = []  smazání 2. řádku matice A
A =
    1.0000         0    2.0000    1.0000
    3.2000    1.0000   -6.0000    0.0000
```

*Poznámka.* Přiřazení `o = []`; `A(:,3) = o` nefunguje, vrací chybové hlášení.

Matice stejných rozměrů lze sčítat a odčítat (operátory `+`, `-`), při přičtení konstanty k matici je konstanta přičtena ke každému prvku matice.

Matice vhodných rozměrů lze násobit (operátor `*`), při násobení matice konstantou je konstantou přenásoben každý prvek matice. Dělení je v MATLABu dvojího druhu – pravostranné a levostranné: `\` a `/`.

Všechny operátory jsou podrobněji popsány v Kapitole 3.

Na matice lze rovněž aplikovat všechny běžné matematické funkce (`sin()`, `sqrt()`, ...), které jsou popsány v Odstavci MATLAB jako kalkulátor. Tyto funkce jsou na matice aplikovány po složkách (člen po členu).

## 4.2 Změna struktury matice

V MATLABu je možné měnit strukturu již existujícím objektům, např. přeskádat matici do matice jiných rozměrů. Příkazem `B = reshape(A, m, n)` lze po sloupcích přeskádat matici `A` do matice `B` o `m` řádcích a `n` sloupcích. Obě matice `A` i `B` musí mít stejný počet prvků, v opačném případě příkaz skončí chybovým hlášením.

```
>> A = [1:6; 7:12]
A =
     1     2     3     4     5     6
     7     8     9    10    11    12
>> B = reshape(A, 3, 4)
B =
     1     8     4    11
     7     3    10     6
     2     9     5    12
```

Pokud bychom chtěli matici přeskádat po řádcích, museli ji bychom nejdřív transponovat a nakonec opět transponovat celý příkaz.

```
>> B = (reshape(A', 4, 3))'
B =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Překlopit matici, tj. obrátit pořadí řádků, resp. sloupců, je možné provést příkazem `flipud()`, resp. `fliplr()`. Matici lze „otočit“ o 90 stupňů proti směru hodinových ručiček příkazem `rot90()`, přičemž další nepovinný parametr udává, kolikrát má být rotace provedena.

```
>> C1 = flipud(B)   převrácené pořadí řádků
C1 =
     9    10    11    12
     5     6     7     8
     1     2     3     4
>> C2 = fliplr(B)  převrácené pořadí sloupců
C2 =
     4     3     2     1
     8     7     6     5
    12    11    10     9
>> C3 = rot90(B, 3)   třikrát otočená matice B o 90° proti směru hodinových ručiček
C3 =
     9     5     1
    10     6     2
    11     7     3
    12     8     4
```

Jedním z dalších příkazů, které využívají dvojtečku, je `A(:)`. `A(:)` na pravé straně přiřazovacího příkazu označuje všechny sloupce matice `A` seskládané do jednoho dlouhého

sloupcového vektoru.

```
>> A = [1 2; 3 4; 5 6]
A =
     1     2
     3     4
     5     6
>> b = A(:)
b =
     1
     2
     3
     4
     5
     6
```

Pokud již matice  $A$  existuje, lze  $A(:)$  použít i na levé straně přiřazovacího příkazu ke změně tvaru nebo velikosti matice. Potom  $A(:)$  označuje matici svými rozměry stejnou matici  $A$ , prvky za přiřazovacím příkazem jsou do matice  $A$  umisťovány po sloupcích. Tato operace je zahrnuta ve funkci `reshape()`.

```
>> A(:) = 11:16   změní šestiprvkový řádkový vektor 11:16 na matici stejných roz-
měrů jako A, tj. 3 × 2
A =
    11    14
    12    15
    13    16
```

### 4.3 Základní funkce lineární algebry

V MATLABu je implementováno velké množství funkcí a algoritmů lineární algebry. Podrobný výpis lze získat pomocí nápovědy `"help matfun"`.

Zde vyjmenujeme jen některé základní:

<code>det()</code>	determinant matice
<code>rank()</code>	hodnost matice
<code>norm()</code>	maticová nebo vektorová norma
<code>trace()</code>	stopa matice (součet diagonálních prvků)
<code>inv()</code>	inverzní matice
<code>pinv()</code>	pseudoinverzní matice
<code>lu()</code>	LU rozklad

`qr()` QR rozklad  
`svd()` singulární rozklad  
`eig()` vlastní hodnoty a vektory matice  
`poly()` charakteristický polynom matice

Syntaxi jednotlivých funkcí zjistíme nejlépe pomocí nápovědy.

```
>> A = [3 -1; 2 0];
>> d = det(A)    determinant
d =
     2
>> h = rank(A)   hodnost matice A
h =
     2
>> tr = trace(A) stopa matice A, ekvivalentní příkaz: sum(diag(A))
tr =
     3
>> [L, U] = lu(A) rozklad matice A na dolní (L) a horní (U) trojúhelníkovou ma-
tici, jejich součinem dostáváme původní matici (A = L*U)
L =
     1.0000     0
     0.6667     1.0000
U =
     3.0000    -1.0000
         0     0.6667
>> p = poly(A)   charakteristický polynom matice A, výstup odpovídá polynomu
 $p(x) = x^2 - 3x + 2$ 
p =
     1     -3     2
```

## 4.4 Další funkce pro manipulaci s maticemi

Dvojitou úlohu má funkce `diag()`. Její aplikací na vektor získáme diagonální matici s argumentem na hlavní diagonále. Pokud funkci použijeme na matici, výstupem je vektor jejích diagonálních prvků. Pokud chceme pracovat s jinou diagonálou než s hlavní, můžeme použít jako druhý parametr funkce `diag()` číslo diagonály, přičemž kladná čísla se použijí pro diagonály nad hlavní diagonálou a záporná pod ní.

```
>> A = round(10*rand(5))    celočíselná matice s prvky od 0 do 10
A =
     8     7     8     4     5
     7     0     7     4     4
     4     3     3     8     6
     7     0    10     8     7
     2     1     0     2     8
>> x = diag(A)    vektor diagonálních prvků
x =
     8
     0
     3
     8
     8
>> B = diag(x,2)    nulová matice s prvky x na 2. diagonále nad hlavní diagonálou
B =
     0     0     8     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     3     0     0
     0     0     0     0     0     8     0
     0     0     0     0     0     0     8
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
>> C = diag(pi, -2)
C =
         0     0     0
         0     0     0
    3.1416     0     0
>> D = diag(diag(A))    diagonální matice
D =
     8     0     0     0     0
     0     0     0     0     0
     0     0     3     0     0
     0     0     0     8     0
     0     0     0     0     8
```

Pomocí funkcí `tril()` a `triu()` vyrobíme z dané matice dolní nebo horní trojúhelníkovou matici, přičemž je možné podobně jako u funkce `diag()` použít další nepovinný parametr pro posunutí diagonály.

```
>> L1 = tril(A)   dolní trojúhelníková matice
L1 =
     8  0  0  0  0
     7  0  0  0  0
     4  3  3  0  0
     7  0 10  8  0
     2  1  0  2  8
>> L2 = tril(A,2)
L2 =
     8  7  8  0  0
     7  0  7  4  0
     4  3  3  8  6
     7  0 10  8  7
     2  1  0  2  8
```

Funkce `max()` hledá maximální prvek vektoru, pokud na výstupu uvedeme i druhý výstupní parametr, uloží se do něj index maximálního prvku. Při použití funkce `max()` na matici se hledají maximální prvky v jednotlivých sloupcích, do volitelného druhého výstupního parametru jsou ukládány řádkové indexy maximálního prvku v daném sloupci. Podobně funguje funkce `min()`.

```
>> x = rand(1,6)
x =
    0.2760    0.6797    0.6551    0.1626    0.1190    0.4984
>> M = max(x)
M =
    0.6797
>> [m, ind] = min(x)   minimální prvek vektoru x i se svou pozicí
m =
    0.1190
ind =
     5
>> A = rand(4)   náhodná matice
A =
    0.9597    0.7513    0.8909    0.1493
    0.3404    0.2551    0.9593    0.2575
    0.5853    0.5060    0.5472    0.8407
    0.2238    0.6991    0.1386    0.2543
>> [M, r_ind] = max(A)
M =
    0.9597    0.7513    0.9593    0.8407
r_ind =
     1     1     2     3
```



U komplexních matic se maximum či minimum hledá mezi absolutními hodnotami prvků.

```
>> K = round(2*rand(3)) + i*round(2*rand(3))   náhodná komplexní matice
s celočíselnými hodnotami reálné a imaginární části
K =
    1.0000 + 2.0000i    0.0000 + 0.0000i    0.0000 + 2.0000i
    1.0000 + 0.0000i    0.0000 + 2.0000i    2.0000 + 2.0000i
    1.0000 + 1.0000i    2.0000 + 0.0000i    1.0000 + 2.0000i
>> absK = abs(K)   absolutní hodnoty prvků komplexní matice K
absK =
    2.2361         0    2.0000
    1.0000    2.0000    2.8284
    1.4142    2.0000    2.2361
>> [m, r] = min(K)   minima komplexní matice K a jejich řádkové indexy
m =
    1.0000 + 0.0000i    0.0000 + 0.0000i    0.0000 + 2.0000i
r =
    2    1    1
>> u = [1 3 0 3];
>> [m, ind] = max(u)   v případě více shodných maximálních prvků je vybrán pr-
vek s nižším indexem
m =
    3
ind =
    2
```

Ku vzestupnému seřazení vektoru se používá funkce `sort()`, u komplexních hodnot se řazení provádí podle absolutních hodnot. Do druhého nepovinného výstupního parametru je umístěna třídící permutace indexů. Při použití funkce `sort()` na matice dochází k řazení v rámci jednotlivých sloupců.

```
>> u = [1 3 0 3];
>> [y, ind] = sort(u)
y =
    0    1    3    3
ind =
    3    1    2    4
>> u(ind)   odpovídá seřazenému vektoru
ans =
    0    1    3    3
```

```
>> A = [1 0 1; 2 1 3; -1 2 1]
A =
     1     0     1
     2     1     3
    -1     2     1
>> [y, ind] = sort(A)
y =
    -1     0     1
     1     1     1
     2     2     3
ind =
     3     1     1
     1     2     3
     2     3     2
```

Pro sčítání a násobení prvků vektoru slouží funkce `sum()` a `prod()`. Aplikujeme-li tyto funkce na matici, příslušná operace je provedena po sloupcích.

```
>> f = prod(1:5)   faktoriál čísla 5
f =
    120
```

*Poznámka.* Obecně lze  $n!$ ,  $n \in \mathbb{N}_0$  spočítat příkazem `prod(1:n)` i pro  $n=0$ , protože součin přes prázdnou matici je roven 1.

```
>> f = prod(1:0)
f =
     1
```

Pro zjištění součtu, popř. součinu, všech prvků matice je třeba funkci `sum()`, popř. `prod()`, použít dvakrát.

```
>> soucet = sum(sum(A))   součet všech prvků matice A
soucet =
    10
```

*Poznámka.* Pro připomenutí, pro zjištění rozměrů matice lze použít funkci `size()` de dvojí formě – `s = size(A)` nebo `[r, sl] = size(A)`. Funkce `length()` vrací délku řádkového či sloupcového vektoru. Pokud ji aplikujeme na matici, dostáváme maximální z rozměrů; je ekvivalentní příkazu `max(size(A))`.

## Příklady k procvičení

1. Vygenerujte celočíselnou náhodnou matici  $A$  rozměrů  $5 \times 6$  s prvky z intervalu  $[-7, 7]$ .
  - a) Vytvořte matici  $A1$ , která bude obsahovat sudé sloupce matice  $A$ .
  - b) Zjistěte rozměry matice  $A1$ .
  - c) Z matice  $A1$  vytvořte matici  $A2$ , která bude obsahovat mít vyměněný druhý a třetí sloupec.
  - d) Dvěma způsoby vytvořte z matice  $A$  matici  $A3$ , která bude mít opačné pořadí řádků.
  - e) Spočítejte součet řádkových maxim matice  $A$ .
  - f) Vytvořte matici  $A4$ , která bude obsahovat pouze diagonální prvky matice  $A$ .
  - g) Vytvořte matici  $A5$ , která bude obsahovat stejné prvky jako matice  $A$  a pod diagonálou hodnoty 3.
  - h) Vytvořte čtvercovou matici  $A6$ , která bude obsahovat matici  $A$  a na posledním řádku její sloupcové součiny.
2. Vytvořte matici  $B$  rozměrů  $10 \times 10$ , která bude po řádcích obsahovat posloupnost čísel 0, 1, 2, atd.
3. Vygenerujte náhodné celé číslo  $a$  z intervalu  $[-3, 3]$  a náhodné přirozené číslo  $n$  z intervalu  $[1, 10]$ . Pak spočítejte  $\sum_{i=0}^n a^n$ .

*Řešení.*

1.  $A = \text{round}(14 * \text{rand}(5, 6) - 7)$ 
  - a)  $A1 = A(:, 2:2:6)$  nebo  $A1 = A(:, 2:2:\text{size}(A,2))$
  - b)  $[r, s1] = \text{size}(A1)$
  - c)  $A2 = A1(:, [1 3 2])$   
nebo pomocí permutační matice  $A2 = A1 * [1 0 0; 0 0 1; 0 1 0]$
  - d) 1. způsob:  $A3 = A(\text{size}(A,1):-1:1, :)$ ,  
2. způsob:  $A3 = \text{flipud}(A)$
  - e)  $\text{sum}(\text{max}(A'))$
  - f)  $A4 = \text{diag}(\text{diag}(A))$
  - g)  $A5 = A - \text{tril}(A, -1) + 3 * \text{ones}(\text{size}(A)) - \text{triu}(3 * \text{ones}(\text{size}(A)))$
  - h)  $A6 = [A; \text{prod}(A)]$
2.  $B = (\text{reshape}(0:99, 10, 10))'$
3.  $a = \text{round}(6 * \text{rand}(1)-3)$ ,  $n = \text{round}(9 * \text{rand}(1)+1)$   
 $\text{sum}(a.^{(0:n)})$

# Kapitola 5

## Logické operace

### Základní informace

Kapitola je zaměřena na relační a logické operátory a funkce, jejichž správné pochopení je důležité nejen pro samotnou práci v MATLABu, ale je rovněž důležitým předpokladem při programování.

