Obsah

Část I

Jak pracovat s MATLABem

Úvod

MATLAB (z anglického MATrix LABoratory) je interaktivní programovací prostředí a skriptovací programovací jazyk pro operační systémy Windows, Linux i MacOS. Jedná se o systém vhodný pro vědecké a inženýrské výpočty, modelování, simulace, analýzu a vizualizaci dat, vývoj algoritmů a aplikací včetně tvorby grafického uživatelského rozhraní. Velké uplatnění má zejména v technických oborech a ekonomii. Možnosti MATLABu rozšiřují knihovny funkcí, tzv. toolboxy, soubory M-funkcí zaměřené na specifické účely (statistika, optimalizace, symbolické výpočty, neuronové sítě, zpracování signálů a obrazu, apod.).
Autor MATLABu, firma The MathWorks, Inc. (`http://www.mathworks.com/`), stejně jako její zástupce pro Českou republiku a Slovensko, firma HUMUSOFT (`http://www.humusoft.cz/`), poskytují svému prostředí značnou podporu. Uživatelé na výše zmíněných webových stránkách mohou získat informace o instalaci, nových produktech a rozšířeních, nabízí kurzy, školení apod. Licenční a instalační soubory jsou pro studenty a zaměstnance Masarykovy univerzity k dispozici na Intranetovém serveru MU. K systému MATLAB existují i volně šiřitelné alternativy jako Octave (`www.octave.org`), Scilab (`www.scilab.org`) nebo FreeMat (`http://freemat.sourceforge.net/`), ty ovšem nedosahují takové kvality jako MATLAB.

Tento studijní text je určen především pro studenty předmětů "M4130 a M4130c Výpočetní matematické systémy", předpokládá pouze základní znalosti lineární algebry, práce na počítači a základy programování. Cílem textu je seznámit studenty se syntaxí MATLABu, maticí jako základním stavebním prvkem tohoto prostředí a maticovým uvažováním, závěr textu je věnován grafickému zobrazení dat a tvorbě vlastních skriptů a funkcí. Text je doplněn o množství jak demonstračních příkladů usnadňující pochopení příkazů, tak příkladů určených pro samostatné řešení.
Návod je psán pro verzi MATLABu 8.3. Přestože MATLAB pracuje od verze 6 s okenní strukturou, příkazová řádka i nadále zůstává základním komunikačním prostředkem a většina příkazů by měla být funkční i ve verzích nižších.

Jan Koláček, Kateřina Konečná

Kapitola 1

Začínáme s MATLABem

Základní informace

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

Výstupy z výuky

Studenti

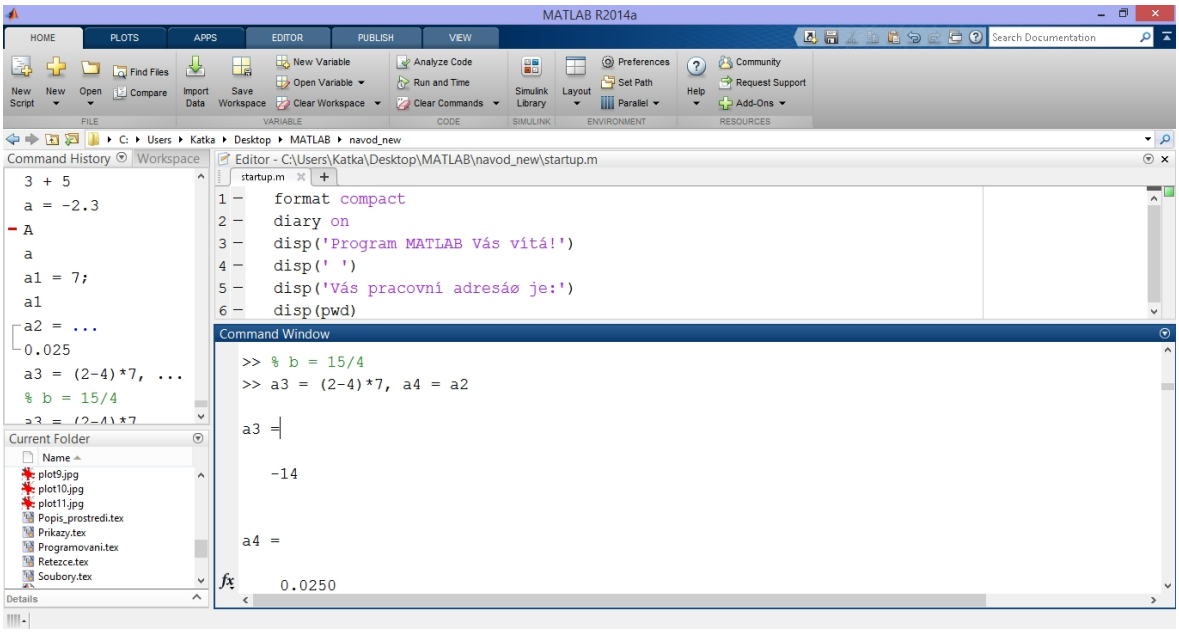
- se seznámí s okenní strukturou MATLABu a jejích využitím
- umí využít příkazové řádky
- ovládají základní syntaxi jazyka
- umí využívat nápovědního systému

1.1 Popis prostředí

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

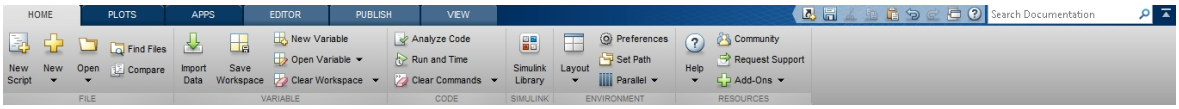
Základem grafického rozhraní MATLABu je *Command Window* (příkazové okno), dále *Current folder* (okno pro správu aktuální složky), *Workspace* (pracovní prostor) a *Command history* (okno historie příkazů). Jednotlivá okna lze uspořádat podle vlastního uvážení.

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



Obr. 1.1. Okenní struktura MATLABu.

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



Obr. 1.2. Nástrojová lišta s ikonami.

Záložka *Home* usnadňuje práci se soubory (skupina ikon *File*), obsahuje ikonky pro pohodlné otvírání již existujících souborů, vytváření nových souborů a jejich ukládání, import dat (*Import Data*), správu proměnných (skupina ikon *Variable*), dále ikonky vlastního uspořádání okenní struktury (*Layout*), nastavení vlastností (*Preferences*) či cesty k adresářům (*Set Path*).
Záložka *Plots* umožňuje grafické znázornění zvolených proměnných, záložka *Apps* obsahuje klikací prostředí pro prokládání křivek daty, optimalizaci, analýzu signálu apod. V případech, kdy je kód upravován (otevřen v okně *Editor*), se zobrazí další tři nové záložky: *Editor*, *Publish* a *View*. Tyto záložky obsahují funkce právě pro práci a úpravu kódu. Záložka *Editor* usnadňuje práci s kódy. Skupina ikon *File* umožňuje otevírání a ukládání existujících kódů, tvorbu nových kódů, prohledávání a porovnávání souborů. Skupina *Edit* obsahuje ikony pro vkládání předdefinovaných funkcí a komentářů, odsazování textu, skupina *Navigate* obsahuje ikony pro pohyb v souboru, *Breakpoints* ikony pro práci s 'breakpoints' a *Run* ikony pro spouštění kódu.
Záložka *Publish* obsahuje ikony pro grafickou úpravu kódu a *View* ikony pro členění okna editoru a vlastního kódu.

1.2 Základní ovládání

Jak již bylo řečeno v odstavci výše, MATLAB spustíme příkazem `matlab` v příkazové řádce nebo poklikáním na odpovídající ikonu. Program se ukončuje příkazem `exit` nebo zavřením okna, v němž program běží.
Po spuštění programu prostředí dominuje *Command window* s příkazovou řádkou, do které jsou zadávány veškeré příkazy. Příkazy je možné zadávat, pokud je aktivní prompt `>>` na začátku řádku, příkazy spouštíme klávesou ENTER. Po spuštění příkazu se na následujících řádcích objevují výstupy.

```
>> 3 + 5    příkaz s výpisem výstupní hodnoty, výstup je dočasně uchován v proměnné
ans. Při spuštění dalšího příkazu, jehož výstup není přiřazen do žádné proměnné, je
hodnota proměnné ans přepsána aktuální hodnotou
ans =
     8
```

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

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

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

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

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

Provedené příkazy nelze upravit přímo, úprava je možná jen editací v dalším příkazu. K listování v historii příkazů slouží klávesy ↑ a ↓. Klávesy → a ← slouží k pohybu kurzoru v aktuální příkazové řádce. Při použití části příkazu následovaného klávesou

↑ budou nabízeny především příkazy se shodným začátkem.
Pro přerušení výpočtu prováděného příkazu lze použít klávesovou zkratku CTRL+C, smazání příkazového řádku se provádí klávesou ESC.
Symbol `%` se používá jako komentář, veškerý text za tímto znakem není vyhodnocován.

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

1.3 Nápověda

Velmi důležitou součástí softwaru MATLAB je nápovědní systém, který obsahuje kompletní dokumentaci k celým knihovnám funkcí.
Jedním ze základních příkazů pro vyvolání nápovědy je samostatný příkaz `help`, který v *Command Window* zobrazuje všechny základní tématické oblasti spolu s cestou a jednoduchým popisem.

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

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

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

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

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

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

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

```
>> help sin   nápověda k funkci sin (sinus)
sin - Sine of argument in radians

This MATLAB function returns the sine of the elements of X.

Y = sin(X)

Reference page for sin

See also asin, asind, sind, sinh

Other functions named sin
fixedpoint/sin, symbolic/sin
```

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

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

```
>> doc sin
```

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

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

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

Příklady k procvičení

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

Řešení.

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

Kapitola 2

Jednoduché výpočty, proměnné a matice

Základní informace

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

Výstupy z výuky

Studenti

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

2.1 MATLAB jako kalkulátor

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

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

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

```
>> pi   Ludolfovo číslo
ans =
     3.1416
>> 3*(15.8+3^8) - sin(pi/12) + sqrt(4.8)
ans =
     1.9732e+04   výstup značí hodnotu 1.9732 * 10^4
>> abs(sin(3*pi/2))
ans =
     1
```

Pro celočíselné zaokrouhlování se používají následující funkce:

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

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

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

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

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

2.2 Proměnné, matice a jejich definování

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

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

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

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

Prázdnou matici lze vytvořit příkazem `[]`.

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

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

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

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

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

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

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

2.3 Funkce pro tvorbu matic

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

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

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

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

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

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

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

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

Další alternativou pro generování ekvidistantních vektorů jsou příkazy `linspace()` a `logspace()`:
`x = linspace(a, b, n)` generuje aritmetickou posloupnost prvků `x` od `a` do `b`. Třetí parametr `n` je volitelný, udává počet prvků posloupnosti `x`, jeho implicitní nastavení na `n=100` lze libovolně měnit.
`x = logspace(a, b, n)` generuje vektor `x` s desítkovou logaritmickou škálou prvků od `10^a` do `10^b`. Třetí parametr délky posloupnosti `n` je volitelný, jeho implicitní nastavení je `n=50`. Příkaz je ekvivalentní příkazu `10.^linspace(a, b, n)`.

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

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

2.4 Některé speciální výrazy a funkce

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

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

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

Způsob výpisu hodnot můžeme ovlivnit příkazem `format`:

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

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

```
>> format rat
>> pi
ans =
     355/113
>> abs(1+i)    absolutní hodnota komplexního čísla a+i·b je definována jako √(a²+b²)
ans =
     1.4142
```

2.5 Obecná pravidla pro příkazy

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

`příkaz1, příkaz2, příkaz3,` atd.,

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

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

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

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

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

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

Příklady k procvičení

1. Spočítejte velikost úhlu α při vrcholu A trojúhelníku ABC, znáte-li délku protilehlé strany $a = 5$ cm a přilehlé strany $b = 3$ cm.

