Jak pracovat s MATLABem

Jiří Zelinka
Jan Koláček

Obsah

1 Úvod **1**
- 1.1 Základní ovládání . . . 1
- 1.2 Nápověda . . . 1
- 1.3 Jednoduché výpočty . . . 2
- 1.4 Proměnné, matice a jejich definování . . . 2
- 1.5 Funkce pro tvorbu matic . . . 3
- 1.6 Další operace s maticemi . . . 4
- 1.7 Tipy a triky . . . 4

2 Maticové operace **5**
- 2.1 Transpozice matic . . . 5
- 2.2 Sčítání a odčítání matic . . . 5
- 2.3 Maticové násobení . . . 6
- 2.4 Maticové dělení . . . 6
- 2.5 Umocňování matic . . . 7
- 2.6 Operace po složkách . . . 8
- 2.7 Logické operace . . . 9
- 2.8 Smíšené operace . . . 9
- 2.9 Příklady . . . 9

3 Příkazy MATLABu **10**
- 3.1 Obecná pravidla . . . 10
- 3.2 Některé speciální výrazy a funkce . . . 10
- 3.3 Manipulace s maticemi . . . 11
- 3.4 Další funkce pro práci s maticemi . . . 12
- 3.5 Základní funkce lineární algebry . . . 13
- 3.6 Polynomy . . . 14
- 3.7 Tipy a triky . . . 14

4 Práce se soubory **15**
- 4.1 Záznam práce . . . 15
- 4.2 Ukládání a načítání proměnných . . . 15
- 4.3 Soubory v systému MATLAB . . . 16
- 4.4 Cesta k souborům . . . 16

4.5 Další příkazy pro práci se soubory 17

5 Logické operace **18**
5.1 Relační operátory 18
5.2 Logické operátory 18
5.3 Logické funkce 19
5.4 Funkce `find` a `exist` 20
5.5 Příklady 21
5.6 Tipy a triky 22

6 Textové řetězce **23**
6.1 Vytváření řetězců 23
6.2 Základní manipulace s řetězci 23
6.3 Funkce pro manipulaci s řetězci 24
6.4 Funkce `eval` a `feval` 25

7 Vyhodnocování výrazů **26**
7.1 Výraz jako textový řetězec 26
7.2 Symbolický výraz 26
7.3 Výraz jako INLINE funkce 27

8 Práce s grafikou **28**
8.1 Grafy funkcí 28
8.2 Vlastnosti grafických objektů 30

9 Programování v MATLABu **32**
9.1 Dávkové soubory (skripty) a funkce 32
9.2 Lokální a globální proměnné 33
9.3 Základní programové struktury 34
9.3.1 Větvení programu 34
9.3.2 Cykly 36
9.4 Nástrahy při programování v MATLABu 37
9.5 Ladění programu 39

Abstrakt

Tento návod je pracovním textem určeným zejména pro studenty mající zapsaný předmět „Výpočetní matematické systémy“, ale je k dispozici i všem dalším zájemcům o práci s programovým systémem MATLAB. Text bude postupně doplňován. Autoři uvítají jakékoliv dobře míněné podněty a připomínky.

Návod je psán pro verzi MATLABu 5.3, u níž se téměř výlučně pracuje s příkazovou řádkou. Od verze 6 má MATLAB okenní strukturu s bohatšími možnostmi, ale příkazová řádka zůstává pro uživatele i nadále základním komunikačním prostředkem. Dá se tedy očekávat, že všechny popsané příkazy budou bez problémů fungovat i ve verzích 6.x, případně vyšších. Většina příkazů by měla být funkční i ve verzích starších.

V textu jsou příkazy MATLABu a jména proměnných uvedeny v uvozovkách, pokud příkaz neleží na samostatném řádku. V apostrofech jsou jednotlivé znaky a textové řetězce.

Kapitola 1

Úvod

1.1 Základní ovládání

Spouštění MATLABu je závislé na použitém operačním systému. Např. v Linuxu zpravidla spouštíme MATLAB zadáním příkazu "`matlab`" v příkazovém řádku. Ve Windows klikneme na příslušnou ikonu nebo program najdeme někde v menu. MATLAB ukončíme příkazem "`exit`" nebo zavřením okna, v němž MATLAB běží. Příkazy MATLABu zadáváme do příkazového řádku, přičemž můžeme využívat v MATLABu definovaných i vlastních funkci a procedur. Na jeden řádek je možné zadat více příkazů, pak je oddělujeme čárkou nebo středníkem. Pokud chceme příkaz rozdělit na více řádků, můžeme použít tři tečky jako pokračovací znaky.
Další základní funkce:

- pokud příkaz ukončíme středníkem, MATLAB nevypisuje výsledek na obrazovku
- klávesy $\boxed{\uparrow}$ $\boxed{\downarrow}$ umožňují pohyb v historii příkazů
- pro přerušení prováděného příkazu použijeme kombinaci CTRL+C
- klávesa ESC vymaže příkazový řádek
- proměnné v MATLABu musí začínat písmenem a mohou mít až 31 znaků
- rozlišují se malá a velká písmena

1.2 Nápověda

Nápovědu získáme příkazem "`help`", např. "`help sin`" nám dá nápovědu k použití známé goniometrické funkce. Příkaz "`help`" samotný vypíše seznam všech tematicky oblasti, na než je pak možné získat nápovědu opět příkazem "`help`" - např. "`help matfun`". Samostatné okno s nápovědou otevřeme příkazem "`helpwin`". HTML stránky s dokumentaci získáme příkazem "`doc`" a demonstrační program k MATLABu vyvoláme zadáním příkazu "`demo`". Informace o aktuální verzi MATLABu a nainstalovaných knihovnách získáme použitím příkazu "`ver`".

1.3 Jednoduché výpočty

V rámci MATLABu lze provádět libovolné výpočty, které lze počítat na kalkulačce: např. "`3*(15.8+3^8)-sin(pi/12)+sqrt(4.8)`" dá výsledek uvedeného výrazu. Přitom "`pi`" je známé Ludolfovo číslo, "`sqrt`" je odmocnina. Další definované konstanty jsou "`i,j`" pro komplexní jednotku. Známé funkce použitelné v MATLABu jsou například:

```
sin, cos, tan, cot, asin, acos, atan, acot,
sinh, cosh, tanh, coth, asinh, acosh, atanh, acoth,
exp, log, log10, log2, pow2, sqrt, abs.
```

Běžné operátory: +, –, *, /, ^ (podrobněji je popisuje Kapitola 2.)

Pomocí kulatých závorek řídíme prioritu operátorů a umisťujeme do nich argumenty funkci.

Pro 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

1.4 Proměnné, matice a jejich definování

Výsledek jakékoliv operace můžeme uložit do proměnné, kterou lze použít při dalších výpočtech. Název proměnné je libovolná posloupnost písmen, číslic a znaku '_' začínající písmenem. Velikost písmen je důležitá, tj. "`A`" je jiná proměnná než "`a`".