### Výstupy z výuky

Studenti

- dokáží vyjmenovat a použít relační operátory
- dokáží uvést příklady logických operátorů a použít je na příkladech
- znají logické funkce pro testování nulových a nenulových, prázdných a reálných matic
- umí použít funkci `find()`

### 5.1 Relační operátory

Relační operátory slouží k porovnávání proměnných. Jedná se o následující operátory:

`==` test rovnosti  
`~=` test nerovnosti  
`<` menší  
`<=` menší nebo rovno  
`>=` větší nebo rovno  
`>` větší

Porovnávat je možné jen matice stejných rozměrů nebo matici se skalárem. Při porovnávání se vytvoří matice příslušných rozměrů obsahující jedničky a nuly podle toho, zda je příslušná relace splněna nebo není.

```
>> A = [1 2 3; 4 5 6];
>> B = [1 1 1; 8 7 6];
>> c = 5;
>> X = (A == B)
X =
     1     0     0
     0     0     1
>> Y = (A >= B)
Y =
     1     1     1
     0     0     1
>> Z = (A < c)
Z =
     1     1     1
     1     0     0
```

Při porovnávání komplexních proměnných se kromě testu na rovnost nebo nerovnost porovnává pouze reálná část.

```
>> u = [1+1i, -3i, 5];
>> d = 1+1i;
>> x = (u == d)
x =
     1     0     0
>> y = (u >= d)
y =
     1     0     1
```

## 5.2 Logické operátory

Mezi logické operátory patří:

- &** logické 'a současně'
- |** logické 'nebo'
- ~** logicky zápor
- xor** vylučovací 'nebo' (buď a nebo)

Jednotlivé operátory se aplikují na matice, případně na matici a skalár, podobně jako relační operátory. Výstupem logických operátorů je, stejně jako u relačních operátorů, matice obsahující nuly a jedničky podle toho, zda bylo porovnání nepravdivé či prav-

divé. Jednotlivé operace jsou prováděny po složkách. Nula je brána jako nepravdivá hodnota a nenulové hodnoty (včetně `Inf` a `NaN`) jsou brány jako pravdivé hodnoty.

```
>> A = -2:2;
>> B = zeros(1,5);
>> C = ones(1,5);
>> X1 = A|B
X1 =
     1     1     0     1     1
>> X2 = A|C
X2 =
     1     1     1     1     1
>> X3 = ~A
X3 =
     0     0     1     0     0
>> X4 = xor(A,pi)
X4 =
     0     0     1     0     0
```

Pro logické operátory rovněž existují ekvivalentní funkce:

logický operátor	ekvivalentní funkce
$A \& B$	<code>and(A,B)</code>
$A B$	<code>or(A,B)</code>
$\sim A$	<code>not(A)</code>

### 5.3 Logické funkce

Logické funkce jsou takové funkce, které na svém výstupu vrací matice či skaláry obsahující nuly a jedničky podle typu dané funkce:

`isinf(A)`

vrací matici stejného typu jako `A` s jedničkami na místech, kde je hodnota `Inf` nebo `-Inf`, na zbylých místech jsou nuly

`isnan(A)`

vrací matici stejného typu jako `A` s jedničkami na místech, kde je hodnota `NaN`, na zbylých místech jsou nuly

`isfinite(A)`

vrací matici stejného typu jako `A` s jedničkami na místech, kde nejsou hodnoty `Inf`, `-Inf` nebo `NaN`, na zbylých místech jsou nuly

**all(A)**

je-li A vektor, funkce vrací jedničku, pokud jsou všechny hodnoty nenulové, jinak vrací nulu. Je-li A matice, funkce `all()` je aplikována na jednotlivé sloupce matice A a výstupem je řádkový vektor obsahující nuly nebo jedničky pro jednotlivé sloupce.

**any(A)**

funguje podobně jako `all()`, vrací jedničku pokud je alespoň jeden prvek nenulový.

**isempty(A)**

vrací 1, pokud A je prázdná matice

**isreal(A)**

vrací 1, pokud A je reálná matice (ne komplexní)

**islogical(A)**

vrací 1, pokud A je logická matice. Charakter logické matice mají všechny výsledky relačních a logických operátorů a logických funkcí.

**isstr(A)**

vrací 1, pokud je A textová matice

**isnumeric(A)**

vrací 1, pokud je A číselná matice (ne textová)

```
>> A = [1, 0, Inf; NaN, -3, 2.5];
>> isfinite(A)
ans =
     1     1     0
     0     1     1
>> isempty(A)
ans =
     0
>> isstr(A)
ans =
     0
>> all(A)
ans =
     1
>> any(A)
ans =
     1     1     1
```

## 5.4 Funkce `find()` a `exist()`

Další užitečnou funkcí je funkce `find()`, která vrací indexy nenulových prvků vstupního vektoru nebo matice.

Možnosti použití funkce `find()`:

<code>I = find(x)</code>	do proměnné <code>I</code> umístí indexy nenulových prvků vektoru <code>x</code>
<code>I = find(A)</code>	do proměnné <code>I</code> umístí indexy nenulových prvků matice <code>A</code> , ty jsou brány po sloupcích
<code>[I,J] = find(A)</code>	umístí do <code>I</code> řádkové indexy a do " <code>J</code> " sloupcové indexy nenulových prvků matice " <code>A</code> "
<code>[I,J,K] = find(A)</code>	do proměnné <code>K</code> navíc umístí příslušné nenulové prvky
<code>[I,J,K] = find(x)</code>	do proměnné <code>I</code> umístí řádkové indexy, do proměnné <code>J</code> sloupcové indexy nenulových prvků vektoru <code>x</code> a do proměnné <code>K</code> jeho nenulové hodnoty

Častým použitím funkce `find()` jsou případy, kdy vektor `x` nebo matice `A` jsou logické vektory/matice, tzn. jsou výstupem porovnávacích operací.

```
>> x = -2:2;
>> [I,J,K] = find(x)
I =
     1     1     1     1
J =
     1     2     4     5
K =
    -2    -1     1     2
>> A = [-2 0; 1 -1];
>> [I,J] = find(A >= 0)   hledá nezáporné prvky matice A a jejich indexy
I =
     2
     1
J =
     1
     2
```

Existenci a typy objektů MATLABu lze testovat pomocí funkce `exist()`. Výstupem příkazu `ex = exist('A')` jsou následující možnosti proměnné `ex`:

- 0 objekt s názvem `A` neexistuje
- 1 `A` je proměnná
- 2 `A` je `m`-soubor v běžném adresáři nebo v `MATLABPATH`
- 3 `A` je `MEX`-soubor v běžném adresáři nebo v `MATLABPATH`
- 4 `A` je kompilovaná funkce v systému `SIMULINK`<sup>1</sup>

<sup>1</sup>`SIMULINK` je nadstavba MATLABu pro simulaci a modelování dynamických systémů.



- 5 A je interní (předkompilovaná) funkce MATLABu
- 6 A je předkompilovaná m-funkce s názvem A.p. Soubor A.p lze vytvořit pomocí příkazu `pcode` (více viz `help pcode`)
- 7 A je adresář

Při kolizi názvů (existují-li různé objekty se stejným názvem) platí následující priorita:

1. proměnná
2. interní funkce
3. funkce MEX (napřed pracovní adresář, pak MATLABPATH od začátku)
4. příkaz jako p-soubor (napřed pracovní adresář, pak MATLABPATH od začátku)
5. příkaz jako m-soubor (napřed pracovní adresář, pak MATLABPATH od začátku)

Funkci `exist()` je možné také použít se dvěma parametry:

```

exist('A', 'var')      testuje, zda je A proměnná
exist('A', 'builtin') testuje, zda je A interní funkce
exist('A', 'file')    testuje, zda je A soubor
exist('A', 'dir')     testuje, zda je A adresář
    
```

Výstupní hodnota je tak 1, pokud A jako daný typ existuje, v opačném případě je výstupní hodnotou 0.

## Příklady k procvičení

1. Vygeneruje náhodné celočíselné matice A a B s rozměry 5x5 a prvky v intervalu  $[-1, 2]$ .
  - a) Zjistěte počet nulových prvků matice A.
  - b) Zjistěte počet nenulových prvků matice A.
  - c) Zjistěte, na kolika pozicích se shodují prvky matice A a B.
  - d) Zjistěte, na kolika pozicích jsou prvky matice A větší než prvky matice B.
  - e) Zjistěte řádkové součty záporných prvků matice B.
  - f) Vytvořte matici C mající hodnotu -5 na pozicích, kde má B kladné prvky, na ostatních pozicích má matice C stejné prvky jako B.
  - g) Zjistěte řádkové počty kladných prvků matice A.
  - h) Zjistěte, kolik obsahuje matice A hodnot plus/minus nekonečno.
  - i) Zjistěte, zda se v řádcích matice A vyskytují nuly.
  - j) Vypište řádkové i sloupcové indexy nulových prvků matice B.

*Řešení.*

1. `A = round(3 * rand(5) - 1)`, `B = round(3 * rand(5) - 1)`
  - a) `sum(sum(A == 0))` nebo `sum(sum(~A))` nebo `length(find(A == 0))` nebo

`length(find(~A))`  
b) `sum(sum(A ~= 0))` nebo  
`sum(sum(~~A))` nebo  
`length(find(A ~= 0))` nebo  
`length(find(A))` nebo (méně efektivní)  
`prod(size(A) - sum(sum(~A)))`, apod.  
c) `sum(sum(A == B))` nebo  
`length(find(A == B))`  
d) `sum(sum(A > B))` nebo  
`length(find(A > B))`  
e) `sum((B .* (B < 0))')`  
f) `C = B; C(B > 0) = -5` nebo  
`C = -5 * (B > 0) + B .* (B <= 0)`  
g) `sum((A > 0)')`  
h) `sum(sum(isinf(A)))`  
i) `any(A')`  
j) `[r, s1] = find(B == 0)` nebo  
`[r, s1] = find(~~B)`

# Kapitola 6

## Textové řetězce

### Základní informace

Textový řetězec je posloupnost jednotlivých znaků uzavřených mezi jednoduchými apostrofy. V této kapitole se zaměříme na práci s textovými řetězci, na jejich vytváření, spojování, vyhledávání a nahrazování. Dále se zaměříme na převod numerických hodnot na textový řetězec a naopak a v samotném závěru kapitoly ukážeme využití textových řetězců při vyhodnocování matematických výrazů.

### Výstupy z výuky

Studenti

- umí vytvářet textové řetězce a skládat je do vektorů i matic
- dokáží převádět textový řetězec na číselný vektor, i naopak
- umí použít funkce pro převod čísla na řetězec, umí vytvářet a odstraňovat mezery z textových řetězců

### 6.1 Vytváření řetězců

Textový řetězec je posloupnost znaků ohraničená apostrofy (`'`). Pokud má řetězec obsahovat jako jeden ze znaků i apostrof, musíme jej zdvojit.

```
>> s1 = 'abcdef'
s1 =
    abcdef
>> s2 = '123''45'
s2 =
    123'45
```

Řetězce je možno skládat do matice, ovšem pouze za předpokladu, kdy jsou všechny řádky stejné délky. V opačném případě je nutné použít funkci `char()` nebo `str2mat()`, které řádky kratší délky doplní mezerami. Takto vytvořené matice mají příznak textových polí, to zjistíme pomocí příkazu `whos` nebo funkce `isstr()`.

```
>> S1 = ['1.radek'; '2.radek'; '3.radek'; '4.radek']
S1 =
    1.radek
    2.radek
    3.radek
    4.radek
>> S2 = char('1.radek', '2.radek', '3.radek', 'posledni radek')
S2 =
    1.radek
    2.radek
    3.radek
    posledni radek
>> S3 = str2mat('1.radek', '2.radek', '3.radek', 'posledni radek')
S3 =
    1.radek
    2.radek
    3.radek
    posledni radek
```

Pokud chceme složit více řetězců do jednoho řádku, použijeme k tomu stejný způsob, jakým definujeme číselné matice, tj. všechny prvky uzavřeme do hranatých závorek. Tento způsob má uplatnění zejména v případech, kdy některý z řetězců získáme jako výstup funkce.

```
>> s3 = ['abcdef', '123', '45']
s3 =
    abcdef12345
>> s4 = ['Číslo pí je rovno přibližně ', num2str(pi), '.']      funkce
num2str() slouží k převodu čísla na textový řetězec
s4 =
    Číslo pí je rovno přibližně 3.1416.
```

## 6.2 Základní manipulace s řetězci

Převod textového řetězce na číselný vektor můžeme provést pomocí funkce `double()`, přičemž jednotlivým znakům se přiřadí jejich kód (0 - 255) podle ASCII tabulky. Zpětný převod číselného vektoru na znakový řetězec lze provést použitím funkce `char()`.

```
>> x = double('ABCDabcd')
x =
    65    66    67    68    97    98    99   100
>> s = char(32:64)
s =
 !"#$%&'()*+,-./0123456789:;<=>?@
```

Znaky se dělí do několika skupin. Znaky s kódem menším než 32 mají speciální význam: konec řádku (kódy 10 a 13), konec stránky (kód 12), tabulátor (kód 8) apod. Znaky s kódy vyššími se běžně zobrazují na obrazovce, přičemž znaky s kódy většími než 127 se mohou lišit podle operačního systému nebo nastaveného jazyka.

V MATLABu existuje několik funkcí k testování typů objektů, jejich název zpravidla začíná slabikou `is*` a výstupem je hodnota 1, pokud objekt daného typu nabývá, v opačném případě je výstupem 0. Celý seznam těchto funkcí lze zobrazit příkazem `doc is*`, níže je uveden seznam nejpoužívanějších z nich.

