M A T L A B
Díl I. – Práce s programem

Blanka Heringová, Petr Hora

H-S 1995

Blanka Heringová
Petr Hora

MATLAB

Předmluva

Vznik této knihy spadá do počátku roku 1994, kdy jsme se začali blížeji seznamovat s novou verzí programu MATLAB, verzí 4.0, která běžela pod Windows. Předchozí verze programu určené pro DOS jsme znali, takže jsme se soustředili hlavně na novinky – grafika, vizualizace, animace, práce se soubory apod. Hned zpočátku práce s MATLABem jsme zjistili, že informace potřebné k vážné práci s programem jsou neúplné a jsou roztroušené v několika příručkách a souborech. Rozhodli jsme se proto zpracovat veškeré informace týkající se grafického systému a souborového vstupu a výstupu v přehledné a ucelené formě, při čemž jsme vycházeli z originálních příruček MATLABu (*User's Guide*, *Reference Guide*, *Release Notes*, *New Features Guide* a *Building a Graphical User Interface*). Tím vznikla v polovině roku 1994 interní zpráva našeho pracoviště, kterou jsme v září představili na semináři firmy Humusoft, s.r.o, která u nás zastupuje výrobce MATLABu, společnost The MathWorks, Inc. Jelikož byl o zprávu velký zájem, rozhodli jsme se vydat tuto knihu, která kromě údajů ve výše zmíněné zprávě obsahuje i popis prostředí a základní koncepce MATLABu. Po dlouhém hledání vhodného (tj. levného) vydavatele jsme se nakonec rozhodli, že knihu vydáme vlastním nákladem, což nám umožnilo dosáhnout podstatně nižší ceny.

Kniha je určena jak uživatelům, kteří již MATLAB znají a potřebují se dozvědět pouze odlišnosti od starých verzí pro DOS, tak novým uživatelům, kteří potřebují proniknout i do samotné filozofie tohoto programu. V současnosti již existuje MATLAB pro Windows verze 4.2. V této knize vycházíme z verze 4.0 a případné změny, kterých je mimochodem málo, jsou uvedeny ve druhém svazku u příslušných funkcí.

Kniha se skládá ze dvou dílů – *Práce s programem* a *Popis funkcí*. K rozdělení knihy do dvou svazků nás vedlo přesvědčení, že po zvládnutí základů již pravděpodobně nebudete pro práci tolik potřebovat informace ze svazku *Práce s programem*, ale budete se obracet především na druhý díl příručky (*Popis funkcí)*.

Dále jsme došli k názoru, že by bylo vhodné zpříjemnit uživatelům (a především sobě) práci s grafickým systémem pomocí nadstavby, která by interaktivně pracovala s vytvořenými grafickými objekty. Jednou z nejdůležitějších vlastností navržené grafické nadstavby E_GRAPH je její schopnost uložit obsah libovolného grafického okna, případně všech grafických oken, do souboru a následně ho načíst. Grafická nadstavba je spolu se svým manuálem na přiložené disketě.

Věříme, že grafický systém MATLABu, který nás okouzlil a jehož popisu je zde věnována značná pozornost, přitáhne pozornost uživatelů, kteří doposud využívali MATLAB především k výpočtům,

i ke grafickému zpracování jejich výpočtů, např. k vizualizaci vektorů, znázornění vrstevnic, řezům těles, animaci, práci s barvou, modelům osvětlení, perspektivnímu zobrazení, atd.

I přes naši snahu o co nejúplnější zpracování příručky se do textu mohly vloudit jisté nepřesnosti nebo v něm mohou nějaké informace chybět. Objevíte-li při práci s MATLABem nějaké nesrovnalosti vzhledem k informacím, které jsou obsaženy v této příručce, budeme rádi, pokud nás na ně upozorníte.

Doufáme, že vám tato příručka ulehčí práci s MATLABem a najde trvalé místo vedle vašeho počítače.

Plzeň, červen 1995

Blanka Heringová
Petr Hora

Obsah

1 Úvod

MATLAB je výkonné, interaktivní prostředí pro vědecké a inženýrské výpočty a vizualizaci dat. MATLAB integruje numerickou analýzu, maticové výpočty, zpracování signálů a grafiku do uživatelsky příjemného prostředí, ve kterém se problémy a řešení zapisují stejně jako v matematice – bez tradičního programování.

MATLAB je interaktivní systém, jehož základním datovým prvkem je matice, u které se nezadává rozměr. To vám umožňuje řešit mnoho numerických problémů podstatně rychleji než při použití klasických programovacích jazyků (Fortran, Basic nebo C).

Snad nejoceňovanější vlastností MATLABu je jeho snadná rozšiřitelnost, která vám umožňuje doplňovat systém o vámi napsané funkce (M-soubory) i celé aplikace. K MATLABu si můžete navíc pořídit celou řadu specificky zaměřených nadstaveb, tzv. *toolboxů*, což je kolekce M-souborů, která je určena pro řešení jistých tříd problémů. V současnosti jsou k dispozici nadstavby pro následující okruhy problémů:

- zpracování signálů,
- zpracování obrazů,
- teorie řízení,
- identifikace systémů,
- optimalizace,
- neuronové sítě,
- spliny,
- statistika,
- symbolická matematika.

MATLAB pro Windows má následující požadavky na hardware:

- procesor 386 nebo vyšší,
- matematický koprocesor 387 nebo 487 (pokud již není součástí procesoru),
- disketovou mechaniku $3^{1/2}$" (1.44 MB),
- 8 MB volného místa na disku,
- 4 MB paměti (8 MB nebo více při používání 3-D grafiky a funkcí pro zpracování obrazu)

Firma The MathWorks, Inc., autor MATLABu, poskytuje svému produktu značnou podporu. Kromě mnoha školení a prezentací vydává čtvrtletně časopis *Newsletter*, ve kterém informuje uživatele o nových produktech, jejich rozšířeních a aplikačním využití MATLABu. Dále je v provozu anonymní FTP server *ftp.mathworks.com*, který obsahuje: archiv M-souborů od uživatelů, archiv M-souborů s technickou pomocí, volně přístupné nadstavby vytvořené uživateli, popis produktů atd. Pro kontakt s firmou lze využít i e-mail:

`support@mathworks.com`	technická podpora,
`subscribe@mathworks.com`	registrace uživatelů,
`service@mathworks.com`	obnovení a přístupové kódy,
`conference@mathworks.com`	informace o konferencích MATLABu,
`info@mathworks.com`	všeobecné informace a ceník.

Zaregistrovaní uživatelé dostávají prostřednictvím e-mailu elektronický časopis, který obsahuje popis novinek, odpovědi na nejčastější dotazy a různé tipy pro co nejefektivnější práci s MATLABem.

Jelikož je komunikace přes Internet do Ameriky pro našince záležitost vyžadující značnou trpělivost, bylo u nás vytvořeno několik kopií (mirror) originálního FTP serveru. Osobně máme nejlepší zkušenosti s mirrorem na anonymním FTP serveru *novell.felk.cvut.cz*, kde se v adresáři */pub/mirrors/mathwork* nachází aktuální obsah originálního serveru *ftp.mathworks.com.*

V České republice je zástupcem společnosti The MathWorks, Inc. (výrobce MATLABu) firma Humusoft, s.r.o., u níž lze získat veškeré informace o možnostech a cenách MATLABu.

V září 1994 vzniklo České sdružení uživatelů MATLABu (CSMUG), kteří se přihlašují, odhlašují, dostávají časopis sdružení a vyměňují si své poznatky přes e-mail. Zde jsou příslušné adresy:

Přihlášení do sdružení	e-mail:	`csmug-request@vscht.cz`
	body:	`subscribe csmug`
Odhlášení ze sdružení	e-mail:	`csmug-request@vscht.cz`
	body:	`unsubscribe csmug`
Nápověda	e-mail:	`csmug-request@vscht.cz`
	body:	`help`
Příspěvky do zpravodaje	e-mail:	`csmug-submit@vscht.cz`
Otázky a odpovědi pro diskusi	e-mail:	`csmug@vscht.cz`
Editor časopisu (Doc. A. Procházka)	e-mail:	`prochaz@vscht.cz`

2 Popis prostředí

2.1 Příkazové okno a jeho menu

Příkazové okno slouží k zadávání jednotlivých příkazů MATLABu a tvoří tak rozhraní mezi uživatelem a MATLABem. Kdykoliv MATLAB zobrazí výzvu (>>), můžete zadat libovolný příkaz. Menu příkazového okna má tyto položky:

File Pod touto položkou jsou zahrnuty příkazy pro vytváření, otvírání a pouštění M-souborů, vytváření nových grafických oken, ukládání pracovního prostoru, tisk, nastavení tiskárny a ukončení práce s MATLABem.

Edit Pod touto položkou jsou zahrnuty příkazy pro kopírování zvoleného textu z příkazového okna MATLABu do schránky a načítání obsahu schránky do příkazového okna.

Options Pod touto položkou jsou zahrnuty příkazy pro formátování výstupu do příkazového okna, nastavení barvy a fontů příkazového okna a volba editoru M-souborů.

Windows Pod touto položkou je seznam všech oken MATLABu. Volbou položky se přepnete do nového aktivního okna.

Help Pod touto položkou jsou zahrnuty příkazy pro nápovědu MATLABu.

2.2 Grafické okno a jeho menu

MATLAB zobrazuje veškeré grafické výstupy v jednotlivých grafických oknech. Grafická okna mají své vlastnosti, které můžete modifikovat příkazy a funkcemi MATLABu (viz příkaz `figure` ve druhém dílu, *Popis funkcí*). Počet grafických oken není nijak omezen. Menu grafického okna má tyto položky:

File Pod touto položkou jsou zahrnuty příkazy pro vytváření nových grafických oken, tisk, nastavení tiskárny a ukončení práce s MATLABem.

Edit Pod touto položkou jsou zahrnuty příkazy pro kopírování obsahu grafického okna do schránky ve formátu Windows Metafile nebo Windows Bitmap a vyčištění grafického okna.

Windows Pod touto položkou je seznam všech oken MATLABu. Volbou položky se přepnete do nového aktivního okna.

Help Pod touto položkou jsou zahrnuty příkazy pro nápovědu MATLABu.

2.3 Použití schránky Windows

Informace mezi MATLABem a ostatními aplikacemi mohou být přenášeny přes schránku Windows (clipboard).

K přenesení grafu z grafického okna do jiné aplikace použijte posloupnost kláves **Alt-PrintScreen**. Stiskem této posloupnosti se zkopíruje obsah aktivního okna do schránky. Potom aktivujte jinou aplikaci a v jejím menu **Edit** zvolte příkaz **Paste**, který přenese obsah schránky do aktivního okna aplikace.

Jinou možností jak přenést graf z grafického okna do jiné aplikace je použít příkazů v menu **Edit** grafického okna MATLABu – **Copy to Bitmap** a **Copy to Metafile**.

Pro zavedení textových dat z jiných aplikací musíte nejprve vyříznout nebo zkopírovat požadovaná data aplikace do schránky. Předpokládejme, že jste zkopírovali do schránky několik hodnot z tabulkového kalkulátoru. Abyste načetli tyto hodnoty do proměnné `Q`, postupujte následovně:

1. V MATLABu zadejte

```
Q = [
```

2. V menu **Edit** zvolte příkaz **Paste**. Obsah schránky je přenesen do příkazového okna za levou otevírací hranatou závorku.

3. Zadejte pravou zavírací hranatou závorku a středník a stiskněte **Enter**.

2.4 Vyvolávání běžných dialogových boxů

MATLAB pro Windows vám umožňuje vyvolávat některé běžné předdefinované dialogové boxy Windows, kterými můžete provádět následující funkce:

- nastavovat fonty u objektů osy a text,
- vybírat soubory,
- volit barvy.

2.4.1 Dialogový box pro výběr fontu

Dialogový box pro výběr fontu vám umožňuje určit a aplikovat zvolený font na textový objekt nebo popis os. Dialogový box se vyvolá příkazem `uisetfont`. Podrobné informace o parametrech tohoto příkazu získáte zadáním příkazu `help uisetfont`.

2.4.2 Dialogové boxy pro volbu souboru

Použitím příkazů `uigetfile` a `uiputfile` můžete vyvolat standardní dialogové boxy pro volbu souboru, ze kterého se bude číst nebo do kterého se bude zapisovat. Např.

```
[filename, path] = uigetfile('*.m', 'Ze souboru')
[filename, path] = uiputfile('*.m', 'Do souboru')
```

Podrobné informace o těchto příkazech získáte zadáním příkazu `help uigetfile` nebo `help uiputfile`.

2.4.3 Dialogový box pro výběr barvy

Dialogový box pro výběr barvy vám umožňuje zobrazit a změnit barvu grafického objektu. Dialogový box se vyvolá příkazem `uisetcolor`. Podrobné informace o parametrech tohoto příkazu získáte zadáním příkazu `help uisetcolor`.

2.5 Struktura adresářů MATLABu

Po instalaci má váš MATLABovský adresář následující soubory a podadresáře, které obsahují různé komponenty systému MATLABu:

`\BIN`	Binární soubory MATLABu
`\TOOLBOX`	Nadstavby MATLABu
`\EXTERN`	Pomocné programy pro sestavení MEX-souborů
`\GHOSTSCR`	GhostScript – program pro převod PostScriptových souborů
`MATLABRC.M`	Inicializační soubor MATLABu
`PRINTOPT.M`	Uživatelsky nastavitelné volby pro tisk

2.6 Vyhledávací cesta MATLABu

V MATLABu existuje tzv. *vyhledávací cesta*, která se používá k nalezení M-souborů. Např. pokud zadáte v příkazovém okně `lagrange`, interpret použije následující kroky k určení způsobu zpracování tohoto textového řetězce:

1. Zkontroluje, zda se nejedná o proměnnou.
2. Pokud ne, zkontroluje, zda se nejedná o vestavěnou funkci.
3. Pokud ne, zkontroluje, zda v aktuálním adresáři existuje soubor s názvem `LAGRANGE.MEX`, `LAGRANGE.DLL` nebo `LAGRANGE.M` (v tomto pořadí).
4. Pokud ne, zkontroluje, zda soubor s názvem `LAGRANGE.MEX`, `LAGRANGE.DLL` nebo `LAGRANGE.M` existuje ve vyhledávací cestě MATLABu (v tomto pořadí).

Aktuální vyhledávací cestu MATLABu si můžete zobrazit funkcí `path`:

```
>> path
```

```
MATLABPATH
c:\matlab\toolbox\local
c:\matlab\toolbox\matlab\datafun
c:\matlab\toolbox\matlab\elfun
c:\matlab\toolbox\matlab\elmat
c:\matlab\toolbox\matlab\funfun
c:\matlab\toolbox\matlab\general
c:\matlab\toolbox\matlab\color
c:\matlab\toolbox\matlab\graphics
. . .
```

Použitím příkazu `path` můžete k vyhledávací cestě MATLABu také přidat nové adresáře. Např. následující příkaz způsobí, že MATLAB bude prohledávat adresář `C:\MFILES` před ostatními adresáři ve vyhledávací cestě MATLABu (ale ne před aktuálním adresářem):

```
path('C:\MFILES', path)
```

Pokud chcete připojit adresář `C:\MFILES` nakonec vyhledávací cesty, použijte příkaz

```
path(path, 'C:\MFILES')
```

Počáteční vyhledávací cesta MATLABu je umístěna v souboru `matlabrc.m`, který je vytvořen během instalace a mění se při každé následující instalaci nadstaveb.

3 Základní koncepce

3.1 Vstup matic

MATLAB pracuje v podstatě s jedním typem dat a tím je matice (reálná nebo komplexní); tj. všechny proměnné jsou matice. V některých situacích jsou matice řádu jedna interpretovány jako skaláry a matice s jedním řádkem nebo sloupcem jako vektory.

Matice může být do MATLABu zavedena několika způsoby:

- napsána jako seznam prvků,
- vygenerována příkazem nebo funkcí,
- vytvořena v souboru lokálním editorem a načtena,
- načtena z externího datového souboru nebo aplikace.

Jazyk MATLABu neobsahuje žádný příkaz pro nastavení dimenze nebo typu matice. Potřebnou paměť alokuje MATLAB automaticky až do velikosti využitelné na konkrétním počítači.

Nejjednodušším způsobem zadání malých matic je napsat matici jako seznam prvků za dodržení následujících pravidel:

- Jednotlivé prvky matice v řádce oddělit mezerou nebo čárkou.
- Jednotlivé řádky matice oddělit středníkem nebo znakem konce řádky.
- Celý seznam vložit mezi hranaté závorky.

Např. zadáním příkazu

```
A = [1 2 3; 4 5 6; 7 8 9]
```

nebo

```
A = [1 2 3
4 5 6
7 8 9]
```

obdržíme výstup

```
A =
1 2 3
4 5 6
7 8 9
```

MATLAB uloží matici `A` pro pozdější využití.

Matice můžete také zadat načtením souborů s příponou `'.m'`. Pokud soubor `gena.m` obsahuje následující tři řádky textu

```
A = [1 2 3
4 5 6
7 8 9]
```

pak příkaz

```
gena
```

načte soubor a vytvoří matici `A`.

Matice vytvořené při předchozích běžích MATLABu nebo matice vytvořené jinými programy mohou být načteny příkazem `load` nebo funkcí `fread`. Viz kapitola *Vstupy a výstupy*.

3.2 Prvky matice

Prvky matice mohou být libovolné výrazy MATLABu; např. příkaz

```
x = [-1.3 sqrt(3) (1+2+3)*4/5]
```

vytvoří řádkový vektor, který se vytiskne jako

```
x =
-1.3000 1.7321 4.8000
```

Jednotlivé prvky matice mohou být zpřístupněny indexy uvnitř kulatých závorek. Pokračujeme-li v příkladu, potom příkaz

```
x(5) = abs(x(1))
```

vytvoří řádkový vektor

```
x =
-1.3000 1.7321 4.8000 0 1.3000
```

Povšimněme si, že se dimenze vektoru `x` automaticky zvětšila, aby obsáhla nový prvek, a že nedefinované mezilehlé prvky jsou nastaveny na nulu.

Velké matice můžete vytvářet pomocí malých matic, na které pohlížíte jako na prvky. Např. přidáme nový řádek k matici `A`:

```
r = [10 11 12];
A = [A; r]
```

Výsledkem je

```
A =
1 2 3
4 5 6
7 8 9
10 11 12
```

Naopak malé matice můžete z velkých matic vyjmout použitím dvojtečky. Např.

```
A = A(1:3,:);
```

vybere první tři řádky a všechny sloupce současné matice `A` a vrátí původní matici `A`. Podrobný popis použití dvojteček je v kapitole *Manipulace s vektory a maticemi*.

3.3 Příkazy a proměnné

MATLAB je *výrazový* jazyk; výrazy, které napíšete, jsou interpretovány a vyhodnoceny. Příkazy MATLABu jsou obvykle ve tvaru

proměnná = výraz

nebo jenom

výraz

Výraz je obvykle složen z operátorů, funkcí a jmen proměnných. Výsledkem vyhodnocení výrazu je matice, která se zobrazí na obrazovce a je přiřazena do proměnné. Pokud název proměnné a rovnítko chybí, je automaticky vytvořena proměnná `ans` (answer), do které je výsledek uložen. Např. výraz

```
1900/81
```

vytvoří

```
ans =
23.4568
```

Příkaz je obvykle ukončen znakem konce řádky (EOL), který je vyvolán stiskem tlačítka **ENTER**. Je-li posledním znakem příkazu středník, je potlačeno zobrazení výsledku. Toto se používá hlavně v M-souborech a v situacích, kde je lokální výsledek značně rozsáhlý nebo nezajímavý. Např.

```
p = conv(r,r);
```

konvoluje vektor `r` se sebou samým, ale nezobrazuje výsledek. Podrobný popis M-souborů je v kapitole *Skripty a funkce*.

Pokud jsou před znak konce řádky vloženy tři tečky, znamená to, že příkaz pokračuje na následující řádce. Např.

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
- 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

vyhodnotí částečný součet řady, přiřadí součet do proměnné `s`, ale nic nezobrazí. Mezery mezi znaménky `=`, `+` a `-` jsou volitelné, ale zde jsou vloženy pro zlepšení čitelnosti.

Naopak několik příkazů může být umístěno na jediné řádce, jsou-li odděleny čárkou nebo středníkem. Např.

```
a = 3; b = 10; c = a/b;
```

Názvy proměnných a funkcí musí začínat písmenem, za kterým následuje libovolný počet písmen, číslic a podtržítek. MATLAB však rozlišuje pouze prvních 19 znaků jména.

MATLAB rozlišuje malá a velká písmena. `A` a `a` nejsou stejné proměnné. Názvy všech funkcí musí být s malými písmeny. Zkuste si, co vyhodnotí MATLAB při zadání `abs(3)` a `ABS(-3)`.

Funkce může být zastíněna proměnnou, která má stejný název. Vyzkoušejte si tuto posloupnost příkazů

```
abs(-3)
abs=10
abs(-3)
```

3.4 Informace o pracovním prostoru

V předchozích příkladech jsme vytvořili proměnné, které jsou uloženy v pracovním prostoru MATLABu. Abyste tyto proměnné vypsali na obrazovku, zadejte příkaz

```
who
```

který vypíše

```
Your variables are:
A p s
ans r x
```

Z výpisu vidíte, že příklady vygenerovaly šest proměnných (včetně `ans`). K detailnějšímu výpisu proměnných slouží funkce `whos`, která v našem případě vytvoří

```
Name Size Elements Bytes Density Complex

A   3 by 3 9 72 Full No
ans 1 by 1 1  8 Full No
p   1 by 5 5 40 Full No
r   1 by 3 3 24 Full No
s   1 by 1 1  8 Full No
x   1 by 5 5 40 Full No

Grand total is 24 elements using 192 bytes
```

Každý prvek reálné matice vyžaduje 8 bytů paměti, takže matice `A` řádu tři zabírá 72 bytů a všechny proměnné dohromady 192 bytů paměti.

3.5 Stálé proměnné

Proměnná `ans` a nevypisující se proměnná `eps` mají v MATLABu speciální význam. Jsou to tzv. *stálé* proměnné, které nemůžete smazat.

Proměnnou `eps` používají některé funkce pro stanovení přesnosti výpočtu. Její počáteční hodnota je dána vzdáleností od 1.0 k nejbližšímu vyššímu FP číslu (číslo ve formátu plovoucí řádové čárky). Pro IEEE aritmetiku používanou na mnoha osobních počítačích a pracovních stanicích se `eps`=2^{-52}, což je přibližně 2.22×10^{-16}. Proměnnou `eps` můžete nastavit na libovolnou hodnotu (včetně nuly).

3.6 Nápověda

Pro nápovědu slouží v MATLABu příkazy `help` a `lookfor`. Příkaz

```
help
```

bez jakéhokoliv argumentu zobrazí seznam adresářů, které obsahují soubory související s MATLABem. Např.

```
HELP topics:

toolbox\local - Local function library.
matlab\general - General purpose commands.
matlab\ops - Operators and special characters.
. . .
```

```
matlab\matfun - Matrix functions - numerical linear algebra.
matlab\graphics - General purpose graphics functions.
matlab\color - Color control and lighting model functions.
. . .
toolbox\stats - Statistics Toolbox.
toolbox\signal - Signal Processing Toolbox.
. . .
```

Každá řádka se skládá z názvu adresáře a popisu obsahu adresáře. Některé adresáře se týkají samotného MATLABu; obsahují informace o vestavěných funkcích a funkcích dodávaných s vlastním MATLABem. Jiné adresáře jsou pro nadstavby (*toolboxy*), např. `stats` a `signal`; tyto adresáře obsahují soubor funkcí týkajících se nějaké speciální aplikační oblasti.

Seznam funkcí obsažených v daném adresáři obdržíte po zadání příkazu `help` následovaném jménem adresáře. Např.

```
help lang
```

vypíše seznam příkazů a funkcí týkajících se použití MATLABu jako programovacího jazyka a

```
help matfun
```

vypíše seznam maticových funkcí významných pro numerickou lineární algebru. Tento druhý příkaz vypíše několik desítek řádek, včetně

```
\ and / - Linear equation solution; use "help slash".
lu - Factors from Gaussian elimination.
inv - Matrix inverse.
. . .
eig - Eigenvalues and eigenvectors.
svd - Singular value decomposition.
. . .
```

Podrobné informace o jednotlivých funkcích obdržíte po zadání příkazu `help` následovaném jménem funkce. Např.

```
help eig
```

poskytne popis funkce `eig`, která počítá vlastní čísla a vlastní vektory matice.

Příkaz `lookfor` slouží k obecnému vyhledávání informací. Např. pokud hledáte možnosti výpočtu Fourierovy transformace, zadejte

```
lookfor Fourier
```

Běh příkazu trvá o něco déle, ale potom obdržíte seznam mnoha funkcí týkajících se Fourierovy transformace, včetně

```
FFT    Discrete Fourier transform.
FFT2   Two-dimensional Fast Fourier Transform.
IFFT   Inverse discrete Fourier transform.
IFFT2  Two-dimensional inverse discrete Fourier transform.
DFTMTX Discrete Fourier transform matrix.
. . .
```

Tři jiné příkazy, `who`, `what` a `which`, poskytují informace o proměnných, souborech a adresářích. Zadejte `help who`, `help what` a `help which`, abyste získali podrobný popis těchto příkazů.

3.7 Ukončení práce a uložení pracovního prostoru

Pokud chcete skončit práci s MATLABem, zadejte příkaz `quit` nebo `exit`. Při ukončení MATLAB smaže veškeré proměnné ve svém pracovním prostoru. Chcete-li si obsah pracovního prostoru uchovat pro pozdější práci s MATLABem, můžete si obsah pracovního prostoru uložit zadáním příkazu

```
save
```

Tento příkaz uloží všechny proměnné do souboru s názvem `matlab.mat`. Když je MATLAB potom později znovu zavolán, můžete zadat příkaz `load` k obnovení pracovního prostoru ze souboru `matlab.mat`.

Příkazy `save` a `load` můžete použít s jiným názvem souboru nebo k uložení pouze zvolených proměnných. Příkaz `save temp` uloží současné proměnné do souboru `temp.mat`. Příkaz

```
save temp X
```

uloží pouze proměnnou `X`, zatímco

```
save temp X Y Z
```

uloží `X`, `Y` a `Z`.

`load temp` načte všechny proměnné ze souboru `temp.mat`. `load` a `save` mohou také načítat a ukládat textové datové soubory; viz podrobnosti v kapitole *Vstupy a výstupy*.

3.8 Čísla a aritmetické výrazy

MATLAB používá obvyklý desítkový zápis čísel s volitelnou desetinnou tečkou, kladným a záporným znaménkem. Za číslo můžete připojit exponent nebo komplexní jednotku. Příklady přípustných čísel jsou

```
3 -99 +0.0001
9.6397238 1.60210E-20 6.02252e23
2i -3.14159i 3e5i
```

POZOR, na omylem vložené mezery před a za symbolem exponentu. Zkuste si, jak bude MATLAB reagovat při vyhodnocování následujících příkazů: `V=[0 1e2]`, `V=[0 1e 2]` a `V=[0 1 e2]`. Náležitou pozornost věnujte také mezerám před a za symbolem imaginární jednotky. Ověřte si výsledky těchto přiřazovacích příkazů: `V=[1 2+3i 4]`, `V=[1 2 +3i 4]`, `V=[1 2+ 3i 4]` a `V=[1 2+3 i 4]`.

Relativní přesnost čísel je `eps`, což je asi 16 desetinných čísel na počítačích s IEEE aritmetikou. Rozsah je zhruba od 10^{-308} do $10^{3}08$.

Výrazy můžete sestavovat pomocí obvyklých aritmetických operátorů a pravidel o prioritě operací:

`+`	sčítání
`-`	odčítání
`*`	násobení
`/`	dělení zprava
`\`	dělení zleva
`^`	mocnění

Pro maticové operace je vhodné mít dva symboly pro dělení. Tyto operace popisuje kapitola *Maticové operace*. Skalární výrazy `1/4` a `4\1` mají stejnou hodnotu, 0.25. Ke změně pravidel o prioritě se používají standardním způsobem závorky.

MATLAB má vestavěny elementární matematické funkce, jaké naleznete na každém dobrém vědeckém kalkulátoru. Tyto funkce zahrnují `abs`, `sqrt`, `log`, `sin`, atd. Další funkce můžete snadno přidat jako M-soubory. Úplný seznam elementárních matematických funkcí je v kapitole *Matematické funkce*.

Některé vestavěné funkce jednoduše vrací běžně užívanou speciální hodnotu. Funkce `pi` vrací π, předpočtené programem jako `4*atan(1)`. Jiný způsob jak generovat π je

```
imag(log(-1))
```

Funkce `Inf`, která zastupuje *nekonečno*, se objevuje v nemnohých kalkulačních systémech nebo počítačových jazycích. Jedním ze způsobů jak vytvořit tuto hodnotu je příkaz

```
s = 1/0,
```

který vrací

```
Warning: Divide by zero
s =
Inf
```

Na strojích s IEEE aritmetikou nevede dělení nulou k ukončení programu. Vytvoří se pouze varovné hlášení a speciální hodnota `Inf`.

Proměnná `NaN` (*Not a Number*) je IEEE číslo podobné `Inf`, ale má odlišné vlastnosti. `NaN` vytvoří výpočty jako `Inf/Inf` nebo `0/0`.

3.9 Komplexní čísla a matice

MATLAB dovede pracovat i s komplexními čísly, která jsou označena speciálními funkcemi `i` nebo `j`. Někteří z vás dávají přednost

```
z = 3 + 4*i
```

zatímco jiní preferují

```
z = 3 + 4*j
```

Jiným příkladem je

```
z = r*exp(i*theta)
```

Při zadávání komplexních matic jsou vhodné dva postupy

```
A = [1 2; 3 4] + i*[5 6; 7 8]
```

a

```
A = [1+5i 2+6i; 3+7i 4+8i]
```

které vytvoří stejný výsledek.

3.10 Výstupní formát

MATLAB zobrazí výsledek jakéhokoliv přiřazení na obrazovku a přiřadí výsledek dané proměnné nebo proměnné `ans`. K řízení formátu zobrazovaných čísel slouží příkaz `format`. Příkaz `format` má vliv pouze na zobrazení matic, ne na jejich výpočet nebo uložení. (MATLAB vykonává všechny výpočty v dvojnásobné přesnosti)

Pokud jsou všechny prvky matice celá čísla, zobrazí se matice ve formátu bez desetinných teček. Např.

```
x = [-1 0 1]
```

vždy zobrazí

```
x =
-1 0 1
```

Pokud ale alespoň jeden prvek matice není celočíselný, můžeme použít několik výstupních formátů. Implicitní formát, nazývaný `short` (krátký) formát, zobrazí asi pět desetinných míst. Ostatní formáty zobrazí více desetinných míst nebo používají vědecký zápis čísel (zápis s exponentem). Jako příklad předpokládejme

```
x = [4/3 1.2345e-6]
```

Formáty a odpovídající výstupy pro tento vektor jsou:

`format short`	1.3333	0.0000
`format short e`	1.3333e+000	1.2345e-006
`format long`	1.33333333333333	0.00000123450000
`format long e`	1.333333333333333e+000	1.234500000000000e-006
`format bank`	1.33	0.00
`format hex`	3ff5555555555555	3eb4b6231abfd271
`format +`	+	+

Pokud u krátkého (`short`) nebo dlouhého (`long`) formátu je největší prvek matice větší než 1000 nebo menší než 0.001, je na celou matici při zobrazení aplikováno měřítko. Např.

```
x = 1e20*x
```

vynásobí `x` číslem 10^20 a výsledek zobrazí

```
x =
1.0e+020 *
1.3333 0.0000
```

Formát `+` je úsporný způsob zobrazení velkých matic. Symboly `+`, `-` a mezera zobrazují kladné, záporné a nulové prvky.

Konečně příkaz `format compact` potlačuje mnohé prázdné řádky ve výpisech, čímž umožňuje dostat na obrazovku více informací.

3.11 Funkce

Za svůj výkon vděčí MATLAB v mnohém značnému množství funkcí. Některé z nich jsou vnitřní, neboli vestavěné. Jiné funkce jsou v knihovnách M-souborů distribuovaných s MATLABem (nadstavby MATLABu). A další funkce pro specializované aplikace mohou být přidány uživateli nebo skupinami uživatelů. Toto je důležitá vlastnost MATLABu; každý uživatel může vytvořit funkce, které se provádějí právě tak jako vnitřní funkce vestavěné v MATLABu. Více informací o M-souborech je v kapitole *M-soubory: Skripty a funkce*.

Obecné kategorie analytických funkcí využitelných v MATLABu zahrnují

- Elementární matematické funkce,
- Speciální funkce,
- Elementární matice,
- Speciální matice,
- Rozklad matic,

- Analýza dat,
- Polynomy,
- Řešení diferenciálních rovnic,
- Nelineární rovnice a optimalizace,
- Numerická integrace,
- Zpracování signálů.

Následující kapitoly v krátkosti popíší tyto kategorie analytických funkcí. Podrobné informace o jednotlivých funkcích jsou k dispozici v nápovědě a ve druhém dílu, *Popis funkcí.*

V předchozích příkladech byly ukázány funkce, které měly pouze jeden vstupní a jeden výstupní argument. V MATLABu však můžete funkce kombinovat mnoha způsoby. Např.

```
x = sqrt(log(z))
```

znázorňuje vnořené použití dvou jednoduchých funkcí. Některé funkce MATLABu používají dva nebo více vstupních argumentů. Např.

```
theta = atan2(y,x)
```

Každý argument může být ovšem výrazem.

Některé funkce vrací dva nebo více výstupních hodnot. Výstupní hodnoty jsou ohraničeny hranatými závorkami a odděleny čárkami:

```
[V, D] = eid(A)
[y, i] = max(x)
```

První funkce vrací dvě matice, `V` a `D`, vlastní vektory resp. vlastní čísla matice `A`. Druhý příklad vrací maximální hodnotu `y` a index `i` maximální hodnoty vektoru `x`.

Funkce, které umožňují vícenásobné výstupní argumenty, ale mohou vracet i méně výstupních argumentů. Např. `max` s jedním výstupním argumentem

```
max(x)
```

vrací právě maximální hodnotu.

MATLAB nikdy nemění vstupní (pravostranné) argumenty funkce. Výstupy funkce jsou vždy vráceny ve výstupních (levostranných) argumentech.

4 Maticové operace

Maticové operace jsou základem MATLABu; kdekoli je to možné, jsou označeny obvyklým matematických způsobem omezeným pouze znakovou sadou počítače.

4.1 Transpozice matic

Transpozici matice označuje apostrof (’). Příkazy

```
A = [1 2 3; 4 5 6; 7 8 0]
B = A’
```

vytvoří

```
A =
1 2 3
4 5 6
7 8 0

B =
1 4 7
2 5 8
3 6 0
```

a

```
x = [-1 0 2]’
```

vytvoří

```
x =
-1
0
2
```

Pokud je Z komplexní matice, potom `Z’` vytvoří komplexně sdruženou transponovanou matici. To může někdy vést k neočekávaným výsledkům, pokud se neopatrně zachází s komplexními čísly. Pro nekonjugovanou transpozici použijte `Z.’` nebo `conj(Z’)`.

