UČEBNÍ TEXTY OSTRAVSKÉ UNIVERZITY

Přírodovědecká fakulta

Základy modelování v MATLABU

Josef Tvrdík, Viktor Pavliska, Petr Bujok

Ostravská univerzita 2010

Základy modelování v MATLABU
KIP/UMATL

texty pro distanční studium

Autor: Josef Tvrdík, Viktor Pavliska, Petr Bujok

Ostravská univerzita v Ostravě, Přírodovědecká fakulta
Katedra Informatiky a počítačů

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

Obsah

1 Úvod

Tento text je určen studentům předmětu Základy modelování v Matlabu. Cílem předmětu je seznámit se se základními operacemi tak, aby je student mohl využívat pro zpracování dat, kreslení grafů a naučil se programovat algoritmy modelující procesy v přírodě i společnosti, viz např [3]. Text je rozdělen do kapitol, ve kterých student postupně získá praktické zkušenosti v užívání Matlabu.

Každá kapitola začíná pokyny pro její studium. Tato část je vždy označena jako **Průvodce studiem** s ikonou na okraji stránky.

Pojmy a důležité souvislosti k zapamatování jsou vyznačeny na okraji stránky textu ikonou.

V závěru každé kapitoly je rekapitulace nejdůležitějších pojmů. Tato rekapitulace je označena textem **Shrnutí** a ikonou na okraji.

Oddíl **Kontrolní otázky** označený ikonou by vám měl pomoci zjistit, zda jste prostudovanou kapitolu pochopili a snad vyprovokuje i vaše další otázky, na které budete hledat odpověď.

U některých kapitol je připomenuta **Korespondeční úloha**. Pro kombinované a distanční studium jsou korespondenční úlohy zadávány v rámci kurzu daného semestru. Úspěšné vyřešení korespondenčních úloh je součástí podmínek pro získání zápočtu v distančním a kombinovaném studiu.

Zdrojové texty algoritmů v Matlabu jsou tištěny strojopisným typem, který vypadá např. takto: `y = x + (b-a) .* rand(1,n)`.

2 Výpočetní prostředí Matlab

Průvodce studiem:

Cílem této kapitoly je seznámit se s Matlabem tak, abyste v něm mohli pohodlně pracovat. Počítejte asi se třemi až čyřmi hodinami studia, zejména praktickými cvičeními u počítače.

2.1 Co je Matlab?

Matlab je výpočetní prostředí pro maticové operace, technické výpočty a vizualizaci dat. Matlab už po řadu let vyvíjí firma MathWorks a má mnoho uživatelů v běžných operačních systémech (Windows, Unix) po celém světě. Je to jak interaktivní výpočetní systém, který interpretuje příkazy zadané v příkazovém okně, tak programovací jazyk vysoké úrovně, který umožňuje rychlou a jednoduchou implementaci algoritmů.

Základním datovým typem jazyka je pole, což může být skalár, vektor, matice nebo dokonce více než dvojrozměrné pole. Prvkem pole může být číselná hodnota v pohyblivé řádové čárce, dvojitá přesnost, rozsah od -10^{308} do 10^{308} (`realmax = 1.7977e+308`, `realmin = 2.2251e-308`), případně alfanumerický znak nebo řetězec znaků stejné délky. Logické hodnoty se vyjadřují číselně, false jako 0, true jako 1. Matlab kromě základních operací s takovým polem a spousty vestavěných dalších funkcí nabízí i programovací příkazy pro vytváření podprogramů (funkcí) a příkazy pro řízení průběhu výpočtu (podmíněný příkaz, switch, cykly) známé z běžných programovacích jazyků.

Výhodou Matlabu je i velmi kvalitní dokumentace, která je přístupná on-line prostřednictvím Helpu a dovoluje jak rychlé vyhledávání témat, zejména funkcí Matlabu, tak rychlou a názornou instruktáž k základním operacím. Pro seznámení s ovládáním Matlabu a jeho základními možnostmi je velmi užitečné spustit Help a projít Getting Started.

2.2 Aritmetické výrazy

Pro užívání Matlabu je důležité zvládnout základní operace s vektory a maticemi a také naučit se „myslet vektorově“, abychom byli připraveni efektivně využít možnosti, které máme v Matlabu k dispozici. Nejdříve si ukážeme pár jednoduchých příkladů.

Příkazy se zadávají na řádcích příkazového okna (command window), vždy na řádku za promt `>>`. Řádkový vektor **a** vytvoříme příkazem, ve kterém prvky vektoru od-

dělené mezerami nebo čárkou zadáme v hranatých závorkách. Po zadání příkazu se výsledek vyhodnocení zadaného výrazu vypíše také v příkazovém okně:

```
>> a = [3 2 5]
a =  3  2  5
```

Pokud bychom chtěli výpis výsledku na obrazovku potlačit, příkaz ukončíme středníkem:

```
>> a = [3 2 5];
```

Tím jsme vytvořili v pracovní oblasti (Workspace) objekt s názvem odpovídajícím užitému identifikátoru, t.j. `a`, který můžeme dále užívat v dalších příkazech. Matlab v identifikátorech rozlišuje malá a velká písmena, takže `a` a `A` nebo `TEMP_1`, `temp_1` a `Temp_1` jsou různé objekty.

Existující objekt můžete užít např. jako vstupní parametr funkce:

```
>> size(a)
ans =  1  3
```

Funkce `size` vrací rozměry matice, první údaj je počet řádků, druhý počet sloupců. Všimněme si, že pokud hodnotu výrazu nepřiřadíme do nějaké námi pojmenované proměnné, přiřadí se do proměnné `ans` (zkratka pro answer, tj odpověď nebo výsledek). Hodnota `ans` zůstává nezměněna do doby, než ji přepíšeme buď dalším zadáním výrazu bez přiřazení do námi pojmenované proměnné nebo námi explicitně zadaným přepisem proměnné `ans`, např.

```
ans(2) = 5;
```

Následujícím příkazem

```
>> b = a + 1
b =  4  3  6
```

jsme přičetli zadanou skalární hodnotu (t.j. 1) ke každému prvku vektoru.

Matici vytvoříme např. příkazem, ve kterém jsou řádkové vektory odděleny středníkem

```
>> B = [a; b; b-5.5]
B =
    3.0000    2.0000    5.0000
    4.0000    3.0000    6.0000
   -1.5000   -2.5000    0.5000
```

Funkce `length` vrací maximální hodnotu z vektoru spočítaného funkcí `size`:

```
>> length(B)
ans =  3
```

Prvky matice nebo vektoru jsou přístupné přes jejich indexy, takže např. prvek na druhém řádku a v prvním sloupci můžeme přepsat:

```
>> B(2, 1) = 0
B =
    3.0000    2.0000    5.0000
         0    3.0000    6.0000
   -1.5000   -2.5000    0.5000
```

S vektory a maticemi můžeme snadno provádět operace známé z lineární algebry (transpozice, inverze matice, pokud je možná, atd.) Jejich zápis je velmi podobný zápisu obvyklém v lineární algebře:

```
>> a = a'
a =
     3
     2
     5
```

```
>> det(B)  % determinant
ans =  54
```

```
>> C = B^-1 % inverse matice B, stejný výsledek uzitím inv(B)
C =
    0.3056   -0.2500   -0.0556
   -0.1667    0.1667   -0.3333
    0.0833    0.0833    0.1667
```

```
>> I = B * C
I =
    1.0000    0.0000         0
         0    1.0000         0
    0.0000   -0.0000    1.0000
```

2.3 Některé příkazové zkratky

Pro vytvoření vektorů a matic jsou užitečné další různé v Matlabu vestavěné funkce, které zkracují práci. Pokud tvoří prvky vektoru sekvenci, pak ji můžeme zadat zkrá-

ceně zadáním dolní a horní hranice (pak je implicitně krok roven jedné) nebo požadovaný krok explicitně zadat:

```
>> a = 1:5
a =  1     2     3     4     5
>> b = 10:-2:1
b =  10     8     6     4     2
```

Matice a vektory můžeme vytvářet pomocí funkcí:

`zeros(n, p)`	matice $(n \times p)$, samé nuly
`ones(n, p)`	matice $(n \times p)$, samé jedničky
`eye(n)`	jednotková matice $(n \times n)$
`rand(n, p)`	matice $(n \times p)$, náhodná čísla z rovnoměrného rozdělení na $[0, 1)$
`randn(n, p)`	matice $(n \times p)$, náhodná čísla z normovaného normálního rozdělení

Zadáme-li těmto funkcím jen jeden parametr, pak vytvořená matice je čtvercová, je-li hodnota tohoto jednoho parametru rovna 1, pak vytvoříme čtvercovou matici rozměru 1×1, čili skalár. Příklady:

```
>> zeros(1, 3)
ans =     0     0     0
```

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
>> rand(3)
ans =
    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169
```

```
>> 10 + 2*randn(3)
ans =
```

```
    7.1181   11.3800   12.5805
   11.1423   11.6312   11.3372
    9.2002   11.4238   12.3817
```

Pro práci s vektory a maticemi je důležité prázdné pole `[]`, které má rozměr (0×0). Zařazením prázdného prvku (matice) vypouštíme odpovídající prvky příslušného pole:

```
a =    1     2     3     4     5
>> a(2) = []
a =    1     3     4     5
```

Řádek nebo sloupec matice můžeme přepsat vektorem

```
>> X = ones(3, 4)
X =
     1     1     1     1
     1     1     1     1
     1     1     1     1
>> X(1, :) = a
X =
     1     3     4     5
     1     1     1     1
     1     1     1     1
```

nebo prázdným polem, tj. vypustit celý třetí sloupec. Symbol `:` v tomto příkladu znamená, že operace se týká všech řádků matice `X`.

```
>> X(:, 3) = []
X =
     1     3     5
     1     1     1
     1     1     1
```

Pole můžeme zvětšovat i dynamicky tj. přidávat nové prvky v právě prováděném příkazu. Pokud by v novém poli nebyla nějaká hodnota dosud přiřazena, automaticky se dodají nuly, jak uvidíme v následujícím příkladu:

```
a =     1     2     3
>> a(5) = -1
a =     1     2     3     0    -1
```

Zvláštní význam má proměnná `end` obsahující aktuální nejvyšší hodnotu indexu. Pak např. přepsání posledního prvku vektoru **a** hodnotou předposledního prvku zapíšeme takto:

```
>> a(end) = a(end-1)
a =     1     2     3     0     0
```

Kromě dosud zmíněných aritmetických operátorů jsou užitečné operace na odpovídajících prvcích dvou polí stejných rozměrů, tj. operátory začínající tečkou, `.*, ./, .^`.

Např. umocnit každý prvek vektoru **a** odpovídajícím prvkem téhož vektoru zapíšeme takto a dostaneme výsledek:

```
>> a.^a
ans =     1     4    27     1     1
```

Příklad 2.1 Matici o rozměru 2×5, jejímiž prvky jsou nezávislé náhodné hodnoty ze spojitého rovnoměrného rozdělení $U \sim (1, 11)$ vygenerujeme následujícím příkazem

```
>> R = 1 + 10*rand(2, 5)
R =
    8.0605    3.7692    1.9713    7.9483   10.5022
    1.3183    1.4617    9.2346    4.1710    1.3445
```

Matici náhodných hodnot z diskrétního rovnoměrného rozdělení s parametry $(1, 10)$ (náhodná celá čísla mezi 1 a 10) získáme tak, že místa za desetinnou tečkou odřízneme pomocí funkce `fix`:

```
>> F = fix(R)
F =
     8     3     1     7    10
     1     1     9     4     1
```

Jak už bylo v jednom z předchozích příkazů ukázáno, k celému řádku nebo sloupci matice se dostaneme využitím symbolu `:` na příslušném místě indexu:

```
>> f2s = F(2, :) %  druhy radek matice F
f2s =
     1     1     9     4     1
```

```
>> f2r = F(:, 2)  %  druhy sloupec matice F
f2r =
     3
     1
```

Pokud výsledek nějaké aritmetické operace „přeteče“ rozsah datového typu, je výsledkem hodnota `Inf`, např.

```
>> realmax * 1.2
ans =   Inf
>> -realmax * 1.2
ans =  -Inf
```

Pokud výsledek aritmetické operace nelze vyhodnotit (výrazy `Inf*0` nebo `Inf/Inf` ap.), má hodnotu `NaN` (not a number), např.

```
>> -realmax * 1.2 / (realmax * 1.2)
ans =   NaN
```

Pro pohodlnější zadávání příkazů z klávesnice můžeme využít svislých šipek na klávesnici, které nám umožňují listovat ve dříve zadaných řádcích a pak je na aktuálním příkazovém řádku editovat. Podobně je možné využívat i dříve zadané příkazy, které jsou zobrazeny na pracovní ploše v okně Command History.

Při interaktivní práci s Matlabem je někdy užitečné si uložit aktuální stav pracovního prostoru (workspace), tj. všechny nebo námi vybrané dosud inicializované proměnné (jejich seznam, typy a rozměry se objevují v okně Workspace), abychom v příštím sezení mohli pokračovat od současného stavu. To je možné volbou `save Workspace as` z menu `File`. Soubor se pak uloží pod zadaným jménem s příponou `.MAT`. Tento soubor pak otevřeme z menu `File` ve volbě `Open` nebo příkazem `load`.

Shrnutí:

- Datové typy v Matlabu
- Inicializace vektorů a matic
- Funkce `zeros, ones, rand, randn`
- Prázdné pole `[]`, přístup k celým sloupcům a řádkům matice

Kontrolní otázky:

1. Jak vygenerujete náhodné číslo z rovnoměrného spojitého rozdělení na intervalu $[-1, 1]$?
2. Jak vygenerujete náhodné číslo z normálního rozdělení s parametry $\mu = 100$ a $\sigma^2 = 225$?
3. Je nějaký rozdíl mezi vektory získanými následujícími třemi příkazy:
   `a = 1:5`,
   `b = 1 + fix(5*rand(1, 5))`,
   `c = randperm(5)` ?

3 Matematické funkce, 2D grafy

Průvodce studiem:

Cílem této kapitoly je seznámit čtenáře se základními matematickými funkcemi a s 2D grafikou v Matlabu. V první části kapitoly se naučíme, jak na vektory efektivně aplikovat základní matematické funkce. Druhá část kapitoly je věnována základům jednoduché vizualizace v Matlabu, vytváření 2D grafů. Na prostudování této kapitoly, pochopení a procvičení příkladů si vyhraďte zhruba 4 hodiny.

3.1 Některé funkce

V prostředí Matlab je možné pro snazší výpočty s vektory použít vestavěné `matematické funkce`. Matlab nabízí většinu známých a běžně používaných matematických funkcí. V téhle kapitole si objasníme způsob práce s těmito funkcemi na řešených příkladech a ukážeme si, kde nalézt celkový výčet elementárních i speciálních matematických funkcí Matlab.

Mezi nejpoužívanější funkce v Matlabu patří `min`, `max`, `sum`, `prod`, `sort` a `sortrows`. Nyní si na jednoduchém příkladě ukážeme jak tyto funkce použít.

Příklad 3.1 Nejprve vytvoříme řadu čísel 1 – 15, těmi naplníme matici o rozměru 3×5 a tu pak transformujeme na potřebný tvar funkcí `reshape`.

```
>> x = 1:1:15;
>> A = reshape(x, 3, 5)
A =
     1     4     7    10    13
     2     5     8    11    14
     3     6     9    12    15
% Funkce reshape provede transformaci z vektoru
% na matici a naopak, a zmenu rozmeru matic
>> B = A'
B =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15
```

Pro sečtení hodnot prvků v jednotlivých sloupcích matice použijeme funkci `sum`. Pro stejnou operaci na řádcích matice zadáme k této funkci parametr, jak ukazuje následující příkazy.

```
>> sum(B)
ans =
    35    40    45
% Stejny vypocet provede prikaz sum(B, 1)
% Provede soucet prvku v jednotlivych sloupcich B
>> sum(B, 2)
ans =
     6
    15
    24

% Prikaz provedl soucet prvku v jednotlivych radcich A
```

Stejně jako součet určitých prvků matice lze vypočíst i součiny hodnot řádků (nebo sloupců) matice pomocí funkce `prod`.

```
>> prod(A)
ans =
    6    120    504    1320    2730
% Soucin prvku v jednotlivych sloupcich A
```

Pro vypočtení nejmenší (nebo největší) hodnoty vektoru slouží funkce `min` (`max`). Výsledkem takové funkce v matici dat není jedna hodnota, ale řádkový vektor hodnot obsahující minimální (maximální) hodnoty sloupců, případně i vektor s pozicemi těchto hodnot, jak ukazuje příklad.

```
>> [Amin, Imin] = min(A)
Amin =
     1    4    7    10    13
Imin =
     1    1    1    1    1
>> [Amax, Imax] = max(A)
Amax =
     3    6    9    12    15
Imax =
     3    3    3    3    3
% Funkce min a max vyhledaji v kazdem sloupci matice
```

```
% nejmensi (nejvetsi) hodnotu a spolu s ni umoznuji
% nalezt take index jeji pozice (Imin, Imax)

>> [Amin, Imin] = min(min(A))
Amin =
     1
Imin =
     1
>> [Amax, Imax] = max(max(A))
Amax =
    15
Imax =
     5
% Opakovane pouziti funkce min (nebo max) umoznuje
% nalezt minimum (maximum) v cele matici cisel
```

Funkce `sort` setřídí prvky vektoru, jak ukazuje příklad, kde je vytvořen řádkový vektor pěti prvků z normálního rozdělení N(0,100)a následně setříděn.

```
>> a = 10*randn(1, 5)
a =
   -1.8671    7.2579   -5.8832   21.8319   -1.3640
>> sort(a)
ans =
   -5.8832   -1.8671   -1.3640    7.2579   21.8319
```

Funkce `sort` setřídí prvky matice v jednotlivých řádcích (sloupcích) matice, jak ukazuje příklad, kde je setříděna matice `A` po sloupcích

```
A =
  2     6     1
  8     5     2
>> sort(A, 1)
ans =
  2     5     1
  8     6     2
% Setrideni prvku ve sloupcich matice
```

a následně po řádcích.

```
A =
  2     6     1
  8     5     2
>> sort(A, 2)
ans =
  1     2     6
  2     5     8
% Setrideni prvku na radcich matice
```

Pro třídění matice funkcí `sort` je možné pomoci parametru zobrazit matici původních indexů všech prvků matice.

```
>> [A, i] = sort(A, 1)
A =
  2     5     1
  8     6     2
i =
  1     2     1
  2     1     2
% K matici byla vytvorena navic i matice
% puvodnich indexu prvku pred setridenim
% Obdobne pro radky matice.
```

Oproti tomu funkce `sortrows` neprohází hodnoty ve sloupcích matice nezávisle na ostatních sloupcích, ale třídí řádky matice na základě hodnot v jednom sloupci. Třídí tedy pouze řádky, jak ukazuje příklad.