<code>isletter(a)</code>	testuje, zda jsou jednotlivé složky proměnné <code>a</code> písmena ('A' - 'Z', 'a' - 'z')
<code>isspace(a)</code>	testuje, zda je proměnná <code>a</code> mezerového typu (mezera - kód 32, tabulátor apod.)
<code>isnumeric(a)</code>	testuje, zda je <code>a</code> číselná proměnná
<code>isstrprop(a, 'type')</code>	testuje, zda je proměnná <code>a</code> daného typu, argument <code>'type'</code> je textový řetězec specifikující daný typ: <code>'alpha'</code> (písmena), <code>'alphanum'</code> (číslíce a písmena), <code>'digit'</code> (číslíce), <code>'wspace'</code> (mezera), <code>'lower'</code> (malá písmena), <code>'upper'</code> (velká písmena), další viz <code>doc isstrprop</code>

Převod čísla na řetězec je možné pomocí funkce `num2str()` nebo `int2str()`, která číslo zaokrouhlí. Opačný převod provádí funkce `str2num()`, je možno použít i funkci `eval()`.

Řetězec dané délky obsahující pouze mezery lze vytvořit pomocí funkce `blanks(n)`, kde argument `n` označuje počet mezer. Funkce `deblank(a)` naopak odstraní mezery z konce textového řetězce `a`.

```
>> s1 = ['123', blanks(3), 'abc', blanks(2)]
s1 =
    123   abc
>> d1 = double(s1)   převod na číselný vektor odpovídající kódům ASCII tabulky
d1 =
    49   50   51   32   32   32   97   98   99   32   32
>> s2 = deblank(s1)  odstranění mezer
s2 =
    123   abc
>> d2 = double(s2)
d1 =
    49   50   51   32   32   32   97   98   99
```

### 6.3 Funkce pro manipulaci s řetězci

S řetězci je možné provádět řadu operací pomocí funkcí k tomu určených. Nejdůležitější z nich jsou:

`strcat(s1, s2, ..., sN)` – spojuje řetězce, které jsou na vstupu, do jednoho.

`strvcats(s1, s2, ..., sN)` – umísťuje řetězce do matice jako řádky, přitom je doplňuje mezerami na stejnou délku.

`strcmp(s1, s2)` – porovnává vstupní řetězce `s1` a `s2`. Vrací hodnotu 1, pokud jsou řetězce stejné, v opačném případě vrací 0.

`strncmp(s1, s2, n)` – funguje podobně jako funkce `strcmp()` s tím rozdílem, že porovnává pouze prvních `n` prvků vstupních řetězců.

`strcmpi(s1, s2)` – funguje podobně jako funkce `strcmp()`, ale nerozlišuje malá a velká písmena.

`strncmpi(s1, s2, n)` – funguje podobně jako funkce `strcmpi()`, porovnává pouze prvních `n` prvků vstupních řetězců.

`findstr(s1, s2)` – vyhledává v delším řetězci kratší z nich. Jako výstup vrací indexové pozice, na kterých začíná kratší řetězec v delším. Pokud se v něm nevyskytuje, výstupem je prázdná matice.

`strfind(s1, s2)` – funguje podobně jako funkce `findstr()`, vyhledává řetězec `s2` v řetězci `s1`.

`strrep(s1, s2, s3)` – funkce prohledává řetězec `s1`, pokud v něm najde řetězec `s2`, nahradí jej řetězcem `s3`.

`[s1, s2] = strtok(s)` – najde úvodní část vstupního řetězce ukončenou mezerou a vrátí ji na výstup do proměnné `s1`. Případné počáteční mezery jsou vypuštěny. Ve druhé výstupní proměnné `s2` je obsažen zbytek vstupního řetězce.

`upper(s)` – převede malá písmena ve vstupním řetězci `s` na velká.

`lower(s)` – převede velká písmena ve vstupním řetězci `s` na malá.

## 6.4 Funkce `eval` a `feval`

Funkce `eval()` slouží k vyhodnocení vstupního řetězce. Funkce je užitečná zejména v případě, kdy potřebujeme spočítat hodnotu nějakého výrazu pro různé hodnoty parametrů v něm obsažené.

```
>> s = 'sin(2*pi*a*x)';
>> x = 0:0.01:1;
>> a = 1;
>> y = eval(s);
y =
    0    0.5878    0.9511    0.9511    0.5878    0.0000   -0.5878   -0.9511
-0.9511   -0.5878   -0.0000
```

Funkce `feval(funkce, hodnota)` slouží k vyhodnocení funkce. Jejím prvním vstupním argumentem je textový řetězec obsahující název funkce, která má být vyhodnocena (`funkce`), druhý parametr obsahuje hodnotu (`hodnota`), která se předá volané funkci jako vstupní parametr.

Následující tři příkazy dávají stejný výstup:

```
>> x = 0:0.01:1;
>> y = sin(x);
>> y = feval('sin', x);
>> y = eval('sin(x)');
```

Pokud má volaná funkce více parametrů, jsou uvedeny jako další parametry funkce `feval()`.

```
>> x = -1:2;
>> y = feval('diag', x, 1);
y =
     0     -1     0     0     0
     0     0     0     0     0
     0     0     0     1     0
     0     0     0     0     2
     0     0     0     0     0
```

Funkce `feval()` se využívá zejména v případě, kdy voláme různé funkce, jejichž názvy jsou obsaženy v textovém řetězci.

## Příklady k procvičení

- Definujte následující tři textové řetězce: `t1 = 'Textovy retezec'`, `t2 = 'je posloupnost znaku'` a `t3 = 'uzavrena v apostrofech.'`.
  - Vytvořte řádkový vektor `t` poskládaný z řetězců `t1`, `t2` a `t3`.
  - Vytvořte matici `T` obsahující řetězce `t1`, `t2` a `t3` v řádcích.
  - Zjistěte rozměry matice `T`.
  - Zjistěte počet mezer ve druhém řádku matice `T` s koncovými mezerami i bez nich a porovnejte s počtem mezer v řetězci `t2`.
  - Vypište pozice, na kterých se v textovém řetězci `t1` nachází řetězec `'e'`.
  - Zjistěte, na kolika pozicích vektoru `t` se nachází řetězec `'o'`.
  - Zjistěte, v kolika sloupcích matice `T` se nachází mezera.
  - Každý výskyt řetězce `'z'` ve vektoru `t` nahraďte otazníkem.
- Definujte funkci  $e^{a \cdot x}$  a pro `a=0.5` a ekvidistantní vektor `x` délky 5 s počátečním bodem 0 a koncovým bodem 3 vyčíslete.

*Řešení.*

- `t1 = 'Textovy retezec'`, `t2 = 'je posloupnost znaku'`, `t3 = 'uzavrena v apostrofech.'`
  - `t = [t1, ' ', t2, ' ', t3]` nebo  
`t = [t1, blanks(1), t2, blanks(1), t3]` nebo
  - `T = char(t1, t2, t3)` nebo  
`T = strvcats(t1, t2, t3)`
  - `size(T)`



d) s koncovými mezerami: `sum(isspace(T(2,:)))` nebo  
`sum(isstrprop(T(2,:), 'wspace'))`  
bez koncových mezer: `sum(isspace(deblank(T(2,:))))` nebo  
`sum(isstrprop(deblank(T(2,:)), 'wspace'))`  
v řetězci `t2`: `sum(isspace(t2))` nebo  
`sum(isstrprop(t2, 'wspace'))`

e) `findstr('e', t1)` nebo  
`findstr(t1, 'e')` nebo  
`strfind(t1, 'e')`

f) `length(findstr('o', t))` nebo  
`length(findstr(t, 'o'))` nebo  
`length(strfind(t, 'o'))`

g) `sum(sum(isspace(T)) >= 1)` nebo  
`sum(sum(isstrprop(T, 'wspace')) >= 1)` nebo  
`sum(sum(double(T) == 32) >= 1)`

h) `strrep(t, 'z', '?')`

2. `f = 'exp(a*x)'`  
`a = 0.5, x = linspace(0, 3, 5)`  
`y = eval(f)`

# Kapitola 7

## Vyhodnocování výrazů

### Základní informace

Matematické výrazy a funkce mohou být v MATLABu zadány několika způsoby. V této kapitole si ukážeme možnosti zadání těchto funkcí, způsoby výpočtu funkčních hodnot či hledání jejich kořenů. Poslední část kapitoly je věnována polynomům, pro jejichž manipulaci byly v MATLABu vyvinuty speciální funkce.

### Výstupy z výuky

Studenti

- umí vyhodnotit výraz jako textový řetězec, znají rozdíl mezi vyhodnocováním proměnných jako skalárů, vektorů a matic
- umí pracovat se symbolickými výrazy, umí převádět textový řetězec na symbolický výraz a vyhodnocovat jej
- znají funkce pro derivování, integrování a zjednodušení symbolického výrazu
- definují výraz jako INLINE funkci, dokáží takto definované výrazy vyhodnocovat
- umí konstruovat polynomy, dokáží použít funkce pro určení kořenů a vyhodnocení polynomu v bodě i matici bodů

### 7.1 Výraz jako textový řetězec

Chceme-li vyhodnotit výraz zadaný jako textový řetězec, použijeme funkci `eval()` blíže popsanou v minulé kapitole.

## KAPITOLA 7. VYHODNOCOVÁNÍ VÝRAZŮ

---

Vyhodnocení výrazu  $x^2 + xy + 1$  v bodech  $x = 2$ ,  $y = 3$  pomocí textového řetězce:

```
>> f = 'x^2+x*y+1';
>> x = 2;
>> y = 3;
>> z = eval(f)
z =
    11
```

Trochu jiná situace však nastane, pokud proměnné  $x$  a  $y$  budou vektory, např.  $x = [0, 1]$ ,  $y = [1, 2]$ , a tedy  $z = [1, 4]$ . Je třeba si totiž uvědomit, že MATLAB bude výše definovaný výraz  $f$  vyhodnocovat maticově, tj. operace umocňování a násobení bude provádět maticově. Vyhodnocení v tomto případě selže, protože vektory  $x$  a  $y$  nemají příslušné rozměry pro tyto operace. Navíc nás ani maticové vyhodnocení nezajímá, neboť chceme, aby se jednotlivé operace provedly po složkách. Musíme tedy "dodat" tečky před operace násobení a umocňování (případně ještě dělení). To se dá provést buď ručně nebo pomocí funkce `vectorize()`.

```
>> f = vectorize('x^2+x*y+1')
f =
    x.^2+x.*y+1
>> x = [0 1];
>> y = [1 2];
>> z = eval(f)
z =
    1    4
```

Takový postup platí samozřejmě i v případě, že by  $x$  a  $y$  nebyly vektory, ale matice (samozřejmě stejných rozměrů).

```
>> x = [0 1 2; -1 2 3];
>> y = [1 2 -1; 0 3 1];
>> z = eval(f)
z =
    1    4    3
    2   11   13
```

Pro řešení rovnic slouží funkce `solve()`.

```
>> g = 'x^2 - 2*x - 3';
>> solve(g)
ans =
     3
    -1
```

## 7.2 Symbolický výraz

MATLAB umí pracovat i se symbolickými výrazy (podobně jako např. MAPLE). Je však nutné, aby příslušná verze MATLABu obsahovala knihovnu pro práci se symbolickými výrazy (`Symbolic Toolbox`). To lze zjistit např. příkazem `ver`, který vypíše verzi MATLABu a všechny nainstalované knihovny. Pomocí funkce `sym()` můžeme libovolný textový řetězec převést na symbolický výraz. Vyhodnocení takto definovaného výrazu se provede pomocí funkce `subs(f, old, new)`, kde `f` je symbolický výraz nebo textový řetězec, `old` udává proměnné jako textové řetězce. Pokud je proměnných více, musí být odděleny čárkou, uzavřeny ve složených závorkách a záleží na jejich pořadí. Argument `new` udává hodnoty proměnných, ve kterých má být funkce vyčíslena. Jejich pořadí se řídí pořadím proměnných uvedeným v argumentu `old`.

Vyhodnocení výrazu  $x^2 + xy + 1$  v bodech  $x = 2$ ,  $y = 3$  pomocí symbolického výrazu:

```
>> f = sym('x^2+x*y+1');
>> z = subs(f, 'x', 'y', 2, 3)
z =
     11
>> z1 = subs('sin(x)', 'x', pi/2)   vyhodnocení funkce sin(x) v bodě x = pi/2
z1 =
     1
```

V případě, že proměnné  $x$  a  $y$  jsou vektory, resp. matice, syntaxe je podobná. POZOR! Symbolické výrazy nevektorizujeme!

```
>> f = sym('x^2 + x*y + 1');
>> z = subs(f, 'x', 'y', [0 1], [1 2])
z =
     [1,    4]
```

Pro řešení rovnic slouží funkce `solve()`.

```
>> g = sym('x^2 - 2*x - 3');
>> solve(g)
ans =
     3
    -1
```

Práce se symbolickými výrazy má mnoho výhod. K dispozici jsou funkce pro derivování (`diff()`), integrování (`int()`), zjednodušení výrazů (`simplify()`), převod výrazu do jeho  $\text{\LaTeX}$ ovské reprezentace (`latex()`) a mnoho dalších.

### 7.3 Výraz jako INLINE funkce

Poslední možností, jak vyhodnotit daný výraz, je definovat ho jako tzv. INLINE funkci příkazem `inline()`. Samotné vyhodnocení je prováděno pouhým zapsáním dané hodnoty do kulatých závorek za funkci.

```
>> f = inline('x^2 + 1')
>> z = f(2)
z =
    5
```

V případě, že funkce obsahuje více proměnných, je jejich pořadí dáno abecedně. Vlastní pořadí proměnných lze definovat pomocí dalších argumentů – čárkami oddělené textový řetězec názvů proměnných v požadovaném pořadí.

```
>> f1 = inline('x^2 + x*y + 1', 'x', 'y');   příkaz ekvivalentní příkazu s im-
plicitním nastavení pořadí proměnných f = inline('x^2 + x*y + 1');
>> z1 = f1(2, 3)
z1 =
    11
>> f2 = inline('x^2 + x*y + 1', 'y', 'x');   výměna pořadí proměnných pro
vyhodnocování funkce
>> z2 = f2(2, 3)
z2 =
    16
```

V případě, že proměnné  $x$  a  $y$  jsou vektory, resp. matice, syntaxe je podobná – aby byly operace prováděny po složkách, je potřeba dodat tečky před příslušné operace nebo funkci vektorizovat pomocí funkce `vectorize()`.

```
>> f = inline('x^2 + x*y + 1', 'x', 'y');
>> f = vectorize(f);
>> z = f([0 1], [1 2])
z =
    1    4
```

### 7.4 Polynomy

V MATLABu existuje několik funkcí usnadňujících práci s polynomy. Jsou to především funkce pro tvorbu polynomů, jejich vyhodnocování, výpočet kořenů, apod. S polynomy

## KAPITOLA 7. VYHODNOCOVÁNÍ VÝRAZŮ

---

lze samozřejmě zacházet stejně jako s výrazy popsány výše, tento přístup ovšem není tak efektivní.