4.2 Sčítání a odčítání matic

Znaménka `+` a `-` označují sčítání a odčítání matic. Matice musí mít shodné dimenze. Např. s výše uvedenými maticemi `A+x` není korektní, neboť `A` má rozměr $[3 \times 3]$ a `x` $[3 \times 1]$. Avšak

```
C = A + B
```

je akceptovatelné a vytvoří

```
C =
2 6 10
6 10 14
10 14 0
```

Sčítání a odčítání je také definováno, je-li jeden operand skalár, tj. matice řádu jedna. V tomto případě je skalár přičten ke všem prvkům nebo odečten od všech prvků matice. Např.

```
y = x - 1
```

dá

```
y =
-2
-1
1
```

4.3 Násobení matic

Symbol `*` označuje násobení matic. Operace je definována, pokud vnitřní rozměry dvou operandů jsou stejné. `X*Y` se vykoná, je-li druhý rozměr `X` stejný jako první rozměr `Y`. Např. výše uvedené `x` a `y` mají stejné rozměry $[3 \times 1]$, takže výraz `x*y` není definován a výsledkem je chybové hlášení. Je však definováno několik jiných vektorových součinů, které jsou velice užitečné. Mezi nejběžnější patří vnitřní součin (skalární); tj.

```
x'*y
```

jehož výsledek je

```
ans =
4
```

Stejný výsledek dá `y'*x`. Dále existují vnější součiny, mezi nimiž je vztah transpozice:

```
x*y' =
2 1 -1
0 0 0
-4 -2 2

y*x' =
2 0 -4
1 0 -2
-1 0 2
```

Součin matice s vektorem je speciálním případem součinu matice s maticí. Např.

```
b = A*x
```

vytvoří

```
b =
5
8
-7
```

Samozřejmě skalár může násobit matici nebo může být maticí násoben:

```
pi*x

ans =
-3.1416
0.0000
6.2832
```

4.4 Dělení matic

V MATLABu existují dva symboly pro dělení matic, `\` a `/`. Je-li `A` regulární čtvercová matice, potom `A\B` resp. `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). Obecně

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

Levostranné dělení, `A\B`, je definováno, má-li `B` tolik řádek jako `A`. Pokud je matice `A` čtvercová, používá se Gaussova eliminace. Výsledkem je matice `X` se stejnými rozměry jako u matice `B`. Pokud je `A` téměř singulární, zobrazí se varovné hlášení.

Není-li matice `A` čtvercová, aplikuje se Householderova ortogonalizace se sloupcovým výběrem. Výsledkem je matice `X` o rozměrech $[m \times n]$, kde m je počet sloupců `A` a n je počet sloupců `B`.

Pravostranné dělení, `B/A`, je definováno pomocí levostranného dělení jako `B/A=(A'\B')'`.

Např. poněvadž vektor `b` byl vypočten jako `A*x`, příkaz

```
z = A\b
```

vrátí

```
z =
-1
0
2
```

4.5 Užití mocnin s maticemi

Výraz `A^p`, který má význam p-té mocniny matice `A`, je definován, je-li `A` čtvercová matice a `p` je skalár. Pokud je `p` celočíselné větší než jedna, vypočte se tento výraz opakovaným násobením. Pro jiné hodnoty `p`, výpočet vyvolá vyhodnocení vlastních čísel a vlastních vektorů. Je-li `[V,D]=eig(A)`, potom `A^p=V*D.^p/V`.

Pokud je `P` matice a `a` je skalár, výraz `a^P` se řeší opět přes vlastní čísla a vlastní vektory. Výraz `X^P`, kde jak `X`, tak `P` jsou matice, nahlásí chybu.

4.6 Transcendentní a elementární maticové funkce

MATLAB považuje výrazy jako `exp(A)` a `sqrt(A)` za prvkové operace, které jsou definovány jako operace na jednotlivých prvcích matice `A`. MATLAB ale umí vypočítat i maticové transcendentní funkce, jakými jsou maticová exponenciála, maticový logaritmus a další. Tyto speciální funkce jsou definované pouze pro čtvercové matice.

Transcendentní matematická funkce se interpretuje jako maticová, pokud je k názvu funkce přidáno písmeno `m`; např. `expm(A)` a `sqrtm(A)`. V MATLABu jsou definovány tři takovéto funkce:

`expm`	maticová exponenciála
`logm`	maticový logaritmus
`sqrtm`	maticová odmocnina

Seznam může být rozšířen přidáním dalších M-souborů nebo použitím funkce `funm`. Podrobné informace o těchto funkcích jsou ve druhém dílu, *Popis funkcí*.

Elementární maticové funkce zahrnují:

`poly`	charakteristický polynom
`det`	determinant
`trace`	stopa
`kron`	Kroneckerův tenzorový součin

Podrobnosti jsou opět uvedeny ve druhém dílu, *Popis funkcí*.

5 Prvkové operace

Výraz prvkové operace znamená aritmetické operace prováděné na prvcích matic. Pro odlišení těchto operací od operací maticových předchází příslušné operátory tečka.

5.1 Prvkové sčítání a odčítání

U sčítání a odčítání jsou maticové operace totožné s operacemi prvkovými, takže `+` a `-` může být považováno buď za maticovou nebo prvkovou operaci.

5.2 Prvkové násobení a dělení

Symbol `.*` označuje prvkové násobení. Pokud `A` a `B` jsou stejného typu, potom `A.*B` vytvoří matici, jejíž prvky jsou jednoduše součiny jednotlivých prvků matice `A` a `B`. Např.

```
x = [1 2 3]; y = [4 5 6];
```

potom

```
z = x.*y
```

vrací

```
z =
4 10 18
```

Výrazy `A./B` a `A.\B` počítají podíly jednotlivých prvků. Takže

```
z = x.\y
```

vrátí

```
z =
4.0000 2.5000 2.0000
```

5.3 Prvkové použití mocnin

Symbol `.^` označuje prvkové mocniny. Následuje několik příkladů, které využívají výše zavedené vektory `x` a `y`. Zadejte

```
z = x.^y
```

což vrátí

```
z =
1 32 729
```

Exponent může být skalár:

```
z = x.^2

z =
1 4 9
```

nebo může být skalárem báze:

```
z = 2.^[x y]

z =
2 4 8 16 32 64
```

5.4 Relační operace

Pro porovnání dvou matic shodných rozměrů existuje šest relačních operátorů.

`<`	menší než
`<=`	menší nebo rovno
`>`	větší než
`>=`	větší nebo rovno
`==`	rovno
`~=`	nerovno

MATLAB porovnává dvojice odpovídajících prvků; výsledkem je matice jedniček a nul (jednička znamená splnění podmínky, nula nesplnění podmínky). Např.

```
2 + 2 ~= 4
```

je prostě 0.

Relační operátory se hodí zvláště ke stanovení vzoru maticových prvků, které splňují dané podmínky. Např. máme magický čtverec řádu šest

```
A = magic(6)

A =
35 1 6 26 19 24
3 32 7 21 23 25
31 9 2 22 27 20
8 28 33 17 10 15
30 5 34 12 14 16
4 36 29 13 18 11
```

Magický čtverec řádu n je matice $[n \times n]$ sestavená z celých čísel od 1 do n^2, která má řádkové a sloupcové součty stejné. Budete-li na tuto matici zírat dosti dlouho, všimnete si, že prvky, které jsou dělitelné třemi, se vyskytují na každé třetí diagonále. Pro zobrazení této kuriozity zadejte

```
P = (rem(A,3) == 0)
```

Symbol `==` je operátorem rovnosti, `rem(A,3)` je matice zbytků po dělení, 0 se rozšíří na matici nul odpovídající velikosti a `P` pak vrací matici jedniček a nul

```
P =
0 0 1 0 0 1
1 0 0 1 0 0
0 1 0 0 1 0
0 0 1 0 0 1
1 0 0 1 0 0
0 1 0 0 1 0
```

Poznamenejme, že toto je příklad *řídké matice*. Většina jejích prvků jsou nuly.

Ve spojitosti s relačními operátory je velice prospěšnou funkcí funkce find, která nachází nenulové prvky v matici, což mohou být datové prvky vyhovující nějaké relační podmínce. Např. je-li `Y` vektor, potom `find(Y<3.0)` vrací vektor indexů prvků v `Y`, které jsou menší než 3.0.

Příkazy

```
i = find(Y>3.0);
Y(i) = 10*ones(i);
```

nahradí všechny prvky v `Y`, které jsou větší než 3.0, číslem 10.0. Tyto příkazy fungují i pro `Y`, které je maticí, neboť na matici lze pohlížet jako na dlouhý sloupcový vektor s jednoduchým indexováním.

Pro testování hodnot `NaN` nejsou relační operátory vhodné s ohledem na specifikaci IEEE aritmetiky. Pro testování těchto hodnot slouží funkce `isnan(X)`, která vrací jedničky v místech prvků rovnajících se hodnotě `NaN` a nuly jinde. Další užitečnou funkcí je funkce `finite(X)`, která vrací jedničky, pokud $-\infty < x < \infty$.

5.5 Logické operace

Operátory `&` resp. `|` resp. `~` jsou logické operátory logického součinu ('and') resp. logického součtu ('or') resp. negace ('not').

`C=A&B` je matice, jejíž prvky jsou jedničky, kde `A` i `B` mají nenulové prvky, a nuly, kde alespoň v jedné z matic `A` nebo `B` je nulový prvek. Matice `A` a `B` musí být stejného typu, popř. jedna z matic může být skalár.

`C=A|B` je matice, jejíž prvky jsou jedničky, kde alespoň v jedné z matic `A` nebo `B` je nenulový prvek, a nuly, kde `A` i `B` mají nulové prvky. Matice `A` a `B` musí být stejného typu, popř. jedna z matic může být skalár.

`B=~A` je matice, jejíž prvky jsou jedničky, kde `A` má nulové prvky, a nuly, kde `A` má nenulové prvky.

Ve spojitosti s logickými operátory jsou velice užitečné funkce `any` a `all`. Funkce `any(x)` vrací jedničku, pokud alespoň jeden z prvků vektoru `x` je nenulový, a jinak nulu. Funkce `all(x)` vrací jedničku, pokud všechny prvky vektoru `x` jsou nenulové, a jinak nulu. Tyto funkce jsou užitečné zejména v příkazu `if`, např.

```
if all(A<0.5)
  něco dělej
end
```

Při maticových argumentech pracuje `any` a `all` sloupcově, tj. vrací řádkový vektor s výsledky za každý sloupec. Použijete-li tyto funkce dvakrát, např. `any(any(A))`, zredukujete tím maticovou podmínku na podmínku skalární.

Relační a logické funkce v MATLABu jsou:

`any`	logické podmínky
`all`	logické podmínky
`find`	nalezení indexů logických prvků
`exist`	kontrola existence proměnných a souborů
`isnan`	detekce hodnot `NaN`
`finite`	kontrola konečnosti proměnných
`isempty`	detekce prázdných matic
`isstr`	detekce řetězcových proměnných
`isglobal`	detekce globálních proměnných
`issparse`	detekce řídkých matic

5.6 Matematické funkce

Základní matematické funkce se aplikují na každý prvek matice. Např.

```
A = [1 2 3; 4 5 6]
B = fix(pi*A)
C = cos(pi*B)
```

vytvoří

```
A =
1 2 3
4 5 6

B =
3 6 9
12 15 18

C =
-1 1 -1
1 -1 1
```

MATLAB obsahuje tyto trigonometrické funkce:

`sin`	sinus
`cos`	kosinus
`tan`	tangens
`asin`	arkussinus
`acos`	arkuskosinus
`atan`	arkustangens
`atan2`	čtyř-kvadrantový arkustangens
`sinh`	hyperbolický sinus
`cosh`	hyperbolický kosinus
`tanh`	hyperbolický tangens
`asinh`	argument hyperbolického sinu
`acosh`	argument hyperbolického kosinu
`atanh`	argument hyperbolické tangenty

MATLAB obsahuje tyto základní funkce:

`abs`	absolutní hodnota nebo modul komplexního čísla
`angle`	fáze komplexního čísla
`sqrt`	druhá odmocnina
`real`	reálná část komplexního čísla
`imag`	imaginární část komplexního čísla
`conj`	komplexně sdružené číslo
`round`	zaokrouhlení k nejbližšímu celému číslu
`fix`	zaokrouhlení na celé číslo bližší k nule
`floor`	zaokrouhlení na celé číslo bližší k $-\infty$
`ceil`	zaokrouhlení na celé číslo bližší k ∞
`sign`	funkce signum
`rem`	zbytek po celočíselném dělení
`gcd`	největší společný dělitel
`lcm`	nejmenší společný násobek
`exp`	exponenciální funkce
`log`	přirozený logaritmus
`log10`	dekadický logaritmus

Seznam některých speciálních funkcí v MATLABu:

`bessel`	Besselova funkce
`beta`	funkce beta
`gamma`	funkce gama
`rat`	racionální aproximace
`erf`	chybová funkce
`erfinv`	inverzní chybová funkce
`ellipke`	eliptický integrál prvního a druhého druhu
`ellipj`	Jacobiho eliptická funkce

Speciální funkce pracují při maticovém argumentu stejně jako základní funkce, tj. po prvcích. Podrobné informace jsou uvedeny ve druhém dílu, *Popis funkcí*.

6 Manipulace s vektory a maticemi

Indexovací schopnosti MATLABu umožňují manipulaci s řádky, sloupci, jednotlivými prvky a submaticemi matic. Indexování se provádí na vektorech, které jsou vytvářeny pomocí dvojtečkového zápisu. Vektory a indexace jsou jedněmi z nejmocnějších operací MATLABu, jimiž se snadno dociluje i dosti složitých manipulací s daty.

6.1 Vytváření vektorů

Dvojtečka je důležitým znakem v MATLABu. Příkaz

```
x = 1:5
```

vygeneruje řádkový vektor obsahující čísla od 1 do 5 s jednotkovým krokem. Tedy

```
x =
1 2 3 4 5
```

Samozřejmě můžete zvolit jiný krok než jedna. Např.

```
y = 0:pi/4:pi
```

vytvoří

```
y =
0.0000 0.7854 1.5708 2.3562 3.1416
```

Lze použít rovněž záporný krok. Např.

```
z = 6:-1:1
```

dá

```
z =
6 5 4 3 2 1
```

Dvojtečkový zápis umožňuje snadné vytváření tabulek. Abychom získali sloupcovou tabulku, je třeba transponovat řádkový vektor získaný dvojtečkovým zápisem, vypočítat sloupec funkčních hodnot a potom vytvořit matici ze dvou sloupců. Např.

```
x = (0.0:0.2:3.0)';
y = exp(-x).*sin(x);
[x y]
```

vytvoří

```
ans =
0.0000 0.0000
0.2000 0.1627
0.4000 0.2610
0.6000 0.3099
0.8000 0.3223
1.0000 0.3096
1.2000 0.2807
1.4000 0.2430
1.6000 0.2018
1.8000 0.1610
2.0000 0.1231
2.2000 0.0896
2.4000 0.0613
2.6000 0.0383
2.8000 0.0204
3.0000 0.0070
```

Jinou možností jak vytvořit vektor je použití funkce `logspace`, která vytvoří vektor s logaritmickým rozložením, nebo `linspace`, která vám umožní určit počet bodů vektoru namísto volby kroku

```
k = linspace(-pi,pi,4)

k =
-3.1416 -1.0472 1.0472 3.1416
```

6.2 Indexace

Na jednotlivé prvky v matici se můžeme odkázat pomocí jejich indexů uzavřených v kulatých závorkách. Výrazy použité na místě indexů jsou zaokrouhleny na nejbližší celé číslo. Např. mějme matici `A`

```
A =
1 2 3
4 5 6
7 8 9
```

potom příkaz

```
A(3,3) = A(1,3)+A(3,1)
```

vytvoří

```
A =
1 2 3
4 5 6
7 8 10
```

Indexem může být i vektor. Jsou-li `x` a `v` vektory, potom `x(v)` je `[x(v(1)),x(v(2)), ..., x(v(n))]`. Je-li `x` matice, umožní vektorové indexy zpřístupnit spojité či nespojité submatice. Např. předpokládejme, že `A` je matice řádu deset, potom

```
A(1:5,3)
```

specifikuje submatici $[5 \times 1]$ (sloupcový vektor), která je tvořena prvními pěti prvky třetího sloupce matice `A`.

Podobně

```
A(1:5,7:10)
```

je submatice $[5 \times 4]$, která je tvořen z prvků prvních pěti řádek a posledních čtyř sloupců.

Samotná dvojtečka na místě indexu označuje všechny odpovídající řádky nebo sloupce. Např.

```
A(:,3)
```

označuje třetí sloupec a

```
A(1:5,:)
```

označuje prvních pět řádek.

Krásně rafinovaných efektů se dosáhne použitím odkazů na submatice na obou stranách přiřazovacího příkazu. Např.

```
A(:,[3 5 10]) = B(:,1:3)
```

nahradí třetí, pátý a desátý sloupec matice `A` prvními třemi sloupci matice `B`.

Obecně, je-li `v` a `w` vektor s celočíselnými hodnotami, pak

```
A(v,w)
```

je matice získaná vybráním prvků matice `A` s řádkovými indexy ve vektoru `v` a se sloupcovými indexy ve vektoru `w`. Tak

```
A(:,n:-1:1)
```

převrací sloupce matice `A` a

```
v = 2:2:n;
w = [3 1 4 1 6]
A(v,w)
```

je legální, leč pravděpodobně ne moc často použitelná skupina příkazů.

Jedním z dalších vypečených triků, které využívají dvojtečku, je `A(:)`. `A(:)` na pravé straně přiřazovacího příkazu označuje všechny prvky matice `A` navlečené do dlouhého sloupcového vektoru. Tedy příkazy

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

vytvoří

```
A =
1 2
3 4
5 6

b =
1
3
5
2
4
6
```

Pokud již matice `A` existuje, lze `A(:)` použít i na levé straně přiřazovacího příkazu ke změně *tvaru* nebo *velikosti* matice. Potom `A(:)` označuje matici `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 šesti-prvkový řádkový vektor na matici $[3 \times 2]$:

```
A =
11 14
12 15
13 16
```

Tato operace je zahrnuta ve funkci `reshape`.

6.3 Užití logických vektorů při indexaci

K tvorbě submatic lze s výhodou použít *logických* vektorů, což jsou vektory obsahující pouze nuly a jedničky, které většinou vzniknou při aplikaci relačních operátorů. Předpokládejme, že `A` je matice $[m \times n]$ a `L` je logický vektor dimenze m. Potom

```
A(L,:)
```

specifikuje ty řádky matice `A`, kde jsou prvky vektoru `L` nenulové.

Následuje příklad, který popisuje, jak z vektoru odstranit prvky, které jsou větší než trojnásobek standardní odchylky:

```
x = x(x<=3*std(x));
```

Podobně

```
L = X(:,3)>100
X = X(L,:)
```

ponechá v matici `X` pouze ty řádky, jejichž prvek ve třetím sloupci je větší než 100.

6.4 Prázdné matice

Příkaz

```
x = []
```

přiřadí do `x` matici řádu nula. Následné použití této funkce nevyvolá chybu; pouze operace s touto maticí způsobí vznik další prázdné matice. Něco jiného je ale příkaz

```
clear x
```

který vymaže `x` ze seznamu současných proměnných. Prázdné matice existují v pracovním seznamu; mají pouze nulový řád. K otestování existence matice v pracovním prostoru slouží funkce `exist`, zatímco funkce `isempty` testuje, zda se jedná o prázdnou matici.

Také je možné vytvořit prázdný vektor. Je-li `n` menší než jedna, potom `1:n` neobsahuje žádný prvek a tak

```
x = 1:n
```

je komplikovanější cestou jak vytvořit prázdné `x`.

Důležitější využití prázdné matice je ale při odstraňování řádek a sloupců matice, což se provádí přiřazením prázdné matice rušeným řádkům či sloupcům. Tak

```
A(:,[2 4]) = []
```

smaže druhý a čtvrtý sloupec matice `A`.

Některé maticové funkce vrací po aplikaci na prázdnou matici *podezřelé* hodnoty. Jedná se zejména o tyto funkce: `det`, `cond`, `prod`, `sum` a mnohé jiné. Např.

```
prod([])

ans =
1

det([])

ans =
1

sum([])

ans =
0
```

6.5 Speciální matice

Speciální matice, které se vyskytují v lineární algebře a zpracování signálů, generují následující funkce:

`compan`	matice přidružená k charakteristickému polynomu
`diag`	diagonální matice
`gallery`	testovací matice
`hadamard`	Hadamardova matice
`hankel`	Hankelova matice
`hilb`	Hilbertova matice
`invhilb`	inverzní Hilbertova matice
`kron`	Kroneckerův tenzorový součin
`magic`	magický čtverec
`pascal`	Pascalův trojúhelník
`toeplitz`	Töplitzova matice
`vander`	Vandermondeova matice

Např. vytvořme přidruženou matici k charakteristickému polynomu $x^3 - 7x + 6$.

```
p = [1 0 -7 6];
A = compan(p)

A =
0 7 -6
1 0 0
0 1 0
```

Vlastní čísla matice `A` jsou kořeny charakteristického polynomu.

```
eig(A) =
-3.0000
2.0000
1.0000
```

Töplizova matice s neshodou na diagonále je

```
c = [1 2 3 4 5];
r = [1.5 2.5 3.5 4.5 5.5];
t = toeplitz(c,r)

Column wins diagonal conflict.
t =
1.0000 2.5000 3.5000 4.5000 5.5000
2.0000 1.0000 2.5000 3.5000 4.5000
3.0000 2.0000 1.0000 2.5000 3.5000
4.0000 3.0000 2.0000 1.0000 2.5000
5.0000 4.0000 3.0000 2.0000 1.0000
```

Další funkce generují ne tak zajímavé, ale o to užitečnější matice:

`zeros`	nulová matice
`ones`	matice jedniček
`rand`	matice náhodných čísel s rovnoměrným rozdělením
`randn`	matice náhodných čísel s normálním rozdělením
`eye`	jednotková matice
`linspace`	vektor s lineárním rozložením
`logspace`	vektor s logaritmickým rozložením

Funkce pro vytvoření jednotkové matice byla nazvána `eye`, jelikož se `I` a `i` často používají jako indexy nebo jako označení komplexní jednotky, ale jejich výslovnost je v angličtině stejná.

Skupina funkcí zahrnuje `zeros` a `ones`, které generují konstantní matice nul a jedniček, a `rand` a `randn`, které generují matice náhodných prvků s rovnoměrným nebo normálním rozdělením. Např. vytvořme náhodnou matici typu $[4 \times 3]$

```
A = rand(4,3)

A =
0.2190 0.9347 0.0346
0.0470 0.3835 0.0535
0.6789 0.5194 0.5297
0.6793 0.8310 0.6711
```

6.6 Tvorba velkých matic

Velké matice můžete tvořit z malých matic, když je vložíte do hranatých závorek. Např. je-li `A` čtvercová matice, potom

```
C = [A A'; ones(size(A)) A.^2]
```

vytvoří matici dvojnásobného řádu oproti matici `A`. Menší matice musí být rozměrově konzistentní, jinak nastane chyba.

6.7 Manipulace s maticemi

Několik funkcí slouží k rotaci, překlápění, změně tvaru nebo vyjímání určitých částí z matice.

`rot90`	rotace
`fliplr`	horizontální překlopení
`flipud`	vertikální překlopení
`diag`	vyjmutí nebo vytvoření diagonály
`tril`	dolní trojúhelníková část matice
`triu`	horní trojúhelníková část matice
`reshape`	změna tvaru
`' nebo .'`	transponování
`:`	obecné přeskupení

Např. změnit matici $[3 \times 4]$ na matici $[2 \times 6]$

```
A =
1 4 7 10
2 5 8 11
3 6 9 12

B = reshape(A,2,6)

B =
1 3 5 7 9 11
2 4 6 8 10 12
```

Funkce `diag` resp. `triu` resp. `tril` provádí přístup k diagonále resp. horní resp. dolní trojúhelníkové části matice. Např.

```
tril(A)
```

vytvoří

```
ans =
1 0 0 0
2 5 0 0
3 6 9 0
```

Velice užitečné jsou funkce `size` a `length`. Funkce `size` vrací dvojprvkový vektor obsahující informaci o typu matice. Je-li proměnná vektor, `length` vrací jeho dimenzi, pro matici vrací `max(size(V))`.

7 Příklady a triky

Příklady k procvičení a různé triky jsme původně chtěli zařadit za každou kapitolu, ale nakonec jsme se rozhodli, že pro toto důležité téma vyčleníme samostatnou kapitolu. Zařadili jsme ji na toto místo z následujících důvodů : a) jsou probrány základní kapitoly pojednávající o koncepci MATLABu, maticových a prvkových operacích a manipulacích s vektory a maticemi; b) tyto základní kapitoly obsahují mnoho nových nezvyklých konstrukcí a triků, které nacházejí široké uplatnění při programování v MATLABu, a bylo by je proto třeba poněkud více objasnit na příkladech; c) kapitoly, které následují, neobsahují *skoro žádné* záludnosti, a proto u nich nebudeme žádné zvláštní příklady uvádět.

Za každou úlohou je bezprostředně uvedeno řešení, takže si hned můžete ověřit správnost vašeho řešení. Uvedené řešení může být jedním z mnoha správných. Pokud se vaše řešení neshoduje s uvedeným, vyzkoušejte si pro ověření správnosti obě řešení přímo v MATLABu.

Pokud není v příkladech uvedeno jinak, má matice n řádků a m sloupců.

7.1 Generování posloupností

Vygenerujte posloupnost	**Řešení**

$0, 0.1, 0.2, \ldots, 1.0$	`0:0.1:1`
	nebo
	`linspace(0,1,11)`

$1, 3, 5, \ldots, 19$	`1:2:19`
$0, \pi/4, \pi/2, 3/4\pi, \pi$	`0:pi/4:pi`

$5, 4, 3, 2, 1$	`5:-1:1`
	nebo
	`linspace(5,1,5)`

$0, 1, 4, 9, 16, \ldots, 100$	`(0:10).^2`
$20, 200, 2000, 20000$	`2*logspace(1,4,4)`

7.2 Tvorba submatic

Z matice A vytvořte matici B, která bude obsahovat	**Řešení**
třetí sloupec matice `A`	`B=A(:,3)`
druhý řádek matice `A`	`B=A(2,:)`
první tři řádky matice `A`	`B=A(1:3,:)`
druhý a čtvrtý sloupec matice `A`	`B=A(:,[2 4])`
třikrát druhý sloupec matice `A`	`B=A(:,[2 2 2])`
submatici o rozměru $[2 \times 3]$, jejíž prvek $(1,1)$ bude obsahovat prvek $(2,1)$ matice `A`	`B=A([2 3],[1 2 3])`
matici `A` jako sloupcový vektor	`B=A(:)`
matici `A` s převráceným pořadím sloupců	`B=A(:,n:-1:1)` nebo `B=fliplr(A)`

Úloha

Obraťte pořadí vektoru `x` o dimenzi n.

Řešení

```
x=x(n:-1:1)
```

nebo

```
x=fliplr(x)
```

7.3 Mazání řádků a sloupců

Smažte	**Řešení**
druhý řádek matice `A`	`A(2,:)=[]`
třetí sloupec matice `A`	`A(:,3)=[]`
první, druhý a pátý sloupec matice `A`	`A(:,[1 2 5])=[]`

7.4 Náhrada sloupců a řádek

Úloha

Nahraďte třetí sloupec matice `A` typu $[m,n]$ vektorem `x` o dimenzi n.

Řešení

```
A(:,3) = x(:)
```

Úloha

Nahraďte druhý, čtvrtý a pátý řádek matice `A` prvními třemi řádky matice `B`. (Předpokládejte, že počet sloupců v `A` i `B` je stejný.)

Řešení

```
A([2 4 5],:) = B(1:3,:)
```

Úloha

Nahraďte druhý řádek matice `A` pátým sloupcem matice `B`. (Předpokládejte, že počet sloupců matice `A` je roven počtu řádek matice `B`.)

Řešení

```
A(2,:) = B(:,5)'
```

Úloha

Nahraďte druhý, třetí a čtvrtý řádek matice `A` prvním řádkem matice `B`. (Předpokládejte, že počet sloupců v `A` i `B` je stejný.)

Řešení

```
A([2 3 4],:) = B([1 1 1],:)
```

7.5 Prohazování sloupců a řádek

Úloha

Prohoďte v matici `A` druhý a pátý řádek.

Řešení

```
A([5 2],:) = A([2 5],:)
```

Úloha

Prohoďte v matici `A` první a třetí sloupec.

Řešení

```
A(:,[3 1]) = A(:,[1 3])
```

7.6 Logická pole

Úloha

Vypusťte z vektoru `x` nekladné prvky.

Řešení

```
L=x>0   % vytvoření logického pole L, které má stejnou dimenzi
        % jako x a obsahuje nuly a jedničky
x=x(L)  % aplikace logického pole L na vektor x
```

nebo zkráceně

```
x=x(x>0)
```

Úloha

Vypusťte malé prvky z vektoru `x`.

Řešení

```
x = x(abs(x)>eps)
```

Úloha

Vypusťte z matice `A` sloupce, jejichž sumy jsou menší než 10.

Řešení

```
A(:,sum(A)<10)=[]
```

7.7 Funkce any, all a find

Úloha

Nahraďte v matici `A` prvky s hodnotami 5 a 10 hodnotou -2.

Řešení

```
L = A==5 | A==10          % logické pole odpovídající podmínce
ind = find(L)             % převod na indexy
A(L) = -2*ones(size(ind)) % přiřazení nové hodnoty
```

nebo zkráceně

```
A(A==5 | A==10) = -2*ones(size(find(A==5 | A==10)))
```

Úloha

Zjistěte, ve kterém sloupci matice `A` existuje prvek větší než 20.

Řešení

```
L = A>20        % logické pole odpovídající podmínce
ind = any(L)    % sloupcové hledání nenulových prvků L
find(ind)       % převod na index hledaného sloupce
```

nebo zkráceně

```
find(any(A>20))
```

Úloha

V matici `A` změňte znaménko u prvků, které jsou větší než 4.

Řešení

```
ind = find(A>4);
A(ind) = -A(ind);
```

Úloha

Nahraďte v matici `A` prvky s hodnotami `NaN` nulami.

Řešení

```
A(isnan(A)) = zeros(size(find(isnan(A))))
```

Úloha

Zjistěte, zda v matici `A` existuje prvek větší než 20.

Řešení

```
any(any(A>20))
```

Úloha

Zjistěte, zda v matici `A` jsou všechny prvky menší než 20.

Řešení

```
all(all(A<20))
```

7.8 Triky s funkcí ones

Úloha

Vytvořte dvou-řádkovou matici `A`, jejíž první řádek bude obsahovat 10-krát hodnotu a a druhý řádek 10-krát hodnotu `b`.

Řešení

```
A = [a(1,ones(1,10)); b(1,ones(1,10))]
```

Úloha

Vytvořte matici `A` s 30 sloupci. Tyto sloupce budou tvořeny sloupcovým vektorem `x`.

Řešení

```
A = x(:,ones(1,30))
```

nebo

```
A = x(:,ones(3,10))
```

atd.

Úloha

Mějte vektor `a=[1 0 1]`. Změňte ho na vektor `b=[1 1 1 0 0 0 1 1 1]`, který vznikne opakováním každého prvku vektoru `a` třikrát.

Řešení

```
a = [1 0 1];
N = 3;
b = a(ones(1,N),:)
b = b(:)'
```

Úloha

Vytvořte tabulku malé násobilky.

Řešení

```
v = (1:10)';
V = v(:,ones(1,10))*diag(v);
```

8 Řídicí struktury

MATLAB obsahuje řídicí struktury, jaké se nacházejí ve většině programovacích jazyků. Existence těchto struktur povyšuje MATLAB z obyčejného kalkulátoru na plnohodnotný vyšší programovací jazyk.

8.1 Cyklus FOR

MATLAB má svoji vlastní verzi cyklu DO nebo FOR, který se nachází v jiných programovacích jazycích. Cyklus `for` slouží pro předem daný počet opakování příkazu nebo skupiny příkazů.

Např.

```
for i=1:n, x(i)=0, end
```

přiřadí nulu prvním `n` prvkům vektoru `x`. Je-li `n` menší než jedna, bude příkaz stále legální, ale vnitřní příkaz nebude proveden. Pokud `x` ještě neexistuje nebo má méně než `n` prvků, potom se automaticky alokuje dodatečný prostor.

Cykly můžete vnořovat; z důvodu lepší čitelnosti se obvykle odsazují. Např.

```
for i=1:m
  for j=1:n
    A(i,j)=1/(i+j-1);
  end
end
A
```

Středník ukončující vnitřní příkaz potlačuje zobrazování mezivýsledků, zatímco `A` následující po cyklech zobrazí konečný výsledek.

Pozor, každý `for` musí mít svůj `end`. Jestliže zadáte

```
for i=1:n, x(i)=0
```

systém čeká, až zadáte zbývající příkazy do těla cyklu. Nic se neděje, dokud nezadáte `end`.

Pro následující příklad předpokládejme, že

```
t =
-1
0
1
3
5
```

a že chcete vytvořit Vandermondeovu matici, což je matice, jejíž sloupce jsou mocniny vektoru `t`.

```
A =
1 -1 1 -1 1
0 0 0 0 1
1 1 1 1 1
81 27 9 3 1
625 125 25 5 1
```

Zde je nejobvyklejší řešení – dva cykly

```
n=length(t);
for j=1:n
  for i=1:n
    A(i,j)=t(i)^(n-j);
  end
end
```

Avšak následující jednoduchý cyklus s vektorovými operacemi je podstatně rychlejší a také ilustruje skutečnost, že cykly `for` mohou jít také pozpátku.

```
A(:,n)=ones(n,1);
for j=n-1:-1:1
  A(:,j)=t.*A(:,j+1);
end
```

Obecný tvar cyklu `for` je

`for v=`*výraz*
příkazy
`end`

Výraz je ve skutečnosti matice, neboť nic jiného v MATLABu neexistuje. Sloupce této matice jsou postupně přiřazovány proměnné v a následně jsou provedeny *příkazy*. Jasněji lze celou záležitost vyjádřit jako

```
E=výraz;
[m,n]=size(E);
for j=1:n
   v=E(:,j);
   příkazy
end
```

Obvykle je výraz ve tvaru `m:n` nebo `m:i:n`, což je matice s jednou řádkou, takže sloupce jsou skaláry. V tomto speciálním případě se chová cyklus `for` MATLABu jako cykly FOR a DO v jiných jazycích.

8.2 Cyklus WHILE

MATLAB má svoji vlastní verzi cyklu `while`, který umožňuje opakovat příkaz nebo skupinu příkazů v závislosti na logické podmínce. Zde je jednoduchý problém na ilustraci cyklu `while`. Jaké první celé číslo `n` má svůj faktoriál stociferný? Následující cyklus `while` ho najde. Pokud již neznáte odpověď, zadejte

```
n=1;
while prod(1:n)<1.0e100,
  n=n+1;
end
n
```

Praktičtějším výpočtem ilustrujícím použití `while` je výpočet exponenciální funkce matice; v MATLABu nazvané `expm(A)`. Jednou z možných definic exponenciální funkce je mocninná řada:

```
expm(A) = I + A + A^2/2! + A^3/3! + ...
```

Obecný tvar cyklu while je

```
while výraz
   příkazy
end
```

Příkazy se opakují tak dlouho, dokud jsou všechny prvky ve *výrazu* (výraz je matice) nenulové. *Výraz* je téměř vždy skalárním relačním výrazem, takže nenulové hodnoty odpovídají logické hodnotě TRUE. Pokud *výraz* není skalár, můžete ho redukovat funkcí `any` nebo `all`.