```
A =
  9     6     5
  8     5     2
>> sortrows(A, 1)
ans =
  8     5     2
  9     6     5
% Setridi vsechny radky matice podle hodnot
% sloupce, jehoz cislo je uvedeno jako druhy
% vstupni parametr funkce, v tomto pripade "1"
```

Výčet elementárních matematických funkcí v Matlabu vyvolá příkaz:

```
>> help elfun
```

Stejným způsobem lze pak z helpu vyvolat výčet *speciálních funkcí* klíčovým slovem `specfun` či *funkcí pro výpočty s maticemi* klíčovým slovem `matfun`.

Elementární matematické funkce Matlabu jsou rozděleny do čtyř kategorií:

`goniometrické`

`exponenciální` – pro výpočet mocnin a logaritmů

`komplexní` – kterými se hlouběji zabývat nebudeme

`zaokrouhlovací` – pro převod čísel na vyšší řády a celočíselné dělení

Argumentem každé funkce v Matlabu může být bud'to skalár, vektor nebo matice.

Příklad 3.2 Nejprve vytvoříme pětisetprvkový vektor `x`, který je následně argumentem funkce sinus.

```
>> x = 0:pi/499:2*pi;
>> y = sin(x);
```

Pokud bychom za příkazem pro výpočet funkce *sinus* nezadali ukončovací znak (`;`), proběhl by výpis všech vypočtených hodnot sinu na obrazovku.

Z výčtu exponenciálních funkcí si v následujícím příkladu ukážeme výpočet hodnoty exponenciální funkce, mocninné funkce a logaritmu.

Příklad 3.3 Je vytvořena matice `A`, jejíž prvky jsou argumentem exponenciální a mocninné funkce.

```
>> A = [1 2; 3 4]
A =
     1     2
     3     4
>> B = exp(A) % Vypocte exponencialni funkce prvku matice A
B =
    2.7183    7.3891
   20.0855   54.5982
>> C = sqrt(A) % Vypocte odmocninu prvku matice A
C =
    1.0000    1.4142
    1.7321    2.0000
```

Následující příkaz výpočte přirozený logaritmus z prvků původní matice `A` násobených číslem π.

```
>> D = log(pi * A)
D =
    1.1447    1.8379
    2.2433    2.5310
```

Obdobně proveďte výpočet pro binární a dekadický logaritmus matice `A` z předchozího příkladu.

Příklad 3.4 Vytvoříme matici rozměru (3×5), jejíž prvky jsou pseudonáhodná čísla z normálního, respektive z rovnoměrného rozdělení.

```
>> A = 10*randn(3, 5); % Hodnoty z normalniho rozdeleni N(0, 100)

>> B = 10*rand(3, 5); % Hodnoty z rovnomerneho rozdeleni, tedy <0;10>

>> C = abs(A); % Absolutní hodnoty prku matice A
```

Srovnejte hodnoty prvků výsledných matic `B` a `C` z předchozího příkladu.

Příklad 3.5 Následující příkazy vytvoří čtvercovou matici 5. řádu pseudonáhodných hodnot z normálního rozdělení a tyto hodnoty zaokrouhlí na nejbližší celá čísla.

```
>> A = 100 * randn(5);
>> B = round(A);
```

Pomoci nápovědy zjistěte další možnosti zaokrouhlování reálných čísel. Zaokrouhlete prvky matice `A` více způsoby a výsledky porovnejte.

Matlab obsahuje také množství základních statistických funkcí jsou např.: *průměr*, *medián*, *modus*, *rozptyl* či *směrodatná odchylka*.

Příklad 3.6 V následujícím příkladu vytvoříme matici `A` a vypočteme vektory obsahující průměry (m) a směrodatné odchylky (s) jednotlivých sloupců.

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

```
    -1    -6     5     0
>> m = mean(A)
m =
     2     1     5     4
>> s = std(A)
s =
    3.0000    6.2450    4.0000    4.0000
```

Kdybychom chtěli provést stejný výpočet pro řádky matice, matici bychom před aplikováním funkce museli transponovat příkazem `A'`.

Příklad 3.7 Příklad ukazuje, jak nalézt největší hodnotu matice `A` z předchozího příkladu.

```
>> max_A = max(max(A))
max_A =
     9
```

3.2 Nakreslení 2D grafu

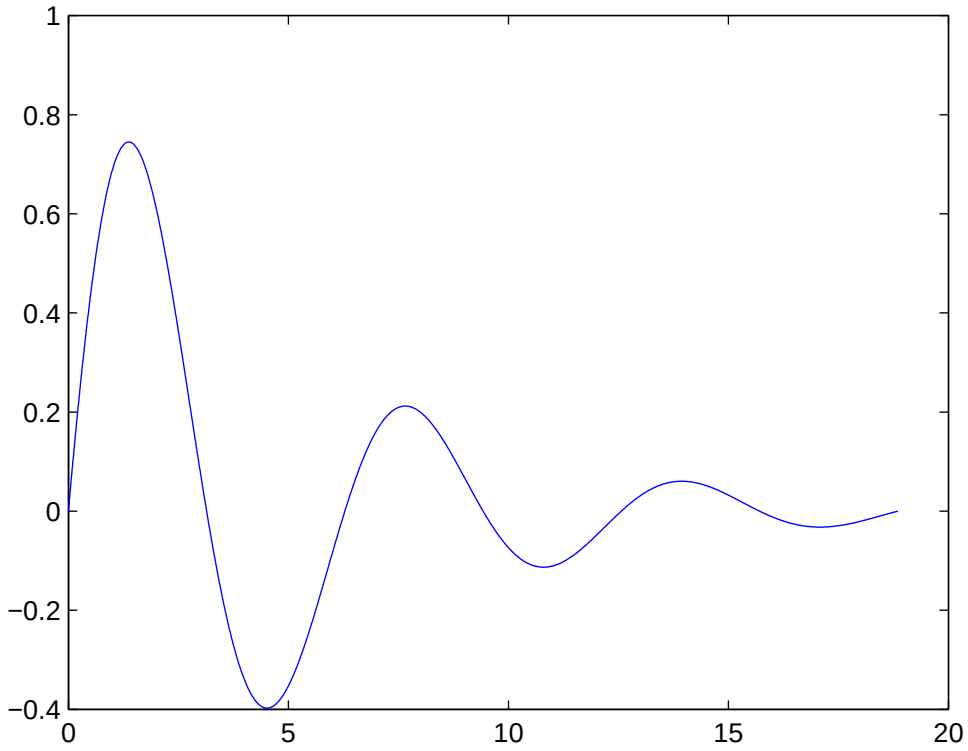
Silnou stránkou Matlabu je vizualizace dat. V této části se budeme zabývat dvourozměrnou vizualizací, neboli vytvářením 2D grafů.

K vytvoření 2D grafu jsou nezbytná data (naměřená, vypočtená), nejčastěji vektor hodnot argumentu a k němu odpovídající vektor funkčních hodnot. Podstatné je zvolit k datům vhodný typ grafu. V Matlabu se nejčastěji používá k vykreslení 2D grafu funkce `plot`.

Příklad 3.8 Vytvoříme vektor x ekvidistantně rozdělených hodnot v intervalu $< 0, 6\pi >$. K tomuto vektoru vytvoříme vektory stejné délky hodnot zadáním funkce `y`. Z těchto dvou vektorů je pak sestrojen graf.

```
>> x = 0:pi/400:6*pi;
>> y = exp(-0.2 * x) .* sin(x);
>> plot(x, y)
```

Zadáním těchto tří příkazů dostaneme v okně Figure(1) následující graf.



Podrobnější popis příkazu plot, jeho dalších nepovinných parametrů, možností interaktivní práce s grafem, export obrázku do různých formátů atd. nalezneme v helpu. Další možnosti 2D vizualizace jsou pod klíčovým slovem `graph2d`.

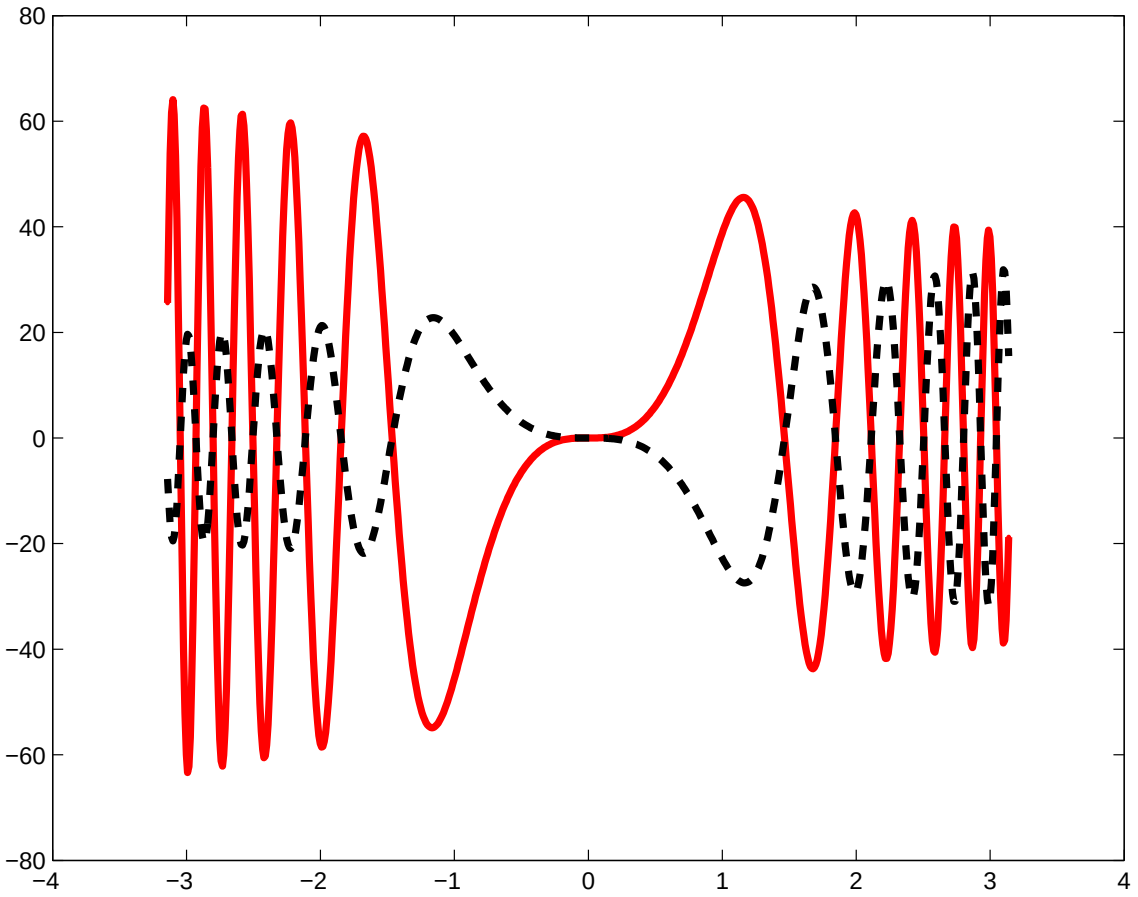
V praxi často nastává situace, kdy potřebujeme několik výstupů vizuálně srovnat (např. srovnat dvě křivky vývoje produktivity dvou firem, ap.). V Matlabu jsou k dispozici dvě možnosti, jak takovéto srovnání povést. V první řadě máme možnost vykreslit oba grafy do téhož grafického okna Figure (grafy se mohou protínat, prolínat, doporučuje se pro menší počet grafů). Dále je možné celé grafické okno Figure rozdělit na několik samostatných `podoken`, a do každého pak vykreslíme jeden z výstupů.

Příklad 3.9 Vytvoříme vektor ekvidistantně rozložených hodnot v intervalu $< -\pi, \pi >$. K tomuto vektoru pak vytvoříme vektory `y1` a `y2` zadanými funkcemi. Pak sestrojíme graf křivek obou funkcí do jednoho grafického okna.

```
>> x = -pi:0.01:pi;
>> y1 = 50 * exp(-0.08 * x) .* sin(x.^3);
>> y2 = 25 * exp(0.08 * x) .* (-sin(x.^3));
>> plot(x, y1, 'r.-', x, y2, 'k--', 'lineWidth', 3)
```

Způsob zadání současného vykreslení dvou grafů do jednoho okna je velmi jednoduchý a intuitivní. Za každou definicí dvojice veličin `x` a `y` následuje v apostrofech kombinace tří vlastností, v pořadí *barva křivky*, *značka bodu funkce* a *styl čáry*. Parametrem `'lineWidth'` jsme zvětšili tloušťku křivek.

V následujícím obrázku je výsledek:

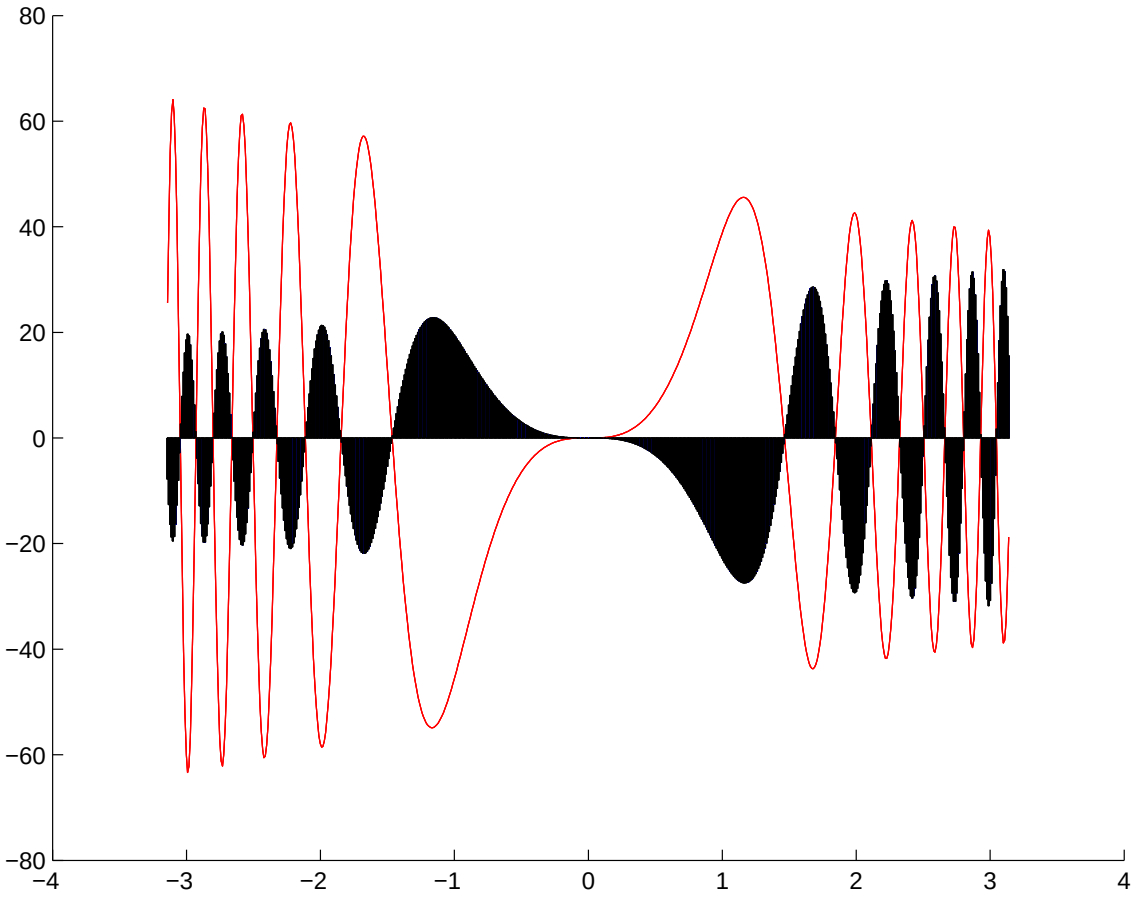


Další způsob zobrazení více grafů v jediném obrázku je pomocí příkazu `hold`. Aktivace příkazu `hold` zabrání přepisování grafického okna novým grafem, další grafy jsou do okna přikreslovány až do deaktivace tohoto příkazu. Oproti předchozímu přístupu, příkaz hold umožňuje současné vykreslení různých typů grafů v jednom grafickém okně.

Příklad 3.10 Příklad ukazuje jak vytvořit pro data z minulého příkladu každý graf jiného typu a vložit jej do jednoho grafického okna na sebe. Přičemž první z grafů bude vykreslen jako spojnicový, a druhý graf jako plošný.

```
>> hold on;
% Dale zadane grafy kresli do aktualniho okna
>> plot(x, y1, 'r');
```

```
>> bar(x, y2);
>> hold off;
```



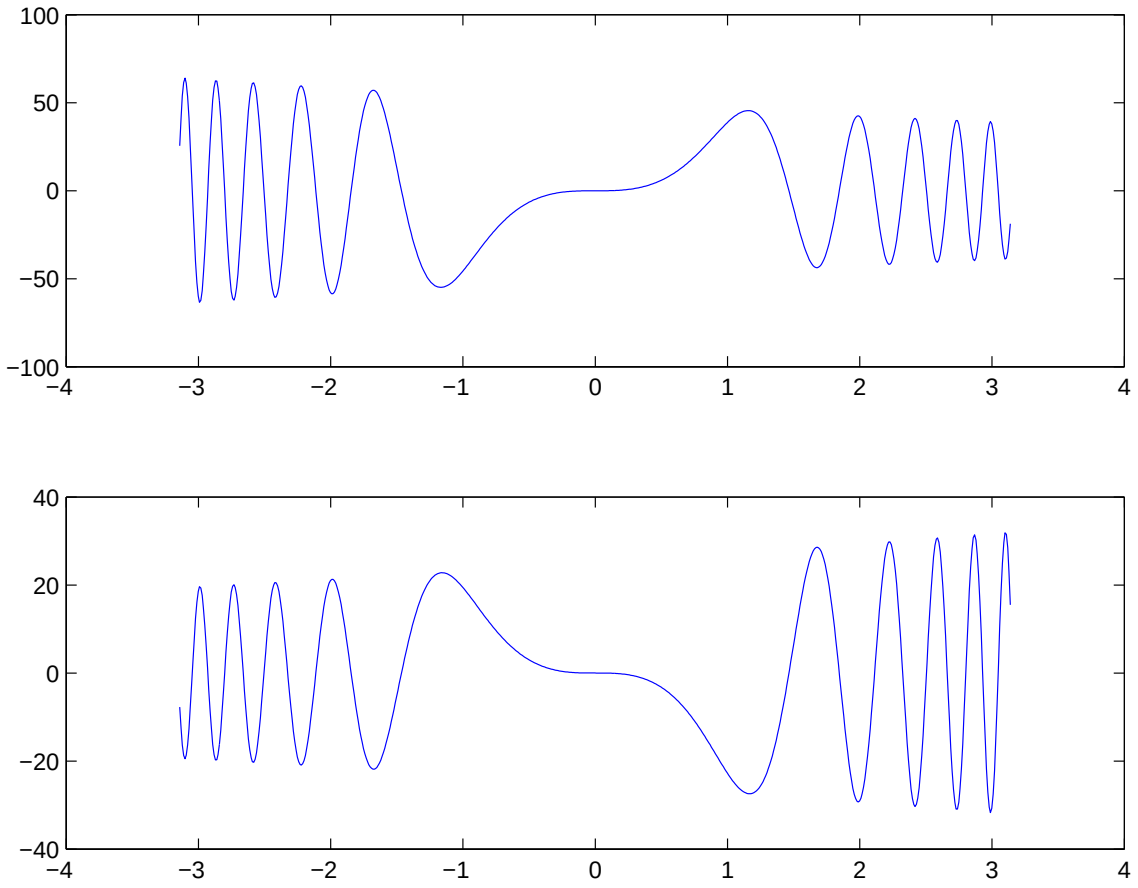
Aktivace příkazu `hold` zabrání přepisování grafického okna novým grafem, další grafy jsou do okna přikreslovány až do deaktivace tohoto příkazu.

Nyní si ukážeme i druhý způsob vykreslování více grafů do grafického okna rozděleného na podokna. K tomu slouží příkaz `subplot`, který umožňuje rozdělit grafické okno do několika podoken. Příkaz `subplot` má tři parametry, kde první číslo je počet dělení celého grafického okna *horizontálně*, druhé číslo je počet dělení okna *vertikálně* a třetí je *číslo podokna*, do kterého má být následující graf vykreslen (pozn. číslování podoken začíná z levého horního rohu grafického okna postupně doprava a dál dolů). Okno lze rozdělit i na různě velké části.

Příklad 3.11 Vytvoříme vektor ekvidistantně rozložených hodnot `x` v intervalu $< -\pi, \pi >$ a k němu dva vektory stejné délky `y1` a `y2`. Dále je horizontálně rozděleno grafické okno a do každé poloviny je nakreslen jeden z grafů.