Polynom je v MATLABu definován jako posloupnost koeficientů seřazených od členu s nejvyšší mocninou po absolutní člen. Nulové hodnoty v této posloupnosti odpovídají chybějícím členům polynomu.

```
>> p = [2 -3 0 5 -4];   zápis polynomu  $p(x) = 2x^4 - 3x^3 + 5x - 4$ 
```

Základní funkce pro práci s polynomy:

<code>y = polyval(p, x)</code>	vyhodnocení polynomu <code>p</code> v daném bodě/vektoru <code>x</code>
<code>y = polyvalm(p, A)</code>	vyhodnocení polynomu <code>p</code> v matici bodů <code>A</code>
<code>k = roots(p)</code>	kořeny <code>k</code> polynomu <code>p</code>
<code>p = poly(k)</code>	sestrojení polynomu <code>p</code> , jehož kořeny jsou <code>k</code>
<code>pd = polyder(p)</code>	derivace <code>pd</code> polynomu <code>p</code>
<code>pint = polyint(p, k)</code>	integrál <code>pint</code> polynomu <code>p</code> . <code>k</code> označuje integrační konstantu, není-li uvedena, je volena nulová integrační konstanta
<code>p = conv(p1, p2)</code>	součin <code>p</code> polynomů <code>p1</code> a <code>p2</code>
<code>[podil, zbytek] = deconv(p1, p2)</code>	dělení polynomů <code>p1</code> a <code>p2</code> se zbytkem, <code>podil</code> obsahuje podíl polynomů, <code>zbytek</code> obsahuje jejich zbytek

```
>> y1 = polyval(p, 1)   vyhodnocení polynomu p v bodě 1
y1 =
     0
>> y2 = polyval(p, -1:1)   vyhodnocení polynomu p v bodech -1, 0, 1
y2 =
    -4    -4     0
>> A = [-1 0; 1 2];
>> Y1 = polyval(p, A)   vyhodnocení polynomu p po složkách v matici A
Y1 =
    -4    -4
         0    14
>> Y1 = polyvalm(p, A)   vyhodnocení polynomu p (maticově) v matici A, ekviva-
lentní příkaz: 2*A^4 - 3*A^3 + 5*A - 4*eye(2)
Y1 =
    -4     0
         6    14
>> k = roots([1 0 -1])   kořeny polynomu  $x^2 - 1$ 
k =
    -1
     1
```

```

>> p1 = poly([-1 1])    polynom p s kořeny -1, 1
p1 =
    1  0 -1
>> pd = polyder(p)     derivace polynomu p
pd =
    8  -9  0  5
>> p = conv([1 -1], [2 0])    součin  $p(x) = 2x^2 - 2x$  polynomů  $p_1(x) = x - 1$  a
 $p_2(x) = 2x$ 
p =
    2  -2  0
>> [podil, zbytek] = deconv(p, p1)    podíl a zbytek při dělení polynomu p po-
lynomem p1
podil =
    2  -3  2
zbytek =
    0  0  0  2  -2

```

## Příklady k procvičení

- Definujte funkci  $f(x) = x^4 + 4x^3 + 3x^2 - 4x - 4$  jako:
  - textový řetězec **f1**,
  - symbolický výraz **f2**,
  - inline funkci **f3**,
  - polynom **f4**.
- Spočítejte funkční hodnoty funkcí **f1**, **f2**, **f3** a **f4** v bodech  $x = [0 \ 1 \ 2 \ 3]$ .
- Pro **f1**, **f2** a **f4** řešte rovnici  $f(x) = 0$ .
- Vyčíslete první derivaci funkce  $f(x)$  v bodech  $x$  pomocí symbolického výrazu a polynomu.
- Vyčíslete integrál z funkce  $f(x)$  v bodech  $x$  pomocí symbolického výrazu a polynomu.

*Řešení.*

- f1** = 'x^4 + 4\*x^3 + 3\*x^2 - 4\*x - 4'
  - f2** = sym('x^4 + 4\*x^3 + 3\*x^2 - 4\*x - 4')
  - f3** = inline('x^4 + 4\*x^3 + 3\*x^2 - 4\*x - 4')
  - f4** = [1 4 3 -4 -4]

2. `x = 0:3`
  - a) `y1 = eval(vectorize(f1))`
  - b) `y2 = subs(f2, 'x', x)`
  - c) `f3 = vectorize(f3), y3 = f3(x)`
  - d) `y4 = polyval(f4, x)`
  
3.
  - a) `res1 = solve(f1)`
  - b) `res2 = solve(f2)`
  - c) `res4 = roots(f4)`
  
4.
  - a) `yd2 = eval(vectorize(diff(f2)))`
  - b) `yd4 = polyval(polyder(f4), x)`
  
5.
  - a) `yint2 = eval(vectorize(int(f2)))`
  - b) `yint4 = polyval(polyint(f4), x)`



# Kapitola 8

## Práce se soubory

V praktických situacích často pracujeme s daty, která jsou uložena v externích souborech, v průběhu práce potřebujeme ukládat záznam práce, některé proměnné obsahující důležité výstupy či celý workspace. V této kapitole se seznámíme s příkazy pro manipulaci se soubory a adresáři a nastavení cesty k nim.

### Základní informace

### Výstupy z výuky

Studenti

- umí prováděné příkazy i s výsledky ukládat do souboru
- znají funkce pro ukládání a načítání proměnných
- dokáží zjistit název pracovního adresáře a vypsát jeho obsah, dokáží se pohybovat mezi adresáři

### 8.1 Záznam práce

Prováděné příkazy i s výstupy je možné zaznamenávat do tzv. žurnálu, ukládat lze pomocí příkazu `diary`. Příkaz `diary on` zapne ukládání a všechny další příkazy i s jejich výstupy budou ukládány v pracovním adresáři do souboru s názvem `diary`. Příkaz `diary off` toto ukládání přeruší a uzavře soubor, přičemž nové použití příkazu smaže původní obsah souboru a nově provedené příkazy i s výstupy do souboru přidá. Použití samotného příkazu `diary` bez parametrů přepne režim ukládání, tj. pokud bylo ukládání zapnuto, pak ho vypne a naopak.

Ukládání do souboru s jiným názvem dosáhneme pomocí příkazu `diary jmeno_souboru` nebo ekvivalentním příkazem `diary('nazev_souboru')`, kde `nazev_souboru` je libovolný název souboru. Další použití příkazu `diary` bez parametrů se pak bude vztahovat ke zvolenému souboru.

## 8.2 Ukládání a načítání proměnných

Pro uložení proměnných do souboru slouží příkaz `save`. Jeho použití bez parametrů uloží všechny definované proměnné do souboru s názvem `matlab.mat`.

Pro uložení pouze vybraných proměnných do námi zvoleného souboru zadáme jako první parametr název souboru a jako další parametry (odděleny čárkami nebo mezerami) názvy proměnných, které chceme uložit. Uložené soubory mají automaticky příponu `.mat`. Je možné zadat i příponu jinou.

```
>> who seznam definovaných proměnných
      Your variables are:
      a  A  b  c  u  v  x
>> save data uloží všechny definované proměnné do souboru data.dat
>> save data1 a b c proměnné a, b a c uloží do souboru data1.mat
>> save('data1', 'a', 'b', 'c') ekvivalentní příkaz k příkazu save data1 a
b c
```

Načíst data z uloženého souboru můžeme příkazem `load`. Použijeme-li ho bez parametrů, načtou se všechny uložené proměnné ze souboru `matlab.mat`. Příkaz `load` je možné použít podobně jako příkaz `save`.

Pokud má soubor s uloženými daty jinou příponu než `.mat`, musí se při načítání jeho obsahu použít parametr `"-MAT"` (lze použít i malá písmena: `-mat`).

```
>> load data1 načte všechny proměnné uložené v souboru data1.mat, ekvivalentní
příkaz: load('data1')
>> load data1.dat -MAT načte všechny proměnné uložené v souboru data1.dat
```

Data lze uložit do souboru i v tzv. ASCII tvaru, který je běžně čitelný v textovém editoru. Dosáhneme toho užitím volby `-ASCII`:

```
save data2.dat X -ASCII
```

V tomto souboru ovšem není uložen název proměnné `X`, pouze její obsah. Při načítání souboru, který má jinou koncovku než `.mat`, se automaticky předpokládá, že jsou v ASCII tvaru. Z takového souboru je možné ovšem načíst pouze jednu proměnnou, která má navíc stejný název jako je název původního souboru (bez přípony). Takové soubory je možné vytvářet i ručně, případně jako výstup práce jiných programů. Je ovšem nutno mít na paměti, že data v nich obsažená musí mít tvar matice, tj. každý řádek musí mít stejný počet sloupců.

## 8.3 Soubory v systému MATLAB

MATLAB většinu příkazů, které provádí, hledá v souborech, které tyto příkazy obsahují jako funkce. Přípona těchto souborů je `.m`, proto se taky nazývají m-soubory (m-file). Tyto soubory obsahují jednak zápis algoritmu, pomocí něhož se provádí daný výpočet či dané operace, a nápovědu, která se vypisuje příkazem `help`.

Některé funkce jsou tzv. vnitřní, ty jsou uloženy v předkompilované podobě v knihovně funkcí a příslušný m-soubor obsahuje jen nápovědu. Pomocí příkazu `which` zjistíme, kde se soubor s daným programem či funkcí nachází, nebo zda se jedná o vnitřní funkci MATLABu.

```
>> which poly
C:\Program Files\MATLAB\R2014a\toolbox\matlab\polyfun \poly.m
>> which eig
built-in (C:\Program Files\MATLAB\R2014a\toolbox\matlab\matfun\@single\
\eig) % single method
```

Další funkce mohou být uloženy v tzv. mex-souborech (s příponou `.mex`). Ty jsou vytvořeny v některém jiném programovacím jazyce (C, FORTRAN) a jsou speciálně zpracovány tak, aby je bylo možné používat v MATLABu.

Datové soubory, jak už bylo řečeno, mají standardně příponu `.mat`.

## 8.4 Cesta k souborům

Programy a funkce spouštěné při práci v MATLABu jsou vyhledávány v adresářích, k nimž je nastavená cesta – tzv. `matlabpath`. Její obsah zjistíme příkazem `path` nebo `matlabpath`. Pokud chceme do této cesty přidat nějaký další adresář, můžeme to provést pomocí funkce `path(path, 'dalsi_adresar')` nebo příkazem `addpath dalsi_adresar`. Proměnná `dalsi_adresar` musí obsahovat textový řetězec s cestou k danému adresáři. Konkrétní tvar cesty závisí na použitém operačním systému.

```
>> path(path, 'd:\user\matlab')   přidání nové cesty v MS Windows
>> addpath d:\user\matlab         ekvivalentní způsob pro přidání nové cesty v MS Win-
dows
>> addpath /home/user/matlab     přidání nové cesty v UNIXu
```

Pokud na konci příkazu `addpath` přidáme přepínač `-BEGIN`, nový adresář se uloží na začátek seznamu a bude pak prohledáván jako první. Uložení na konec seznamu je možné zadat přepínačem `-END`. Ke smazání adresáře ze seznamu slouží příkaz `rmpath`.

## 8.5 Další příkazy pro práci se soubory

Výpis obsahu pracovního adresáře zjistíme příkazem `dir` nebo `ls`. Můžeme také použít hvězdičkovou konvenci – příkazem `dir *.mat` vypíšeme všechny datové soubory s příponou `.mat` v daném adresáři.

Chceme-li zjistit, ve kterém adresáři se právě nacházíme, použijeme příkaz `pwd`, který zobrazí textový řetězec kompletní cesty do aktuální složky. Pro změnu pracovního adresáře slouží příkaz `cd jiny_adresar`, kde `jiny_adresar` je cesta k novému adresáři.

```
>> addpath(pwd)   aktuální adresář je možné přidat do cesty pro prohledávání
>> cd matlab/data  změna pracovního adresáře
```

Pro výpis obsahu zvoleného souboru můžeme použít funkci `type(filename)`, kde `filename` je název zvoleného souboru. Pokud předem zadáme příkaz `more on`, výpis dlouhého souboru bude zobrazován po stránkách. Vypnutí stránkování provedeme příkazem `more off` (implicitní nastavení).

Další užitečné funkce:

- `copyfile('source', 'destination')` funkce pro kopírování souborů z původního umístění `'source'` na nové místo `'destination'`
- `delete('fileName')` funkce pro mazání souborů
- `rmdir('folderName')` funkce pro mazání adresářů
- `lookfor topic` pro zadaný výraz `topic` prohledává nápovědu
- `edit` příkaz pro vyvolání editoru pro psaní kódu, jeho nastavení závisí na operačním systému. Editor lze rovněž vyvolat z menu *Editor* → *New*.

## Příklady k procvičení

1. Veškerý záznam práce uložte do souboru `ZaznamPrace`.
2. Načtete soubor `v.mat`, ve kterém je uložen vektor `v`. Zjistěte jeho délku a počet záporných prvků.
3. Vektor `v` seskládejte po sloupcích do čtvercové matice `A` a tu uložte do souboru `A.mat`.
4. Ukončete ukládání záznamu práce do souboru.