8.3 Příkazy IF a BREAK

Následující příklady ilustrují použití příkazu `if`. První příklad ukazuje, jak může být výpočet rozdělen na tři případy v závislosti na znaménku a paritě proměnné `n`.

```
if n<0
  A=negative(n)
elseif rem(n,2)==0
  A=even(n)
else
  A=odd(n)
end
```

Druhý příklad pojednává o vzrušujícím problému z teorie čísel. Mějme libovolné kladné celé číslo. Pokud je sudé, dělme ho dvěma. Pokud je liché, násobme ho třemi a přičtěme jedničku. Tento postup opakujme, dokud nedostaneme jedničku. Problém zní: Existuje nějaké celé číslo, pro které tento proces nikdy nekončí? Následující program ilustruje hlavně příkazy `while` a `if`. Dále ukazuje použití funkce `input`, která slouží pro vstup z klávesnice, a příkazu `break`, který přeruší provádění nejblíže nadřazeného cyklu.

```
% Klasický problém '3n+1' z teorie čísel
while 1
  n=input('Zadej n, záporné=konec ');
  if n<=0, break, end
  while n>1
    if rem(n,2)==0
      n=n/2;
    else
      n=3*n+1;
    end
  end
end
```

9 M-soubory: Skripty a funkce

MATLAB se obvykle používá v příkazovém módu; když zadáte jednořádkový příkaz, MATLAB ho okamžitě provede a zobrazí výsledky. Kromě toho může MATLAB také spouštět posloupnosti příkazů, které jsou uloženy v souborech.

Soubory, které obsahují příkazy MATLABu, se nazývají *M-soubory*, neboť mají příponu `'.m'`. Např. soubor s názvem `bessel.m` obsahuje příkazy MATLABu, které vypočtou hodnotu Besselovy funkce.

M-soubory obsahují posloupnost normálních příkazů MATLABu, které se mohou dále odkazovat na jiné M-soubory. M-soubor může volat rekursivně sám sebe. M-soubor můžete vytvořit libovolným textovým editorem.

Rozlišujeme dva typy M-souborů: *skripty* a *funkce*. *Skripty*, neboli skriptové soubory, automatizují dlouhé posloupnosti příkazů. *Funkce*, neboli funkční soubory, poskytují MATLABu rozšiřitelnost. Funkce vám umožňují přidávat nové funkce k funkcím existujícím. Za svoji oblíbenost vděčí MATLAB v mnohém právě své schopnosti vytvářet nové funkce, které řeší uživatelem specifikované problémy. *Skripty* i *funkce* jsou obyčejné textové (ASCII) soubory.

9.1 Skriptové soubory

Když je spuštěn *skript*, MATLAB jednoduše spouští příkazy, které nalezne v souboru. Příkazy ve *skriptovém souboru* operují globálně s daty v pracovním prostoru. *Skripty* jsou užitečné k provedení analýz, řešení problémů nebo konstruování dlouhých posloupností příkazů, které se interaktivně dají dělat jenom těžkopádně a zdlouhavě.

Např. předpokládejme, že soubor `fibno.m` obsahuje příkazy:

```
% M-soubor pro výpočet Fibonacciho čísel
f=[1 1]; i=1;
while f(i)+f(i+1)<1000
  f(i+2)=f(i)+f(i+1);
  i=i+1;
end
plot(f)
```

Zadáním příkazu `fibno`, MATLAB provede příkazy v tomto souboru a vypočte tak prvních šestnáct Fibonacciho čísel a vytvoří jejich graf. Po ukončení výpočtu zůstávají proměnné `f` a `i` v pracovním prostoru.

Demonstrační příklady, které se dodávají s MATLABem jsou dobrou ukázkou, jak lze skriptů použít pro řešení složitějších úloh. Zkuste si to, zadejte `demos`.

Když spustíte MATLAB, automaticky se spustí skript s názvem `startup.m`. Do něho si můžete zadat fyzikální konstanty, inženýrské konverze nebo cokoli jiného, co chcete mít předdefinováno ve vašem pracovním prostoru. Na víceuživatelských nebo síťových systémech je skript `matlabrc.m` rezervován pro systémový manažer.

9.2 Funkční soubory

M-soubor, který obsahuje slovo `function` na začátku první řádky, je *funkční soubor*. *Funkce* se liší od *skriptu* v následujícím:

- funkci mohou být předány vstupní parametry,
- ve funkci mohou být definovány proměnné, které jsou lokální,
- funkce může předat výstupní parametry.

Funkční soubory jsou významné pro rozšíření MATLABu, tj. vytvoření nových funkcí MATLABu za použití jazyka MATLABu samotného.

Zde je jednoduchý příklad. Soubor `mean.m`, dodávaný s MATLABem, obsahuje příkazy:

```
function y = mean(x)
% MEAN Average or mean value.
% For vectors, MEAN(X) is the mean value of the elements in X.
% For matrices, MEAN(X) is a row vector containing the mean value
% of each column.
%
% See also MEDIAN, STD, MIN, MAX.

% Copyright (c) 1984-94 by The MathWorks, Inc.

[m,n] = size(x);
if m == 1
  m = n;
end
y = sum(x)/m;
```

Existence tohoto souboru zároveň definuje novou funkci zvanou `mean`. Tato nová funkce `mean` se používá právě tak jako kterákoli jiná funkce MATLABu. Např. je-li `z` vektor celých čísel od 1 do 99,

```
z = 1:99;
```

pak střední hodnotu nalezneme zadáním

```
mean(z)
```

což dá

```
ans =
50
```

Zde jsou některé podrobnosti k `mean.m`:

- První řádka deklaruje název funkce, vstupní argumenty a výstupní argumenty. Bez této řádky by byl tento soubor *skriptem* a ne *funkcí*.
- Znak % označuje, že zbytek řádky je komentář, který je ignorován.
- Několik prvních komentářových řádek popisuje M-soubor a zobrazí se, když zadáte `help mean`.
- První komentářová řádka, známá jako řádka „H1“, je zahrnuta do souboru contents.m, který se nachází v každém příbuzném adresáři MATLABu, a používá ji příkaz `lookfor`.
- Proměnné `x`, `m`, `n` a `y` jsou lokální vzhledem k `mean` a neexistují v pracovním prostoru. (Pokud v pracovním prostoru existovaly před spuštěním funkce `mean`, zůstanou nezměněny.)

Nyní vytvoříme o trochu složitější verzi funkce `mean` nazvanou `stat`. Tato funkce vypočte navíc standardní odchylku:

```
function [mean,stdev] = stat(x)
[m,n] = size(x);
if m == 1
  m = n;
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m-mean.^2);
```

Funkce `stat` ilustruje možnost vrácení více výstupních argumentů.

Funkce, která vypočte hodnost matice, užívá naopak více vstupních argumentů:

```
function r=rank(X,tol)
% hodnost matice
s = svd(X);
if (nargin == 1)
  tol = max(size(X))*s(1)*eps;
end
r = sum(s>tol);
```

Tento příklad demonstruje použití permanentní proměnné `nargin` k získání počtu vstupních argumentů. Proměnná `nargout`, zde nepoužitá, obsahuje počet výstupních argumentů.

9.3 Tvorba nápovědy pro vaše M-soubory

Nápovědu pro vaše M-soubory vytvoříte zadáním jedné nebo několika komentářových řádek počínaje druhou řádkou souboru. Např. M-soubor `angle.m` obsahuje na začátku tyto řádky

```
function p = angle(h)
% ANGLE Phase angle.
% ANGLE(H) returns the phase angles, in radians, of a matrix with
% complex elements.
%
% See also ABS, UNWRAP.

% Copyright (c) 1984-94 by The MathWorks, Inc.
```

Když zadáte `help angle`, zobrazí se řádky 1 až 5. Tyto řádky jsou prvním spojitým blokem komentářových řádek. Systém nápovědy ignoruje komentářové řádky, které se objeví v souboru později po nějakém proveditelném příkazu nebo prázdné řádce.

První komentářová řádka v M-souboru musí obsahovat co nejvíce informací, neboť v některých případech se zobrazuje pouze tato první komentářová řádka (např. při použití příkazu `lookfor`).

9.4 Globální proměnné

Obvykle každá funkce MATLABu, definovaná jako M-soubor, má své vlastní lokální proměnné, které jsou oddělené od lokálních proměnných jiných funkcí a od proměnných základního pracovního prostoru. Pokud však několik funkcí popř. základní prostor deklarují název proměnné jako globální, potom společně sdílení jednu kopii této proměnné.

Globální proměnné se obvykle z důvodu lepší čitelnosti M-souborů píší velkými písmeny.

Předpokládejme, že chceme studovat vliv vazebních koeficientů α a β v Lotka-Volterrově modelu predátora a kořisti

$$\begin{aligned} \dot{y}_1 &= y_1 - \alpha y_1 y_2 \\ \dot{y}_2 &= -y_2 + \beta y_1 y_2 \end{aligned}$$

Vytvořme M-soubor `lotka.m`:

```
function yp = lotka(t,y)
% LOTKA Lotka-Volerrův model dravce a kořisti
global ALPHA BETA
yp = [y(1)-ALPHA*y(1)*y(2); -y(2)+BETA*y(1)*y(2)];
```

Potom zadejte následující příkazy:

```
global ALPHA BETA
ALPHA = 0.01
BETA = 0.02
[t,y] = ode23('lotka', 0, 10, [1;1]);
plot(t,y)
```

Deklarace globálních proměnných `ALPHA` a `BETA` a přiřazení hodnot umožní jejich použití ve funkci `lotka.m`. Globální proměnné `ALPHA` a `BETA` mohou být modifikovány interaktivně a nová řešení získáme bez editace souboru `lotka.m`.

9.5 Textové řetězce

Textové řetězce se do MATLABu zadávají v apostrofech. Např.

```
s = 'Hello'
```

vrátí

```
s =
Hello
```

Text je uložen ve vektoru, co znak to prvek. V našem případě

```
size(s)

ans =
1 5
```

oznamuje, že `s` má pět prvků (pět znaků). Znaky jsou uloženy jako ASCII hodnoty a funkce `abs` tyto hodnoty (pořadí v tabulce ASCII znaků) ukáže:

```
abs(s)

ans =
72 101 108 108 111
```

Funkce `setstr` nastaví zobrazovací mód vektoru tak, aby zobrazoval text uložený ve vektoru a ne odpovídající ASCII hodnoty. Funkce `disp` zobrazí text v proměnné, funkce `isstr` detekuje řetězce a funkce `strcmp` řetězce porovnává.

Použitím hranatých závorek můžete textové proměnné spojovat do velkých řetězců:

```
s = [s, ' World']

s =
Hello World
```

Čísla se převádí na řetězce funkcemi `sprintf`, `num2str` a `int2str`. Pro vložení popisu grafu, který obsahuje číselné hodnoty, se často přetransformovaná čísla spojují do velkých řetězců:

```
f = 70; c = (f-32)/1.8;
title(['Room temperature is ', num2str(c), ' degrees C'])
```

9.6 Funkce eval

Funkce `eval` pracuje s textovými proměnnými a patří mezi nejvýkonnější (ale také nejzákeřnější) funkce MATLABu. `eval(t)` provede vyhodnocení textu uloženého v proměnné `t`. Např. mějme proměnnou `t`, ve které je uložen text 'sin(pi/2)'. Po zadání

```
t
```

obdržíme pouze výpis textu

```
t =
sin(pi/2)
```

ale

```
eval(t)
```

text vyhodnotí a my dostaneme

```
ans =
1
```

Jiný příklad

```
t = '1/(i+j-1)';
for i=1:n
```

```
  for j=1:n
    a(i,j) = eval(t);
  end
end
```

vytvoří Hilbertovu matici řádu n.

Závěrečný příklad ukáže, jak můžete použít `eval` spolu s funkcí `load` k načtení deseti postupně očíslovaných souborů `'data1'`, `'data2'`, ..., `'data10'`:

```
fname = 'data';
for i=1:10
  eval(['load ', fname, int2str(i)])
end
```

Pozor na mezeru mezi `load` a názvem souborů.

9.7 Jak zvýšit rychlost a ušetřit paměť

Operace s vektory a maticemi, které jsou vestavěné v MATLABu, jsou daleko rychlejší než operace vyžadující kompilaci a interpretaci. To znamená, že chcete-li získat co nejvyšší rychlost zpracování vašich M-souborů, musíte se pokusit vaše algoritmy vektorizovat. Kdekoli je to možné, nahraďte cykly `for` a `while` vektorovými či maticovými operacemi. Např. jedním ze způsobů, jak vypočítat sinus od 0 do 100 s krokem 0.01, je

```
i=0;
for t=0:0.01:100
  i=i+1;
  y(i)=sin(t);
end
```

Vektorizovaná verze stejného kódu je

```
t = 0:0.01:100;
y = sin(t);
```

anebo přímo

```
y = sin(0:0.01:100);
```

Na pomalém počítači trval první příklad 10.4 sekundy, zatímco druhý pouze 0.16 sekundy, tedy 65-krát rychleji. Při složitějším kódu není vždy snadné zjistit, jakým způsobem kód optimalizovat.

Avšak pokud je pro vás rychlost důležitá, budete muset vždy hledat cestu jak váš algoritmus vektorizovat.

Pokud část svého kódu vektorizovat nemůžete, máte ještě jednu možnost jak provádění svých cyklů `for` urychlit: provést předběžnou alokaci vektorů, do kterých se v cyklu ukládají výsledky. Např. první příkaz v následujícím příkladu značně urychlí provedení cyklu `for`:

```
y=zeros(1,100);
for i=1:100
  y(i)=det(X^i);
end
```

Pokud by nebyla provedena předběžná alokace vektoru `y`, musel by interpreter MATLABu zvětšovat velikost vektoru `y` při každém průchodu cyklem o jeden prvek a tím by se výpočet značně zpomalil.

Pokud pracujete s velikými maticemi na počítačích s relativně malou pamětí, má pro vás metoda předběžné alokace ještě jednu výhodu: efektivnější využití paměti. V průběhu práce s MATLABem dochází k fragmentaci paměti. I když máte relativně dosti volného místa v paměti, nemusíte ještě mít dost spojitého prostoru pro uložení velké proměnné. Předběžná alokace pomáhá redukovat fragmentaci paměti.

10 Vstupy a výstupy

10.1 Manipulace se soubory

Příkazy `dir`, `type`, `delete` a `cd` slouží pro manipulaci se soubory. Následující tabulka uvádí ekvivalentní příkazy v jednotlivých operačních systémech:

MATLAB	MS-DOS	UNIX	VAX/VMS
`dir`	`dir`	`ls`	`dir`
`type`	`type`	`cat`	`type`
`delete`	`del`	`rm`	`delete`
`cd`	`chdir`	`cd`	`set default`

Ve většině příkazů můžete používat obvyklé označení disku, název adresáře a *žolíkové* znaky (* a ?).

Příkaz `type` se liší od systémového příkazu `type`. Pokud zadáte za příkazem `type` název souboru bez přípony, MATLAB použije implicitně příponu `'.m'`. Tato odlišnost je velice výhodná, neboť nejčastější použití příkazu `type` v MATLABu je zobrazení M-souborů na obrazovce.

Příkaz `diary` vytvoří na disku textový soubor s popisem vaší práce s MATLABem. (Grafická okna se neukládají.)

10.2 Spouštění externích programů

Znak vykřičník slouží ke spouštění externích programů. Znaky následující za vykřičníkem musí tvořit platný příkaz operačního systému nebo název externího programu. Např.

```
!notepad angle.m
```

vyvolá editor `notepad` a načte do něj soubor `angle.m`. Po opuštění editoru předá operační systém řízení zpátky MATLABu.

10.3 Import a export dat

Do MATLABu můžete zavádět data z jiných programů několika způsoby. Obdobně můžete exportovat data MATLABu do jiných programů. Vaše programy mohou také přímo spolupracovat s MAT-soubory; tj. datové soubory MATLABu. Tato kapitola popisuje techniky pro import dat do a export dat z MATLABu.

10.3.1 Import dat

Nejlepší metoda pro import dat závisí na objemu dat, formátu dat atd. Zde jsou některé možnosti importu dat:

- Zadání dat jako explicitní seznam prvků. Jestliže máte malé množství dat, řekněme méně než 15 prvků, je snadné zadat data explicitně pomocí hranatých závorek. Tato metoda je nevhodná pro velké množství dat, neboť nemůžete opravit chybně zadaný vstup. Podrobnější informace naleznete v kapitole *Vstup matic*.
- Vytvoření dat v M-souboru. Použijte textový editor pro vytvoření skriptového M-souboru, který zadá vaše data jako explicitní seznam prvků. Na rozdíl od první metody zde můžete textovým editorem chyby kdykoli změnit nebo opravit.
- Načtení dat z textového (ASCII) souboru, ve kterém jsou jednotlivé řádky dat zakončeny znakem konce řádku a mezi jednotlivými sloupci jsou mezery. Takovéto soubory mohou být přímo do MATLABu načteny příkazem `load`. Výsledek je uložen do proměnné, jejíž název se shoduje s názvem souboru.
- Čtení dat pomocí `fopen`, `fread` a ostatních nízkoúrovňových funkcí MATLABu pro vstup a výstup. Tato metoda je výhodná pro načítání datových souborů z aplikací, které mají své vlastní zavedené formáty datových souborů.
- Vytvoření MEX-souboru pro načtení dat. Tuto metodu použijte, pokud jsou již k dispozici podprogramy pro načítání datových souborů z jiných aplikací.
- Vytvoření programu ve Fortranu nebo C pro transformaci vašich dat do MAT-souboru a následné načtení MAT-souboru do MATLABu pomocí příkazu `load`.

10.3.2 Export dat

Metody, kterými exportujeme data z MATLABu, jsou následující:

- Pro malé matice můžete použít příkaz `diary`. Pro následnou úpravu souboru vytvořeného příkazem `diary` můžete použít textový editor.
- Uložit data v textovém (ASCII) formátu pomocí příkazu `save` s volbou `-ascii`. Např.

  ```
  A=rand(4,3);
  save temp.dat A -ascii
  ```

  vytvoří textový soubor s názvem `temp.dat`, který obsahuje např.

  ```
  0.7012 0.0475 0.7564
  0.9103 0.7361 0.9910
  0.7622 0.3282 0.3653
  0.2625 0.6326 0.2470
  ```

- Zapsat data ve speciálním formátu pomocí funkcí `fopen`, `fwrite` a ostatních nízkoúrovňových funkcí MATLABu pro vstup a výstup. Tato metoda je výhodná pro ukládání datových souborů ve formátu, který vyžadují jiné aplikace.
- Vytvoření MEX-souboru pro uložení dat. Tuto metodu použijte, pokud jsou již k dispozici podprogramy pro ukládání datových souborů ve formátu požadovaném jinými aplikacemi.
- Vytvoření programu ve Fortranu nebo C pro transformaci vašich dat do MAT-souboru a následné načtení MAT-souboru do MATLABu pomocí příkazu `load`.
- Uložte data jako MAT-soubor pomocí příkazu `save` a potom napište program ve Fortranu nebo C, který převede MAT-soubor do vašeho vlastního speciálního formátu.

11 Nízkoúrovňový vstup a výstup

Funkce MATLABu pro souborový vstup a výstup umožňuje načíst data uložená v jiném formátu přímo do MATLABu nebo zapsat data vytvořená MATLABem ve formátu, který požaduje jiný program nebo zařízení. Funkce zpracovávají (čtou a zapisují) formátované textové soubory a binární datové soubory.

Funkce MATLABu pro souborový vstup a výstup jsou založeny na funkcích souborového vstupu a výstupu jazyka C. Jestliže znáte jazyk C, jste pravděpodobně důvěrně seznámeni s těmito příkazy. Referenční manuál k jazyku C obsahuje podrobné informace o tom, jak tyto funkce pracují na vašem systému.

11.1 Otvírání a zavírání souborů

Před prvním použitím souboru (čtení nebo zápis) musíme soubor otevřít příkazem `fopen`, ve kterém určíme název tohoto souboru a řetězec definující režim přístupu. Např. příkaz

```
fid = fopen('pen.dat', 'r')
```

otevře datový soubor `pen.dat` pro čtení. Použitelné přístupové řetězce jsou:

`'r'`	pro čtení
`'w'`	pro zápis
`'a'`	pro připsání
`'r+'`	pro čtení i zápis

Systémy jako např. VMS, které rozlišují mezi textovými a binárními soubory, mohou požadovat dodatečné znaky v přístupovém řetězci, např. `'rb'` pro otevření binárního souboru pro čtení.

Funkce `fopen` vrací identifikátor souboru, což je nezáporné celé číslo přiřazené souboru operačním systémem. Identifikátor souboru je v podstatě zkratkou pro odkaz na soubor. Funkce MATLABu pro souborový vstup a výstup používají identifikátor souboru pro rozlišení, s jakým souborem se má pracovat (otevřít, číst, zapisovat, zavřít).

Jestliže nemůže být soubor otevřen (např. pokus o otevření neexistujícího souboru pro čtení), vrací `fopen` jako identifikátor souboru hodnotu -1. Je dobrým zvykem ihned po otevření souboru otestovat identifikátor souboru. Druhá vrácená hodnota poskytuje doplňující informace o chybě. Např. pokud MATLAB nemůže najít soubor pen.dat, příkaz

```
[fid, message] = fopen('pen.dat', 'r')
```

nastaví `fid` na hodnotu -1 a message na řetězec

```
No such file or directory.
```

Poznamenejme, že hlášení jsou závislá na systému. Informace o chybách poskytuje také funkce `ferror`.

Jednou otevřený soubor je použitelný pro čtení nebo zápis. Jakmile ukončíme čtení nebo zápis, uzavřeme soubor funkcí `fclose`. Např.

```
status = fclose(fid)
```

uzavírá soubor sdružený s identifikátorem `fid` a

```
status = fclose('all')
```

uzavírá všechny otevřené soubory. Obě formy vrací hodnotu 0, jestliže byla operace úspěšná, nebo hodnotu -1, pokud byla neúspěšná.

11.2 Čtení binárních datových souborů

Funkce `fread` čte binární datové soubory. Ve své nejjednodušší formě načte celý soubor do matice. Např.

```
fid = fopen('penny.dat', 'r');
A = fread(fid);
status = fclose(fid);
```

načte všechna data ze souboru `penny.dat` (viz `demo penny`) jako neznaménkové znaky a zapíše je do matice `A`.

Další dva volitelné argumenty funkce `fread` umožňují řídit počet načtených hodnot a jejich přesnost. Tak např. první volitelný argument zajistí, že

```
fid = fopen('penny.dat', 'r');
A = fread(fid, 100);
status = fclose(fid);
```

načte prvních 100 datových hodnot do sloupcového vektoru `A`. Nahrazením čísla 100 rozměry matice $[10, 10]$ dojde k načtení stejných 100 prvků do matice typu $(10, 10)$. Příkaz

```
A = fread(fid, Inf);
```

čte až do konce souboru; vyplňuje matici `A` jako sloupcový vektor. Stejný efekt má chybějící argument o počtu načtených hodnot.

Druhý z volitelných argumentů, argument numerické přesnosti, řídí počet bitů načtených pro každou hodnotu a interpretaci těchto bitů jako znak, celé číslo nebo reálné číslo. Numerická přesnost je

závislá na hardware. Neznáme-li, jakým způsobem je implementována přesnost na našem počítači, je třeba nahlédnout do referenční příručky hardware.

MATLAB podporuje mnoho přesností, které můžeme určit jak specifickým názvem MATLABu, tak ekvivalenty z jazyka C nebo Fortranu. Některé běžné přesnosti jsou:

`'char'` a `'uchar'`	pro znaménkové a neznaménkové znaky (obvykle 8 bitů)
`'short'` a `'long'`	pro krátká a dlouhá celá čísla (obvykle 8 a 32 bitů)
`'float'` a `'double'`	pro jednoduchou a dvojnásobnou přesnost reálných čísel (obvykle 32 a 64 bitů)

Kompletní seznam použitelných přesností viz `fread` ve druhém dílu této knihy, *Popis funkcí*.

Pokud se `fid` odkazuje na otevřený soubor obsahující reálné hodnoty, potom

```
A = fread(fid, 10, 'float')
```

načte 10 reálných čísel s jednoduchou přesností do sloupcového vektoru `A`.

Následující příklad otevírá soubor obsahující popis funkce `fread`, potom jej načte a celý soubor zobrazí.

```
freadid = fopen('fread.m', 'r');
F = fread(freadid, Inf, 'uchar');
disp(setstr(F'))
status = fclose(freadid);
```

Poznamenejme, že příkaz `fread` v této ukázce je ekvivalentní příkazu

```
F = fread(freadid);
```

11.3 Zápis binárních datových souborů

Funkce `fwrite` zapisuje prvky matice do souboru ve specifikované numerické přesnosti; funkce vrací počet zapsaných hodnot. Např.

```
fwriteid = fopen('magic5.bin', 'w');
count = fwrite(fwriteid, magic(5), 'integer*4');
status = fclose(fwriteid);
```

vytvoří 100-bytový binární soubor obsahující 25 prvků (magický čtverec 5×5) uložených jako 4-bytová celá čísla. Proměnná `count` se nastaví na 25.

11.4 Ovládání pozice v souboru

Funkce `fseek` resp. `ftell` nám umožní nastavit resp. dotázat se na pozici v souboru, se kterou bude pracovat příští vstupní nebo výstupní operace.

Funkce `ftell` vrací polohu (v bytech) ukazovátka pozice v souboru, který vstupuje do funkce jako její argument.

Funkce `fseek` změní polohu ukazovátka pozice v souboru. To nám umožní přeskočit data nebo se navrátit k dřívější části souboru. Argumenty této funkce jsou identifikátor souboru; kladná nebo záporná hodnota posunu (v bytech), která určuje, zda dojde k posunu vpřed či vzad; a počátek, od kterého se provádí výpočet posunu. Počátek může být aktuální pozice v souboru ('cof'), začátek ('bof') či konec ('eof') souboru.

V následujícím příkladu se do souboru `five.bin` zapíší čísla 1 až 5 (jako dvou-bytová celá čísla), po znovuotevření souboru se prvním voláním `fseek` přeskočí prvních šest bytů (obsahují čísla 1, 2 a 3), prvním voláním `fread` se přečte hodnota 4, pozice vrácená prvním `ftell` je osm bytů od začátku souboru. Druhé volání `fseek` posune pozici o 4 byty zpět a následný `fread` přečte hodnotu 3.

```
A = [1:5];
fid = fopen('five.bin', 'w');
fwrite(fid, A, 'short');
status = fclose(fid);
fid = fopen('five.bin', 'r');
status = fseek(fid, 6, 'bof');
four = fread(fid, 1, 'short');
position = ftell(fid);
status = fseek(fid, -4, 'cof');
three = fread(fid, 1, 'short');
status = fclose(fid);
```

11.5 Zápis formátovaných textových souborů a řetězců

Funkce `fprintf` konvertuje data na znakové řetězce a posílá je na obrazovku nebo do souboru. Výstupní formát je určen formátovacím řetězcem, který obsahuje konverzní specifikátory a text. Konverzní specifikátory řídí výstup prvků matic. Text je zkopírován přímo. Konverzní specifikátory začínají znakem %; běžné konverze zahrnují:

`%e`	pro exponenciální notaci
`%f`	pro pevnou řádovou čárku
`%g`	pro automatickou volbu kratší z formátů %e nebo %f

Volitelné položky v konverzních specifikátorech řídí minimální šířku položky a její přesnost. Např.

```
x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('exptable.txt', 'w');
fprintf(fid, 'Exponential Function\n\n');
fprintf(fid, '%6.2f %12.8f\n', y);
status = fclose(fid);
```

vytvoří textový soubor obsahující krátkou tabulku exponenciální funkce. První volání `fprintf` vytvoří titulek, následuje dvoje odřádkování (`\n\n`) a druhé volání `fprintf` vytvoří vlastní tabulku. Formátovací řetězec poskytuje formát pro každou řádku tabulky:

- číslo v pevné řádové čárce šest znaků široké s dvěma desetinnými místy,
- dvě mezery,
- číslo v pevné řádové čárce dvanáct znaků široké s osmi desetinnými místy.

Prvky matice `y` jsou transformovány *po sloupcích* podle specifikátorů ve formátovacím řetězci. Funkce používá formátovací řetězec opakovaně, dokud nejsou zkonvertovány všechny prvky matice.

Příbuzná funkce `sprintf` posílá své výsledky do řetězce místo do souboru nebo na obrazovku. Např.

```
root2 = sprintf('The square root of %f is %10.8e.\n', 2, sqrt(2));
```

11.6 Čtení formátovaných textových souborů a řetězců

Funkce pro textový vstup `fscanf` je podobná funkci `fprintf`. Funkce `fscanf` má jako své argumenty identifikátor otevřeného textového souboru a formátovací řetězec, který obsahuje řádné znaky a konverzní specifikátory. Konverzní specifikátory pro `fscanf` začínají znakem %; běžné konverze zahrnují:

`%s`	pro načtení řetězce
`%d`	pro načtení celého čísla
`%f`	pro načtení reálného čísla

Řádné znaky ve formátovacím řetězci jsou jeden za druhým vyhledávány ve vstupu až na to, že jeden oddělovací znak (white space) odpovídá libovolnému řetězci oddělovacích znaků. Jak se čte vstup, MATLAB přizpůsobuje řídící řetězec vstupu a vrací datové hodnoty zkonvertované podle formátovacích specifikátorů. Následující příklad čte soubor s exponenciálními daty, který byl vytvořen dříve:

```
fid = fopen('exptable.txt', 'r');
title = fscanf(fid, '%s');
[table, count] = fscanf(fid, '%f %f');
status = fclose(fid);
```

Řádek s titulkem zpracuje specifikátor %s při prvním volání `fscanf`. Druhé volání `fscanf` načte tabulkové hodnoty (opakovaně se načítají dvě čísla s pevnou řádovou čárkou, dokud se nedosáhne konce souboru). `count` vrací počet přečtených hodnot.

Volitelný argument velikosti řídí počet prvků matice pro čtení. Např. jestliže `fid` odpovídá otevřenému souboru obsahujícímu řetězce desetinných čísel, potom

```
A = fscanf(fid, '%5d', 100);
```

načte 100 desetinných hodnot do sloupcového vektoru `A` a

```
A = fscanf(fid, '%5d', [10,10]);
```

načte 100 desetinných čísel do matice `A` typu $(10, 10)$.

Příbuzná funkce `sscanf` bere svůj vstup z řetězce místo ze souboru. Např.

```
rootvalues = sscanf(root2, 'The square root of %f is %f.');
```

vrací sloupcový vektor obsahující hodnotu 2 a její druhou odmocninu.

12 Grafický systém

Grafický systém MATLABu obsahuje různé způsoby pro zobrazení dat. Tento systém je vybudován na základě sady grafických objektů (`line`, `surface`, ...), jejichž vzhled lze řídit nastavením parametrů jejich vlastností. Poněvadž MATLAB obsahuje dostatečné množství 2-D a 3-D grafických funkcí vyšší úrovně, není nezbytné používat přímý přístup k vlastnostem objektů.

Následující kapitoly týkající se 2-D a 3-D grafiky popisují, jak použít k zobrazení uživatelských dat grafických funkcí vyšší úrovně. V kapitole Objektová grafika je vysvětleno, jak vytvořit grafické objekty a jak s nimi manipulovat.

12.1 Dvojrozměrná grafika

K zobrazení dat ve tvaru 2-D grafů včetně popisů a komentářů je v MATLABu k dispozici řada funkcí, které jsou popsány v této kapitole. Na jednoduchých příkladech je ukázáno použití některých z nich.

12.1.1 Elementární funkce pro kreslení grafů

Následující seznam obsahuje přehled funkcí, které vytvářejí graf daných dat. Tyto funkce se liší pouze užitím jiné stupnice os. Všechny akceptují vstupní data ve tvaru vektorů nebo matic a automaticky provádějí transformaci os podle rozsahu hodnot vstupních dat.