```
>> x = -pi:0.01:pi;
>> y1 = 50 * exp(-0.08 * x) .* sin(x.^3);
>> y2 = 25 * exp(0.08 * x) .* (-sin(x.^3));
>> subplot(2, 1, 1);
% Tento prikaz vykresli nasledujici graf do horni
% poloviny grafického okna
>> plot(x, y1);
>> subplot(2, 1, 2);
% Tento prikaz vykresli nasledujici graf do dolni
% poloviny grafického okna
>> plot(x, y2);
```

Grafický výstup předchozích příkazů je zobrazen na obrázku:



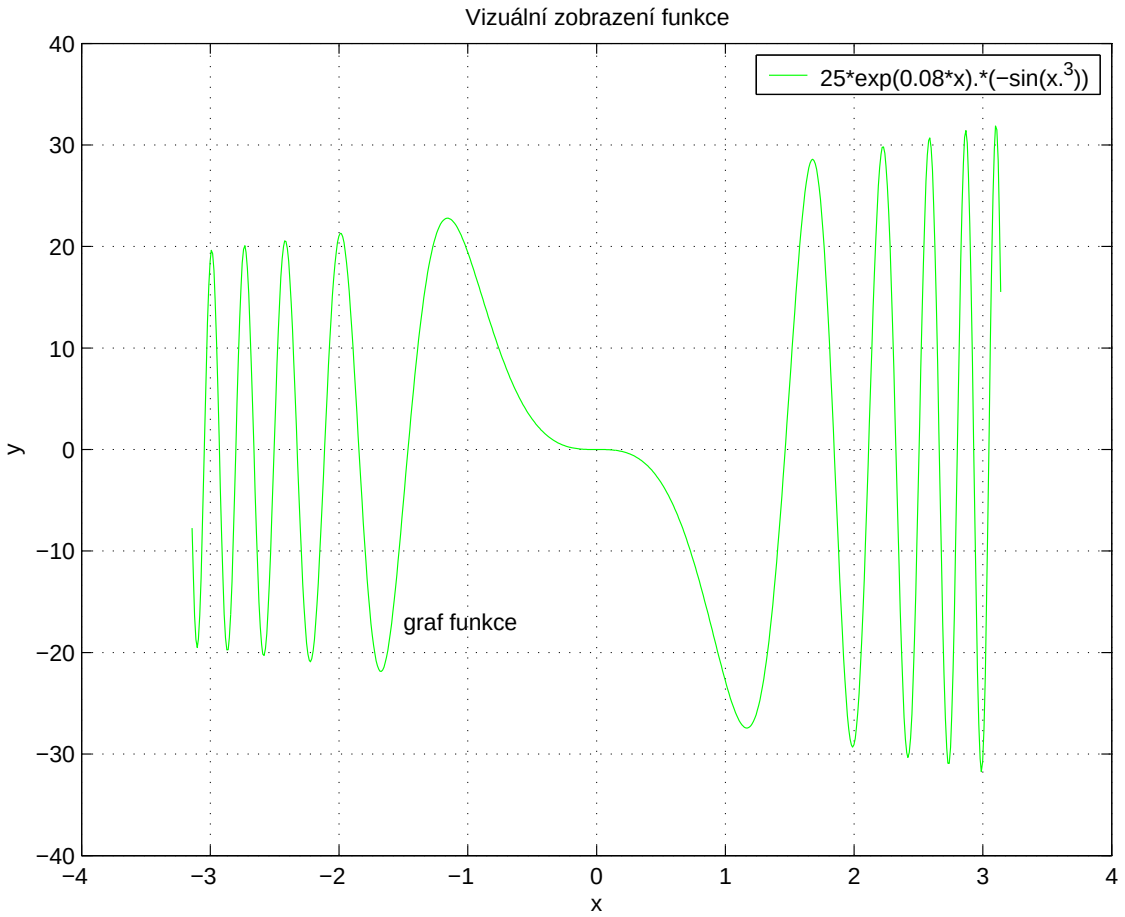
Pozor je třeba dát na škály vyznačené na osách grafů. Jako v předchozím příkladu se oba grafy zdají být co do vertikálních rozměrů shodné, při pohledu na vertikální osu však zjistíme, že první graf má dva a půl krát větší amplitudu. Je to způsobeno tím, že grafické okno Matlabu je rozděleno rovnoměrně bez ohledu na měřítka os jednotlivých grafů.

3.3 Editace 2D grafu

Důležitou součástí grafických výstupů jsou jejich popis a další grafické doplňky, které napomáhají k snadnější čitelnosti grafu. Matlab umožňuje zobrazení těchto informací přímo v okně grafu jednoduchými příkazy. Každý graf je možné popsat nadpisem (`title`), popisy os (`xlabel` a `ylabel`), legendou (`legend`), libovolně umístěným textovým polem v okně grafu (`text`) a grafickou mřížkou (`grid`):

Příklad 3.12 Vytvoříme vektory ekvidistantně rozložených hodnot `x` v intervalu $< -\pi, \pi >$ a k němu vektor stejné délky funkce `y`. Pak vykreslíme graf této funkce zelené barvy a k němu nadpis, popisy os, legendu, ve které je aritmetický výraz pro výpočet hodnot funkce a textové pole, umístěné podle zadaných hodnot souřadnice.

```
>> x = -pi:0.01:pi;
>> y = 25 * exp(0.08 * x) .* (-sin(x.^3));
>> plot(x, y, 'g');
>> title('Vizuální zobrazení funkce');
>> xlabel('x');
>> ylabel('y');
>> legend('25 * exp(0.08 * x) .* (-sin(x.^3))');
>> text(-1.5, -17, 'graf funkce');
>> grid on;
```



Příkaz `text`, vypíše zadaný textový řetězec na určené souřadnice v grafu. Existuje i obdobný příkaz `gtext`, který po spuštění nabízí zvolit místo pro vložení řetězce do grafu interaktivně kliknutím myší.

Další z parametrů zobrazení grafu je vyvolán příkazem `axis`. S jeho pomocí jsme schopni nastavit poměr os grafu x a y (např. intervaly hodnot na osách, stejné měřítko, nebo stejně dlouhé osy). Podrobný popis tohoto příkazu i s mnoha dalšími parametry naleznete v helpu (pro srovnání tento příkaz aplikujte na jeden graf nejprve s parametrem `equal` a následně s parametrem `square`).

Shrnutí:

- Příkaz subplot k rozdělení grafického okna.
- Nastavení poměru a krajních mezí os grafu.
- Přikreslování grafů do grafického okna příkazem `hold`.
- Interaktivní práce v grafickém okně.

Kontrolní otázky:

1. Jak budete generovat náhodné číslo z rovnoměrného rozdělení z intervalu $[a,\ b)$?
2. Znáte příklad, kdy je výsledek funkce zároveň vstupním parametrem jiné funkce?
3. Kdy je vhodné použít vykreslování více grafů do jednoho obrázku současně a kdy dělením okna pomoci subplot?

4 Logické výrazy

Průvodce studiem:

Cílem této kapitoly je naučit se pracovat s logickými výrazy, které je pak dále možno používat především ve složitějších podmíněných příkazech `if - else`. V úvodu se seznámíme se základními relačními operátory, poté se naučíme skládat logické výrazy do složitějších konstrukcí pomocí logických operátorů. Dále se budeme věnovat hledání a zjišťování přítomnosti prvků ve vektorech a na závěr se dotkneme i tématu práce s množinami. Probíraná látka je poměrně jednoduchá a celá kapitola by vám neměla zabrat více než 3 hodiny i s vypracováním úkolů.

4.1 Reprezentace logických hodnot

Hodnoty logických výrazů jsou v Matlabu reprezentovány pomocí číselných hodnot `0` a `1`. Jak se snadno můžeme přesvědčit, existují i předdefinované konstanty `false` a `true`:

```
>> false
ans = 0

>> true
ans = 1
```

4.2 Relační operátory

Relační operátory ve své základní podobě slouží především pro porovnávání čísel a jejich výsledkem je logická hodnota:

```
>> 4 > 2
ans = 1

>> 10 <= 1
ans = 0
```

V Matlabu můžeme používat následující relační operátory:

Operátor	Alternativa	Význam
`a == b`	`eq(a, b)`	a je rovno b
`a ~= b`		a je různé od b
`a < b`	`lt(a, b)`	a je menší než b
`a > b`	`gt(a, b)`	a je větší než b
`a <= b`		a je menší rovno b
`a >= b`		a je větší rovno b

Pozor! Je velký rozdíl mezi `==` a `=`

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

Poznámka 4.1

Srovnávat lze také celé matice, pokud jsou rozměrově stejné. Výsledkem je stejně velká matice odpovědí – srovnání probíhá prvek po prvku:

```
>> A = [1 2; 3 4]; B = [1 1; 4 4];
>> A < B
ans =
  0  0
  1  0
```

Výsledek lze číst tak, že pouze ve druhém řádku a prvním sloupci je hodnota v matici `A` ostře menší než v matici `B`.

4.3 Logické operátory

S logickými hodnotami lze také pracovat při logických operacích. V těchto případech je číslo nula vždy chápáno jako nepravda a jakékoliv jiné, tj. nenulové číslo jako pravda. K dispozici pak máme následující operátory:

Operátor	Alternativa	Význam
`x & y`	`and(x, y)`	x a zároveň y
`x | y`	`or(x, y)`	x nebo y
`~x`	`not(x)`	negace x
	`xor(x, y)`	exkluzivní nebo

Příklad použití:

```
>> 0 & 1
ans = 0

>> 0 | 1
ans = 1

>> ~6
ans = 0
```

K dispozici jsou též operátory pro neúplné vyhodnocování logického součinu a součtu, kdy vyhodnocování skončí v okamžiku, kdy je již znám výsledek na základě části výrazu. Jedná se vlastně o zdvojené symboly pro tyto operace:

```
>> 0 && (4 < 5)
ans = 0

>> 1 || (4 < 5)
ans = 1
```

Ani v jednom z těchto příkladů se nevyhodnocoval výraz `4 < 5`, neboť výsledek celého výrazu je jasný již z první části.

Výše uvedená vlastnost neúplného vyhodnocování může mít někdy nežádoucí vedlejší účinky a je proto vždy zapotřebí jednat obezřetně a zvážit, zda je vhodné tohoto způsobu využít, či ne, případně v jakém pořadí části složeného výrazu zapsat. Například při použití části kódu jako je tento:

```
if ((value > 3) && check(value))
  ...
end
```

se volání funkce `check` provede pouze v případě, že hodnota proměnné `value` je větší než 3.

Konečně stejně jako u relačních operátorů lze logickými operátory spojovat celé matice, pokud mají stejné rozměry:

```
>> A = [1; 1; 0; 0]; B = [1; 0; 1; 0];
>> C = [A, B, A & B, A | B, xor(A, B)]

C =
  1  1  1  1  0
```

```
  1   0   0   1   1
  0   1   0   1   1
  0   0   0   0   0
```

4.4 Hledání prvků

Hodnoty pravda a nepravda také vrací funkce (kvantifikátory) `any` a `all`. Je-li jejich parametrem vektor, vracejí skalár, tzn. logickou hodnotu, zda alespoň jeden, resp. všechny prvky vektoru mají nenulovou hodnotu (logickou hodnotu `true`). Např.:

```
>> V = [0 1 2 3]; W = [1 2 3 4]; X = [0 0 0 0];
>> [any(V), any(W), any(X)]
ans =
  1   1   0

>> [all(V), all(W), all(X)]
ans =
  0   1   0
```

Pokud tyto funkce aplikujeme na matici, jako výsledek získáme vektor odpovědí – ke každému sloupci matice jednu odpověď:

```
>> A = [0 1 0; 1 2 0]
A =
  0   1   0
  1   2   0

>> any(A)
ans =
  1   1   0

>> all(A)
ans =
  0   1   0
```

Kde je v odpovědi jednička, tak v takovém sloupci v pořadí daná podmínka platí. Chceme-li získat jedinou odpověď pro celou matici, můžeme funkce aplikovat vícekrát. Dotaz

```
>> any(any(A))
ans = 1
```

zjistí, zda v celé matici je alespoň jedno číslo nenulové; kombinací funkcí

```
>> all(any(A))
ans = 0
```

zase zjistíme, zda všechny sloupce matice obsahují alespoň jedno nenulové číslo apod.

Poznámka 4.2

Podobně fungují i některé numerické funkce, např. `mean, std, median, min, max, sum, prod`, takže můžeme velmi snadno spočítat řádkový vektor sloupcových průměrů, směrodatných odchylek atd.

Příklad 4.1 Pomocí operátorů `any` a `all` můžeme zjistit, zda aspoň jeden či všechny prvky matice nebo vektoru splňují nějakou logickou podmínku, např. zda všechny prvky matice jsou nezáporné:

```
>> B = [-1, 3, 3; -1, 0, 3]
B =
  -1   3   3
  -1   0   3

>> all(all(B >= 0))
ans = 0
```

Zatímco funkce `any` a `all` dávaly odpovědi ano a ne, s pomocí funkce `find` lze najít pozice prvků, které nějakou podmínku splňují. Přesně definováno, funkce `find` vrací sloupcový vektor pozic nenulových prvků v matici:

```
>> A = [0 1 0; 1 2 0]
A =
  0   1   0
  1   2   0

>> find(A)
ans =
  2
```

```
  3
  4
```

V takovémto jednoduchém případě se pozice počítají od jedničky po sloupcích směrem od shora dolů, aktuální pozice tedy odpovídá přepočtu:

$$\text{pozice} = \text{řádek} + \text{počet řádků} \times (\text{sloupec} - 1)$$

```
A =          %  Pozice v A =
  0  1  0    %             1  3  5
  1  2  0    %             2  4  6
```

Poznámka 4.3

Funkci `find` lze „donutit“, aby pozice prvků vracela jako dva vektory, kdy v jednom budou indexy řádků a ve druhém indexy sloupců; správné souřadnice pak získáme, vezmeme-li dvojice čísel ze stejných pozic v těchto vektorech:

```
>> [radky, sloupce] = find(A)
radky =
  2
  1
  2
sloupce =
  1
  2
  2
```

Pro lepší přehlednost pak můžeme tyto sloupcové vektory vypsat vedle sebe v jedné matici:

```
>> [radky, sloupce]
ans =
  2  1  % cteno po radcich vidime souradnice
  1  2
  2  2
```

Výsledek nyní čteme tak, že nenulové prvky jsou v matici `A` ve druhém řádku a prvním sloupci, v prvním řádku a druhém sloupci a také v druhém řádku a druhém sloupci.

Funkce `find` se nejčastěji používá pro nalezení prvků splňujících logickou podmínku. Např. pozice záporných prvků vektoru `b` nalezneme pomocí:

```
b = [0.2621, -0.0435, -0.4815, 0.3214, -0.0553];
>> find(b < 0)
ans = 2 3 5
```

Jejich vypuštění z vektoru `b` můžeme dosáhnout elegantně např. takto:

```
>> b(find(b < 0)) = []
b = 0.2621 0.3214
```

V některých situacích se neobejdeme bez funkce `logical`, která převádí číselné hodnoty na logické:

```
>> logical(3.8)
ans = 1
```

Jedná se například o případ, kdy potřebujeme vybrat prvky z matice v závislosti na logických hodnotách v jiné matici. Vše vysvětlí následující příklad:

Příklad 4.2 Budeme chtít získat prvky z hlavní diagonály matice `A` a z demonstračních důvodů k tomu nepoužijeme vestavěnou funkci `diag(A)`. Pomocí funkce `eye(n)` získáme n-rozměrnou jednotkovou matici:

```
>> eye(3)
ans =
  1  0  0
  0  1  0
  0  0  1
```

Tuto matici bychom chtěli použít jako „masku“ pro výběr prvků z matice `A`. Když však použijeme přímo následující konstrukci, obdržíme:

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

>> A(eye(3))
??? Subscript indices must either be real positive integers
or logicals.
```

V tomto případě nám pomůže výše zmíněná funkce `logical`:

```
>> A(logical(eye(3)))
ans =
     1
     5
     9
```

4.5 Množiny

Vektor lze chápat jako množinu, pokud neobsahuje žádné duplicitní prvky – tento požadavek splní funkce `unique`:

```
>> v = [7 7 2 3 3 3 7 7 2];
>> mnozina = unique(v)
mnozina =
  2  3  7
```

Poznámka 4.4

Můžeme si povšimnout, že funkce vrací výsledek setříděný od nejmenšího čísla po největší.