*Řešení.*

1. `diary ZaznamPrace` nebo `diary('ZaznamPrace')`

2. `load v, delka = length(v), zap = sum(v < 0)`
3. `A = reshape(v, sqrt(delka), sqrt(delka)), save A.mat A`
4. `diary off`

# Kapitola 9

## Práce s grafikou

### Základní informace

Nedílnou součástí vědeckotechnických výpočtů, analýz, modelů či simulací je prezentace dat či grafické výstupy. Následující kapitola obsahuje základní typy grafů, seznamuje s potřebnými funkcemi pro potřebnou úpravu grafů a dává návod, jak lze požadovaného vzhledu grafu dosáhnout interaktivně.

### Výstupy z výuky

Studenti

- ovládají základní příkazy pro vykreslení grafu, dokáží určit definiční obor a spočítat funkční hodnoty
- dokáží použít parametry ovlivňující barvu, styl čáry a znak pro vykreslení jednotlivých bodů
- umí konstruovat více grafů do jednoho grafického okna, znají funkce pro zapínání a vypínání přepisování v grafu
- dokáží zobrazit mřížku v grafu, znají příkazy pro popisky os i název grafu
- umí vykreslit více grafů do jednoho grafického okna
- zobrazí funkce dvou proměnných, umí zvolit barevné škálování a měnit úhel pohledu na trojrozměrný obrázek
- umí měnit vlastnosti grafických objektů

## 9.1 Funkce `plot()` a její použití

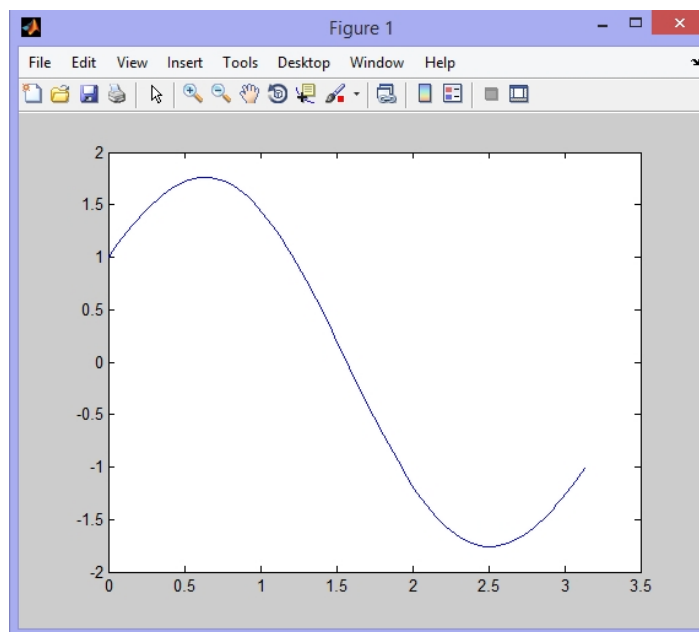
Základním příkazem pro kreslení grafů funkcí jedné proměnné je funkce `plot()`, která při implicitním nastavení spojuje zadané body úsečkami.

`plot(x)` pro vstupní vektor  $x$  vykreslí spojnicový graf s indexy  $1:\text{length}(x)$  na ose  $x$  a hodnotami vektoru  $x$  na ose  $y$ . Pro vstupní matici  $x$  vykreslí v jednom okně spojnicový graf pro každý sloupec matice.

`plot(x, y)` vykreslí spojnicový graf hodnot  $y$  na pozicích  $x$ .

Graf funkce  $\sin(2x) + \cos(x)$  na intervalu  $[0, \pi]$ :

```
>> x = linspace(0, pi);    vygenerování (implicitně 100) hodnot na ose x, pro které  
    má být funkce zobrazena  
>> y = sin(2*x) + cos(x);  výpočet funkčních hodnot  
>> plot(x, y)             vykreslení grafu (Obr. 9.1)
```



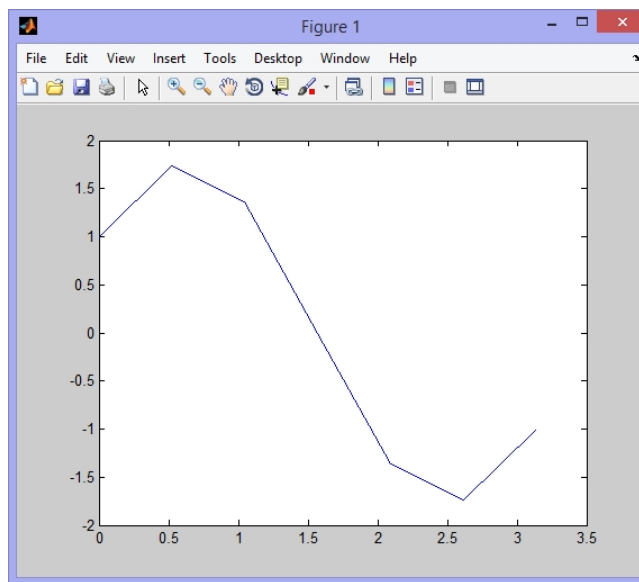
Obr. 9.1. Výstup funkce `plot(x, y)`.

*Poznámka.* Vykreslená funkce vypadá hladce, ve skutečnosti se ovšem jedná o lomenou čáru, jednotlivé body jsou spojovány úsečkami. Vzhled grafu tak velmi ovlivňuje hustota bodů na ose  $x$  (Obr. 9.2).

## KAPITOLA 9. PRÁCE S GRAFIKOU

---

```
>> x = linspace(0, pi, 7);    vygenerování menšího počtu hodnot na ose x  
>> y = sin(2*x) + cos(x);  
>> plot(x, y)    vykreslení grafu (Obr. 9.2)
```



Obr. 9.2. Výstup funkce `plot(x, y)` pro menší počet vygenerovaných bodů.

Funkce `plot()`, může obsahovat ještě další volitelný parametr. Jedná se o textový řetězec, pomocí něhož můžeme ovlivňovat barvu a styl vykreslené čáry a symbol pro zobrazení jednotlivých bodů, které jsou spojovány úsečkami. Každá barva, styl čáry a symbol mají svůj znak, jejich kombinací do textového řetězce zvolíme vzhled grafu.

Znaky pro barvu:

- y žlutá (yellow)
- m fialová (magenta)
- c modrozelená (cyan)
- r červená red
- g zelená (green)
- b modrá (blue)
- w bílá (white)
- k černá (black)

Znaky pro styl čáry:

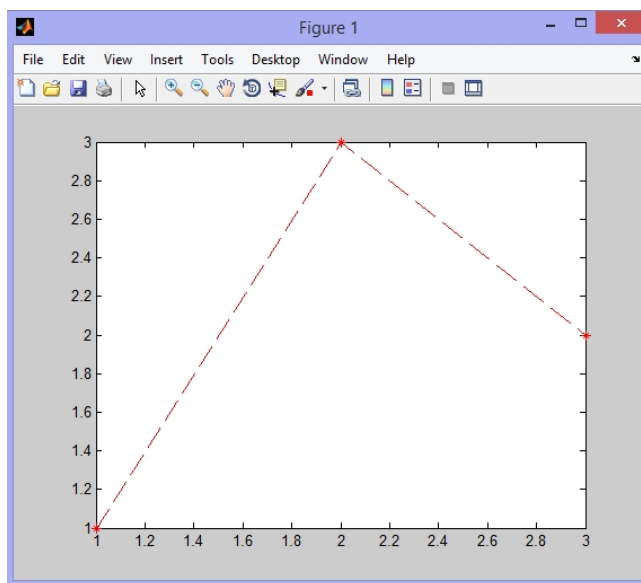
- plnou čarou
- čárkovaně
- : tečkovaně
- . čerchovaně



Znaky pro symboly zobrazených bodů:

- . tečka
- o kroužek
- + křížek
- \* hvězdička
- s čtvereček (square)
- d kosočtverec (diamond)
- v trojúhelník (otočený dolů)
- ^ trojúhelník (otočený nahoru)
- < trojúhelník (otočený doleva)
- > trojúhelník (otočený doprava)
- p pentagram
- h hexagram

```
>> x = [1, 2, 3];  
>> y = [1, 3, 2];  
>> plot(x, y, 'r--*')   vykreslení daných bodů červenou čárkovanou čarou s body  
vyznačenými hvězdičkami (Obr. 9.3)  
>> plot(x, y)           vykreslení grafu (Obr. 9.3)
```



Obr. 9.3. Výstup funkce `plot(x, y, 'b--k*')`.

## KAPITOLA 9. PRÁCE S GRAFIKOU

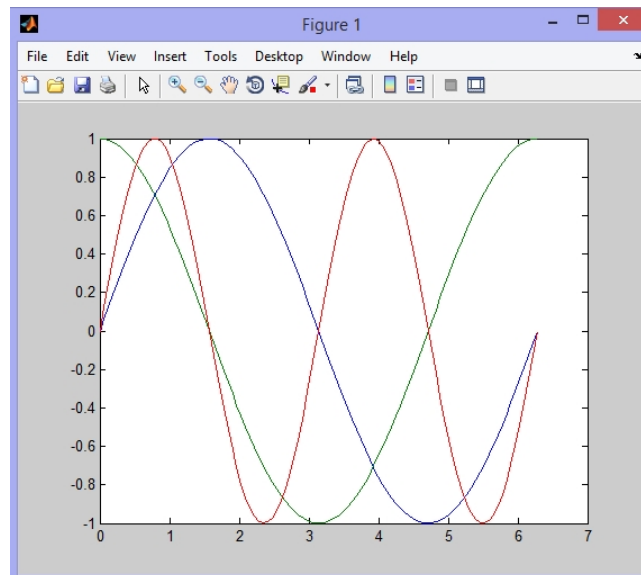
---

```
>> plot(x, y, '+r')
```

 vykreslí pouze zadané body bez spojování úsečkou. V případě, že explicitně nezadáme barvu, je volena automaticky.

Funkce `plot()` umožňuje kreslit více grafů najednou, stačí je zadat jako další její parametry. Pro každý graf můžeme uvést parametry pro styl vykreslení.

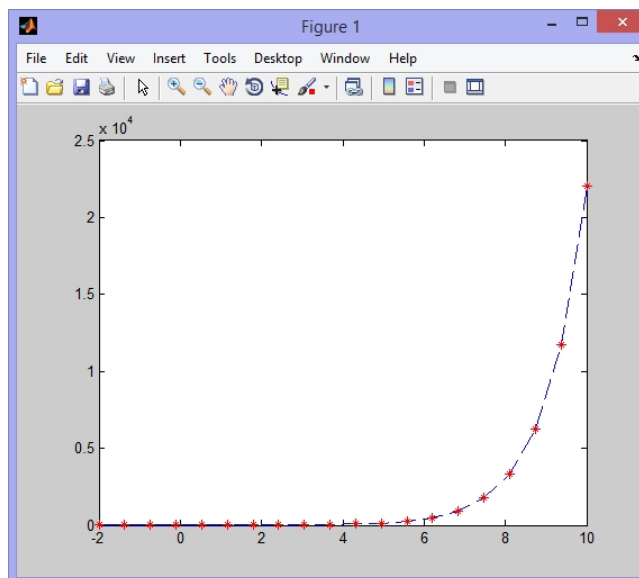
```
>> x = linspace(0, 2*pi);  
>> y1 = sin(x);  
>> y2 = cos(x);  
>> y3 = sin(2*x);  
>> plot(x, y1, x, y2, x, y3)
```

 vykreslí tři grafy uložené v proměnných `y1`, `y2` a `y3`, všechny pro stejný definiční obor daný proměnnou `x` (Obr. 9.4)

Obr. 9.4. Výstup funkce `plot(x, y1, x, y2, x, y3)`.

```
>> x = linspace(-2, 10, 20);  
>> y = exp(x);  
>> plot(x, y, 'b--', x, y, 'r*')
```

 vykreslení bodů jinou barvou než je barva spojovacích čar (Obr. 9.5)

Obr. 9.5. Výstup funkce `plot(x, y, 'b--', x, y, 'r*')`.

## 9.2 Vzhled grafu

MATLAB obsahuje několik funkcí, díky nimž si uživatel může přizpůsobit vzhled grafu podle vlastních představ, obsahuje funkce pro popisky os, název grafu, vkládání textu do grafu, apod.