Do proměnné přiřadíme výsledek výrazu pomocí znaku '='. Např. příkaz

```
x=12*sin(pi/20)
```

definuje proměnnou "`x`", kterou můžeme dále použít: "`x8=8*x`".

Proměnné mohou být i maticemi, vlastně i běžná čísla se pokládají za matice typu 1x1. Matice můžeme zadat výčtem jejich prvků v hranatých závorkách, přičemž jednotlivé prvky na řádku oddělujeme mezerou nebo čárkou, jednotlivé řádky matice oddělujeme středníkem. Příklad:

```
A=[1 -1 2 -3; 3 0 4 5;3.2, 5, -6 12]
u=[1 2 3 4]
v=[-1;-2;-3]
```

Prázdnou matici vytvoříme příkazem

```
o=[]
```

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

```
y=[x, 2*x, 3*x]
B=[A; u]
C=[A, v]
```

Seznam definovaných proměnných lze získat příkazem "`who`". Příkaz "`whos`" zobrazí seznam proměnných i s jejich typy a dalšími informacemi. Zrušit definovanou proměnnou je možno přikáže "`clear`", např.

```
clear i
```

můžeme zrušit proměnou označenou "`i`", kterou jsem omylem předefinovali komplexní jednotku. Zadáním "`clear all`" zrušíme všechny definované proměnné.

1.5 Funkce pro tvorbu matic

Pro vytvoření matic větších rozměrů je možné použít některých funkci MATLABu. Příkaz

```
Z=zeros(2,5)
```

vytvoří nulovou matici příslušného typu, příkazem

```
O=ones(3,4)
```

vyrobíme matici ze samých jedniček. Příkaz

```
I=eye(5,8)
```

nám dá 'jednotkovou' matici, přičemž jedničky probíhají hlavní diagonálu. Všechny výše uvedené příkazy je možné zadat i s jedním parametrem, v tom případě je výsledkem čtvercová matice příslušného řádu. Funguje taky příkaz

```
o=ones(0,5)
```

Matici s náhodnými prvky lze vytvořit příkazem

```
R1=rand(3,5),
```

kdy prvky budou mít hodnoty mezi 0 a 1, nebo příkazem

```
R2=randn(6)
```

pro matici s prvky majícími standardizované normální rozdělení. Vektory jsou považovány za matice s jedním řádkem nebo jedním sloupcem. Vektor, jehož prvky tvoří aritmetickou posloupnost (tzv. „ekvidistantní“ vektor), je možné vytvořit následujícím způsobem:

```
x=1:10
```

vytvoří vektor postupně obsahující čísla 1 až 10. Pro jiný rozdíl mezi jednotlivými prvky než 1 lze použít např. příkaz

```
y=0:2:12
```

pro y se sudými čísly od 0 do 12 nebo

```
z=0:0.01:2
```

pro z s čísly od 0 do 2 s krokem 0.01. Je možno použít i záporný krok:

```
x1=10:-1:1
```

Příkaz "`a:b`" případně "`a:d:b`" lze nahradit příkazem "`colon(a,b)`" nebo "`colon(a,d,b)`". Tj. např. poslední uvedenou proměnnou můžeme analogicky definovat:

```
x1=colon(10,-1,1)
```

Další alternativou pro generování ekvidistantních vektorů jsou příkazy "`linspace`" a "`logspace`". Ty jsou podrobněji popsány v odstavci 3.3.

1.6 Další operace s maticemi

Rozměr matice zjistíme příkazem "`size`". Na jednotlivé prvky matice je možné se odkázat pomocí kulatých závorek - tj. "`A(3,2)`" je prvek matice "`A`" ve 3. řádku a 2. sloupci. Toto vyjádření ovšem lze použít obecněji, kdy první parametr je vektor obsahující indexy vybraných řádků a druhý parametr je vektor sloupcových indexu. Pak

```
A([1 3],[5 2])
```

je vybrána submatice z "`A`" tvořena 1. a 3. řádkem a 5. a 2. sloupcem. Je možno taky provádět přiřazení do takto vybraných prvků při zachování správných rozměrů:

```
A([1 3],[5 2])=eye(2)
```

Pokud chceme vybrat celý řádek nebo sloupec použijeme symbol ':', např. "`B(3,:)`" je 3. řádek matice "`B`". Příkazem "`B(3,:)=[]`" vypustíme z matice B tento řádek. Zajímavé je, že nefunguje přiřazení

```
o=[];
B(3,:)=o
```

Matic stejných rozměrů lze sčítat a odčítat $(+, -)$, matice vhodných rozměrů se dají násobit (*). Je také možné násobit konstantou nebo přičíst konstantu k matici, pak se přičte ke všem prvkům. Dělení jsou v MATLABu dvě – pravé a levé: '\', '/', operátor " .' " se používá pro transpozici matic. Podrobněji všechny operátory popisuje Kapitola 2. Na matice lze aplikovat taky všechny běžné funkce ("`sin`", "`cos`",...), které se provádějí člen po členu.

1.7 Tipy a triky

Pokud máme matici "`A`" typu $m \times n$, pak pokus o určení hodnoty "`A(k,l)`", kde `k`> m nebo `l`> n, skončí chybou. Ovšem příkaz "`A(k,l)=1`" přiřazení provede, přičemž chybějící členy jsou doplněny nulami.

Náhodnou matici s celočíselnými členy v daném rozmezí lze vytvořit vhodnou kombinací funkce "`rand`" a některé zaokrouhlovací funkce. Například náhodnou matici řádu 4 s prvky mezi -5 a 5 získáme příkazem

```
A=round(10*rand(4))-5
```

Kapitola 2

Maticové operace

2.1 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. Avšak pokud matice `A` má komplexní prvky, je třeba dát pozor. Příkazem

```
X=A'
```

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

```
A=[1 1+i; 2-3*i i]
X=A'
```

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

```
X=A.'
```

2.2 Sčítání a odčítání matic

Znaménka "+" a "−" označují sčítání a odčítání matic. Matice musí mít shodné dimenze. Operace se provádějí po složkách, tj. např. pro

```
A=[1 5; 2 3];
B=[0 1; -3 2];
X=A+B
```

je $x_{ij} = a_{ij} + b_{ij}$.
Lze také příčítat konstantu k matici a odčítat konstantu od matice. V tom případě se pak konstanta přičte nebo odečte od každého prvku matice. Např. pro

```
c=2;
```

`X=A+c`

je $x_{ij} = a_{ij} + c$, pro

`X=c-B`

je $x_{ij} = c - b_{ij}$.

2.3 Maticové násobení

Symbol " * " označuje násobení matic. Operace je definována, pokud vnitřní rozměry dvou operandů jsou stejné. Součin `X=A*B` se vykoná, je-li druhý rozměr `A` stejný jako první rozměr `B`, tj. pokud `A` je typu $n \times \underline{k}$, `B` musí být typu $\underline{k} \times m$, výsledná matice `X` bude typu $n \times m$ a platí $x_{ij} = \sum_{r=1}^{k} a_{ir} b_{rj}$.