2. Vypište hodnotu Ludolfova čísla, pak:
   a) jej zaokrouhlete k nejbližšímu celému číslu,
   b) jej zaokrouhlete k plus nekonečnu,
   c) jej zobrazte jako zlomek,
   d) jej zobrazte na plný počet desetinných míst.

3. Třemi způsoby vygenerujte vektor `u1` délky 10 s počáteční hodnotou 2 a krokem 0.2.

4. Vygenerujte náhodný vektor délky 5
   a) s názvem `u2` a prvky z intervalu $[0, 1]$,
   b) s názvem `u3` a prvky z intervalu $[-2, 4]$,
   c) s názvem `u4` a celočíselnými prvky z intervalu $[-8, 1]$.

5. Vygenerujte matici s rozměry 5x3
   a) s názvem `A1` a prvky z intervalu $[-12, 5]$,
   b) s názvem `A2` a celočíselnými prvky z intervalu $[1, 7]$.

6. Vytvořte matici `B` rozměrů 4x5 sestavenou z vektoru `u4` a matice `A2`.

7. Vytvořte matici rozměrů 4x9
   a) s názvem `C1`, maticí `B` v levém bloku a jednotkovou maticí v pravém bloku,
   b) s názvem `C2`, maticí `B` v levém bloku a maticí obsahující hodnoty 0.5 v pravém bloku.

Řešení.

1.
```
a = 5, b = 3
alpha = tan(a/b)
```

2. `exp(1)`
   a) `round(exp(1))`
   b) `ceiling(exp(1))`
   c) `format rat, exp(1)`
   d) `format long, exp(1)`

3. 1. způsob: `u1 = 2:0.2:3.8`,
   2. způsob: `u1 = colon(2, 0.2, 3.8)`,
   3. způsob: `u1 = linspace(2, 3.8, 10)`

4. a) `u2 = rand(1, 5)`
   b) `u3 = 6*rand(1, 5) - 2`
   c) `u4 = round(9*rand(1, 5) - 8)`

5. a) `A1 = rand(17*rand(5, 3) - 12)`
   b) `A2 = round(6*rand(5, 3)+1)`

6. `B = [u4; A2']`

7. a) `C1 = [B, eye(4)]`
   b) `C2 = [B, 0.5 + zeros(4)]` nebo
   `C2 = [B, 0.5 + zeros(4, 4)]` nebo
   `C2 = [B, 0.5 + ones(4)]` nebo
   `C2 = [B, 0.5 + ones(4, 4)]`

Kapitola 3

Maticové operace

Základní informace

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

Výstupy z výuky

Studenti

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

3.1 Transpozice matic

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

získáme matici `X`, která vznikne transpozicí matice `A`, ale také její prvky budou komplexně sdružené k prvkům matice `A`, tj. $x_{ij} = \overline{a_{ji}}$.
Pokud bychom chtěli získat pouze transponovanou matici `X` bez komplexně sdružených prvků, tj. $x_{ij} = a_{ji}$, je třeba použít jednoduchý apostrof s tečkou (`.'`).

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

3.2 Sčítání a odčítání matic

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

```
>> A = [1 5; 2 3];
>> B = [0 1; -3 2];
>> X = A + B
X =
       1  6
      -1  5
>> c = 2;
>> X = A + c   odpovídá x_ij = a_ij + c
X =
      3  7
      4  5
>> X = c - B   odpovídá x_ij = c - b_ij
X =
      2  1
      5  0
```

3.3 Maticové násobení

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

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

3.4 Maticové dělení

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

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

Je-li `A` regulární čtvercová matice, potom `X = A\B`, resp. `X = B/A`, formálně odpovídají levostrannému, resp. pravostrannému, násobení matice `B` maticí inverzní k matici `A`; tj. `inv(A)*B` resp. `B*inv(A)`. Funkce `inv()` slouží pro výpočet inverzní matice, výpočet pomocí levostranného, resp. pravostranného, dělení je získán přímo, bez výpočtu inverze.
Je-li matice `A` obecně typu $\underline{k} \times n$, `B` musí být typu $\underline{k} \times m$, výsledná matice `X = A\B` bude typu $n \times m$. Pravostranné dělení `X = B/A` je definováno pomocí levostranného dělení jako `X = B/A`, což je ekvivalentní `(A'\B')'`.
Při dělení matice konstantou se dělení provádí po složkách, výsledek pravostranného i levostranného dělení je stejný.

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

3.5 Umocňování matic

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

1. `X = A^p`
   (a) $p > 0$ celé číslo $\Rightarrow X = \underbrace{A * A * \cdots * A}_{p}$
   (b) $p = 0 \Rightarrow X = I$ ... jednotková matice (lze vytvořit příkazem `eye(size(A))`)
   (c) $p < 0$ celé číslo $\Rightarrow X = \underbrace{A^{-1} * A^{-1} * \cdots * A^{-1}}_{p}$
   (d) p není celé číslo $\Rightarrow$ uvažujme diagonální matici $D = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$, kde $\lambda_1, \ldots, \lambda_n$ jsou vlastní čísla matice A, a matici V, jejíž sloupce jsou vlastní vektory příslušné postupně těmto vlastním číslům (`[V,D] = eig(A)`). Platí tedy vztah $A*V = V*D$, což je v tomto případě jakási motivace pro výpočet mocniny. Lze ukázat, že platí $A^p * V = V * D^p$, kde $D^p = \begin{pmatrix} \lambda_1^p & & 0 \\ & \ddots & \\ 0 & & \lambda_n^p \end{pmatrix}$. Odtud dostáváme vztah pro výpočet $X = V * D^p/V$.
2. `X = p^A`
   Při výpočtu opět vycházíme ze vztahu $A * V = V * D$, kde V a D jsou výše popsané matice. Odtud je $p^A * V = V * p^D$, kde $p^D = \begin{pmatrix} p^{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & p^{\lambda_n} \end{pmatrix}$. Odtud opět dostáváme vztah pro výpočet $X = V * p^D/V$.

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

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

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

3.6 Operace po složkách

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

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

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

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

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

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

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

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

3.7 Logické operace

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

3.8 Smíšené operace

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

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

Prioritu výše uvedených operátorů můžeme řídit pomocí kulatých závorek.
Všechny operátory pro práci s maticemi uvedené v této kapitole mají své ekvivalenty:

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

Příklady k procvičení

1. Jsou dány matice `A = [1 0 -1; 2 1 1]`, `B = [2 -2 1; 0 1 2]`, `C = [1 0 1; 0 1 1; 1 0 0]` a `D = [3 5 -6; 1 1 -2; 0 -2 0]`. Vyřešte následující rovnice:
   a) `A + X1 = B`,
   b) `(A - X2) * C = B`,
   c) `A'*(X3 - B) = D`.

2. Vytvořte komplexní matici `F`, jejíž reálná část bude tvořena maticí `X1` a imaginární část maticí `X2`.
   a) Vytvořte k ní transponovanou matici `F1`.
   b) Vytvořte k ní transponovanou matici `F2`, která bude navíc obsahovat komplexně sdružené prvky.

3. Je dána matice `E = [1 2; -1 1]`.
   a) Vytvořte matici `E1`, která bude druhou mocninou matice `E`.
   b) Vytvořte matici `E2`, která bude obsahovat druhé mocniny prvků matice `E`.
   c) Vytvořte matici `E3` funkčních hodnot funkce kotangens v bodech `E`.

Řešení.

1. `A = [1 0 -1; 2 1 1], B = [2 -2 1; 0 1 2], C = [1 0 1; 0 1 1; 1 0 0], D = [3 5 -6; 1 1 -2; 0 -2 0]`
   a) `X1 = B - A`
   b) `X2 = A - B/C`
   c) `X3 = A'\ D+B`

2. `F = X1 + i*X2`
   a) `F1 = F.'`
   b) `F2 = F'`

3. `E = [1 2; -1 1]`
   a) `E1 = E^2`
   b) `E2 = E.^2`
   c) `E3 = cot(E)`

Kapitola 4

Manipulace s maticemi

Základní informace

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

Výstupy z výuky

Studenti

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

4.1 Základní manipulace s maticemi

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

rozměr.

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

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

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

```
>> A([1 3],[4 2]) = eye(2)
```
do takto vybraných prvků je možno také při zachování správných rozměrů provádět přiřazení

```
A =
      1.0000       0   2.0000  1.0000
      3.0000       0   4.0000  5.0000
      3.2000  1.0000  -6.0000  0.0000
```

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

```
>> u1 = A(:, 3)
```
výběr 3. sloupce matice `A`

```
u1 =
       2
       4
      -6
>> u2 = A(2,:)
```
výběr 2. řádku matice `A`

```
u2 =
     3   0   4   5
>> A(2,:) = []
```
smazání 2. řádku matice `A`

```
A =
      1.0000       0   2.0000  1.0000
      3.2000  1.0000  -6.0000  0.0000
```

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

Matice stejných rozměrů lze sčítat a odčítat (operátory `+`, `-`), při přičtení konstanty k matici je konstanta přičtena ke každému prvku matice.
Matice vhodných rozměrů lze násobit (operátor `*`), při násobení matice konstantou je konstantou přenásoben každý prvek matice. Dělení je v MATLABu dvojího druhu – pravostranné a levostranné: \ a `/`.
Všechny operátory jsou podrobněji popsány v Kapitole 3.

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

4.2 Změna struktury matice

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

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

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

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

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

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

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

sloupcového vektoru.

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

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

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

4.3 Základní funkce lineární algebry

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

Zde vyjmenujeme jen některé základní:

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

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

Syntaxi jednotlivých funkcí zjistíme nejlépe pomocí nápovědy.

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

4.4 Další funkce pro manipulaci s maticemi

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Příklady k procvičení

1. Vygenerujte celočíselnou náhodnou matici `A` rozměrů 5x6 s prvky z intervalu $[-7, 7]$.
   a) Vytvořte matici `A1`, která bude obsahovat sudé sloupce matice `A`.
   b) Zjistěte rozměry matice `A1`.
   c) Z matice `A1` vytvořte matici `A2`, která bude obsahovat mít vyměněný druhý a třetí sloupec.
   d) Dvěma způsoby vytvořte z matice `A` matici `A3`, která bude mít opačné pořadí řádků.
   e) Spočítejte součet řádkových maxim matice `A`.
   f) Vytvořte matici `A4`, která bude obsahovat pouze diagonální prvky matice `A`.
   g) Vytvořte matici `A5`, která bude obsahovat stejné prvky jako matice `A` a pod diagonálou hodnoty 3.
   h) Vytvořte čtvercovou matici `A6`, která bude obsahovat matici `A` a na posledním řádku její sloupcové součiny.

2. Vytvořte matici `B` rozměrů 10x10, která bude po řádcích obsahovat posloupnost čísel 0, 1, 2, atd.

3. Vygenerujte náhodné celé číslo `a` z intervalu $[-3, 3]$ a náhodné přirozené číslo `n` z intervalu $[1, 10]$. Pak spočítejte $\sum_{i=0}^{n} a^n$.

Řešení.

1. `A = round(14 * rand(5, 6) - 7)`
   a) `A1 = A(:, 2:2:6)` nebo `A1 = A(:, 2:2:size(A,2))`
   b) `[r, sl] = size(A1)`
   c) `A2 = A1(:, [1 3 2])`
   nebo pomocí permutační matice `A2 = A1 * [1 0 0; 0 0 1; 0 1 0]`
   d) 1. způsob: `A3 = A(size(A,1):-1:1, :)`,
   2. způsob: `A3 = flipud(A)`
   e) `sum(max(A'))`
   f) `A4 = diag(diag(A))`
   g) `A5 = A - tril(A, -1) + 3*ones(size(A)) - triu(3*ones(size(A)))`
   h) `A6 = [A; prod(A)]`

2. `B = (reshape(0:99, 10, 10))'`

3. `a = round(6 * rand(1)-3), n = round(9 * rand(1)+1)`
   `sum(a.^(0:n))`

Kapitola 5

Logické operace

Základní informace

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

Výstupy z výuky

Studenti

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

5.1 Relační operátory

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

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

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

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

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

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

5.2 Logické operátory

Mezi logické operátory patří:

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

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

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

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

Pro logické operátory rovněž existují ekvivalentní funkce:

logický operátor	ekvivalentní funkce
`A & B`	`and(A,B)`
`A|B`	`or(A,B)`
`~A`	`not(A)`

5.3 Logické funkce

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

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

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

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

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

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

`isempty(A)`
: vrací 1, pokud `A` je prázdná matice

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

`islogical(A)`
: vrací 1, pokud `A` je logická matice. Charakter logické matice mají všechny výsledky relačních a logických operátorů a logických funkcí.

