ZÁPADOČESKÁ
UNIVERZITA

Institut technologie a spolehlivosti

Doc. Ing. Miroslav Balda, DrSc.

Úvod do MATLABu

Plzeň, prosinec 1994

Západočeská univerzita Doc. Ing. Miroslav Balda, DrSc.

Úvod do MATLABu

Učební text vznikl jako jedna část řešení projektu z fondu rozvoje vysokých škol v roce 1994. Jde o podklad pro výuku nástroje – programovacího jazyka `MATLAB` – určeného pro práce ve cvičeních posluchačů přednášek předmětu "Statistická mechanika" na Fakultě aplikovaných věd Západočeské univerzity. Jde o úplné základy, které je nutno zvládnout pro sestavování jednoduchých programů. **Klíčová slova:** `MATLAB`, programování, statistická mechanika

Plzeň, prosinec 1994

Obsah

Kapitola 1

Úvod

Pro překonání softwarové krize vznikají stále důmyslnější programové prostředky, které umožňují uživateli efektivní tvorbu aplikačního programového vybavení a i využití strojního času počítače. Mezi ně patří i výrobek firmy The Maths Works, který dostal jméno MATLAB - Matrix Laboratory. Jde o interaktivní program, jímž lze řešit mnohé problémy s nimiž se setkáváme v technické praxi. Komprimuje v sobě mnohé z výsledků programových systémů lineární algebry EISPACK a LINPACK, které jsou k dispozici jak na větších počítačích, tak v současné době i na osobních počítačích.

MATLAB je interpretační jazyk (podobně jako BASIC). To má za následek, že uživatel dostane odpověď na svůj povel takřka vzápětí. Je vysoce optimalizován. Celý je zapsán v jazyku C a části patrně i ve strojovém kódu k dosažení maximální rychlosti výpočtu. Uživatelé znalí jiných programovacích jazyků (jako např. FORTRAN nebo PASCAL) velmi brzo zjistí, že základní maticové operace jsou v MATLABu rychlejší než v jimi užívaném jazyku. Spolu se značně úsporným způsobem programování je žádaný výsledek k dispozici podstatně dříve a s nižšími náklady, je-li řešen v MATLABu. S ohledem na skutečnost, že MATLAB umožňuje uživateli rozšiřovat paletu řešených úloh a začleňovat do sebe i vlastní uživatelský software zpracovaný v jazycích FORTRAN a C, jsou jeho možnosti velmi široké.

Otcem MATLABu je Cleve Moler, známý matematikům z oblasti lineární algebry. Pracoval na projektu EISPACK a sám se snažil jeho využití zrealizovat v první verzi operativního prostředku pro maticové výpočty MATLABu. Ve srovnání s dnešním stavem byla první verze (naprogramovaná ve FORTRANu) velice chudá. Až později s několika nadšenci zakládá firmu MathWorks, která se věnuje vývoji již profesionální verze MATLABu stavěného na bázi jazyka C.

V současné době je MATLAB k dispozici na řadě počítačů, takže není problém přenositelnosti uživatelského softwaru zapsaného v jazyku MATLAB. Mezi prostředí, pro něž je inplementován patří pracovní stanice SUN, Apollo, HP, VAX, MicroVax, osobní počítače XT, AT, 386, 486, Macintosh a pro řadu paralelních strojů.

Na osobních počítačích je MATLAB k dispozici jak pro prostředí DOSu ve verzi 3.5, tak i pro prostředí Windows ve verzi 4.2. Obě verze se navzájem liší zejména v možnostech, které obě prostředí poskytují. Verze 4.x představuje velký kvalitativní skok. Přesto však základy jsou stejné a mohly proto být uvedeny na následujících stránkách tohoto úvodu společně. Na drobné odchylky verzí je vždy upozorněno.

Podstatné rozdíly jsou výrazné až u pokročilých aplikací při užití MATLABu verze 4.x a to zejména v

- grafice
- práci s řídkými maticemi
- podstatně bohatějších knihovnách (toolboxech)
- simulační nadstavbě – SIMULINKu.

Cílem tohoto stručného popisu jazyka MATLAB je uvést potenciálního uživatele do filozofie jazyka a být mu vodítkem při počátečních pokusech o jeho využití. Vychází z první varianty stručné příručky vydané v roce 1989 pro MATLAB v. 3.2 [3]. Tento původní materiál byl zásadně přepracován a zaktualizován. Probíraná látka je doprovázena řadou příkladů, které usnadní čtenáři ji snáze pochopit. Příklady nemají jen akademickou cenu, ale mohou sloužit i jako podklad pro vlastní aplikace.

K druhému vydání:

Od doby prvního dotisku v roce 1996 došlo k dalšímu mohutnému rozvoji nejen výpočetní techniky, ale i programových prostředků, MATLAB nevyjímaje. Nová verze MATLABu 5.1, která je začátkem roku 1998 k dispozici, dovoluje mnohem více, než je uvedeno v tomto stručném úvodu. Nicméně základy stručně uvedené na následujících stránkách jsou i v nové verzi platné a umožní začátečníkovi se orientovat a sestavovat jednoduché programy. Pro pokročilé programování s používáním struktur a všech možností grafiky je zapotřebí prostudovat manuály nebo specializované tisky.

V Plzni, leden 1998

Kapitola 2

Charakteristika jazyka

Struktura jazyka MATLAB je navržena tak, aby splňoval takřka všechny potřeby uživatele z oblasti aplikované matematiky a přitom aby zůstal z uživatelské úrovně jednoduchý v použití. Proto MATLAB zahrnuje nejrůznější algoritmy, lineární algebrou počínaje, přes prostředky nelineární analýzy, až velice snadnou grafikou konče.

Paměťové nároky MATLABu jsou dosti veliké. Základní charakteristiky:

- Maximální velikost pole	:	8188 prvků u MATLABu verze 3.x pro PC 286 (řád cca 90)
	:	„libovolná“ u MATLABu verze 4.x
- Jeden prvek:	:	8 bajtů, 16 míst, rozsah čísel 10^{-308} až 10^{+308}
- Předdefinovaná jména	:	`eps` „machine epsilon“ $\approx 2.10^{-16}$
		`pi` $4 * \arctan(1) \approx \pi$
		`Inf` „1/0“ – [infinity]
		`NaN` „0/0“ – [not a number]
		`ans` jméno výsledku nepřiřazeného výrazu.
		`i, j` imaginární jednička (lze přepsat)

Tabulka 2.1: Základní informace

Základními objekty, se kterými MATLAB pracuje, jsou matice. Nad nimi provádí veškeré operace.

Matice je obecně obdélníková tabulka prvků, mající `m` řádek a `n` sloupců. O takové matici říkáme, že je *typu* `m,n`. Je-li `m = n`, mluvíme o čtvercové matici *řádu* `m`. V případě, že `m` nebo `n` jsou jednotkové, mluvíme o *vektorech*. Prvky mohou být čísla (reálná i komplexní) a nebo i znaky (!). Při tom musí být prvky v celé matici homogenní.

Vektory jsou dvojího typu, a to vektor-řádka při `m = 1`
vektor-sloupec při `n = 1`.
Počet prvků ve vektoru se nazývá *dimenzí*. Je-li dimenze vektoru jednotková, jde o speciální matici řádu 1, která se nazývá **skalár**.

Jednodušší operace nad maticemi provádí MATLAB z paměťově rezidentních programových modulů zpracovaných v jazyku C. Složitější operace realizuje za pomoci modulů - funkcí uložených na externí paměti, zapsaných v jazyku MATLAB a nazývaných M-soubory (M-files). Z hlediska MS DOS jsou to znakové soubory s příponou „.m“. Kromě dodaných M-souborů může uživatel budovat i vlastní M-soubory pomocí libovolného editoru a z nich pak

vytvářet knihovny modulů - *toolboxy*. Výrobce sám dodává toolboxy profesionální úrovně pro mnoho aplikačních oblastí.

2.1 Speciální znaky

K zápisu příkazů se využívají v MATLABu také znaky se zvláštním významem. Jejich stručný přehled je uveden v tabulce:

!	uvádí běžný příkaz operačního systému užitý z MATLABu
%	uvádí text poznámky (do konce řádky)
.	desetinná tečka v číslech, příznak prvkové operace
,	oddělovač indexů, argumentů funkcí, prvků matic a příkazů v řádce
;	konec příkazu s potlačením výstupu výsledku, konec řádky v matici
:	generování vektorů - lineárních posloupností, indexování
..	pokračování příkazu na další řádce u MATLAB v. 3.x
...	pokračování příkazu na další řádce u MATLAB v. 4.x
( )	závorky výrazů, indexové závorky
[ ]	maticové závorky
=	operátor přiřazení

Tabulka 2.2: Funkce speciálních znaků

- MATLAB obsahuje několik **systémových příkazů** DOSu psaných prostředky jazyka:

dir	zobrazení běžného adresáře
type	zobrazení souboru, jehož jméno (bez .M) náleduje
del	vypuštění souboru, jehož jméno (bez .M) náleduje
chdir	změna běžného adresáře

  Pokud potřebuje uživatel jiný systémový příkaz, užije znak „!“.

  Příklad:

  ```
  !a:     přepne na 1. floppy disk (FDU)
  !edt    vyvolá soubor o jménu edt.exe
  ```

- **Poznámky** lze psát od libovolné pozice v řádce za znakem %. Serie úvodních poznámkových řádek z M-souborů vystoupí po povelu

  help jméno M-souboru

  a to nejen u standardních modulů, ale i u uživatelských. První řádka nezačínající znakem % ukončuje výstup poznámky.

- Ostatní znaky se užívají ke konstrukci příkazů a matic ve jménech a jako operátory.

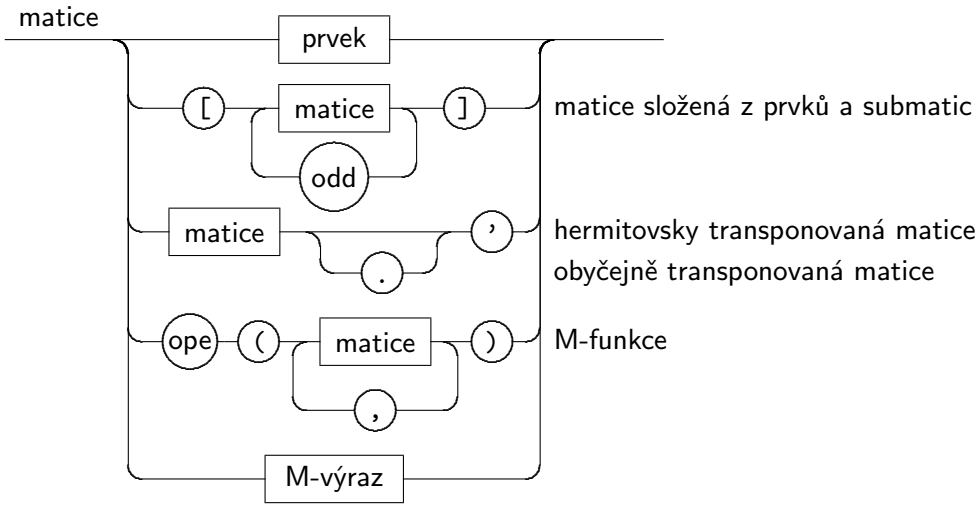
Poznámka:

V komentářovém sloupci tabulek bude obvykle použit MATLABovský způsob zápisu výrazů, tj. se znaky operátorů popsanými dále.

Kapitola 3

Matice

Matice jsou základními útvary MATLABu. Jejich syntaxe je uvedená v diagramu:



Obrázek 3.1: Syntaktický diagram popisu matice

Symbolem `ope` je v diagramu označen operátor funkce, který má formu jména (identifikátoru) funkce. Oddělovačem `odd` může být mezi prvky (maticemi) mezera nebo čárka, mezi řádkami středník nebo znak nové řádky.