Níže jsou uvedeny nejpoužívanější funkce a příkazy:

- `hold on`    přepínač pro zapnutí přikreslování dalších grafů do již existujícího grafu. Pro vypnutí této vlastnosti slouží příkaz `hold off`
- `grid on`    zapne zobrazení mřížky pro lepší orientaci v grafu, vypnout lze příkazem `grid off`
- `text(x, y, 'text')`    do obrázku na místo o souřadnicích  $[x, y]$  umístí popisek `text`
- `gtext('text')`    do obrázku na místo interaktivně zvolené myší umístí popisek `text`
- `xlabel('popisx')`    vytvoří popis osy  $x$
- `ylabel('popisy')`    vytvoří popis osy  $y$
- `title('nazev')`    vytvoří název obrázku
- `legend(pozice, 'popis')`    vytvoří legendu grafu tvořenou čárkami oddělenými textovými řetězci s popisem, je zobrazena na místě daném umerickou hodnotou parametru `pozice` (`pozice` může být dána i textovým řetězcem – více viz `doc legend`):

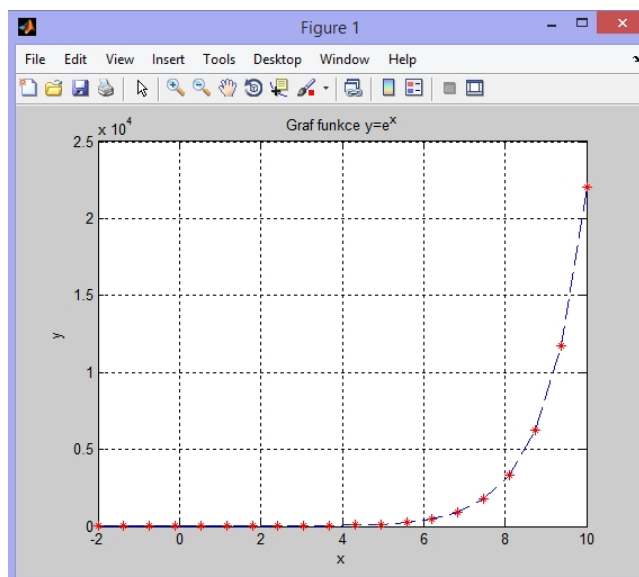
## KAPITOLA 9. PRÁCE S GRAFIKOU

---

- 1 vpravo mimo osy
- 0 uvnitř os
- 1 pravý horní roh
- 2 pravý levý roh
- 3 dolní levý roh
- 4 dolní pravý roh

- `axis([xmin xmax ymin ymax])` numerické hodnoty `xmin`, `xmax`, `ymin`, `ymax` pro úpravu rozsahů os
- `figure()` otevře nové prázdné okno

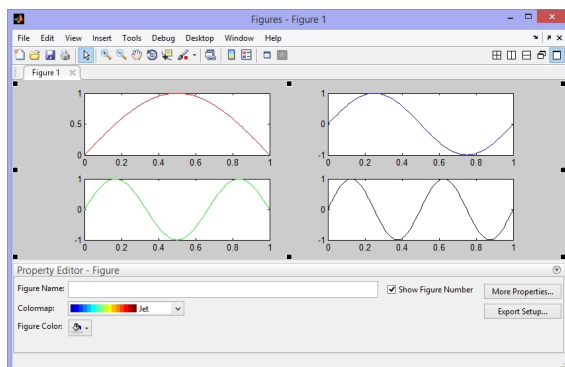
```
>> plot(x, y, 'b--')  výstup je zobrazen na Obr. 9.6
>> grid on  zapnutí mřížky
>> xlabel('x')  popis osy x
>> ylabel('y')  popis osy y
>> title('Graf funkce y=e^x')  název grafu
>> hold on  zapne funkci přikreslování dalších objektů do vytvořeného grafu
>> plot(x, y, 'r*')  přidání bodů do grafu
```



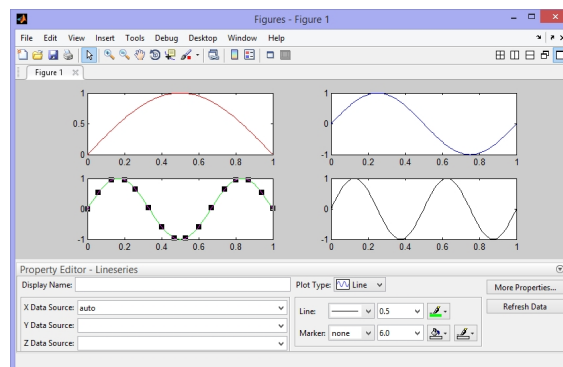
Obr. 9.6. Demonstrace funkcí `title()`, `xlabel()`, `ylabel()` a příkazů `grid on` a `hold on`.

Vzhled grafu lze měnit také přímo z menu grafu *Edit* → *Figure properties ...*, kde lze měnit barva pozadí či název grafu (Obr. 9.7). Poklikáním na vykreslenou křivku lze zobrazit další menu (Obr. 9.8), kde lze měnit typ grafu, barvu, styl a tloušťku čáry a symbolů.

## KAPITOLA 9. PRÁCE S GRAFIKOU

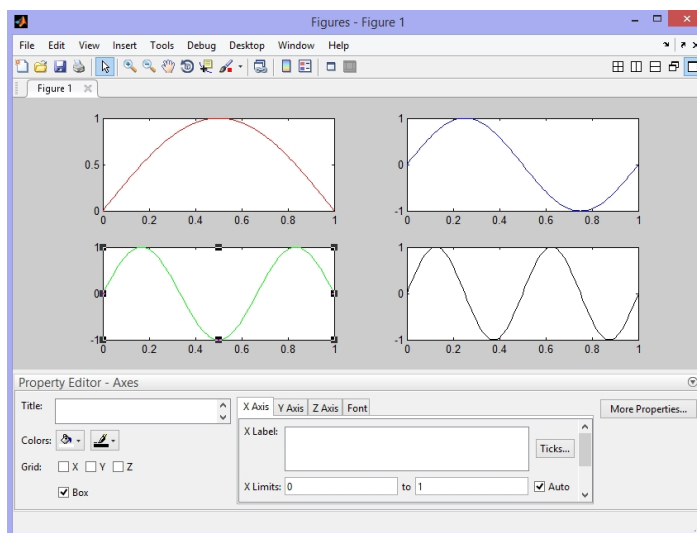


Obr. 9.7. Menu *Figure Properties* ....



Obr. 9.8. Menu *Figure Properties* ....

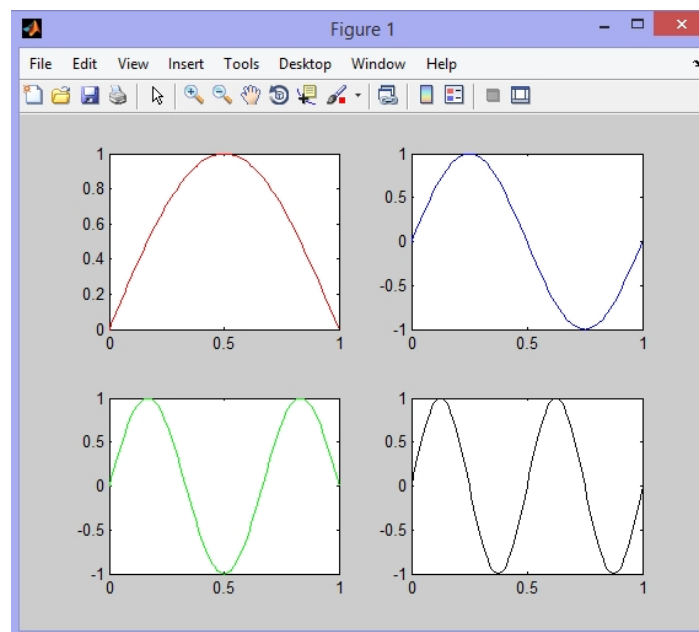
Poklikáním na osy grafu nebo v menu *Edit* → *Axes properties ...* lze v dalším otevřeném menu (Obr. 9.9) měnit popisky, rozsahy a barvy os, lze zapnout mřížka, ohraničení grafu, apod.



Obr. 9.9. Menu *Axes Properties* ....

Složený obrázek obsahující více grafů v jednom okně lze vytvořit funkcí `subplot(m, n, p)`. Parametr `m` udává počet obrázků svisle, `n` počet obrázků vodorovně, parametr `p` pozici pro umístění grafu při aktuálním volání funkce `plot()`. Pozice jsou číslovány po řádcích.

```
>> x = 0:0.01:1;
>> y1 = sin(pi*x); y2 = sin(2*pi*x); y3 = sin(3*pi*x); y4 =
sin(4*pi*x);
>> subplot(2, 2, 1)
>> plot(x, y1, 'r')
>> subplot(2, 2, 2)
>> plot(x, y2, 'b')
>> subplot(2, 2, 3)
>> plot(x, y3, 'g')
>> subplot(2, 2, 4)
>> plot(x, y4, 'k')
```

Obr. 9.10. Funkce `subplot()`.

Vytvořený graf lze uložit v menu grafického okna *File* → *Save as...*

### 9.3 3D grafika

Pro kreslení grafů funkcí dvou proměnných můžeme použít základní funkce:

`mesh(X, Y, Z)` vytvoří 3D síťovaný graf

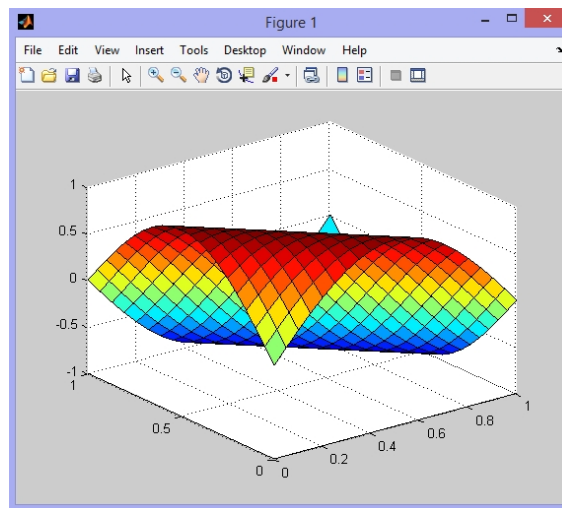
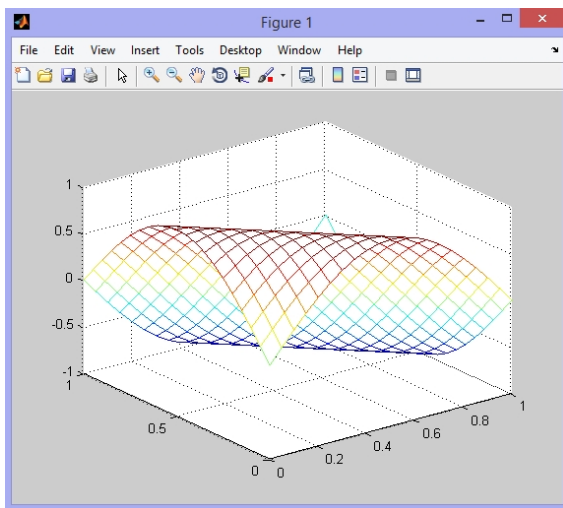
`surf(X, Y, Z)` vytvoří 3D síťovaný graf s barevně vyplněnými ploškami

Parametry *X* a *Y* jsou matice nezávislých proměnných, třetí parametr *Z* obsahuje

matici funkčních hodnot. V případě vynechání nezávislých proměnných je graf indexován rozměry matice  $Z$ .

Pro snadnější vytváření dvojrozměrných grafů slouží funkce `meshgrid()`, která vytvoří z jednorozměrných vektorů nezávislých proměnných dvourozměrné síť vhodné pro definování grafu dané funkce. Použitím `[X, Y] = meshgrid(x, y)` vytvoříme z vektorů  $x$  a  $y$  síť bodů, jejíž souřadnice jsou uloženy v maticích  $X$  a  $Y$ .

```
>> x = 0:0.05:1;
>> y = 0:0.05:1;
>> [X, Y] = meshgrid(x, y);
>> Z = sin(pi*(X+Y));   výpočet funkčních hodnot. POZOR! Je nutno počítat s maticemi X a Y, ne s vektory x a y!
>> mesh(X, Y, Z)      vykreslení síťovaný graf funkce sin(pi*(x+y)) pro x,y in [0,1] (Obr. 9.11)
>> surf(X, Y, Z)     vykreslení síťovaný graf s vyplněnými ploškami pro funkci sin(pi*(x+y)), x,y in [0,1] (Obr. 9.12)
```



Obr. 9.11. Výstup funkce `mesh(X, Y, Z)` Obr. 9.12. Výstup funkce `surf(X, Y, Z)`

Vykreslené graf obsahují celou škálu barev podle funkčních hodnot v jednotlivých bodech. Barevnou stupnici tohoto škálování lze při okraji grafu zobrazit příkazem `colorbar`.

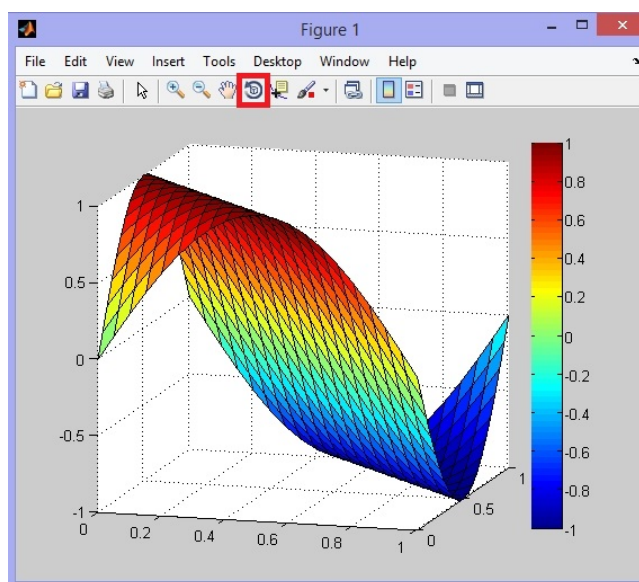
Úhel pohledu na trojrozměrný obrázek lze nastavit funkcí `view(az, el)`. Parametr `az` udává azimut v rovině nezávislých proměnných, tj. otočení ve stupních kolem osy  $z$ ; kladné hodnoty udávají otočení proti směru hodinových ručiček. Parametr `el` udává tzv. elevaci, což je úhel směru pohledu s rovinou nezávislých proměnných. Implicitně je nastaven azimut  $-37.5^\circ$  a elevace  $30^\circ$ .

## KAPITOLA 9. PRÁCE S GRAFIKOU

---

V MATLABu je od verze 5.3 možné nastavovat úhel pohledu pomocí myši přímo v obrázku volbou v panelu nástrojů (na Obr. 9.13 vyznačeno červeně). Podobně lze výběrem položky v menu obrázku nastavovat i vlastnosti dvojrozměrných grafů.

```
>> surf(X, Y, Z)
>> colorbar
>> view(16, 12)
```



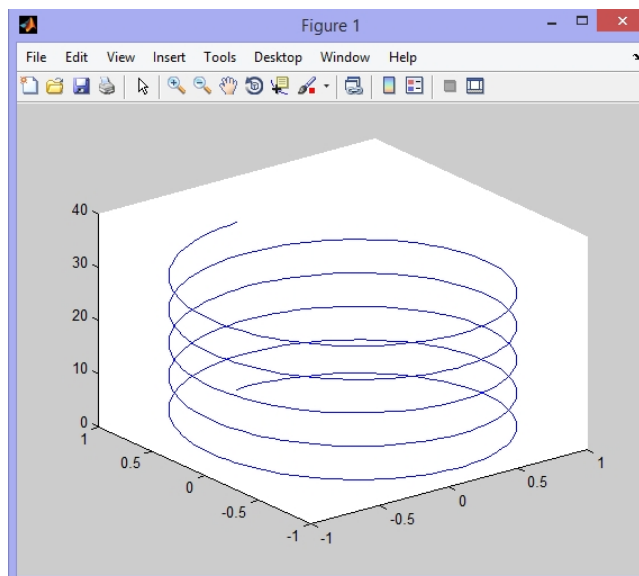
Obr. 9.13. Zobrazení stupnice škálování barev a natočení grafu.