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

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

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

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

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

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

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

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

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

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

[1] SIMULINK je nadstavba MATLABu pro simulaci a modelování dynamických systémů.

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

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

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

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

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

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

Příklady k procvičení

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

Řešení.

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

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

Kapitola 6

Textové řetězce

Základní informace

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

Výstupy z výuky

Studenti

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

6.1 Vytváření řetězců

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

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

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

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

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

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

6.2 Základní manipulace s řetězci

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

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

Znaky se dělí do několika skupin. Znaky s kódem menším než 32 mají speciální význam: konec řádku (kódy 10 a 13), konec stránky (kód 12), tabulátor (kód 8) apod. Znaky s kódy vyššími se běžně zobrazují na obrazovce, přičemž znaky s kódy většími než 127 se mohou lišit podle operačního systému nebo nastaveného jazyka.
V MATLABu existuje několik funkcí k testování typů objektů, jejich název zpravidla začíná slabikou `is*` a výstupem je hodnota `1`, pokud objekt daného typu nabývá, v opačném případě je výstupem `0`. Celý seznam těchto funkcí lze zobrazit příkazem `doc is*`, níže je uveden seznam nejpoužívanějších z nich.

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

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

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

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

6.3 Funkce pro manipulaci s řetězci

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

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

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

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

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

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

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

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

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

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

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

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

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

6.4 Funkce `eval` a `feval`

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

```
>> s = 'sin(2*pi*a*x)';
>> x = 0:0.1:1;
>> a = 1;
>> y = eval(s);
y =
     0   0.5878   0.9511   0.9511   0.5878   0.0000   -0.5878   -0.9511
-0.9511   -0.5878   -0.0000
```

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

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

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

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

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

Příklady k procvičení

1. Definujte následující tři textové řetězce: `t1 = 'Textovy retezec'`, `t2 = 'je posloupnost znaku'` a `t3 = 'uzavrena v apostrofech.'`.
   a) Vytvořte řádkový vektor `t` poskládaný z řetězců `t1`, `t2` a `t3`.
   b) Vytvořte matici `T` obsahující řetězce `t1`, `t2` a `t3` v řádcích.
   c) Zjistěte rozměry matice `T`.
   d) Zjistěte počet mezer ve druhém řádku matice `T` s koncovými mezerami i bez nich a porovnejte s počtem mezer v řetězci `t2`.
   e) Vypište pozice, na kterých se v textovém řetězci `t1` nachází řetězec `'e'`.
   f) Zjistěte, na kolika pozicích vektoru `t` se nachází řetězec `'o'`.
   g) Zjistěte, v kolika sloupcích matice `T` se nachází mezera.
   h) Každý výskyt řetězce `'z'` ve vektoru `t` nahraďte otazníkem.

2. Definujte funkci $e^{a \cdot x}$ a pro `a=0.5` a ekvidistantní vektor `x` délky 5 s počátečním bodem 0 a koncovým bodem 3 vyčíslete.

Řešení.

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

   b) `T = char(t1, t2, t3)` nebo
   `T = strvcat(t1, t2, t3)`

   c) `size(T)`

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

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

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

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

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

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

Kapitola 7

Vyhodnocování výrazů

Základní informace

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

Výstupy z výuky

Studenti

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

7.1 Výraz jako textový řetězec

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

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

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

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

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

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

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

Pro řešení rovnic slouží funkce `solve()`.

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

7.2 Symbolický výraz

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

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

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

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

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

Pro řešení rovnic slouží funkce `solve()`.

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

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

7.3 Výraz jako INLINE funkce

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

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

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

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

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

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

7.4 Polynomy

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

lze samozřejmě zacházet stejně jako s výrazy popsanými výše, tento přístup ovšem není tak efektivní.
Polynom je v MATLABu definován jako posloupnost koeficientů seřazených od členu s nejvyšší mocninou po absolutní člen. Nulové hodnoty v této posloupnosti odpovídají chybějícím členům polynomu.

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

Základní funkce pro práci s polynomy:

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

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

```
>> p1 = poly([-1 1])    polynom p s kořeny -1, 1
p1 =
     1 0 -1
>> pd = polyder(p)    derivace polynomu p
pd =
     8   -9   0   5
>> p = conv([1 -1], [2 0])    součin p(x) = 2x^2 - 2x polynomů p1(x) = x - 1 a
p2(x) = 2x
p =
     2   -2   0
>> [podil, zbytek] = deconv(p, p1)    podíl a zbytek při dělení polynomu p po-
lynomem p1
podil =
     2
zbytek =
     0   0   0   2   -2
```

Příklady k procvičení

1. Definujte funkci $f(x) = x^4 + 4x^3 + 3x^2 - 4x - 4$ jako:
   a) textový řetězec `f1`,
   b) symbolický výraz `f2`,
   c) inline funkci `f3`,
   d) polynom `f4`.

2. Spočítejte funkční hodnoty funkcí `f1`, `f2`, `f3` a `f4` v bodech `x = [0 1 2 3]`.

3. Pro `f1`, `f2` a `f4` řešte rovnici $f(x) = 0$.

4. Vyčíslete první derivaci funkce $f(x)$ v bodech `x` pomocí symbolického výrazu a polynomu.

5. Vyčíslete integrál z funkce $f(x)$ v bodech `x` pomocí symbolického výrazu a polynomu.

Řešení.

1. a) `f1 = 'x^4 + 4*x^3 + 3*x^2 - 4*x - 4'`
   b) `f2 = sym('x^4 + 4*x^3 + 3*x^2 - 4*x - 4')`
   c) `f3 = inline('x^4 + 4*x^3 + 3*x^2 - 4*x - 4')`
   d) `f4 = [1 4 3 -4 -4]`

2. `x = 0:3`
   a) `y1 = eval(vectorize(f1))`
   b) `y2 = subs(f2, 'x', x)`
   c) `f3 = vectorize(f3), y3 = f3(x)`
   d) `y4 = polyval(f4, x)`

3. a) `res1 = solve(f1)`
   b) `res2 = solve(f2)`
   c) `res4 = roots(f4)`

4. a) `yd2 = eval(vectorize(diff(f2)))`
   b) `yd4 = polyval(polyder(f4), x)`

5. a) `yint2 = eval(vectorize(int(f2)))`
   b) `yint4 = polyval(polyint(f4), x)`

Kapitola 8

Práce se soubory

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

Základní informace

Výstupy z výuky

Studenti

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

8.1 Záznam práce

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

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

8.2 Ukládání a načítání proměnných

Pro uložení proměnných do souboru slouží příkaz `save`. Jeho použití bez parametrů uloží všechny definované proměnné do souboru s názvem `matlab.mat`.
Pro uložení pouze vybraných proměnných do námi zvoleného souboru zadáme jako první parametr název souboru a jako další parametry (odděleny čárkami nebo mezerami) názvy proměnných, které chceme uložit. Uložené soubory mají automaticky příponu `.mat`. Je možné zadat i příponu jinou.

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

Načíst data z uloženého souboru můžeme příkazem `load`. Použijeme-li ho bez parametrů, načtou se všechny uložené proměnné ze souboru `matlab.mat`. Příkaz `load` je možné použít podobně jako příkaz `save`.
Pokud má soubor s uloženými daty jinou příponu než `.mat`, musí se při načítání jeho obsahu použít parametr `-MAT` (lze použít i malá písmena: `-mat`).

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

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

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

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

8.3 Soubory v systému MATLAB

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

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

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

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

8.4 Cesta k souborům

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

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

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

8.5 Další příkazy pro práci se soubory

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

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

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

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

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

Příklady k procvičení

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

Řešení.

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

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

Kapitola 9

Práce s grafikou

Základní informace

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

Výstupy z výuky

Studenti

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

9.1 Funkce `plot()` a její použití

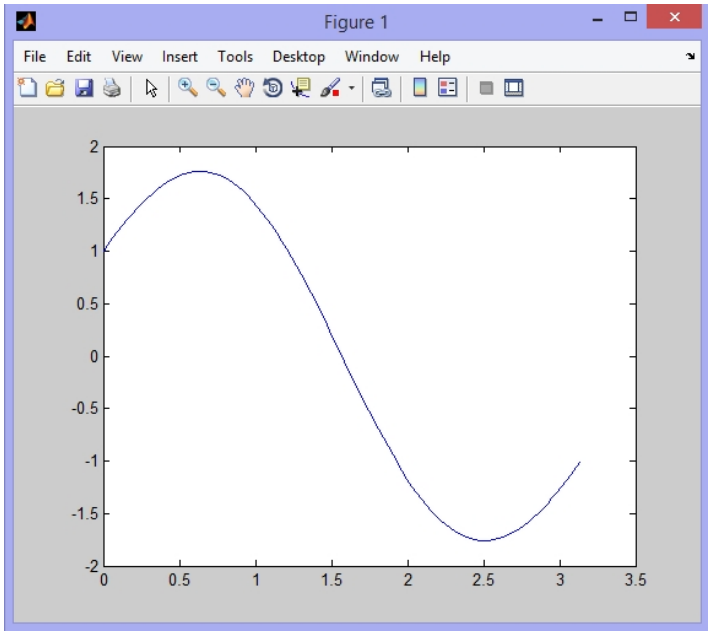
Základním příkazem pro kreslení grafů funkcí jedné proměnné je funkce `plot()`, která při implicitním nastavení spojuje zadané body úsečkami.

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

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

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

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



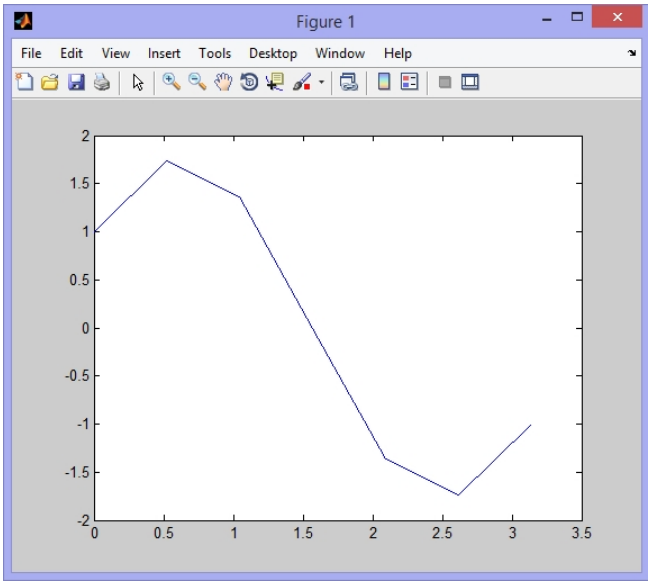
Obr. 9.1. Výstup funkce `plot(x, y)`.