Příklady:

```
[  1 2 3
   4 5 6                  matice zadaná prvky (vždy po řádcích!)
   7 8 0  ]
[1 2 3; 4 5 6; 7 8 0]     tatáž matice
[1 2 3].'   obyčejná transpozice řádky na sloupec
expm(A)     exponenciála matice A
exp(A)      exponenciála prvků matice A
[]          prázdná matice
```

3.1 Vektory s lineární posloupností hodnot

Vygenerujeme je buď funkcí `linspace` (viz help), anebo jednodušeji zápisem

`j:i:k`

zajistíme vygenerování vektoru-řádky *hodnot* (nikoliv výrazů) `[j,j+i,j+2i,...,k]`

Příklady:

`1:2:9`	vygeneruje vektor lichých čísel `[1,3,5,7,9]`
`0:pi/4:pi`	vygeneruje `[0, pi/4, pi/2, 3*pi/4, pi]`
`2:-.5:0`	vygeneruje `[2, 1.5, 1, 0.5, 0]`

Zvláštní případy: pro `j>k` s `i>0` nebo pro `j<k` s `i<0` se vygeneruje prázdný vektor `[]`

Je-li krok v indexu vektoru `i=1`, může mít zápis vektoru tvar

`j:k`

V některých případech lze zápis vektoru ještě více zjednodušit. Je-li `j = i = 1` a `k =` dimenze vektoru, lze pro indexování všech jeho prvků použít pouhý znak „dvojtečka“ (viz dále)

`:`

3.2 Vektory s logaritmickou posloupností hodnot

Tuto činnost zajišťuje funkce `logspace`, která je logaritmickým ekvivalentem operátoru „:“ u lineární posloupnosti. Funkce `logspace` rozdělí lineárně interval logaritmů argumentů.

`logspace`	`(d1,d2)`	generuje vektor 50 hodnot z intervalu 10^{d1} až 10^{d2} s logaritmickým dělením
	`(d1,d2,n)`	generuje vektor o `n` hodnotách
	`(d1,pi)`	generuje vektor z intervalu 10^{d1} až π, což je vhodné pro číslicové zpracování signálů

Tabulka 3.1: Funkce logaritmického dělení intervalu

3.3 Submatice

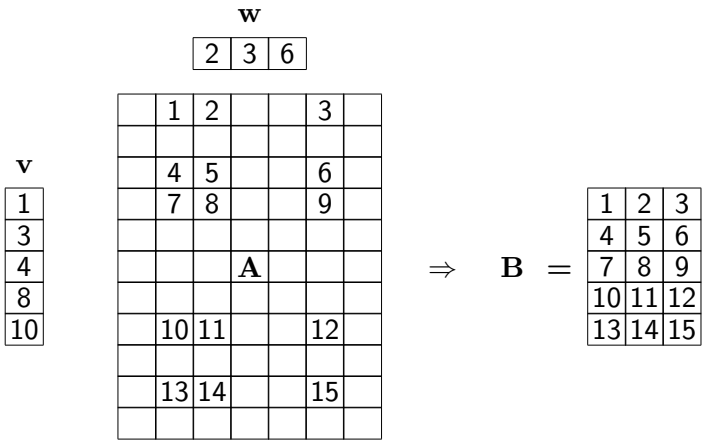
Nejobecnější způsob vyjmutí submatice z matice se uskuteční výrazem typu `A(v,w)`, kde `v` a `w` jsou vektory obsahující

- buď indexy řádek (`v`) a sloupců (`w`), které se mají z původní matice vyjmout. Při tom dimenze vektorů `v, w` jsou max. rovny dimenzi řádkového či sloupcového vektoru původní matice,
- nebo výsledky logických operací (tj. 0 nebo 1). V tomto případě **musí** dimenze vektorů `v` odpovídat počtu řádek a `w` počtu sloupců původní matice.

Vektory `v, w` mohou být pojmenované (fyzické), anebo pouze explicitními množinami indexů. První případ je uveden výše, kde indexové vektory měly jména `v, w`. Druhým případem jsou vektory např. `[3, ix, 1]` , `1:3:25` nebo dokonce i `a~>0`. Je pochopitelné, že oba typy mohou být i kombinovány, jako např. v zápisu `A(n:-1:1,j)`. V tomto případě nově vzniklá matice bude složena ze sloupců, jejichž indexy byly obsahem vektoru `j` a z řádek v opačném pořadí, než bylo v matici `A`.

Příklady:

a) `A(v,w)`



b)	`A([1 3 4 8 10], [2 3 6])`	totožno s příkladem a)
c)	`A(2:2:10, 2:2:6)`	vytáhne prvky se sudými indexy
d)	`A(2:10, 2:6)`	„odrámuje“ matici `A` z příkladu a)
e)	`A(i,:)`	`i`-tá řádka `A`
f)	`A(:,j)`	`j`-tý sloupec `A`
g)	`A(i,j)`	`(i,j)`-tý prvek `A`
h)	`A(:)`	celá matice `A` jako vektor sloupcových vektorů (t.j. `vec A`)
i)	`A(:,n:-1:1)`	matice `A` s převráceným pořadím sloupců
j)	`A(:,sum(A)>0)`	matice `A` jen s těmi sloupci, jejichž sumy prvků jsou kladné

Pozor: Je-li některý indexový výraz prázdný, je i M-výraz prázdný!

3.4 Speciální matice

Speciální matice se generují pomocí vnitřních funkcí uvedených v tab. 3.2.

Matice konstant	`zeros`	nulová	Argumenty: `(n)` - řádu `n`
	`ones`	jedniček	`(m,n)` - typu `m, n`
	`eye`	jednotková	`(A)`, `(size(A))` - typu jako `A`
	`rand`	pseudonáhodná	
(ko)diagonální	`X=diag(v)`	diagonální matice z vektoru `v`	
	`X=diag(v,k)`	matice s k-tou kodiagonálou z vektoru `v`	
	`v=diag(X)`	vektor z hlavní diagonály `(k=0)` matice `X`	
	`v=diag(X,k)`	vektor z k-té kodiagonály `(-m+1≤k≤n-1)`	
Zobecněný trojúhelník	`triu(X)`	horní trojúh. matice s hlav.diagonálou (`k=0`)	
	`triu(X,k)`	dtto od k-té kodiagonály vč. `(-m+1≤k≤n-1)`	
	`tril(X)`	dolní trojúh. matice s hlav.diagonálou (`k=0`)	
	`tril(X,k)`	dtto od k-té kodiagonály vč. `(-m+1≤k≤n-1)`	

Speciální	`hankel(c)` `hankel(c,r)`	Hankelova matice s `c` v prvním sloupci dtto a s poslední řádkou `r`
	`toeplitz(c)` `toeplitz(c,r)`	Töplitzova matice s 1. sloupcem `c` dtto a s první řádkou `r`
	`hilb(n)` `invhilb(n)`	Hilbertova matice řádu `n` „přesná“ inverze Hilbertovy matice
	`hadamard(k)`	Hadamardova matice řádu 2^k

	`gallery(m)`	Testovací matice řádu `m`:
		`m=3` špatně podmíněná matice
		`m=5` zajímavý problém vlastn. hodnot (`EVP`)
		`m=8` Rosserova matice pro symetrický `EVP`
		`m=21` Wilkinsonova `W21+; EVP`

	`magic(n)`	matice řádu n celých čísel `(1,`n^2`)` se stejnými sumami přes řádky i sloupce

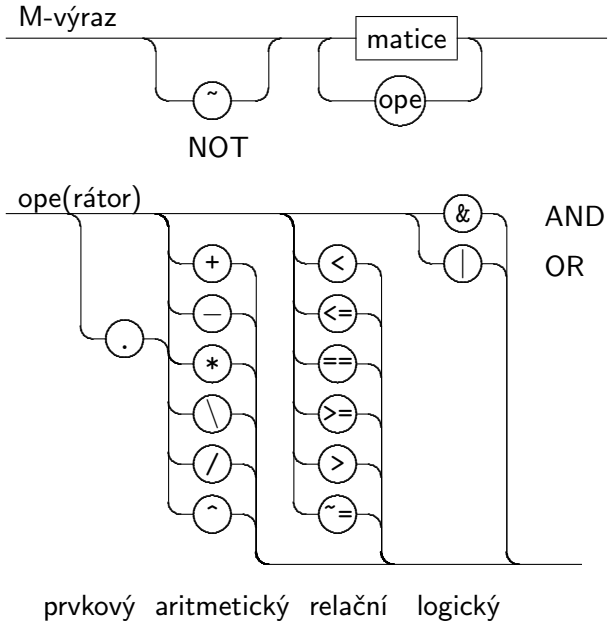
Tabulka 3.2: Přehled speciálních matic

Příklad: Nulování matice kromě diagonály: `diag(diag(A))`

3.5 M-výrazy

Maticový výraz – `M`-výraz – je složen z matic a operátorů a jeho vyhodnocením vznikne matice. To však jen tehdy, pokud použité operace byly nad danými maticemi přípustné (např. s ohledem na typy matic). Z běžných pravidel maticového počtu je povolena následující výjimka:

Matici řádu 1 (skalár) lze užít ve spojení s maticí libovolného typu při aritmetických operacích tak, že se příslušná operace aplikuje ke každému prvku matice.



Obrázek 3.2: Přehled operátorů

3.5.1 Aritmetické operátory

Aritmetické operátory slučování se aplikují ke stejnolehlým prvkům matic stejného typu. Jde tedy o operaci mezi prvky. Podobný charakter „prvkové operace“ mají i ostatní aritmetické operátory, pokud jim předchází znak „tečka“.

Pozornost zasluhují operátory „dělení“. Realizují velice komplikované algoritmy, které lze stručně interpretovat následujícím způsobem:

„Nahraď příslušné lomítko hvězdičkou a matici, ke které bylo lomítko nakloněno, (pseudo) invertuj!“

Ve skutečnosti se ani u čtvercových matic inverze neprovádí, ale řeší se systém lineárních algebraických rovnic (pokud nejde o pouhé dělení každého prvku skalárem).

Prostý operátor mocnění, který se aplikuje jen u čtvercových matic, má tři verze.

1. Je-li exponenciální výraz celočíselným skalárem p, pak se realizuje pronásobováním matice.
2. V případě, že je p necelé, vypočte se mocnina jako funkce matice:

   $$\boldsymbol{A}^p = \boldsymbol{V}\boldsymbol{S}^p\boldsymbol{V}^{-1},$$

   kde $\boldsymbol{V}$ je modální matice (tj. matice vlastních vektorů matice $\boldsymbol{A}$) a $\boldsymbol{S}$ je diagonální spektrální matice složená z vlastních hodnot (tedy Jordanova s poli řádu 1).
3. Jen v případě umocňování skaláru může být exponentem maticový výraz. Jinak je hlášena chyba.

3.5.2 Relační a logické operátory

Realizují **vždy** jen prvkové operace. Užívají se mezi maticemi téhož typu. Výsledkem je běžná matice stejného typu jako byly matice operandů, obsahuje však pouze prvky s hodnotou 1 (true) nebo 0 (false) podle toho, zda dané prvky matic podmínce vyhovují, nebo nevyhovují.