`plot`	vytváří graf užitím lineární stupnice pro obě osy
`loglog`	vytváří graf užitím logaritmické stupnice pro obě osy
`semilogx`	vytváří graf užitím logaritmické stupnice pro x-ovou osu a lineární stupnice pro y-ovou osu
`semilogy`	vytváří graf užitím logaritmické stupnice pro y-ovou osu a lineární stupnice pro x-ovou osu

Do grafu lze přidat nadpis, popisy os, text nebo zobrazit síť pomocí následujících funkcí:

`title`	přidá nadpis do grafu (doprostřed nad graf)
`xlabel`	přidá popis x-ové osy (doprostřed pod osu)
`ylabel`	přidá popis y-ové osy (doprostřed podél osy)
`text`	přidá textový řetězec na určenou pozici
`gtext`	umístí text do grafu na místo vybrané myší
`grid`	zobrazí síť

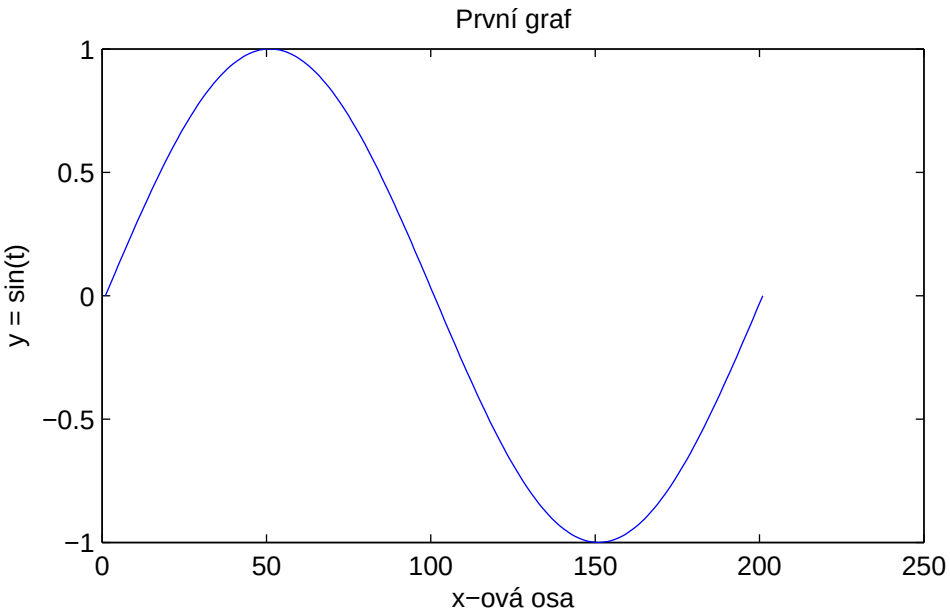
12.1.2 Vytváření grafu užitím funkce plot

Je-li `x` vektor, `plot(x)` vytvoří lineární graf prvků vektoru `x` vzhledem k indexu prvků tohoto vektoru

```
t = 0:pi/100:2*pi;
plot(sin(t))
```

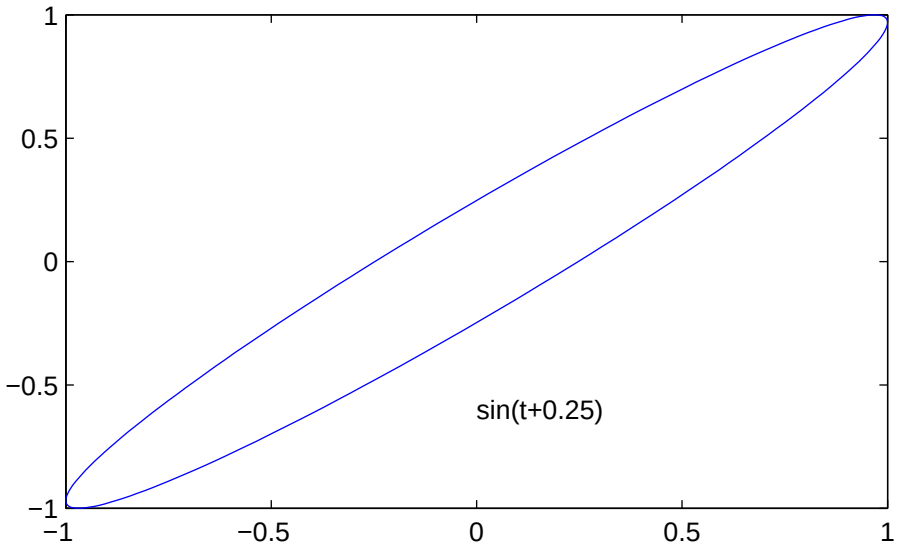
Následující příkazy přidají do grafu nadpis a popisy os

```
title('První graf')
xlabel('x-ová osa')
ylabel('y = sin(t)')
```



Jestliže jsou jako argumenty funkce plot specifikovány dva vektory `x` a `y`, `plot(x,y)` vytvoří graf `y` jako funkci `x`

```
x = sin(t);
y = sin(t+.25);
plot(x,y)
text(0,-0.6,'sin(t+0.25)')
```



Příkazem `text` je umístěn textový řetězec na pozici $(0, -0.6)$. Text je možné také umístit na zvolenou pozici myší příkazem

```
gtext('sin(t+0.25)')
```

12.1.3 Typy čar, značky a barvy

Pro kreslení čar, případně vyznačení bodů, je k dispozici sada typů čar a barev uvedených v následující tabulce.

symbol	**barva**	**symbol**	**typ čáry**
`y`	žlutá	`.`	bod
`m`	fialová	`o`	kroužek
`c`	tyrkysová	`x`	značka x
`r`	červená	`+`	plus
`g`	zelená	`*`	hvězdička
`b`	modrá	`-`	plná
`w`	bílá	`:`	tečkovaná
`k`	černá	`-.`	čerchovaná
		`--`	čárkovaná

Třetím parametrem funkce `plot` lze specifikovat typ čar a jejich barvu. V příkazu

```
plot(x,y,s)
```

s značí jedno-, dvou- nebo tří-znakový řetězec (oddělený apostrofy) z předcházející tabulky. Kombinovat lze pouze symboly z různých sloupců. Např. `plot(x,y,'c+')` vykreslí značku `+` tyrkysové barvy v každém bodě dat, `plot(t,x,'r-',t,y,'g--')` vykreslí graf funkce `x` plnou červenou čarou, graf funkce `y` zelenou čárkovanou čarou. Není-li barva určena, funkce `plot` automaticky vybírá barvy z tabulky v uvedeném pořadí. Pro jednu čáru je implicitní žlutá barva, protože je nejlépe

viditelná na černém pozadí. Pro násobné čáry vybírá funkce `plot` cyklicky prvních šest barev z výše uvedené tabulky. Tloušťku čáry i velikost značek lze měnit.

12.1.4 Přidání čar do existujícího grafu

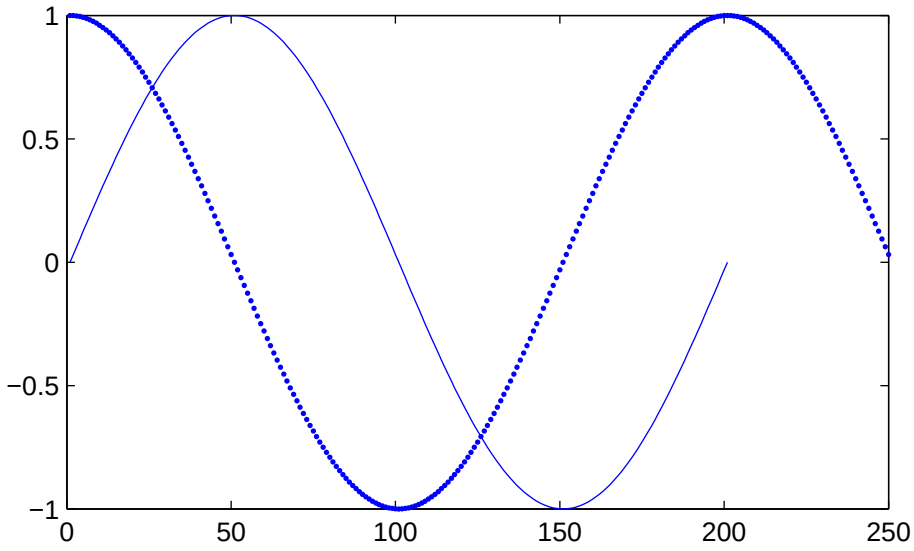
Vyvolání dalšího příkazu `plot` běžně způsobí vymazání aktuálních os a nakreslení nových. Tomu lze zabránit příkazem `hold on`. Potom MATLAB neodstraní existující osy, ale přidá do aktuálních os nové čáry. Pokud je ale rozsah nových dat větší než rozsah původních dat, změní zároveň rozsah os podle nových dat,

```
t = 0:pi/100:2*pi;
plot(sin(t))
hold on
tt = 0:pi/100:4*pi;
plot(cos(tt),'.')
hold off
```

Tyto příkazy zobrazí do jednoho grafu funkce sinus a kosinus ve zvoleném rozsahu, měřítko os je voleno podle rozsahu funkce kosinus. Zmrazením měřítka os lze ale dosáhnout vykreslení další čáry do původního rozsahu os .

```
t = 0:pi/100:2*pi;
plot(sin(t))
hold on
axis(axis)
tt = 0:pi/100:4*pi;
plot(cos(tt),'.')
hold off
```

Příkazem `axis(axis)` se zmrazí měřítko os po vykreslení první čáry $\sin(t)$, další čára $\cos(tt)$ se zobrazí jen částečně v rozsahu původních os. Příkazem `hold off` se umožní následné přepsání os.



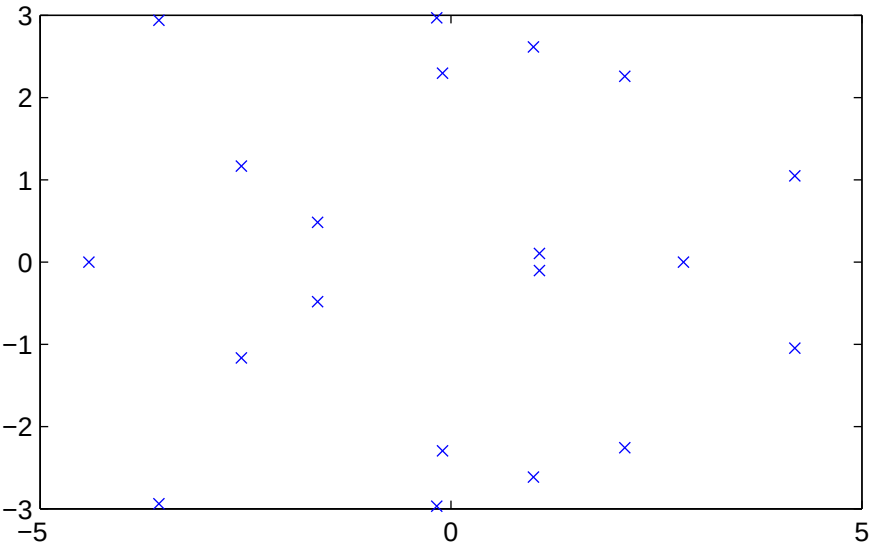
12.1.5 Imaginární a komplexní data

Jsou-li argumenty funkce `plot` komplexní, tj. mají nenulové imaginární části, jsou tyto imaginární části ignorovány. Pouze v případě, kdy je argument funkce `plot` jediný, tj. `plot(Z)`, kde `Z` je komplexní vektor nebo matice, je vykreslena závislost imaginárních částí prvků `Z` vzhledem k reálným částem. Tento příkaz je zkráceným zápisem příkazu

```
plot(real(Z), imag(Z))
```

Následující příklad uvádí rozložení vlastních čísel matice náhodných čísel řádu 20

```
plot(eig(randn(20,20)), 'x')
```



K vykreslení více než jedné komplexní matice nebo vektoru nelze použít žádný zkrácený zápis, reálné a imaginární části musejí být explicitně vyjádřeny.

12.1.6 M-soubor peaks

Mnoho příkladů v následujících kapitolách užívá ke generování matice dat m-soubor nazvaný `peaks`. Data vychází z funkce dvou proměnných, která má tři lokální maxima a minima,

$$f(x,y) = 3(1-x)^2\, e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3}\, e^{-(x+1)^2-y^2}$$

M-soubor `peaks` vytváří matici, která obsahuje hodnoty funkce pro x a y v rozsahu od -3 do $+3$. Proměnná x se mění podél sloupců a proměnná y podél řádek. Vstupní parametr funkce `peaks` určuje řád výsledné matice, např.

```
M = peaks(20);
```

vytvoří matici dat řádu 20. Není-li uveden žádný vstupní argument, funkce implicitně generuje matici řádu 49.

12.1.7 Kreslení matic

Je-li argumentem funkce plot jediná matice `Y`

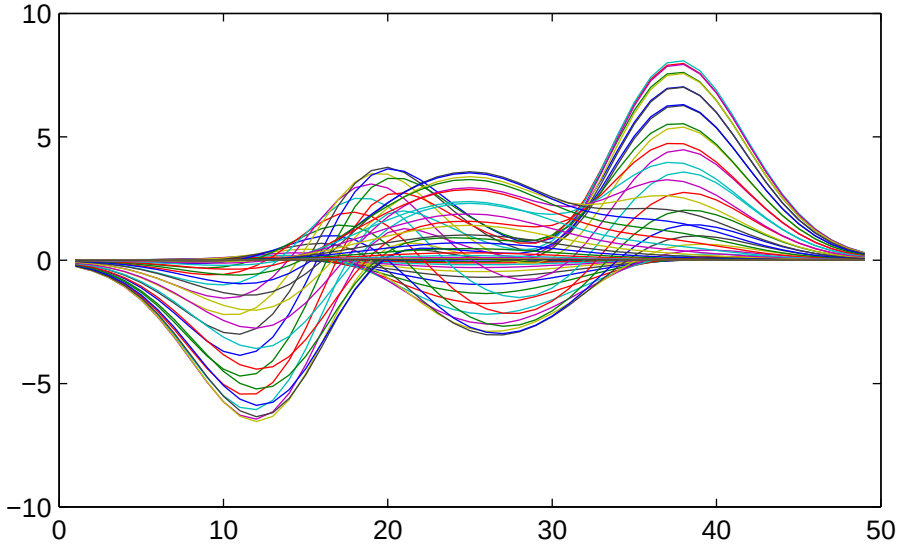
```
plot(Y)
```

vykreslí se pro každý sloupec matice `Y` jedna čára. Osa x je popsána řádkovým indexem sloupcového vektoru $[1:m]$, kde m je počet řádek matice `Y`.

Např. příkaz

```
plot(peaks)
```

vytvoří graf se 49 čarami

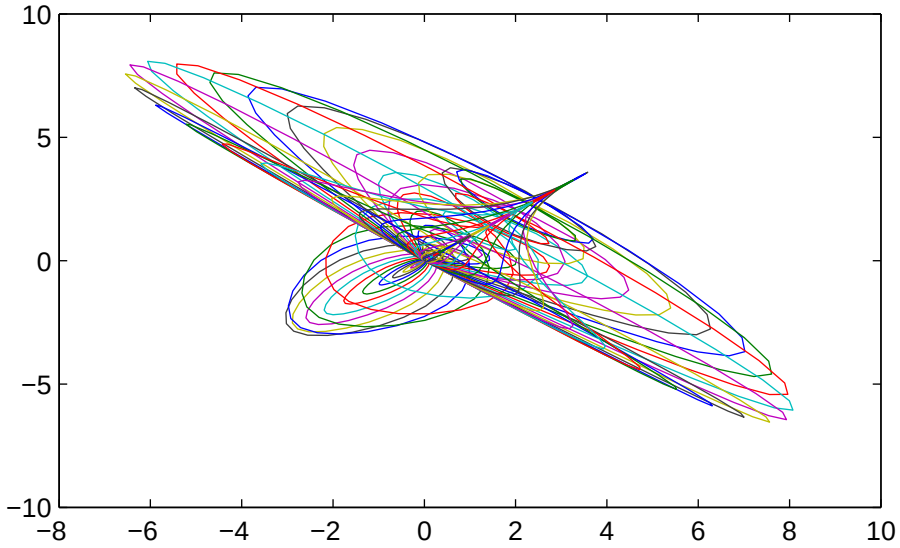


Jedná se o pohled na plochu funkce `peaks` při azimutu 90° a elevaci 0°.

Funkce `plot` dovoluje také použít jako argumentu dva vektory nebo matice. Např.

```
plot(peaks, rot90(peaks'))
```

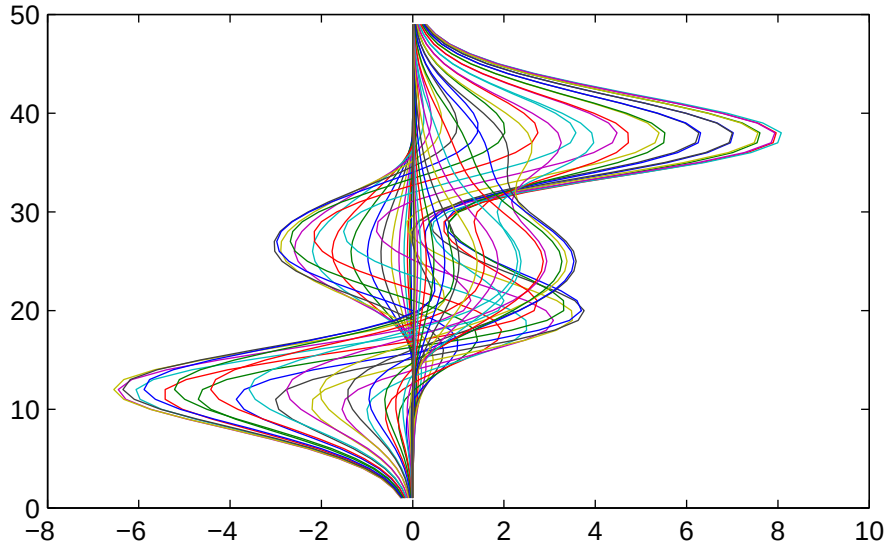
vytvoří následující zajímavý graf funkce `peaks` vůči rotaci a posuvu této funkce



Obecně platí: je-li funkce `plot` použita se dvěma argumenty a má-li buď `X` nebo `Y` více než jednu řádku nebo jeden sloupec, potom

- je-li `Y` matice a `x` vektor, `plot(x,Y)` kreslí řádky nebo sloupce matice `Y` vzhledem k vektoru `x` a pro každou čáru použije jinou barvu nebo jiný typ čáry. O tom, zda budou kresleny řádky nebo sloupce rozhoduje počet prvků vektoru `x`. Řádková nebo sloupcová orientace je vybrána podle shody počtu prvků řádků nebo sloupců matice `Y` s počtem prvků vektoru `x`. Pokud je matice `Y` čtvercová, jsou vykresleny její sloupce.
- je-li `X` matice a `y` vektor, `plot(X,y)` kreslí každý řádek nebo sloupec matice `X` vzhledem k vektoru `y`. Např. vykreslení matice `peaks` vůči vektoru hodnot od 1 do 49 podél osy `y`, a tím změnu orientace grafu, provedou následující příkazy

```
y = 1:49;
plot(peaks,y)
```



- jsou-li `X` i `Y` matice téže velikosti, `plot(X,Y)` zobrazí sloupce matice `X` vůči sloupcům matice `Y`.

Samozřejmě lze též použít funkci `plot` s několika dvojicemi maticových argumentů

```
plot(X1,Y1,X2,Y2, ...)
```

V každé dvojici musí být matice stejného typu, rozdílné dvojice mohou mít různou dimenzi.

12.1.8 Speciální funkce pro kreslení grafů

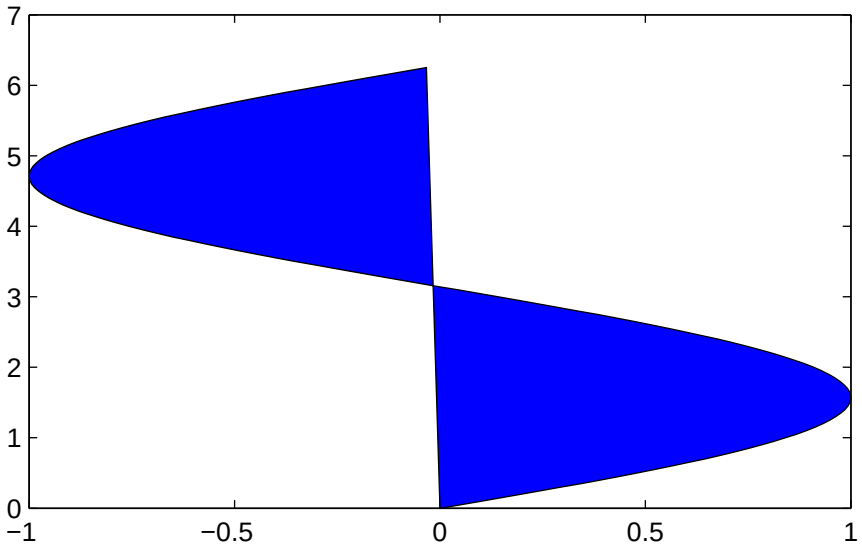
Následující seznam popisuje stručně speciální funkce MATLABu pro vynášení 2-D grafů. Podrobnější popis je uveden u jednotlivých funkcí ve druhém dílu této knihy, *Popis funkcí*.

`bar`	vytváří sloupcový graf
`compass`	vytváří graf komplexních čísel ve formě šipek vycházejících z počátku
`errorbar`	vytváří sloupcový graf chyb
`feather`	vytváří graf komplexních čísel ve formě šipek vycházejících z ekvidistantně rozložených bodů podél horizontální osy
`fplot`	vykreslí graf funkce
`hist`	vytváří histogram
`polar`	vytváří graf v polárních souřadnicích
`quiver`	vytváří graf gradientu nebo jiného vektorového pole
`rose`	vytváří úhlový histogram
`stairs`	vytváří graf ve tvaru schodů
`fill`	vykreslí mnohoúhelník a vyplní jej

12.1.9 Plné mnohoúhelníky

Funkce `fill` vykreslí mnohoúhelník a jeho vnitřek vyplní zadanou barvou. Jsou-li určeny barvy pro každý bod mnohoúhelníku, MATLAB použije bilineární transformaci k určení barvy vnitřku. Mnohoúhelník může být i konkávní nebo se jeho hrany mohou protínat. Následující příklad generuje mnohoúhelník vyplněný modrou barvou

```
t = 0:0.05:2*pi;
x = sin(t);
fill(x,t,'b')
```



Plochu omezenou mnohoúhelníkem lze též vyplnit barevnými odstíny pomocí interpolace barev z barevné mapy např. `hot`

```
colormap(hot)
fill(x,t,x)
```

Tyto odstíny barev se mění od tmavé po světlou v závislosti na hodnotách funkce $\sin(t)$ od minimálních po maximální, tj. v tomto případě zleva doprava.

12.1.10 Vykreslení matematických funkcí

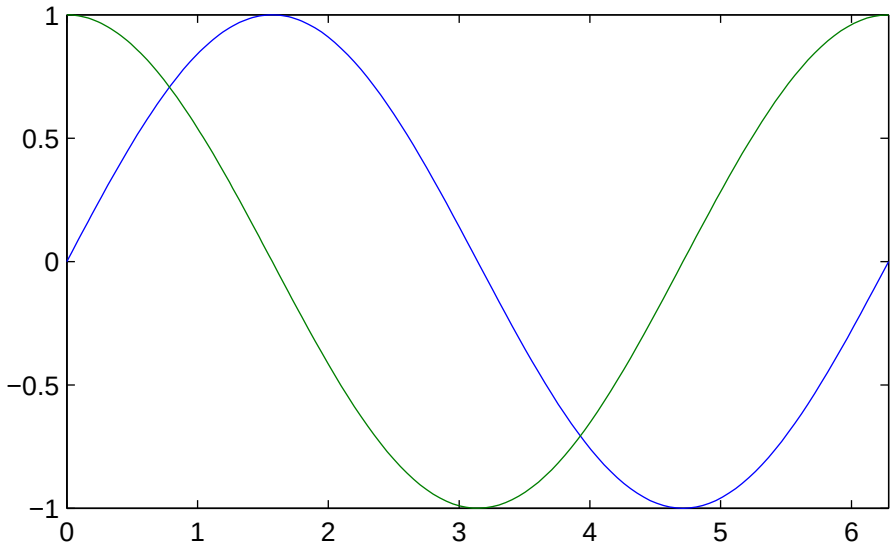
Grafy funkcí lze v MATLABu vykreslit několika způsoby. Nejjednodušší je použití funkce `plot`. Např. následující příkazy by vykreslily průběh funkcí sinus a kosinus.

```
x = (0:0.01:2*pi);
y1 = sin(x);
y2 = cos(x);
plot(x,y1,x,y2)
```

Elegantnější přístup volí funkce `fplot`, která přizpůsobuje periodu vzorkování funkce tak, aby měla k dispozici dostatečný počet funkčních hodnot k vykreslení dané funkce i v intervalech, ve kterých se rychle mění funkční hodnoty. Tedy s použitím příkazu `fplot`

```
fplot('[sin(x) cos(x)]',[0 2*pi])
```

dostáváme



12.2 Třírozměrná grafika

MATLAB poskytuje množství funkcí pro zobrazení 3-D dat ve tvaru čar, ploch nebo drátových modelů ploch. V následujících tabulkách je uveden jejich seznam se stručným popisem každé funkce. Podrobný popis jednotlivých funkcí je ve druhém dílu této knihy, *Popis funkcí*.

`plot3`	kreslí čáry a body v prostoru
`contour` `contour3`	vytváří vrstevnice grafu 2-D vytváří vrstevnice grafu 3-D
`pcolor`	vytvoří obdélníkové pole buněk, kterým jsou přiřazeny barvy podle velikosti prvků v matici
`image`	zobrazí matici jako obraz, každý prvek matice určuje barvu políčka v obraze. Prvky matice jsou užity jako indexy aktuální mapy barev k určení barvy

`mesh`
`meshc`
`meshz`
	vytváří síť v prostoru,
	prvky matice zobrazí jako výšky nad základnou

`surf`
`surfc`
`surfl`
	jako `mesh`, ale vytvoří plochu složenou z čtyřúhelníků, jejichž vrcholy tvoří prvky zadané matice. Jednotlivé čtyřúhelníky jsou vybarveny odpovídající barvou.

`fill3`	vytvoří mnohoúhelník v prostoru a vyplní jej

Kromě funkcí uvedených u 2-D grafiky používá MATLAB k popisu os navíc následující funkce

`zlabel`	vytvoří popis z-ové osy
`clabel`	přidá popis vrstevnic

MATLAB dovoluje určit bod pohledu na graf. Obecně je pohled definován transformační maticí čtvrtého řádu, kterou MATLAB používá k transformaci třírozměrného grafu na dvourozměrnou obrazovku. Následující dvě funkce umožňují jednoduchým způsobem definovat bod pohledu

`view`	nastaví aktuální bod pohledu pomocí azimutu a elevace nebo pomocí transformační matice
`viewmtx`	vypočte transformační matici čtvrtého řádu jak pro pravoúhlou, tak pro perspektivní transformaci

12.2.1 Kreslení čar

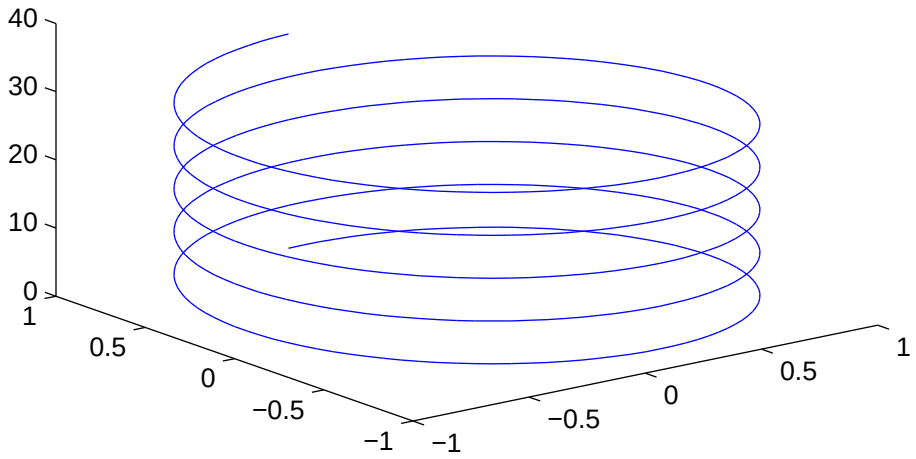
Funkce `plot3` pro kreslení čar v prostoru je analogická funkci `plot`. Jestliže `x`, `y`, a `z` jsou tři vektory téže délky,

```
plot3(x,y,z)
```

vytvoří v prostoru čáru procházející body, jejichž souřadnice jsou prvky vektorů `x`, `y` a `z`, a provede 2-D projekci této čáry na obrazovku. Např.

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

zobrazí spirálu



Jestliže `X`, `Y` a `Z` jsou matice téhož typu, příkaz

```
plot3(X,Y,Z)
```

vykreslí čáry získané ze sloupců matic `X`, `Y` a `Z`. Chceme-li určit typ čar, symboly a barvy, použijeme příkaz `plot3(X,Y,Z,s)`, kde `s` je 1-, 2- nebo 3-znakový řetězec (viz popis funkce `plot`). Podobně,

```
plot3(x1, y1, z1, s1, x2, y2, z2, s2, x3, y3, z3, s3, ...)
```

vykreslí jednotlivé čtveřice.

12.2.2 Funkce meshgrid

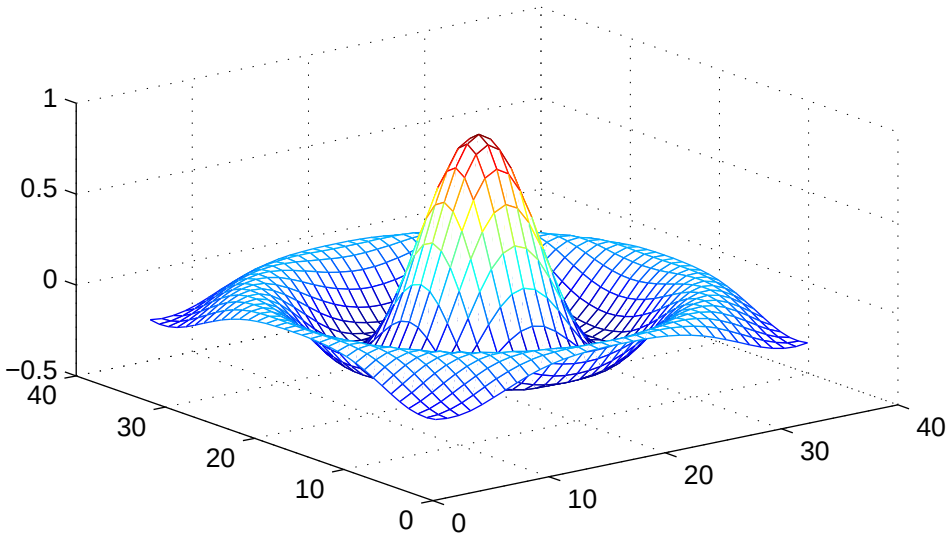
MATLAB definuje pomocí z-ových souřadnic bodů síť plochy nad obdélníkovou mříží v rovině $x-y$. Vytváří síť spojením sousedních bodů přímými čarami.

Tento způsob zobrazení je vhodný pro grafické znázornění matice, která je v numerickém tvaru příliš velká a nepřehledná, nebo pro grafy funkcí dvou proměnných.

Ke znázornění funkce dvou proměnných $z = f(x, y)$ musíme nejprve vytvořit matice `X` a `Y` (definiční oblast funkce). Funkce `meshgrid` transformuje oblast určenou dvěma vektory `x` a `y` na matice `X` a `Y`, které pak využijeme k vyhodnocení a znázornění funkce dvou proměnných. Řádky matice `X` jsou kopiemi vektoru `x`, sloupce matice `Y` jsou kopiemi vektoru `y`. Použití funkce `meshgrid` ukážeme na vykreslení funkce $\sin(r)/r$, která vytváří plochu známou jako „sombréro“. Funkci zobrazíme v rozsahu $x \in \langle -8, +8 \rangle$, $y \in \langle -8, +8 \rangle$. Funkce `meshgrid` vytvoří požadované matice `X`, `Y`

```
x = -8:0.5:8;
y = x;
[X, Y] = meshgrid(x, y);
R = sqrt( X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(Z)
```

Přidáním `eps` v matici `R` zamezíme dělení nulou, které by vytvořilo v matici `Z` pro počátek hodnotu `NaN`. Funkce `mesh` s argumentem `Z` pak vytvoří následující síť plochy „sombréra“.

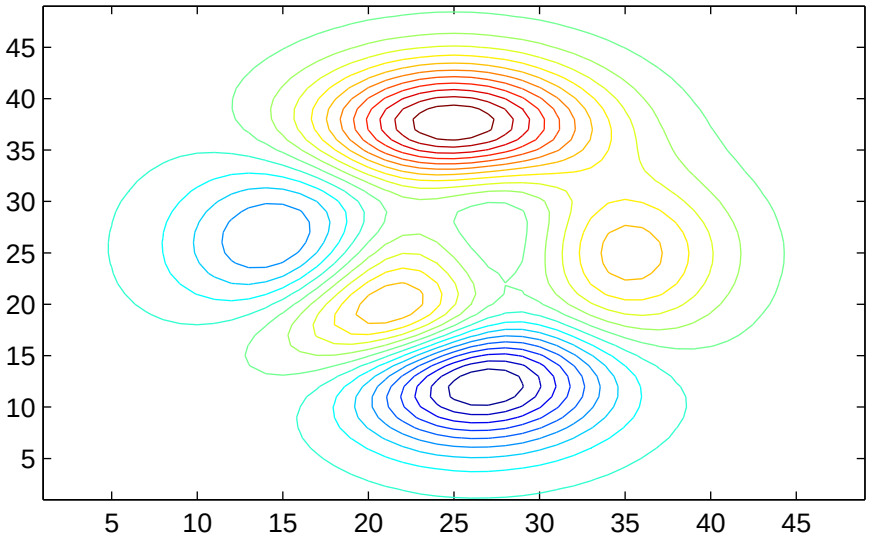


12.2.3 Kreslení vrstevnic

MATLAB umožňuje kreslit vrstevnice v ploše i v prostoru. Funkce `contour` a `contour3` vytváří grafy složené z čar spojujících body téže z-ové souřadnice získané interpolací dat vstupní matice.

Můžeme určit počet vrstevnic, které mají být zobrazeny, měřítko os a hodnoty vstupních dat vykreslovaných vrstevnic. Následující příklad vytvoří graf s 20 vrstevnicemi pro vstupní m-soubor `peaks`.

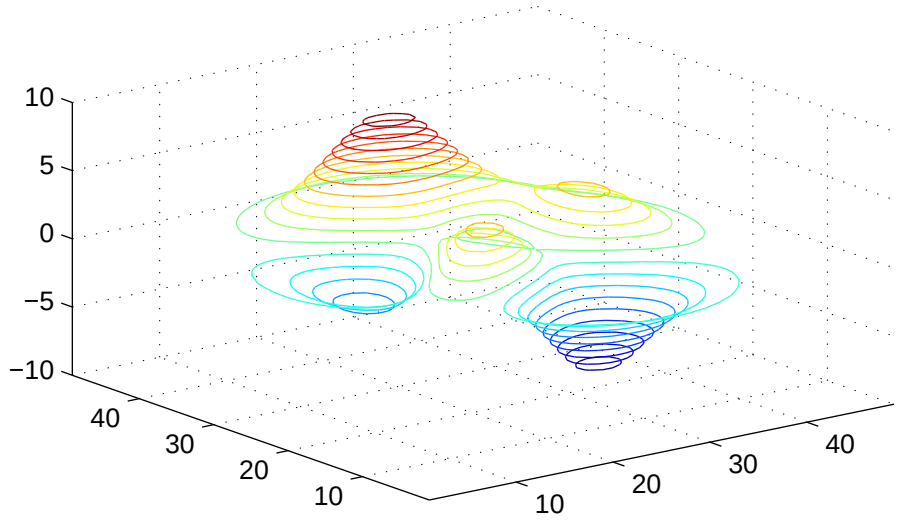
```
contour(peaks, 20)
```



Funkce `contour` používá typ čar, symboly a barvy stejným způsobem jako funkce `plot`.

Vrstevnice pro stejná data jako v předcházejícím příkladě, ale v prostoru vytvoří funkce `contour3`

```
contour3(peaks, 20)
```



12.2.4 Funkce pcolor

Název funkce `pcolor` je zkratkou slova pseudocolor. Každý bod `z(i,j)` v následujícím příkladě je transformován podle rozsahu aktuální mapy barev (colormap); maximální hodnota `z(i,j)` se nastaví na maximální hodnotu indexu mapy barev.

```
z = peaks;
pcolor(z)
colormap(hot)
```

Výsledkem je obdélníkové pole buněk, jejichž barvy odpovídají hodnotám jednotlivých prvků vstupní matice v závislosti na zvolené mapě barev. Mapa barev je tvořena maticí o třech sloupcích, které určují intenzitu tří viditelných složek: červené, zelené a modré. V tomto případě zvolená mapa barev transformuje minimální hodnoty dat do černé barvy a maximální hodnoty do bílé barvy. Hodnoty mezi jsou transformovány na odstíny červené, oranžové a žluté. Volba této mapy barev je pro tento příklad vhodná, neboť spojitá oblast barev od černé k bílé se hodí pro zobrazení obrysů funkce `peaks`. Dobrou volbou by byla také šedivá mapa barev (`colormap(gray)`).

V kapitolách *Mapy barev* a *Ovládání barev* je uveden dostatek informací pro výběr vhodné mapy barev pro konkrétní aplikace.