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

```
>> x = linspace(0, pi, 7);
>> y = sin(2*x) + cos(x);
>> plot(x, y)
```

vygenerování menšího počtu hodnot na ose x

vykreslení grafu (Obr. 9.2)



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

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

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

Znaky pro styl čáry:

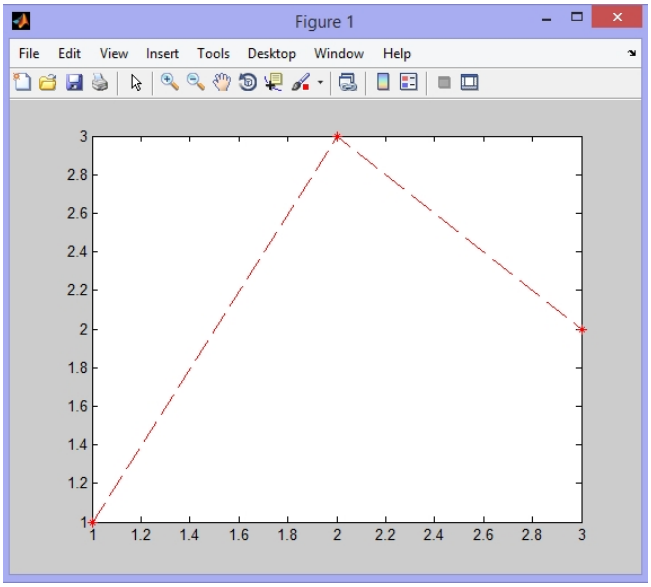
- `-` plnou čarou
- `--` čárkovaně
- `:` tečkovaně
- `-.` čerchovaně

Znaky pro symboly zobrazených bodů:

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

```
>> x = [1, 2, 3];
>> y = [1, 3, 2];
>> plot(x, y, 'r--*')
```

vykreslení daných bodů červenou čárkovanou čarou s body vyznačenými hvězdičkami (Obr. 9.3)



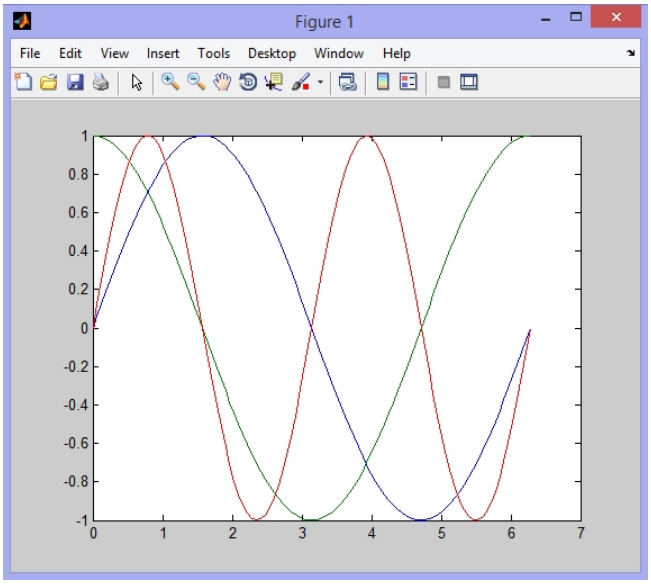
Obr. 9.3. Výstup funkce `plot(x, y, 'r--*')`.

`>> plot(x, y, '+r')` vykreslí pouze zadané body bez spojování úsečkou. V případě, že explicitně nezadáme barvu, je volena automaticky.

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

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

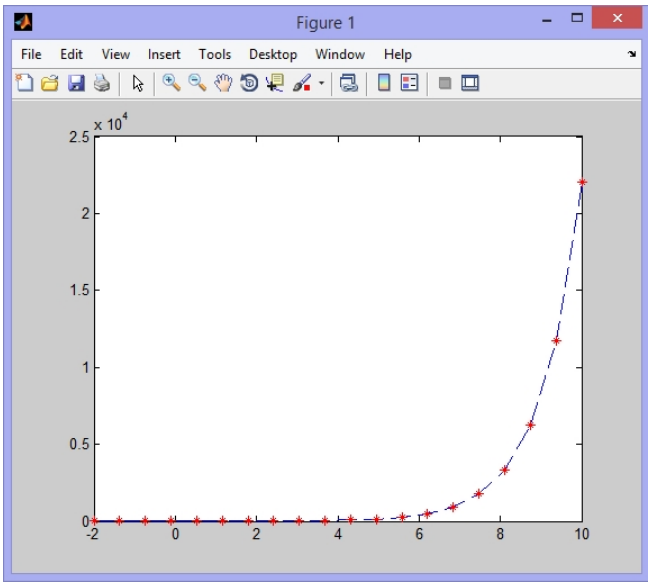
`>> plot(x, y1, x, y2, x, y3)` vykreslí tři grafy uložené v proměnných `y1`, `y2` a `y3`, všechny pro stejný definiční obor daný proměnnou `x` (Obr. 9.4)



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

```
>> x = linspace(-2, 10, 20);
>> y = exp(x);
```

`>> plot(x, y, 'b--', x, y, 'r*')` vykreslení bodů jinou barvou než je barva spojovacích čar (Obr. 9.5)



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

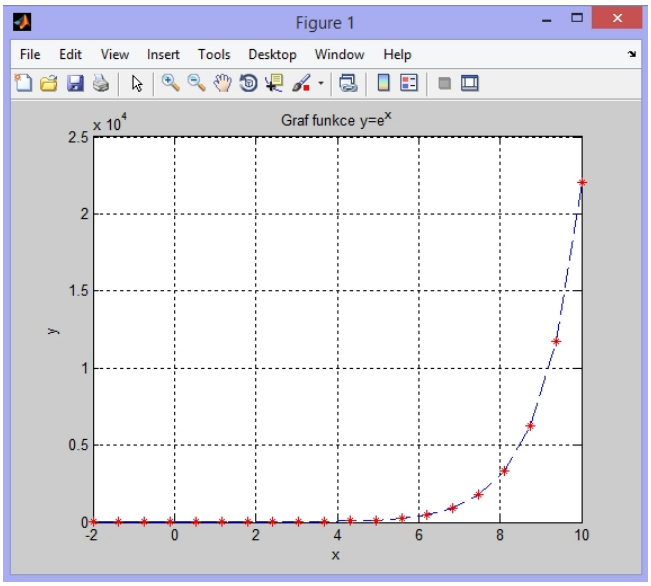
9.2 Vzhled grafu

MATLAB obsahuje několik funkcí, díky nimž si uživatel může přizpůsobit vzhled grafu podle vlastních představ, obsahuje funkce pro popisky os, název grafu, vkládání textu do grafu, apod.
Níže jsou uvedeny nejpoužívanější funkce a příkazy:
• `hold on` přepínač pro zapnutí přikreslování dalších grafů do již existujícího grafu. Pro vypnutí této vlastnosti slouží příkaz `hold off`
• `grid on` zapne zobrazení mřížky pro lepší orientaci v grafu, vypnout lze příkazem `grid off`
• `text(x, y, 'text')` do obrázku na místo o souřadnicích [`x`, `y`] umístí popisek `text`
• `gtext('text')` do obrázku na místo interaktivně zvolené myší umístí popisek `text`
• `xlabel('popisx')` vytvoří popis osy x
• `ylabel('popisy')` vytvoří popis osy y
• `title('nazev')` vytvoří název obrázku
• `legend(pozice, 'popis')` vytvoří legendu grafu tvořenou čárkami oddělenými textovými řetězci s popisem, je zobrazena na místě daném umerickou hodnotou parametru `pozice` (pozice může být dána i textovým řetězcem – více viz `doc legend`):

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

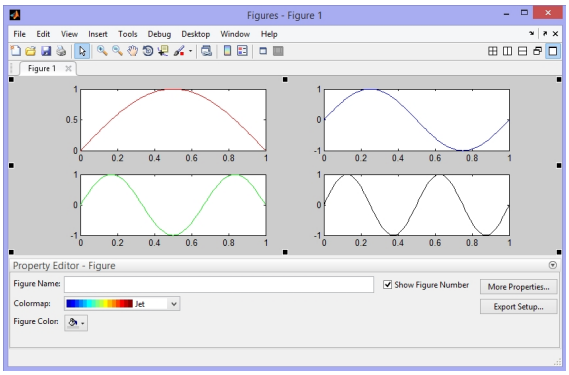
• `axis([xmin xmax ymin ymax])` numerické hodnoty `xmin`, `xmax`, `ymin`, `ymax` pro úpravu rozsahů os
• `figure()` otevře nové prázdné okno

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

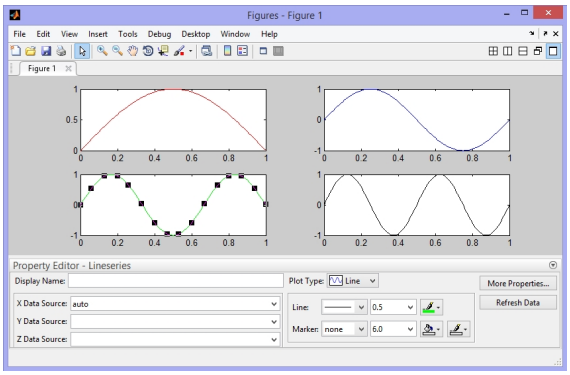


Obr. 9.6. Demonstrace funkcí `title()`, `xlabel()`, `ylabel()` a příkazů `grid on` a `hold on`.

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

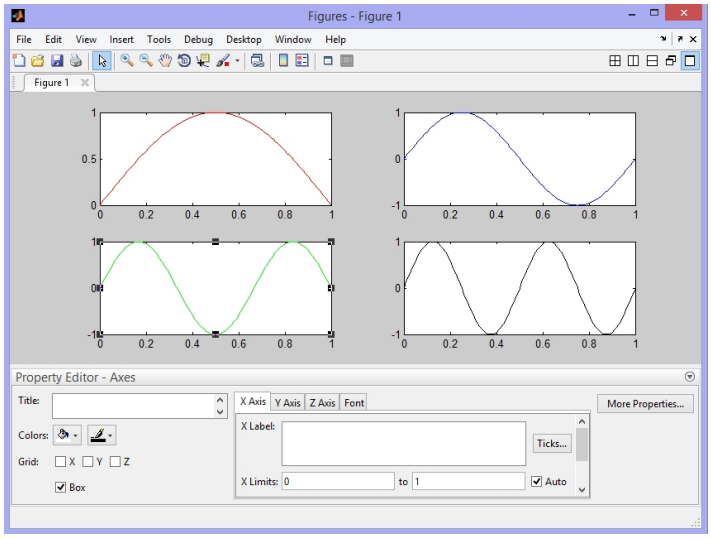


Obr. 9.7. Menu *Figure Properties ....*



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

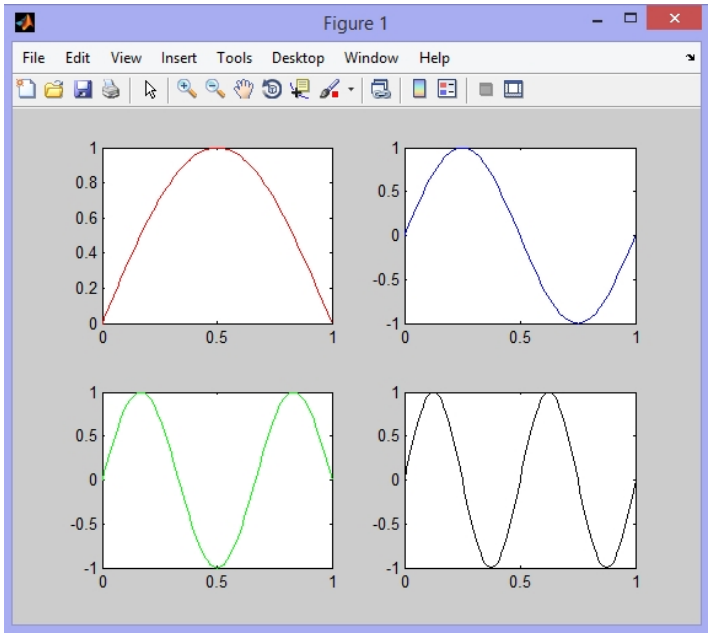
Poklikáním na osy grafu nebo v menu *Edit* → *Axes properties ...* lze v dalším otevřeném menu (Obr. 9.9)měnit popisky, rozsahy a barvy os, lze zapnout mřížka, ohraničení grafu, apod.



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

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

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



Obr. 9.10. Funkce `subplot()`.