Definujme matice `A` a `B`

```
A=[1 5 0; -1 2 3];
```

```
B=[0 1 1; 1 -3 2; -1 4 2; 1 0 3];
```

Matice `A` je tedy typu 2×3, matice `B` je typu 4×3. Můžeme vyzkoušet, že součin

`X=A*B`

není definován, protože nesouhlasí uvedené rozměry. Pokud však v součinu použijeme transponovanou matici `B'`, která je typu 3×4,

`X=A*B'`

operace je platná a výsledná matice `X` je typu 2×4.

Matice lze také násobit konstantou. V tomto případě se vynásobí každý prvek matice danou konstantou. Např. pro

```
c=2;
```

`X=A*c`

je $x_{ij} = a_{ij} * c$, pro

`X=c*B`

je $x_{ij} = c * b_{ij}$.

2.4 Maticové dělení

V MATLABu existují dva symboly pro dělení matic, "\" a "/". Obecně platí, že

`X=A\B`	je řešením $A * X = B$
`X=B/A`	je řešením $X * A = B$.

Je-li `A` regulární čtvercová matice, potom `X=A\B` resp. `X=B/A` formálně odpovídají levostrannému resp. pravostrannému násobení matice `B` maticí inverzní k matici `A`; tj.

`inv(A)*B` resp. `B*inv(A)`, ale výsledek je získán přímo (bez výpočtu inverze).
Je-li `A` obecně typu $\underline{k} \times n$, `B` musí být typu $\underline{k} \times m$, výsledná matice `X=A\B` bude typu $n \times m$. Pravostranné dělení `X=B/A`, je definováno pomocí levostranného dělení jako `X=B/A =(A'\B')'`.
Definujme matici `A` a vektor `b`

```
A=[1 5 0; -1 2 3; 1 2 1];
b=[1 3 2]';
```

Matice `A` je typu 3×3, vektor `b` je typu 3×1. Můžeme vyzkoušet, že podíl

```
x=A\b
```

je stejný jako

```
x=inv(A)*b
```

Při dělení matice konstantou se dělí každý prvek matice danou konstantou a výsledek pravostranného i levostranného dělení je stejný. Vyzkoušejte

```
c=2;
x=c\A
x=A/c
```

2.5 Umocňování matic

Pro maticové umocňování se používá symbol " ^ ". Předpokládejme, že `A` je čtvercová matice a `p` je skalár. Při umocňování mohou nastat tyto případy:

1. `X=A^p`
   (a) $p > 0$ celé číslo $\Rightarrow X = \underbrace{A * A * \cdots * A}_{p}$
   (b) $p = 0 \Rightarrow X = I$ ...jednotková matice (= `eye(size(A))`)
   (c) $p < 0$ celé číslo $\Rightarrow X = \underbrace{A^{-1} * A^{-1} * \cdots * A^{-1}}_{p}$
   (d) p není celé číslo $\Rightarrow$ uvažujme diagonální matici $D = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$, kde $\lambda_1, \ldots, \lambda_n$ 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 jakási motivace pro výpočet mocniny v tomto případě. Dá se 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^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, MATLAB zahlásí chybu. Vyzkoušejte např.

```
A=[1 3; 3 2 ]
p=0.5
[V,D]=eig(A)
X=A^p
X=V*D^p/V
Y=p^A
Y=V*p^D/V
```

2.6 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 ” . ”. Kromě operací sčítání a odčítání, které se automaticky provádějí po složkách a tudíž zde tečka nemá smysl, lze předřadit tečku před jakýkoliv maticový operátor. Uvažujme tedy libovolný maticový operátor $\circ \in \{*, /, \backslash, \hat{}\}$. Pro provádění operací po složkách musí být obě matice shodné dimenze, výsledek bude mít tutéž dimenzi. Obecně tedy pro prvky matice `X=A.`$\circ$`B` platí $x_{ij} = a_{ij} \circ b_{ij}$.
Vyzkoušejte rozdíly mezi maticovými operacemi a operacemi po složkách, např. pro násobení

```
A=[1 2;3 4]
B=[2 4;6 8]
X=A*B
X=A.*B
```

pro dělení

```
X=B/A
X=B./A
```

a pro umocňování

`X=B^A` (neexistuje)

```
X=B.^A
```

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. Vyzkoušejte si, že např. při dělení dostaneme tentýž výsledek

```
c=2
X=A/c
X=A./c
```

Naopak při umocňování je tečka podstatná

```
X=A^c
X=A.^c
X=c^A
X=c.^A
```

2.7 Logické operace

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

2.8 Smíšené operace

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

1. umocňování: `^ .^`
2. násobení, dělení: `* .* \ .\ / ./`
3. sečítání, odčítání: `+ -`
4. relační operátory: `== ~= < <= > >=`
5. negace: `~`
6. logické spojky 'a' a 'nebo': `& |`

Pomocí kulatých závorek můžeme řídit prioritu výše uvedených operátorů.

2.9 Příklady

Vyzkoušejte následující příkazy:

```
x=0:0.01:2*pi;
y=sin(x.^2).*cos(3*x);
plot(x,y)
```

Kapitola 3

Příkazy MATLABu

3.1 Obecná pravidla

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 ". . ." a 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 funkci MATLABu. Toto volání má v obecném případě tvar:

```
[v1,v2,...,vm]=jmeno_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. Vyzkoušejte například:

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

Vykřičník na začátku příkazu způsobí vykonání příkazu operačního systému, např. příkazem "`!netscape`" spustíme známý internetový prohlížeč.

Symbol '%' na příkazové řádce způsobí ignorování zbytku řádky (komentář).

3.2 Některé speciální výrazy a funkce

Zvláštní postavení mezi proměnnými má proměnná "`ans`" (z answer), do které se automaticky přiřazují hodnoty, jež nebyly přiřazeny explicitně – stačí zadat např. "`pi`" . Dále "`i`" a "`j`" jsou imaginární jednotky pro práci s komplexními čísly. Další speci-

ální 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 (+ nebo −). Hodnotu "`NaN`" (Not a Number) dostaneme jako výsledek neurčitého výrazu – např. 0/0.

Způsob výpisu hodnot můžeme ovlivnit příkazem "`format`":

`format long`	výpis na plný počet desetinných míst
`format short`	výpis na omezený počet desetinných míst (implicitní)
`format hex`	hexadecimální výpis
`format rat`	čísla jsou aproximovaná zlomky
`format compact`	potlačí se vynechávání řádku při výpisu
`format loose`	zapne se vynechávání řádku při výpisu

3.3 Manipulace s maticemi

Základní příkazy pro práci s maticemi byly uvedeny v první kapitole. Pro vytváření vektoru, jehož prvky tvoří konečnou aritmetickou posloupnost je možné kromě znaku ':' použít také příkaz "`linspace`":

`x=linspace(a,b,n)`,

kde "`a`", "`b`" jsou první a poslední prvek vektoru "`x`" a "`n`" je počet prvků. Třetí parametr je nepovinný a pokud není uveden, bere se roven 100. Podobně je taky možné definovat vektor s desítkovou logaritmickou škálou:

`x=logspace(a,b,n)`,

který je ekvivalentní příkazu

`x=10.^linspace(a,b,n)`.

Zde je ovšem implicitně n=50.

Příkazem "`reshape`" je možné přeskládat matici do jiného typu. Zadáním