Funkce `contour` a `pcolor` zobrazují v podstatě stejné informace v témže měřítku. Často je vhodné použít do jednoho grafu obě najednou. Abychom odstranili v grafu `pcolor` linky sítě (obrysy obdélníkových buněk), zvolíme typ odstínu `flat`. Vrstevnice vykreslíme černou čarou tak, že pro barvu čáry ve funkci `contour` definujeme barvu `'k'`.

```
colormap(hot)
pcolor(peaks)
shading flat
hold on
contour(peaks,20,'k')
hold off
```

12.2.5 Objekty image

MATLAB vytváří obraz (image) tím, že pro každý prvek v matici vyhledá přímo hodnotu barvy v mapě barev. Pro objekty `image` je příznačné, že mají své vlastní mapy barev.

12.2.5.1 Získání správného poměru vzhledu

Ve většině případů je pro vytvoření objektu `image` důležité použít vhodný poměr vzhledu obrazu, a tím předejít případnému zkreslení. Příkaz

```
axis('equal')
```

zajistí, že jsou objekty `image` zobrazovány ve správném poměru bez ohledu na změnu velikosti okna figure. Nechceme-li, aby byly osy a jejich popis viditelné, můžeme viditelnost os a jejich popisů vypnout příkazem

```
axis('off')
```

12.2.5.2 Porovnání objektů image a grafů, kreslených funkcí pcolor

Funkce `image` a `pcolor` jsou podobné. Obě vytváří dvourozměrné obrázky s jasem nebo hodnotami barev v závislosti na prvcích dané matice. Ale jak již jména funkcí napovídají, funkce `image` je zamýšlena pro zobrazování fotografií, obrazů apod., zatímco funkce `pcolor` pro zobrazování více abstraktních matematických objektů. Následující seznam uvádí rozdíly mezi těmito dvěma funkcemi pro zobrazení matice `A` typu (m, n).

- `image(A)` vytvoří pole o $m * n$ buňkách, zatímco `pcolor(A)` vytváří $m * n$ čar sítě, a tudíž pole o počtu pouze $(m-1) * (n-1)$ buněk.
- `image(A)` používá číslování os pomocí `'ij'`, `pcolor(A)` implicitně používá číslování os `'xy'`.
- `image(A)` užívá vždy obdélníkovou rovnoměrnou síť, zatímco `pcolor(X,Y,A)` může vytvořit parametrickou síť v jiném souřadnicovém systému.
- `image(A)` tvoří 2-D objekty, na které se můžeme dívat pouze ze standardního bodu pohledu (azimut=0°, elevace=90°), `pcolor(A)` tvoří plochu, na kterou se můžeme dívat z libovolného úhlu.
- funkce `image` používá prvky matice `A` pro vyhledání hodnot barev přímo v mapě barev. Tyto prvky jsou celá čísla v rozsahu od 1 do počtu prvků v mapě barev (`length(colormap)`). Vstupní matice ve funkci `pcolor` je transformována podle rozsahu barevné osy (`caxis`), funkce `image` není rozsahem `caxis` ovlivnitelná.
- `image` zmrazí osy tak, aby obraz úplně vyplnil celý rozsah os, `pcolor` ne.

12.2.6 Kreslení ploch

Funkce `mesh` a `surf` zobrazují plochy v prostoru. Jestliže `Z` je matice, jejíž prvky `Z(i,j)` definují výšku plochy nad základnou vyjádřenou sítí `(i,j)`, potom

```
mesh(Z)
```

zobrazí barevný drátový model plochy. Podobně

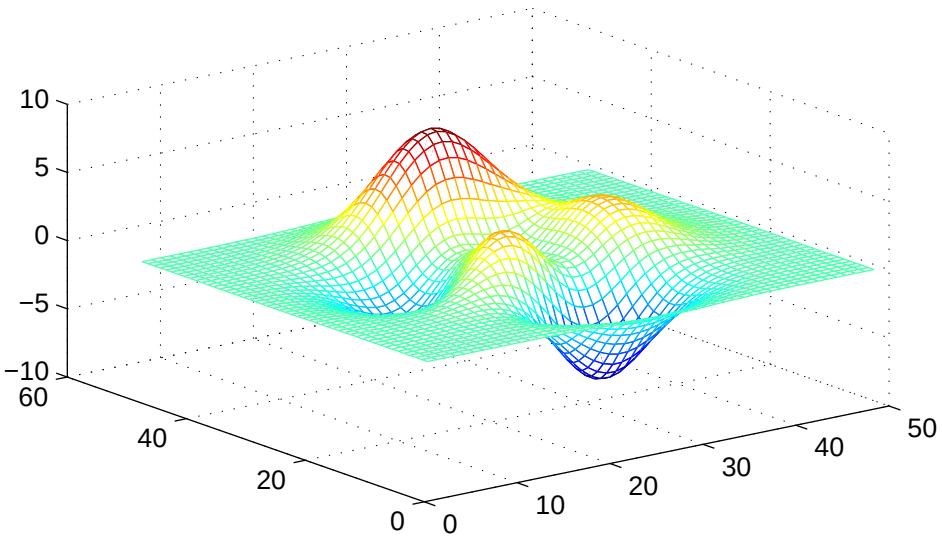
```
surf(Z)
```

zobrazí barevný plný model plochy. Plošky jsou obvykle čtyřúhelníky konstantní barvy, ohraničené černými čarami sítě. Funkcí `shading` můžeme tyto čáry odstranit nebo zvolit interpolované stínování plošek.

Mají-li funkce `mesh` a `surf` jako argument jedinou matici `Z`, pak tato matice definuje jak výšku, tak barvu plochy. Např.

```
mesh(peaks)
```

zobrazí drátový model plochy funkce `peaks`.



12.2.6.1 Volba barev

Následující příkazy se dvěma maticovými argumenty

```
mesh(Z,C)
```

```
surf(Z,C)
```

specifikují barvu použitou jako druhý argument. Hodnoty prvků matice `C` jsou transformovány a použity jako indexy v aktuální mapě barev (podobně jako u funkce `pcolor(C)`).

Následující příklad ukazuje, jak vhodnou volbou barev zlepšit informace zobrazené v grafu. V tomto příkladě je k obarvení plochy užit diskrétní Laplacián, který odečítá každý prvek vstupní matice od průměru získaného ze čtyř sousedních prvků (prvky na hraně nebo v rozích mají pouze tři nebo dva sousedy). Laplacián plochy má vztah k její křivosti, je kladný pro funkce tvaru `i^2+j^2` a záporný pro funkce tvaru `-(i^2+j^2)`. Funkce `del2` počítá diskrétní Laplacián libovolné matice. Vytvoření barevných polí aplikací Laplaciánu na vstupní data je výhodné, protože oblasti se shodnou křivostí jsou zobrazeny stejnou barvou. Např. pro barevnou mapu `hot` generují následující příkazy plochu funkce `peaks` s téměř bílými ploškami v místech největší kladné křivosti až po tmavé plošky v oblastech největší záporné křivosti

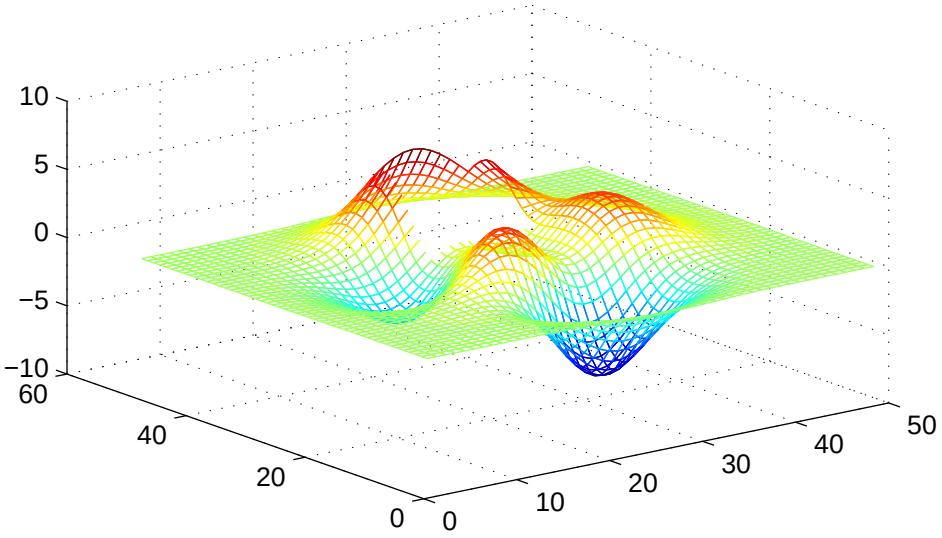
```
C = del2(peaks)
surf(peaks,C)
colormap(hot)
```

Nastavení barevného limitu os lze měnit funkcí `caxis` (viz kapitola *Ovládání barevné osy*, kde jsou uvedeny podrobnější informace o této funkci).

12.2.6.2 Vyříznutí části plochy hodnotami NaN

Jestliže data plochy obsahují prvky `NaN`, nejsou tyto prvky zobrazeny. Tím lze na odpovídajících místech vytvářet v ploše díry. Vhodný způsob, jak vytvořit na ploše neviditelné oblasti, je nahradit jednotlivé prvky v matici barev hodnotami `NaN`. Např. tyto příkazy vytvoří následující graf funkce `peaks`

```
p = peaks;
p(30:40,20:30) = nan*p(30:40,20:30);
mesh(peaks,p)
hidden off
```



12.2.6.3 Parametrické plochy

Funkce `mesh`, `surf` a `pcolor` mohou mít ještě dalších dva vektorové nebo maticové argumenty popisující plochu určenou k zobrazení. Je-li `Z` matice typu (m, n), `x` vektor délky n a `y` vektor délky m, potom

```
mesh(x,y,Z,C)
```

popisuje drátový model plochy, jejíž vrcholy mají barvu `C(i,j)` a jsou umístěny v bodech

```
(x(i), y(i), Z(i,j))
```

`x` odpovídá sloupcům matice `Z` a `y` jejím řádkům.

Obecněji, jsou-li `X`, `Y`, `Z` a `C` matice téhož typu, pak

```
mesh(X,Y,Z,C)
```

generuje drátový model plochy, jejíž vrcholy mají barvu `C(i,j)` a jsou umístěny v bodech `(X(i), Y(i), Z(i,j))`.

Totéž platí pro funkce `surf(X,Y,Z,C)`, atd.

Následující příklad ukazuje, jak použít sférické souřadnice pro vytvoření barevné koule se vzorem plusů a minusů v Hadamardově matici (ortogonální matice, která se používá v teorii zpracování signálů). Vektory `phi` a `theta` leží v rozmezí $-\pi/2 \leq phi \leq \pi/2$ a $-\pi \leq theta \leq \pi$. `Theta` je řádkový vektor a `phi` sloupcový vektor. Pomocí těchto vektorů jsou pak generovány matice `X`, `Y` a `Z`.

```
colormap(cool)
k = 5;
n = 2^k-1;
theta = pi*[-n:2:n]/n;
phi = (pi/2)*[-n:2:n]'/n;
X = cos(phi)*cos(theta);
Y = cos(phi)*sin(theta);
Z = sin(phi)*ones(size(theta));
C = hadamard(2^k);
surf(X, Y, Z, C)
```

Více informací o sférických souřadnicích obsahuje popis funkce `sphere` ve druhém dílu, *Popis funkcí*.

12.2.6.4 Obměny funkcí surf a mesh

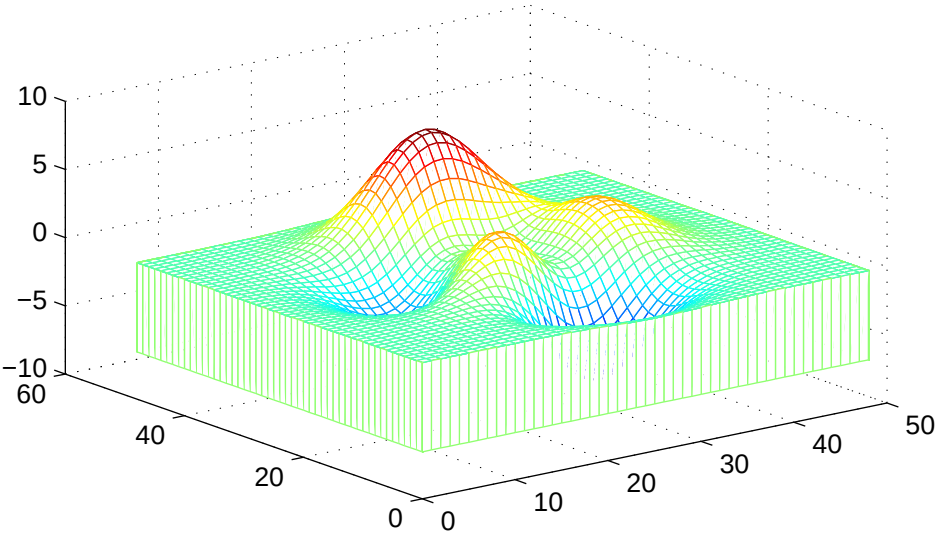
MATLAB obsahuje též funkce, které vytvářejí kombinované grafy. Např. funkce `surfc` vytvoří plný model plochy a navíc vykreslí v rovině $z = 0$ vrstevnice této plochy

```
surfc(peaks)
```

Funkce `meshz` vytvoří drátový model plochy s referenční rovinou v místě minimální hodnoty dat. Např.

```
meshz(peaks)
```

kreslí plochu definovanou funkcí `peaks` a rovinu v minimu této plochy. Tato rovina je spojena s rovinou $z = 0$ svislými čarami



Užitím funkce `surfl` můžeme vykreslit osvětlenou plochu včetně stínů. MATLAB počítá odrazy plochy kombinací rozptylu, zrcadlení a okolního osvětlení modelů. Můžeme zvolit směr bodového zdroje světla v kartézských nebo sférických souřadnicích. Následující příkazy vytvoří plochu a osvětlí ji z bodu s azimutem 10° a elevací 50°. Tyto příkazy užívají pole řádu 200, protože pro odrážející se světlo je důležité definovat data na dostatečně jemné síti, abychom z větší části odstranili diskretizační vlivy.

```
surfl(peaks(200),[-10,50])
colormap(gray)
shading flat
```

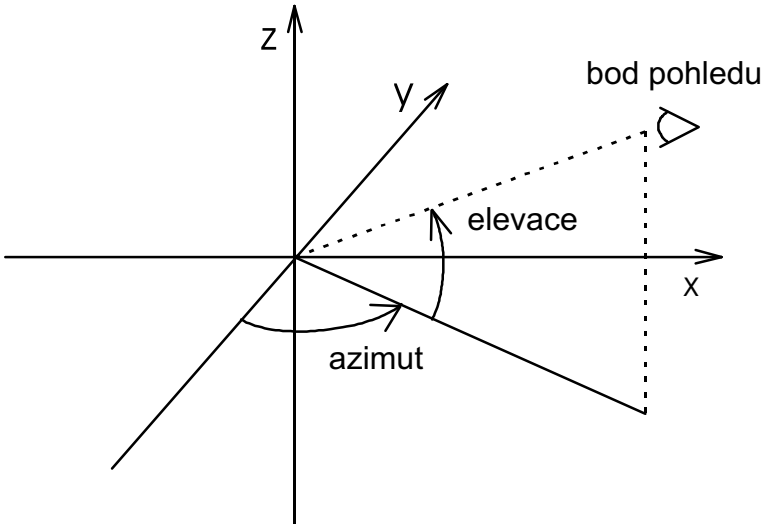
Výsledkem je plný osvětlený model plochy `peaks` v odstínech šedé barvy s neviditelnými čarami sítě.

12.3 Obecné grafické funkce

V této kapitole se seznámíme s tím, jak modifikovat graf, který byl již vytvořen některou z funkcí popsaných v předcházejících kapitolách.

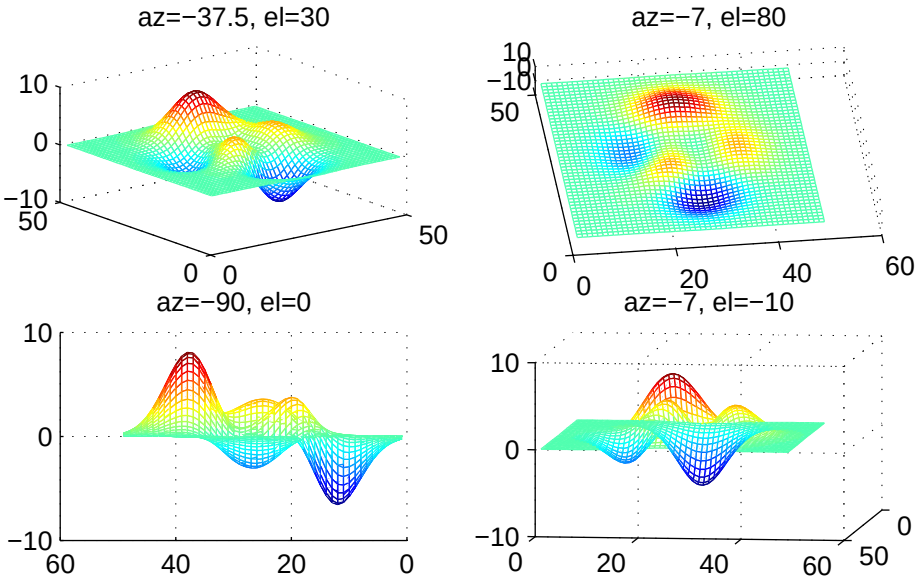
12.3.1 Funkce view

MATLAB umožňuje zvolit úhel pohledu na vytvořený 3-D graf. Zadáním azimutu a elevace bodu pohledu vzhledem k počátku nastaví funkce `view` úhel pohledu ve sférických souřadnicích. Azimut je úhel otočení v rovině $x-y$, který svírá průmět průvodiče bodu pohledu do roviny $x-y$ s osou y (s kladnými hodnotami ve směru proti pohybu hodinových ručiček), elevace je úhel mezi průvodičem bodu pohledu a jeho průmětem do roviny $x-y$. Ve směru kladné z-ové osy má elevace kladné hodnoty, ve směru záporné z-ové osy má záporné hodnoty.



Následující obrázek ukazuje čtyři rozdílné pohledy na matici `peaks`

```
view(-37.5,30), view(-7,80)
view(-90,0),    view(-7,-10)
```



12.3.2 Ovládání os funkcí axis

Funkce `axis` má řadu různých volitelných parametrů, které dovolují nastavit rozsah a poměr os. MATLAB obvykle najde minimální a maximální hodnotu vstupních dat a zvolí vhodný rozsah os a jejich popis. Použitím funkce `axis` můžeme též implicitně nastavené meze os změnit

```
axis([xmin xmax ymin ymax])
```

nebo pro 3-D grafy

```
axis([xmin xmax ymin ymax zmin zmax])
```

`axis('auto')` nastavuje rozsah os zpět na implicitní hodnoty (automatický režim).

`v=axis` uloží rozsah os aktuálního grafu do vektoru `v`. Chceme-li, aby následující grafické příkazy měly stejný rozsah os, zapíšeme za ně `axis(v)`. Tím se i dalšímu grafu přiřadí tentýž rozsah os.

`axis(axis)` způsobí zmrazení rozsahu os na aktuálních hodnotách.

`axis('ij')` nastaví MATLAB do maticového režimu os. Počátek souřadného systému je nyní v levém horním rohu, osa `i` je svislá číslovaná shora dolů, osa `j` je osa vodorovná číslovaná zleva doprava.

`axis('xy')` nastaví MATLAB na implicitní kartézský režim os. Počátek souřadného systému je v dolním levém rohu, osa `x` je vodorovná a je číslovaná zleva doprava, osa `y` je svislá a je číslovaná zdola nahoru.

`axis('square')` a `axis('equal')` mají vliv na poměr velikostí os a na vztah mezi x-ovou a y-ovou osou.

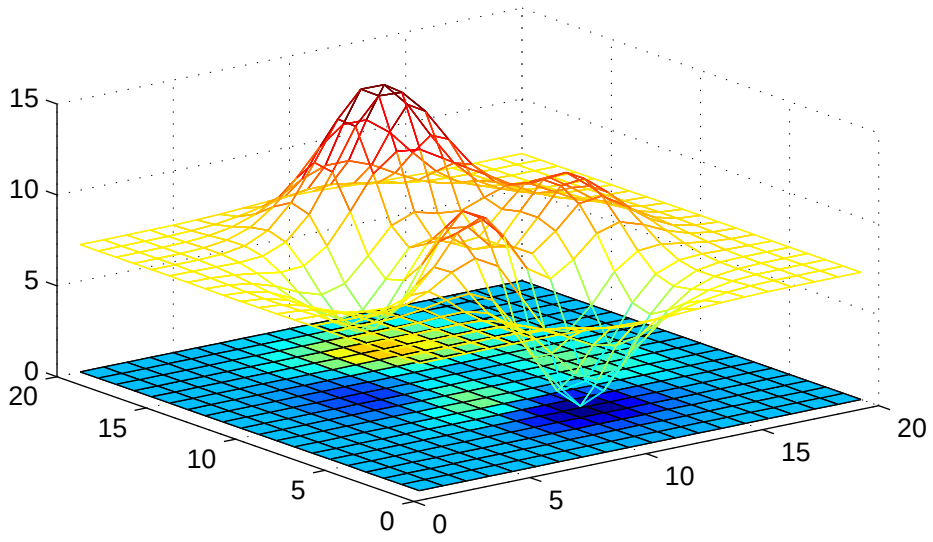
Popis a vyznačení vynášecích čárek na osách (tick) můžeme zapnout příkazem `axis('on')` nebo vypnout pomocí `axis('off')`. Podrobněji viz druhý díl, *Popis funkcí,* nebo `help axis`, kde jsou uvedeny další možné volby parametrů funkce `axis`.

Funkce `axis` pracuje s objektem `axes`, což je grafický objekt popisovaný v kapitole *Objektová grafika*.

12.3.3 Odstranění skrytých čar

Příkaz `hidden` dovoluje nebo zakazuje odstranit skryté čáry. Přidáváme-li do jednoho grafu další, může být vhodné zakázat odstranění skrytých čar. Následující příkazy zobrazují drátový model plochy `peaks` nad stejnou plochou generovanou funkcí `pcolor`. Protože příkaz `hidden off` neumožní odstranit skryté čáry, je drátový model průhledný a vidíme i celou plochu generovanou funkcí `pcolor`.

```
mesh(peaks(20)+7)
hold on
pcolor(peaks(20))
hidden off
```

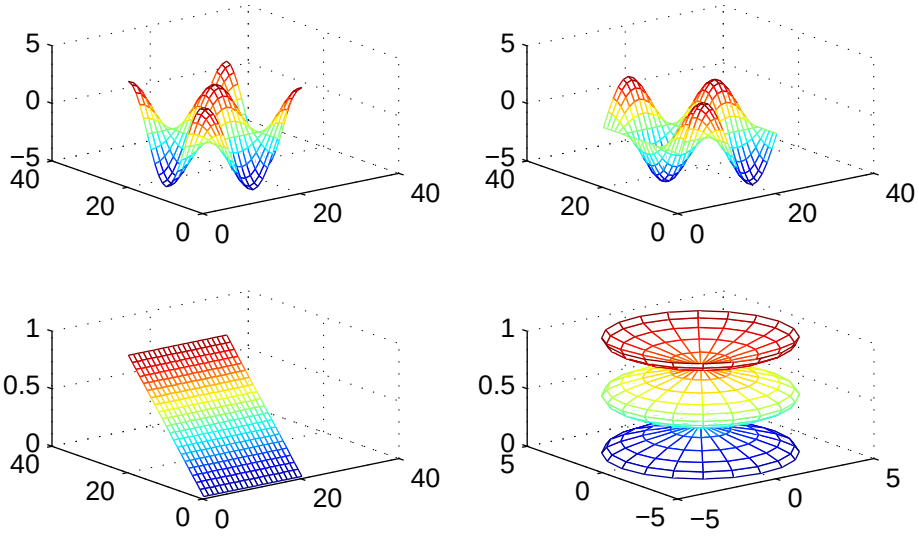


12.3.4 Funkce subplot

Funkcí `subplot` můžeme do stejného grafického okna umístit více grafů.

`subplot(m, n, p)` rozdělí grafické okno na `m*n` oblastí (os) a vybere pro aktuální graf `p`-tou z nich. Oblasti jsou číslovány po řádcích shora. Následující příkazy znázorní data do čtyř různých podoblastí grafického okna

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1)
mesh(X)

subplot(2,2,2)
mesh(Y)

subplot(2,2,3)
mesh(Z)

subplot(2,2,4)
mesh(X,Y,Z)
```



12.3.5 Funkce figure

Funkce `figure` bez argumentů otevře nové grafické okno.

`figure(N)` zaktualizuje `N`-té grafické okno, grafické příkazy budou nyní zobrazovat data do tohoto okna. Pokud `N`-té grafické okno neexistuje, MATLAB vytvoří další nové okno, ovšem ne nutně s číslem `N`.

12.3.6 Animace (movie)

MATLAB umožňuje uložit posloupnost snímků (grafických oken) pomocí funkce `getframe` a následně je zobrazit zpět pomocí funkce `movie`. Následující příklad generuje 16 snímků, které postupně zobrazují funkci `fft(eye(n))`.

```
M = movie(16);
for j = 1:16
  plot(fft(eye(j+16)))
  M(:,j) = getframe;
end
```

Funkce `getframe` vrací obraz snímku v pixlech. Každý snímek obsahuje binární data uložená ve sloupcovém vektoru. Obsah snímku nemá vliv na délku požadovaného sloupce, ale velikost snímku ano. Větší snímek vyžaduje více prostoru!

Funkce `moviein(n)` vytvoří dostatečně velkou matici pro uchování `n` sloupců, kde každý sloupec obsahuje jeden snímek. Tento krok není nutný, ale pokud není vytvořena matice `M`, kódování probíhá mnohem pomaleji, protože prostor pro `M` je vždy znovu přiřazován, když se objeví nový sloupec. Pokud byla animace již jednou generována, můžeme ji kdykoliv přehrát. Příkaz

```
movie(M,30)
```

ji přehraje třicetkrát. Na většině počítačů můžeme snadno generovat a plynule přehrávat snímky rychlostí až 12 snímků za sekundu. Delší animace však vyžaduje více paměti nebo velmi výkonný virtuální systém paměti.

12.3.7 Grafický vstup

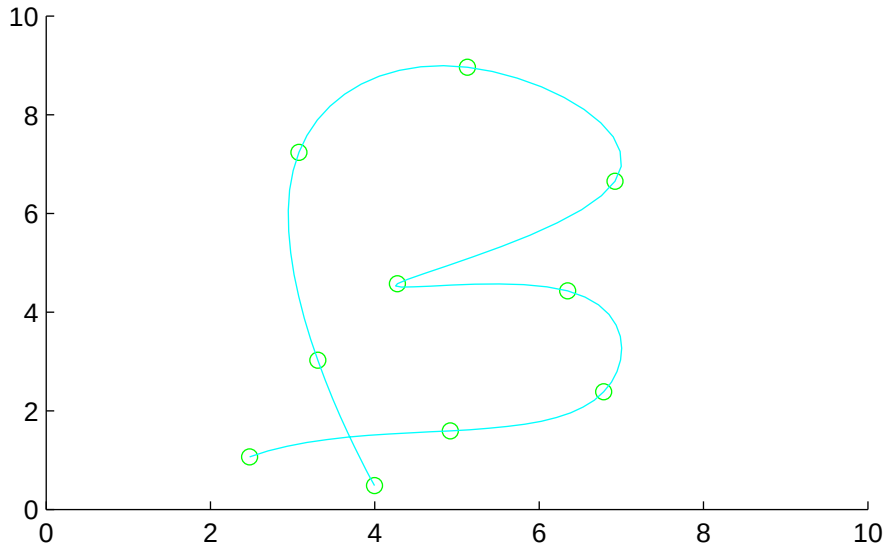
Funkce `ginput` umožňuje použít pro výběr bodů v grafu myš nebo šipky na klávesnici. Vrací souřadnice grafického kurzoru; buď aktuální pozici nebo pozici v okamžiku stisknutí tlačítka myši nebo klávesy.

Následující příklad popisuje, jak použít funkci `ginput` spolu s funkcí `spline` pro vytvoření 2-D křivky pomocí interpolace. (Tento příklad nevyužívá *Spline Toolbox*, který obsahuje více funkcí pro práci se spliny). Nejprve je funkcí `ginput` vybrána posloupnost bodů `[x,y]` a body jsou vyneseny do grafu. Označenými body jsou proloženy dva spliny s krokem 1/10 původní vzdálenosti a výsledek je opět vykreslen do grafu.

```
% Inicializace
clf
axis([0 10 0 10])
hold on

% Na počátku je seznam bodů prázdný
x = [ ];
y = [ ];
n = 0;

% Cyklus vybírání bodů
disp('Výběr bodů se provádí stisknutím levého tlačítka myši.')
disp('Výběr posledního bodu se provádí stisknutím pravého tlačítka myši.')
but = 1;
while but == 1
  [xi,yi,but] = ginput(1);
  plot(xi,yi,'go')
  n = n+1;
  x(n,1) = xi;
  y(n,1) = yi;
end

% Interpolace dvěma spliny
t = 1:n;
```

```
ts = 1:0.1:n;
xs = spline(t,x,ts);
ys = spline(t,y,ts);

% Graf interpolované křivky
plot(xs,ys,'c-');
hold off
```

Následující obrázek ukazuje jeden z typických výstupů.



12.3.8 Tisk grafických oken

Pomocí příkazu `print` můžeme vytvořit i PostScriptový výstup obsahu libovolného grafického okna. `print` posílá výstup přímo na naši implicitní tiskárnu nebo jej zapíše do zvoleného souboru, pokud zadáme jeho název. Dále můžeme zvolit typ PostScriptového souboru. Podporované typy jsou následující

PostScript	`-dps`
Color PostScript	`-dpsc`
Level 2 PostScript	`-dps2`
Level 2 color PostScript	`-dpsc2`
Encapsulated PostScript	`-deps`
Encapsulated color PostScript	`-depsc`
Encapsulated level 2 PostScript	`-deps2`
Encapsulated level 2 color PostScript	`-depsc2`

Např. příkaz

```
print meshdata -depsc2
```

uloží obsah aktuálního grafického okna jako PostScriptový soubor typu level 2 color do souboru `meshdata.eps`.

Obecně jsou soubory typu level 2 menší a tisk probíhá rychleji. Avšak ne všechny PostScriptové tiskárny podporují typ level 2, takže před použitím příkazu print je třeba znát schopnosti tiskárny.

Při nebarevném PostScriptu (tj. pokud nezadáme ve volbě zařízení znak c) vytváří MATLAB objekty `surface` a `patch` pomocí šedivých odstínů. Objekty `line` a `text` jsou tištěny černě nebo bíle.

Implicitně MATLAB mění původní černé pozadí na bílé a mění původní bílé osy (čáry a popis) na černé. Pokud tuto změnu nechceme (tj. chceme si podržet původní barvy), nastavíme vlastnost objektu figure `InvertHardCopy` na `off`:

```
set(gcf,'InvertHardCopy','off')
```

Pokud chceme získat výtisk odpovídající tomu, co je na obrazovce, nastavíme `InvertHardCopy` na `off` a určíme barevnou PostScriptovou tiskárnu (`-dpsc`, `-dpsc2`, `depsc` nebo `-depsc2`).

K vytisknutí souboru (např. `meshdata.eps`) na tiskárně bez PostScriptu slouží sharewarový program Ghostscript dodávaný s MATLABem.

12.4 Mapy barev a ovládání barev

MATLAB definuje mapu barev ve tvaru matice o třech sloupcích. Každá řádka matice definuje jednotlivou barvu na základě tří hodnot v rozsahu od 0 do 1 (RGB hodnoty). RGB hodnoty mají význam intenzity složek červené (R), zelené (G) a modré (B).

Některé typické barvy obsahuje následující tabulka

Červená	**Zelená**	**Modrá**	**Barva**
0	0	0	černá
1	1	1	bílá
1	0	0	červená
0	1	0	zelená
0	0	1	modrá
1	1	0	žlutá
1	0	1	fialová
0	1	1	tyrkysová
0,5	0,5	0,5	šedá
0,5	0	0	tmavě červená
1	0,62	0,4	měděná
0,49	1	0,83	akvamarinová

Je vhodné rozlišovat následující termíny:

Vyhledávací tabulka barev	závisí na hardware počítače
Mapa barev	tří-sloupcová matice s RGB hodnotami, která je přiřazena každému grafickému oknu v MATLABu a specifikuje jeho dostupné barvy
Mapa pseudocolor	mapa barev nezávislá na jednotlivém objektu image
Paleta barev	mapa barev sdružená s konkrétním objektem image

Počítače s 8-bitovou vyhledávací tabulkou barev umožňuje používat mapy barev až s 256 barvami, které jsou vybírány z možných barev. Používá-li jeden graf všech 256 tabulkových vstupů, potom barvy, které jsou požadovány operačním systémem pro zobrazení pozadí, hran a textů, musí být změněny.

Mapy barev mohou být zadány přímo ve tvaru matice nebo mohou být generovány operacemi MATLABu. Adresář `..\toolbox\matlab\color` obsahuje několik funkcí, které generují příslušné mapy barev, např. `hsv`, `cool`, `pink`, `cooper`, `flag` (viz *Zobrazování map barev*). Každá funkce má volitelný parametr `m`, který specifikuje počet řádků ve zvolené mapě barev (tj. počet barev). Např.

```
hot(m)
```

je matice typu `(m,3)`, jejímiž řádky jsou jednotlivé barvy (RGB vektory), které se mění od černé přes odstíny červené, oranžové a žluté až k bílé. Není-li určena délka mapy barev, MATLAB dosadí implicitní hodnotu `m=64`. To umožňuje, aby tři nebo čtyři grafická okna měla svůj vlastní oddíl ve vyhledávací tabulce barev s 256 barvami. Pokud je v každém z několika grafických oken použita delší mapa barev, může být pro operační systém nutné měnit různé vyhledávací tabulky barev v závislosti na momentálním aktivním okně.

Příkaz

```
colormap(M)
```

instaluje mapu barev matice `M` ve vyhledávací tabulce barev. Např.