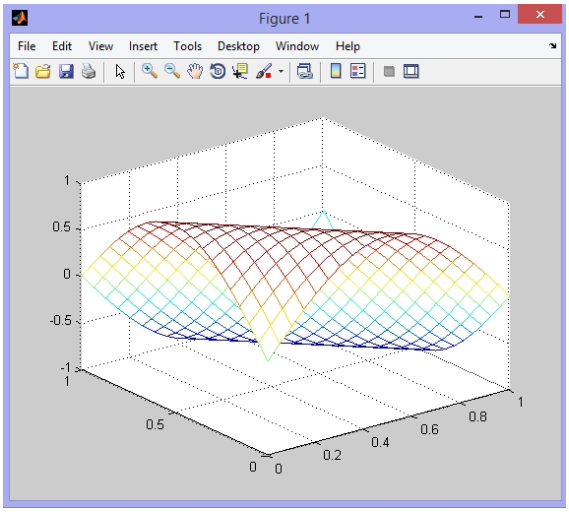
Vytvořený graf lze uložit v menu grafického okna *File* → *Save as...*.

9.3 3D grafika

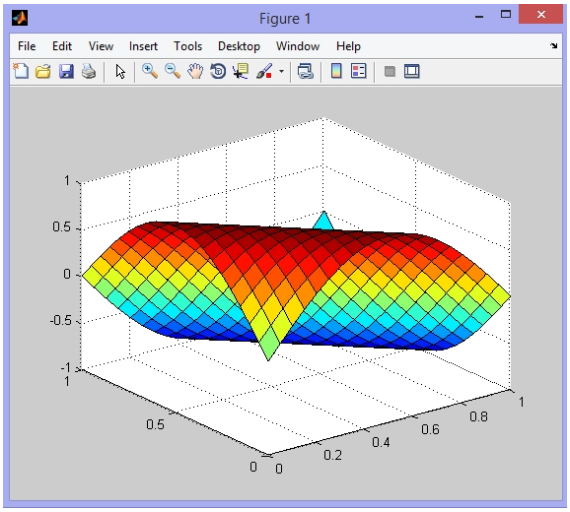
Pro kreslení grafů funkcí dvou proměnných můžeme použít základní funkce:
`mesh(X, Y, Z)` vytvoří 3D síťovaný graf
`surf(X, Y, Z)` vytvoří 3D síťovaný graf s barevně vyplněnými ploškami
Parametry `X` a `Y` jsou jsou matice nezávislých proměnných, třetí parametr `Z` obsahuje

matici funkčních hodnot. V případě vynechání nezávislých proměnných je graf indexován rozměry matice `Z`.
Pro snadnější vytváření dvojrozměrných grafů slouží funkce `meshgrid()`, která vytvoří z jednorozměrných vektorů nezávislých proměnných dvourozměrné sítě vhodné pro definování grafu dané funkce. Použitím `[X, Y] = meshgrid(x, y)` vytvoříme z vektorů `x` a `y` síť bodů, jejíž souřadnice jsou uloženy v maticích `X` a `Y`.

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



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

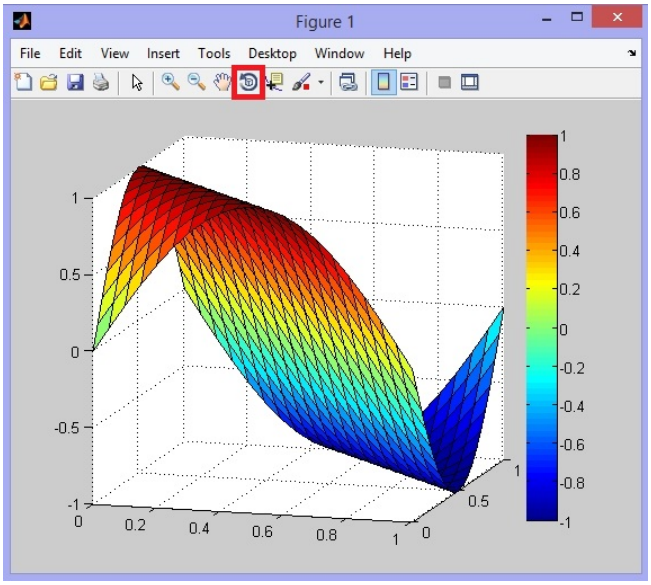


Obr. 9.12. Výstup funkce `surf(X, Y, Z)`

Vykreslené graf obsahují celou škálu barev podle funkčních hodnot v jednotlivých bodech. Barevnou stupnici tohoto škálování lze při okraji grafu zobrazit příkazem `colorbar`.
Úhel pohledu na trojrozměrný obrázek lze nastavit funkcí `view(az, el)`. Parametr `az` udává azimut v rovině nezávislých proměnných, tj. otočení ve stupních kolem osy z; kladné hodnoty udávají otočení proti směru hodinových ručiček. Parametr `el` udává tzv. elevaci, což je úhel směru pohledu s rovinou nezávislých proměnných. Implicitně je nastaven azimut -37.5° a elevace 30°.

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

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

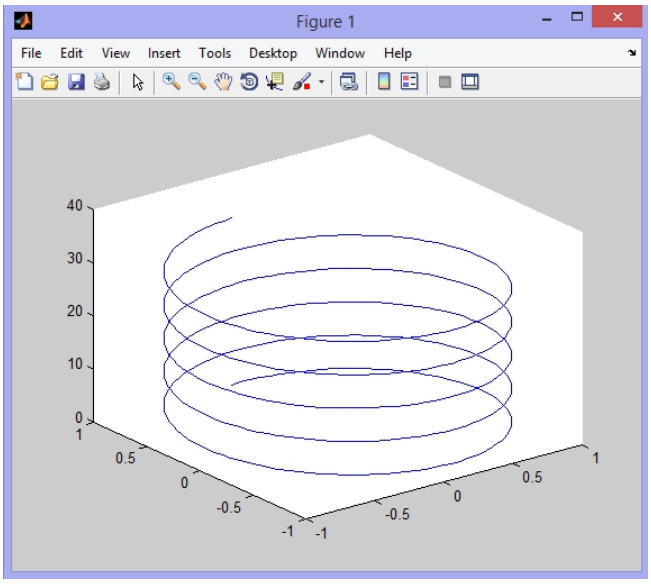


Obr. 9.13. Zobrazení stupnice škálování barev a natočení grafu.

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

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

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



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

9.4 Vlastnosti grafických objektů

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

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

vykreslení funkce sinus spolu s přiřazením do proměnné `p`. Protože příkaz není ukončen středníkem, vypíše se hodnota definované proměnné `p`, ovšem vlastní hodnota není důležitá. Zadáním příkazu `get(p)` získáme výpis všech vlastností vykresleného grafu.

```
p =
     176.0278
```

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

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

Příklady k procvičení

1. Nakreslete graf funkce $f1(x) = x^2$ pro $x \in [-2, 2]$. Graf řádně otitulkujte.
2. Červeně přikreslete graf funkce $f2(x) = x^4$.
3. Do nového grafického okna nakreslete tyto grafy vedle sebe.
4. Jedním příkazem vykreslete pro $x \in [-6, 6]$ graf funkce

$$f3(x) = \begin{cases} (x+3)^3 + x & x <= 0 \\ (x-2)^4 + (x-5)^2 + 1 & x > 0 \end{cases}$$

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

Řešení.

1. ```
   x = linspace(-2, 2);
   plot(x, x.^2)
   title('Graf 2. mocniny'), xlabel('osa x'), ylabel('osa y')
   ```

2. ```
   hold on
   plot(x, x.^4, 'r')
   ```

3. ```
   figure
   subplot(1, 2, 1)
   plot(x, x.^2), title('2. mocnina'), xlabel('x'), ylabel('y')
   subplot(1, 2, 2)
   plot(x, x.^4), title('4. mocnina'), xlabel('x'), ylabel('y')
   ```

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

5. ```
   x = linspace(-2, 2); y = x; figure
   [X, Y] = meshgrid(x, y);
   surf(X, Y, X.^2 + Y.^2)
   ```
   ```

Kapitola 10

Programování v MATLABu

Základní informace

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

Výstupy z výuky

Studenti

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

10.1 Dávkové soubory (skripty) a funkce

Programy v MATLABu, které si může uživatel běžně vytvořit, lze rozdělit do dvou skupin: dávkové soubory neboli skripty a funkce. Hlavní rozdíl mezi nimi je v tom, že

funkce může pracovat se vstupními a výstupními proměnnými, dávkový soubor nikoliv. Další rozdíl je v lokálních a globálních proměnných, ten bude popsán dále. Obě skupiny programů řadíme mezi tzv. M-fajly, neboť jsou uloženy v souborech s příponou `.m` – např. `dávka1.m`, `funkce2.m` apod.

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

Příklad obsahu souboru `startup.m`:

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

Funkce musí začínat hlavičkou, která má tvar:

`function [vystupni_parametry] = nazev_funkce(vstupni_parametry)`

a měla by končit příkazem `end`.

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

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

`function funkce1`

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

Příklad jednoduché funkce:

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

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

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

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

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

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

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

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

10.2 Lokální a globální proměnné

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

Výjimku tvoří dávkové soubory, ve kterých jsou známé proměnné definované v prostředí, odkud byly zavolány. Naopak pokud v dávce nějaké proměnné definujeme, jsou pak známé i po jejím ukončení.
Mějme soubor `davka1.m` obsahující příkazy

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

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

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

10.3 Základní programové struktury

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

10.3.1 Větvení programu

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

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

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

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

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

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

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

```
s = input('Zadejte libovolne cislo: ')
if s < 0
     disp('Zadali jste zaporne cislo.');
elseif s > 0
     disp('Zadali jste kladne cislo.');
else
     disp('Vami zadana hodnota je bud 0 nebo to neni cislo!');
end
```

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

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

```
  otherwise
    příkazy_jiné
end
```

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

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

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

10.3.2 Cykly

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

```
while podmínka
  příkazy
end
```

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

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

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

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

```
for prom=výraz
  příkazy
end
```

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

```
for k=1:n
  příkaz
end
```

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

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

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

Pro tento účel stačí v MATLABu napsat příkaz

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

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

10.4 Nástrahy při programování v MATLABu

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

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

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

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

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

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

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

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

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

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

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

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

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

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

10.5 Ladění programu

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

K ladění slouží následující příkazy:

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

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

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

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

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

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

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

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

Příklady k procvičení

1. Vytvořte funkci `nasobky()`, která pro vstupní parametry `k` a `n` vypíše prvních `k` násobků čísla `n`.

2. Vytvořte funkci `test()`, která pro vstupní vektor známek (1 – 5) testu z matematiky určí četnosti jednotlivých hodnocení. Výstup uložte do matice – v prvním sloupci hodnocení, ve druhém sloupci počet hodnocení.
   Funkce by také měla zkontrolovat, zda se opravdu jedná o celočíselný vstupní vektor s hodnotami 1, 2, ..., 5, v opačném případě by měla skončit chybovým hlášením.

3. Vytvořte funkci `soucet()`, která bude náhodně generovat hodnoty z intervalu $[0, 1]$, dokud jejich součet nepřevýší hodnotu vstupního parametru `s` (`s > 1`). Funkce na svém výstupu vypíše vektor vygenerovaných hodnot `vektor` a jejich součet `suma`.
   Funkce by měla ověřit, zda je `s` numerická hodnota větší než 1, v opačném případě by měla skončit chybovým hlášením.

Řešení.

1. ```
   function[vystup] = nasobky(k, n)
       % pro zadana "k"a "n"vypise prvnich "k"nasobku cisla "n"

       vystup = (1:k) .* n;

   end
   ```

2. ```
   function[vystup] = test(v)
       % pro zadany vektor hodnoceni testu vypocita jejich cetnosti

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

       for i = 1:5
   ```

```
        cetnost(i) = sum(v == i);