Příklady:

`i = sqrt(-1)`	je imaginární jednotka
`A + i*B`	je komplexní maticí pro `B`$\neq$`0` a `A` a `B` reálné
`x = A\b`	je vektor řešení soustavy lineárních algebraických rovnic `Ax = b`
`A.*A`	je matice stejného typu jako `A` s kvadráty prvků
`~(A == B)`	`NONEQUIVALENCE`. Prvek je = 1, když $a_{ij} \neq b_{ij}$.
`A + 1`	přičte ke každému prvku `A` jedničku
`A/3`	vydělí každý prvek matice `A` třemi
`x(x>0)`	vypustí z vektoru `x` nekladné prvky a vektor komprimuje, tj. prvky nevyhovující podmínce vypustí a zbytek přeindexuje
`[ zeros(m), eye(m)` `-M\[K, B] ];`	$\begin{bmatrix} 0 & , & I \\ -M^{-1}K & , & -M^{-1}B \end{bmatrix}$ se všemi maticemi řádu `m`
`(0:10).^2`	vektor-řádka o prvcích `0, 1, 4, ..., 100`
`linspace(-2,3,6)`	vektor-řádka o prvcích `[-2, -1, 0, 1, 2, 3]`
`linspace(-2,3)`	vygeneruje vektor-řádku o 100 prvcích stejně jako výraz `-2:5/99:3`

Dále jsou uvedeny menší procedury, v nichž je patrno využití některých výrazů v maticových příkazech a funkcích, které budou probrány dále.

```
%-------------------------------------------------------

% I2S.M    Integer to string conversion
% *****
%    s   výstupní řetěz o n znacích s úvodními nulami
%    i   celé číslo

function s = i2s(i,n)
%        ************
if nargin<2, n=2; end
k = 10^n;
m = abs(i);
s = [];
while n>0
  n=n-1;
  k = k/10;
  j = fix(m/k);
  m = m-j*k;
  s = [s int2str(j)];
end

%-------------------------------------------------------

% TODAY.M    rok-měsíc-den
% *******

function s = today
%        *********
T=clock;
s = [i2s(T(1)),'-',i2s(T(2)),'-',i2s(fix(T(3)))];

%-------------------------------------------------------

% HOUR.M    hodiny:minuty:sekundy  dne
% ******

function s = hour
%        ********
T=clock;
s = [i2s2(T(4)),':',i2s2(T(5)),':',i2s2(fix(T(6)))];

%-------------------------------------------------------

% ERASE.M    Vypust radky a sloupce matice
% *******
%    B   výsledná matice
%    A   vstupní matice
%    ix  vektor indexů vypouštěných řádek
%    iy  vektor indexů vypouštěných sloupců

function B = erase(A,ix,iy)
%        ******************
[m,n] = size(A);
jx = 1:m;
jy = 1:n;
jx(ix) = zeros(size(ix));
jy(iy) = zeros(size(iy));
B = A(find(jx),find(jy));

%-------------------------------------------------------
```

Kapitola 4

Příkazy

Příkazy MATLABu se píší obvykle na samostatné řádky. Pokud se píší v jedné řádce, oddělují se čárkou, případně středníkem. Dělí se na

- řídící a informační příkazy
- přiřazovací příkazy
- příkaz funkce
- podmíněné příkazy
- příkazy cyklů
- makropříkazy
- příkazy vstupu a výstupu

4.1 Řídící a informační příkazy

Základní řídící příkazy jsou uvedeny v tabulce tab. 4.1. Další řídící příkazy jsou v odstavci s názvem `Grafické výstupy`.

Uvolňování paměti	`A = [ ]`	vytvoří matici `A` s nulovým řádem
	`clear`	vymaže proměnné z pracovní oblasti paměti
	`clear X A`	vymaže definované matice nebo funkce `(X A)`
	`clear functions`	vynuluje z paměti všechny funkce
Setřásání	`pack`	- uloží všechny proměnné na `pack.tmp` - vynuluje všechny proměnné v paměti - zavede `pack.tmp`. - zruší `pack.tmp`
Ukončení výstupu		z klávesnice `(Ctrl)(C)`
Ukončení práce	`quit` `exit`	nebo `Ctrl-Z` u v. 3.x v `DOS`u, anebo `ALT-F4` u v. 4.x ve `Windows`
Pomocné	`help`	tisk informací o modulech `MATLABu`
	`help jméno`	tisk úvodních poznámek se souborem `jméno.m`
	`demo`	demostrační příklady

Informace o pamětech	`who`	seznam proměnných v paměti
	`whos`	jako who s dimenzemi proměnných
	`what`	seznam M-souborů na disku
	`length(x)`	dimenze vektoru, počet řádek matice
	`size(A)`	typ matice (počet řádek, počet sloupců)
Návrat z M-souboru		na konci M-souboru prázdný, jinde `return`
	`return`	v podmíněném příkaze ve funkci
Předání řízení klávesnici	`keyboard`	pro libovolné povely. Končí se `(Ctrl)(Z)` následuje návrat do původní řady povelů
Časové údaje	`t=clock`	`t = [rok, měs., den, hod., min, sek.]`
	`etime(t2,t1)`	uplynulý čas $t_2 - t_1$ v sekundách `t1,t2 =` časy změřené pomocí `clock`
	`etime(clock,t)`	čas od posledního měření `t=clock`
Čekání	`pause`	čeká na stlačení klávesy
	`pause(n)`	čeká n sekund (`n` nemusí být celé)

Tabulka 4.1: Výběr řídících a informačních příkazů

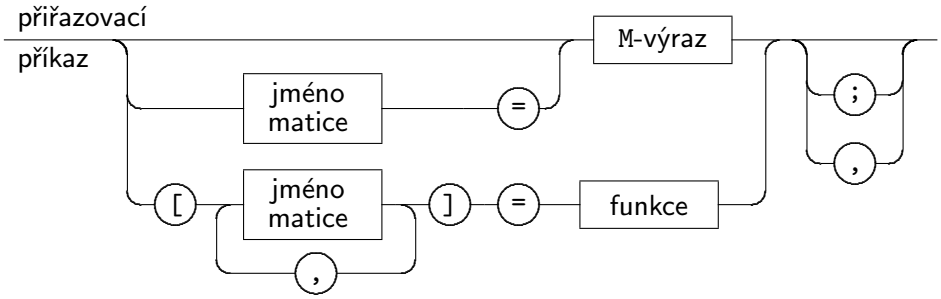
Poznámka:

Pro měření času je v MATLABu funkce `clock`. Ta dodá vektor o 6 prvcích, z nichž jen poslední - sekundy - není celým číslem. Výstup tohoto vektoru tiskem nemá pěknou úpravu. Vzhlednější výstup se získá pomocí příkazu `fix(clock)`. U verze 4.x lze navíc použít pro start stopek povel `tic` a pro jejich zastavení a vytisknutí časového údaje povel `toc`.

Příklad: `x = rand(1,1024); tic, y=fft(x); toc`

4.2 Přiřazovací příkaz:

Přiřazovací příkaz MATLABu má strukturu uvedenou v obr. 4.1.



Obrázek 4.1: Syntaxe přiřazovacího příkazu

Výsledek přiřazovacího příkazu se zobrazuje na monitoru. Chceme-li výstup na obrazovku potlačit, zakončíme příkaz středníkem.

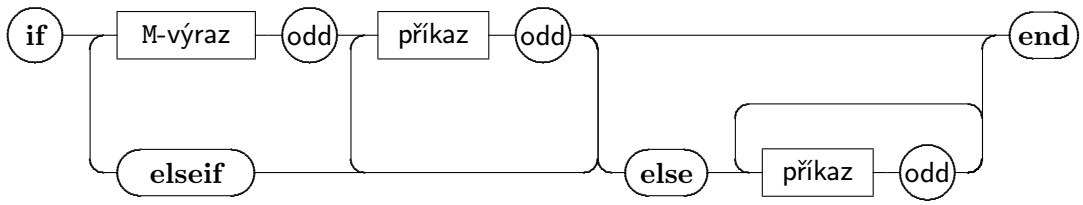
1. **První tvar** se užije tehdy, chceme-li zobrazit výsledek `M`-výrazu. Ten se kromě zobrazení uloží ještě do matice se jménem `ans` (answer = odpověď).

2. **Druhý tvar** přiřazovacího příkazu je nejobvyklejší. Generuje se jím matice, jejíž jméno je uvedené na levé straně příkazu.

3. **Třetí tvar** obsahující na levé straně formálně matici složenou ze submatic je povolen pouze ve spojitosti s voláním funkce s více výstupními parametry. Formalita spočívá v tom, že zdánlivé výstupní „submatice“ nemusí tentokrát mít stejný počet řádek, aby mohly vytvářet skutečnou matici. Jde v podstatě o seznam výstupních argumentů.

4.3 Podmíněný příkaz:

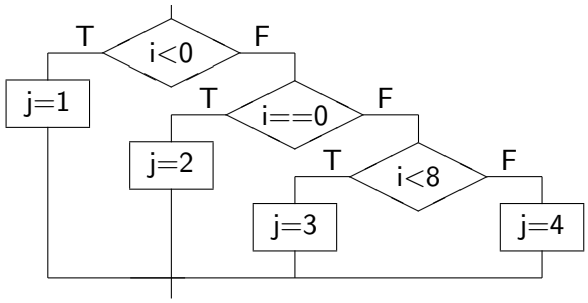
Strukturu podmíněného příkazu lze vyjádřit syntaktickým diagramem obr. 4.2, kde oddělovačem „odd“ je buď `ENTER`, `čárka` nebo `středník`.



Obrázek 4.2: Podmíněný příkaz

Příklad:

```
if i<0, j=1;
elseif i==0, j=2;
elseif i<8, j=3;
else
j=4;
end
```



Podmínka je `TRUE`, jestliže výsledek `M`-výrazu má **všechny** prvky nenulové. Ve spojení s podmíněným příkazem se užívají funkce `any`, `all`, `exist`, a `break` (viz dále v odstavci `Přerušení cyklu`). Funkcí typu `is...(x)` existuje celá řada. Všechny testují, zda `x` splňuje určitou podmínku. Je-li podmínka splněna, je hodnota funkce rovna 1, v opačném případě 0. Tak např.:

`isstr(x)` testuje, zda `x` je řetězem (string),
`isempty(x)`, zda `x` je prázdnou maticí, ale i další, jako

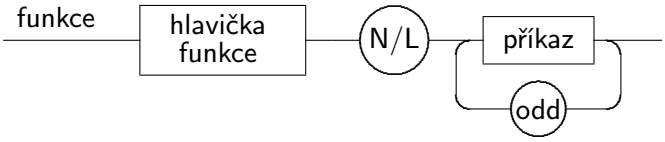
`isglobal(x)`, `isinf(x)`, `isreal(x)`, `ishold(x)`, `isletter(x)`, `isspace(x)`, `issparse(x)`, ..., z nichž některé jsou definovány pouze pro `MATLAB` v. 4.x.

4.4 Příkaz funkce

Funkce je v `MATLAB`u tvořena `M`-souborem o syntaxi z obr. 4.3.

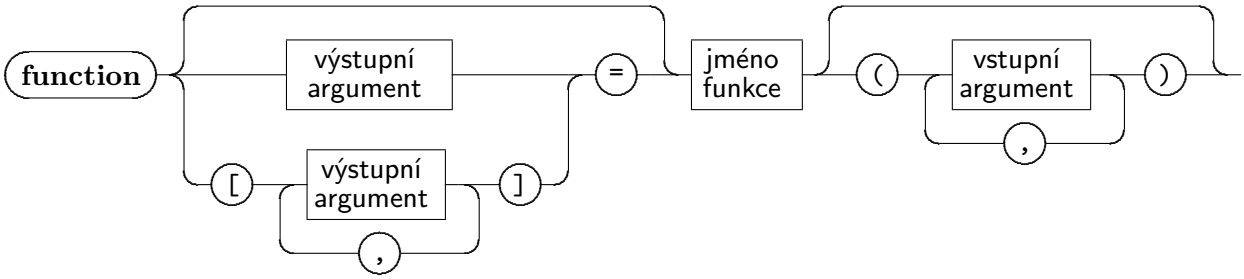
funkce	**argum.**	**výsledek**
`all`	vektor matice	skalár=1, když **všechny** prvky jsou nenulové řádka skalárů s ohledem na sloupce matice
`any`	vektor matice	skalár=1, když alespoň jeden prvek je nenulový řádka skalárů s ohledem na sloupce matice
`exist`	„jméno“	0 - neexistuje 1 - existuje jako proměnná 2 - existuje jako `M`-soubor
`finite`	matice	matice jedniček, kde prvky jsou konečné, a nul jinde
`isnan`	matice	matice jedniček, kde prvky jsou `NaN`, a nul jinde
`find`	vektor	vektor indexů nenulových prvků vstupního vektoru

Tabulka 4.2: Funkce užívané v podmíněných příkazech



Obrázek 4.3: Syntaxe funkce v MATLABu

Symbol `N/L` zde zastupuje znak „nová řádka“. Hlavička funkce má podobný tvar jako přiřazovací příkaz s voláním funkce. Oproti němu jí předchází slovo `function`:



Obrázek 4.4: Struktura hlavičky funkce

Autor funkce musí zajistit, aby celý modul funkce byl uložen jako `M`-soubor, jehož jméno je rozhodující, a má být totožné se jménem funkce z hlavičky (vpravo od rovnítka). V hlavičce funkce jsou plné seznamy argumentů, které zajistí vazbu na místo, odkud je funkce právě volána. Při volání není zapotřebí využít všechny argumenty. Skutečný počet užitých argumentů lze v těle funkce testovat pomocí systémových proměnných `nargin` a `nargout`. Ve funkcích (`M`-souborech) může být příkazem k návratu z funkce (při splnění či nesplnění podmínky) povel `return`. Na fyzickém konci funkce není povinný, a proto se nepíše.