```
colormap(hot)
```

instaluje 64 barev mapy barev `hot`.

Mapu barev používají funkce `mesh`, `surf`, `pcolor` a `image` a funkce z nich odvozené. Funkce `plot`, `plot3`, `contour` a `contour3` mapu barev nepoužívají.

Příkaz

```
surf(Z,C)
```

způsobí, že jsou prvky matice `C` transformovány tak, že minimum matice `C` odpovídá prvnímu řádku v mapě barev a maximum matice `C` poslednímu řádku. (K modifikaci této transformace můžeme také použít funkce `caxis`).

12.4.1 Ovládání barevné osy

K tomu, aby MATLAB přiřadil každé hodnotě matice určené k vykreslení (barevné pole) odpovídající barvu (index v mapě barev), provádí následující lineární transformaci:

Předpokládejme, že `c` je barevné pole, `cmin=min(c)`, `cmax=max(c)` a `m=length(colormap)`.

Jestliže `cmin<=c<cmax`,

```
index = fix((c-cmin)/(cmax-cmin)*m)+1
```

je-li `c==cmax`,

```
index = m.
```

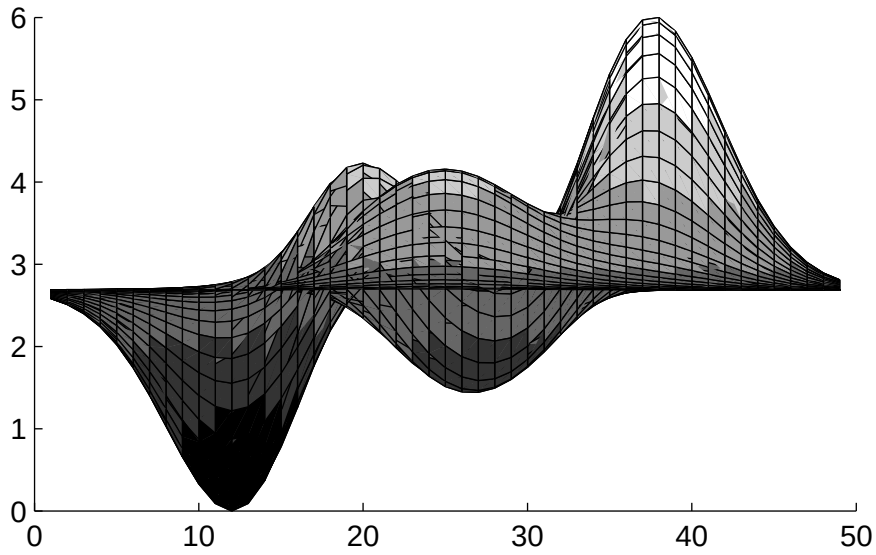
Pokud jsou hodnoty `cmin` a `cmax` zvoleny uživatelem (`cmin` nemusí být rovno `min(c)`, podobně `cmax` nemusí být rovno `max(c)`) a je-li `c<cmin`, `c>cmax` nebo `c==NaN`, je hodnota ustřižena, tj. je zneviditelněna.

POZOR! Od verze MATLABu 4.2 se používá pro hodnoty menší než `cmin` nebo větší než `cmax` jiného algoritmu. Je-li `c<cmin`, je `c` nastaveno na `cmin`, je-li `c>cmax`, je `c` nastaveno na `cmax`, tj. hodnoty již nejsou zneviditelňovány.

Následující příklad ukazuje vliv transformace. Zobrazíme plochu matice `peaks` v pohledu podél x-ové osy směrem k počátku a pro snadnější představu transformace upravíme rozsah dat od 0 do 6. Omezíme barevnou mapu `hsv` pouze na šest barev: červenou, žlutou, zelenou, tyrkysovou, modrou a fialovou. Příkazy

```
M = peaks;
M1 = M+abs(min(min(M)));
M = 6*M1./max(max(M1));
colormap(hsv(6))
surf(M)
view(90,0)
grid
```

generují graf, ve kterém jsou všechna data v rozsahu transformována na červenou barvu, všechna data z rozsahu na žlutou barvu atd. Minimální hodnota dat je transformována na první barvu v mapě barev (v tomto případě na červenou), maximální hodnota dat na poslední barvu mapy barev (fialovou). Tyto minimální a maximální hodnoty tvoří meze barevné osy, které můžeme změnit také funkcí `caxis`.

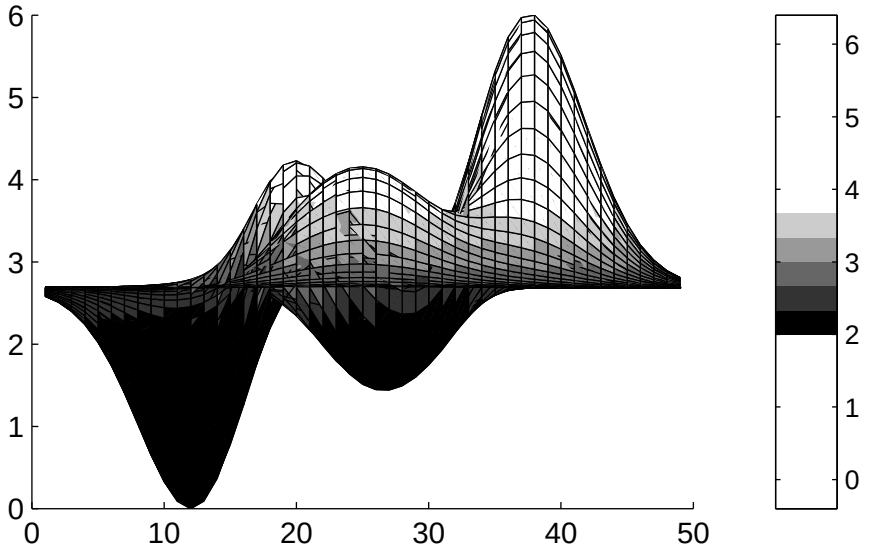


`caxis([cmin cmax])` umožňuje zvolit v prostoru dat minimální a maximální hodnoty (`cmin` a `cmax`), které určují, na jaké indexy mapy barev se data transformují. (Pokud určíme barevné pole pomocí `pcolor`, `mesh` nebo `surf`, MATLAB aplikuje meze barevné osy na tyto hodnoty, ne na data plochy).

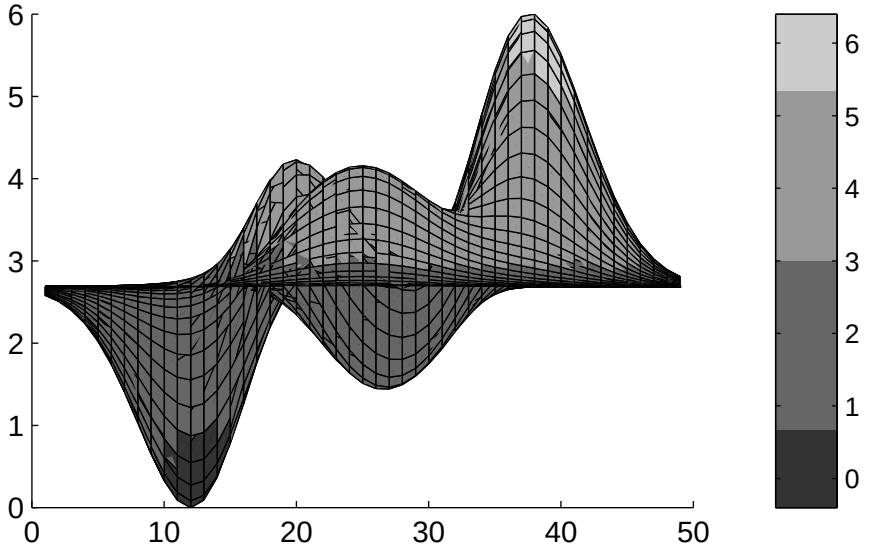
MATLAB implicitně přiřazuje proměnné `cmin` minimální hodnotu dat a proměnné `cmax` maximální hodnotu dat . Tím se využije pro celý rozsah dat úplný rozsah mapy barev. Funkci `caxis` můžeme použít i k dosažení následujících dvou efektů:

- Nastavíme-li `cmin`, popř. `cmax` na hodnoty, které jsou menší než rozsah dat plochy, potom data menší než `cmin` a větší než `cmax` se nebudou transformovat do barev.
  POZOR! Od verze MATLABu 4.2 se data menší než `cmin` resp. větší než `cmax` budou transformovat do krajních hodnot mapy barev, tj. `cmin`, resp. `cmax`.
- Nastavíme-li `cmin`, popř. `cmax` na hodnoty, které jsou větší než rozsah dat plochy, MATLAB transformuje mapu barev do většího rozsahu, jako kdyby data byla rozprostřena od `cmin` až do `cmax`. V důsledku toho jsou aktuální data zobrazena použitím pouze části mapy barev.

Uvažujme nastavení dat z předcházejícího příkladu, kdy jsou data v rozmezí od 0 do 6. Nastavením mezí barevné osy na `cmin=2` a `cmax=4` se transformují na úplnou barevnou mapu pouze hodnoty dat v tomto rozsahu. Data menší než 2 se transformují na `cmin` a data větší než 4 se transformují na `cmax` (Použita konvence MATLABu 4.2). Následující obrázek ukazuje tuto situaci pro data plochy `peaks`. Svislý proužek vlevo ukazuje rozsah barev použitých na ploše a korespondenci barev a datových hodnot.



Nastavíme-li naopak `cmin=-4` a `cmax=10`, je vykreslena celá plocha, ale s použitím pouze části mapy barev. V tomto případě MATLAB použije jen čtyři ze šesti barev mapy barev. Svislý proužek vlevo nyní ukazuje, na které barvy se hodnoty dat transformují.



12.4.2 Filozofie map barev

Je-li `M` mapa barev, pak příkaz

```
plot(M)
```

znázorní změny intenzit jednotlivých barevných složek (červená, zelená a modrá). Při zobrazení jsou tyto tři složky vykresleny žlutou, fialovou a tyrkysovou barvou. Funkce

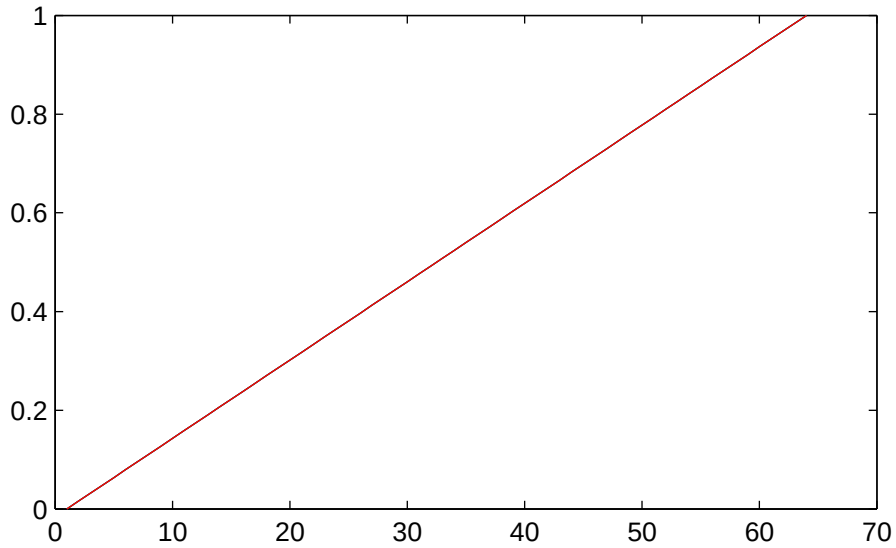
```
rgbplot(M)
```

je vykreslí červeně, zeleně a modře.

Velmi jednoduchá mapa barev je mapa `gray`, která je tvořena pouze odstíny šedé. V tomto případě jsou všechny tři sloupce matice této mapy barev stejné a mění se od 0 do 1. Graf generovaný příkazem

```
plot(gray)
```

zobrazí pouze tři přímé čáry na sobě:

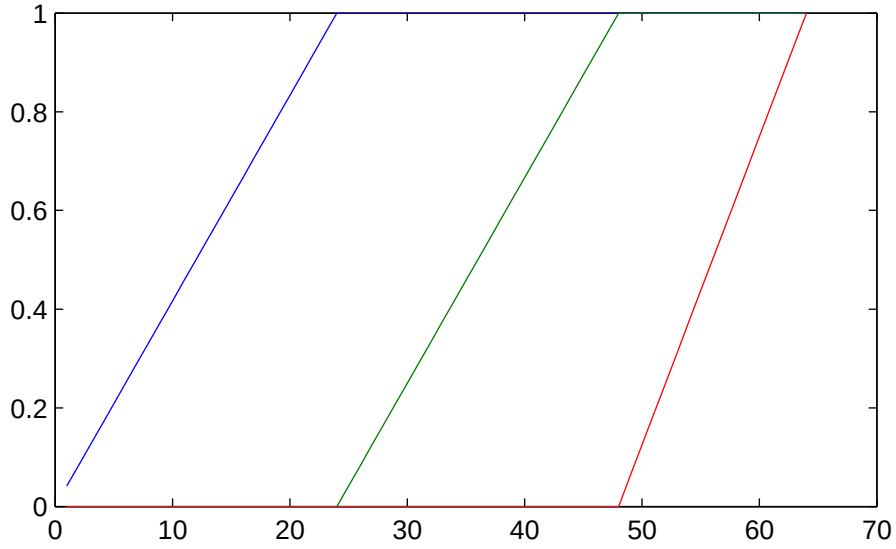


Mapa barev `hsv` je vhodná např. pro zobrazení periodických funkcí v polárních souřadnicích, viz demo MATLABu o funkcích komplexní proměnné.

Mapa barev `hot` se hodí pro neperiodické funkce. Začíná u černé barvy, pokračuje přes odstíny červené, oranžové a žluté, a končí bílou barvou. V matematickém zápisu se nejprve zvyšuje intenzita červené, potom zelené a nakonec modré barvy. Příkaz

```
plot(hot)
```

vytvoří následující graf:

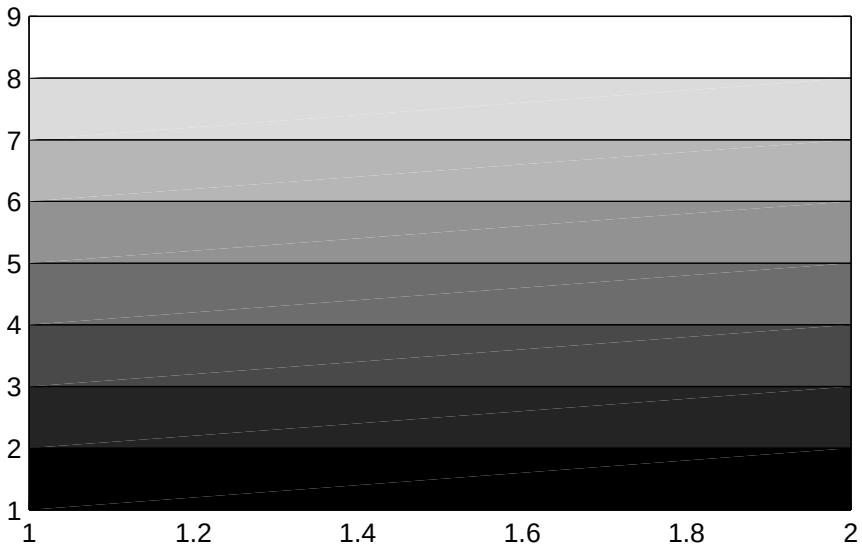


Tato mapa barev je také vhodná pro osvětlené modely, pokud je nejjasnější světlo poblíž bílé barvy.

12.4.2.1 Zobrazování map barev

K zobrazování map barev je vhodná funkce `pcolor`. Následující příkazy generují šedou mapu barev s osmi odstíny a vytvoří graf této aktuální mapy barev

```
colormap(gray(8));
pcolor([1:9; 1:9]')
```



V MATLABu je k dispozici deset map barev

`hsv`	Mapa barev hsv (Hue-saturation-value)
`gray`	Lineární šedá mapa barev
`white`	Bílá mapa barev
`hot`	Černo-červeno-žluto-bílá mapa barev
`cool`	Mapa barev s odstíny tyrkysové a fialové
`bone`	Šedá mapa barev se zabarvením do modra
`copper`	Mapa barev s lineárními tóny mědi
`pink`	Mapa barev s pastelovými odstíny růžové
`prism`	Mapa barev prism
`jet`	Varianta mapy barev hsv (přechod od fialové přes modrou a žlutou k červené; jako v kartografii)
`flag`	Mapa barev tvořená střídavě červenou, bílou, modrou a černou

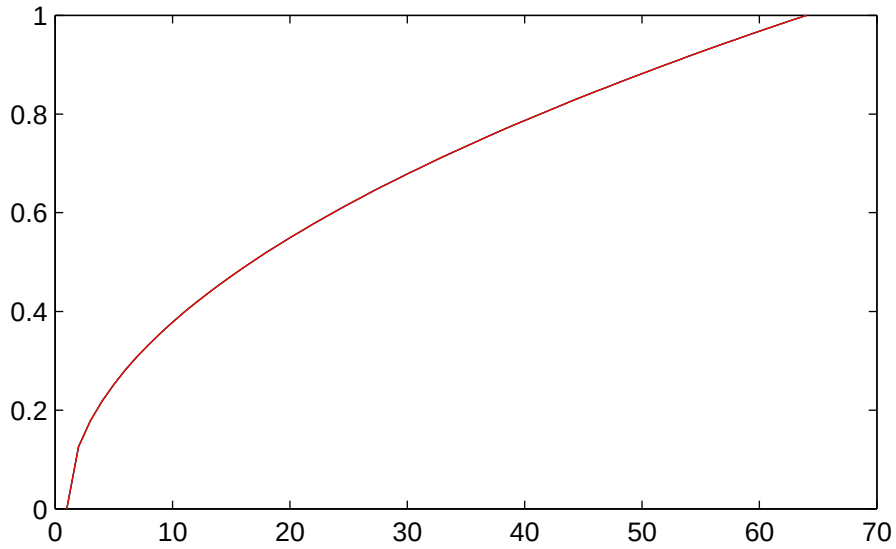
Všechny lze zobrazit funkcí `pcolor`.

12.4.2.2 Změna mapy barev

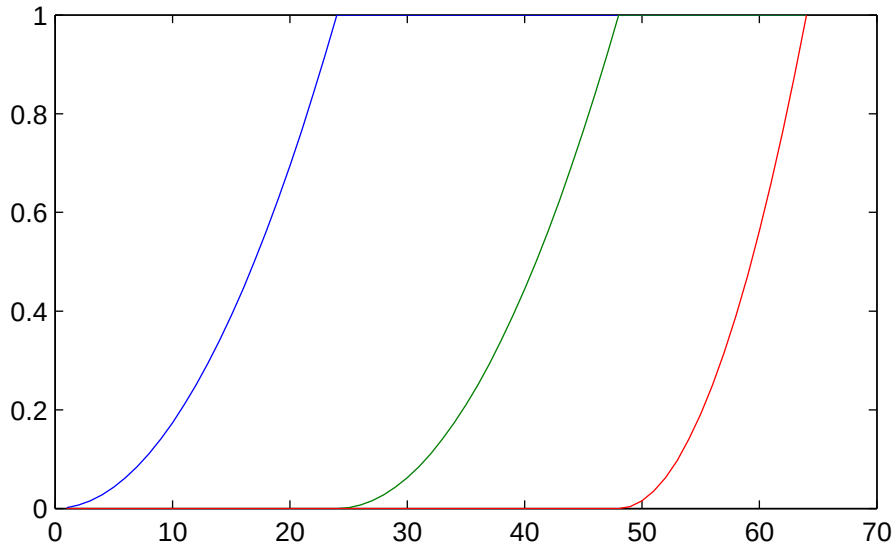
Mapy barev jsou ve skutečnosti matice, což nám dovoluje s nimi pracovat jako s ostatními maticemi. Výhodou funkce `brighten` je, že dokáže snížením nebo zvýšením intenzity tmavých barev upravit danou mapu barev. Např.

```
brighten(gray, 0.5)
```

má graf podobný $\sqrt{x}$:



Příkaz `brighten(hot,-0.5)`, který ztmavuje mapu barev, má graf se složkami tvarově podobný funkci x^2.



Mapy barev můžeme aritmeticky skládat, ale výsledky mohou být někdy nepředvídatelné, protože aritmetika v prostoru RGB je ošidná. Např. mapa nazvaná `pink` je tvořena kombinací map `gray` a `hot`

```
sqrt(2/3*gray+1/3*hot)
```

Jemné pastelové odstíny této mapy barev jsou někdy příjemnou změnou na rozdíl od hrubých tónů v mapách `hsv` a `hot`.

12.4.2.3 Palety barev

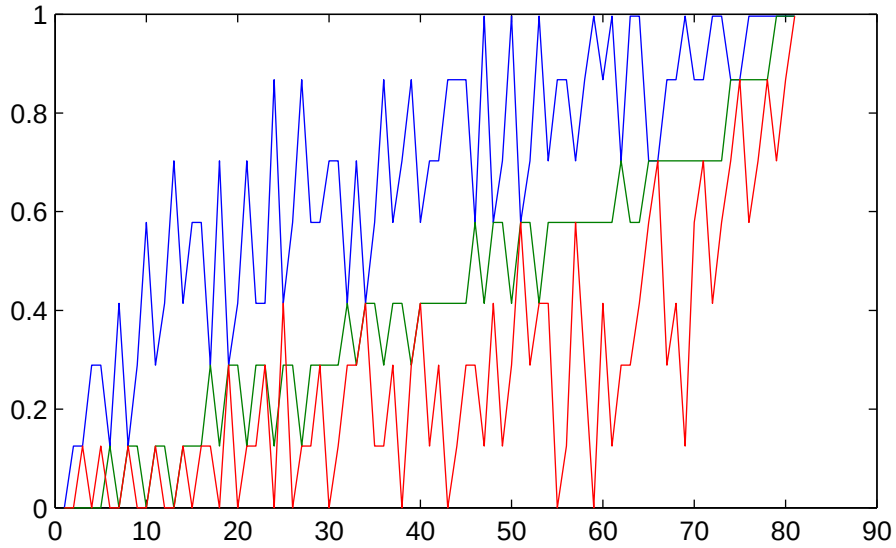
Palety barev fotografií jsou obvykle součástí aktuálních dat objektu image. Např.

```
load clown
```

jehož výsledkem je matice `X` typu `(200,300)` a mapa barev `map` o třech sloupcích a 81 řádku. Fotografii můžeme zobrazit příkazy

```
image(X)
colormap(map)
```

Následující graf zobrazuje použitou mapu barev `map`.



Vidíme, že v něm převažují červené složky (plná čára), proto je červená barva dominantní barvou v obrázku klauna. Mapa barev `map` je paletou barev, protože není pravděpodobné, že bude použita pro jiný objekt image než pro tuto matici `X`.

Abychom podrobněji zjistili, jak pracuje paleta barev, můžeme označit na nose klauna bod, např. `i=112`, `j=48`. Dostaneme odpověď, že bod `X(i,j)` má hodnotu 72. Protože minimální hodnota prvků v matici `X` je 0, maximální 80 a map má 81 řádek, mohou být prvky matice `X` zvětšené o 1 použity jako indexy mapy barev `map`. Pro námi zvolený bod `X(i,j)` zvětšený o hodnotu 1 dostaneme 73. řádek v mapě barev

```
map(X(i,j)+1,:) = map(73,:)
```

což je vektor `[0.9961 0.7031 0.5781]`. Této hodnotě odpovídá jasně červená barva.

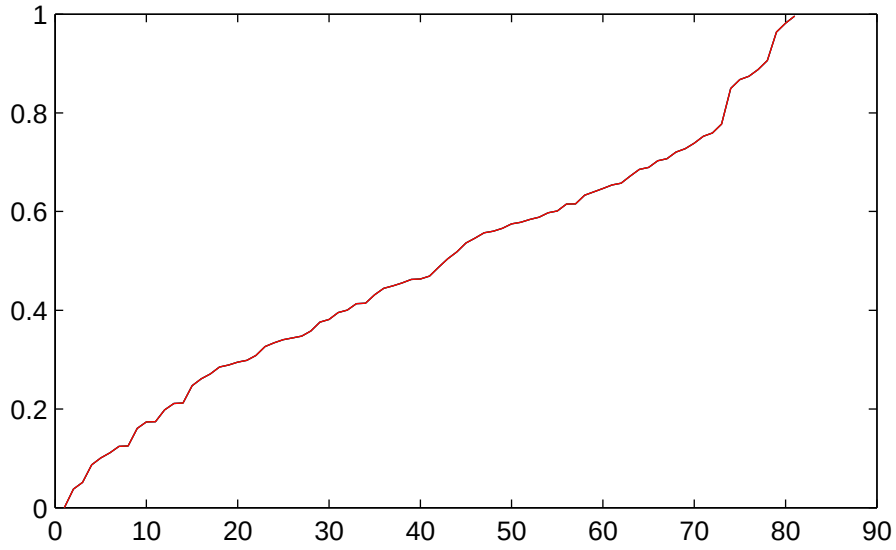
Složky jasu televizního signálu používají barevné kódovací schéma NTSC ($0.30 * red + 0.59 * green + 0.11 * blue$):

```
b = sum(diag([0.30 0.59 0.11])*map')';
```

Je-li paleta barev permutována tak, že tato funkce je monotónně rostoucí, potom objekt image vypadá uspokojivě i v případě, je-li mapa barev změněna na lineární šedou mapu. Použitím nelineárního měřítka šedé mapy

```
colormap([b b b])
```

se konvertuje barevný objekt image do černobílého NTSC ekvivalentu. Následující graf znázorňuje jas obrazu klauna.



Graf je téměř monotónně rostoucí, což znamená, že můžeme klaunovu barevnou mapu nahradit šedou mapou

```
colormap(gray)
```

nebo dokonce i některou umělou pseudomapou např. `pink` a výsledný objekt image vypadá stále uspokojivě.

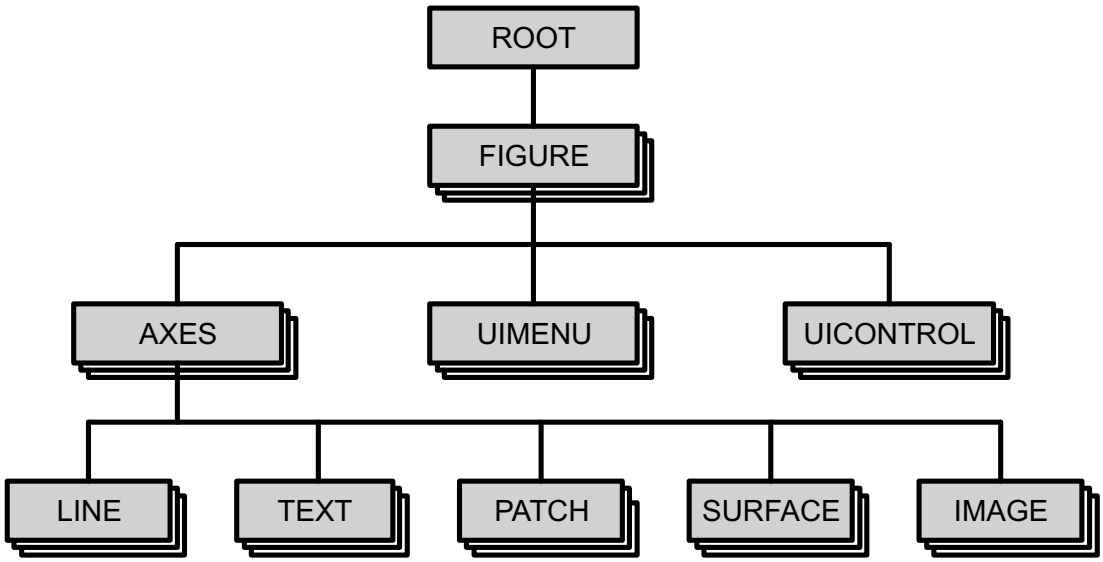
12.5 Objektová grafika

Grafické funkce, o kterých jsme se dosud zmínili, byly tzv. funkce vyšší úrovně. Pro generování složitějších a komplikovanějších grafů obsahuje grafický systém MATLABu také množství funkcí nižší úrovně, kterými můžeme vytvářet čáry, plochy a další grafické objekty a modifikovat je. Tento systém se nazývá Objektová grafika.

12.5.1 Grafické objekty

MATLAB definuje grafické objekty jako základní grafické jednotky svého grafického systému a organizuje je do stromově strukturované hierarchie. Tyto objekty zahrnují root (obrazovka), figure (grafické okno), axes (osy), line (čára), patch (vyplněný mnohoúhelník), surface (plocha), image (fotografie), text, uicontrol (uživatelem řízené rozhraní) a uimenu (uživatelské menu).

Následující obrázek ukazuje hierarchickou strukturu těchto objektů.



- Objekt *root* je kořenem této hierarchie. Odpovídá obrazovce počítače. Objekt root je jediný, všechny další objekty jsou jeho potomky.
- Objekty *figure* jsou samostatná grafická okna. Těchto objektů může existovat libovolný počet, všechny jsou dětmi objektu root. Všechny další grafické objekty jsou potomky objektu figure a jsou zobrazeny v tomto objektu (grafickém okně). Všechny funkce, které generují objekty, a všechny funkce vyšší úrovně vytvoří objekt figure i v případě, kdy tento objekt ještě neexistuje. Pomocí funkce `figure` můžeme objekt figure vytvořit přímo.
- Objekty *axes* definují oblast v grafickém okně a orientují své děti uvnitř této oblasti. Axes jsou dětmi objektů figure a jsou rodiči pro objekty line, surface, text, image a patch. Všechny funkce, které generují tyto objekty, a všechny funkce vyšší úrovně vygenerují objekt axes i v případě, kdy tento objekt ještě neexistuje. Přímo lze objekt axes generovat funkcí `axes`.
- Objekty *line* jsou základní grafické jednotky užívané pro generování většiny 2-D a některých 3-D grafů. Jsou dětmi objektu axes a jejich umístění je určeno souřadným systémem, který byl definován jejich rodiči. Objekty line jsou vytvářeny funkcemi `plot`, `plot3`, `contour` a `contour3`.
- Objekty *patch* jsou vyplněné mnohoúhelníky s hranami. Jsou dětmi objektu axes a jejich umístění je určeno souřadným systémem, který byl definován jejich rodiči. Tyto objekty mohou být vybarveny. Grafické objekty patch jsou generovány funkcemi `fill` a `fill3`.
- Objekty *surface* jsou třírozměrnou reprezentací matice dat. Jsou složeny ze čtyřúhelníků, jejichž vrcholy jsou definovány prvky matice. Objekty surface mohou být kresleny jako plné nebo barevně interpolované, nebo pouze drátové (mesh). Jsou dětmi objektu axes a jejich poloha je určena souřadným systémem, který byl definován jejich rodiči. Objekty surface jsou generovány funkcemi `pcolor` a `surf` a skupinou funkcí `mesh`.
- Objekty *image* jsou výsledkem transformace prvků matice na indexy aktuální mapy barev. Objekty image mívají zpravidla své vlastní mapy barev, palety barev, které definují barvy použité pouze v konkrétním objektu image. Objekty image jsou zpravidla dvourozměrné, a proto nemohou být vidět z jiného úhlu, než je implicitní 2-D pohled. Jsou dětmi objektu axes a jejich poloha je určena souřadným systémem, který byl definován jejich rodiči. Tyto objekty jsou generovány funkcí `image`.
- Objekty *text* jsou znakové řetězce. Jsou dětmi objektu axes a jejich poloha je určena souřadným systémem, který byl definován jejich rodiči. Objekty text jsou generovány funkcemi `text`, `gtext`, `title`, `xlabel`, `ylabel`, `zlabel`.
- Objekty *uicontrol* jsou uživatelem řízená rozhraní, která vykonávají příslušné funkce, jsou-li k tomu uživatelem vyzvány. Jsou dětmi objektů figure, a jsou proto nezávislé na objektu axes.

- Objekty *uimenu* jsou uživatelská menu, která nám umožňují vytvořit si v objektu figure vlastní menu. Jsou dětmi objektu figure, a tedy jsou nezávislé na objektech axes.

12.5.1.1 Identifikátory objektů (handle)

Každý samostatný grafický objekt má svůj vlastní identifikátor, tzv. handle, který je tomuto objektu přiřazen při jeho vytvoření. Některé grafy, např. vrstevnice, jsou složeny z několika objektů a každý z nich má svůj vlastní identifikátor, tj. každá vrstevnice má svůj identifikátor.

Identifikátor objektu root je vždy nulový. Identifikátor objektu figure je celé kladné číslo, které je implicitně zobrazeno v názvu grafického okna. Identifikátory ostatních objektů jsou reálná čísla, která obsahují informace používané MATLABem.

POZOR!!! Při odkazech na tyto identifikátory je nutné zachovat jejich úplnou přesnost. Proto je lepší uložit jejich hodnotu do proměnné a použít tuto proměnnou vždy, když je identifikátor požadován, než přečíst identifikátor z obrazovky a přepsat jej ručně.

K jednoduchému přístupu k identifikátorům objektů definuje MATLAB následující funkce:

`gcf`	vrací identifikátor aktuálního objektu figure
`gca`	vrací identifikátor aktuálního objektu axes

Tyto funkce můžeme použít jako vstupní argumenty pro jiné funkce, které požadují identifikátor objektů figure nebo axes.

Libovolný objekt lze zrušit funkcí `delete` s použitím identifikátoru tohoto objektu jako argumentu. Např. můžeme vymazat aktuální osy, a tím i všechny jejich děti, příkazem

```
delete(gca)
```

Všechny funkce MATLABu, které vytvářejí objekty, vrací identifikátory (nebo vektor identifikátorů) vytvořených objektů. A to jak funkce vyšší úrovně jako `surfc` (generuje jak plochu, tak čáry), tak i funkce nižší úrovně, které generují pouze jeden objekt, např. funkce `surface`.

12.5.1.2 Funkce vytvářející objekty

Všechny objekty mohou být generovány funkcemi, které mají totéž jméno jako jimi generovaný objekt (funkce `text` vytvoří objekt text, funkce `figure` vytvoří objekty figure atd.). Grafické funkce vyšší úrovně volají odpovídající funkce nižší úrovně.

Mnoho funkcí vyšší úrovně, k získání podrobného přesného výsledku, také nastavuje vlastnosti objektů. Mohou např. změnit rozsah objektu axes nebo nastavit implicitní 2D nebo 3-D pohled.

Funkce nižší úrovně generují jeden z devíti grafických objektů (nemůžeme vytvořit nový objekt root) a umístí jej do odpovídajícího objektu rodičů. Např. vyvoláme-li funkci