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

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

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

  suma = 0;
  vektor = [];
  while suma <= s
     prvek = rand(1);
     suma = suma + prvek;
     vektor = [vektor, prvek];
  end

end
```

Seznam použité literatury

[1] DUŠEK, F. *MATLAB a SIMULINK - úvod do užívání*. Univerzita Pardubice, 2000. 147 s. ISBN 80-7194-273-1

[2] PÄRT-ENANDER, Eva. *The Matlab handbook*. Harlow: Addison-Wesley, 1997. 423 s. ISBN 0-201-87757-0

[3] The Matworks, autoři MATLABu a SIMULINKu. *The Mathworks.*[online], [září 2014]. Dostupné z WWW: <http://www.mathworks.com/>

[4] ZAPLATÍLEK, K., DOŇAR, B. *MATLAB pro začátečníky*. 1. vydání. Praha: BEN - technická literatura, 2003. 144 s. ISBN 80-7300-095-4

Část II

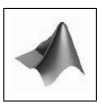
Výuka jazyka R

Úvod

R je volně dostupný jazyk používaný zejména v akademické a vědecké sféře. Jedná se o prostředí pro statistickou analýzu dat a jejich grafické zobrazení. Standardní distribuce R obsahuje množství funkcí pro manipulaci s daty, výpočty a grafické výstupy, široká škála dalších funkcí je součástí mnoha podpůrných balíčků. Jedná se o programovací prostředí umožňující definovat rovněž své vlastní funkce a skripty.
Bezesporným kladem jazyka R je jeho grafika. Ta je založena na systému high-level a low-level funkcí, jejichž vhodnou kombinací může uživatel dosáhnout grafického výstupu přesně dle své představy.

Následující studijní text seznamuje studenty se syntaxí jazyka R, jeho datovými strukturami, funkcemi pro tvorbu grafiky stejně jako se základy programování. Text je doplněn o množství jak ukázkových příkladů, tak příkladů určených pro samostatné řešení a je vytvořen pro práci v prostředí klasické příkazové řádky jazyka R. V současné době je ovšem možnost využít různých nadstaveb a uživatelsky přívětivějšího prostředí, např. RStudio, RKWard apod.
Program je volně dostupný na webových stránkách projektu (`http://www.r-project.org`, [19]), kde jsou k dispozici i podrobné manuály ([17], [10]) a online nápověda. Srozumitelným a velmi přehledně zpracovaným materiálem s množstvím názorných příkladů a doplňkových otázek je skriptum P. Drozd: Cvičení z biostatistiky [3], z anglické literatury stojí za zmínku studijní materiály Theresy A. Scott [11], [12], [13], [14]. Velmi přínosné jsou dokumenty srovnávající příkazy v R s ekvivalentními příkazy v MATLABu [5], [4].

Někdy se syntaxe či chování R a MATLABu výrazně liší. V těchto případech je v textu na tyto odlišnosti upozorněno a pro zvýraznění a snadnější orientaci je na okrajích stránek použita ikonka MATLABu [20].



Kateřina Konečná, Jan Koláček

Kapitola 1

První setkání s jazykem R

Základní informace

R je volně šiřitelný programovací jazyk, který umožňuje špičkové zpracování dat včetně grafických výstupů. Jeho široké spektrum využití zahrnuje možnost doplnění základních funkcí ohromným množstvím balíčků s funkcemi pro různé typy analýz.
V této kapitole popíšeme, jak prostředí R vypadá, naučíme se zde orientovat, zaměříme se na základní syntaxi a pravidla pro práci s proměnnými a funkcemi. Cílem není znát všechny argumenty funkcí zpaměti, ale dobře jim porozumět a vědět, kde tyto informace najít. Důležitou součástí kapitoly je proto naučit se pracovat s nápovědním systémem jazyka, popř. vědět, kde najít podrobnější manuály vhodné zejména pro začátečníky.
Jazyk R pracuje nad datovými strukturami - těm nejčastěji používaným je stručně věnována poslední část kapitoly.

Výstupy z výuky

Studenti

- se seznámí s prostředím jazyka R a jeho základními příkazy,
- umí využívat nápovědního systému,
- se orientují v pracovním prostoru a příkazech s ním spjatých,
- rozlišují datové typy objektů, dokáží popsat jednotlivé datové struktury.

1.1 Základní ovládání

R je volně dostupný programovací jazyk a softwarové prostředí pro statistické výpočty a grafiku, který je možný provozovat na operačních systémech UNIX, Windows i MacOS. R je skutečně „jen“ programovací jazyk - žádné grafické rozhraní, které by se dalo ovládat myší, zde není. Pro grafické rozhraní (GUI) je potřeba vyzkoušet některé grafické nástavby (Poor Man's GUI, Rcommander, ...). Program lze získat z webových stránek projektu (`http://www.r-project.org`, [19]).

Po spuštění programu se objeví okno s konzolou (R console), ve kterém se dají spouštět funkce, vkládají se objekty, provádí se výpočty a zobrazují se i základní výstupy. Dále se můžeme setkat i s dalšími typy oken:

- **okno nápovědy** - volá se po zadání příkazu `help()`
- **grafické okno** - při spuštění grafické prezentace
- **editor programu** - zobrazíme ho v menu *File* → *New Script*. Toto okno je určeno k tvorbě programů. (U delších zdrojových kódů je výhodné mít k dispozici editor s číslovanými řádky, kontrolu syntaxe R a závorek - jako freeware PSPad, `http://www.pspad.com` , nebo shareware WinEdt, `http://www.winedt.cz`)

To, že je konzole připravena k používání, symbolizuje prompt `>`, za nímž následují jednotlivé příkazy. Pokud nezměníme základní nastavení, řádky, do nichž vepisujeme příkazy, jsou odlišeny červenou barvou. Výstupní řádek je psán modrým písmem, v případě hodnot začíná pořadovým číslem dané hodnoty. Příkaz v konzole spustíme klávesou ENTER. Takto provedený příkaz nelze opravit přímo, ale v již napsaných příkazech můžeme listovat pomocí kláves ↑ ↓ , jež umožňují pohyb v historii příkazů. Historii příkazů můžeme uložit v menu *File* → *Save History*. Konzole se všemi příkazy může být uložena v menu *File* → *Save to File* ....

Jednotlivé příkazy jsou vkládány na nový řádek, v případě více příkazů za sebou jsou navzájem oddělovány středníkem. Složitější příkazy, které mají být spuštěny bezprostředně po sobě, jsou seskupovány pomocí složených závorek. Každý příkaz je následován kulatými závorkami s argumenty v případě, že jsou požadovány. Argumenty lze uvádět ve stejném pořadí, v jakém byly definovány, nebo je možno je uvádět v libovolném pořadí ve tvaru `nazev.argumentu=hodnota`. Argumenty je často možné zkracovat (jestliže zkratka nemůže znamenat jiný argument), např. `round(x, digits=2)` má naprosto stejný význam jako `round(x, d=2)` (jedná se o funkci zaokrouhlení na 2 desetinná místa).

Příkazem `<-` (popř. `=`, ten se ovšem nedoporučuje) vložíme daný výraz do proměnné. K jejímu zobrazení musí být v konzole z příkazové řádky zavolán jeho název. Další možností je zadání celého příkazu do kulatých závorek.

```
> a <- 2
> a
[1]  2
> (a <- 2)
[1]  2
```

Pro neúplné příkazy se na následujícím řádku objeví prompt `+` , který dává možnost dodat chybějící část příkazu, a pokračuje dál ve čtení.

```
> a <- (2+3
> + ) * 5
[1]  25
```

Symbol `#` na příkazové řádce způsobí ignoraci zbytku řádky (komentář).

```
> a <- 2 # komentar
> a
[1]  2
```

Při zadávání názvu jednotlivých proměnných či objektů bychom měli dodržovat určitá pravidla. Názvy se mohou skládat z písmen (`a`–`z`, `A`–`Z`), číslic (`0`–`9`), tečky a podtržítka. Název musí začínat písmenem, nesmí začínat číslicí, tečka rovněž není doporučována. Matematické operátory (`+`, `-`, `*`, `/`), mezera a jiné další speciální znaky také nejsou povoleny. Chceme-li použít víceslovný název, rozdělení názvu na dvě části se provádí pomocí tečky (např. průměrnou výšku chlapců můžeme nazvat `chlapci.prumer`), popř. pomocí podtržítka (`chlapci_prumer`). Rovněž bychom se měli vyvarovat používání názvů vestavěných proměnných. R je *case sensitive*, tzn. rozlišuje malá a velká písmena.

```
> a <- 2
> a
[1]  2
> A
Error: object 'A' not found
```

Běžící program může být zastaven klávesou ESC ve Windows, CTRL + C v Linuxu. Konzoli můžeme vymazat klávesovou zkratkou CTRL + L nebo v menu *Edit* → *Clear Console*.

Pro ukončení programu se používá příkaz `q()`.

1.2 Nápověda

Příkazem `help()` je vyvoláno další okno – okno nápovědy. V levé části můžeme najít abecedně seřazený seznam objektů, poklikáním na některý z nich se v hlavním okně zobrazí popis objektu. Pro zobrazení nápovědy ke konkrétním objektům či funkcím slouží příkaz `help(tema)`, alternativou může být rovněž `?tema` nebo v menu *Help* → *R-*

functions (text) .... Chceme-li znát argumenty příkazu, můžeme použít funkci `args(nazev_funkce)`.

Pro vyhledávání témat v kontextu (tzn. ne pouze v názvech a zkrácených popisech funkcí a objektů, ale i v klíčových slovech) slouží příkaz `help.search("tema")`, v menu *Help* → *Search Help* nebo v okně nápovědy pod záložkou *Vyhledávat*. Příkazem `apropos("nazev")` zobrazíme příbuzné příkazy k příkazu `nazev`, pro příklady k nějakému tématu použijeme příkaz `example(tema)`.

K dispozici jsou rovněž PDF manuály, které obsahují základní manuály a reference o funkcích. Spouští se přes menu *Help* → *Manuals (in PDF)*. HTML nápovědu, obsahující manuály, seznamy funkcí, apod. zobrazíme příkazem `help.start()` nebo přes menu *Help* → *Html help*.

Jazyk R nabízí i několik funkcí, které uživatelům usnadňují pochopení některých příkazů. Funkce `demo` je uživatelsky příznivé prostředí, ve kterém běží demonstrace (ukázkové programy) R skriptů. Příkaz `demo()` vypíše seznam všech dostupných demonstrací. Argumenty funkce `demo`:

`topic`	téma, které má být demonstrováno
`package`	název balíčku, ze kterého má být spuštěno `topic`

```
> demo(persp)
```
v novém okně spustí ukázky použití funkce `persp`

V MATLABu je analogickým příkazem funkcí `help(funkce)` a `?funkce` příkaz `help funkce`.

1.3 Workspace (pracovní prostor)

Program R načítá data z jakéhokoliv adresáře. Nejjednodušší pro manipulaci s nimi je ukládat si data do tzv. pracovního adresáře (workspace). Chceme-li zjistit, ve kterém adresáři se právě nacházíme, použijeme příkaz `getwd()`. Ten zobrazí kompletní cestu do aktuální složky. Pro změnu pracovního adresáře použijeme příkaz `setwd()`, jehož argumentem je cesta k adresáři v jednoduchých apostofech (') nebo uvozovkách ("). Změna adresáře je rovněž možná přes menu *File* → *Change dir* ....

```
> (setwd("C:/Documents and Settings/PC/Plocha/R"))
```
nastaví za pracovní adresář složku R umístěnou na ploše a vypíše cestu složky, ze které jsme vycházeli.
```
[1]  "C:/Documents and Settings/PC/Dokumenty"
```

O tom, že se opravdu nacházíme v adresáři R, se můžeme přesvědčit příkazem

```
> getwd()
[1]  "C:/Documents and Settings/PC/Plocha/R"
```

Příkazy `dir()` nebo `list.files()` zobrazíme názvy souborů v aktuálním nebo zadaném adresáři.