Funkce `unique` (podobně jako i další dále uvedené funkce) má několik variant použití. Pokud potřebujeme znát indexy prvků výsledné množiny v původní množině, můžeme použít následující konstrukci:

```
>> [B, I, J] = unique([7 7 2 3 3 3 7 7 2])

B =
  2  3  7

I =
  9  6  8

J =
  3  3  1  2  2  2  3  3  1
```

V proměnné `B` je uložena výsledná množina a `I` a `J` jsou indexové vektory tak, že platí: `B = A(I)` a `A = B(J)`. Z vektoru `I` tedy poznáme, že např. první prvek

výsledné množiny `B` byl v původním vektoru na devátém (posledním) místě. Toto můžeme ovlivnit přidáním dalšího parametru `'first'`, kterým zajistíme, že budou vybírány první výskyty ze vstupního vektoru:

```
>> [B, I, J] = unique([7 7 2 3 3 3 7 7 2], 'first')

B =
  2  3  7

I =
  3  4  1

J =
  3  3  1  2  2  2  3  3  1
```

Množinové operace lze provádět i s maticemi. V tomto případě musíme ale specifikovat, zda chceme pracovat s prvky matice, anebo s jejími řádky. Toto provedeme jednoduše přidáním parametru `'rows'` v případě, kdy požadujeme aby prvky množin byly jednotlivé řádky matice.

```
>> M = [1 2 3 3; 2 5 1 7; 1 2 1 1; 2 5 1 7]
M =
  1  2  3  3
  2  5  1  7
  1  2  1  1
  2  5  1  7

>> unique(M)
ans =
  1
  2
  3
  5
  7

>> unique(M, 'rows')
ans =
  1  2  1  1
  1  2  3  3
  2  5  1  7
```

Další funkce pro práci s množinami jsou `union`, `intersect`, `setdiff` a `setxor`, přičemž jejich význam a varianty použití jsou shrnuty v následující tabulce:

Funkce	Varianta	Popis

`union`	`V = union(M, N)`
	`V = union(M, N, 'rows')`
	`[V, I, J] = union(M, N)`
		sjednocení množin: $V = M \cup N$
		řádkové sjednocení matic M, N
		$V = M \cup N$
		$V = M(I) \cup N(J)$

`intersect`	`V = intersect(M, N)`
	`V = intersect(M, N, 'rows')`
	`[V, I, J] = intersect(M, N)`
		průnik množin: $V = M \cap N$
		řádkový průnik matic M, N
		$V = M \cap N$
		$V = M(I), V = N(J)$

`setdiff`	`V = setdiff(M, N)`
	`V = setdiff(M, N, 'rows')`
	`[V, I] = setdiff(M, N)`
		rozdíl množin: $V = M \setminus N$
		řádkový rozdíl matic M, N
		$V = M \setminus N$
		$V = M(I)$

`setxor`	`V = setxor(M, N)`
	`V = setxor(M, N, 'rows')`
	`[V, I, J] = setxor(M, N)`
		symetrický rozdíl množin:
		$V = M \triangle N$
		sym. rozdíl řádků matic M, N
		$V = M \triangle N$
		$V = M(I) \cup V = N(J)$

Zajímavá je také funkce `ismember`, která slouží pro zjišťování, zda nějaký prvek v množině leží nebo neleží. Její použití má opět několik variant:

- `ismember(x, S)` – vrací `true`, pokud prvek $x \in S$, jinak vrací `false`.

```
>> ismember(5, [2 5 8 10])
ans =
     1
```

- `ismember(M, S)` – vrací pole nul a jedniček o stejné velikosti jako množina `M`, přičemž jedničky jsou na místech prvků z množiny `A`, které leží v množině `S`.

```
>> ismember([1, 2, 3, 5, 10], [2 5 8 10])
ans =
  0  1  0  1  1
```

- `ismember(M, S, 'rows')` – maticová varianta
- `[TF, I] = ismember(M, S)` – navíc ještě do proměnné `I` vloží indexy odpovídající pozicím, na kterých se prvky z množiny `M` v množině `S` nacházejí

```
>> [TF, I] = ismember([1, 2, 3, 5, 10], [2 5 8 10])
TF =
  0  1  0  1  1

I =
  0  1  0  2  4
```

Je tedy zřejmé, že platí:

```
>> all(TF == logical(I))
ans =
     1
```

V souvislosti s množinami nesmíme opomenout ani funkci `isempty(M)`, která vrací `true` v případě, že množina `M` je prázdná.

Shrnutí:

- Logické hodnoty `true`, `false`
- Operátory `==`, `~=`, `<`, `>`
- Operátory `any`, `all`
- Množinové operace: sjednocení, průnik, rozdíl, příslušnost prvku do množiny.

Kontrolní otázky:

1. Vysvětlete rozdíl mezi:
   - `x = (z == y)` a
   - `x == (z == y)`
2. Vysvětlete rozdíl mezi použitím `(x & y)` a `(x && y)`
3. Sestavte příkaz, který pro zadanou matici zjistí, zda jsou všechny její prvky větší než 3.
4. Je dána matice `A = [1 2 3; 4 5 6; 1 2 3]`. Jak zjistíte, zda jsou hodnoty prvků v prvním řádku matice `A` menší než hodnoty prvků ve třetím sloupci matice `A`. Zapište výsledek operace.

5 Znakové řetězce

Průvodce studiem:

V této kapitole se naučíme pracovat s řetězcovými proměnnými (angl. string), které v Matlabu slouží pro reprezentaci textových dat. Po základních operacích a konstrukcích se budeme zabývat porovnáváním a vyhledáváním v řetězcích a v závěru si uvedeme přehled nejčastěji používaných funkcí, které se při práci s řetězci znaků používají. Na kapitolu si vyhraďte zhruba 2 až 3 hodiny.

5.1 Základní operace s řetězci

Znakový řetězec v Matlabu je řádkový vektor, jehož prvky jsou jednotlivé znaky. Pro vytvoření řetězce slouží apostrofy, do kterých se vepíše obsah řetězce, přičemž takto vytvořený řetězec můžeme uložit do proměnné:

```
>> nazev = 'Uvod do Matlabu'
nazev =
Uvod do Matlabu
```

Pokud je potřeba mít v řetězci znak apostrof, je potřeba dát toto najevo zpětným lomítkem zapsaným před požadovaným znakem. Jedno zpětné lomítko se také zapíše jako dvě zpětná lomítka:

```
>> '\'\\'
ans =
  '\
```

Poznámka 5.1

Jelikož je řetězec vektor složený ze znaků, můžeme s ním pracovat znak po znaku jako s běžným polem – k jednotlivým znakům tedy lze přistupovat pomocí indexů stejně jako u číselných vektorů a matic:

```
>> nazev = 'Uvod do Matlabu';
>> nazev(1)
ans =
  U

>> nazev(9:14)
ans =
```

```
  Matlab

>> length(nazev)
ans =
  15
```

Poznámka 5.2

Pokud chceme vypsat obsah řetězce bez onoho „`ans =`“ na začátku, je na místě použití funkce `disp`:

```
>> disp(nazev(9:14))
Matlab
```

Zapsání více řetězců jako prvků ve vektoru má za následek jejich spojení – zřetězení:

```
>> J = 'Josef'; P = 'Petr';
>> oba = [J, ' a ', P]
oba =
  Josef a Petr
```

5.2 Konverze mezi čísly a řetězci

Je-li částí nějakého vektoru řetězec, chápe se jako řetězec celý vektor. Proto se následující příklad nechová tak jak by se na první pohled mohlo zdát.

```
x = 35;
>> disp(['Vysledkem je cislo: ', x])
Vysledkem je cislo: #
```

Předchozí příklad proto namísto čísla 35 vypsal znak „`#`“, který se nachází na 35. pozici v tabulce znaků. Pro nápravu této situace existuje funkce `num2str`, která převádí čísla na jejich textovou podobu:

```
>> disp(['Vysledkem je cislo: ', num2str(x)])
Vysledkem je cislo: 35
```

Opačná funkce `str2num` převádí platnou textovou podobu čísla na číslo, se kterým je možné dále počítat – provádět matematické operace.

```
>> c = str2num('1') + str2num('1')
c =
  2
```

Ještě větší možnosti při formátování výstupu dává funkce `sprintf`, která se chová obdobně jako stejnojmenná funkce z jazyka C. Prvním argumentem funkce je tzv. „formátovací řetězec“ a za ním následuje předem neurčený počet vlastních dat, která mají být do výstupu zahrnuta. Jejich počet je určen až za běhu programu a musí odpovídat počtu formátovacích sekvencí z formátovacího řetězce.

Formátovací možnosti, jež tato funkce nabízí jsou opravdu široké, takže pro podrobnosti odkazujeme spíše na help. Nejčastější formátovací sekvence jsou následující:

- `%d`: celé číslo
- `%e`: exponenciální notace zápisu desetinného čísla
- `%f`: fixní zápis desetinného čísla
- `%g`: obdobně jako `%e` nebo `%f`, ale netisknou se zbytečné nuly
- `%s`: řetězec

U desetinných čísel můžeme ovlivnit na kolik míst nebo platných číslic chceme výstup naformátovat. Vše nejlépe vysvětlí příklad převzatý přímo z helpu Matlabu:

Příkaz	Výstup
`sprintf('%0.5g', (1 + sqrt(5))/2)`	`1.618`
`sprintf('%0.5g', 1/eps)`	`4.5036e+15`
`sprintf('%15.5f', 1/eps)`	`4503599627370496.00000`
`sprintf('%d', round(pi))`	`3`
`sprintf('%s', 'hello')`	`hello`
`sprintf('The array is %dx%d.', 2, 3)`	`The array is 2x3`

Poznámka 5.3

Užitečné často bývá použití `\n`, které způsobí zalomení výstupu na nový řádek:

```
>> sprintf('%1d\n%2d\n%3d\n%4d\n%5d', 1, 2, 3, 4, 5)
ans =
1
 2
  3
   4
    5
```

5.3 Pole řetězců

Řetězce lze ukládat také do matic. Podmínkou však je, že všechny řetězce musejí mít stejnou délku. Když se pokusíme o následující konstrukci, obdržíme chybové hlášení:

```
>> jmena = ['Josef'; 'Petr'; 'Viktor']
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

Nápravu sjednáme buď tak, že ručně doplníme mezery tak, aby měly všechny řetězce stejnou délku, anebo, což je vhodnější, použijeme k tomu určenou funkci `char`, která přidá mezery na konce kratších řetězců, aby vzniklá matice měla ve všech řádcích stejný počet sloupců:

```
>> jmena = char('Josef', 'Petr', 'Viktor')
jmena =
  Josef
  Petr
  Viktor
```

Poznámka 5.4

Díky funkci `strjust` nemusí být texty v matici nutně zarovnány doleva. Přípustnými parametry jsou `left`, `center` a `right`:

```
>> strjust(jmena, 'right')
ans =
 Josef
  Petr
Viktor
```

Pokud nám z nějakého důvodu požadavek na stejnou délku řetězců nevyhovuje, můžeme místo matice použít tzv. pole buněk (angl. cell array). Rozdíl oproti maticím je vesměs jen v typu závorek – u pole buněk se používají složené:

```
>> M = {'Josef', 'Tvrdik'; 'Petr', 'Bujok'; 'Viktor', 'Pavliska'}
M =
    'Josef'      'Tvrdik'
    'Petr'       'Bujok'
    'Viktor'     'Pavliska'
```

5.4 Porovnávání a vyhledávání řetězců

Chceme-li porovnat dva řetězce, zda jsou shodné, není relační operátor „==“ na místě, pokud nás nezajímá shodnost jednotlivých znaků, která navíc funguje jen pro stejně dlouhé řetězce jako u číselných vektorů:

```
>> 'Jana' == 'Dana'
ans =
  0  1  1  1

>> 'Ano' == 'Ne'
??? Error using ==> eq
Matrix dimensions must agree.
```

Pro srovnávání podle obvyklých představ je k dispozici funkce `strcmp`:

```
>> strcmp('Jana', 'Dana')
ans =
  0
```

Pomocí funkce `strfind(text, vzor)` slouží k vyhledávání podřetězce v jiném řetězci. Výsledkem je vektor všech vyhovujících pozicí.

```
>> strfind('krabice hranice slepice', 'ice')
ans =
  5  13  21
```

Poznámka 5.5

Je možno také použít funkci `findstr(s1, s2)`, která se liší od předchozí pouze v tom, že u ní nezáleží na pořadí parametrů `s1` a `s2` a vždy provádí vyhledávání kratšího řetězce v delším (v případě, že mají řetězce stejnou délku, tak se v podstatě jedná o porovnání dvou řetězců).

Příklad 5.1 Někdy je však použití operátoru `==`, resp. `~=` ve spojitosti s řetězci elegantní. Například pro odstranění mezer z řetězce můžeme použít následující konstrukci:

```
>> s = 'retezec s mezerami';
>> neprazdne_znaky = (s ~= ' ');
>> s(neprazdne_znaky)
```

```
ans =
retezecsmezerami
```

Proměnná `neprazdne_znaky` je v tomto příkladě logický vektor obsahující jedničky v místech, kde se v řetězci `s` vyskytují znaky různé od mezery.

Nahrazování podřetězců: Funkce `strrep` odpovídající vzorky v zadaném řetězci nahradí jiným zadaným řetězcem:

```
>> strrep('zena Bozena neni zenata', 'zena', 'spor')
ans =
  spor Bospor neni sporta
```

Funkcí pracujících s řetězci je ještě více, ale nebudeme je všechny dopodrobna rozebírat, takže alespoň v rychlosti zmíníme:

- `lower`, `upper` – mění velikost písmen
- `dec2base`, `base2dec` – převod mezi mezi desítkovou a zvolenou soustavou
- `ischar` – dává odpověd, zda zadaný parametr je skutečně řetězec
- `isletter`, `isstrprop`, `isspace` apod. – vrací matici odpovědí, zda dané znaky v řetězci jsou písmena, čísla, prázdné znaky, ..., podobných funkcí je celá řada.

Více se můžete dozvědět z nápovědy k jednotlivým funkcím – jejich přehled získáte příkazem:

```
>> help strfun
```

Shrnutí:

- Vytváření řetězců a vnitřní reprezentace v Matlabu
- Přístup k jednotlivým znakům řetězce
- Konverze číselných hodnot na řetězce a naopak
- Řetězce v matici a v poli buněk
- Porovnávání řetězců
- Vyhledávání v řetězcích
- Náhrada v řetězcích

Kontrolní otázky:

1. Vytvořte příkaz, kterým zjistíte index posledního výskytu řetězce v proměnné `sub` v textové proměnné `txt`.
2. Vytvořte příkaz, který z plného jména souboru (včetně absolutní cesty a přípony) vypíše pouze vlastní jméno souboru. Např. pro vstup `C:\temp\readme.txt` vypíše pouze `readme`.
3. Předpokládejte, že v poli buněk `M` máte uloženy řetězce. Vytvořte příkaz, jehož výsledkem bude matice jedniček a nul korespondující s porovnáním jednotlivých řetězců s řetězcem `'yes'`.

6 Programování v Matlabu

Průvodce studiem:

Cílem této kapitoly je seznámit se se základními možnostmi zápisu algoritmů v Matlabu. Počítejte asi se třemi až šesti hodinami studia, v závislosti na tom, jak jste zdatní v algoritmizaci a jak zkušení v programování v jiných programovacích jazycích.

6.1 Řídicí příkazy

V Matlabu máme k dispozici běžné příkazy pro zápis algoritmu včetně tzv. řídicích struktur jako je větvení a cyklus, známé z jiných programovacích jazyků.

Sekvence příkazů se zapíše buď jako posloupnost řádků, kdy na každém řádku je jeden příkaz nebo více příkazů na jednom řádku stačí oddělit středníkem. Je-li příkaz ukončen středníkem, vyhodnocený výraz vyhodnocený v tomto příkazu se nevypisuje na obrazovku. Pokud má zápis příkazu pokračovat na dalším řádku, je předcházen třemi tečkami na konci předchozího řádku. Identifikátor muže mít až 31 alfanumerických znaků nebo podtržítek, začíná písmenem, velká a malá písmena se rozlišují. Jako identifikátory neužívejte Matlabem užívané názvy konstant, proměnných a funkcí jako např. `pi, eps, ans, Inf, NaN, realmax, realmin` (a také `i, j`, pokud budete počítat s komplexními čísly), neboť tím byste si přepsali jejich nastavené hodnoty. Vše za znakem procento (`%`) do konce řádku je komentář.

Podmíněný příkazy má tvar

```
if podminka1
    prikaz1
elseif podminka2
    prikaz2
else
    prikaz3
end
```

Části `elseif` a `else` jsou nepovinné, podmíněný příkaz může být zapsán i na jednom řádku.

Přepínač má tvar

```
switch(výraz)
    case value1 prikaz1
    case value2 prikaz2
```

```
    ...
    ...
    ...
    case valuen prikazn
    otherwise nejaky prikaz
```

Část `otherwise` je nepovinná.

Cykly lze zapsat buď jako

```
while podminka
     prikaz
end
```

nebo

```
for  identifikátor = vektor
     prikaz
end
```

Všude, kde je možno zapsat příkaz, může to být i sekvence více příkazů.

Příklad 6.1 V cyklu typu `for` proběhne desetkrát přiřazení vektoru náhodných čísel do $i-$tého řádku matice:

```
for i = 1:10
    P(i, :) = 2 + 3*rand(1, 5);
end
```

Příklad 6.2 V cyklu typu `while` proběhne rozebírání hromady, dokud hromada není menší nebo rovna 1. Na obrazovku se vypíše počet provedených odběrů.

```
hromada = 10;
poc = 0;
while hromada > 1
    hromada = hromada - rand(1);
    poc = poc + 1;
end
display(poc)
```

6.2 M-soubory

Sekvence příkazů je možné zapsat do textových souborů, jejichž jméno má příponu „.m“. Celou sekvenci příkazů pak vyvoláme jménem souboru bez přípony. Pokud tato sekvence nemá žádné vstupní parametry, říkáme takovému souboru *skript (dávka)*. Při jejím vyvolání je pak sekvence příkazů postupně interpretována stejně jako bychom ji znovu po jednotlivých příkazech znovu zadávali v příkazovém okně. Tím si můžeme ušetřit čas i naši zbytečnou práci.

Velmi užitečným prostředkem je zapsat sekvenci příkazů jako *funkci*, tj. vytvořit podprogram. Tak máme k dispozici postup pro modulární strukturu programů v Matlabu. Vytvořenou funkci můžeme pak volat (spouštět) s různými hodnotami vstupních parametrů. Syntaxi zápisu funkce můžeme ve stručnosti vyjádřit schématem

```
function[seznam vystupnich parametru]...
     = jmeno_funkce(seznam vstupnich parametru)
---- prikazy provedene pri volani funkce ----
```

Pokud funkce vrací jen jednu hodnotu (jednu proměnnou, může to být ale i vektor nebo matice), výstupní parametr nemusí být v hranatých závorkách. Proměnné zavedené uvnitř funkce jsou lokální, takže nepřepisují hodnoty stejně nazvaných proměnných ve volajícím modulu a po skončení běhu funkce lokální proměnné zmizí z paměti počítače a místo, které v paměti zabírali, uvolní. Zvolené jméno funkce musí být *shodné se jménem souboru*, ve kterém je zdrojový text funkce uložen (bez přípony „.m“). Užívání funkcí je i výhodné s ohledem na délku výpočtu, neboť při prvním volání proběhne kompilace funkce a další volání je rychlejší než opakovaná interpretace jednotlivých příkazů.

Příklad 6.3 Jako příklad funkce si ukážeme náhodný výběr k hodnot (bez opakování) z množiny $\{1, 2, \ldots, N\}$, $k \leq N$. Pokud by bylo $N = 49$ a $k = 7$, je to procedura losování Sportky. Příklad pečlivě prostudujte, neboť ilustruje několik užitečných operací s vektory v Matlabu.