```
line
```

MATLAB vykreslí použitím implicitních dat čáru do aktuálního objektu axes. Neexistuje-li objekt axes, MATLAB jej vytvoří. Pokud neexistuje ani objekt figure (grafické okno), ve kterém se bude vytvářet objekt axes, MATLAB vytvoří i tento objekt figure. Voláme-li funkci `line` podruhé, vykreslí se druhá čára do již existujícího objektu axes (na rozdíl od funkce `plot`, která generuje nové osy). Tato vlastnost funkce `line` je vhodná v případě, chceme-li přidat čáru do již existujícího obrázku. Stejného výsledku lze ale také dosáhnout příkazem `hold`.

12.5.2 Vlastnosti objektů

Všechny objekty mají vlastnosti, které rozhodují o tom, jak budou tyto objekty zobrazeny. Tyto vlastnosti zahrnují jak obecné informace (typ objektu, jeho rodiče a děti, zda je nebo není objekt viditelný), tak i informace jedinečné pro jednotlivý typ objektu (např. rozsah x-ové osy objektu axes nebo data, kterými je definován objekt surface).

Tvořený grafický objekt je inicializován množinou implicitních hodnot vlastností.

Aktuální hodnoty všech vlastností můžeme získat a většinu z nich specifikovat. Některé vlastnosti jsou nastaveny MATLABem a jsou určeny pouze ke čtení. Hodnoty vlastností se aplikují jednoznačně na konkrétní objekt, nastavení hodnoty pro jeden objekt neovlivňuje hodnotu u ostatních objektů téhož typu.

Ve druhém dílu této knihy, *Popis funkcí*, jsou podrobně popsány vlastnosti každého grafického objektu.

12.5.2.1 Vlastnosti společné všem objektům

Některé vlastnosti jsou společné pro všechny grafické objekty: `ButtonDownFcn`, `Children`, `Clipping`, `Interruptible`, `Parent`, `Type`, `UserData` a `Visible`.

Ve druhém dílu jsou u popisu každé funkce, která generuje grafické objekty, uvedeny informace o tom, jak použít každou tuto vlastnost pro konkrétní objekt.

12.5.2.2 Poznámka o názvech vlastností

Podle zvyklostí dává MATLAB u názvů vlastností objektů vždy první písmeno každého slova velké, např. `LineStyle` nebo `XMinorTickMode`. Tento způsob je vhodný pro jejich lepší čtení. MATLAB nekontroluje v názvech vlastností velikost písmen, proto lze pro správnou identifikaci názvu vlastnosti použít písmena libovolné velikosti. Lze dokonce použít i zkrácených názvů, ale tak, aby tato zkratka jednoznačně určovala danou vlastnost.

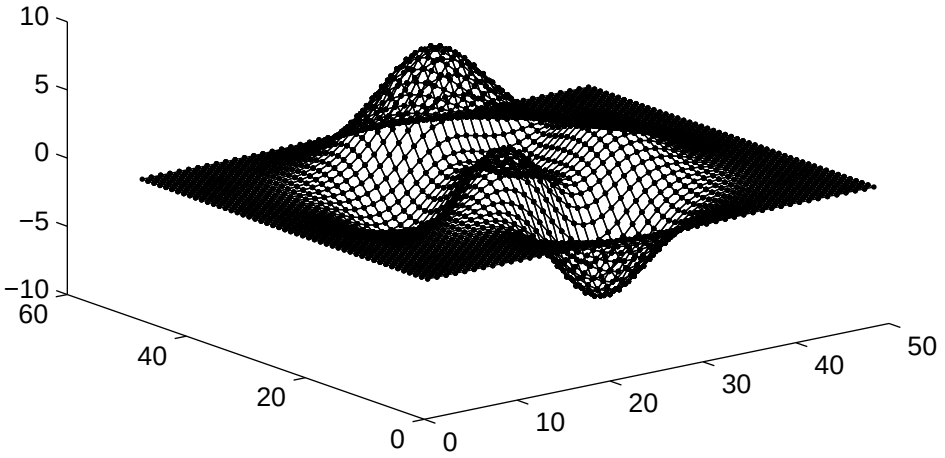
POZOR! Název vlastnosti uvedený v apostrofech nesmí obsahovat žádné mezery.

12.5.2.3 Nastavení a získání vlastností objektů

MATLAB poskytuje dva způsoby pro nastavení hodnot vlastností. Můžeme určit vlastnost objektu buď v době, kdy voláme funkci, která příslušný grafický objekt generuje, nebo můžeme nastavit hodnotu vlastnosti po vytvoření objektu pomocí funkce `set`. Např. příkazy

```
figh = figure('Color', 'white')
axh = axes('View', [-37.5 30], 'XColor', 'k', 'YColor', 'k', 'ZColor', 'k')

surfh = surface(peaks, 'FaceColor', 'none', 'LineStyle', '.')
```

vytvoří tři objekty a přiřadí hodnoty vlastnostem, u kterých nechceme použít implicitně nastavené. Následující obrázek ukazuje výsledek.



Vytvořilo se grafické okno s bílým pozadím, původní nastavení ostatních vlastností objektu figure nebylo změněno. Dále se vygeneroval objekt axes, kde jsme nastavením černé barvy pro všechny souřadné osy změnily implicitní bílou barvu (která by samozřejmě na bílém pozadí nebyla vidět).

Vlastnost `View` objektu axes byla nastavena na azimut $-37.5°$ a elevaci $30°$, a tím byly přepsány implicitní hodnoty $0°$ a $90°$.

Dále byla vyvolána funkce `surface`, která generuje plochu funkce `peaks`. Vlastnost `FaceColor` (barva čtyřúhelníků, ze kterých je plocha složena) je nastavena na hodnotu `none`. Tím se zabrání vykreslování čelních plošek čtyřúhelníků, ale neovlivní se tím jejich hrany (existuje také vlastnost `EdgeColor`). Protože hrany jsou čáry, je jejich výskyt řízen vlastností `LineStyle`. V tomto případě je každý vrchol zobrazen jako bod.

Identifikátor objektu je uložen do proměnné vždy, když je funkce (`figure`, `axes` a `surface`), která tento objekt vytváří, vyvolána. Následující kapitola ukazuje, jak změnit vlastnosti již existujících objektů nastavením těchto identifikátorů.

12.5.2.4 Funkce set a get

Vlastnosti objektu můžeme nastavit také až po jeho vytvoření. K tomu využijeme identifikátory, které vracejí vytvářející funkce.

Funkce `set` umožňuje nastavit vlastnosti objektu pomocí identifikátoru objektu a dvojice PropertyName/PropertyValue.

V předcházejícím příkladě byl definován objekt surface a do proměnné `surfh` byl uložen jeho identifikátor. Vlastnost plochy `LineStyle` můžeme nyní změnit z tečkované čáry použité v obrázku na plnou čáru příkazem

```
set(surfh, 'LineStyle', '-')
```

Chceme-li se na plochu podívat z jiného úhlu, změníme vlastnost `View` objektu axes

```
set(axh, 'View', [-45 45])
```

Seznam všech nastavitelných vlastností pro konkrétní objekt obdržíme vyvoláním funkce `set` s identifikátorem objektu:

```
set(surfh)

Cdata
EdgeColor: [ none | {flat} | interp ] -or- a ColorSpec.
EraseMode: [ {normal} | background | xor | none ]
FaceColor: [ none | {flat} | interp | texturemap ] -or- a ColorSpec.
LineStyle: [ {-} | -- | : | -. | + | o | * | . | x ]
LineWidth
MarkerSize
MeshStyle: [ {both} | row | column ]
Xdata
Ydata
Zdata
ButtonDownFcn
Clipping: [ {on} | off ]
Interruptible: [ {no} | yes ]
Parent
UserData
Visible: [ {on} | off ]
```

Chceme-li znát hodnoty nastavených vlastností, použijeme funkci `get`.

`get(h)`, kde `h` je identifikátor objektu, vrací seznam všech vlastností pro tento objekt s aktuálními hodnotami každé vlastnosti. Např.

```
get(surfh)

CData = [ (49 by 49) ]
EdgeColor = [0 0 0]
EraseMode = normal
FaceColor = none
LineStyle = -
LineWidth = [0.5]
MarkerSize = [6]
MeshStyle = both
XData = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
YData = [ (49 by 1) ]
ZData = [ (49 by 49) ]

ButtonDownFcn =
Children = [ ]
Clipping = on
Interruptible = no
Parent = [0.000366211]
Type = surface
UserData = [ ]
Visible = on
```

Uvedeme-li identifikátor objektu a název vlastnosti, obdržíme aktuální hodnotu této vlastnosti. Chceme-li např. určit minimální hodnotu plochy, získáme příslušná data z vlastnosti `ZData` a funkcí `min` určíme minimální hodnotu:

```
z = get(surfh, 'zdata');
min(min(z))

ans =
-6.5466
```

Uvažujme nyní situaci, kdy máme na obrazovce řadu objektů a chceme zjistit identifikátory všech objektů v aktuálních osách, tj. všech dětí objektu axes:

```
h = get(gca, 'Children');
```

Obecně je `h` sloupcový vektor identifikátorů, jehož prvky jsou seřazeny podle pořadí, ve kterém byly objekty zobrazovány. Prvním prvkem je identifikátor objektu, který byl zobrazen naposledy.

Funkcí `get` a vlastností `Type` můžeme také určit, který typ objektu odpovídá jednotlivým identifikátorům:

```
get(h(1), 'Type')
get(h(2), 'Type') ...
```

Vygenerujme funkcí `surfc(peaks)` plochu a odpovídající vrstevnice. Protože grafické funkce vyšší úrovně také vrací identifikátory jednotlivých objektů, můžeme tyto identifikátory získat zadáním proměnné `h` jako výstupního argumentu této funkce

```
h = surfc(peaks)

h =
1.0002
6.0001
7.0001
8.0001
9.0001
10.0001
11.0001
12.0001
13.0001
```

Tato funkce vytváří objekt surface a řadu objektů line. Abychom mohli identifikovat jednotlivé objekty, použijeme funkci `get` a vlastnost `Type` pro jednotlivé identifikátory

```
get(h(1), 'Type')

ans =
surface

get(h(2), 'Type')

ans =
line
```

Jestliže máme objekt jednou již identifikovaný, můžeme změnit libovolnou jeho vlastnost bez „roztržení“ celého grafu. Tuto techniku můžeme použít pro přístup k jednotlivým grafickým objektům, a tím modifikovat chování funkcí MATLABu, jak ukazuje následující příklad.

Funkce `surfc(peaks)` generuje plochu s využitím aktuální mapy barev a vrstevnice s implicitně definovanými barvami (viz kapitola *Typy čar, značky a barvy*).

Chceme-li změnit barvy vrstevnic tak, aby souhlasily s barvami plochy v odpovídajících výškách, můžeme v každém samostatném objektu line nastavit vlastnost `Color`. Abychom zjednodušili proces identifikace čar, definujme následující funkci MATLABu:

```
function l = findlines(h)
l = [ ];
for jj = h'
  if strcmp(get(jj, 'Type'), 'line')
    l = [l; jj];
  end
end
```

Tato funkce vezme vektor identifikátorů `h`, který je získán funkcí `surfc`, a vrátí identifikátory všech objektů line.

Definujme dále funkci, která nastaví barvu každé čáry:

```
function changelines(h)
lh = length(h);
map = colormap;
lm = length(map);
for jj = 1:lh
  set(h(jj), 'Color', map(jj*lm/lh,:))
end
```

Funkce `changelines` používá k nastavení vlastnosti `Color` pro každou čáru podle barvy aktuální mapy barev funkci `set` tak, že všechny čáry obsáhnou celý rozsah mapy barev.

Nyní sestavíme pro barevné vrstevnice plochy novou funkci MATLABu, která použije funkce `findlines`, `changelines` a `surfc`, a nazveme ji `surfcc`:

```
function out = surfcc (x, y, z, c)
error(nargchk(1, 4, nargin));
if nargin == 1,
  h = surfc(x);
elseif nargin == 2,
  h = surfc(x, y) ;
elseif nargin == 3,
  h = surfc(x, y, z);
elseif nargin == 4,
  h = surfc(x, y, z, c);
end
hlines = findlines(h);
changelines(hlines)
if nargout > 0
  out = h;
end
```

Funkce `surfcc` kontroluje nejprve počet vstupních parametrů a volá funkci `surfc` s požadovaným počtem argumentů. Pak volá funkce `findlines` a `changelines`. Výsledkem je vygenerovaná plocha funkce `peaks` a vrstevnice, jejichž barva souhlasí s barvou odpovídajících výšek na ploše.

12.5.2.5 Fonty

Objekty text nám umožňují měnit velikost fontů a typ písma. Tím získáváme více možností pro úpravu vzhledu popisů os a nadpisů výsledného grafu. Podobně lze též specifikovat typ písma a velikost popisu os. Více informací viz popis vlastností objektu text ve druhém dílu, *Popis funkcí*.

12.5.2.6 Šířka čáry

Šířku čáry jednotlivých grafických objektů můžeme nastavit vlastností `LineWidth`, např.

```
set(identifikátor, 'LineWidth', 2)
```

kde nová šířka je zadaná v bodech (1 bod = 1/72 palce). Tato vlastnost je přístupná v objektech axes, line, surface a patch.

12.5.2.7 Transformace ploch typu texture

Texture je technika transformace 2-D obrazu na 3-D plochu, kdy se barevná data přizpůsobí tvaru 3-D plochy. Tím lze na 3-D plochu aplikovat různé textury, jako např. povrchy materiálů, bez složitého 3-D geometrického modelování výsledné plochy s těmito rysy. Barevná data mohou také být v podobě libovolného obrazu nebo fotografie.

MATLAB převede barevná data textury do vlastnosti `CData` objektu surface. Zatímco barva objektu surface je vždy určena hodnotami obsaženými v jeho vlastnosti `CData`, je transformace texture odlišná v tom, že rozměr pole `CData` může být u tohoto objektu surface jiný než rozměr jeho pole `ZData`. Tím je umožněna aplikace obrazu libovolné velikosti na jakoukoliv plochu. MATLAB interpoluje barevná data textury tak, aby pokryla úplnou plochu objektu surface.

Před nastavením libovolné velikosti matice `CData` je nutné nastavit vlastnost `FaceColor` objektu surface na hodnotu `texturemap`. To lze udělat dvěma způsoby:

- Funkcí `surface`, která umožňuje specifikovat objektové vlastnosti v době, kdy objekt vytváříme.
- Nastavením vlastnosti `FaceColor` již existujícího objektu surface a určením nových dat pro `CData` funkcí `set`.

Následující příklad ukazuje transformaci obrazu `mandrill` na válcovou plochu. MATLAB implicitně transformuje obraz na celou plochu, ale tento příklad doplňuje data tak, že mandrill je zobrazen pouze na polovině válce z důvodu lepšího využití plochy.

```
load mandrill
colormap(map)
[x, y, z] = cylinder;
Xhalf = [ones(480, 375)*max(max(X))/2, X, ones(480, 125)*max(max(X))/2];

surface(x, y, z, 'FaceColor', 'texturemap', 'EdgeColor', 'none',...
        'CData', flipud(Xhalf))
view(3)
```

Matice dat mandrilla má rozměr $480 * 500$. Pro zobrazení těchto dat na polovině válce je proto přidáno dalších 500 sloupců dat. Sloupce jsou přidány před a za existující data tak, aby byl původní obraz správně umístěn pro implicitní 3-D pohled.

Hodnoty dat použité pro doplnění dat mandrilla jsou nastaveny na polovinu maximální hodnoty dat mandrilla. V důsledku toho je pak barva poloviny válce, která neobsahuje obraz mandrilla, nastavena na barvu, která leží uprostřed mapy barev mandrilla (map).

Navíc kromě nastavené vlastnosti `FaceColor` na hodnotu `texturemap` je též nastavena vlastnost `EdgeColor` na hodnotu `none`. Pak nejsou zobrazeny žádné čáry sítě.

Protože data objektů image jsou zobrazována v měřítku `'ij'`, jsou data mandrillu ve vertikálním směru „obrácena“ pomocí `flipud`. (Více informací viz popis funkce `axis` ve druhém dílu, *Popis funkcí*).

Chceme-li získat tentýž obrázek pomocí grafických funkcí vyšší úrovně, musíme nejprve obdržet identifikátor objektu surface, a pak změnit důležité vlastnosti funkcí `set`:

```
[x, y, z] = cylinder ;
h = surf(x, y, z) ;
set(h, 'FaceColor', 'texturemap', 'EdgeColor', 'none', 'CData', flipud(Xhalf))
```

12.5.2.8 Poloautomatické meze os

Pokud chceme použít režim poloautomatických mezí os, definujeme jednu mez rozsahu souřadnic nebo barevné osy (vlastnost `XLim`, `YLim`, `ZLim` nebo `CLim`) a druhou mez nastavíme do automatického režimu zadáním hodnoty `+Inf` nebo `-Inf`.

Následující příkaz např. nastaví minimální mez x-ové osy na hodnotu 0, ale umožní automatické nastavení maximální meze:

```
set(gca, 'XLim', [0 inf])
```

Podobně nastavení horní meze x-ové osy na hodnotu 40 a ponechání automatického nastavení dolní meze se provede příkazem

```
set(gca, 'XLim', [-inf 40])
```

Bez ohledu na to, kterou mez určujeme, musí být minimum vždy menší než maximum.

12.5.2.9 Logaritmická stupnice

MATLAB vykresluje v logaritmické stupnici také záporná data. Nemůže ale zobrazit záporná a kladná data současně do jediných os. Obsahují-li data kladná i záporná čísla, jsou záporná čísla ignorována a dolní mez osy je nastavena automaticky tak, aby byla znázorněna nejmenší kladná hodnota dat. Zápornou logaritmickou osu vytváří MATLAB pouze tehdy, pokud jsou všechna kreslená data záporná.

Nastavení mezí jednotlivých os (vlastnost `XLim`, `YLim`, `ZLim` nebo `CLim`) na hodnotu `+Inf` nebo `-Inf`, jak je popsáno výše, odpovídá automatickému nastavení mezí. U logaritmické stupnice tomu může odpovídat také hodnota 0 pro mez osy. Minimální mez (je-li rozsah $[0 + n]$) nebo maximální mez (jsou-li meze $[-n0]$) bude změněna tak, aby vyhovovala hodnotě dat, která je nejblíže nuly.

Následující příkazy např. nastaví horní mez y-ové osy na 10 a umožní automatické nastavení dolní meze (kromě nulových dat, která jsou transformována na $-\infty$):

```
plot(rand(1:10))
set(gca, 'YLim', [0 0.1])
set(gca, 'YScale', 'log')
```

Nastavíme-li rozsah osy před vykreslením všech dat, snaží se MATLAB vybrat vhodný rozsah, a pak při vykreslování aktuálních dat nastavený rozsah revidovat. Pokud pro minimální mez osy určíme záporné číslo a pro odpovídající maximální mez osy číslo kladné (pro logaritmickou stupnici), zachází MATLAB s dolní mezí tak, jako by byla nulová a použije automatické nastavení této meze.

12.5.2.10 Funkce grafického okna závislé na akci tlačítka myši

Můžeme definovat funkce zpětného volání (callback functions), které se provedou na základě určitých akcí tlačítka myši uvnitř grafického okna. Tyto akce jsou: stisknutí tlačítka myši, pohyb myši při držení stisknutého tlačítka a uvolnění tlačítka myši. Funkce zpětného volání jsou definovány pro grafické okno vlastnostmi objektu figure `WindowButtonDownFcn`, `WindowButtonMotionFcn` a `WindowButtonUpFcn`.

Typický příklad může být takový, kdy `WindowButtonDownFcn` má definovanou `WindowButtonMotionFcn`. Potom `WindowButtonMotionFcn` (která může např. táhnout libovolný grafický objekt, na který ukazuje kurzor) je aktivní pouze po stisknutí tlačítka myši.

Funkce `WindowButtonUpFcn` může oddefinovat funkci `WindowButtonMotionFcn` jejím nastavením na prázdný řetězec `' '`, takže pohyb kurzoru již nemá více vliv (tj. netáhne již žádný grafický objekt). V tomto případě pokračuje funkce `WindowButtonMotionFcn` ve vykonávání svých příkazů do té doby, dokud není tlačítko myši uvolněno (dokonce i v případě, kdy je kurzor vně grafického okna).

Vlastnosti objektu figure `WindowButtonDownFcn`, `WindowButtonMotionFcn` a `WindowButtonUpFcn` jsou ovlivnitelné vlastností objektu figure `Interruptible`, která je též popsána v tomto dokumentu.

12.5.2.11 Funkce grafického objektu závislé na akci tlačítka myši

Můžeme definovat funkce zpětného volání, které se provedou v závislosti na určitých akcích tlačítka myši na základních grafických objektech. Všechny tyto objekty (kromě objektů uimenu a objektu root) podporují novou vlastnost `ButtonDownFcn`. Ta nám umožňuje definovat funkci zpětného volání, která se vykoná, stiskneme-li tlačítko myši v době, kdy je kurzor na příslušném objektu.

Funkci zpětného volání definujeme jako řetězec `s`, který je MATLABem vyhodnocen příkazem `eval(s)`, když je funkce vyvolána. Řetězcem může proto být libovolný platný výraz MATLABu nebo jméno m-souboru. Řetězec je vykonán v pracovním prostoru MATLABu.

Funkce zpětného volání `ButtonDownFcn` definovaná pro vlastnost objektu se liší od funkce zpětného volání, kterou definujeme pro vlastnost `CallBack` objektu uicontrol a vlastnost `WindowButtonDownFcn` objektu figure. Ve skutečnosti mohou všechny funkce zpětného volání pracovat současně, ale je důležité pochopit posloupnost, podle které jsou funkce zpětného volání vykonávány a kritéria platná pro jejich výběr.

12.5.2.12 Implicitní hodnoty vlastností

Všechny vlastnosti objektů mají své implicitní hodnoty vestavěné v MATLABu (factory settings). Navíc ale můžeme definovat své vlastní implicitní hodnoty v libovolném bodu hierarchie objektů.

Hledání implicitních hodnot začíná u aktuálního objektu a pokračuje přes předky do té doby, dokud není nalezena implicitní hodnota definovaná uživatelem nebo dokud není dosaženo vestavěných implicitních hodnot. Proto je hledání implicitních hodnot vždy úspěšné.

Implicitní hodnoty můžeme nastavit pomocí řetězce začínajícího slovem Default, za kterým následuje typ objektu a nakonec vlastnost objektu. Např. nastavení implicitní barvy čáry na bílou barvu v úrovni aktuálního objektu figure provede příkaz

```
set(gcf, 'DefaultLineColor', 'w')
```

Řetězec `DefaultLineColor` jednoznačně určuje vlastnost `Color` jako vlastnost objektu line, ne jako vlastnost `Color` objektu figure. Pokud chceme určit barvu pro objekt figure, musíme použít řetězec `DefaultFigureColor`. Implicitní barvu pro objekt figure můžeme samozřejmě specifikovat pouze na úrovni objektu root:

```
set(0, 'DefaultFigureColor', 'b')
```

Bod hierarchie, ve kterém definujeme implicitní hodnotu, určuje, které objekty tuto hodnotu použijí. Můžeme např. nastavit na úrovni objektu root implicitní bílou barvu pro objekt figure

```
set(0, 'DefaultFigureColor', 'w')
```

Potom všechny následně vytvořené objekty typu figure mají bílé pozadí.

Funkce `set` akceptuje též řetězce `default`, `factory` a `remove` jako hodnoty vlastností. Specifikujeme-li hodnotu `default`, nastaví se vlastnost na první nalezenou implicitní hodnotu pro tuto vlastnost.

Následující příkazy nastaví vlastnost `EdgeColor` (barva hran), která přísluší objektu surface, na zelenou:

```
h = surf(peaks)
set(0, 'DefaultSurfaceEdgeColor', 'g')
set(h, 'EdgeColor', 'default')
```

Pokud již existuje na úrovni axes nebo figure implicitní hodnota pro vlastnost surface `EdgeColor`, je nalezena dříve a použita místo implicitní hodnoty `EdgeColor`, která je definována na úrovni root.

Zadáme-li hodnotu `factory`, nastaví se vlastnost na svou hodnotu vestavěnou v MATLABu. Následující příkazy nastaví vlastnost `EdgeColor` plochy `h` na černou barvu (tj. nastavení MATLABu) bez ohledu na definovanou implicitní hodnotu.

```
h = surf(peaks)
set(0, 'DefaultSurfaceEdgeColor', 'g')
set(h, 'EdgeColor', 'factory')
```

Řetězcem `remove` můžeme odstranit implicitní hodnoty nastavené uživatelem. Příkaz

```
set(0, 'DefaultSurfaceEdgeColor', 'remove')
```

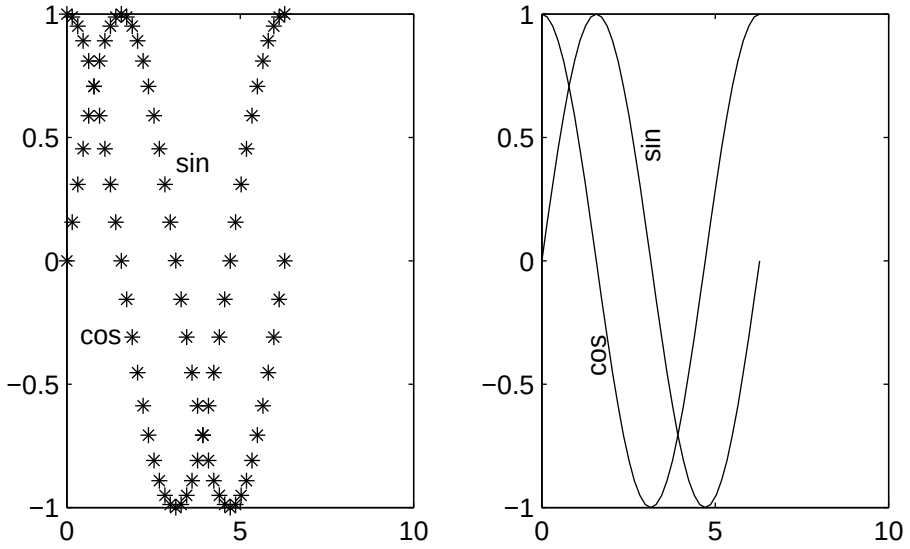
odstraní z úrovně root definici implicitně nastavené barvy hran.

Implicitní hodnoty jsou respektovány **pouze** grafickými funkcemi nejnižší úrovně (`figure`, `axes`, `line`, `text`, `surface`, `patch` a `image`) !!!

Následující příklad ukazuje, jak lze definováním implicitních hodnot řídit vlastnosti objektů. Tyto příkazy vytvoří dva objekty axes v jednom grafickém okně, nastaví na úrovni figure a úrovni axes implicitní hodnoty.

```
figure;
set(gcf, 'DefaultAxesBox', 'on');

subplot(1, 2, 1);
set(gca, 'DefaultLineLineStyle', '*');
line('XData', [0:pi/20:2*pi], 'YData', [sin(0:pi/20:2*pi)]);
hold on;
line('XData', [0:pi/20:2*pi], 'YData', [cos(0:pi/20:2*pi)]);
text('Position', [pi 0.4], 'String', 'sin');
text('Position', [pi/2 -0.3], 'String', 'cos', 'HorizontalAlignment', 'right');

subplot(1, 2, 2);
set(gca, 'DefaultTextRotation', 90);
```

```
line('XData', [0:pi/20:2*pi], 'YData', [sin(0:pi/20:2*pi)]);
hold on;
line('XData', [0:pi/20:2*pi], 'YData', [cos(0:pi/20:2*pi)]);
text('Position', [pi 0.4], 'String', 'sin');
text('Position', [pi/2 -0.3], 'String', 'cos', 'HorizontalAlignment', 'right');
```



Zatímco jsou v obou oblastech použity tytéž příkazy `line` a `text`, výsledek se řídí odlišným nastavením implicitních hodnot v každé oblasti.

Protože vlastnost objektu axes `Box` je nastavena v hierarchii na úrovni figure, vytvoří MATLAB oba objekty axes s rámečkem.

Protože levý objekt axes (oblast 1, 2, 1) definuje hvězdičkový implicitní styl čáry (`'*'`), je při každém volání funkce `line` použita hvězdičkovaná čára. Objekt axes napravo nedefinuje žádný implicitní styl čáry, a proto MATLAB použije plnou čáru (nastavení MATLABu). Naopak je zde implicitně definována vlastnost rotace textu o 90°, a proto jsou všechny texty otočeny o tuto hodnotu. Všechny další hodnoty vlastností získá MATLAB z vestavěných hodnot.

Poznámky:

MATLAB má vestavěn implicitní font písma Helvetica. Pokud tento font není na našem počítači, nebude otočení textu provedeno. V tomto případě je vhodné nastavit pro text na úrovni figure implicitní název fontu (z dostupných fontů) pomocí vlastnosti `DefaultTextFontName`.

Chceme-li mít v MATLABu definované určité hodnoty vždy, je vhodné je definovat v m-souboru `startup.m`.

12.5.3 Užitečné funkce

MATLAB obsahuje některé funkce, které zjednodušují proces získávání a nastavování hodnot vlastností v aktuálním objektu. Ve všech případech provádějí funkce `get` a `set` tutéž činnost, ale musíme

jim určit identifikátory cílových objektů. Následující seznam uvádí přehledně tyto funkce i vlastnosti, které jimi jsou ovlivňovány, popř. které jsou jimi získány.

Funkce	**Typ objektu**	**Nastavené nebo vrácené vlastnosti**
`axis`	axes	`XLim`, `YLim`, `ZLim`, `XLimMode`, `YLimMode`, `ZLimMode`, `View`, `YDir`, `Position`, `Visibility`, `AspectRatio`
`caxis`	axes	`CLim`, `CLimMode`
`cla`	axes	odstraní děti
`clf`	figure	odstraní děti
`colormap`	figure	`ColorMap`
`gca`	figure	`CurrentAxis`
`gcf`	root	`CurrentFigure`
`grid`	axes	`XGrid`, `YGrid`, `ZGrid`
`hold`	axes figure	`NextPlot` `NextPlot`
`orient`	figure	`PaperOrientation`, `PaperPosition`
`reset`	axes, figure	vše kromě `Position`
`subplot`	figure axes	`NextPlot`, `CurrentAxis` `Position`
`view`	axes	`View`, `XForm`

13 Odlaďování

Ačkoli je MATLAB jako programovací jazyk podstatně jednodušší než jiné programovací jazyky, což je způsobeno jeho syntaxí, přesto budete někdy nuceni lokalizovat a opravovat chyby ve vašich M-souborech.

Syntaktické chyby MATLAB nalezne během kompilace. Tyto chyby se obvykle snadno odstraňují. Horší je to v situaci, když MATLAB narazí na běhovou chybu, neboť při výskytu těchto chyb se provede návrat do příkazového okna s ohlášením chyby, což vede ke ztrátě lokálního pracovního prostoru funkce, v níž došlo k chybě. Pokud užíváte středník k potlačení zobrazování mezivýsledků, nebudete znát místo chyby.

Pro zobrazení mezivýsledků můžete použít tyto metody:

- Odstranit středníky.
- Přidat příkazy keyboard, které vám umožní prověřit pracovní prostor v místech aplikace těchto příkazů.
- Udělat z funkčního M-souboru skriptový M-soubor (z první řádky se udělá komentář), takže všechny mezivýsledky budou uloženy v hlavním pracovním prostoru.
- Použít příkazy MATLABu pro odladování.

První tři metody vyžadují zásahy do M-souborů. Poslední metoda je popsána v následující kapitole.

13.1 Odlaďovací příkazy

Odlaďovací příkazy jsou:

`dbstop`	Nastavení bodu přerušení.
`dbclear`	Odstranění bodu přerušení.
`dbcont`	Pokračování v programu.
`dbdown`	Změna kontextu lokálního pracovního prostoru.
`dbstack`	Výpis volání.
`dbstatus`	Výpis všech bodů přerušení.
`dbstep`	Spuštění jedné nebo několika řádek.
`dbtype`	Výpis M-souboru s čísly řádek.
`dbup`	Změna kontextu lokálního pracovního prostoru.
`dbquit`	Ukončení odlaďování.

13.2 Použití odladovacích nástrojů

Když narazíte v M-souboru na chybu, použijte odladovacích příkazů pro nastavení bodů přerušení, které vám pomohou nalézt chybu. Když se program zastaví v bodě přerušení, zobrazí se řádka, ve které došlo k přerušení, v příkazovém okně. Potom můžete zadat libovolný příkaz MATLABu.

Při práci s odladovacími příkazy MATLABu je třeba mít na paměti, že

- Odladovací příkazy pracují s funkčními a ne se skriptovými M-soubory.
- Informace o bodech přerušení jsou úzce svázány se zkompilovanými M-soubory. Jestliže je M-soubor smazán (úpravou nebo příkazem `clear`), jsou všechny body přerušení zrušeny. Tuto metodu můžete s výhodou použít , potřebujete-li vymazat řadu bodů přerušení z jednoho nebo více M-souborů.

13.3 Ukázka odladování

1. Ukázku začneme vytvořením dvou M-souborů: `test.m` a `test1.m`.
   `test.m`:

   ```
   function a = test(b)
   c=sqrt(b)*cos(b);
   a=test1(b,c);
   ```

   `test1.m`:

   ```
   function a = test1(b,c)
   q=cond(b);
   [w,e]=eig(c);
   a=w*q;
   ```

2. Použijte příkaz `dbtype` k očíslovanému výpisu těchto funkcí a k výpisu funkce `cond`.

   ```
   dbtype test
   1 function a = test(b)
   2 c=sqrt(b)*cos(b);
   3 a=test1(b,c);

   dbtype test1
   1 function a = test1(b,c)
   2 q=cond(b);
   3 [w,e]=eig(c);
   4 a=w*q;
   ```

```
dbtype cond
1 function y = cond(x)
2 % COND Matrix condition number.
3 % COND(X) is the ratio of the largest singular value of X
4 % to the smallest, which is the condition number of X in 2-norm
5
6 % See also RCOND and NORM
7
8 % J.N.Little 11-15-85
9 % Revised 3-9-87 JNL, 2-11-92 LS.
10 % Copyright (c) 1985-92 by the MathWorks, Inc.
11
12 if length(x) == 0 % Handle null matrix.
13 y = NaN;
14 return
15 end
16 if issparse(A)
17 error('Matrix must be non-sparse.')
18 end
19 s = svd(A);
20 if any(s == 0) % Handle singular matrix
21 disp('Condition is infinite')
22 y = Inf;
23 return
24 end
25 y = max(s)./min(s);
```

3. Vytvořte v pracovním prostoru proměnnou `hi`.

```
hi = 'hello again'
```

13.3.1 Nastavení bodů přerušení

Použijte příkaz `dbstop` k vytvoření bodu přerušení v M-souboru `test.m`.

```
dbstop in test
```

Tento příkaz přeruší program před první proveditelnou řádkou, což je stejné, jako když zadáte `dbstop at 2 in test`. Dále nastavte bod přerušení na 19. řádek v M-souboru `cond`:

```
dbstop at 19 in cond
```

13.3.2 Spuštění M-souboru a zobrazení volání

1. Po nastavení bodů přerušení spusťte funkci `test`. Podle nastavených bodů přerušení se program zastaví na druhé řádce:

```
test(magic(3))
2 c=sqrt(b)*cos(b);
```

2. Po zastavení funkce použijte příkaz `dbstack` pro zobrazení posloupnosti volání jednotlivých funkcí, která vede k bodu přerušení. Poněvadž se funkce právě rozběhla, zobrazí se pouze jedno funkční volání:

```
dbstack
In test.m at line 2
```

3. Pokračujte v provádění funkce:

```
dbcont
19 s = svd(x);
```

   Spuštění se zastaví na 19. řádce M-souboru `cond`.

4. Opět použijte příkaz `dbstack` k zobrazení posloupnosti volaných funkcí:

```
dbstack
In cond.m at line 19
In test1.m at line 2
In test.m at line 3
```

13.3.3 Kontrola lokálního pracovního prostoru a proměnných

1. Zkontrolujte proměnné v lokálním pracovním prostoru funkce `cond`:

```
who
Your variables are:
x y
```

2. Zkontrolujte obsah proměnné `x`:

```
x
x =
8 1 6
3 5 7
4 9 2
```

13.3.4 Spuštění následující řádky a kontrola proměnných

1. Použijte příkaz `dbstep` pro spuštění 20. řádky, což je další řádka, která se má provést:

```
dbstep
20 if any(s==0) % Handle singular matrix
```

2. Zkontrolujte obsah proměnné `s`:

```
s
s =
15.0000
6.9282
3.4641
```

3. Dále znovu zkontrolujte obsah lokálního pracovního prostoru funkce `cond`. Všimněte si, že nyní již zahrnuje proměnnou `s`:

```
who
Your variables are:
s x y
```

13.3.5 Změna pracovního prostoru a kontrola kontextu

1. Nyní se přepněte do lokálního pracovního prostoru funkce, která volala funkci `cond`:

```
dbup
In workspace belonging to test1.m.
```

2. Zkontrolujte lokální pracovní prostor funkce `test1`:

```
who
Your variables are:
a b c
```

3. Zkontrolujte obsah proměnné `b` ve funkci `test1`:

```
b
b =
8 1 6
3 5 7
4 9 2
```

4. Zkontrolujte obsah proměnné `c`:

```
c
c =
-3.0026 -0.4199 2.4503
-4.1951 -0.8405 2.2478
-4.1854 0.6431 3.5935
```

5. A teď zkontrolujte obsah proměnné `a`:

```
a
a =
[]
```

Poněvadž tato proměnná nemá přiřazenu žádnou hodnotu, zobrazí se prázdné hranaté závorky.

6. Přepněte se do lokálního pracovního prostoru funkce `test`; tj. funkce, která volala funkci `test1`:

```
dbup
In workspace belonging to test.m.
```

7. Přepněte se do základního pracovního prostoru (funkce `test` byla volána ze základního pracovního prostoru) a zkontrolujte obsah:

```
dbup
  In base workspace.
  who
  Your variables are:
  hi
```

13.3.6 Vytvoření nové proměnné

Nyní vytvoříme v základním pracovním prostoru novou proměnnou, přepneme se do pracovního prostoru funkce `test` a zkontrolujeme obsah.

1. Zadejte novou proměnnou:

```
new_var=123
new_var =
123
```

Po skončení běhu funkce `test` tuto proměnnou prověříme (viz kapitola *Zobrazení základního pracovního prostoru*).

2. Přepněte se do lokálního pracovního prostoru funkce `test`. Tentokráte použijte funkci `dbdown`.