Příklad: (na zřetězené volání funkcí)

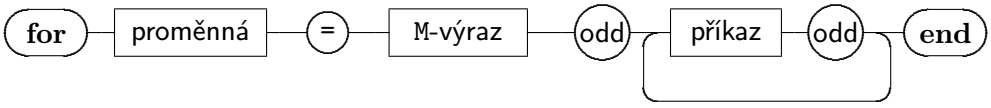
Vypuštění malých prvků z vektoru: `x = x(find(abs(x)≥eps))`,
Nahrazení `NaN` nulami v matici `A`: `A(isnan(A)) = zeros(size(find(isnan(A))))`

4.5 Příkazy cyklů

MATLAB má dva druhy cyklů, a to cykl typu `for` a cykl typu `while`.

4.5.1 Cykl typu for

Cykl typu `for` má strukturu vyjádřenou syntaktickým diagramem v obr. 4.5.



Obrázek 4.5: Syntaxe příkazu cyklu typu **for**

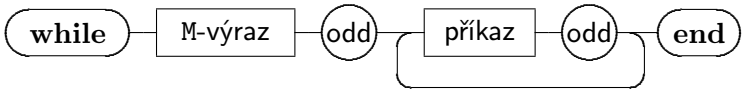
Je-li `M`-výrazem skalár, pak cykl proběhne jen jedenkrát. Je-li jím vektor, pak proměnné na levé straně se v jednotlivých průbězích přidělují postupně prvky vektoru. V případě, že `M`-výrazem je matice typu `(m,n)`, potom v `j`-tém kroku cyklu je proměnné cyklu přiřazen `j`-tý sloupcový `vektor`(!). Cykl proběhne `n`-krát.

Příklad:

`for i=1:n`	Pravá strana je řádkový vektor, tedy matice typu `(1,n)`. Cykl proběhne s `i=1, 2, ..., n`.
`for v=V`	Pravá strana je matice `V` typu `(m,n)` Cykl proběhne s `v = V(:,1)` až `v = V(: ,n)`

4.5.2 Cykl typu while

Struktura cyklu `while` je dána diagramem obr. 4.6.

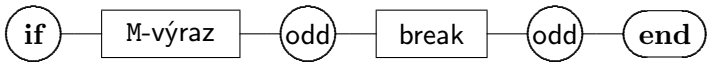


Obrázek 4.6: Syntaxe příkazu cyklu typu **while**

Cykl probíhá tak dlouho, pokud `M`-výraz bude mít **všechny** prvky nenulové.

Přerušení cyklu:

Cykly `for` i `while` lze nuceně přerušit i uprostřed těla cyklu pomocí splněné dodatečné podmínky zařazené mezi příkazy a využívající příkaz „`break`" pro přerušení cyklu. Celý příkaz s podmínkou má tvar podmíněného příkazu podle obr. 4.7:



Obrázek 4.7: Schéma příkazu pro přerušení cyklu

Při alespoň jednom nulovém prvku M-výrazu se cykl přeruší a řízení se předá prvnímu příkazu za cyklem (za příkazem `end`).

4.6 Makropříkazy

V MATLABu rozlišujeme 4 druhy makropříkazů, z nichž příkaz funkce byl již probrán dříve:

1	`funkce`	úsek programu uložený jako M-soubor, který ze vstupních argumentů vyhodnotí výstupní argumenty. Všechna jména užitá ve funkci jsou v ní `lokální`
2	`skript`	úsek programu uložený jako M-soubor pod určitým jménem. Na rozdíl od funkce nemá `žádné` argumenty a jím užívané identifikátory jsou `globální`
3	`eval(s)`	funkce, která vyhodnotí řetězové (textové) proměnné `s` jako M-výraz. Jeho hodnotu lze pak přiřadit matici Př.: `r = eval('A*x-b')` vyhodnotí řetěz a provede
4	`feval(F,x1,.,xn)`	vyvolá vyhodnocení funkce uložené v M-souboru o jménu definovaném v řetězové (textové) proměnné `F`; funkce `F` je závislá na vstupních argumentech $x_1, .., x_n$.

Tabulka 4.3: Makropříkazy

Poznámka:

Sám základní modul programu je `skript`em. Obvykle se rovněž člení na logické celky, které rovněž mohou být `skripty`. V žádném případě však nelze doporučit vyvolávání `skriptů` z cyklů nebo funkcí. Zatímco funkce se při prvním vyvolání přeloží do jakéhosi metajazyka a její další vyvolání již probíhá rychle, skript se *vždy* pouze interpretuje. To znamená, že se i při opakovaném volání každá řádka vždy znovu podrobuje lexikální a syntaktické analýze, což chod programu značně zpomaluje. Tato skutečnost však nevadí u `skriptů` volaných pouze jedenkrát.

Typy `skript` a `funkce` jako M-soubory mají svá jména daná uživatelem. Naproti tomu `eval` a `feval` jsou dvě jména funkcí MATLABu, která se přímo zapisují do uživatelova programu. Poslední příkaz se užívá obvykle ve funkcích, u nichž některý ze vstupních argumentů byl jménem funkce:

Příklad

```
function ren(fold,fnew)  %          Rename a file
%        **************
%   fold   old name of the file
%   fnew   new name of the file

if isstr(fold) & isstr(fnew)
   eval(['!ren ',fold,' ',fnew]);
else
   error('Wrong arguments in REN');
   pause
end
```

Kapitola 5

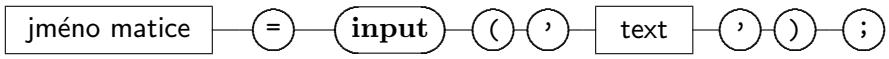
Příkazy vstupu a výstupu

Slouží ke spojení MATLABu s vnějším světem přes běžné periferie počítače - klávesnici, monitor, disky, zvukovou kartu apod. Kromě dále uvedených příkazů `input`, `load` a `save` existuje ještě řada příkazů vstupu a výstupu umožňující práci s informací na úrovni až bajtů. Zde se však budeme zabývat jen těmi nejjednoduššími příkazy.

5.1 Příkazy vstupu:

1. **Základní způsob** komunikace uživatele s MATLABem je přímé vkládání libovolných příkazů přes klávesnici počítače.

2. **Jednoduchý způsob** vstupu malého objemu dat lze uskutečnit pomocí přiřazovacího příkazu, anebo z programu pomocí příkazu `input` o struktuře:



Obrázek 5.1: Vstup dat pomocí příkazu **input**

Jde vlastně o přiřazovací příkaz se speciálním vedlejším účinkem. Při jeho vyvolání vystoupí na obrazovce uživatelem zadaný text a program čeká na vložení dat z klávesnice.

Příklady:

```
A = input('matice[A] = ')
```
Vloží se např.:
```
[ 1 2 3
  4 5 6
  7 8 9 ]
```

```
f = input('frekv. interval = ')
```
se vstupem např.: `0:.5:20`
pro vektor frekvencí 0, 0.5, 1, ..., 20

Lze vložit i prázdnou matici `[]`. Nelze dobře editovat.

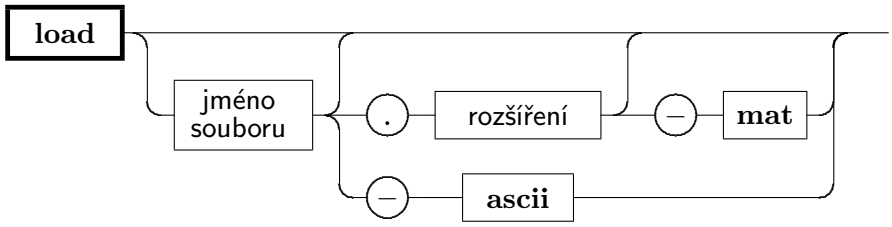
3. **Složitější způsob**, avšak jedině doporučitelný pro rozsáhlejší data, která pak lze i editovat, je vstup dat ze souborů předem připravených. Mohou to být:

(a) **M-soubory**, kde jsou data součástí přiřazovacích příkazů. M-soubor se vyvolá z vlastního programu jménem jako t.zv. `skript` (viz odstavec „Makropříkazy“).

(b) **MAT-soubory**, na nichž jsou data uložena z MATLABu pomocí příkazu `save` (viz dále), anebo jsou vytvořeny z obecných souborů, generovaných jinými programy. Mohou to být:

- tzv. ASCII flat files, tj. data v ASCII znacích po řádkách s prvky oddělenými mezerami a řádky potom znaky nové řádky (N/L, ENTER),
- binární soubory včetně fortranských neformátovaných souborů,
- výstupy z tabulkových procesorů (spreadsheet)

Každý takový soubor se čte příkazem `load` o struktuře:



Obrázek 5.2: Syntaxe příkazu **load**

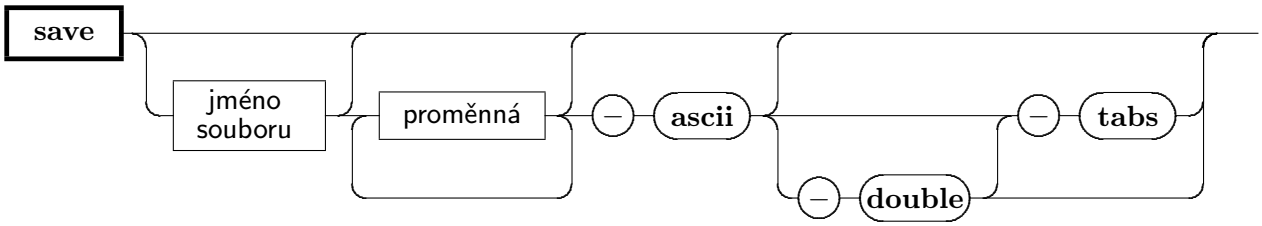
První varianta příkazu čte buď všechny proměnné z MAT-souboru vytvořeného podobným příkazem `save`, anebo i z tzv. ASCII flat files do proměnné stejného jména jako je jméno MAT-souboru.

5.2 Příkazy výstupu

Výstupní příkazy jsou podstatně rozmanitější než vstupní, neboť obsahují příkazy pro výstupy znakové a výstupy grafické.

5.2.1 Znakové výstupy

Výstupy umožňují předávat uživateli informace o běhu, ale i výsledky výpočtu na vhodnou periferii. Obvykle je výstupním prostředkem obrazovka. Jiným druhem výstupu jsou MAT-soubory získané příkazem, jehož syntaktický diagram je v obr. 5.3.



Obrázek 5.3: Syntaxe příkazu **save**

Mají strukturu vhodnou pro čtení podobným příkazem `load`. Podrobněji o struktuře `MAT`-souborů viz manuál `MATLAB`u. Přehled základních příkazů pro znakové výstupy je uveden v tab. 5.1.