```
% random sample, k of N without repetition
%
function result = nahvyb(N, k)
opora = 1:N;    % N micku je v osudi
nahv = [];      % zadny micek neni zatim vytazen
for i = 1:k
  index = 1 + fix(rand(1) * length(opora)); % vybran micek
  nahv(end+1) = opora(index);     % uz je vytazen
```

```
  opora(index) = [];        % v osudi vytazeny micek zrusit
end
result = nahv;
```

Uvedený způsob implementace náhodného výběru je jen jedním z mnoha možných. Je to vlastně velmi přímočarý model fyzického losování z osudí.

Příklad 6.4 Jiný způsob implementace může využít funkci `randperm`. Pak je zápis algoritmu ještě kratší:

```
% random sample, k of N without repetition
%
function result = nahvybperm(N, k)
pom = randperm(N);   % utvor nahodnou permutaci cisel 1:N
result = pom(1:k);   % vyber k prvnich
```

Nyní můžeme porovnat, který z těchto dvou algoritmů je rychlejší. Pro testování si napíšeme jednoduchý skript.

```
N =    % sem budeme pri spousteni zadavat hodnoty, napr. 49 nebo 200
k =    % sem budeme pri spousteni zadavat hodnoty, napr. 7 nebo 3
opak = 10000;
tic
for ii = 1:opak
    pom = nahvyb(N, k);
end
toc
tic
for ii = 1:opak
    pom = nahvybperm(N, k);
end
toc
```

Na obrazovce se pak objeví tyto výsledky:

```
N =
    49
k =
     7
```

```
Elapsed time is 0.756440 seconds.
Elapsed time is 0.455496 seconds.

N =
   200
k =
     3
Elapsed time is 0.458998 seconds.
Elapsed time is 0.797050 seconds.
```

Vidíme, že pro větší k a menší N je rychlejší algoritmus `nahvybperm`, zatímco pro menší k a větší N je tomu naopak.

Příklad 6.5 Někdy potřebujeme náhodný výběr s vyloučením některých prvků, tzn. že tyto prvky se v něm nesmí objevit. Toho můžeme snadno dosáhnout malou modifikací funkce `nahvyb`:

```
% random sample, k of N without repetition,
% numbers given in vector expt are not included
%
function vyb = nahvyb_expt(N, k, expt);
opora = 1:N;
if nargin == 3 % pokud zadame jen N a k, funguje jako nahvyb
    opora(expt) = []; % pred losovanim vyjmeme zakazane micky
end
vyb = zeros(1, k); % zabereme misto pro vsech k micku
for i=1:k
    index = 1 + fix(rand(1) * length(opora));
    vyb(i) = opora(index);
    opora(index) = [];
end
```

Pokud bychom chtěli naprogramovat takový výběr s vyloučením některých prvků modifikací funkce `nahvybperm`, taková modifikace by mohla vypadat třeba takto:

Příklad 6.6

```
% random sample, k of N without repetition,
% numbers given in vector expt are not included
%
```

```
function vyb = nahvybperm_expt(N, k, expt)
vyb = randperm(N);
if nargin == 3
    zrus = zeros(1, length(expt));
    for ii = 1:length(expt)
        zrus(ii) = find(vyb == expt(ii));
    end
    vyb(zrus) = [];
end
vyb = vyb(1:k);
```

Při volání funkce nemusí být vždy zadán plný počet vstupních i výstupních parametrů. Pro ošetření takového volání jsou určeny funkce `nargin` a `nargout` (bez parametrů), které vracejí aktuální počet příslušných parametrů, viz předchozí ukázka. Pokud programujete nějakou funkci, vždy si dobře rozvažte, zda dovolíte neúplné zadání vstupních parametrů. Pokud to není pro uživatele nutné nebo extrémně výhodné, tak si nekomplikujte řešení a vyžadujte při volání funkce všechny vstupní parametry.

Někdy potřebujeme, abychom mohli jako vstupní parametr pro nějaký algoritmus zadat funkci, která má být v tomto algoritmu použita. Např. máme naprogramovaný algoritmus (v Matlabu jako podprogram `function`) hledající minimum funkce, v tomto algoritmu potřebujeme minimalizovanou funkci vyhodnocovat, a přitom chceme, abychom takový podprogram mohli využívat nejen pro jednu pevně danou funkci, ale abychom požadovanou funkci mohli zadat. K tomu nám v Matlabu slouží funkce `feval`, jejímiž vstupními parametry jsou jméno funkce a hodnota argumentu funkce, výstupem je pak funkční hodnota zadané funkce v zadaném bodu. Příkazem

```
  y = feval(fnname, x);
```

vyhodnotíme v bodě `x` tu funkci, jejíž jméno je zadáno. Jméno funkce `fnname` může být zadáno jako řetězec se jménem funkce nebo nebo jako tzv. *handle*, což je jméno funkce uvedené znakem `@`. Samozřejmě funkce tohoto jména musí být dostupná, např. jako m-soubor nebo jako vestavěná funkce Matlabu.

Příklad 6.7 Volání funkce `feval` ukážeme v jednoduchém příkladu. Máme naprogramovanou funkci (uloženou v m-souboru, která může být požadovanou alternativou ke standardnímu vyhodnoceni druhé odmocniny:

```
function y = mojeodmocnina(x)
%
%  druhou odmocninu pocita funkce sqrt,
%       pokud je argument zaporny,
%       tak vysledek je komplexni cislo
%
%  mojeodmocnina bude pro x > 0 vracet hodnotu NaN
%
if x >= 0
    y = sqrt(x);
else
    y = NaN;
end
```

Pak v příkazu můžeme specifikací funkce zadat, jakou verzi výpočtu odmocniny si přejeme:

```
>> y = feval('sqrt', -25)
y =
        0 + 5.0000i

>> y = feval(@mojeodmocnina,-25)
y =
   NaN
```

6.3 Doporučení pro vhodný výběr příkazů

Při zápisu algoritmů v Matlabu se vyplatí „uvažovat vektorově“, neboť řadu operací je možno v Matlabu zapsat efektivněji než užitím postupů známých z jazyků nepodporujících vektorové a maticové operace (např. Pascal nebo C). Především je nutno velmi uvážlivě užívat cyklů, zejména vnořených. Různou časovou náročnost ukazují následující příklady výpočtu součtu druhých mocnin prvků vektoru **a**.

Příklad 6.8 Časově nejnáročnější je „klasický postup“ načítání druhých mocnin jednotlivých prvků v cyklu:

```
N = 10000
a = rand(50, 1)
tic
for i = 1:N
```

```
    s = 0;
    for j = 1:length(a)
        s = s + a(j)^2;
    end
end
toc
Elapsed time is 1.2700
```

Využitím aritmetické operace pro všechny prvky vektoru (zde `a .* a` nebo `a.^2`) dosáhneme téměř osminásobného zkrácení potřebného času:

```
tic
for i = 1:N
    s = sum(a .* a);
end
toc
Elapsed time is 0.1600
```

A využijeme-li skalární součin, potřebný čas ještě o trochu zkrátíme:

```
tic
for i = 1:N
    s = a' * a;
end
toc
Elapsed time is 0.1100
```

Příklad 6.9 V tomto příkladu porovnáme časovou náročnost dynamické a statické alokace paměti. Prohlédněme si zdrojové texty funkcí `nahvyb` a `nahvyb_expt`. V prvním případě se paměť pro vylosované míčky obsazuje dynamicky vždy po vylosování každého míčku, ve druhém případě staticky obsadíme paměť pro všechny vylosované míčky na začátku, dopředu víme, pro vylosované míčky budeme potřebovat vektor délky k. Pro časové porovnání si napíšeme následující skript:

```
opak = 1000;
N = 500;
k = 100;
tic
for ii = 1:opak
```

```
    v1 = nahvyb(N, k);
end
toc
tic
for ii = 1:opak
    v1 = nahvyb_expt(N, k);
end
toc
```

Po spuštění skriptu dostaneme následující výsledky.

```
Elapsed time is 1.343568 seconds.
Elapsed time is 1.007942 seconds.
```

Vidíme, že statické alokace v tomto případě vede k zhruba o třetinu menším časovým nárokům.

Shrnutí:

- Řídicí struktury v Matlabu (větvení, cykly, přepínač)
- Skripty a funkce v Matlabu, vstupní a výstupní parametry funkcí
- Lokální proměnné
- Využití vektorových a maticových operací
- Statická a dynamická alokace paměti počítače

Kontrolní otázky:

1. Kdy použít skript, kdy funkci?
2. Jak budete generovat náhodné číslo z diskrétního rovnoměrného rozdělení z intervalu $[1,\ 100]$?
3. Kdy může být jméno funkce vstupním parametrem jiné funkce?
4. Můžete generovat náhodný výběr k hodnot z $[1, 2, \ldots, N]$ pomocí funkce `randperm` jiným způsobem, než je uvedeno v textu? Zkuste navrhnout možné řešení.

7 3D grafika

Průvodce studiem:

Cílem této kapitoly je pochopit princip práce s 3D grafikou v Matlabu, zejména pak různé možnosti výpočtu hodnot Z-souřadnic a interaktivní editace 3D grafů. Pro pochopení látky a zvládnutí cvičení si vyhraďte alespoň čtyři hodiny.

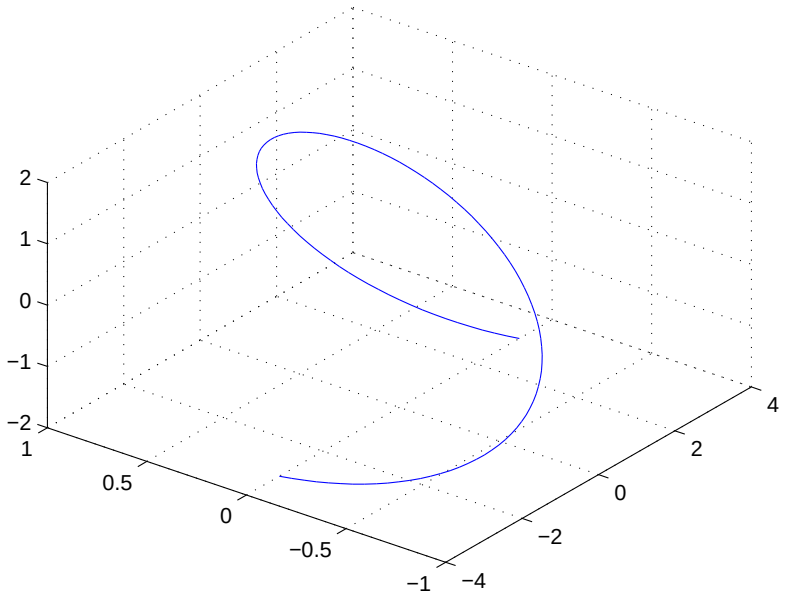
7.1 Jednoduché čárové grafy

Nejjednodušší případ vykreslování 3D grafů v Matlabu je v případě, kdy máme tři vektory proměnných stejných rozměrů. V takovém případě se jedná o tzv. *čárové grafy* a vytváří se pomoci příkazu `plot3`.

Příklad 7.1 Vytvoříme vektor ekvidistantně rozložených hodnot `x` a nakreslíme čárový graf, kde souřadnice `y` a `z` jsou dopočteny podle zadaných funkcí.

```
>> x = -pi:pi/60:pi;
>> plot3(x, sin(x), 2 .* cos(x));
```

Grafem je pak následující křivka v 3D prostoru:



7.2 Kreslení složitějších 3D grafů

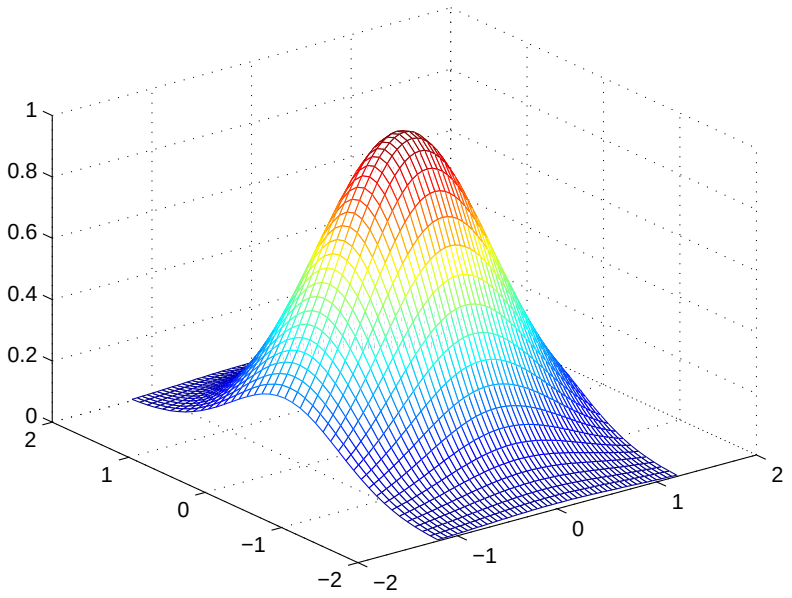
Kreslení grafů složitějších funkcí v 3D si ukážeme na jednoduchých příkladech.

Příklad 7.2 Chceme nakreslit graf funkce $f(x, y) = \exp(-(x^2 + y^2))$ pro hodnoty $x \in [-1,2;\ 1.2]$ a $y \in [-2;\ 2]$. Nejdříve vytvoříme vektory hodnot x-ové i y-ové souřadnice pro síť vhodné hustoty:

```
x = -1.2:0.05:1.2;
y = -2:0.1:2;
```

Pak příkazem `meshgrid` vytvoříme matice `X` a `Y` typu (`length(y)` $\times$ `length(x)`), v tomto případě tedy typu (41×49), ve kterých odpovídající prvky představují všechny dvojice hodnot prvků vektorů `x` a `y`. Pak už jen vyhodnotíme z-tovou souřadnici (matice Z typu (41×49)) a příkazem `mesh` nakreslíme tzv. *drátový graf* do okna Figure.

```
[X Y] = meshgrid(x, y);
Z = exp(-X.^2 - Y.^2);
mesh(X, Y, Z);
% Vykresleni stejneho grafu dosahneme
% zadanim prikazu: 'mesh(x, y, Z)'
```



O trochu složitější je nakreslení trojrozměrného grafu funkce tehdy, když matici z-tových souřadnic nemůžeme spočítat přímo, ale máme jen funkci vyhodnocující z-tovou souřadnici v daném bodě zadaném souřadnicemi x a y.

Příklad 7.3 Chceme nakreslit graf funkce, jejíž hodnotu umíme spočítat jen v jed-

nom zadaném bodu. Vytvoříme vektory `x` a `y` ekvidistantně rozložených hodnot a k nim síť souřadnic. Dopočteme souřadnice vektoru `z` a vykreslíme drátový graf s isoliniemi dané funkce.

```
function z = ackley(x)
% Vyhodnocujeme funkci v bodĄ zadaném vektorem x
  d = length(x);
  a = -20 * exp(-0.02 * sqrt(sum(x .* x)));
  b = -exp(sum(cos(2 * pi * ones(1, d) .* x))/d);
  z = a + b + 20 + exp(1);
```

Síť souřadnic v rovině xy pak vytvoříme podobně jako v předchozím příkladu.

```
>> x = -3:0.05:3;
>> y = -3:0.05:3;
>> [X Y] = meshgrid(x, y);
```

Vypočíst hodnoty funkce umíme pouze v bodech zadaných jako dvouprvkový vektor. Pak nezbývá než spočítat matici `Z` po jednotlivých prvcích

```
for i = 1:length(y)
    for j = 1:length(x)
        Z(i, j) = ackley([X(i, j) Y(i, j)]);
    end
end
```

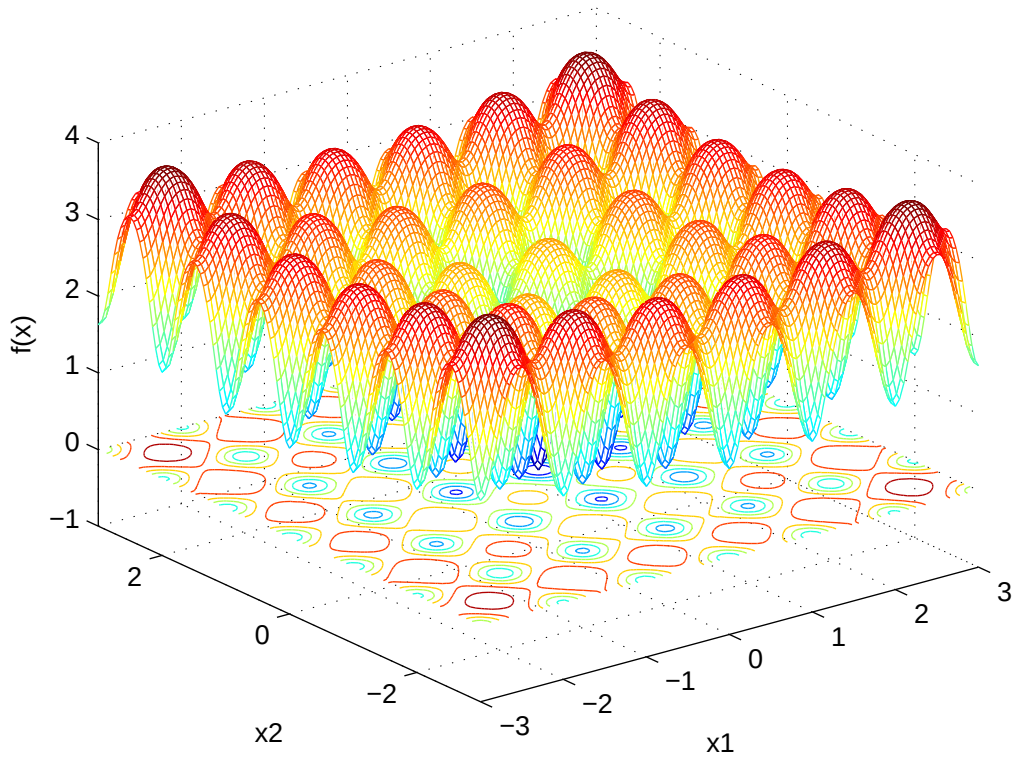
Výpočet je možno trochu zrychlit, když z matic `X` a `Y` vytvoříme vektory, výsledný vektor `zz` předem inicializujeme na požadovanou délku, aby uvnitř cyklu nebylo jeho prvky alokovat dynamicky až v průběhu výpočtu a pak z něho příkazem `reshape` vytvoříme matici požadovanou pro kreslení 3D grafu:

```
xx = X(:); yy = Y(:);
zz = zeros(length(xx), 1);
for i = 1:length(xx)
    zz(i) = ackley([xx(i) yy(i)]);
end
Z = reshape(zz, length(y), length(x));
```

Pak teprve můžeme nakreslit graf následujícími příkazy:

```
meshc(X, Y, Z);
axis([-3 3 -3 3 -1 4]);
xlabel('x');
ylabel('y');
zlabel('z');
```

Graf Ackleyho funkce po předchozích příkazech vypadá, jak ukazuje následující obrázek:



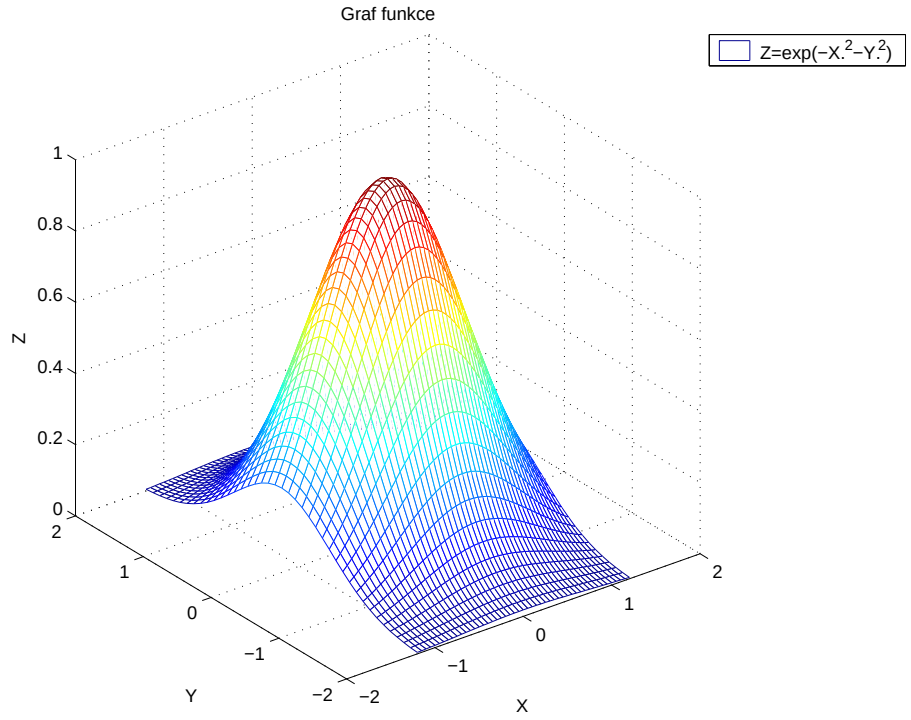
7.3 Editace 3D grafu

Stejně jako u 2D grafiky, i zde je možné vytvořené grafy editovat. V grafickém okně Figure máme mnoho dostupných nástrojů pro interaktivní úpravy výsledného obrázku (např. popis a editaci os grafu, vkládání textů a grafických symbolů, rotace obrázku, uložení do souboru, export do jiného formátu atd).

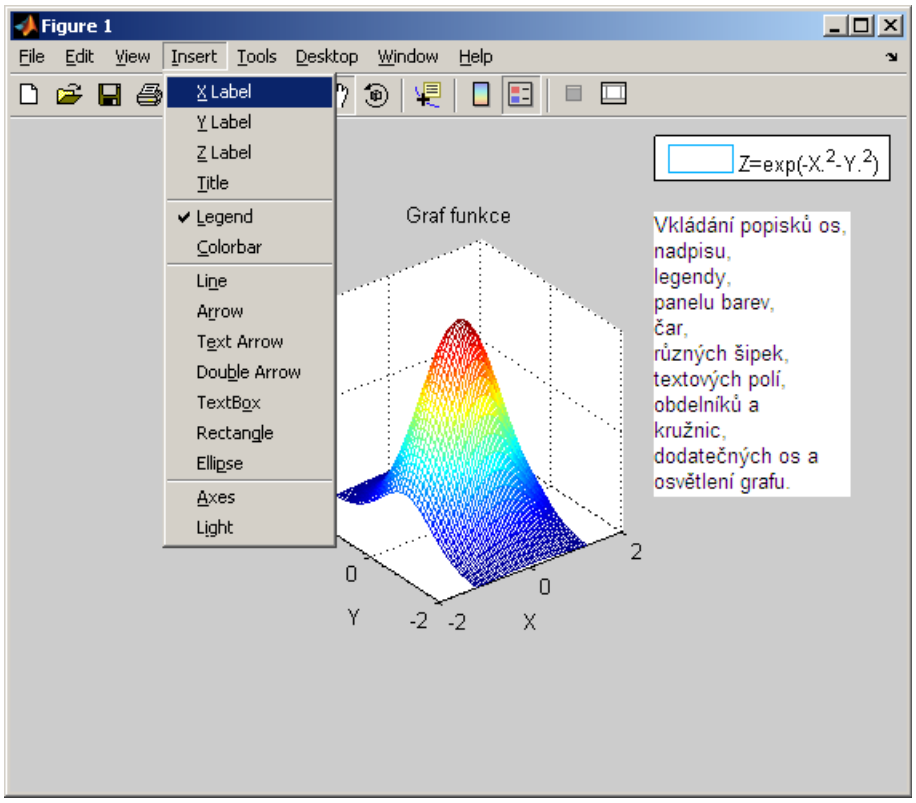
Příklad 7.4 Nakreslíme drátový graf funkce $f(x, y) = exp(-x^2 - y^2)$.

```
>> x = -1.2:0.05:1.2;
>> y = -2:0.1:2;
>> [X Y] = meshgrid(x, y);
```

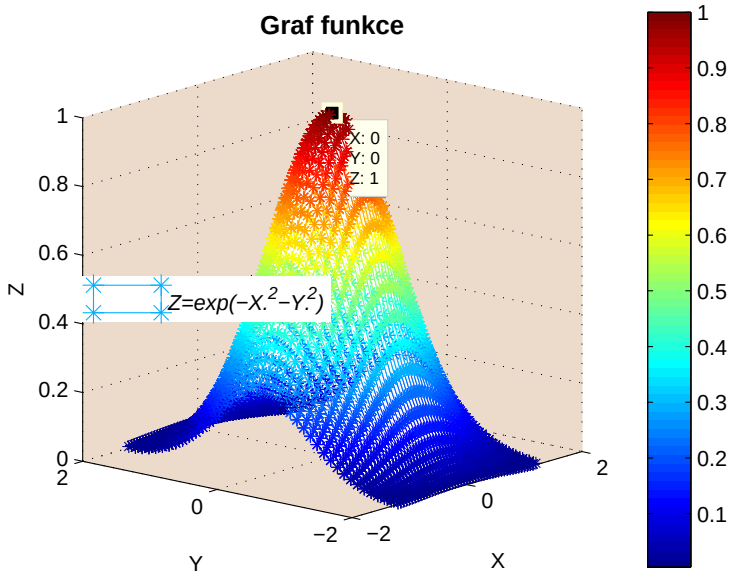
```
>> Z = exp(-X.^2 - Y.^2);
>> mesh(X, Y, Z);

% popisky jednolivych os grafu
>> xlabel('X');
>> ylabel('Y');
>> zlabel('Z');

% nastavení meznich hodnot jednotlivzch os v poradí:
% ([xmin xmax ymin ymax zmin zmax])
>> axis([-2 2 -2 2 0 1]);

% vlozeni legendy do grafu
>> legend('Z = exp(-X.^2 - Y.^2)');

% vlozeni nazvu grafu funkce
>> title('Graf funkce Z = exp(-X.^2 - Y.^2)');
```



Okno pro interaktivní editaci grafu je na následujícím obrázku v jehož pravé části jsou vysvětleny popisy nejpoužívanějších funkcí editace grafu nabídky *Insert*.



Po aplikaci některých funkcí interaktivní editace může graf vypadat např. takto:



Editace grafu v Matlabu lze provádět jak pomocí příkazů, tak interaktivně v grafickém okně Figure. Užitečnou interaktivní editací grafu je rotace grafu. Rotaci grafu si ukážeme v následujícím příkladu.

Příklad 7.5 Vytvoříme vektory `x` a `y` ekvidistantně rozložených hodnot. Vykres-

líme drátový graf Schwefelovy funkce spolu s popisy a pak nakreslený graf v okně Figure interaktivně rotujeme.

Schwefelova funkce je definovaná jako:

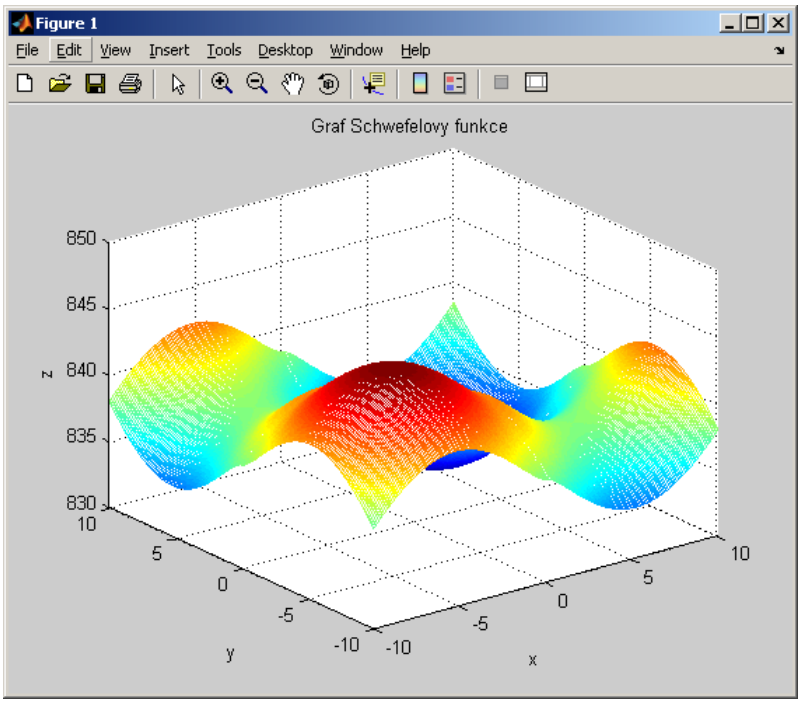
```
function y=schwefel(x)
%   Schwefel's  function, <-500, 500>,
%   glob. min f(x_star)~0, (for d < 80 0 <= f(x_star) < 1e-10),
%   x_star = [s, s, ..., s], s = 420.968746
d = length(x);
sz = size(x);
if  sz(1) == 1
    x = x';
end
y = 418.9828872724338 * d - sum(x .* sin(sqrt(abs(x))));
```

Dopočteme prvky sítě souřadnic této funkce:

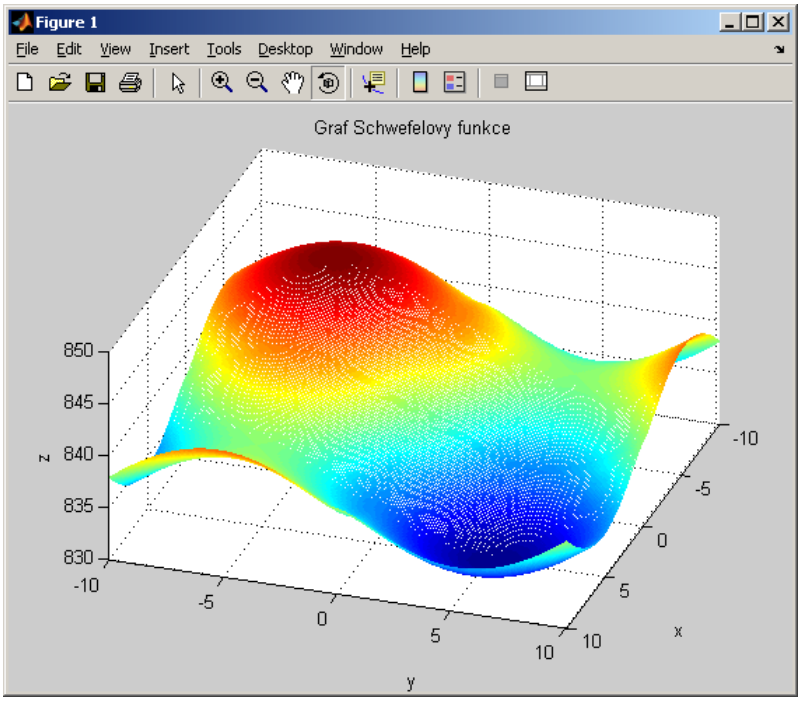
```
>> x = -10:0.1:10;
>> y = -10:0.1:10;
>> [X Y] = meshgrid(x, y);
>> for i = 1:length(y)
        for j = 1:length(x)
            Z(i, j) = schwefel([X(i, j) Y(i, j)]);
        end
end
```

a následujícími příkazy pak vykreslíme její graf:

```
>> mesh(X, Y, Z);
>> title('Graf Schwefelovy funkce');
>> xlabel('x');
>> ylabel('y');
>> zlabel('z');
```



Tento graf můžeme z výchozí polohy následně pomoci tlačítka *Rotace 3D* rotovat tak, abychom docílili lepší pohled na graf, jak ukazují následující obrázky:



Podrobnosti ponecháváme na čtenáři, který může potřebné informace získat v helpu, kde také se dozví i o dalších možnostech 3D grafiky v Matlabu.

7.4 Další typy 3D grafů

V následujících příkladech jsou nakresleny různé typy 3D grafů Rastriginovy funkce, která je definovaná následovně.

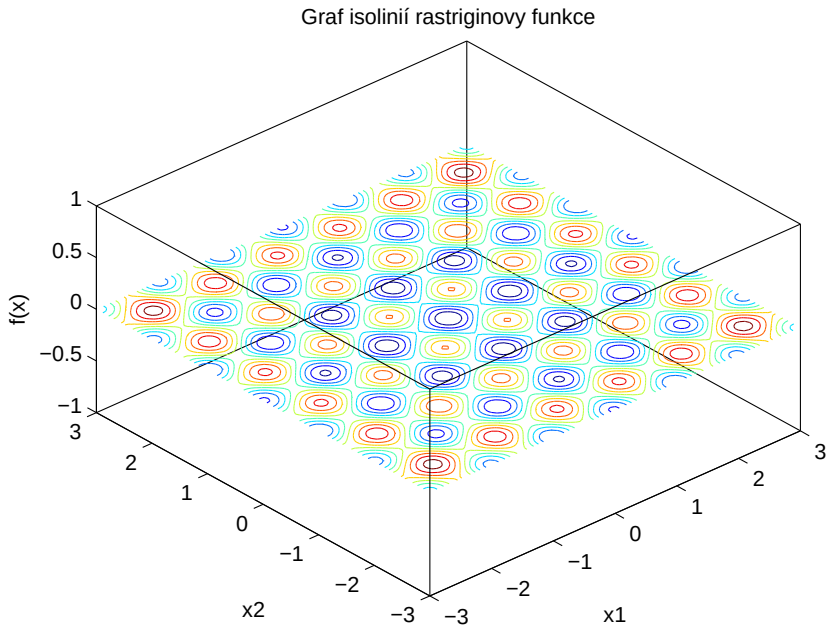
```
function y = rastrig(x)
%  Rastrigin's  function, <-5.12, 5.12>, glob. min f(0) = 0
  d = length(x);
  sz = size(x);
  if  sz(1) == 1  x = x'; end
  y = 10*d + sum(x .* x - 10 * cos(2 * pi * x));
```

Nejprve vypočteme síť souřadnic společnou pro všechny typy grafů.

```
>> x = -3:0.05:3;
>> y = -3:0.05:3;
>> [X Y] = meshgrid(x, y);
for i = 1:length(y)
    for j = 1:length(x)
        Z(i, j) = rastrig([X(i,j) Y(i,j)]);
    end
end
```

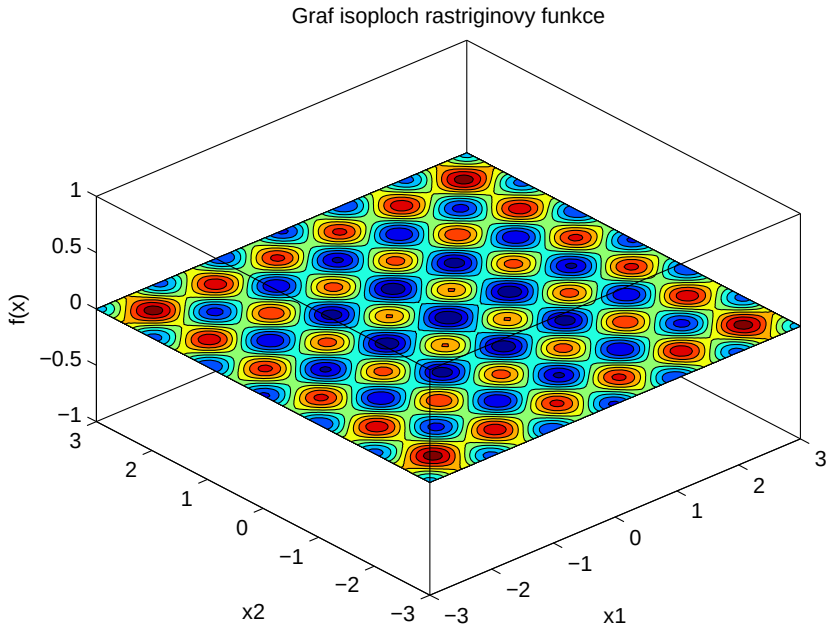
Příklad 7.6 Nakreslíme *graf isolinií* příkazem `contour`.

```
contour(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Graf isolinii rastriginovy funkce');
```



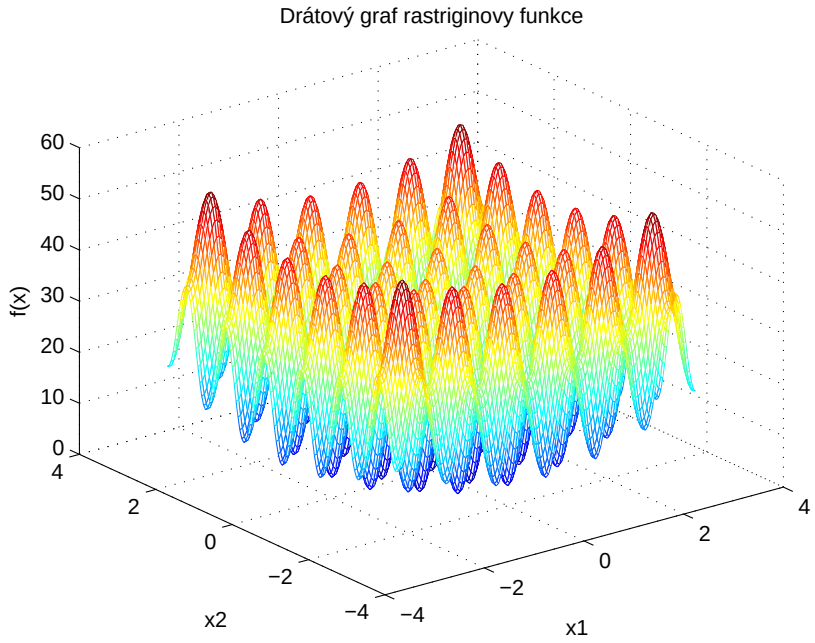
Příklad 7.7 Nakreslíme *graf isoploch* příkazem `contourf`.