```
A=rand(10); B=reshape(A,5,20);
```

vytvoříme z prvků matice "`A`" matici "`B`" o 5 řádcích a 20 sloupcích, přičemž prvky se při přeskládání berou po sloupcích. Pokud bychom chtěli matici "`A`" přeskládat po řádcích, museli bychom nejdřív matici "`A`" transponovat a pak opět transponovat výsledek:

```
B=reshape(A',5,20)'
```

Překlopit matici, tj. obrátit pořadí řádku nebo sloupců je možné provést příkazem "`flipud`" resp. "`fliplr`". Také je možné matici „*otočit*“ o 90 stupňů příkazem "`rot90`", přičemž další nepovinný parametr udává, kolikrát se má rotace provést.

Jedním z dalších triků, které využívají dvojtečku, je `A(:)`. `A(:)` na pravé straně přiřazovacího příkazu označuje všechny sloupce matice `A` přeskládané do dlouhého sloupcového vektoru. Vyzkoušejte

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

Pokud již matice `A` existuje, lze `A(:)` použít i na levé straně přiřazovacího příkazu ke

změně tvaru nebo velikosti matice. Potom `A(:)` označuje matici A uspořádanou pouze v rámci daného přiřazovacího příkazu do sloupcového vektoru (se sloupci A pod sebou). Např. výše uvedená matice A má tři řádky a dva sloupce, takže

```
A(:) = 11:16
```

změní šestiprvkový řádkový vektor na matici typu 3×2. Tato operace je zahrnuta ve funkci "`reshape`".

3.4 Další funkce pro práci s maticemi

Všechny operátory pro práci s maticemi uvedené v předchozí kapitole mají své funkci 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`	právě maticové dělení	/
`ldivide`	levé dělení po prvcích	.\
`rdivide`	právě dělení po prvcích	./

Dvojí úlohu má funkce "`diag`". Jeho aplikaci na vektor získáme diagonální matici s argumentem na hlavní diagonále. Pokud ji použijeme na matici, funkce "`diag`" vybere z matice hlavní diagonálu a umístí ji do vektoru. Pokud chceme pracovat s jinou diagonálou než s hlavní, můžeme použít jako druhý parametr funkce "`diag`" číslo diagonály, přičemž kladná čísla se použijí nad hlavní diagonálou a záporná pod ní. Příklad:

```
A=rand(5)
x=diag(A)
B=diag(x,2)
C=diag(pi,-4)
d=diag(diag(A))
```

Pomocí funkci "`tril`" a "`triu`" vyrobíme z daně matice dolní nebo horní trojúhelníkovou matici, přičemž je možné podobně jako u "`diag`" použít další nepovinný parametr.

Příkaz "`max`" najde maximální prvek ve vektoru, přičemž pokud na výstupu uvedeme i druhý výstupní parametr, uloží se do něj index tohoto prvku. Při použití příkazu "`max`" na matici se hledají maximální prvky v jednotlivých sloupcích. Podobně funguje příkaz "`min`":

```
x=rand(1,6)
```

```
M=max(x)
[m,k]=min(x)
A=rand(4)
[m,k]=min(A)
```

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

K setřídění vektoru (vzestupné) se používá příkaz "`sort`", u komplexních hodnot se opět provádí třídění podle absolutních hodnot. Do druhého nepovinného výstupního parametru je umístěna třídicí permutace indexu. Tj. pokud zadáme "`[y,ind]=sort(x)`" pak platí `y=x(ind)`. Při použití příkazu "`sort`" na matice se třídí jednotlivé sloupce.

Příkazy "`sum`" a "`prod`" sečtou nebo vynásobí všechny prvky vektoru. Aplikovaný na matici provedou příslušnou operaci po jednotlivých sloupcích. Např. faktoriál čísla n zjistíme snadno pomocí příkazu "`f=prod(1:n)`", přičemž příkaz funguje správné i pro "`n`" rovno 0, protože součin přes prázdnou matici je roven 1.

Pokud bychom chtěli zjistit součet všech prvků matice, musíme příkaz "`sum`" použít dvakrát:

```
sum(sum(A))
```

Příkazem "`size`" zjistíme rozměry matice, přičemž je možné jej použít ve formě "`s=size(a)`" nebo "`[m,n]=size(a)`". Příkaz "`length`" dává délku vektoru, přičemž pokud jej aplikujeme na matici, dostaneme maximální z rozměrů, tj. "`length(A)`" dává stejný výsledek jako "`max(size(A))`".

3.5 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:

`det`	determinant matice
`rank`	hodnost matice
`norm`	maticová nebo vektorová norma
`trace`	stopa matice (součet diagonálních prvků)
`inv`	inverzní matice
`pinv`	pseudoinverzní matice
`lu`	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.

3.6 Polynomy

Polynomy v Matlabu se zadávají pomocí vektoru jejich koeficientů od nejvyšší mocniny. Takže třeba polynom $p(x) = 2x^4 - 3x^3 + 5x - 4$ zadáme jako

```
p=[2,-3,0,5,-4];
```

Pro vyhodnocení polynomu v daném bodě používáme funkci "`polyval`":

```
y=polyval(p,x);
```

V proměnné "`x`" může být samozřejmě obsažen vektor, pak se polynom vyčíslí pro všechny hodnoty vektoru "`x`". Pokud bychom chtěli aplikovat polynom "`p`" na matici "`A`", použijeme funkci "`polyvalm`":

```
y=polyvalm(p,A);
```

Kořeny polynomu lze najít pomocí funkce "`roots`". Tak např. polynom $p(x) = x^2 - 1$, který je v Matlabu reprezentován vektorem `[1 0 -1]`, má kořeny -1, 1:

```
p=[1 0 -1];
k=roots(p);
```

Naopak sestrojit polynom, který má dané kořeny můžeme pomocí funkce "`poly`". Takže původní polynom $p(x)$ sestrojíme v Matlabu příkazem:

```
p=poly([-1,1]);
```

Funkce "`polyder`" slouží k derivování polynomu, funkce "`conv`" k násobení polynomu a funkce "`deconv`" k dělení polynomu se zbytkem.

3.7 Tipy a triky

Pomocí funkce "`sort`" lze náhodně zpermutovat danou konečnou posloupnost čísel. Chtějme například vytvořit náhodnou permutaci čísel od 1 do 10. Vytvořme nejprve pomocný náhodný vektor daného rozměru:

`p=rand(1,10)`, pak vektor "`p`" setřídíme:

```
[p,I]=sort(p);
```

a v proměnné "`I`" máme hledanou náhodnou permutaci. V případě, že bychom chtěli zpermutovat prvky nějakého vektoru "`x`", stačí zadat

`x=x(I)`,

kde "`I`" je třídící permutace pro indexy náhodného vektoru stejného rozměru jako "`x`".

Kapitola 4

Práce se soubory

4.1 Záznam práce

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

Ukládání do souboru s jiným jménem dosáhneme pomocí příkazu "`diary jmeno_souboru`" nebo "`diary('jmeno_souboru')`", kde "`jmeno_souboru`" je libovolně jméno souboru. Další použití příkazu "`diary`" se pak bude vztahovat k uvedenému souboru.

4.2 Ukládání a načítání proměnných

Pro uložení proměnných do souboru slouží příkaz "`save`". Jeho použití bez parametrů způsobí uložení všech definovaných proměnných do souboru s názvem "`matlab.mat`". Pro uložení jen vybraných proměnných do námi zvoleného souboru žádáme jako první parametr jméno souboru a jako další parametry názvy proměnných, které chceme uložit. Tedy příkaz

```
save data
```

uloží všechny definované proměnné do souboru data a příkaz

```
save data1 a b c
```

uloží do souboru "`data1.mat`" proměnné s názvy "`a`", "`b`" a "`c`".

Stejný výsledek dá příkaz

```
save('data1','a','b','c').
```

Uložené soubory mají automaticky příponu `mat`. Je možné zadat i příponu jinou.

Načíst data z uloženého souboru můžeme příkazem "`load`". Použijeme-li ho bez parametrů, načtou se všechny uložené proměnné ze souboru "`matlab.mat`". Jinak je

možné použít příkaz

`load`

podobně jako příkaz "`save`", tedy např.

`load data1`

nebo `load('data1')`

načte všechny proměnné uložené v souboru "`data1.mat`". Pokud soubor s uloženými daty má jinou příponu než "`mat`", musí se při načítání jeho obsahu použít parametr "`-MAT`":

`load data1.dat -MAT` (lze použít i malá písmena: "`-mat`").

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

`save data2.dat X -ASCII`

V tomto souboru ovšem není uloženo jméno 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 stejné jméno, jako původní soubor (bez přípony). Takové soubory je možné vytvářet i ručně, případně jako výsledek práce jiných programu. Je ovšem nutno mít na paměti, že data v nich obsažena musí mít tvar matice, tj. každý řádek musí mít stejný počet sloupců.

4.3 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 taky nápovědu, která se vypisuje příkazem "`help`".

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

```
which poly
which eig
```

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, aby je bylo možné používat v MATLABu.

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

4.4 Cesta k souborům

Programy a funkce spouštěné při práci v MATLABU se hledají v adresářích, do nichž je nastavená cesta – tzv. "`matlabpath`". Její obsah zjistíme příkazem "`path`" nebo

"`matlabpath`". Pokud chceme do této cesty přidat nějaký další adresář, můžeme to udělat pomocí příkazu "`path`":

```
path(path,'dalsi_adresar')
```

nebo "`addpath`:

```
addpath dalsi_adresar
```

Proměnná "`dalsi_adresar`" musí obsahovat řetězec s daným adresářem. Konkrétní tvar závisí na použitém operačním systému, např. v MS Windows bychom mohli zadat

```
path(path,'d:\user\matlab')
```

či

```
addpath d:\user\matlab
```

a v UNIXu

```
addpath /home/user/matlab
```

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

4.5 Další příkazy pro práci se soubory

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

Pracovní adresář zjistíme příkazem "`pwd`", který vrací jako hodnotu řetězec s tímto adresářem. Je tedy možné jej přidat do cesty pro prohledávání:

```
d=pwd;
addpath(pwd)
```

Změnit pracovní adresář můžeme příkazem "`cd`", např.:

```
cd matlab/data
```

Pro výpis obsahu zvoleného souboru můžeme použít příkaz "`type`", přičemž výpis dlouhého souboru se zastavuje po stránkách, pokud předtím žádáme příkaz "`more on`". Vypnutí stránkování provedeme příkazem "`more off`".

Kopírovat libovolně soubory v rámci MATLABu lze pomocí příkazu "`copyfile`", mazat je můžeme příkazem "`delete`". Příkaz "`lookfor`" prohledává nápovědu pro zadaný výraz:

```
lookfor polynomial
```

Editor pro psaní souboru lze vyvolat příkazem "`edit`", jeho nastavení závisí na operačním systému. V některých verzích MATLABu lze tento editor vyvolat z menu.

Kapitola 5

Logické operace

5.1 Relační operátory

Relační operátory slouží k porovnávání proměnných. Jedna 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. Vyzkoušejte třeba:

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

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

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. Produktem operace je matice odpovídajícího typu s jedničkami a nulami podle výsledků dané operace. Jednotlivé operace se provádějí po prvcích. Přitom nula je brána jako nepravdivá hodnota a nenulové číslo (včetně "Inf" a "NaN") se bere jako pravdivá hodnota. Př.:

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

Místo "`A&B`" můžeme psát "`and(A,B)`", místo "`A|B`" "`or(A,B)`" a místo "`~A`" "`not(A)`".

5.3 Logické funkce

Logické funkce vrací jako hodnoty 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, aplikuje se funkce "`all`" na jednotlivé sloupce matice "`A`" a výsledkem je řádkový vektor obsahující výsledky 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. Příznak logické matice mají všechny výsledky relačních a logických operátorů a logických funkci.

`isstr(A)`
- vrací 1, pokud je "`A`" textová matice

`isnumeric(A)`
- vrací 1, pokud je "`A`" číselná matice (ne textová)

5.4 Funkce `find` a `exist`

Další užitečnou funkci je funkce "`find`", která vrací indexy nenulových prvků vektoru nebo matice. Použití je následující: Pro vektor "`x`" příkaz

```
I=find(x)
```

umístí do proměnné "`I`" indexy nenulových prvků vektoru "`x`". Pro matici "`A`" příkaz

```
I=find(A)
```

najde indexy nenulových prvků bráno po sloupcích. Příkaz

```
[I,J]=find(A)
```

umístí do "`I`" řádkové indexy a do "`J`" sloupcové indexy nenulových prvků matice "`A`". Příkaz

```
[I,J,H]=find(A)
```

umístí navíc do proměnné "`H`" příslušné nenulové prvky. Pokud chceme zjistit hodnoty nenulových prvků i pro vektor, musíme příkaz použít stejně jako pro matici, tedy

```
[I,J,H]=find(x)
```

Typické použití příkazu `find` je například

```
[I,J]=find(A>0)
```

kterým najdeme indexy kladných prvků matice "`A`".

Funkcí "`exist`" testujeme existenci a typy objektů MATLABu. Výsledkem příkazu "`ex=exist('A')`" jsou následující možnosti proměnné "`ex`":

0 objekt se jménem "`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
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 (viz. "`help pcode`")
7 "`A`" je adresář

Při kolizi jmen (existují různé objekty téhož jména) 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 jen, zda "`A`" je proměnná
`exist('A','builtin')`	testuje jen, zda "`A`" je interní funkce
`exist('A','file')`	testuje jen, zda "`A`" je soubor
`exist('A','dir')`	testuje jen, zda "`A`" je adresář

Vrácena hodnota je pak 1 (existuje) nebo 0 (neexistuje).

5.5 Příklady

1. Utvořme matici "`A`":