```
dbdown
In workspace belonging to test.m.
```

3. Zkontrolujte obsah pracovního prostoru:

```
whos
Name Size Elements Bytes Density Complex
a 0 by 0 0  0 Full No
b 3 by 3 9 72 Full No
c 3 by 3 9 72 Full No
Grand total is 18 elements using 144 bytes
```

4. Zkontrolujte obsah proměnné `b`:

```
b
b =
8 1 6
3 5 7
4 9 2
```

5. Přepněte se do lokálního pracovního prostoru funkce `cond`:

```
dbdown
In workspace belonging to test1.m.
dbdown
In workspace belonging to cond.m.
```

13.3.7 Krokování funkce

1. Pokračujte v krokování (spouštění po řádcích) funkce `cond`. Po ukončení funkce `cond` se bude krokovat funkce `test1`:

```
dbstep
25 y = max(s)./min(s);
dbstep
End of M-file function cond.
dbstep
3 [w,e]=eig(c);
```

2. Zobrazte posloupnost volání funkcí:

```
dbstack
In test1.m at line 3
In test.m at line 3
```

3. Spusťte další řádek ve funkci `test1`:

```
dbstep
4 a=w*q;
```

4. Pokračujte ve výpočtu funkce, dokud se neobjeví nějaký bod přerušení nebo dokud se funkce nevrátí do základního pracovního prostoru:

```
dbcont
ans =
2.0428 -1.9138 - 0.9902i -1.9138 + 0.9902i
3.6832  0.5722 - 0.5802i  0.5722 + 0.5802i
1.0056 -2.9190 - 2.2187i -2.9190 + 2.2187i
```

13.3.8 Zobrazení základního pracovního prostoru

Když je výpočet proveden, zkontrolujte pracovní prostor.

```
whos

Name Size Elements Bytes Density Complex
ans     3 by 3 9 144 Full Yes
new_var 1 by 1 1   8 Full No
Grand total is 10 elements using 152 bytes
```

Všimněte si, že proměnná `new_var`, kterou jste vytvořili, je v základním pracovním prostoru:

```
new_var

new_var =
123
```

13.3.8.1 Ukončení odladďování

Odladďování můžete ukončit kdykoli, kdy si myslíte, že znáte příčinu problému. K ukončení slouží příkaz `dbquit`, který vás vrátí do základního pracovního prostoru.

Poznamenejme, že `dbquit` neruší body přerušení. Body přerušení ruší příkaz `dbclear`.

14 Stručný přehled funkcí

Základní kategorie funkcí MATLABu	
`color`	Funkce pro řízení barev a osvětlení
`datafun`	Funkce pro analýzu dat a Fourierovu transformaci
`demos`	Demonstrace a příklady
`elfun`	Elementární matematické funkce
`elmat`	Elementární matice a operace s maticemi
`funfun`	Funkce pro práci s funkcemi
`general`	Funkce k obecnému použití
`graphics`	Obecné grafické funkce
`iofun`	Nízkoúrovňové funkce pro práci se soubory
`lang`	Jazykové konstrukce a odlaďování
`matfun`	Maticové funkce – numerická lineární algebra
`ops`	Operátory a speciální znaky
`plotxy`	2-D grafika
`plotxyz`	3-D grafika
`polyfun`	Funkce pro práci s polynomy a interpolace
`sparfun`	Funkce pro práci s řídkými maticemi
`specfun`	Speciální matematické funkce
`specmat`	Speciální matice
`sounds`	Funkce pro práci se zvukem
`strfun`	Funkce pro práci se znakovými řetězci

14.1 Funkce k obecnému použití

Správa funkcí a příkazů	
`demo`	Spuštění demonstračního příkladu
`help`	Nápověda
`info`	Informace o MATLABu a jeho výrobci
`lookfor`	Hledání klíčových slov v nápovědě
`matlabroot` [4.1]	Adresář, ve kterém je nainstalován MATLAB
`path`	Správa vyhledávacích cest MATLABu
`type`	Výpis M-souboru
`version` [4.1]	Verze MATLABu
`what`	Seznam M-, MAT- a MEX-souborů v daném adresáři
`whatsnew` [4.1]	Zobrazení příslušného README-souboru
`which`	Nalezení adresáře dané funkce nebo souboru

Správa proměnných a pracovního prostoru	
`clear`	Vymazání proměnných a funkcí z paměti
`disp`	Zobrazení matice nebo textu
`length`	Délka vektoru
`load`	Načtení proměnných z disku
`pack`	Setřesení pracovního prostoru
`save`	Uložení proměnných na disk
`size`	Rozměry matice
`who`	Seznam aktuálních proměnných
`whos`	Seznam aktuálních proměnných s popisem

Správa souborů a operačního systému	
`cd`	Změna pracovního adresáře
`delete`	Vymazání souboru
`diary`	Ukládání obsahu příkazového okna do souboru
`dir`	Výpis obsahu zvoleného adresáře
`getenv`	Výpis systémové proměnné
`ls` [4.1]	Výpis obsahu aktuálního adresáře
`pwd` [4.1]	Název aktuálního adresáře
`unix`	Spuštění příkazu operačního systému
`!`	Spuštění příkazu operačního systému

Správa příkazového okna	
`clc`	Vyčištění příkazového okna
`cedit` [4.1]	Nastavení parametrů řízení editace příkazové řádky
`echo`	Řízení zobrazování příkazů ve skriptových souborech
`format`	Nastavení výstupního formátu
`home`	Nastavení kurzoru vlevo nahoru
`more`	Řízení stránkového výstupu v příkazovém okně

Spuštění a ukončení MATLABu	
`matlabrc`	M-soubor spouštěný vždy při startu MATLABu
`quit`	Ukončení MATLABu
`startup`	M-soubor spouštěný při startu MATLABu (existujeli)

14.2 Operátory a speciální znaky

Operátory a speciální znaky	
`+`	Plus
`-`	Minus
`*`	Násobení matic
`.*`	Násobení polí; tj. prvků matic
`^`	Umocnění matice
`.^`	Umocnění pole; tj. prvků matice
`kron`	Kroneckerův tenzorový součin
`\`	Zpětné lomítko – levostranné dělení
`/`	Lomítko – pravostranné dělení
`./`	Dělení polí; tj. prvků matic
`:`	Generování vektorů, indexování
`()`	Závorky výrazů, indexové závorky
`[ ]`	Maticové závorky
`.`	Desetinná tečka v číslech, příznak operace s poli
`..`	Nadřazený adresář
`...`	Pokračování příkazu na další řádce
`,`	Oddělovač indexů, argumentů funkcí, prvků matic
`;`	Konec příkazu bez výstupu, konec řádky v matici
`%`	Komentář do konce řádky
`!`	Uvádí běžný příkaz operačního systému
`'`	Hermitovská transpozice nebo apostrof u řetězců
`.'`	Transpozice
`=`	Přiřazení
`==`	Rovnost
`<>`	Relační operátory (menší, větší)
`&`	Logický součin (AND)
`|`	Logický součet (OR)
`~`	Negace (NOT)
`xor`	Neekvivalence (EXCLUSIVE OR)

Logické funkce	
`all`	Pravda, jsou-li všechny prvky vektoru pravdivé
`any`	Pravda, je-li alespoň jeden prvek vektoru pravdivý
`exist`	Kontrola existence proměnné nebo funkce
`find`	Nalezení indexů nenulových prvků
`finite`	Pravda, jde-li o prvky konečné velikosti
`isempty`	Pravda, jde-li o prázdnou matici
`isieee`	Pravda, jde-li o IEEE aritmetiku plovouci řádové čárky
`isinf`	Pravda, jde-li o prvky Inf
`isnan`	Pravda, jde-li o prvky NaN
`issparse`	Pravda, jde-li o řídkou matici
`isstr`	Pravda, jde-li o textový řetězec

14.3 Jazykové konstrukce a odlaďování

MATLAB jako programovací jazyk	
`eval`	Vyhodnocení řetězce obsahujícího výrazy MATLABu
`feval`	Vyhodnocení funkce specifikované jménem funkce
`function`	Přidání nových funkcí
`global`	Definování globálních proměnných
`lasterr` [4.2]	Vypsání poslední chyby
`nargchk` [4.1]	Ověřování počtu vstupních argumentů

Řídicí struktury	
`break`	Ukončení běhu cyklu
`else`	Používáno s if
`elseif`	Používáno s if
`end`	Ukončení rozsahu platnosti příkazů for, while a if
`error`	Zobrazení hlášení a ukončení funkce
`for`	Cyklus for
`if`	Podmíněné spuštění příkazů
`return`	Návrat do vyvolávající funkce
`while`	Cyklus while

Interaktivní vstup	
`input`	Výzva pro uživatelský vstup
`keyboard`	Předání řízení uživateli
`menu`	Vytvoření menu
`pause`	Čekání na odezvu uživatele

Odladťování	
`dbclear`	Odstranění bodu přerušení
`dbcont`	Pokračování v programu
`dbdown`	Změna kontextu lokálního pracovního prostoru
`dbquit`	Ukončení odladťovacího režimu
`dbstack`	Seznam volání funkcí
`dbstep`	Krokování
`dbstop`	Nastavení bodu přerušení
`dbtype`	Očíslovaný výpis M-souboru
`dbup`	Změna kontextu lokálního pracovního prostoru

14.4 Elementární matice a operace s maticemi

Elementární matice	
`eye`	Jednotková matice
`linspace`	Vytvoření lineárně ekvidistantního vektoru
`logspace`	Vytvoření logaritmicky ekvidistantního vektoru
`meshgrid`	Vytvoření X- a Y-ových polí pro 3-D grafiku
`ones`	Matice jedniček
`rand`	Matice náhodných čísel s rovnoměrným rozdělením
`randn`	Matice náhodných čísel s normálníl rozdělením
`zeros`	Nulová matice
`:`	Vektor všech indexů vektoru, ekvidistantní vektor

Speciální proměnné a konstanty	
`ans`	Nejaktuálnější výsledek
`computer`	Typ počítače
`eps`	Přesnost
`flops`	Čítač operací v pohyblivé řádové čárce
`i, j`	Imaginární jednotky
`inf`	Nekonečno
`NaN`	Nejde o číslo (Not-a-Number)
`nargin`	Počet vstupních argumentů funkce
`nargout`	Počet výstupních argumentů funkce
`pi`	3,1415926535897...
`realmax`	Největší číslo v pohyblivé řádové čárce
`realmin`	Nejmenší číslo v pohyblivé řádové čárce

Čas a datum	
`clock`	Datum a čas
`cputime`	Čas v sekundách od prvního spuštění tohoto příkazu
`date`	Datum
`etime`	Výpočet uplynulé doby mezi dvěma časy
`tic, toc`	Stopky

Operace s maticemi	
`diag`	Vytvoření nebo vyjmutí diagonál
`fliplr`	Horizontální překlopení
`flipud`	Vertikální překlopení
`reshape`	Změna velikosti
`rot90`	Rotace matice
`tril`	Vyjmutí dolní trojúhelníkové části
`triu`	Vyjmutí horní trojúhelníkové části
`:`	Indexace matice, přeskupení matice

Součiny	
`cross` [4.2]	Vektorový součin
`dot` [4.2]	Skalární součin

14.5 Speciální matice

Speciální matice	
`compan`	Sdružená matice
`gallery` [4.1]	Testovací matice
`hadamard`	Hadamardova matice
`hankel`	Hankelova matice
`hilb`	Hilbertova matice
`invhilb`	Inverzní Hilbertova matice
`magic`	Magická matice
`pascal` [4.1]	Pascalova matice
`rosser`	Testovací matice algoritmů na výpočet vlastních čísel
`toeplitz`	Töplitzova matice
`vander`	Vandermondeova matice
`wilkinson`	Wilkinsonova matice

14.6 Elementární funkce

Exponenciální funkce	
`exp`	Exponenciální funkce
`log`	Přirozený logaritmus
`log10`	Dekadický logaritmus
`sqrt`	Druhá odmocnina

Komplexní funkce	
`abs`	Absolutní hodnota
`angle`	Fázový úhel
`conj`	Komplexně sdružené číslo
`imag`	Imaginární část komplexního čísla
`real`	Reálná část komplexního čísla

Triginometrické funkce	
`acos`	Arkuskosinus
`acosh`	Argument hyperbolického kosinu
`acot` [4.1]	Arkuskotangens
`acoth` [4.1]	Argument hyperbolické kotangenty
`acsc` [4.1]	Arkuskosekans
`acsch` [4.1]	Argument hyperbolické kosekanty
`asec` [4.1]	Arkussekans
`asech` [4.1]	Argument hyperbolické sekanty
`asin`	Arkussinus
`asinh`	Argument hyperbolického sinu
`atan`	Arkustangens
`atan2`	Čtyř-kvadrantový arkustangens
`atanh`	Argument hyperbolické tangenty
`cos`	Kosinus
`cosh`	Hyperbolický kosinus
`cot` [4.1]	Kotangens
`coth` [4.1]	Hyperbolický kotangens
`csc` [4.1]	Kosekans
`csch` [4.1]	Hyperbolický kosekans
`sec` [4.1]	Sekans
`sech` [4.1]	Hyperbolický sekans
`sign`	Funkce signum
`sinh`	Hyperbolický sinus
`tan`	Tangens
`tanh`	Hyperbolický tangens

Numerické funkce	
`ceil`	Zaokrouhlení na celé číslo bližší k ∞
`fix`	Zaokrouhlení na celé číslo bližší k nule
`floor`	Zaokrouhlení na celé číslo bližší k -∞
`rem`	Zbytek po celočíselném dělení
`round`	Zaokrouhlení k nejbližšímu celému číslu
`sign`	Funkce signum

14.7 Speciální matematické funkce

Speciální matematické funkce	
`besselj` [4.1]	Besselova funkce prvního druhu
`bessely` [4.1]	Besselova funkce druhého druhu
`besseli` [4.1]	Modifikovaná Besselova funkce prvního druhu
`besselk` [4.1]	Modifikovaná Besselova funkce druhého druhu
`beta`	Funkce beta
`betainc`	Neúplná funkce beta
`betaln`	Přirozený logaritmus funkce beta
`ellipj`	Jacobiho eliptická funkce
`ellipke`	Eliptický integrál prvního a druhého druhu
`erf`	Chybová funkce
`erfc`	Doplňková chybová funkce
`erfcx`	Doplňková chybová funkce vynásobená exp(x^2)
`erfinv`	Inverzní chybová funkce
`expint` [4.1]	Exponentciální integrální funkce
`gamma`	Funkce gama
`gammainc`	Neúplná funkce gama
`gammaln`	Přirozený logaritmus funkce gama
`gcd` [4.1]	Největší společný dělitel
`lcm` [4.1]	Nejmenší společný násobek
`legendre` [4.2]	Legenderova funkce
`log2`	Dvojkový logaritmus
`pow2`	Exponenciální funkce se základem 2
`rat`	Racionální aproximace
`rats`	Racionální aproximace
`cart2sph` [4.1]	Převod kartézských souřadnic na sférické
`cart2pol` [4.1]	Převod kartézských souřadnic na polární
`pol2cart` [4.1]	Převod polárních souřadnic na kartézské
`sph2cart` [4.1]	Převod sférických souřadnic na kartézské

14.8 Maticové funkce – numerická lineární algebra

Analýza matic	
`cond`	Číslo podmíněnosti matice
`det`	Determinant
`norm`	Norma vektoru nebo matice
`null`	Nulový prostor matice
`orth`	Ortogonalizace
`rcond`	Odhad čísla podmíněnosti matice
`rank`	Hodnost matice
`rref`	Gauss-Jordanova eliminace
`trace`	Stopa matice

Lineární rovnice	
`chol`	Choleského algoritmus
`inv`	Inverze matice
`lscov`	Zobecněné řešení při znalosti kovarianční matice
`lu`	LU-rozklad
`nnls`	Nezáporné zobecněné řešení, metoda nejmenších čtverců
`pinv`	Pseudoinverze matice
`qr`	QR-transformace
`qrdelete` [4.1]	Smaž sloupec při QR-transformaci
`qrinsert` [4.1]	Vlož sloupec při QR-transformaci
`\ a /`	Řešení soustavy lineárních rovnic

Vlastní čísla a singulární hodnoty	
`balance`	Vyvažování matice
`cdf2rdf`	Převod komplexní diagonální formy na reálnou blokovou diagonální formu
`eig`	Vlastní čísla a vlastní vektory
`hess`	Hessenbergova forma
`poly`	Charakteristický polynom
`qz`	Zobecněná vlastní čísla
`rsf2csf`	Převod reálné blokové diagonální formy na komplexní diagonální formu
`schur`	Schurova forma
`svd`	Singulární rozklad matice

Maticové funkce	
`expm`	Maticová exponenciála
`expm1`	M-soubor pro výpočet maticové exponenciály
`expm2`	Maticová exponenciála pomocí Taylorových řad
`expm3`	Maticová exponenciála pomocí vlastních čísel a vektorů
`funm`	Vyhodnocení obecné maticové funkce
`logm`	Maticový logaritmus
`sqrtm`	Maticová druhá odmocnina

14.9 Analýza dat a Fourierova transformace

Základní operace	
`cumprod`	Kumulativní součin prvků
`cumsum`	Kumulativní součet prvků
`max`	Největší prvek
`mean`	Průměr
`median`	Medián
`min`	Nejmenší prvek
`prod`	Součin prvků
`sort`	Setřídění
`std`	Standardní odchylka
`sum`	Součet prvků
`trapz`	Numerická integrace lichoběžníkovou metodu

Konečné diference	
`del2`	5-ti bodový diskrétní Laplacián
`diff`	Diference
`gradient` [4.1]	Gradient
`subspace` [4.1]	Úhel mezi dvěma podprostory

Korelace	
`corrcoef`	Korelační koeficienty
`cov`	Kovarianční matice

Filtrace a konvoluce	
`conv`	Konvoluce a násobení polynomů
`conv2`	2-D konvoluce
`deconv`	Dekonvoluce a dělení polynomů
`filter`	1-D číslicový filtr
`filter2`	2-D číslicový filtr

Fourierova transformace	
`abs`	Modul
`angle`	Fázový úhel
`cplxpair`	Setřídění komplexních čísel
`fft`	Diskrétní Fourierova transformace
`fft2`	Diskrétní Fourierova transformace ve 2-D
`fftshift`	Přeorganizování výstupu z fft nebo fft2
`ifft`	Inverzní diskrétní Fourierova transformace
`ifft2`	Inverzní diskrétní Fourierova transformace ve 2-D
`nextpow2`	Nejbližší vyšší mocnina dvou
`unwrap`	Odstranění skoků ve fázovém úhlu

14.10 Funkce pro práci s polynomy a interpolace

Polynomy	
`conv`	Násobení polynomů
`deconv`	Dělení polynomů
`poly`	Vytvoření polynomu z jeho kořenů
`polyder` [4.1]	Derivace polynomu
`polyfit`	Proložení dat polynomem
`polyval`	Vyhodnocení polynomu
`polyvalm`	Vyhodnocení polynomu s maticovými argumenty
`residue`	Rozklad na parciální zlomky
`roots`	Nalezení kořenů polynomu

Interpolace dat	
`griddata`	Zachycení do ekvidistantní sítě
`interp1`	Interpolace v 1-D
`interp2`	Interpolace ve 2-D
`interpft`	Interpolace v 1-D pomocí FFT

14.11 Funkce pro práci s funkcemi

Funkce pro práci s funkcemi – nelineární numerické metody	
`fmin`	Nalezení minima funkce jedné proměnné
`fmins`	Nalezení minima funkce více proměnných
`fplot`	Vykreslení funkce
`fzero`	Nalezení nulového bodu funkce jedné proměnné
`ode23`	Řešení diferenciálních rovnic metodou nízkého řádu
`ode45`	Řešení diferenciálních rovnic metodou vysokého řádu
`quad`	Numerický výpočet integrálu metodou nízkého řádu
`quad8`	Numerický výpočet integrálu metodou vysokého řádu

14.12 Funkce pro práci s řídkými maticemi

Elementární řídké matice	
`spdiags`	Řídká matice vytvořená z diagonál
`speye`	Řídká jednotková matice
`sprand`	Řídká matice náhodných čísel
`sprandsym`	Řídká symetrická matice náhodných čísel

Konverze	
`find`	Nalezení indexů nenulových prvků
`full`	Konverze řídké matice na matici plnou
`sparse`	Vytvoření řídké matice z nenulových prvků a indexů
`spconvert`	Konverze z externího formátu řídké matice

Práce s nenulovými prvky řídkých matic	
`issparse`	Pravda, jde-li o řídkou matici
`nnz`	Počet nenulových prvků
`nonzeros`	Nenulové prvky
`nzmax`	Množství prostoru alokovaného pro nenulové prvky
`spalloc`	Alokace paměti pro nenulové prvky
`spfun`	Aplikování funkce na nenulové prvky
`spones`	Nahrazení nenulových prvků jedničkami

Vizualizace řídkých matic	
`gplot`	Vykreslení grafu používaného v teorii grafů
`spy`	Vizualizace struktury řídké matice

Třídící algoritmy	
`colmmd`	Slopcové přetřídění – minimální stupeň
`colperm`	Sloupcové přetřídění podle počtu nenulových prvků
`dmperm`	Dulmage-Mendelsohnova dekompozice
`randperm`	Náhodná permutace
`symmmd`	Přetřídění – minimální stupeň symetrie
`symrcm`	Reverzní Cuthill-McKeeho přetřídění

Podmíněnost, odhady	
`condest`	Odhad čísla podmíněnosti
`normest`	Odhad normy matice
`sprank`	Strukturální hodnost řídké matice

Pomocné funkce	
`spaugment`	Tvorba rozšířené matice k výpočtu zobecněného řešení
`spparms`	Nastavení parametrů pro výpočty s řídkými maticemi
`symbfact`	Symbolická analýza rozkladu matice

14.13 Dvojrozměrná grafika

Základní grafy	
`fill`	Vyplněný 2-D mnohoúhelník
`loglog`	Graf s logaritmickou stupnicí pro obě osy
`plot`	Graf s lineární stupnicí pro obě osy
`semilogx`	Graf s logaritmickou stupnicí pro x-ovou osu a lineární stupnicí pro y-ovou osu
`semilogy`	Graf s lineární stupnicí pro x-ovou osu a logaritmickou stupnicí pro y-ovou osu

Popis grafu	
`grid`	Čáry sítě
`gtext`	Text umístěný myší
`legend` [4.2]	Popis průběhů v grafu
`text`	Popis grafu
`title`	Nadpis grafu
`xlabel`	Popis x-ové osy
`ylabel`	Popis y-ové osy

Speciální grafy	
`bar`	Sloupcový graf
`comet` [4.1]	Pohybující se bod
`compass`	Graf tvaru růžice
`errorbar`	Chybový graf
`feather`	Graf tvaru ptačího pera
`fplot`	Graf funkce
`hist`	Histogram
`polar`	Graf v polárních souřadnicích
`rose`	Úhlový histogram
`stairs`	Schodový graf
`stem` [4.1]	Graf tvaru stonků

14.14 Trojrozměrná grafika

Příkazy pro kreslení čar a plných mnoúhelníků	
`comet3` [4.1]	Pohybující se bod
`fill3`	Vyplněný 3-D mnohoúhelník
`plot3`	Čáry a body ve 3-D

Vrstevnice a ostatní 2-D grafy pro 3-D data	
`clabel`	Popis výšky vrstevnic
`contour`	Vykreslení vrstevnic
`contour3`	Vykreslení vrstevnic ve 3-D
`contourc`	Výpočet vrstevnic (pro funkci contour)
`image`	Zobrazení objektu image
`pcolor`	Pseudobarevný graf
`quiver`	Graf tvaru jehelníčku

Grafy ploch	
`mesh`	Drátový model 3-D plochy
`meshc`	Kombinace funkcí mesh a contour
`meshz`	Drátový model 3-D plochy včetně nulové roviny
`slice` [4.1]	Objemová vizualizace pomocí řezů
`surf`	Stínovaná 3-D plocha
`surfc`	Kombinace funkcí surf a contour
`surfl`	Stínovaná 3-D plocha s osvětlením
`waterfall` [4.1]	Graf tvaru vodopádu

Vzhled grafu	
`axis`	Rozsah a vzhled os
`caxis`	Transformace barevné osy
`colormap`	Mapa barev
`hidden`	Režim zobrazování skrytých čar grafu typu mesh
`shading`	Nastavení vlastnosti pro barevné odstíny
`view`	Definování bodu pohledu na graf ve 3-D
`viewmtx`	Transformační matice pohledu

Popis grafu	
`grid`	Čáry sítě
`gtext`	Text umístěný myší
`text`	Popis grafu
`title`	Nadpis grafu
`xlabel`	Popis x-ové osy
`ylabel`	Popis y-ové osy
`zlabel`	Popis z-ové osy

3-D objekty	
`cylinder`	Generování válce
`sphere`	Generování koule

14.15 Obecné grafické funkce

Vytvoření a řízení grafických oken	
`clf`	Vyčištění aktuálního grafického okna
`close`	Zavření grafického okna
`gcf`	Získání identifikátoru aktuálního grafického okna
`figflag` [4.1]	Test existence okna daného jména
`figure`	Vytvoření grafického okna
`refresh` [4.2]	Zaktualizavání obsahu grafického okna

Vytvoření a řízení os	
`axes`	Vytvoření os
`axis`	Rozsah a vzhled os
`caxis`	Transformace barevné osy
`cla`	Vyčištění aktuálních os
`gca`	Získání identifikátoru aktuálních os
`hold`	Podržení aktuálního grafu
`subplot`	Vytvoření několika os vedle sebe

Základní grafické objekty	
`axes`	Vytvoření objektu axes
`figure`	Vytvoření objektu figure (grafické okno)
`image`	Vytvoření objektu image
`imagesc` [4.1]	Vytvoření objektu image s měřítkováním
`line`	Vytvoření objektu line
`patch`	Vytvoření objektu patch
`surface`	Vytvoření objektu surface
`text`	Vytvoření objektu text
`uicontrol`	Vytvoření objektu uicontrol (uživatelské rozhraní)
`uimenu`	Vytvoření objektu uimenu (uživatelské rozhraní)

Operace objektové grafiky	
`delete`	Vymazání objektu
`drawnow`	Zpracování nevyřízených žádostí o zobrazení
`findobj` [4.1]	Nalezení objektů, které mají dané vlastnosti
`gco` [4.1]	Získání identifikátoru aktuálního objektu
`get`	Získání vlastností objektu
`reset`	Nastavení implicitních hodnot pro objekty figure a axes
`set`	Nastavení vlastností objektu

Dialogové boxy	
`errordlg` [4.2]	Vytvoření dialogového boxu chybového hlášení
`helpdlg` [4.2]	Vytvoření dialogového boxu nápovědy
`questdlg` [4.2]	Vytvoření dialogového boxu dotazu.
`warndlg` [4.2]	Vytvoření dialogového boxu varování

Tisk	
`orient`	Orientace papíru při tisku
`print`	Tisk grafu nebo ukládání grafu do souboru
`printopt`	Konfigurace implicitní tiskárny

Animace	
`getframe`	Získání dat pro animaci
`movie`	Přehrávání animační matice
`moviein`	Inicializace animační matice

Různé	
`ginput`	Grafický vstup pomocí myši
`ishold` [4.1]	Test nastavení funkce hold
`newplot` [4.1]	Standardní úvodní grafický příkaz
`rotate` [4.2]	Rotace grafickým objektem
`uigetfile` [4.1]	Interaktivní volba souboru pro načtení
`uiputfile` [4.1]	Interaktivní volba souboru pro uložení
`uisetcolor`	Interaktivní volba barvy
`uisetfont`	Interaktivní volba fontu
`whitebg` [4.1]	Nastavení pozadí na bílou
`graymon` [4.1]	Nastavení pro monitory se stupni šedi
`zoom` [4.2]	Transfokace

14.16 Funkce pro řízení barev a osvětlení

Řízení barev	
`caxis`	Transformace barevné osy
`colormap`	Mapa barev
`shading`	Nastavení vlastnosti pro barevné odstíny

Mapy barev	
`bone`	Šedá mapa barev se zbarvením do modra
`cool`	Mapa barev s odstíny tyrkysové a fialové
`copper`	Mapa barev s lineárními tóny mědi
`flag`	Mapa barev tvořená červenou, bílou, modrou a černou
`gray`	Lineární šedá mapa barev
`hot`	Černo-červeno-žluto-bílá mapa barev
`hsv`	Mapa barev hsv (Hue-saturation-value)
`jet`	Varianta mapy barev hsv
`pink`	Mapa barev s pastelovými odstíny růžové
`prism` [4.1]	Mapa barev prism
`white` [4.1]	Bílá mapa barev

Funkce modifikující mapy barev	
`brighten`	Zesvětlení nebo ztmavení mapy barev
`colorbar` [4.2]	Zobrazení barevného měřítka
`contrast` [4.1]	Řízení kontrastu obrázku
`hsv2rgb`	Konverze HSV hodnot na RGB hodnoty
`rgb2hsv`	Konverze RGB hodnot na HSV hodnoty
`rgbplot`	Graf mapy barev
`spinmap`	Rotace mapy barev

Modely osvětlení	
`diffuse`	Odrazivost difúzní plochy
`specular`	Zrcadlový odraz
`surfl`	Stínovaná 3-D plocha s osvětlením
`surfnorm`	Výpočet a zobrazení normál plochy

14.17 Funkce pro práci se zvukem

Obecné funkce	
`saxis`	Měřítko zvukové osy
`sound`	Konverze vektoru na zvuk

Čtení a zápis zvukových souborů	
`auread` [4.1]	Čtení zvuku ze souboru ve formátu Sun
`auwrite` [4.1]	Zápis zvuku do souboru ve formát Sun
`lin2mu` [4.1]	Konverze
`mu2lin` [4.1]	Konverze
`wavread`	Čtení zvuku ze souboru ve formátu MS Windows
`wavwrite`	Zápis zvuku do souboru ve formátu MS Windows

14.18 Funkce pro práci se znakovými řetězci

Obecné funkce	
`abs`	Konverze řetězce na číslo
`eval`	Interpretace řetězce obsahující výrazy MATLABu
`setstr`	Nastavení příznaku řetězce pro zobrazení
`str2mat`	Transformace řetězce na číslo
`strings`	Přístup MATLABu k řetězcům

Porovnávání řetězců	
`blanks` [4.1]	Generování řetězce mezer
`deblank` [4.1]	Odstranění mezer z konce řetězce
`findstr` [4.1]	Hledá výskyt jednoho řetězce v druhém
`isletter` [4.1]	Test znaku z abecedy
`isspace` [4.2]	Test *bílých* mezer
`isstr`	Test řetězce
`lower`	Konverze na malá písmena
`strcmp`	Porovnání řetězců
`strrep` [4.2]	Nahrazení části řetězce jiným řetězcem
`strtok` [4.2]	Nalezení rámce (token) v řetězci
`upper`	Konverze na velká písmena

Konverze řetězec – číslo	
`int2str`	Konverze celého čísla na řetězec
`num2str`	Konverze čísla na řetězec
`sprintf`	Zápis formátovaných dat do řetězce
`sscanf`	Čtení formátovaných dat z řetězce
`str2num`	Konverze řetězce na číslo

Konverze čísel	
`dec2hex`	Konverze dekadického čísla na číslo hexadecimální
`hex2dec`	Konverze hexadecimálního čísla na číslo dekadické
`hex2num`	Konverze hexadecimálního čísla na číslo double

14.19 Nízkoúrovňové funkce pro práci se soubory

Otevření a zavření souboru	
`fclose`	Uzavření souboru
`fopen`	Otevření souboru

Neformátovaný vstup/výstup	
`fread`	Čtení binárních dat ze souboru
`fwrite`	Zápis binárních dat do souboru

Formátovaný vstup/výstup	
`fgetl` [4.1]	Čtení řádku ze souboru, ruší znak konce řádky
`fgets` [4.1]	Čtení řádku ze souboru, zachovává znak konce řádky
`fprintf`	Zápis formátovaných dat do souboru
`fscanf`	Čtení formátovaných dat ze souboru

Pozice v souboru	
`ferror`	Získání chyby souborového vstupu nebo výstupu
`feof` [4.1]	Test konce souboru
`frewind` [4.1]	Přetočení souboru na začátek
`fseek`	Nastavení pozice souboru
`ftell`	Získání pozice v souboru

Konverze řetězců	
`sprintf`	Zápis formátovaných dat do řetězce
`sscanf`	Čtení formátovaných dat z řetězce

Konverze řetězců	
`tempdir` [4.1]	Vrácení jména dočasného adresáře
`tempname` [4.1]	Generování jedinečného jména souboru

File Import/Export Routines.	
`csvread` [4.2]	Načtení souboru s čísly oddělenými čárkami do matice
`csvwrite` [4.2]	Zapsání matice do souboru s čísly oddělenými čárkami
`dlmread` [4.2]	Načtení souboru s čísly oddělenými ASCII znakem do matice
`dlmwrite` [4.2]	Zapsání matice do souboru s čísly oddělenými ASCII znakem
`wk1read` [4.2]	Načtení souboru v Lotus formátu WK1 do matice
`wk1write` [4.2]	Zapsání matice do souboru v Lotus WK1 formátu

Literatura

[1] *MATLAB User's Guide*. The MathWorks, Inc., 1993

[2] *MATLAB Reference Guide*. The MathWorks, Inc., 1993

[3] *MATLAB Release Notes*. The MathWorks, Inc., 1993

[4] *MATLAB New Features Guide*. The MathWorks, Inc., 1993

[5] *MATLAB Building a Graphical User Interface*. The MathWorks, Inc., 1993

[6] Žára, J. a kol.: *Počítačová grafika – principy a algoritmy*. GRADA, a.s., Praha, 1992

[7] Fořt, I.: *Windows 3.1 techniky programování*. GRADA, a.s., Praha, 1993

[8] *TURBO C++ referenční příručka*. Borland International, 1990