Se jménem matice	přiřazovací příkaz bez „;“	vystoupí na obrazovce ve tvaru jméno matice = hodnoty prvků matice
bez jména	`disp('text')`	vystoupí text bez `ans` =
	`disp(matice)`	vytiskne matici bez `ans` =
mezery	`blanks(n)`	vystoupí `n` mezer v řádce
řádky	`blanks(n)'`	vystoupí `n` prázdných řádek

formáty	`format`
	`format short`
	`format long`
		implicitně, jako format short
		s pevnou des. tečkou s 5 číslicemi
		s pevnou des. tečkou s 15 číslicemi

	`format short e` `format long e`	v exponenciálním tvaru s 5 číslicemi v exponenciálním tvaru s 15 číslicemi
	`format hex`	v hexadecimálním tvaru
	`format +`	`+` za kladné prvky vytiskne mezeru za nulové prvky `-` za záporné prvky
	`format compact` `format loose`	potlačuje mezilehlé prázdné řádky vkládá mezilehlé prázdné řádky (implic.)
konverze skaláru na řetěz	`s=num2str(x)`	konverze z čísla na řetěz znaků s přesností cca 4 číslic
	`s=int2str(x)`	konverze do celočíselného formátu
	`s=sprintf('form', x,y,...,)`	Konverze dle požadavků s výstupem na obrazovku. Řídící řetěz `form` je formát, který obsahuje postupně: • nepovinný text, který vystoupí • `%` znak začátku definice konverze • `-` nepovinný mínus při dorazu vlevo • `m.n, m =` počet míst před tečkou `. =` desetinná tečka `n =` počet míst za tečkou • `f` fix format (s pevnou tečkou) `e` exponenciální formát `g` `e` nebo `f`, ten který je kratší. Kdekoliv ve formátu může stát příkaz odřádkování = dvojice znaků `\n`
	`s=fprintf('file', 'form',x,y,..)`	Konverze dle požadavku uživatele do souboru, pokud je dán `file` = jméno souboru; `PRN` na tiskárnu `COM1` na prvním `RS 232C`

Tabulka 5.1: Výběr funkcí pro řízení výstupu informace

Dále uvedené příklady ilustrují některé možnosti MATLABu při vstupu a výstupu informací a při operacích s vytvářením a používáním indexovaných matic, u nichž index je součástí jména.

```
%-------------------------------------------------------

function data = inp(prompt,deflt,nsp,form)
%        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% INP   Input data      v 9.0   Jan 1998
% ~~~
%   prompt  string of characters
%   deflt   default value of input data
%   nsp     number of leading spaces before prompt
%   form    format of default output

if nargin>0
  if nargin<4, form='%9.4f';
    if nargin<3, nsp=10;
      if nargin<2, deflt=[];
  end, end, end
else
  nsp=10;   prompt='input';     deflt=[];
end

str = [blanks(nsp), prompt, ' = '];

if isempty(deflt)       % in case without default value:
  while 1
    data = input(str);
    if ~isempty(data), break, end
  end
else                % in case with default value:
  if isstr(deflt)
    data = input([str, deflt, ' => '],'s');
  else
    [md,nd]=size(deflt);
    if md*nd>1
      data=input([str sprintf(form,deflt(1,1)) ' ... ' ...
                      sprintf(form,deflt(md,nd)) ' => ']);
    else
       data = input([str sprintf(form,deflt) ' => ']);
    end
  end
  if isempty(data)
    data = deflt;
  end
end

%-------------------------------------------------------
```

Použití příkazu `inp` má ve srovnání s příkazem `input` řadu výhod. Ty spočívají nejen v jednodušším zadání výzvy s lepší úpravou a volitelným odsazením od začátku řádky pro zvýraznění vkládaných dat, ale zejména v možnosti zadávat implicitní hodnotu vkládané informace jako nabídku, kterou uživatel buď může potvrdit klávesou `ENTER`, anebo přepsat novou hodnotou. To se ocení jak při ladění tak i při variantních výpočtech.

Další příklad umožňuje ukládat matice matic pomocí MATLABu i nižších verzí než 5.x. Jde vlastně o jakési fiktivní indexování submatic v matici. Od verze 5.x je snazší pracovat s vícerozměrnými strukturami.

```
%-------------------------------------------------------------

for k= 1:5
   for mx = ['A' 'B']
      eval(sprintf('%s%i = rand(5);',mx,k)) %             bez tisku
   end % mx
   eval(sprintf('C%i = A%i*B%i',k,k,k)) %                 s tiskem
end % k

%-------------------------------------------------------------
```

V posledním příkladu se ve vnitřním cyklu `mx` vytvoří vždy dvě matice řádu 5 o jménech An a Bn, kde n je postupně 1, 2, 3, 4, a 5, naplní se pseudonáhodnými čísly a nakonec se součin těchto matic uloží do matice Cn.

```
%-------------------------------------------------------------

% Matrix to string conversion
% ~~~~~~~~~~~~~~~~~~~~~~~~~~
%
% s = sprintfm(format, mx)
%     s       output string
%     format  format of an output in the form '%m.n*', where
%             * should be substituted by conversion code
%             d, e, f, g, etc.
%     mx      name of a matrix to be converted

function s=sprintfm(par1, par2)
%        ~~~~~~~~~~~~~~~~~~~~~
if nargin==2
   format=[' ' par1]; mx=par2;
else
   format=[' %9.4f']; mx=par1;
end
nrow=size(mx,1);
s=sprintf(format, mx');
s((1:nrow)*length(s)/nrow+1)=10;%   put N/L
s(1)='';

%-------------------------------------------------------------
```

Konverze pomocí procedury `sprintfm` je velmi rychlá, protože pouze nahradí znak „mezera“ znakem „nová řádka“. Výsledný řetěz je však řádka znaků! Používá se zejména pro výstup matic v předepsaném formátu současně s textem za pomoci příkazu `disp`. Tento příkaz má jediný argument – matici, ať již čísel, nebo znaků. Jakmile se mají tisknout řetězy i čísla současně, je zapotřebí zadat pouze matici znaků, jako je tomu v následujícím příkladu:

```
disp(['A = ', lines(2), sprintfm('%8.3',A)]) % Výstup na obrazovku
```

Povel `lines` užitý v příkladu následuje. Generuje rovněž řádku znaků „nová řádka“, a proto ho lze kombinovat s libovolnými řádkovými řetězy.

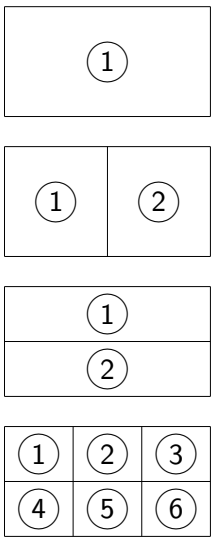
```
%-------------------------------------------------------------

% LINES     Output n lines
%   n   number of lines to be output

function s=lines(n)
%        ~~~~~~~~~~
s=10*ones(1,n);%        Char 10 = NL

%-------------------------------------------------------------
```

5.2.2 Grafické výstupy

Grafické výstupy se programují relativně jednoduše s možností využití automatického měřítkování. Využívá se k tomu 5 typů příkazů:

Rozdělení obrazovky

Obrazovku lze rozdělit na pole do tvaru matice pomocí příkazu `subplot(mnp)` nebo `subplot(m,n,p)`, kde `m, n, p` jsou celá čísla. Jím uživatel vyjadřuje, který diagram (v pořadí `p`-tý, čítaný po řádkách) z matice diagramů o `m` „řádcích“ a `n` „sloupcích“ se má vynést.

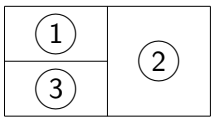



```
subplot(1,1,p) %     p = 1 (implicitně)
subplot(1,2,p) %     p = 1; 2
subplot(2,1,p) %     p = 1; 2
subplot(2,3,p) %     p = 1; 2; 3; 4; 5; 6
```

Obrázek 5.4: Grafické výstupy

V některých případech lze okna rozdělit i nepravidelně, jak je to provedeno v příkladě uvedeném na obr. 5.5. Nejdříve je však bylo zapotřebí zahájit příkazem `subplot(m,n,1)`.

Příklad:




```
subplot(2,2,1), plot(x1,y1), ...
subplot(2,2,3), plot(x2,y2), ...
subplot(1,2,2), plot(x3,y3), ...
```

Obrázek 5.5: Nepravidelné rozdělení grafického okna

Řízení obrazovky

Existuje rozdíl mezi řízením zobrazení u verzí `MATLAB`u pracujících pod `DOS`em (verze do 3.x) a pod Windows (od verze 4.0). Rozdíl je způsoben jinými vyjadřovacími prostředky obou prostředí. Pod `DOS`em existují dvě obrazovky, jedna znaková a druhá grafická. Ve Windows je k dispozici libovolný počet oken. Zacházení s nimi je zcela podřízeno tomuto prostředí. V tabulce tab. 5.2 jsou uvedeny funkce, které existují v obou verzích, ale v prostředí Windows

jsou někdy mnohem rozvinutější. Jejich podrobný popis však vybočuje z rámce této stručné informace.

nastavení kurzoru na počátek obrazovky	`home`	bez vynulování obrazovky
nuluj okno znakové grafické	`clc` `clg`	**cl**ear **c**haracter **cl**ear **g**raphic
zobraz grafické okno zobraz znakové okno	`shg` `lib. klávesa`	**sh**ow **g**raphic screen
ponechání starých grafů i s měřítky, popisem ap. (držení)	`hold`	`hold on` zapni držení `hold off` vypni držení `hold` přepíná

manuální měřítkování os	`axis(xy)`	`xy` je vektor o 4 prvcích
		`xmin xmax ymin ymax`
		`axis` samotná přepíná `on-of`

menu s výběrem	`menu('hlavička','1.varianta',...,'n-tá')`	

Tabulka 5.2: Funkce pro řízení zobrazení

Poznámka: Protože samotný příkaz `hold` pouze přepíná při každém vyvolání stavy držení resp. uvolnění grafického okna (obrazovky), je lépe užívat příkaz `hold on` – „podrž obraz“ a `hold off` – „uvolni obraz“, které jsou jednoznačné.

<table>
<tr><td rowspan="6">lineární</td><td rowspan="4">data
Re</td><td>plot(y)</td><td colspan="3">y jako funkce x = [1,2,3,...size(y)]</td></tr>
<tr><td rowspan="2">plot(x,y{,symb})</td><td colspan="3">y jako funkce x</td></tr>
<tr><td>čáry</td><td>body</td><td>barvy</td></tr>
<tr><td>- plná
-- čárk.
: tečk.
-. čerch.</td><td>.
+
*
o
x</td><td>r red
g green
b blue
w white
i invisible</td></tr>
<tr><td>plot(x1,y1,x2,y2)</td><td colspan="3">více funkcí do jednoho diagramu</td></tr>
<tr><td>plot(x,Y)
plot(X,y)</td><td colspan="3">vynese sloupce matice vůči prvkům vektoru nezávisle proměnné</td></tr>
<tr><td rowspan="2">Cx</td><td>plot(x,Z)</td><td colspan="3">vynese sloupce Re Z vůči prvkům vektoru nezávisle proměnné; Z komplexní</td></tr>
<tr><td>plot(Z)</td><td colspan="3">vynese jako plot(real(Z),imag(Z))</td></tr>
<tr><td colspan="2">logaritmická</td><td>loglog(<arg>)</td><td colspan="3">arg jako u plot</td></tr>
<tr><td colspan="2">semilogaritmická</td><td>semilogx(<arg>)
semilogy(<arg>)</td><td colspan="3">osa x v logaritmické stupnici
osa y v logaritmické stupnici</td></tr>
<tr><td colspan="2">sloupkový</td><td>bar(y)
bar(Y)</td><td colspan="3">vynese sloupkový diagram vektoru y
vynese sloupkový diagram vektoru Y(:,1)</td></tr>
<tr><td colspan="2">polární</td><td>polar(theta,rho)</td><td colspan="3">theta = angle(z)
rho = abs(z), % z je komplexní vektor</td></tr>
</table>

Tabulka 5.3: Funkce pro vynášení diagramů

Řízení vynášení diagramu

Grafy se vynášejí do aktivní oblasti grafické obrazovky (okna) přidělené příkazem `subplot` pomocí příkazů uvedených v tab. 5.3. Není-li tato oblast v režimu držení (po `hold on`), vyvolání těchto příkazů způsobí vynulování dané oblasti a její přepsání novou informací. `MATLAB` zanalyzuje zobrazovanou informaci a automaticky nastaví vhodná měřítka kromě případu, kdy byl užit příkaz `axis` uvedený výše.

Je-li zapotřebí diagram doplňovat postupnými výpočty, je vhodné pro zamezení zhasínání grafické obrazovky při výpočtu zařadit za volání grafické funkce standardní příkaz `pause(0)`, který grafický obraz podrží i po dobu výpočtu.

Úpravy grafu

Uživatel má obvykle zájem graf popsat. Týká se to zejména os, ale i nadpisů a vysvětlivek. Popisy se obvykle doplňují až po nakreslení grafu. Očíslování stupnic na osách je automatické.

popisy	`title('nadpis')`	hlavička diagramu (nadpis)
	`xlabel('popis x')`	popis osy `x` (pod)
	`ylabel('popis y')`	popis osy `y` (vedle, pod sebou)
	`text(x,y,'text')`	umístí `text` ke všem bodům `x,y`
	`text(x,y,'text','sc')`	umístí `text` v rel. bodě grafu `(0<(x,y)<1)` (v. 3.x)
	`gtext('text')`	dej `text` na pozici myši
osnova	`grid`	nakresli síť u posled. grafu (i polárního)

Tabulka 5.4: Funkce pro popisování grafů

3D grafy

Příkazy uvedené v tab. 5.5 představují pouze základní množinu příkazů `MATLAB`u, které jsou k dispozici. Lze jimi vytvářet matice `(X,Y)` souřadnic pravoúhlé sítě pro rychlý výpočet funkčních hodnot `Z(X,Y)`. Implicitní hodnota `p=[37.5°, 30°]` pro `mesh(Z)`.

vytvoření sítě	`[X,Y] = meshdom(x,y)`	`x,y` = vektory souřadnic `x` a `y` `X,Y` = matice souřadnic uzlů

3D diagramy	`mesh(Z)`
	`mesh(Z,p`
	`mesh(Z,s)`
	`mesh(Z,p,s)`
		axonometrický pohled na `Z`
		`Z` = reálná matice funkčních hodnot z(x,y)
		`p = [azimut, elevace]` bod pohledu ve stupních
		`s = [sx,sy,sz]` rozměry objektu ve 3 směrech

vrstevnice 2D	`contour(Z)`
	`contour(Z,n)`
	`contour(Z,v)`
	`contour(n,x,y)`
		vrstevnice v implicitních výškách
		`n` = počet čar - vrstevnic
		`v` = vektor výšek řezů
		`x,y` = vektory specifikující osy diagramu

Tabulka 5.5: Základní funkce pro 3D grafiku

Kapitola 6

Standardní funkce

Funkce představují mohutný nástroj MATLABu. O jejich syntaxi bylo již řečeno výše. Zde uvedeme další podrobnosti o konkrétních funkcích kromě těch, o nichž již byla řeč. Kromě funkcí speciálního určení existují v MATLABu dvě skupiny funkcí – elementární a maticové.