```
A=round(20*rand(5)-10)
```

tj. "`A`" je náhodná matice řádu 5 s celočíselnými prvky mezi -10 a 10. Naším úkolem je vytvořit matici "`B`" mající hodnotu 10 na místech, kde má "`A`" kladné prvky, na ostatních pozicích má matice "`B`" stejné prvky jako "`A`".
Příkaz "`(A>0)`" umístí na hledaná místa 1 jinam 0. To je možné využít následovně:

```
B=A.*(A<=0)+10*(A>0)
```

2. Další možností je využít příkaz "`find`". Zadáním

```
I=find(A>0)
```

najdeme příslušné indexy. Chybou by teď bylo

`A(I)=10`
poněvadž tím by se matice "`A`" předefinovala. Správě řešení je
`B=A; B(I)=10;`
Celé je to ovšem možné zkrátit:
`B=A; B(find(A>0))=10`

5.6 Tipy a triky

Počet nulových prvků libovolně matice "`X`" můžeme zjistit např. jako:
`sum(sum(~X))`
Počet nenulových prvků matice "`X`" bychom mohli získat příkazem
`prod(size(X))-sum(sum(~X))`
ale efektivněji vypadá
`length(find(X))`

Kapitola 6

Textové řetězce

6.1 Vytváření řetězců

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

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

Řetězce je možno skládat do matice:

```
S1=['1.radek';'2.radek';'3.radek';'4.radek']
```

Při tomto způsobu vytváření musí ovšem všechny řádky mít stejnou délku. Lepší metodou je použití příkazu "`char`" nebo "`str2mat`":

```
S2=char('1.radek','2.radek','3.radek','posledni radek')
S3=str2mat('1.radek','2.radek','3.radek','posledni radek')
```

Takto vytvořené matice mají příznak textových polí, jak zjistíme pomocí "`whos`" nebo "`isstr`".

Pokud chceme složit více řetězců do jednoho řádku, použijeme k tomu stejný způsob, jakým definujeme číselné matice, tj. všechny jednotlivé části uzavřeme do hranatých závorek

```
s3=['abcdef','123','45']
```

Tento způsob má uplatnění zejména v případech, kdy některý z řetězců získáme jako výsledek funkce:

```
s4=['Číslo pí je rovno přibližně ',num2str(pi)]
```

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')
s=char(32:64)
```

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. Příkazem "`isletter`" se testuje, zda daný znak je písmeno ('A' - 'Z', 'a' - 'z'), příkaz "`isspace`" slouží k testování, zda daný znak je mezerového typu (mezera - kód 32, tabulátor apod.)

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 samé mezery se dá vytvořit funkcí "`blanks`", funkce "`deblank`" naopak odstraní mezery z konce textového řetězce. Zkuste:

```
s1=['123',blanks(3),'abc',blanks(2)]
d1=double(s1)
s2=deblank(s1)
d2=double(s2)
```

6.3 Funkce pro manipulaci s řetězci

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

`strcat` – spojuje řetězce, které jsou na vstupu, do jednoho.

`strvcat` – umísťuje řetězce do matice jako řádky, přitom je doplňuje mezerami na stejnou délku.

`strcmp` – porovnává vstupní argumenty, dává hodnotu 1, pokud jsou to stejné řetězce, v opačném případě vrací 0.

`strncmp` – funguje podobně jako "`strcmp`" s tím rozdílem, že porovnává prvních `n` prvků vstupních řetězců. Číslo `n` je třetím vstupním parametrem funkce "`strncmp`".

`strcmpi` – funguje podobně jako "`strcmp`", ale nerozlišuje malá a velká písmena.

`strncmpi` – funguje podobně jako "`strcmpi`", porovnává prvních `n` prvků vstupních řetězců.

`findstr` – má na vstupu dva textové řetězce, hledá první řetězec ve druhém. Jako výstup dává indexové pozice, kde začíná první řetězec ve druhém. Pokud se v něm nevyskytuje, výstupem je prázdná matice.

`isstrprop` – vstupem je řetězec a typ, vrací jedničky na pozicích, na kterých se v řetězci vyskytují výrazy daného typu, jinde vrací nuly. Jako typ se zadává např. "`lower`" (malé písmeno), "`digit`" (číslice) nebo "`wspace`" (mezera) atd.

`strrep` – vstupem jsou tři řetězce. Funkce prohledává první z nich a pokud v něm najde druhý vstupní řetězec, nahradí jej třetím. Výsledek je vrácen jako výstup funkce.

`strtok` – najde úvodní část vstupního řetězce ukončenou mezerou a vrátí ji na výstup. Případné počáteční mezery jsou vypuštěny. V případě, že jsou požadovány dvě výstupní hodnoty, je ve druhé obsažen zbytek vstupního řetězce.

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

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

6.4 Funkce `eval` a `feval`

Funkce "`eval`" zpracuje vstupní řetězec stejně, jako by byl napsán ve formě příkazového řádku. Je užitečná zejména v případě, když potřebujeme spočítat hodnotu nějakého výrazu pro různé hodnoty parametrů v něm obsažené. Zkuste třeba:

```
s='sin(2*pi*a*x)';
x=0:0.01:1;
a=1;
y=eval(s);
plot(x,y)
a=2;
y=eval(s);
plot(x,y)
a=3;
y=eval(s);
plot(x,y)
```

Funkce "`feval`" má dva vstupní parametry, prvním z nich je jméno funkce, která se má zavolat, druhý parametr obsahuje hodnotu, která se předá volané funkci jako vstupní parametr. Následující tři příkazy tedy dávají stejný výsledek:

```
y=sin(x);
y=feval('sin',x);
y=eval('sin(x)');
```

Pokud volaná funkce má více parametrů, jsou uvedeny jako další parametry funkce "`feval`":

```
y=feval('diag',x,1);
```

Funkce "`feval`" se hodí zejména v případě, kdy voláme různé funkce, jejichž jména jsou obsažena v textovém řetězci.