```
> dir()   výpis souborů v aktuálním adresáři
[1]  "cviceni"  "prednasky"  "data"
> dir("C:/Documents and Settings/PC/Plocha/R/cviceni")
[1]  "cviceni 1"  "cviceni 2"  "cviceni 3"  "cviceni 4"
[5]  "cviceni 5"  "cviceni 6"
> list.files("C:/Documents and Settings/PC/Plocha/R/cviceni")
[1]  "cviceni 1"  "cviceni 2"  "cviceni 3"  "cviceni 4"
[5]  "cviceni 5"  "cviceni 6"
```

Všechny objekty, které v průběhu práce vytvoříme, se ukládají do paměti a můžeme s nimi kdykoliv manipulovat. Vytvořené objekty tvoří tzv. workspace (pracovní prostor), chceme-li je uchovat pro pozdější práci, celý workspace uložíme pomocí *File* → → *Save Workspace* .... Tímto způsobem vytvoříme soubor s koncovkou .R, který se při dalším spuštění ze stejného adresáře automaticky nahraje spolu s historií příkazů. V případě, že zavřeme program R bez uložení, při dalším spuštění už dříve definované objekty nejsou k dispozici. Více o ukládání jednotlivých objektů viz odstavec 6.2.

Další užitečné příkazy:

`objects()`, `ls()` vypíše všechny názvy objektů, které jsou momentálně definovány v aktuálním adresáři

`ls(name, pattern, all.names=T)` vypíše ty objekty, které jsou omezeny volitelnými parametry:

`name` výpis konkrétního objektu podle čísla nebo názvu prostředí

`pattern` vyhledávací výraz (např. objekty začínající na "a")

`all.names` nastavený na hodnotu `TRUE` vyhledává také objekty začínající tečkou

`rm()` maže vybrané objekty

`rm(list=ls())` maže všechny objekty

```
> ls(pattern="v")
[1]  "v"  "vec"  "vector"
> ls(pattern="v", all.names=T)
[1]  ".vec"  "v"  "vec"  "vector"
> rm(v,vec)   příkaz analogický příkazu rm(list=c("v","vec"))
> ls()
[1]  ".vec"  "vector"
```

1.4 Datové typy objektů

Jazyk R používá následující datové typy:

- *numerické hodnoty* vypisujeme klasickým způsobem, k oddělení desetinných míst používáme desetinnou tečku, tzn.

```
> 1.234
```

- *komplexní* - numerická hodnota je doplněna o komplexní jednotku i, např.

```
> 1 + 2i
[1]  1 + 2i
> 3 + 1i   Pozor! Komplexní jednotka musí být vždy doplněna o numerickou hodnotu
[1]  3 + 1i
> 3 + i
Error: object 'i' not found
```

- *logické hodnoty* - hodnoty `TRUE`, `T` a `FALSE`, `F` (Pozor! Jazyk R je citlivý na velká a malá písmena, proto logické hodnoty nelze psát jinak než uvedeným způsobem.) Logické hodnoty se používají pro některé argumenty funkcí, jsou výsledkem testování výrazů, např.

```
> 2.3 > 3
[1]  FALSE
```

- *řetězec* - používá se pro textové hodnoty, pro popisky os, název grafu, ... Řetězec musí být zadán do jednoduchých apostrofů (') nebo dvojitých uvozovek ("):

```
> r <- "retezec"
[1]  "retezec"
> r <- 'retezec'
[1]  "retezec"
```

K dispozici jsou rovněž speciální hodnota `NA` ("Not Available"), která reprezentuje chybějící nebo neznámou hodnotu, a další speciální konstanty: `NULL`, odpovídající prázdnému objektu, `Inf`, odpovídající nekonečnu (např. `1/0`) a `NaN` ("Not-a-Number"), která je výsledkem numerických výpočtů, jež nejsou definovány (např. `0/0` nebo `Inf - Inf`).

1.5 Datové struktury

Vedle datových typů objektů, jazyk R poskytuje i množství datových struktur, které umožňují vícero hodnot specifikovat jako samostatný objekt. Mezi základní datové

struktury patří vektor, faktor, matice, pole, tabulka dat a seznam. V následujících kapitolách se dozvíme, jak jednotlivé datové objekty definujeme, jak s nimi pracujeme. Protože každý objekt má svá specifika, následující tabulka souhrnně uvádí základní datové struktury a odpovídající datové typy, kterými mohou být tvořeny.

datová struktura	**datový typ**	**možnost více typů dat současně**
vektor	numerický, komplexní, logický, řetězec	ne
faktor	numerický, řetězec	ne
matice	numerický, komplexní, logický, řetězec	ne
pole	numerický, komplexní, logický, řetězec	ne
tabulka dat	numerický, komplexní, logický, řetězec	ano
seznam	numerický, komplexní, logický, řetězec, funkce, výraz, ...	ano

Tab. 1.1. Přehled datových typů objektů

Příklady k procvičení

1. Několika způsoby zobrazte nápovědu funkce `mean` a stručně ji popište.
2. Vyhledejte příklad použití funkce `dim`.
3. Spusťte ukázku použití funkce `image`.
4. Vypište cestu k adresáři, ve kterém se nacházíte.
5. Definujte následující proměnné:
   - proměnnou `a` s textovou hodnotou `"pokus"`
   - proměnnou `a1` s hodnotou `1-i`
   - proměnnou `b` s hodnotou `TRUE`
   - proměnnou `c` s hodnotou `-1.333`
   - proměnnou `d` s hodnotou ∞

   a) Vypište seznam všech definovaných proměnných.
   b) Vypište seznam všech proměnných začínajících na "`a`".
   c) Smažte proměnnou `c` a vypište nový seznam definovaných proměnných.

6. Jakou datovou strukturu byste volili pro zobrazení:
   a) údajů o nadmořské výšce dané obdélníkové oblasti,
   b) informace o provedeném experimentu obsahující jeho stručný popis v textové podobě, vstupní data a porovnání výsledků pro různé metody,
   c) pohlaví respondentů,
   d) údajů pro 100 respondentů, u nichž známe jméno, příjmení, věk, výšku a váhu.

Řešení.

1. `help(mean)`, `?mean` nebo v menu *Help* → *Search help ...*, `mean` - funkce pro výpočet aritmetického průměru

2. `example(dim)`

3. `demo(image)`

4. `getwd()`

5. `a <- "pokus"` nebo `a <- 'pokus'`,
   `a1 <- 1-1i`,
   `b <- TRUE` nebo `b <- T`,
   `c <- -1.333`,
   `d <- Inf`
   a) `ls()` nebo `objects()`
   b) `ls(pattern="a")`
   c) `rm(c)`, `ls()`

6. a) matice,
   b) seznam,
   c) vektor, faktor,
   d) datová tabulka.

Kapitola 2

Vektory

Základní informace

V této kapitole uvedeme nejjednodušší datovou strukturu – vektor. Vektory jsou posloupnosti hodnot s jistou informační hodnotou – mohou udávat hmotnosti kaprů v kádi, koncentrace mikroorganismů v ovzduší během určitého časového období či počty krtinců na několika sousedních zahradách. Seznámíme se s několika možnostmi, jak definovat vektory různých datových typů a pracovat s nimi, a s některými speciálními příklady vektorů poměrně často využívanými v praxi – s posloupnostmi a vektory náhodných čísel.
Závěr kapitoly je věnován dalšímu speciálnímu příkladu vektorů – faktoru. Tyto struktury umožňují jednoduše a přehledně definovat a efektivně pracovat s vektory nominálních nebo ordinálních znaků.

Předpokládá se znalost práce s počítačem a základy lineární algebry.

Výstupy z výuky

Studenti

- ovládají základní příkazy pro tvorbu vektorů,
- umí vytvářet posloupnosti a generovat vektory náhodných čísel,
- dokáží pracovat se subvektory,
- umí definovat, vytvářet a používat faktory.

2.1 Základní příkazy, tvorba vektorů

Jazyk R pracuje nad datovými strukturami. Nejjednodušší takovou strukturou je vektor. Např. jednoduchá hodnota (logická hodnota `TRUE`, numerická hodnota `2`, ...) je vektor délky 1. Vektory jsou 1-dimenzionální struktury, skládající se z posloupnosti prvků. Všechny prvky vektoru musí být stejného datového typu (modu) – *numerický*, *komplexní*, *logický* nebo typu *řetězec*.

Některé prvky vektoru ovšem nemusí být známy – v těchto případech je na místo příslušného prvku vektoru umístěna speciální hodnota `NA` („Not Available“). Všechny operace nad `NA` vrací hodnotu `NA`. Funkce `is.na(x)` vrací hodnoty `TRUE` na těch pozicích vektoru `x`, na kterých mají prvky hodnotu `NA`, na ostatních pozicích vrací hodnotu `FALSE`. Rovněž některé prováděné operace nemusí dávat smysl (např. `0/0`, `Inf - Inf`), v těchto případech vrací příkaz hodnotu `NaN` („Not a Number“).

Pro vytvoření numerického vektoru `x` o hodnotách -1.2, 31.8., 10.7, 5.6, 9.22 použijeme příkaz `x <- c(-1.2, 31.8, 10.7, 5.6, 9.22)`. Analogicky můžeme použít příkazu `assign("nazev", c())`, v našem případě tedy `assign("x", c(-1.2, 31.8, 10.7, 5.6, 9.22))`. Přiřazení může být rovněž provedeno v opačném směru: `c(-1.2, 31.8, 10.7, 5.6, 9.22) -> x`.

Naprosto analogickým způsobem můžeme zadávat komplexní vektory, např. `y <- c(1+2i, 3, 5i)`, vektory logických hodnot, např. `z <- (x<12 & x>-1)`, nebo vektory textových hodnot, např. `ovoce <- c("banan", "broskev", "jablko", "pomeranc")`.

Funkce `c()` svůj výstup zobrazuje jako řádkový vektor, ovšem je s ním nakládáno jako s vektorem sloupcovým.

```
> c(2, 4, "v", F)   jedna z hodnot je znak, vektor tedy bude textový
[1]  "2"  "4"  "v"  "FALSE"
```

Příkaz `vector(mode, length)` vytvoří vektor daného typu `mode` a dané délky `length` s nulovými hodnotami.

```
> vector(mode="logical", length=3)
[1]  FALSE   FALSE   FALSE
> vector(m="character", length=5)
[1]  ""  ""  ""  ""  ""
```

Nejjednodušším způsobem ke generování posloupností čísel je použít operátoru `:`. Příkaz `a:b` vygeneruje aritmetickou posloupnost prvků od `a` do `b` s krokem 1, resp. −1. Funkce `sequence(n)` slouží ke generování posloupnosti hodnot od čísla 1 do čísla (vektoru hodnot) uvedeného v argumentu funkce.

```
> 4:10
[1]  4   5   6   7   8   9   10
```

```
> 2.4:7
[1]  2.4  3.4  4.4  5.4  6.4
> sequence(6)
[1]  1  2  3  4  5  6
> sequence(c(3,5))
[1]  1  2  3  1  2  3  4  5
```

Ke generování posloupností s daným krokem MATLAB používá funkci `a:krok:b`. R tuto syntaxi nezná, je proto třeba použít jiných funkcí.

Pro generování složitějších posloupností se používá funkce `seq()`, která má 5 argumentů, ale pouze některé z nich mohou být specifikovány současně při jednom volání.

- První 2 argumenty specifikují počáteční a koncový bod posloupnosti, příkaz `seq(2,10)` je ekvivalentní příkazu `2:10`. Argumenty mohou být rovněž volány pomocí `from=a, to=b`.
  Příkazy `seq(1,30)`, `seq(from=1, to=30)` a `seq(to=30, from=1)` vrací stejný výsledek.
- Argument velikosti kroku: `by=krok`.
- Argument délky posloupnosti: `length=delka`.
- V případě, že je použit argument `along.with`, musí být jediným argumentem funkce. Argument `along.with=v` generuje posloupnost počínající hodnotou 1 a délkou shodnou s délkou vektoru `v`.