```
contourf(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Graf isoploch Rastriginovy funkce');
```



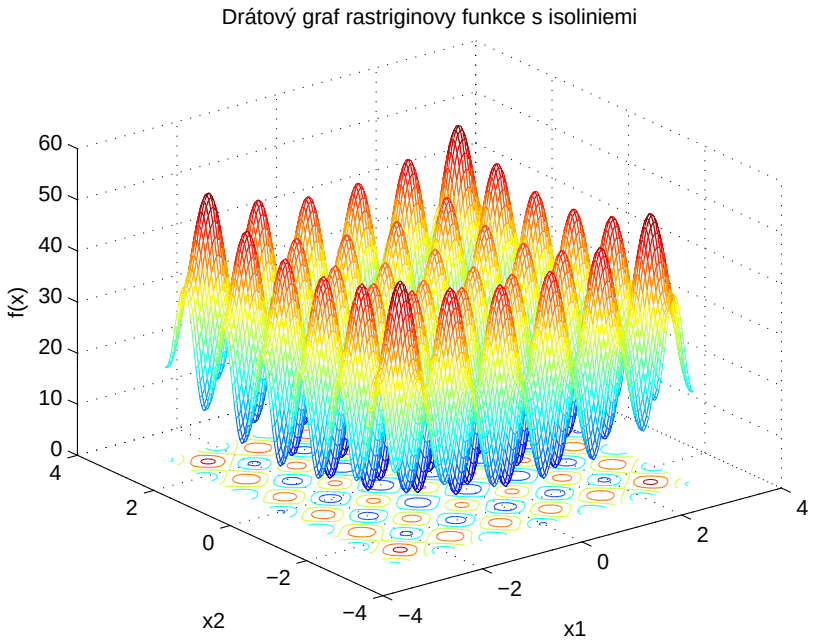
Příklad 7.8 Nakreslíme *drátový graf* příkazem `mesh`.

```
mesh(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Dratovy graf Rastriginovy funkce');
```



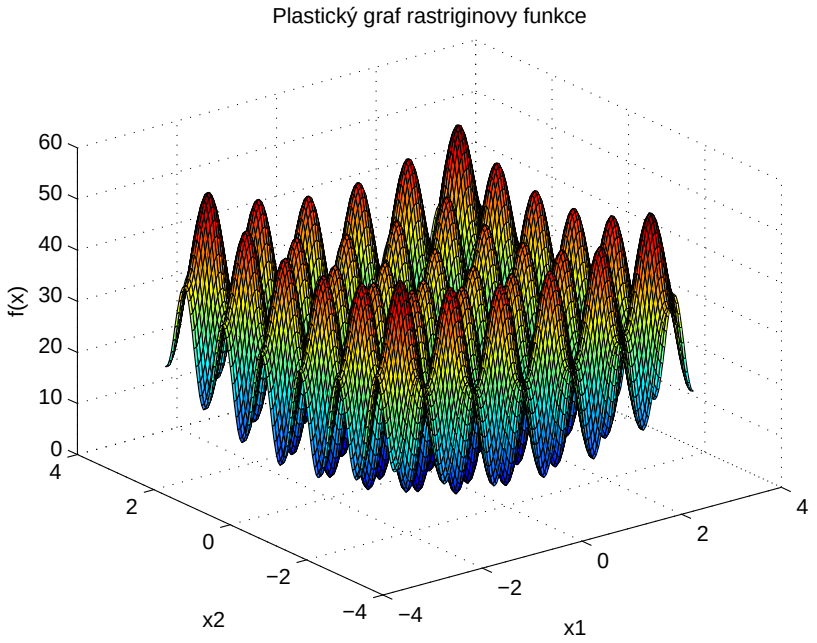
Příklad 7.9 Nakreslíme *drátový graf s isoliniemi* příkazem `meshc`.

```
meshc(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Dratovy graf Rastriginovy funkce s isoliniemi');
```



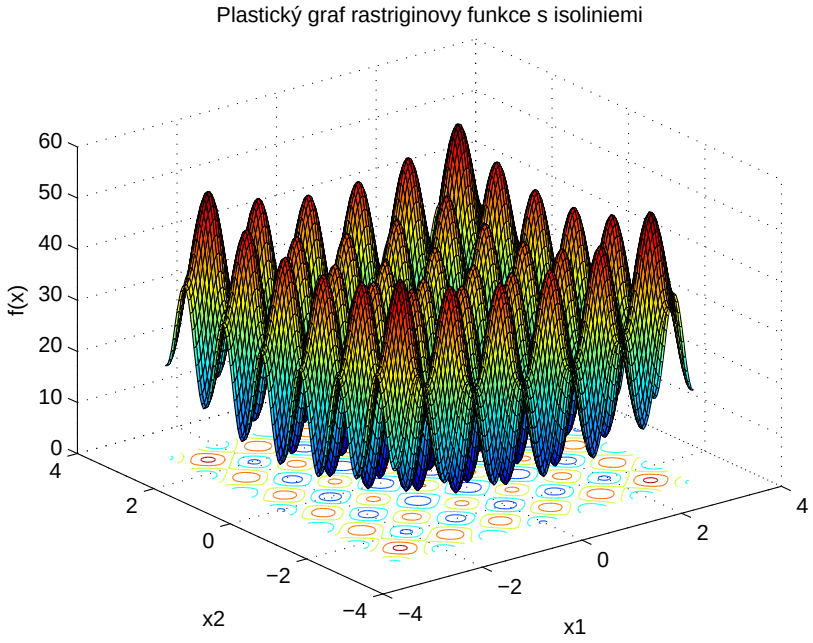
Příklad 7.10 Nakreslíme *plastický graf* příkazem `surf`.

```
surf(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Plasticky graf Rastriginovy funkce');
```



Příklad 7.11 Nakreslíme *plastický graf s isoliniemi* příkazem `surfc`.

```
surfc(X, Y, Z);
xlabel('x1'); ylabel('x2'); zlabel('f(x)');
title('Plasticky graf Rastriginovy funkce s isoliniemi');
```



Shrnutí:

- 3D čárový graf.
- Dopočtení Z-ové složky souřadnic grafu.
- Interaktivní editace a úprava 3D grafu.
- Práce s help pro širší možnosti vytváření 3D grafů.

Kontrolní otázky:

1. Kdy je možné použít vykreslování 3D grafu příkazem `plot3`? Jaké má tento příkaz omezení?
2. Co provádí příkaz meshgrid a kdy jej použijeme?
3. Kdy pro vykreslení funkce použijeme příkaz `plot3` a kdy příkaz `mesh`?
4. Jaké jsou výhody interaktivní editace grafu a oproti příkazové a naopak?
5. Co umožňuje rotace 3D grafu?

8 Práce se soubory

Průvodce studiem:

Cílem této kapitoly je objasnit čtenáři, jak v Matlabu efektivně ukládat a zpětně načítat data. Dále je zde popsán nízkoúrovňový přístup do souboru. Pro pochopení a procvičení tématu počítejte alespoň se třemi hodinami studia.

Matlab umožňuje práci s mnoha typy dat jako jsou číselné vektory a matice, textové soubory, tabulkové soubory, ale i různé grafické objekty, objekty zvuku či dokonce videa. V Matlabu máme řadu možností, jak importovat data ze souborů a exportovat data do souborů různých formátů. Podrobnosti nalezneme v helpu pod hesly `fileformats`, `iofun` a odkazech na další položky. Zde uvedeme jen některé příklady práce se soubory, které nám budou užitečné při práci s Matlabem, při načítání vstupních dat z textových souborů a při ukládání výsledků do textové tabulky se sloupci s konstantní šířkou.

8.1 Ukládání a načítání proměnných, práce s workspace

Nejprve si ukážeme nejjednodušší způsob, jak v Matlabu ukládat proměnné do binárního souboru a zpětně je načíst.

Příklad 8.1 Vytvoříme skalár, vektor ekvidistantně rozložených hodnot a matici 5 × 5, která má na hlavní diagonále hodnoty z rovnoměrného rozdělení a na ostatních pozicích nuly. Skalár a matici uložíme do souboru s názvem 'Ulozeni.mat'.

```
>> a = 1;
b = -0.5:0.01:0.5;
C = rand(5) .* eye(5);

% Nasledujici prikaz ulozi promenne a a C
% do souboru s nazvem Ulozeni.mat
>> save('Ulozeni.mat', 'a', 'C');

% Promenne ulozime stejne prikazem
>> save Ulozeni.mat a C;
```

Uložené proměnné `a` a `C` do binárního souboru lze pak kdykoliv možné do Matlabu načíst.

Příklad 8.2 Načteme proměnné ze souboru s názvem 'Ulozeni.mat' do pracovního prostředí Matlab. Proměnné s jejich podrobnostmi vypíšeme příkazem `whos` na obrazovku.

```
>> load 'Ulozeni.mat';
>> whos -file 'Ulozeni.mat'

  Name      Size                    Bytes  Class

  C         5x5                       200  double array
  a         1x1                         8  double array
```

Příkaz `whos` s parametrem `-file` a `nazev_souboru` vypíše proměnné uložené v souboru *nazev_souboru*.

Příklad 8.3 Matici (6×2) uloženou v souboru, jehož jméno je v řetězci (proměnná) `fdatname` a soubor má následující obsah

```
      2.138E0        1.309E0
      3.421E0        1.471E0
      3.597E0        1.490E0
      4.340E0        1.565E0
      4.882E0        1.611E0
      5.660E0        1.680E0
```

načteme snadno do proměnné (matice) `X` příkazem `dlmread`

```
X = dlmread(fdatname, ' ');
```

Druhý parametr znamená specifikaci oddělovače (delimiter) datových položek, v tomto případě je to mezera (alespoň jedna mezera znamená konec číselné položky, u jiných oddělovačů jako je tabulátor nebo středník to je jinak, tam každý výskyt oddělovače znamená novou položku). Podobně snadno lze načítat data i z jiných formátů, např. tabulky z Excelu příkazem `xlsread` ap.

8.2 Nízkoúrovňový přístup do souboru

Nízkoúrovňový přístup do souboru slouží k nastavení námi určeného vzhledu a formátování souboru. Tento přístup umožňuje kombinovat různé datové typy v jednom

souboru. Příkaz nízkoúrovňového zapisování do souboru má tvar `fprintf(fid, 'formát_proměnné', název_proměnné)`, nejprve je ovšem nutné zajistit otevření existujícího či vytvoření nového souboru příkazem `fopen`, v obecném tvaru

```
fid = fopen('nazev_souboru', 'parametr');
```

kde hodnota *parametru* specifikuje, jakým způsobem má být soubor otevřen:

`'r'`: otevře soubor pro čtení (read)

`'w'`: otevře soubor pro zápis (write) – neexistující se založí, existující se přepíše

`'a'`: otevře soubor pro připsání (append) – neexistující se založí, existující se rozšíří

K zápisu jednotlivých položek proměnné v Matlabu do řádku souboru slouží příkaz `fprintf`. Seznam položek, které mají být zapsány, je předcházen specifikací formátu, poslední příkaz ukončuje řádek ve výstupním souboru. Následující příklady ukazují, jak formátovat jednotlivé typy dat. Jeden specifikátor formátu odpovídá jednomu typu dat, tedy jednomu sloupci dat ve výsledném souboru.

Příklad 8.4 Zápis dat do souboru. Seznam jednotlivých proměnných je přecházen stejným počtem specifikátorů formátu. Do souboru s identifikátorem *fid* zapíšeme řádky s hodnotami proměnnými `mod_name`, `d`, první prvky z vektorů `a` a `b`. Do téhož řádku zapíšeme proměnné alfa a RSS a dále prvních d prvků z vektoru `beta`.

```
fopen('soub1', 'w');
fprintf(fid, ' %-10s %4.0f %11.3e %11.3e', mod_name, d, a(1), b(1));
fprintf(fid, ' %7.2f' ,  alfa );
fprintf(fid, ' %8.0f' , evals);
fprintf(fid, ' %15.7e', RSS);
for i = 1:d
    fprintf(fid, ' %14.5e' , beta(i));
end
fprintf(fid, '%1s\n', ' ');
```

Další záznam (řádek) do souboru pak zapíšeme voláním stejné sekvence příkazů, obvykle je tedy tato sekvence umístěna uvnitř nějakého cyklu. Na závěr soubor zavřeme příkazem `fclose(fid)`.

Příklad 8.5 Vytvoříme a zapíšeme matici čísel 1–15 rozměru 3×5 do textového souboru.

```
% Vytvoreni rady cisel
>> x = 1:15;
% Zmena rozmeru vektoru s cisly na matici
>> A = reshape(x, 3, 5);
A =
     1     4     7    10    13
     2     5     8    11    14
     3     6     9    12    15
```

Matici poté zapíšeme do souboru:

```
>> fid = fopen('matA.txt', 'a');
>> fprintf(fid, '%2d %2d %2d\n', A);
% Prvky matice At jsou cteny po sloupcich a zapisovany
% do jednotlivych radku souboru, viz. nasledujici obrazek
>> fclose(fid);
```

Význam parametru formátování souboru je jednoduchý (%*w.df*):

w – definuje celkový počet znaků,

d – definuje počet desetinných míst,

f – definuje formu výstupu (*f*-float, *d*-decimal, *e*-exponential, *s*-string, *c*-character, aj.).

Specifikaci formátu zápisu do souboru doplňují některé speciální znaky:

`\n` – pro přechod na nový řádek,

`\t` – pro vložení tabulátoru,

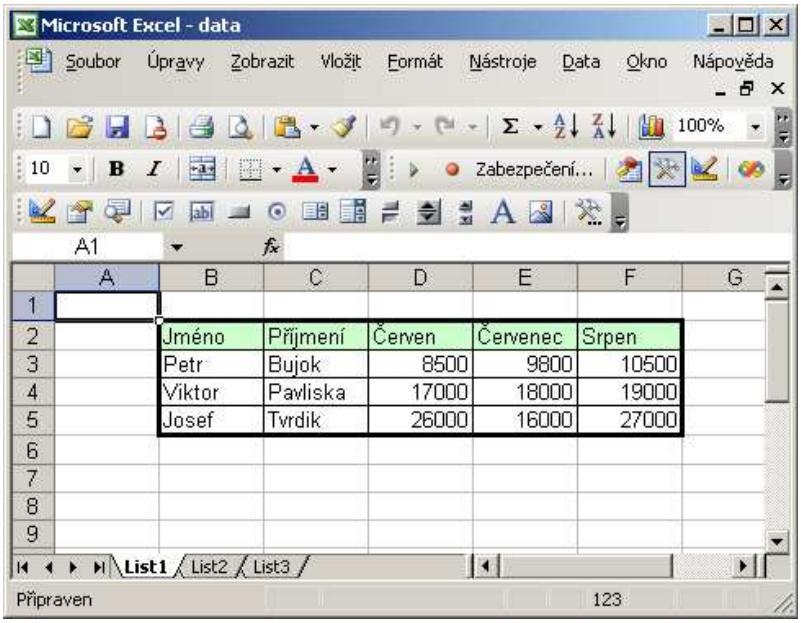
`\'` – pro vložení apostrofu, aj.

Z dalších užitečných funkcí pro efektivní práci se soubory uvedeme: `fpos`, která vrací aktuální pozici indikátoru v daném souboru, a `fseek`, která přesouvá indikátor pozice v daném souboru s ohledem na parametr orientace ('bof', 'cof', 'eof'). Indikátor je ukazatel ("kurzor") aktuální pozice (pro zápis nebo čtení) v souboru.

8.3 Načtení dat z Excelu

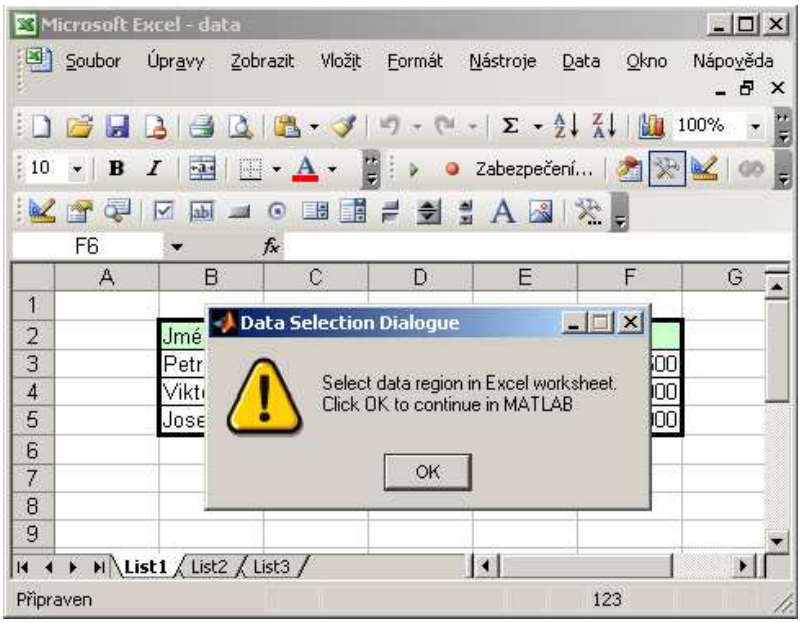
Matlab umožňuje práci i se soubory některých tabulkových procesorů, jako je například Excel. K tomuto v Matlabu slouží příkaz `xlsread`. Načtení dat ze souboru Excelu demonstruje následující příklad.

Příklad 8.6 Načteme číselná a textová data z Excelu do Matlabu tak, aby nedošlo ke ztrátě či znehodnocení dat. Data ze souboru 'Data.xls' načteme do proměnných 'cisla' a 'texty'.



Následující příkaz v Matlabu načte číselné hodnoty do proměnné `cisla` a textové řetězce do proměnné `texty`.