Kapitola 7

Vyhodnocování výrazů

7.1 Výraz jako textový řetězec

Chceme-li vyhodnotit výraz zadaný jako textový řetězec, použijeme funkce "`eval`". Ta je podrobněji popsána v minulé kapitole. Funkce "`eval`" zpracuje vstupní řetězec stejně, jako by byl napsán ve formě příkazového řádku. Takže chceme-li např. vyhodnotit výraz $x^2 + xy + 1$ v bodech $x = 2,\ y = 3$, zadáme to takto:

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

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. Tudíž vyhodnocení 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 příkazem "`vectorize`". Zadání v MATLABu by tedy vypadalo nějak takto:

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

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

7.2 Symbolický výraz

MATLAB již 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 se dá zjistit např. příkazem `"ver"`, který vypíše verzi Matlabu a všechny nainstalované knihovny. Příkazem `"sym"` můžeme libovolný textový řetězec převést na symbolický výraz. Vyhodnocení takto definovaného výrazu se provede příkazem `"subs"`, funguje i `"eval"`. Příklad z minulé podkapitoly by se tedy pomocí `"subs"` řešil takto:

```
f=sym('x^2+x*y+1');
z=subs(f,{'x','y'},{2,3});
```

Všimněme si, že jako argument funkce `"subs"` byla zadána nejen funkce f, ale také ve složených závorkách proměnné, aby Matlab věděl, v jakém pořadí a kam co dosadit.
V případě, že proměnné x a y jsou vektory, resp. matice, syntaxe je podobná. Takže příklad z minulé podkapitoly by se řešil takto:

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

V některých starších verzích Matlabu tato syntaxe nefunguje a je opět nutné dodat tečky před příslušné operace příkazem `vectorize`, který funguje i pro symbolické výrazy.
Počítání se symbolickými výrazy má mnoho výhod. Můžeme derivovat výraz (příkaz `"diff"`), integrovat (příkaz `"int"`), zjednodušovat výrazy (příkaz `"simplify"`) a mnoho dalších. Jako zajímavost uveďme příkaz `"latex"`, který převede daný výraz do jeho LaTeXovské reprezentace.

7.3 Výraz jako INLINE funkce

Poslední možností, jak vyhodnotit daný výraz, je definovat ho jako tzv. "INLINE" funkci příkazem `"inline"`. Samotné vyhodnocení se pak provede pouhým zapsáním dané hodnoty do kulatých závorek za funkci. Příklad z minulé podkapitoly by se tedy pomocí `"inline"` řešil takto:

```
f=inline('x^2+x*y+1','x','y');
z=f(2,3);
```

Všimněte si prosím, jakým způsobem se zadávají jednotlivé proměnné a jejich pořadí.
V případě, že proměnné x a y jsou vektory, resp. matice, syntaxe je podobná. Nesmíme však zapomenout dodat tečky před příslušné operace, aby se prováděly po složkách. Takže příklad z minulé podkapitoly by se řešil takto:

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

Kapitola 8

Práce s grafikou

8.1 Grafy funkcí

Základním příkazem pro kreslení grafů funkcí jedné proměnné je příkaz `plot`. Jeho použití nejlépe vysvětlíme na příkladu. Potřebujeme třeba graf funkce $\sin(2x) + \cos(x)$ na intervalu $< 0, \pi >$. Nejprve určíme hodnoty `x`, v nichž se má funkce vypočítat, např. `x=linspace(0,pi);`, pak spočítáme funkční hodnoty `y=sin(2*x)+cos(x);`. Nakreslení grafu dosáhneme příkazem `plot(x,y)`.

Nakreslená funkce vypadá hezky hladce, ve skutečnosti ovšem se jedná o lomenou čáru, protože jednotlivé body jsou spojeny úsečkami. To by bylo dobře vidět, pokud bychom na začátku zadali např. `x=linspace(0,pi,20);`.

Příkaz `plot`, může mít jako další parametr textový řetězec, s jehož pomocí můžeme ovlivňovat barvu a styl čáry a znak pro kreslení jednotlivých bodů, které jsou spojovány úsečkami. Znaky pro barvu jsou:

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

Styly pro čáru jsou:

`-`	plnou čarou
`--`	čárkovaně
`:`	tečkovaně
`-.`	čerchovaně

Jednotlivé znaky mohou být nastaveny pomocí:

`.` 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

Všechny tyto parametry lze kombinovat. Položme například `x=[1,2,3];` a `y=[1,3,2];`. Příkazem `plot(x,y,'b--*')` nakreslíme čárkovanou lomenou čáru složenou ze dvou úseček, kde zadané body budou zobrazeny hvězdičkami, vše bude modrou barvou. Zadáním `plot(x,y,'+r')` se nakreslí jen zadané body bez spojování úsečkou. V případě, že explicitně nezadáme barvu, je volena automaticky.

Příkaz `plot` umožňuje kreslit více grafů najednou, stačí zadat je jako další jeho parametry: `plot(x,y1,x,y2,x,y3)` nakreslí tři grafy uložené v proměnných `y1`, `y2` a `y3`, všechny pro stejný definiční obor daný proměnnou `x`. Pro každý graf můžeme uvést parametry pro styl kreslení, takže pokud bychom chtěli ve výše uvedeném příkladě chtěli body nakreslit jinou barvou než spojovací čáry, mohli bychom toho dosáhnout příkazem `plot(x,y,'b--',x,y,'r*')`.

Pokud chceme, aby se další graf kreslil do obrázku, který jsem již vytvořili, musíme zadat příkaz `hold on`, tato vlastnost se vypne příkazem `hold off`. Příkazem `grid on` zapneme zobrazení mříže pro lepší orientaci v grafu, vypnutí opět zajistí příkaz `grid off`.

Do obrázku je možné vložit text pomocí příkazu `text`, jehož první dva parametry udávají místo v obrázku, kam se má text umístit, třetí proměnná je vlastní text, který do obrázku vkládáme. Popis os můžeme udělat pomocí příkazů `xlabel` a `ylabel`, jejichž parametry jsou textové řetězce popisující osy. Nadpis k obrázku vytvoříme příkazem `title`, jehož parametrem je opět textový řetězec s textem nadpisu.

Složený obrázek obsahující několik grafů uspořádaných do tabulky lze vytvořit příkazem `subplot`. Tento příkaz má tři parametry – první dva dávají rozměry tabulky, třetí je místo, kam se umístí obrázek při následujícím příkazu `plot`, přičemž pozice místa se počítá postupně po řádcích. Můžeme si vyzkoušet

```
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)
subplot(2,2,2)
plot(x,y2)
subplot(2,2,3)
plot(x,y3)
```

```
subplot(2,2,4)
plot(x,y4)
```

Pro kreslení grafů funkcí dvou proměnných můžeme použít příkazy `mesh`, `surf` nebo `waterfall`. V základním tvaru mají tři parametry obsahující postupně první a druhou proměnnou jako vektory a funkční hodnoty uloženy v matici. Je možné první dvě proměnné vynechat, v tom případě je graf indexován rozměry matice. Pro snadnější vytváření dvojrozměrných grafů slouží funkce `meshgrid`, která vytvoří z jednorozměrných vektorů dvourozměrné sítě vhodné pro definování grafu dané funkce. Takže pokud bychom chtěli například nakreslit graf funkce $\sin(\pi(x+y))$, můžeme postupovat takto:

```
x=0:0.05:1;
y=0:0.05:1;
[X,Y]=meshgrid(x,y);
Z=sin(pi*(X+Y));
mesh(X,Y,Z)
```

Všimněme si, že vykreslený graf obsahuje celou škálu barev podle hodnot v jednotlivých bodech. Barevnou stupnici tohoto škálování lze zobrazit příkazem `colorbar`.