6.1 Elementární matematické funkce

Elementární funkce jsou matematickými předpisy, které se aplikují samostatně na každý prvek (skalár) reálných matic `X, Y` nebo komplexní matice `Z`. Výsledkem je matice téhož typu jako byla matice vstupního argumentu.

Zaokrouhlování	`round(X)`	zaokrouhlení na nejbližší celé číslo
	`fix(X)`	zaokrouhlení na celé číslo bližší k nule
	`floor(X)`	zaokrouhlení na celé číslo bližší k $-\infty$
	`ceil(X)`	zaokrouhlení na celé číslo bližší k $+\infty$
Zbytek po dělení	`rem(x,y)`	zbytek po dělení: `x-y.fix(x/y)`
Racionální aprox.	`rat(X)`	aproximuje každý prvek X řetězovým zlomkem `a/b=`d_1`+1/(`d_2`+1/(...+1/`$d_{\texttt{len}}$`)...))` s implicitními `len=5` a d_k`<100`
	`rat(len,max)`	mění `len` a `max` na zadaná
	`[A,B] = rat(X)`	generuje celočíselné matice `A,B: X=A./B`

Transformace	`sign(X)`	`-1` pro záporné prvky `X`
		`0` pro nulové prvky
		`1` pro kladné prvky
		`x/abs(x)` pro komplexní `x`

	`abs(X)`	absolutní hodnota, modul `x`
	`angle (X)`	fázový úhel v radiánech, `arg(x)`
Komplexní	`real(Z)`	reálná část komplexní veličiny `Z`
	`imag(Z)`	imaginární část `Z`
	`conj(Z)`	komplexně sdružená veličina k `Z`

Trigonometrické	`sin(Z)`	sinus
	`cos(Z)`	kosinus
	`tan(Z)`	tangens
Cyklometrické	`asin(Z)`	arkussinus
	`acos(Z)`	arkuscosinus
	`atan(Z)`	arkustangens
	`atan2(X,Y)`	`atan(Y,X)` ve čtyřech kvadrantech
Hyperbolické	`sinh(Z)`	hyperbolický sinus
	`cosh(Z)`	hyperbolický kosinus
	`tanh(Z)`	hyperbolický tangens
Hyperbolometrické	`asinh(Z)`	argument sinu hyperbolického
	`acosh(Z)`	argument kosinu hyperbolického
	`atanh(Z)`	argument tangens hyperbolického
Odmocnina	`sqrt(Z)`	druhá odmocnina
Exponenciální	`exp(Z)`	exponenciála prvků matice $e^{z_{ij}}$
Logaritmické	`log(Z)`	přirozený logaritmus
	`log10(Z)`	dekadický logaritmus
Speciální funkce	`bessel(a,X)`	Besselova funkce 1. druhu, `a = Re` skalár
	`besselh(n,X)`	Hankelova funkce pro celočíselný skalár `n`
	`gamma(a)`	Gama funkce reálného skaláru `a`

Tabulka 6.1: Elementální maticové funkce

Uživatel může ve svých funkcích využívat libovolné z funkcí MATLABu. Je však zapotřebí dodržet zásadu, že všechny argumenty (vstupní i výstupní) musí být deklarovány v hlavičce funkce. Na rozdíl od jiných jazyků není nutné všechny argumenty vždy využívat při volání funkce. Pochopitelně, že tato skutečnost musí být v těle funkce ošetřena dotazy na systémové proměnné `nargin` a `nargout` obsahující počet vstupních resp. výstupních argumentů. Chybí-li některý ze vstupních argumentů, je zapotřebí ho doplnit implicitní hodnotou. Při opomenutí této zásady dojde při pokusu použít nedefinovaný argument k ukončení výpočtu s hlášením chyby. Je rovněž nezbytné, aby poslední řádka M-souboru končila znakem ENTER, i když jiné požadavky se na ni nekladou. Při opomenutí zakončení řádky se hlásí chyba bez další identifikace, takže ji lze hůře odhalit.

6.2 Maticové funkce

Na rozdíl od „prvkových“ funkcí popsaných v předešlém odstavci maticové funkce jsou mnohem složitější a jejich argumenty jsou matice. V MATLABu jsou zpracovány 3 funkce explicitně, a to `expm`, `logm`, `sqrtm`. Ostatní funkce se počítají pomocí obecného algoritmu realizovaného ve funkci o jménu `funm`.

Obecná funkce matice `funm` využívá k výpočtu výsledku Parlettovu metodu, která je dosti pracná. Protože existují jednodušší algoritmy pro výpočet maticové exponenciály, logaritmu a odmocniny, jsou zpracovány samostatně. Pro maticovou exponenciálu je k dispozici několik algoritmů včetně metody Padého aproximace s měřítkováním.

exponenciála matice	`expm(Z)`	e^Z aproximována Padého poměrem
logaritmus matice	`logm(Z)`	ekvivalentní `funm(Z,'log,')`
odmocnina matice	`sqrtm(Z)`	ekvivalentní `funm(Z,'sqrt')`
funkce matice	`funm(Z,'f')`	obecná funkce matice definovená jménem elementární funkce `'f'` Př.: `funm(Z,'sin')`
maticový polynom	`polyvalm(c,A)`	hodnota maticového polynomu o koeficientech ve vektoru koeficientů `c` pro matici `A`
charakter. polynom	`poly(A)`	koeficienty `c` charakteristického polynomu `det(`λ`I - A) =` $c_1 \lambda^n + \cdot + c_n \lambda + c_{n+1}$
Kroneckerův součin	`kron(X,Y)`	`Z(i,j)`$\otimes$`Y` pro všechna `i, j`

Tabulka 6.2: Maticové funkce

6.3 Funkce lineární algebry

U mnoha funkcí MATLABu pro lineární algebru nezáleží na tom, zda je argumentem funkce matice komplexní či reálná. Tuto skutečnost MATLAB rozezná sám a užije optimální algoritmus. Vzorovým příkladem této vlastnosti je řešení soustavy lineárních algebraických rovnic `A*x = b` pomocí příkazu `x = A\b`.

6.3.1 Rozklady matice

`LU`	`[L,U] = lu(A)`	`L` = permutovaná dolní Δ matice `U` = horní Δ matice `A = L.U`	`x = A\b`
`QR`	`[Q,R] = qr(A)`	`Q` = unitární matice `Q*Q'= I` `R` = horní Δ matice	
	`[Q,R,P] = qr(A)`	`P` = permutační matice: `A*P = Q*R` `R` = horní Δ s klesajícími prvky na diagonále	
	`qr(A)`	`R = triu(qr(A),` `R` je prac. matice	
Choleského	`U = chol(A)`	`U` = horní Δ: `A = R'*R % A` posit. def	
singulární	`s = svd(A)`	`s` = vektor singulárních hodnot `A`	
	`[U,S,V] = svd(A)`	`A = U*S*V'` `U, V` = unitární matice `S` = diagonální matice singulárních hodnot	
	`[U,S,V] = svd(A,0)`	Pro `A(m,n)` a `m>n` počítá pouze `U(1:m,1:n)` a `S(1:n,1:n)`	
Hessenberg. forma	`H = hess(A)`	`H` = Hessenbergova forma	
	`[U,H] = hess(A)`	`U` = unitární matice: `A = U*H*U'`	
Schurova forma	`T = schur(A)`	`T` = Schurova matice	
	`[U,T] = schur(A)`	`U` = unitární matice: `A = U*T*U'`	

Tabulka 6.3: Funkce lineární algebry – rozklady matic

6.3.2 Báze, nulový prostor

báze (range)	`R = orth(A)`	ortonormální báze: `R'*R = I` počet sloupců `R` = hodnost `A`
null-space	`N = null(A)`	ortonormální báze pro nulový prostor `A*N = 0`; počet sloupců `N` = nulita `A`

Tabulka 6.4: Funkce lineární algebry

6.3.3 Inverze

inverze	`X = inv(A)`	pro `(n,n) = size(A):A=L*U`
		`X = inv(U)*inv(L)`
		`det(A) = det(L)*det(U)`

pseudoinverze	`X = pinv(A)`	pro `[m,n] = size(A)` platí `[n,m] = size (X)`, `A*X*A = A`, `X*A*X = X`, `A*X = (X*A)'` s tolerancí na singulární hodnoty `tol = max(size(A))*norm(A)*eps`
	`X = pinv(A,tol)`	dtto s volenou tolerancí

Tabulka 6.5: Funkce lineární algebry – inverze

Při řešení soustav lineárních argebraických rovnic se inverzím vyhýbáme a užíváme operátory „/“ nebo „\“.

6.3.4 Problém vlastní hodnoty (EVP)

EVP	`s = eig(A)`	`s` = vektor vlastních hodnot vyhovujících rovnici `A.V = V.S` `S = diag (s)` = spektrální matice `V` = modální matice (pravostranných vlastních vektorů)
	`[V,S] = eig(A)`	
zobecněný EVP	`s = eig(A,B)`	`s` = vektor vlastních hodnot zobecněného problému `A*V = B*V*S` `S` = spektrální matice `V` = modální matice
	`[V,S] = eig(A,B)`	
	`[At,Bt,Q,Z,V] = qz(A,B)`	`At = Q*A*Z;  Bt = Q*B*Z` `Q, Z` = transformační matice `V` = matice zobecněných vlastních vektorů
vyvažování matice	`Ab = balance(A)` `[T,Ab] = balance(A)`	`Ab = T\A*T` má stejná vlastní čísla jako `A`

Tabulka 6.6: Funkce pro řešení problému vlastní hodnoty

Byla-li matice `A` před řešením `EVP` vážena, nalezneme správné vlastní vektory původní úlohy jako `V = T*Vb/T`, kde `Vb` je modální matice vyvážené matice `Ab` a `T` transformační matice z funkce `balance`.

Příklad

Výpočet úplného problému vlastních čísel:

```
function [F,G,H] = evp(C,D,E)        Úplný problém vlastních čísel
%        ********************

%  EVP:  problém                     Podmínky ortonormality
%  ----
%  a) A*X   = X*S                        Y.'*X   = I
%     Y.'*A = S*Y.'                      Y.'*A*X = S
%
%  b) P*U   = N*U*S                      T.'*N*U = I
%     T.'*P = S*T.'*N                    T.'*P*U = S
%
%  c) M*V*S^2  + B*V*S  + K*V  = 0    |  W.'*B*V + S*W.'*M*V + W.'*M*V*S = I
%     S^2*W.'*M + S*W.'*B + W.'*K = 0 | -W.'*K*V +S*W.'*M*V*S            = S
%
%  Volání :
%
%  s      =evp(A)     A je čtvercová matice, s je vektor vlast. hodnot
%  s      =evp(P,N)   P a N jsou čtvercové matice
%  s      =evp(M,B,K) M je hmotnostní, B je útlumová a K je tuhostní m.
%  [X,S]  =evp(A)     X je pravostranná modálni, S je spektrální matice
%  [U,S]  =evp(P,N)   U je pravostranná modální matice
%  [V,S]  =evp(M,B,K) V je pravostranná modální matice
%  [X,S,Y]=evp(A)     Y je levostranná  modální matice
%  [U,S,T]=evp(P,N)   T je levostranná  modální matice
%  [V,S,W]=evp(M,B,K) W je levostranná  modální matice

if nargout==1
   if nargin==1
      F=eig(C);
   elseif nargin==2
      F=eig(C,D);
   else
      F=eig(ambk(C,D,E));
   end
else
   if nargin==1
      [F,G]=eig(C);
   elseif nargin==2
      [F,G]=eig(C,D);
   else
      [F,G]=eig(ambk(C,D,E));
   end
   if nargout==3
      if nargin==1
         H=inv(F).';
      elseif nargin==2
         H=inv(D*F).';
      else
         m=1:length(C);
         H=inv([D C ; C zeros(size(C))]*F).';
         F=F(m,:);
         H=H(m,:);
      end
      P=diag(sqrt(max(H)./max(F)));
      F=F*P;
      H=H/P.';
   end
end
```

Modální matice `V` a `W` mají jednotkové normy a platí, že `W'*V = I` a `W'*A*V = S`.