```
>> [cisla,texty] = xlsread('data.xls',-1)
```



Pomocí parametru `-1` příkazu xlsread v datech vybereme interaktivně čísla, které chceme načíst do Matlabu a potvrdíme tlačítkem `OK`. Data jsou načtena do proměnných a vypsána na obrazovku:

```
cisla =
        8500        9800       10500
       17000       18000       19000
       26000       16000       27000

texty =
    'Petr'      'Bujok'
    'Viktor'    'Pavliska'
    'Josef'     'Tvrdik'
```

Další možnosti práce s datovými soubory Excelu nalezne čtenář opět v Helpu.

Uvedené příklady jen ilustrují základní možnosti práce se soubory, o dalších operacích se soubory včetně práce s grafickými formáty se potřebné informace naleznou v helpu.

Shrnutí:

- Matlab umožňuje snadný import a export dat. Mimo to lze přímo za běhu Matlabu ukládat a načítat proměnné, případně celé pracovní prostředí.
- Matlab umožňuje nízkoúrovňový přístup do souboru, který dovoluje podrobnější volby formátování dat.

- Nízkoúrovňový přístup do souboru umožňuje ukládání dat z různých typů do jednoho souboru.

Kontrolní otázky:

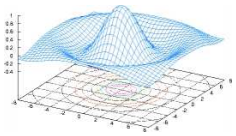
1. Jaká je hlavní výhoda nízkoúrovňového přístupu do souboru?
2. Co se stane, jestliže neuzavřeme soubor po uložení příkazem `fclose`?
3. Co je zapotřebí udělat, abychom byli schopni načíst z Excelu číselné i textové hodnoty?

9 Alternativy MATLABU

Průvodce studiem:

Cílem této kapitoly je seznámit čtenáře s jinými softwarovými alternativami systému Matlab, které jsou s ním více-méně kompatibilní, jednak co se týče syntaxe zápisu a do značné míry také souborem vestavěných funkcí. Tato kapitola je spíše přehledová a jejím hlavním účelem je poskytnout odkazy na nejčastěji používané programy, které jsou poskytovány pod různými, ale vesměs svobodnými licencemi. Čas věnovaný této kapitole záleží především na zájmu čtenáře, nakolik se bude chtít věnovat uvedeným odkazům a může se pohybovat v rozmezí od půl hodiny až po několik večerů strávených brouzdáním po internetu.

9.1 Octave



Octave je matematický program, který je v rámci open source alternativ s programem Matlab asi nejvíce kompatibilní. Octave nebyl vůbec původně zamýšlen jako program, který by mohl nahrazovat komerční program Matlab, ale byl navržen jako uživatelsky příjemný program pro psaní vysokoškolské učebnice týkající se návrhu chemických reaktorů. Ačkoli byl takto plánován, byl používán v mnoha dalších vysokoškolských a absolventských kurzech chemického inženýrství na univerzitě v Texasu. Matematické oddělení této univerzity jej používalo také k výuce diferenciálních rovnic a lineární algebry.

Rozšíření Octave

Matlabu znalý uživatel velice záhy u Octave zjistí, že řada funkcí jemu důvěrně známých zde prostě chybí. Největší absenční známky základní instalace Octave se snaží napravit repozitář funkcí *Octave-forge* sídlící na adrese `http://octave.sourceforge.net`, které lze doinstalovat k již instalovanému základnímu Octave. V praxi to funguje tak, že si uživatel stáhne celý balík funkcí včetně instalačních skriptů, v průběhu instalace si volitelně zvolí, o které funkce má zájem, následně kompiluje, a pokud se vše zdaří, může si užívat větší kompatibility s Matlabem. Konkrétně si tak uživatel může dopomoci například k sadám funkcí z oblasti:

- ekonometrie,
- paralelní zpracování dat,
- třírozměrná vizualizace (VRML),
- spousta nových typů grafů,
- symbolické výpočty.

QtOctave

QtOctave je grafický frontend pro konzolovou aplikaci GNU/Octave, napsaný v Qt4. Svým rozvržením a funkcemi se snaží vyrovnat proprietárnímu programu Matlab. Kromě grafického frontendu pro GNU/Octave nabízí též integrovaný editor skriptů.

V současné době jsou dostupné verze pro GNU/Linux a MS Windows.

Odkazy

- Oficiální stránky Octave: http://www.octave.org
- Doplnujňující balíčky pro Octave: http://octave.sourceforge.net/index.html
- Český průvodce programem: http://www.octave.cz
- 3D vizualizace do Octave: http://octaviz.sourceforge.net
- Grafický systém do Octave: http://octplot.sourceforge.net
- Uživatelský frontend (QtOctave): http://qtoctave.wordpress.com
- Souhrn rozdílů mezi Octave a Matlabem: http://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB a http://wiki.octave.org/wiki.pl?MatlabOctaveCompatibility
- Seriál o Octave na serveru AbcLinuxu: http://www.abclinuxu.cz/serialy/octave

9.2 FreeMat

Jedná se o projekt, za kterým stojí Samit Basu, ale dnes již velkou část vývoje provádí komunita. Pokud jde o kompatibilitu s Matlabem, pak stránky produktu uvádějí přibližně 95% kompatibilitu a na dalších vylepšeních se pracuje. Naopak oproti Matlabu není možno vytvářet grafická uživatelská rozhraní k aplikacím, ale některé partikulární části jsou postupně zapracovávány.

Silnou stránkou aplikace je možnost provádět paralelní výpočty skrze MPI, což výrazně rozšiřuje škálu funkcí a možností tohoto nástroje. Nechybí ani schopnosti 3D vizualizací, což je právě jedna z domén Matlabu. Pozitivem je také to, že k dispozici jak pro Linux, tak také pro Mac OS i Windows.

Celá skupina funkcí je věnována například grafickým prostředkům analýzy dat. Na základě zadaných dat zvládne vygenerovat přímé proložení nejrůznějšími funkcemi. Nechybí ani nástroje pro interpolaci či extrapolaci dat, proložení Gaussovou křivkou

či libovolným polynomem. Pokud jde o řešení diferenciálních rovnic je možné užít i numerické metody, což je právě pro oblast inženýrství zásadním způsobem důležité. Program si bez potíží poradí i s řídkými maticemi, což je další, ne zcela triviální dovednost. Podobně může být užitečná i podpora práce s více než 2GB poli (pro 64bitové operační systémy).

Odkazy

- Domovská stránka: http://freemat.sourceforge.net/

9.3 Scilab

SciLab = Scientific Laboratory (vědecká laboratoř). Scilab je vědecký softwarový balík pro numerické výpočty. Poskytuje otevřené programovací prostředí pro inženýrské a vědecké aplikace.

Scilab byl vyvíjen od roku 1990 výzkumnými institucemi INRIA (Institut National de Recherche en Informatique et en Automatique) a ENPC (École Nationale des Ponts et Chaussées). Od roku 1994 byl distribuován jako freeware i se zdrojovým kódem. V současné době je používán pro výukové a průmyslové prostředí po celém světě. Scilab nyní spravuje Scilab Consorcium, založené v květnu 2003.

Scilab obsahuje stovky matematických funkcí s možností přidat další programy z různých programovacích jazyků (FORTRAN, C, C++, JAVA...). Má propracovanou strukturu dat, překladač, a dovoluje používat vyšší programovací jazyky. Dále obsahuje Scicos, což je něco jako Simulink pro Matlab, a nakonec ještě překladač Matlab to Scilab.

Toolboxy

Toolboxy (v rámci SciLabu se jim říká atomy) jsou souborem funkcí, které rozšiřují základní sadu funkcí Scilabu. Mohou být buď komerční a nebo, což je mnohem častější případ, open source či volně šiřitelné. Autory těchto toolboxů jsou buď samotní autoři Scilabu nebo jiní programátoři, často z univerzitního prostředí.

Určité množství toolboxů je dodáváno se Scilabem a lze si při instalaci zvolit, zda budou nainstalovány. Základní sadu můžeme rozšířit doinstalováním nových toolboxů. Největší databází volně šiřitelných toolboxů je na stránkách autoru Scilabu v tzv. toolboxes center. Zde jsou toolboxy řazeny do kategorií, takže v nich je možno přehledně vyhledávat. Alespoň namátkou můžeme vyjmenovat:

- neuronové sítě,

- data mining,
- metody konečných prvků.
- genetické algoritmy
- statistika
- 2-D a 3-D grafika,
- animace
- simulace
- klasické a robustní řízení,
- řízení signálů
- grafy
- paralelní Scilab využívající PVM
- prostředí počítačové algebry (Maple, MuPAD)

Scilab dokáže pracovat pod operačními systémy Windows 9X/2000/XP, GNU/Linux a UNIX. Zdrojové kódy jsou volně ke stažení na domovské stránce http://www.scilab.org.

Odkazy

- Domovská stránka: http://www.scilab.org
- Adresa ke stažení: http://www.scilab.org/products/scilab/download
- Rozšíření SciLabu: http://atoms.scilab.org/
- Elektronická kniha *Basics on digital controller with scilab/scicos* volně ke stažení: http://www.ebookaktiv.com/ebookcontroller/eBook_SCILAB.htm
- Stručný manuál a úvod do Scilabu v češtině: http://scilab.ic.cz

10 Řešené úlohy

Průvodce studiem:

Poslední kapitola tohoto textu je věnována příkladům řešení úloh v Matlabu. Věnujte dostatečný čas důkladnému pochopení formulace úlohy i způsobu implementace v Matlabu. Možná naleznete i jiný způsob implementace, třeba i jednodušší a vhodnější. Odhadovaný čas potřebný ke studiu je 3 až 6 hodin.

10.1 Náhodná procházka

Představte si, že opilý námořník v přístavu má nastoupit na loď. K tomu potřebuje přejít nástupní molo k lodi. K jeho přejití potřebuje udělat určitý počet kroků dopředu. Jelikož je opilý, ne každý krok, který se mu podaří, je dopředu, někdy se vydá doleva nebo doprava. Úskalí je v tom, že pokud vykoná příliš mnoho kroků do strany, spadne přes okraj mola do vody. Do odplutí lodi stihne námořník udělat nejvýše zadaný maximální počet kroků.

Nyní si naprogramujeme model námořníkovy cesty po molu. Musíme nejdříve přesněji zformulovat pravidla pohybu námořníka:

- krok vpřed s pravděpodobností 0.6
- krok vpravo s pravděpodobností 0.2
- krok vlevo s pravděpodobností 0.2

Uvedenými hodnotami pravděpodobnosti modelujeme úroveň námořníkovy opilosti.

Dále musíme znát další vstupní parametry námořníkova pohybu:

- Délka mola je 50 kroků.
- Námořník vychází z pozice na počátku mola, z níž k levému i pravému okraji má 10 kroků.
- Do odplutí lodi stihne nejvýše 100 kroků.

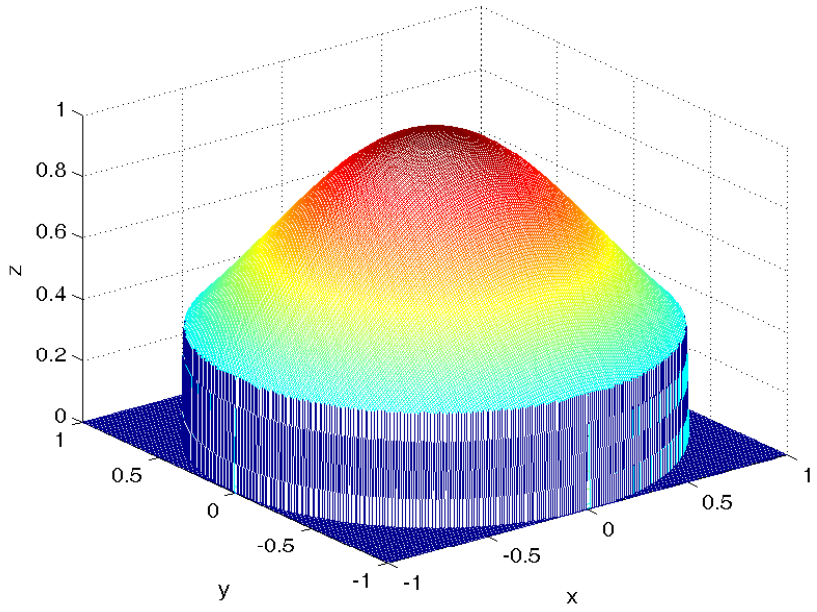
Pak už můžeme napsat program modelující námořníkovu cestu:

```
N = input('Pocet pokusu: ');
dosel = 0; nestih = 0; voda = 0;
for i = 1:N
    kroky = 0;          % Vynulovani kroku pro novou prochazku
    x = 0;              % Souradnice pocatku prochazky
```

```
    y = 0;
    while x < 50 && abs(y) <= 10 && kroky < 100
        kroky = kroky + 1; % Dalsi krok
        r = rand;
        if r < 0.6    % rozhodovani kam
            x = x + 1;     % krok dopredu
        elseif r  <0.8
            y = y + 1;     % krok vpravo
        else
            y = y - 1;     % krok vlevo
        end;
    end;
    if x >= 50
        dosel = dosel + 1;
    end;
    if abs(y) > 10
        voda = voda + 1;
    end
    if kroky >= 100
        nestih = nestih + 1;
    end
end;
dosel = 100 * dosel / N;
disp('Celkova uspesnost (%): ')
disp(dosel)
voda = 100 * voda / N;
nestih = 100 * nestih / N;
disp('Voda (%): ')
disp(voda)
disp('Nestihnul (%): ')
disp(nestih)
```

10.2 Objem tělesa metodou Monte Carlo

Máme určit objem tělesa (tvarem připomínajícím bábovku) na následujícím obrázku:



Toto těleso je průnikem tělesa pod plochou grafu funkce

$$z = \exp(-x^2 - y^2) \tag{1}$$

a válce o poloměru 1, výšce 1 a středem podstavy v bodě $(0, 0, 0)$. Povšimněme si, že funkce (1) má největší hodnotu rovnou jedné v bodě $(0, 0)$ a její vodorovné řezy jsou kružnice. Zdrojový kód této funkce upravený pro nakreslení grafu tělesa na obrázku následuje:

```
function z = babovka(x, y, r)
if (x^2 + y^2) <= r^2
    z = exp(-x^2 - y^2);
else
    z = 0;
end
```

Objem tohoto tělesa v tomto příkladu můžeme určovat různým způsobem, např. jako součet objemu jeho válcové části a objemu zakulaceného vršku, který bychom snadno určili jednoduchou numerickou metodou:

```
h = 0.001
x = 0: h: 1;
d = length(x);
z = exp(-x.^2);
for ii = 1:d-1
    prumer(ii) = (x(ii) + x(ii + 1)) /2;
    V(ii) = pi * prumer(ii)^2 * (z(ii) - z(ii + 1));
end
V = sum(V);
V = V + pi * exp(-1)
```

Takto spočítáme, že objem tělesa je 1.9859.

Jsou úlohy, kdy vhodná numerická metoda není k dispozici. Proto si zde ukážeme řešení naší úlohy stochastickou metodou Monte Carlo. Její princip spočívá v tom, že náhodně vygenerujeme velké množství bodů rozdělených rovnoměrně ve větším tělese, jehož objem umíme snadno určit. Zjistíme relativní četnost bodů, které jsou současně i uvnitř tělesa, jehož objem zjišťujeme, a pak hledaný objem spočítáme jako součin relativní četnosti bodů uvnitř a objemu většího tělesa. Hledaný objem tělesa lze tedy vypočítat takto:

$$V_t = V_v \, \frac{n_t}{n_v},$$

kde V_v je objem většího tělesa, n_v je počet bodů vygenerovaných ve větším tělese a n_t počet bodů v tělese, jehož objem zjišťujeme. V naší úloze je větším tělesem válec o objemu $V_v = l \, \pi \, r^2$, po dosazení $l = 1, \; r = 1$ dostaneme $V_v = \pi$.

Skript řešící úlohu pro těleso definované na začátku této části textu následuje. Všimněme si, že vzhledem k symetrii tělesa stačí zjišťovat četnost bodů jen pro jeden kvadrant $(x \geq 0, y \geq 0)$.

```
r = 1;          %  polomer valce
nvalec = 0;
nbabovka = 0;
for ii = 1:100000
    x = r * rand(1);
    y = r * rand(1);
    if (x^2 + y^2) <= r^2     % je v kruhu o polomeru r
        nvalec = nvalec + 1;  % pocet bodu uvnitr valce
        v = rand(1);          % svisla souradnice bodu
        z = babovka(x, y, r);
        if v <= z             % vygenerovany bod je v babovce
            nbabovka = nbabovka + 1;
        end
```

```
    end
end
nvalec
objem_valec = pi * r^2;
objem_babovka = objem_valec * nbabovka / nvalec
```

Výsledky z 5 opakovaných spuštění skriptu jsou v následující tabulce. Zjištěný objem tělesa je přibližně 1.99.

	Počet bodů	Objem
	78496	1.9900
	78610	1.9887
	78477	1.9894
	78694	1.9823
	78409	1.9919
Průměr	78537	1.9885

Průměrný počet bodů uvnitř válce o průměru 1 je 78537 ze 100000 bodů náhodně vygenerovaných ve čtverci o straně 1. Tedy poměr těchto hodnot by měl být přibližně roven $\pi/4$. Čtyřnásobek tohoto poměru je 3.141488 a je opravdu dosti přesně roven π.

Shrnutí:

- Pečlivá a jasná formulace řešeného problému, vstupní a výstupní data.
- Volba co nejjednoduššího řešení a vhodné implementace.

Kontrolní otázky:

1. Jak byste modelovali menší a větší opilost námořníka v první úloze? Upravte skript a vyzkoušejte
2. Jak byste spočítali objem tělesa, které vznikne průnikem tělesa pod plochou grafu funkce (1) a kvádru o rozměrech $a \times b \times 1$ s podstavou $a \times b$ v rovině $z = 0$, s hranami rovnoběžnými s osami a zadanou polohou vrcholu nejbližšímu počátku $(0, 0)$? Upravte skript k řešení této úlohy a ověřte na počítači pro $a = 1$, $b = 0.5$ a polohu vrcholu nejbližšímu počátku $(0.3, 0.3)$.

Literatura

[1] Manuál Matlab, součást instalovaného softwarového prostředí.

[2] Büllow J.: Sbírka jednoduchých příkladů pro řešení elektrotechnických a fyzikálních úloh, Západočeská univerzita v Plzni, 2007.

[3] Engelbrecht A. P.: Computatinal Intelligence: An Introduction. Chichester: John Wiley & Sons, 2007.

[4] Hahn B., Valentine D.: Essential MATLAB for Engineers and Scientists, Elsevier, 2007.

[5] Hanselman D., Littlefield B.: Mastering MATLAB 7, Pearson Prentice Hall, 2005.

[6] Heringová B., Hora P.: MATLAB Díl I. – Práce s programem, 1995, http://www.cdm.cas.cz/czech/hora/vyuka/mvs/tutorial.pdf.

[7] Heringová B., Hora P.: MATLAB Díl II. – Popis funkcí, 1995, http://www.cdm.cas.cz/czech/hora/vyuka/mvs/reference.pdf.

[8] Karban P.: Výpočty a simulace v programech Matlab a Simulink, Computer Press, 2006.

[9] Knight A.: Basics of Matlab and Beyond, Chapman and Hall, 2000.

[10] Majerová D.: Lekce Matlabu v češtině: http://uprt.vscht.cz/majerova/matlab/index.html.

[11] Poláček J.: Seriál o Octave na serveru AbcLinuxu: http://www.abclinuxu.cz/serialy/octave.