Pro vykreslování křivek v 3D prostoru se používá funkce `plot3(x, y, z)`, jejíž vstupními argumenty jsou vektory  $x$ ,  $y$  a  $z$  stejných rozměrů. Pokud by vstupními argumenty byly matice, funkce by vykreslila křivky postupně pro sloupce těchto matic.

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t), cos(t), t)   vykreslí křivku v prostoru (Obr. 9.14)
```

Dalšími užitečnými funkcemi pro tvorbu 3D grafiky mohou být např. funkce `contour()` (vrstevnicový graf), `meshc()` (síťovaný graf doplněný vrstevnicemi), `waterfall()` (podobný síťovanému grafu, dělající iluzi vodopádu), `quiver()` (vektorové pole), apod.



Obr. 9.14. Výstup funkce `plot3(sin(t), cos(t), t)`.

## 9.4 Vlastnosti grafických objektů

Každý grafický objekt, jako je například graf funkce, popis os, titulek obrázku, ale i obrázek jako celek, má spoustu grafických vlastností. Mezi tyto vlastnosti patří třeba barva grafu, tloušťka čáry grafu, velikost písma a použitý druh písma (font) apod. Výpis všech vlastností lze získat pomocí funkce `get()`, nastavit vlastnosti lze pomocí funkce `set()`. Předtím je ale potřeba, aby byl definován ukazatel na daný grafický objekt, tzv. *handle*. Ten vytvoříme přiřazením grafického příkazu do proměnné.

```
>> x = linspace(0, pi);  
>> y = sin(x);  
>> p = plot(x, y)   vykreslení funkce sinus spolu s přiřazením do proměnné p. Pro-  
tože příkaz není ukončen středníkem, vypíše se hodnota definované proměnné p, ovšem  
vlastní hodnota není důležitá. Zadáním příkazu get(p) získáme výpis všech vlastností  
vykresleného grafu.  
p =  
    176.0278
```

```
>> get(p)   výpis vlastností grafu
  DisplayName: ''
  Annotation: [1x1 hg.Annotation]
  Color: [0 0 1]
  LineStyle: '- '
  LineWidth: 0.5000
  Marker: 'none'
  MarkerSize: 6
  MarkerEdgeColor: 'auto'
  MarkerFaceColor: 'none'
  XData: [1x100 double]
  YData: [1x100 double]
  ZData: [1x0 double]
  BeingDeleted: 'off'
  ButtonDownFcn: []
  Children: [0x1 double]
  Clipping: 'on'
  CreateFcn: []
  DeleteFcn: []
  BusyAction: 'queue'
  HandleVisibility: 'on'
  HitTest: 'on'
  Interruptible: 'on'
  Selected: 'off'
  SelectionHighlight: 'on'
  Tag: ''
  Type: 'line'
  UIContextMenu: []
  UserData: []
  Visible: 'on'
  Parent: 175.0216
  XDataMode: 'manual'
  XDataSource: ''
  YDataSource: ''
  ZDataSource: ''
```

Vlastnosti grafu lze měnit pomocí funkce `set(p, 'PropertyName', PropertyValue)`, kde `'PropertyName'` označuje název vlastnosti (ve výpisu text před dvojtečkou) a `PropertyValue` její novou hodnotu.

## Příklady k procvičení

1. Nakreslete graf funkce  $f_1(x) = x^2$  pro  $x \in [-2, 2]$ . Graf řádně otitulujte.
2. Červeně přikreslete graf funkce  $f_2(x) = x^4$ .
3. Do nového grafického okna nakreslete tyto grafy vedle sebe.
4. Jedním příkazem vykreslete pro  $x \in [-6, 6]$  graf funkce

$$f_3(x) = \begin{cases} (x+3)^3 + x & x \leq 0 \\ (x-2)^4 + (x-5)^2 + 1 & x > 0 \end{cases}$$

5. Pro  $x, y \in [-2, 2]$  vykreslete graf funkce  $g(x) = x^2 + y^2$ .

*Řešení.*

1. 

```
x = linspace(-2, 2);  
plot(x, x.^2)  
title('Graf 2. mocniny'), xlabel('osa x'), ylabel('osa y')
```
2. 

```
hold on  
plot(x, x.^4, 'r')
```
3. 

```
figure  
subplot(1, 2, 1)  
plot(x, x.^2), title('2. mocnina'), xlabel('x'), ylabel('y')  
subplot(1, 2, 2)  
plot(x, x.^4), title('4. mocnina'), xlabel('x'), ylabel('y')
```
4. 

```
x = linspace(-6, 6); figure  
y = ((x+3).^3 + x).*(x <= 0) + ((x-2).^4 + (x-5).^2 + 1).*(x > 0);  
plot(x, y)
```
5. 

```
x = linspace(-2, 2); y = x; figure  
[X, Y] = meshgrid(x, y);  
surf(X, Y, X.^2 + Y.^2)
```

# Kapitola 10

## Programování v MATLABu

### Základní informace

MATLAB je prostředí nejen pro výpočty, modelování, simulace a grafické zobrazení dat, ale také programovací prostředí, které svým uživatelům umožňuje vytváření svých vlastních programů či přizpůsobení již existujících funkcí podle vlastních potřeb. Pro tvorbu těchto funkcí je nezbytně nutné, aby uživatel dobře porozuměl rozdílu mezi dávkovým souborem a funkcí a dalším základním věcem jako lokální a globální proměnné a základní programové struktury.

### Výstupy z výuky

Studenti

- umí vytvářet jednoduché vlastní skripty a funkce, znají rozdíl mezi nimi
- rozlišují lokální a globální proměnné
- ovládají schémata větvení programu pomocí příkazů "if" a "switch", umí tyto příkazy demonstrovat na jednoduchých příkladech
- orientují se v příkazech cyklů "while" a "for", znají rozdíly v jejich použití
- seznámí se s některými alternativami pro větvení programů a cyklů
- dokáží použít příkazy pro ladění programu

## 10.1 Dávkové soubory (skripty) a funkce

Programy v MATLABu, které si může uživatel běžně vytvořit, lze rozdělit do dvou skupin: dávkové soubory neboli skripty a funkce. Hlavní rozdíl mezi nimi je v tom, že funkce může pracovat se vstupními a výstupními proměnnými, dávkový soubor nikoliv. Další rozdíl je v lokálních a globálních proměnných, ten bude popsán dále. Obě skupiny programů řadíme mezi tzv. M-fajly, neboť jsou uloženy v souborech s příponou `.m` – např. `dávka1.m`, `funkce2.m` apod.

**Dávkový soubor** obsahuje příkazy MATLABu, které bychom mohli zadávat přímo z klávesnice. Důvodem jejich uložení do souboru může být třeba to, že stejnou sekvenci příkazů budeme potřebovat vícekrát. Důležitou roli zde má soubor s názvem `"startup.m"`, který se vykoná při každém spuštění programu MATLAB, pokud existuje v adresáři, v němž MATLAB používáme. V souboru `"startup.m"` může být např. úvodní nastavení formátu, otevření záznamu práce příkazem `diary`, atd.

Příklad obsahu souboru `startup.m`:

```
format compact
diary on
disp('Program MATLAB Vás vítá!')
disp(' ')
disp('Váš pracovní adresář je:')
disp(pwd)
```

**Funkce** musí začínat hlavičkou, která má tvar:

```
function [vystupni_parametry] = nazev_funkce(vstupni_parametry)
```

a měla by končit příkazem `end`.

Vytvořená funkce musí být uložena v souboru s příponou `.m`. Je doporučováno volit shodný název pro funkci i soubor, v opačném případě je při volání funkce rozhodující název souboru.

Seznamy vstupních a výstupních parametrů jsou seznamy proměnných oddělených čárkami. Tyto proměnné je možné libovolně používat v příkazech uvnitř funkce, přičemž všechny výstupní proměnné by měly mít přiřazenou hodnotu před ukončením běhu funkce. Pokud je výstupní proměnná jen jedna, nemusí být uzavřena v hranatých závkách. Vstupní ani výstupní proměnné nejsou povinné, hlavička funkce může v tomto případě vypadat následovně:

```
function funkce1
```

Na řádcích pod hlavičkou může být umístěna nápověda k funkci. Jedná se o řádky začínající znakem `%` (komentáře) obsahující popis chování funkce. Nápověda se zobrazí pomocí funkce `help nazev_funkce` nebo `doc nazev_funkce`.

Příklad jednoduché funkce:

```
function [P,o] = trojuh(a, b, c)
% [P,o] = trojuh(a, b, c)
% Funkce pro výpočet obvodu a obsahu trojúhelníka
% a, b, c - délky stran
% P - obsah, o - obvod
o = a+b+c;
s = o/2;
P = sqrt(s*(s-a)*(s-b)*(s-c));
end
```

Uvedené příkazy uložíme do souboru s názvem `trojuh.m` v pracovním adresáři nebo v adresáři, do kterého je nastavena cesta.

Názvy skutečných proměnných, které předáváme funkci jako parametry, se samozřejmě nemusí shodovat se jmény proměnných ve funkci samotné, vstupní parametry je možné zadávat i přímo pouze hodnotami.

```
>> [x,y] = trojuh(3, 4, 5)
x =
    6
y =
   12
```

V případě, že při volání použijeme méně výstupních parametrů, než je v definici funkce, jsou funkcí přiřazeny příslušné hodnoty zleva, pokud tuto situaci neřeší funkce samotná.

```
>> trojuh(3, 4, 5) získáme pouze obsah trojúhelníka, jehož hodnota bude je přiřazena v proměnné ans
```

Funkce je ukončena po vykonání všech příkazů, které obsahuje. Je také možné funkci ukončit dříve pomocí příkazu `return`. Předčasné ukončení činnosti funkce dosáhneme též funkcí `error('chyba')`, která navíc vyvolá zvukový signál a vypíše textový řetězec `chyba`.

```
>> A = [];
>> [m, n] = size(A);
>> if m*n == 0 % matice a je prazdna
error('Prazdna matice!'); % vypis chyboveho hlaseni (cervene) v
pripade, ze je podminka splnena
end
Prazdna matice!
```

## 10.2 Lokální a globální proměnné

Všechny proměnné definované v MATLABu jsou implicitně považovány za lokální, tj. nejsou známy mimo aktivní prostředí, kterým může být volaná funkce nebo pracovní okno s příkazovým řádkem. Tedy pokud jsme v pracovním okně definovali na příkazovém řádku proměnnou `x`, pak ve volané funkci bude tato proměnná neznámá. Pokud si uvnitř funkce definujeme také proměnnou `x` nebo tak bude označen vstupní, případně výstupní parametr, nebude to mít žádný vliv na proměnnou `x` definovanou v příkazovém řádku. Naopak jakékoliv proměnné definované během práce nějaké funkce nejsou známy mimo tuto funkci.

Výjimku tvoří dávkové soubory, ve kterých jsou známé proměnné definované v prostředí, odkud byly zavolány. Naopak pokud v dávce nějaké proměnné definujeme, jsou pak známé i po jejím ukončení.

Mějme soubor `davka1.m` obsahující příkazy

```
[P1, o1] = trojuh(3, 4, 5);  
[P2, o2] = trojuh(7, 8, 9);
```

Po zadání příkazu `davka1` z příkazové řádky jsou definovány proměnné `P1`, `o1`, `P2`, `o2` obsahující spočtené hodnoty. Podobně bychom mohli tyto hodnoty použít v nějaké funkci, která by obsahovala příkaz `davka1`.

V případě, že potřebujeme použít nějakou proměnnou definovanou v příkazové řádce i v nějaké funkci, musíme ji deklarovat jako globální pomocí příkazu `global`, a to jak v příkazové řádce tak v těle funkce. Tato deklarace by se měla použít před přiřazením hodnoty této proměnné.

## 10.3 Základní programové struktury

Mezi základní programové struktury patří příkaz větvení a příkaz cyklu. Tyto struktury je samozřejmě možné použít i v příkazové řádce MATLABu. Nejprve se tedy zmíníme o větvení programu.

### 10.3.1 Větvení programu

Větvení se provádí příkazem `if`. Syntaxe jeho použití se řídí následujícím schématem:

```
if podmínka1  
    příkazy1  
elseif podmínka2  
    příkazy2  
else  
    příkazy3  
end
```

Větvě `else` a `elseif` jsou samozřejmě nepovinné, přičemž `elseif` je možné použít vícekrát. Příkazů v každé větvi může být víc. Znázorněné odsazení je nepovinné a je použito kvůli větší přehlednosti. Všechny podmínky, příkazy i klíčová slova je možné uvést v jediném řádku, v tomto případě je nutno použít oddělovač příkazů, tedy čárku nebo středník.

Podmínky po klíčových slovech `if` a `elseif` jsou obecně matice. Platí, že podmínka je splněna, jestliže všechny její prvky jsou nenulové. Například pokud má *podmínka1* tvar `A==B`, kde `A`, `B` jsou matice, pak tento výraz vrátí matici s jedničkami nebo nulami, podle toho, zda se jednotlivé odpovídající prvky shodují nebo ne. Větev *příkazy1* by se provedla jen v tom případě, že výsledná matice obsahuje jenom jedničky.

Klíčové slovo `elseif` je možné nahradit dvojicí `else` a `if`, ovšem potom je potřeba o jeden `end` více. Schéma by pak vypadalo následovně:

```
if podmínka1
    příkazy1
else
    if podmínka2
        příkazy2
    else
        příkazy3
    end
end
```

Jako příklad vytvoříme soubor `davka1.m`. Po zadání příkazu `davka1` z příkazové řádky je uživatel vyzván, aby zadal libovolné číslo. Po zadání hodnoty a stisknutí klávesy ENTER se vypíše na obrazovku, jestli uživatel zadal kladné nebo záporné číslo.