Funkce `ambk` použitá v `evp` má tvar:

```
function A=ambk(M,B,K)
%        *************

%  Matice A stavovych rovnic pro diskretni mechanicky system
%  definovany maticemi   M - hmotnostni
%                        B - utlumova
%                        K - tuhostni
%  pro stavovy vektor    x = [ y ; dy/dt ]'

A = [ zeros(M)  eye(M) ; -M\[ K B ]];
```

Problém lze zadat jedním ze třech způsobů:

1. obyčejný vlastní problém matice `A`,
2. zobecněný vlastní problém matic `P` a `N`,
3. kvadratický problém s maticemi `M, B, K`

6.3.5 Pomocné funkce

stopa	trace(A)	stopa matice (součet diag. prvků)
norma matice	norm(A)	největší singulární číslo matice A
	norm(A,1)	max(sum(abs(A)))
	norm(A,2)	kvadratická norma matice A jako norm(A)
	norm(A,inf)	Čebyševova norma = max(sum(abs(A')))
	norm(A,'fro')	Frobeniova norma = sqrt(sum(diag(A'*A)))
norma vektoru	norm(v,p)	Lp-norma = sum(abs(v).^p)^(1/p)
	norm(v)	Euklidova norma = norm(v,2)
	norm(v,inf)	Čebyševova norma = max(abs(v))
	norm(v,-inf)	Čebyševova norma = min(abs(v))
determinant	det(A)	determinant čtevrcové matice
hodnost	rank(A)	hodnost matice jako počet singulárních hodnot větších než max size(A)*norm(A)*eps
	rank(A,tol)	dtto, kde místo eps je tol
podmíněnost	cond(A)	číslo podmíněnosti matice v L_2-normě = poměr největšího a nejmenšího singulárního čísla
	rcond(A)	reciproká hodnota čísla podmíněnosti matice v L_1-normě = 1 - dobře podmíněná = 0 - singulární matice

Tabulka 6.7: Pomocné funkce lineární algebry

6.4 Funkce matematické analýzy

MATLAB obsahuje velký počet funkcí z matematické analýzy. Zde je uveden jen jejich stručný výběr. Podrobnější informaci je třeba hledat v manuálech případně operativně vyvoláním M-funkce `help jméno`, kde `jméno` je jménem funkce, jejíž popis potřebujeme.

Neznáme-li jméno potřebné M-funkce, stačí u verze MATLAB 4.x vyvolat pouze funkci `help`. Nato vystoupí přehled názvů skupin funkcí, podle kterého již lze odhadnout, kde by bylo možno hledanou funkci nalézt. Potom se vyvolá `help „jméno skupiny"` s výstupem seznamu funkcí ve skupině. Příklady na použití mnohých funkcí lze nalézt v demonstračních úlohách pro vyvolání funkce `demo`. Z předložených menu lze vybírat celé oblasti zájmu. Následným studiem zdrojových M-funkcí se lze poučit o efektivním způsobu programování pomocí MATLABu.

Polynomy	`c = poly(A)`	vektor koeficientů charakteristického polynomu matice `A`
	`c = poly(r)`	vektor koeficientů polynomu daného vektorem kořenů `r`
	`r = roots(c)`	vektor kořenů polynomu daného vektorem koeficientů `c`
	`p = polyval(c,x)`	hodnota polynomu daného koeficienty `c` v `x`
	`c = polyfit(x,y,n)`	koeficienty `c` náhradního polynomu stupně `n` v sestupných mocninách `x` ve smyslu nejmenších čtverců
	`c = conv(a,b)`	koeficienty `c` součinu polynomů `A` a `B` s koeficienty `a`, `b`
	`[q,r] = deconv(b,a)`	koeficienty `q` podílu a `r` zbytku po dělení polynomů s koeficienty `b` a `a`
	`[r,p,k] = residue(b,a)`	residua`(r)` póly `(p)` přímý člen `(k)` parciálního rozkladu poměrů `B` a `A`
	`[b,a] = residue(r,p,k)`	s koeficienty `b` resp.`a`
Po částech polynomiální funkce	`P = mkpp(xb, C)`	matice `P` informaci o po částech polynomické náhradě s hranicemi úseků `xb` a koeficienty polynomů v úsecích v řádkách matice `C`
	`[xb,C,l,k] = unmkpp(P)`	dekompozice matice `P` na soubor informací o po částech polynominální funkci `l` = počet úseků, `k` = stupeň polynomů
	`v = ppval (P,x)`	hodnota po částech polynomiální funkce v `x`
	`P = spline(x,y)`	výpočet matice `P` pro užití s `ppval` atd.
Interpolace	`yi = spline(x,y,xi)`	jednorázová interpolace funkce v bodě nebo vektoru `xi`
	`yi = table1([x,Y],xi)`	lineární interpolace v tabulce pro `xi` podle rostoucí posloupnosti ve sloupci `x`
	`zi = table2(T,xi,yi)`	lineární interpolace v 2D tabulce `T` podle rostoucích `x` v jejím 1. sloupci a rostoucích `y` v její 1. řádce
	`interp`	viz odst. „Analýza signálů“

Integrace	`q = quad(F,a,b)`	aproximace integrálu funkce `F` v intervalu `(a, b)` s relativní chybou 10^{-3}
	`q = quad(F,a,b,tol)`	dtto pro `tol` danou uživatelem
	`[q,cnt] = (F,a,b,tol)`	dtto, `cnt` = počet volání funkce `F`
	`ode23(...)` `ode45(...)`	integrace obyčejných diferenciálních rovnic met. Runge-Kutta; viz `help`
Diference	`diff(X)`	výsledkem je matice 1. diferencí sloupců
	`diff(X,n)`	výsledkem je matice `n`-tých diferencí sloupců
Kořeny funkce	`roots(c)`	kořeny polynomu daného koeficienty
	`x = zeroin(F,x0,1)`	kořen funkce `F` jedné proměnné `x`, `x0` je výchozí odhad `x` `1` pro tisk mezivýsledků; jinak `0`
Minimalizace	`x = A\b`	řešení systému lineárních rovnic s $A \in R^{m,n}$ ve smyslu metody nejmenších čtverců
	`x = nnls(A,b)`	dtto s podmínkou $x \geq 0$
	`x = nnls(A,b,tol)`	dtto s uživatelovou tolerancí na nulovost `x`
	`[x,w] = nnls(A,b)`	dtto s duálním řešením `w`
	`x = fmins(F,x0)`	Nelder-Meadův algoritmus minimalizace `F` `x0` = výchozí odhad `x` `F` = jméno minimalizované funkce (řetěz)
	`x = fmins(F,x0,tol)`	dtto s uživatelovou tolerancí
	`x = fmins(F,x0,tol,1)`	dtto s výstupem mezivýsledků
	`[x,cnt] = fmins(...)`	dtto s čítáním počtu volání `F`

Tabulka 6.8: Funkce matematické analýzy

Symbol `F` v tabulce představuje řetěz se jménem funkce (tj. `M`-souboru), která se má integrovat, nebo jejíž kořen či minimum hledáme.

Příklady:

Přibližná derivace funkce `y(x)` v bodě (vektoru) `x`:

```
dydx = diff(y)./diff(x)
```

Výpočet minima funkce uložené v `M`-souboru `dolik.m`

```
xy = fmins('dolik',[x0,y0],1e-6,1),
```

Pro výpočet jednoho reálného kořene v zadaném intervalu lze použít standardní funkci `zeroin` z tab. 6.8. V dále uvedeném příkladě se počítají všechny kořeny v zadaném intervalu pomocí nové funkce `root`, v níž je kombinována metoda půlení intervalu s metodou regula falsi. V příkladu volání je funkce `F`, jejíž kořeny se hledají, zadána jménem ve formě řetězu (`'cos'` pro kosinus).

```
[x,y,n] = root('cos',0,10,1,1e-10,1e-10),
```

kde `M`-funkce `root` může mít tvar uvedený dále.

```
%-------------------------------------------------------

% ROOT.M    Všechny reálné kořeny v intervalu funkce jedné proměnné
% ******
%     fun   jméno M-funkce s hledaným kořenem
%     x     spodní mez intervalu
%     xmax  horní mez intervalu
%     dx    krok v intervalu
%     epsx  tolerance na kořen v souřadnici x
%     epsy  tolerance na kořen v souřadnici y

function [X,Y,Cnt] = root(fun,x,xmax,dx,epsx,epsy)
%        ***************************************

X=[];   Y=[];   Cnt=[];   Dx=dx;
while x<xmax %                      Cykl kořenů
  y = feval(fun,x);
  y1 = y;
  xmin = x;
  ymin = y;
  cnt=1;

  while y1*y>0 %                    Hledání změny znaménka
    x1 = x;
    y1 = y;
    x  = x + Dx;
    if x>xmax, cnt=-cnt; break, end
    y  = feval(fun,x);
    if abs(y)<abs(ymin)
      xmin=x; ymin=y;
    end
    cnt=cnt+1;
  end
  if cnt<0, cnt=-cnt; break; end

  hlv=1;
  while 1 %                         Cykl zpřesňování kořene
    if y1*y<0,
      x2=x;   y2=y;
    else
      x1=x;   y1=y;
    end
    if hlv
      dx=(x2-x1)/2; %               Půlení intervalu
    else
      dx=(x2-x1)/(y1-y2)*y1; %      Regula falsi
    end
    x=x1+dx;
    y=feval(fun,x);
    if abs(y)<abs(ymin), xmin=x; ymin=y; end
    hlv=~hlv;
    cnt=cnt+1;
    if max([abs(x2-x);dx]) < epsx, break, end
    if abs(ymin) < epsy, break, end
  end

  X=[X,xmin];
  Y=[Y,ymin];
  Cnt=[Cnt,cnt];
  x=x+Dx;
end

%-------------------------------------------------------
```

6.5 Analýza dat

Rozdělme procedury pro analýzu dat do dvou skupin – na analýzu statistických dat a analýzu signálů. Obě skupiny mají své zvláštnosti, a proto je účelné je studovat odděleně.

6.5.1 Analýza statistických dat

Statistická data uložená v maticích `X, Y` se analyzují po sloupcích. Každý sloupec, který se má podrobit analýze, musí tedy obsahovat konzistentní data.

Statistické	`mean(X)`	řádkový vektor středních hodnot matice `X`
	`median(X)`	dtto mediánů
	`std(X)`	dtto směrodatných odchylek
	`cov(X)`	dtto rozptylů
	`corr(X)`	matice korelačních koeficientů
Histogramy	`[n,x] = hist(y)`	`n` = vektor četností `x` = vektor mezí `10` třídních intervalů o šířce třídy `(xmax-xmin)/10`
	`n = hist(y,x)`	výpočet četností `n` při zadaných mezích `x` dat `y`
	`[n,x] = hist(y,nc)`	dtto o `nc` třídních intervalech
	`n = hist(y,x)`	četnosti pro třídní intervaly `z,x`
	`histogram(y)`	graf histogramu `y` o `10` třídách
	`histogram(y,nc)`	dtto o `nc` třídách
Meze	`y = max(X)`	řádkový vektor maximálních hodnot ve sloupcích `X`
	`[y,ix] = max(X)`	dtto s indexy max. prvků ve vektoru `ix`
	`Z = max(X,Y)`	matice větších prvků z obou matic `X,Y`
	`Z = min(X,Y)`	stejná funkce i pro minimální hodnoty
Třídění	`Y = sort(X)`	třídí každý sloupec `X` ve vzestupném pořadí
	`[Y,I] = sort(X)`	dtto s pamatováním indexů v matici `I`
Kumulace	`y = sum(X)`	řádkový vektor se sumami prvků ve sloupcích `X`
	`y = prod(X)`	řádkový vektor se součiny prvků ve sloupcích `X`
	`Y = cumsum(X)`	matice částečných součtů ve sloupcích `X`
	`Y = cumprod(X)`	matice částečných součinů ve sloupcích `X`

6.5.2 Analýza signálů

Pro analýzu signálů skýtá `MATLAB` bohatou škálu funkcí s nimiž lze zajistit většinu potřebných operací při zpracování experimentálních dat sejmutých na dynamických systémech. Uživatel ocení vlastnosti `MATLAB`u zejména při zpracování mnohakanálových informací, neboť např. Fourierovu transformaci lze realizovat přes všechny kanály současně. Opět však platí,

že procesy musí tvořit sloupce matice předložené k transformaci. Bohatá je i paleta funkcí pro návrh číslicových a analogových filtrů, které realizují i velmi složité výpočty potřebné při návrzích.