Úhel pohledu na trojrozměrný obrázek lze nastavit příkazem `view`, který má dva parametry. První parametr udává azimut v rovině nezávislých proměnných, druhý parametr pak tzv. elevaci, což je úhel směru pohledu s rovinou nezávislých proměnných. Implicitně je nastaven azimut -37.5° a elevace 30°.

V MATLABu od verze 5.3 je možné nastavovat úhel pohledu pomocí myši přímo v obrázku volbou v panelu nástrojů, který je součástí obrázku. Podobně i vlastnosti dvourozměrných grafů se dají nastavovat výběrem položky v menu na obrázku.

Pro vykreslování křivek v 3-D prostoru se používá příkaz `plot3`, jehož vstupními argumenty jsou vektory x, y a z stejných rozměrů. Pokud by to byly matice, vykreslí se křivky postupně pro sloupce těchto matic. Vyzkoušejte např.

```
t=0:pi/50:10*pi;
plot3(sin(t),cos(t),t);
```

8.2 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`, nastavování se dělá pomocí funkce `set`. Předtím je ale potřeba, aby byl definován ukazatel na daný grafický objekt, tzv. *handle*. To se zajistí tak, že daný grafický příkaz se provede ve formě přiřazení do určené proměnné. Ukážeme si to na příkladu. Nejprve definujme proměnné pro nakreslení grafu:

```
x=linspace(0,pi);
y=sin(x);
```

Pak nakreslíme uvedenou sinusovku spolu s přiřazením do proměnné:

```
p=plot(x,y)
```

Protože příkaz není ukončen středníkem, vypíše se hodnota definované proměnné `p`, ovšem vlastní hodnota není důležitá. Zadáním příkazu

`get(p)`

získáme výpis všech vlastností nakresleného grafu vypadající přibližně takto:

```
Color = [0 0 1]
EraseMode = normal
LineStyle = -
LineWidth = [0.5]
Marker = none
MarkerSize = [6]
MarkerEdgeColor = auto
MarkerFaceColor = none
XData = [ (1 by 100) double array]
YData = [ (1 by 100) double array]
ZData = []

ButtonDownFcn =
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [2.00024]
Selected = off
SelectionHighlight = on
Tag =
Type = line
UIContextMenu = []
UserData = []
Visible = on
```

Kapitola 9

Programování v MATLABu

9.1 Dávkové soubory (skripty) a funkce

Programy v MATLABu, které si může uživatel běžné 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 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 spuštění programu MATLAB, pokud existuje v adresáři, v němž MATLAB pouštíme. 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('Vás pracovní adresář je:')
disp(pwd)
```

Funkce musí začínat hlavičkou, která má tvar:

```
function [výst.parametry]=jmeno_funkce(vst.parametry)
```

Jméno funkce se nemusí shodovat se jménem souboru, v němž je uložena, Rozhodující při volání funkce je jméno 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 závorkách. Vstupní ani výstupní proměnné nejsou povinné, takže hlavička funkce může vypadat třeba takto:

```
function funkce1
```

Na řádcích za hlavičkou může být umístěna nápověda k funkci. Jedna se o řádky začínající znakem '%' (komentáře) obsahující popis chování funkce. Nápověda se zobrazí pomocí příkazu "`help`", jehož parametrem je jméno souboru s funkci (bez přípony "`m`") – např. "`help funkce1`".

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))
```

Uvedené příkazy vložíme do souboru se jménem "`trojuh.m`" v pracovním adresáři nebo v adresáři, kam je nastavena cesta.

Jména skutečných proměnných, které předáváme funkci jako parametry, se samozřejmě nemusí shodovat se jmény proměnných ve funkci samotné, vstupní parametry je možné zadávat přímo hodnotami. Tedy výše uvedenou funkci "`trojuh`" můžeme volat např. takto:

```
[x,y]=trojuh(3,4,5);
```

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. Tedy pokud zadáme na příkazovém řádku jen

```
trojuh(3,4,5)
```

získáme pouze obsah trojúhelníka, kterážto hodnota bude navíc přiřazena do proměnné "`ans`".

Funkce je ukončena po vykonání všech příkazů, které obsahuje. Je možné funkci také ukončit dřív pomocí příkazu "`return`". Předčasné ukončení činnosti funkce dosáhneme též funkcí "`error`", která navíc vyvolá zvukový signál a vypíše textový řetězec, který je jejím parametrem. Příklad:

```
[m,n]=size(a);
if m*n==0 % matice a je prazdna
  error('Prazdna matice');
end
```

9.2 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 v ní 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 známe mimo tuto funkci.

Výjimku tvoří dávkové soubory, ve kterých jsou známé proměnné definované v prostředí, odkud byly zavolány. Naopak pokud v nich nějaké proměnné definujeme, jsou pak známé po jejich ukončení. Mějme například 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.

9.3 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.

9.3.1 Větvení programu

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

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

Větve "`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 dá 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" obsahující následující příkazy:

```
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
```

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.

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 *výrazu* 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" obsahující následující příkazy:

```
s=input('Zadejte poradi dne v tydnu: ')
switch s
case 1
  disp('Zadali jste cislo pro pondeli.');
case 2
  disp('Zadali jste cislo pro utery.');
case 3
  disp('Zadali jste cislo pro stredu.');
case 4
  disp('Zadali jste cislo pro ctvrtek.');
case 5
  disp('Zadali jste cislo pro patek.');
case 6
  disp('Zadali jste cislo pro sobotu.');
case 7
  disp('Zadali jste cislo pro nedeli.');
otherwise
  disp(['Zadali jste cislo ',num2str(s),' a to neodpovida zadnemu dni v tydnu!']);
end
```

Po zadání příkazu "davka2" z příkazové řádky je uživatel vyzván, aby zadal číslo, které odpovídá pořadí dne v týdnu. Po zadání hodnoty a stisknutí klávesy ENTER se vypíše na obrazovku, který den odpovídá zadanému číslu.

9.3.2 Cykly

Pro cyklus jsou v MATLABu dva příkazy. Je to příkaz "`while`" a příkaz "`for`". Použití příkazu "`while`" vypadá takto:

`while` *podmínka*
  příkazy
`end`

Pro vyhodnocení "*podmínky*" 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" obsahující následující příkazy:

```
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;
```

Po zadání příkazu "davka3" 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.

Trochu jinak se používá příkaz "`for`":

```
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. Když chceme vypočítat faktoriál z čísla `n` v běžném programovacím jazyku, postupujeme obvykle následovně:

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

Pro tento účel stačí v MATLABu napsat příkaz

```
faktor=prod(1:n);
```

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

9.4 Nástrahy při programování v MATLABu

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

na jiný způsob práce.

Uveďme 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>=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 jazyku 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 `^` na operátor `.^`, 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 jediná, 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
```

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;
```

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;
```

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

9.5 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. Protože ale tento editor nemusí být vždy dostupný, popíšeme 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

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

Po zastavení běhu programu je možné kontrolovat hodnoty proměnných jejich vý-

pisem, 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 nadřízeného funkci, v níž se právě nacházíme. Opačně funguje funkce `dbdown`.

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