```
s = input('Zadejte libovolne cislo: ')
if s < 0
    disp('Zadali jste zaporne cislo.');
```

```
elseif s > 0
```

```
    disp('Zadali jste kladne cislo.');
```

```
else
```

```
    disp('Vami zadana hodnota je bud 0 nebo to neni cislo!');
```

```
end
```

Dalším typem větvení je `switch`. Syntaxe jeho použití se řídí následujícím schématem:

```
switch výraz
    case případ1
        příkazy1
    case případ2
        příkazy2
    :
```



```
    otherwise
        příkazy_jiné
end
```

Tento typ větvení se používá především v situacích, kdy proměnná **výraz** nabývá více hodnot. Přepínač **switch** sleduje hodnotu proměnné **výraz** a pro jednotlivé případy (**case**) provede příslušné **příkazy**. Pokud nenastane žádný z popsaných případů, vykonají se **příkazy\_jiné**. Znázorněné odsazení je nepovinné a je použito kvůli větší přehlednosti.

Jako příklad vytvoříme soubor `davka2.m`, ve kterém je uživatel vyzván, aby zadal číslo, které odpovídá pořadí dne v týdnu. Po zadání hodnoty a stisknutí klávesy ENTER se vypíše na obrazovku den, který odpovídá zadanému číslu.

```
s = input('Zadejte poradí dne v týdnu: ') switch s
case 1
    disp('Zadali jste číslo pro pondělí.');
```

```
case 2
    disp('Zadali jste číslo pro úterý.');
```

```
case 3
    disp('Zadali jste číslo pro středu.');
```

```
case 4
    disp('Zadali jste číslo pro čtvrtek.');
```

```
case 5
    disp('Zadali jste číslo pro pátek.');
```

```
case 6
    disp('Zadali jste číslo pro sobotu.');
```

```
case 7
    disp('Zadali jste číslo pro neděli.');
```

```
otherwise
    disp(['Zadane číslo ', num2str(s), ' neodpovídá žádnému dni!']);
end
```

### 10.3.2 Cykly

Pro cyklus jsou v MATLABu dva příkazy. Je to příkaz **while** a příkaz **for**. Cyklus **while** se používá v případech, kdy předem neznáme počet průběhů cyklem, který je závislý na předem splnění dané podmínky. Použití příkazu **while** vypadá následovně:

```
while podmínka
    příkazy
end
```

Pro vyhodnocení podmínky `podmínka` platí v podstatě tatáž pravidla jako pro příkaz `if`. Příkazy mezi `while` a `end` se vykonávají, pokud je podmínka pravdivá.

Jako příklad vytvoříme soubor `davka3.m`, po jehož spuštění z příkazové řádky je uživatel vyzván, aby zadal nějaký text. Po zadání textu a stisknutí klávesy `ENTER` se vypíše na obrazovku libovolná permutace textu. Zadá-li uživatel pouze klávesu `ENTER`, dávka se ukončí.

```
s = input('Zadejte libovolny text (konec = ''ENTER''):', 's');
n = length(s);
while n ~= 0
    disp('Libovolna permutace zadaneho textu je:');
    disp(s(randperm(n)));
    s = input('Zadejte libovolny text (konec = ''ENTER''):', 's');
    n = length(s);
end;
```

Příkaz `for` se používá především v případech, kdy předem známe počet průchodů cyklem. Jeho syntaxe je následující:

```
for prom=výraz
    příkazy
end
```

Výraz v uvedeném přiřazení dá obecně matici. Proměnná *prom* je pak sloupcový vektor, který v průběhu cyklu postupně nabývá hodnot jednotlivých sloupců této matice. Velmi typické je následující použití:

```
for k=1:n
    příkaz
end
```

Je třeba si uvědomit, že výraz `"1:n"` vytvoří matici o jednom řádku a *n* sloupcích, hodnoty v tomto řádku budou čísla od 1 do *n*, takže proměnná *k* bude postupně nabývat těchto hodnot. V tomto případě se tedy chová příkaz `for` podobně, jak to známe z jiných programovacích jazyků. Pokud je jako výraz použita nějaká konstantní matice a v průběhu cyklu ji změníme, proměnná *prom* bude nabývat původních hodnot sloupců matice. Běh obou cyklů je možné předčasně přerušit, slouží k tomu příkaz `"return"`. Tato situace může nastat třeba při řešení soustavy lineárních rovnic, kdy během výpočtu zjistíme, že matice soustavy je singulární.

V MATLABu je často možné nahradit použití cyklu jedním nebo několika příkazy, pokud využijeme některé již definované funkce. Jako příklad nám může sloužit výpočet faktoriálu. Pokud chceme vypočítat faktoriál z čísla *n* v běžném programovacím jazyce, postupujeme obvykle následovně:

```
faktor = 1;
for k = 1:n
    faktor = faktor * k;
end
```

Pro tento účel stačí v MATLABu napsat příkaz

```
faktor = 1;
for k = 1:n
    faktor = faktor * k;
faktor = prod(1:n);
end
```

Tento příkaz dá správný výsledek i pro  $n=0$ , neboť součin přes prázdnou matici dává jako výsledek jedničku.

## 10.4 Nástrahy při programování v MATLABu

Výše naznačený postup – totiž práce s vektory a s maticemi nikoliv v cyklech, ale se všemi prvky v jediném příkazu najednou – je pro MATLAB typický. V tom také spočívá jedna z velkých předností tohoto systému – možnost psát programy velmi efektivně. Skrývá se zde ale také kámen úrazu, dokonce i pro zkušené programátory, kteří mohou být navyklí na jiný způsob práce.

Uvedme jednoduchý příklad. Potřebujeme definovat nějakou funkci po částech, dejme tomu  $f(x)$  bude mít hodnotu  $x^2$  pro  $x < 0$  a hodnotu  $x^3$  pro  $x \geq 0$ . První věc, která by mnohé napadla, je udělat to následovně:

```
function y = f(x)
    if x < 0
        y = x^2;
    else
        y = x^3;
    end
```

Tento postup je zajisté správný, pokud by se jednalo o program řekněme v jazyce C nebo Pascal, kde při počítání funkčních hodnot pro nějakou množinu bodů postupujeme v cyklu. Ale v MATLABu je zvykem, že jako argument funkce může být použit vektor nebo matice, funkce pak vrátí vektor či matici stejného řádu, kde na odpovídajících místech budou funkční hodnoty v původních bodech. Tohle výše uvedená funkce evidentně nedělá.

Vypadá to, že stačí, když opravíme operátor  $\wedge$  na operátor  $\wedge$ , který pracuje po složkách. Tato úprava ale nestačí. Jak bylo vysvětleno výše, výraz za klíčovým slovem `if` je matice nul a jedniček stejného řádu jako proměnná  $x$ . Stačí, aby nula byla je-

diná, a provede se druhá větev programu. Tedy jestliže jediná složka matice  $x$  bude nezáporná, pak výsledkem budou na všech místech výstupu třetí mocniny. Pokud ale budou všechny složky záporné, výsledek bude kupodivu správný.

Pokusme se funkci opravit. Jeden ze způsobů, jak to udělat, je provést přiřazování v cyklech. Výsledek pak vypadá takto:

```
function y = f1(x)
    [m,n] = size(x);    % zjištění rozměrů matice
    y = zeros(size(x)); % definice výstupní matice stejných rozměrů
    for k = 1:m
        for l = 1:n
            if x(k,l) < 0
                y(k,l) = x(k,l)^2;
            else
                y(k,l) = x(k,l)^3;
            end
        end
    end
end
```

Tento postup je po stránce výsledků správný, ztrácí se jím ale veškeré výhody MATLABu. Problém lze vyřešit mnohem efektivněji. Jednak by bylo možné použít pouze jeden cyklus ve tvaru `for k = 1:m*n` a při indexování pak zadávat pouze jeden index, např.  $y(k) = x(k)^2$ ;

Lze se ale obejít bez cyklů úplně. Stačí, jestliže vytvoříme dvě matice stejného řádu jako  $x$ , první bude mít jedničky na místech záporných složek  $x$  a nuly jinde, u druhé tomu bude naopak. Tyto matice vynásobíme druhými resp. třetími mocninami složek  $x$  a po sečtení vyjde správný výsledek. Pro lepší pochopení uvedeného postupu uvedeme následující program:

```
function y = f2(x)
    p1 = x < 0;
    p2 = x >= 0;
    y = p1.*x.^2 + p2.*x.^3;
end
```

Je dokonce možné provést zkrácení na jediný řádek, pokud nepočítáme hlavičku funkce:

```
function y = f3(x)
    y = (x<0).*x.^2 + (x>=0).*x.^3;
end
```

Tato verze je nejen daleko kratší, ale rovněž funguje rychleji, protože provádění cyklů je v MATLABu poměrně pomalé, kdežto pro manipulaci s maticemi se používají interní MATLABovské funkce, které pracují daleko efektivněji.

## 10.5 Ladění programu

Běžně se stává, že hotový program sice pracuje, ale chová se podivně nebo jeho výsledky nejsou ve shodě s očekáváním. V tomto případě je potřeba najít chybu, v čemž mohou pomoci ladící prostředky MATLABu. V novějších verzích je ladění umožněno v rámci editoru programů, který je součástí MATLABu. Popíšeme ale i prostředky, které umožňují ladění z příkazové řádky.

K ladění slouží následující příkazy:

`dbstop`, `dbstep`, `dbclear`, `dbcont`, `dbstack`, `dbtype`, `dbquit`, `dbup`, `dbdown`, `dbstatus`

Příkazem `dbstop` je možné nastavit zastavení programu v daném místě. Syntaxe příkazu je

<code>dbstop in m-file</code>	zastaví v daném souboru obsahující funkci na prvním příkazu
<code>dbstop in m-file at line</code>	zastaví v daném souboru na dané řádce
<code>dbstop in m-file at subfun</code>	zastaví v daném souboru na začátku dané podfunkce
<code>dbstop if error</code>	zastaví v případě chyby
<code>dbstop if warning</code>	zastaví v případě varování
<code>dbstop if naninf</code>	zastaví v případě výskytu hodnoty <code>Inf</code> nebo <code>NaN</code>
<code>dbstop if infnan</code>	zastaví v případě výskytu hodnoty <code>Inf</code> nebo <code>NaN</code>

Po zastavení běhu programu je možné kontrolovat hodnoty proměnných jejich výpisem, případně je opravovat. Příkazem `dbstack` můžeme zobrazit posloupnost volání jednotlivých funkcí v místě zastavení. Příkazem `dbtype` můžeme zobrazit daný soubor včetně čísel řádků. Pokud chceme zobrazit řádky v daném rozmezí, použijeme `dbtype mfile n1:n2`.

Příkaz `dbstep` provede příkaz na řádce, kde se běh programu zastavil. Příkazem `dbstep n` se provede `n` řádků od místa zastavení. Pokud je na místě zastavení volání funkce a chceme pokračovat s laděním uvnitř této funkce, použijeme příkaz `dbstep in`.

Pomocí příkazu `dbcont` se spustí další běh programu. Příkazem `dbquit` se předčasně ukončí činnost laděného programu. Pokud chceme zjistit, jaké byly hodnoty proměnných v nadřazené funkci nebo v základním prostředí, lze použít příkaz `dbup`, který

zajistí zavedení proměnných z prostředí nadřazeného funkci, v níž se právě nacházíme. Opačně funguje funkce `dbdown`.

Seznam všech míst zastavení získáme příkazem `dbstatus`. Odstranit bod zastavení lze pomocí `dbclear`.

## Příklady k procvičení

1. Vytvořte funkci `nasobky()`, která pro vstupní parametry `k` a `n` vypíše prvních `k` násobků čísla `n`.
2. Vytvořte funkci `test()`, která pro vstupní vektor známek (1 – 5) testu z matematiky určí četnosti jednotlivých hodnocení. Výstup uložte do matice – v prvním sloupci hodnocení, ve druhém sloupci počet hodnocení. Funkce by také měla zkontrolovat, zda se opravdu jedná o celočíselný vstupní vektor s hodnotami 1, 2, ..., 5, v opačném případě by měla skončit chybovým hlášením.
3. Vytvořte funkci `soucet()`, která bude náhodně generovat hodnoty z intervalu  $[0, 1]$ , dokud jejich součet nepřevýší hodnotu vstupního parametru `s`. Funkce na svém výstupu vypíše vektor vygenerovaných hodnot `vektor` a jejich součet `suma`. Funkce by měla ověřit, zda je `s` numerická hodnota větší než 1, v opačném případě by měla skončit chybovým hlášením.

*Řešení.*

1. 

```
function[vystup] = nasobky(k, n)
    % pro zadana "k" a "n" vypise prvnich "k" nasobku cisla "n"

    vystup = (1:k) .* n;

end
```
2. 

```
function[vystup] = test(v)
    % pro zadany vektor hodnoceni testu vypocita jejich cetnosti

    if (isnumeric(v) == 0) | any(v - round(v) ~= 0) | any(v < 1) ...
        | any(v > 5)
        error('Spatne zadana hodnoceni!')
    end

    for i = 1:5
        cetnost(i) = sum(v == i);
```

```
    vystup = [1:5; cetnost]';  
end
```

```
3. function[vektor, suma] = soucet(s)
```

```
    if isnumeric(s) == 0 | s <= 1  
        error('Spatne zadana hodnota s!')  
    end
```

```
    suma = 0;  
    while suma <= s  
        suma = suma + rand(1);  
    end
```

```
end
```

## Seznam použité literatury

- [1] DUŠEK, F. *MATLAB a SIMULINK - úvod do užívání*. Univerzita Pardubice, 2000. 147 s. ISBN 80-7194-273-1
- [2] PÄRT-ENANDER, Eva. *The Matlab handbook*. Harlow: Addison-Wesley, 1997. 423 s. ISBN 0-201-87757-0
- [3] The Mathworks, autoři MATLABu a SIMULINKu. *The Mathworks*. [online], [září 2014]. Dostupné z WWW: <<http://www.mathworks.com/>>
- [4] ZAPLATÍLEK, K., DOŇAR, B. *MATLAB pro začátečníky*. 1. vydání. Praha: BEN - technická literatura, 2003. 144 s. ISBN 80-7300-095-4