Číslicová filtrace	`y = filter(b,a,x)`	filtrace `Y(z) = (B(z)/A(z)).X(z)`
	`[y,yf] = = filter(b,a,x,yi)`	dtto s přihlédnutím k počátečním podmínkám `yi` a s výpočtem koncových podmínek `yf`
	`y = interp(x,n)`	převzorkování procesu `x` `n`-krát vyšší frekvencí (`n` celé)
Odstranění trendu	`y = detrend(x)`	odstraní lineární trend z dat `x` (vhodné před použitím `fft`)
Fourierova transformace `FT`	`Y = dft(X)` `X = idft(Y)`	diskrétní konečné `FT` sloupců matice `X` inverzní `dft`
	`Y = fft(X)` `X = ifft(Y)`	rychlá dft pro `length(X) = 2↑m` inverzní `fft`
	`Y = fft2(X)` `X = ifft2(Y)`	`2D fft` nad maticí `X` inverzní `fft2`
	`ys = fftshift(y)`	přesouvá střed frekvencí po `fft`, `fft2`
Z-transformace	`bilinear`	viz `help bilinear`
Konvoluce	`y= conv(h,x)`	konvoluce vektorů `h` a `x`
	`Y = conv2(H,X)`	`2D` konvoluce
	`[q,r]= deconv(b,a)`	dekonvoluce vektoru `a` z `b` dělením `q` = výsledek, `r` = zbytek
Korelace	`rxy = xcorr(x,y)`	vzájemná korelační funkce vektorů `x` a `y`
	`rxx = xcorr(x)`	autokorelační funkce vektoru `x`
	`Rxy = xcorr2(X,Y)`	`2D` korelační funkce
Spektrální analýza	`P = spectrum(x,y,m)`	průměrování spektra Welchovou metodou size(`P`) = `[m/2,5]`, kde `P(:,1) = Pxx` = autospektrální výkonová hustota `x)` `P(:,2) = Pyy` = autospektrální výkonová hustota `y)` `P(:,3) = Pxy` = vzájemná výkonová hustota `x` a`y)` `P(:,4) = Fxy` = frekvenční přenos `x` na `y)` `P(:,5) = Cxy` = koherenční funkce mezi `x` a `y`
	`P = spectrum(x,y,m,no)`	dtto s `no` body překrývání úseků
	`Pxx = spectrum(x,m)` `Pxx = spectrum(x,m,no)`	počítá pouze `Pxx` dtto s `no` body překrývání úseků
	`specplot(P,fs)`	vynese diagramy z `P` při vzorkovací frekvenci `fs`

Datová okénka n-bodová	`w = bartlett(n)`	Bartlettovo
	`w = blackman(n)`	Blackmannovo
	`w = boxcar(n)`	obdélníkové
	`w = chebwin(n,r)`	Čebyševovo se zvlněním r-decibelů
	`w = hamming(n)`	Hammingovo
	`w = hanning(n)`	Hannovo
	`w = kaiser(n,beta)`	Kaiserovo
	`w = triang(n)`	trojúhelníkové

Tabulka 6.10: Výběr funkcí pro analýzu signálů

MATLAB má i bohaté prostředky pro analýzu dynamických systémů a pro číslicovou filtraci signálů. Patří mezi ně M-funkce pro výpočty přenosů a pro konstruování známých filtrů. Jejich stručný přehled je uveden v tabulce. Podrobnější popis těchto funkcí a jejich parametrů by vybočil z proporcí této stručné informace. Zájemci o jejich využití se odkazují na manuál MATLABu, případně na nápovědu.

Přenosy	`h = freqs(b,a,w)`	frekvenční přenos `H(j`ω`) = B(j`ω`)/A(j`ω`)` pro `w =` ω, např. `w = logspace(-1,1)`
	`h = freqz(b,a,w)`	frekvenční přenos číslicového filtru pro ω z intervalu `(0,`2π`)`
	`[h,w] = freqz(b,a,n)`	frekvenční přenos na horní polovině jednotkové kružnice v `n` rovnoměrně rozložených bodech vektoru `w` ϵ `(0,`π`)`
	`[h,w] = freqz(b,a,n, 'whole')`	dtto pro celou jednotkovou kružnici `w` ϵ `(0 až 2`π`)`
Návrh číslicových filtrů viz help	`buttap`	póly analogového Butterworthova filtru
	`butter`	návrh číslicového Butterworthova filtru
	`chebap`	póly analogového Čsbyševova filtru
	`cheby`	návrh číslicového Čebyševova filtru
	`firi`	návrh FIR–filtru LP, HP, PB
	`fir2`	návrh FIR–filtru libovolného přenosu
	`remez`	návrh FIR–filtru s lineární fází
	`yulewalk`	návrh IIR–filtru nejmenšími čtverci

Tabulka 6.11: Výběr funkcí pro číslicovou filtraci

Kapitola 7

Práce s MATLABem

Předpokládá se, že MATLAB je již instalován na osobním počítači IBM PC příp. kompatibilním.

7.1 Vkládání příkazů

Pro jednotlivé operace lze užít MATLAB přímo v interpretačním režimu, např. i jako kalkulačku. Již vložené příkazy lze procházet a editovat klávesami pro pohyb kursoru (šipkami). Posloupnost vkládaných příkazů lze archivovat po vyvolání povelu `diary` ve tvaru:

```
diary jméno_souboru
```

Příkaz funguje jako tlačítkový vypínač. Při prvním vyvolání nastartuje ukládání posloupnosti vkládaných příkazů do souboru zvoleného jména, při dalším vyvolání tuto funkci přerušuje. Je proto vhodnější volat podle přání `diary on` resp `diary off`. Vzniklý ASCII-soubor lze otisknout, editovat ap. Pro vytváření dlouhých programů, nelze tento způsob doporučit.

Při programování vytváříme M-soubory nazývané „skripty“ anebo „funkce“. **Skript** je program nebo jeho úsek, který lze kdekoliv a i několikrát vyvolat jménem M-souboru, pod nímž je uložen. Všechny proměnné jsou v něm i bez explicitní definice globální. Naproti tomu **funkce** začíná slovem `function` a proměnné nedefinované jako `global`, jsou v ní lokální. K vytvoření skriptů a funkcí užíváme libovolný editor, který je uživateli nejbližší. Ten voláme systémovým příkazem z MATLABu

```
!jméno_zvoleného_editoru
```

s případným udáním jména budoucího M-souboru jako parametru. Při užívání stejného editoru většinou uživatelů, je možné k tomu užít i příkaz MATLABu.

```
edit jméno_souboru_bez_přípony_.M
```

To však lze zajistit pouze v tom případě, že byl upraven soubor `MATLAB.bat` k nasměrování na vybraný editor.

Pokud potřebujeme počítat s variantami M-souborů, je účelné příponu .M vyhradit jen právě běžící verzi a jinak ostatním verzím dávat přípony .M1, .M2,... pro účely ukládání. Pro konkrétní výpočet se zvolená varianta M-souboru překopíruje na soubor s koncovkou .m.

7.2 Využívání programů ve FORTRANu a jazyku C

Jsou případy, kdy se nevyhneme užití jiného jazyka k tvorbě programů či jejich modulů. Je pak účelné zajistit vazbu z určitého jazyka na MATLAB a naopak.

7.2.1 Vazba přes datové soubory

Je nejjednodušším spojením jiného programu s programem v MATLABu. Zajistí se to příkazy

save	jméno MAT-souboru s daty seznam matic dat k uložení
!	jméno externího exe-programu
	ten přečte data z MAT-souboru, zpracuje je a zapíše nová data na MAT-soubor
load	jméno MAT-souboru

V externím programu se pro čtení a nahrávání MAT-souborů použijí připravené podprogramy SAVEMAT.FOR a LOADMAT.FOR z knihovny MATLABu . Uživatel je přeloží současně se svým programem zapsaným ve FORTRANu. Podobné moduly existují i pro jazyk C, které mají koncovku .c. Pro jiný jazyk je zapotřebí podobné moduly sestavit podle těchto vzorů.

7.2.2 Volání MEX-souborů

Není-li možno použít výše uvedený způsob (např. pro časové prodlevy), pak jen v nejvyšší nouzi vytvoříme soubor s koncovkou .MEX z .EXE modulu původně sestaveného v jazyce FORTRAN nebo C. Pro každý z jazyků je k dispozici 8 (6) modulů pro generování MEX-souboru a 12 (9) podprogramů pro užití v programu pro přenosy různých informací. Pro tuto komplikovanost nebudeme se s vytvářením MEX-souborů dále zabývat. Případný zájemce najde podrobnosti v manuálu MATLABu ([1] nebo [2]).

Od verze 4.2 lze MATLAB dokonce využívat z programů zapsaných v jazycích FORTRAN, C, C++ i jako knihovnu dokonalých podprogramů. Protože však jde o speciální problém pro pokročilé programátory, odkazují se zájemci o využití této možnosti na prostudování manuálu MATLABu.

Kapitola 8

Závěr

V předešlých odstavcích byl učiněn pokus o přehled základních možností velice účinného programovacího prostředku - MATLABu. Jeho efektivita dále stoupá vytvářením problémově orientovaných balíků - toolboxů. Distribuovány jsou toolboxy pro automatické řízení - CONTROL, identifikaci IDENT, statistickou analýzu STATISTICS, optimalizaci OPTIM, splajny SPLINES, neuronové sítě, symbolickou matematiku, fuzzy řízení, práce v reálném čase, zpracování obrazů a mnoho dalších. Ty pochopiteně opět zvyšují účinnost práce uživatele. Místo a rozsah nedovolilo, aby o nich bylo pojednáno zde. Je snad třeba ještě poznamenat, že ne všechny toolboxy jsou použitelné na verzích 3.x.

Přes vynikající vlastnosti tohoto programového prostředku nelze říci, že by byl zcela univerzálním jazykem pro oblast maticových výpočtů. Důvodem pro toto tvrzení je skutečnost, že ne vždy lze využít speciální vlastnosti některých matic. Jako příklad lze uvést práce s řídkými maticemi. Ve verzi 3.x se s nimi muselo pracovat jako s obecnými. To mělo za následek neefektivnost ukládání těchto matic i zbytečně dlouhé výpočetní časy. Naštěstí verze 4.x umožňují pracovat s řídkými maticemi v základních maticových operacích a v několika funkcích. Bohužel mezi ně nepatří některé maticové funkce. Naštěstí verze 5.x rozšířila paletu procedur pro práci s řídkými maticemi. Rovněž symetrie matic pro paměťové a časové úspory není dosud plně využita.

MATLAB je profesionální produkt špičkové úrovně, jíž může ztěžka dosáhnout běžný programátor. Proto je účelné jeho využití ke všem úlohám, které je schopen zvládnout. I když informace uvedené v této příručce nejsou zdaleka vyčerpávající, mohou být dobrým úvodem do hlubšího studia jazyka MATLAB.

Literatura

[1] C. Moler, J.Little, S. Bangert: PC-MATLAB for MS-DOS Personal Computers, Version 3.2 - PC, 1987

[2] The MathWorks: MATLAB Manuals v. 4.2, Natick, Mass., 1994

[3] M. Balda: PC-MATLAB, příručka uživatele.

Sborník: Software pro IBM-PC. ČSVTS ÚVZÚ, ŠKODA Plzeň , 1989.