```
> seq()   vytvoření prázdné posloupnosti
[1]  1
> seq(from=2, to=10)
[1]  2  3  4  5  6  7  8  9  10
> seq(from=2, to=15, by=3)
[1]  2  5  8  11  14
> seq(length=6)
[1]  1  2  3  4  5  6
> seq(from=3, length=6)
[1]  3  4  5  6  7  8
> (k <- seq(to=13, length=6))
[1]  8  9  10  11  12  13
> seq(along.with=k)
[1]  1  2  3  4  5  6
> 7 + seq(along.with=k)
[1]  8  9  10  11  12  13
```

Pro vytvoření vektoru, v němž se opakuje určitý objekt (hodnota nebo vektor), se používá funkce `rep(x)`. Jejími argumenty jsou:

`times=pocet` udává počet, kolikrát má být objekt za sebe poskládán

`each=pocet` udává počet opakování každé složky objektu

`length.out` udává délku výsledného vektoru

```
> x <- 2:5
> rep(x, times=3)   stejný výsledek dává i rep(x, 3)
[1]  2   3   4   5   2   3   4   5   2   3   4   5
> rep(x, each=3)
[1]  2   2   2   3   3   3   4   4   4   5   5   5
> rep(x, length.out=10)
[1]  2   3   4   5   2   3   4   5   2   3
> rep(rep(x, each=2), times=2)
[1]  2   2   3   3   4   4   5   5   2   2   3   3   4   4   5   5
```

Náhodná čísla generovaná v R (a obecně ve všech softwarech) nejsou ve skutečnosti zcela náhodná, ale jsou generována na základě specifických algoritmů tak, aby se náhodným číslům podobala (tzv. pseudonáhodná čísla). Podle pravděpodobnosti výskytu jednotlivých hodnot můžeme generovat čísla z různých typů rozdělení:

`runif(n, min=a, max=b)` Náhodně vygeneruje `n` náhodných čísel z intervalu (`a`, `b`), každé číslo má stejnou pravděpodobnost výskytu.

`rpois(n, lambda)` Náhodně vygeneruje `n` celých čísel z Poissonova rozdělení s parametrem `lambda`.

`rnorm(n, mean=mi, sd=sigma)` Náhodně vygeneruje `n` reálných čísel z normálního rozdělení se střední hodnotou `mi` a směrodatnou odchylkou `sigma`.

```
> runif(10, min=2, max=8)
[1]  3.468575   2.006183   5.151939   4.864977   5.117221   5.981666
[7]  6.186421   6.470627   7.230629   7.726394
> rpois(10, 5)
[1]  7   11   1   4   3   5   6   3   1   5
> rnorm(10, mean=5, sd=2)
[1]  3.880884   5.902384   9.645860   3.955287   3.923085   3.763641
[7]  1.782563   6.003977   7.512364   4.685642
```

Dalším užitečným příkazem ke generování vektorů je funkce `sample()`. Funkce `sample(x, size, replace=FALSE, prob=NULL)` vytvoří náhodnou permutaci prvků objektu `x`. Objekt `x` může být vektor (číselný, komplexní, logický nebo vektor znaků) nebo přirozené číslo. Argument `size` je přirozené číslo udávající délku výsledného vektoru. Argument `replace=TRUE` umožňuje opakování vybraných prvků, zatímco implicitní nastavení `replace=FALSE` opakování nepovoluje. Argument `prob` umožňuje nastavení pravděpodobnostních vah výběru jednotlivých prvků.

```
> x <- c(1, 3, 5, 7, 2, 6)
> sample(x, 3)
[1]  6   7   2
> sample(x, 3, replace=TRUE)
[1]  3   3   2
> x <- letters[1:15]    vektor prvních 15 písmen abecedy (a–o)
> sample(x, 4, prob=c(1, rep(0.1, 13), 1))
[1]  "a"  "o"  "d"  "l"
> sample(x, 8, prob=c(1, rep(0.1, 13), 1), replace=TRUE)
[1]  "e"  "b"  "o"  "o"  "n"  "a"  "a"  "c"
```

2.2 Subvektory

K výpisu určité podmnožiny prvků vektoru můžeme použít hranatých závorek, `[]`. Obecně má pro vektor `x` tento příkaz tvar `x[index]`, kde `index` je vektor jednoho z následujících tvarů:

1. **Vektor přirozených čísel.** Jedná se o vektor indexů, který nabývá hodnot z množiny $\{1, 2, \ldots, \texttt{length(x)}\}$.

   ```
   > x <- 2:16
   > x[7]    vypíše 7. složku vektoru x
   [1]  8
   > x[4:12]    vypíše 4.–12. složku vektoru x
   [1]  5   6   7   8   9   10   11   12   13
   ```

   Hodnota `NA` je vrácena v případě, že vektor `index` obsahuje přirozené číslo větší než je délka vektoru. Podobně R vrací prázdný vektor `numeric(0)` v případě, že vektor `index` obsahuje nulu:

   ```
   > x[13:18]    vypíše 13.–18. složku vektoru x
   [1]  14   15   16   NA   NA   NA
   > x[c(7, 0, 5, 12)]    index 0 se ignoruje
   [1]  8   6   13
   > x[0]
   [1]  integer(0)
   ```

   Vektor `index` můžeme definovat rovněž použitím funkcí ke konstrukci numerických vektorů, např. `c(), :, rep(), sample(), seq()`

   ```
   > x[c(3, 8, 10, 4, 7)]
   [1]  4   9   11   5   8
   > x[c(rep(4, 2), rep(7, 3))]
   [1]  5   5   8   8   8
   ```

```
> x[sample(x, 5, replace=TRUE)]
[1]  6   10   11   4   6
> x[seq(from=4, to=20, by=3)]
[1]  5   8   11   14   NA   NA
```

Můžeme rovněž použít funkcí `head()` a `tail()`, které vrací prvních/posledních `n` prvků vektoru (implicitní nastavení `n=6`):

```
> head(x)
[1]  2   3   4   5   6   7
> tail(x, n=3)
[1]  14   15   16
```

2. **Vektor záporných celých čísel.** Vektorem záporných celých čísel `index` vymezujeme ty indexy vektoru `x`, jejichž odpovídající hodnoty nemají být vytištěny. Všechny prvky vektoru `x`, kromě těch specifikovaných vektorem `index`, jsou do výsledné podoby vypsány v jejich původním pořadí. Délka výsledného vektoru je `length(x) - length(index)`.

```
> x[-c(1:10)]   vynechá prvních deset prvků vektoru
[1]  12   13   14   15   16
> x[-tail(x, 10)]
[1]  2   3   4   5   6   7
```

3. **Logický vektor.** Prvky odpovídající hodnotám `TRUE` vektoru `index` jsou vypisovány, zatímco prvky odpovídající hodnotám `FALSE` jsou vynechány. Výsledný vektor je stejné délky jako počet hodnot `TRUE` vektoru `index`.

```
> x[c(TRUE, FALSE, TRUE, TRUE, FALSE, rep(c(TRUE, FALSE), 5))]
vektor index může být zadán vektorem hodnot TRUE a FALSE
[1]  2   4   5   7   9   11   13   15
```

Poznámka. V případě, že je logický vektor kratší než je počet prvků vektoru, ze kterého prvky vybíráme (vektoru `x`), je uplatněno pravidlo *recycling rule*. Pravidlo spočívá v opakování složek logického vektoru tak dlouho, dokud jeho délka nedosáhne délky vektoru:

```
> x[c(TRUE, FALSE, TRUE, FALSE)]
[1]  2   4   6   8   10   12   14   16
> x[x>7 & x<=12]   vektor index může být dán omezujícími podmínkami –
použitím porovnávacích a logických operátorů (porovnávací a logické operátory
viz odstavec 5.2)
[1]  8   9   10   11   12
```

V případě, že vektor `x` obsahuje hodnoty `NA`, potřebujeme pomocí funkce `is.na()` zajistit, aby byly správně vynechány:

```
> (y <- c(7, 3, NA, 5, NA, NA, 9))
[1]  7  3 NA  5 NA NA  9
> y[y<6]
[1]  3 NA  5 NA NA
> y[y<6 & !is.na(y)]   vypíše ty hodnoty vektoru y, které jsou menší než 6
a jsou různé od hodnoty NA
[1]  3  5
```

Alternativou v této situaci může být funkce `subset()`. Jejím prvním argumentem je vektor, ze kterého chceme získat jeho podmnožinu, druhým argumentem je vektor omezujících podmínek. Výhodou této funkce je, že hodnoty `NA` jsou automaticky považovány za `FALSE`, takže nemusí být odstraňovány pomocí funkce `is.na()`.

```
> subset(y, y<6)
[1]  3  5
> subset(y, c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE))
[1]  7  3 NA  9
> subset(y, c(TRUE, FALSE, FALSE))   použití pravidla recycling rule
[1]  7  5  9
```

4. **Vektor znakových řetězců.** Tento způsob může být použit pouze v případě, kdy prvky vektoru mají názvy. V tom případě může být vektor `index` použit stejným způsobem jako v případě přirozených čísel v odrážce 1, řetězce ve vektoru `index` odpovídají názvům prvků vektoru `x`.

```
> (ovoce <- c(banan=3, broskev=8, jablko=7, pomeranc=5))   přiřazení
názvu jednotlivým prvkům vektoru
 banan  broskev  jablko  pomeranc
     3        8       7         5
> names(ovoce)   každý prvek vektoru má opravdu svůj název
[1] "banan"  "broskev"  "jablko"  "pomeranc"

Jiný způsob definice přiřazení názvu jednotlivým prvkům vektoru:
> ovoce <- c(3, 8, 7, 5)
> names(ovoce) <- c("banan", "broskev", "jablko", "pomeranc")
> ovoce[c("broskev", "jablko")]
 broskev  jablko
       8       7
```

MATLAB používá pro výpis subvektorů kulatých závorek, pro nekladné argumenty vrací chybové hlášení.

2.3 Délka a změna délky vektoru

Každý vektor má svou délku, což je počet jeho prvků. Ke zjištění délky definovaného vektoru slouží funkce `length()`.

Každý prázdný objekt je nějakého datového typu. Např. příkazem

```
> numeric()
[1]  numeric(0)
```

vytvoříme prázdný numerický vektor, analogicky příkazem `character()` vytvoříme prázdný textový vektor, atd. K již existujícímu objektu libovolné délky můžeme přidávat nové složky, a to jednoduše umístěním indexu hodnoty do hranaté závorky. Tak příkazem `u[2] <- 5` vytvoříme nový vektor `u` délky 2 (1. složka není známa, má tedy hodnotu `NA`, 2. složka má hodnotu 5). Tento příkaz můžeme použít pro jakoukoliv strukturu za předpokladu, že datový typ přidávaných prvků je shodný s datovým typem objektu.

Naopak, ke zkrácení délky objektu je třeba jen operátor přiřazení `<-`, kde za název proměnné umístíme do hranatých závorek ty indexy prvků, které nás zajímají. Symbol záporného znaménka před vektorem indexů v hranatých závorkách určuje ty hodnoty, které nemají být vypsány na výstupu.

```
> u <- 5:12
[1]  5   6   7   8   9   10   11   12
> length(u)
[1]  8
> u[c(1, 3, 5)]   vypíše 1., 3. a 5. prvek vektoru u.
[1]  5   7   9
> u[-c(1, 3, 5)]   vynechá 1., 3. a 5. prvek vektoru u.
[1]  6   8   10   11   12
```

V případě, že chceme získat prvních `n` složek vektoru, použijeme příkaz `length(v) <- n`. Stejným způsobem můžeme i prodlužovat vektory, přidané pozice budou mít hodnotu `NA`.

```
> length(u) <- 3   analogické příkazy: u[c(1,2,3)], u[1:3], head(u,3)
> u   přesvědčíme se, že vektor u opravdu obsahuje jen původní 3 složky
[1]  5   6   7
> length(u) <- 5
> u
[1]  5   6   7   NA   NA
```

2.4 Faktory

Faktory jsou speciálním případem vektorů s nominálními nebo ordinálními daty. Jedná se o datovou strukturu, která umožňuje přiřadit smysluplné názvy jednotlivým kategoriím. Na první pohled vypadají faktory podobně jako numerické a textové vektory, ale není tomu tak. Faktory v sobě navíc obsahují informaci `Levels` – jedná se o konečnou množinu hodnot, kterých kategorická proměnná může nabývat. Jednotlivé prvky `Levels` jsou uspořádány podle jejich typu (numericky nebo abecedně), hodnoty `NA` zde ovšem nejsou zahrnuty.

```
> factor(c("kocka", "kun", NA, "pes", "kocka", "pes", "pes"))
[1]  kocka   kun   <NA>   pes   kocka   pes   pes
 Levels:  kocka   kun   pes
```

Poznámka. Je důležité si uvědomit, že prvky numerického faktoru nejsou interpretovány jako numerické hodnoty:

```
> mean(factor(1:5))    funkce mean slouží k výpočtu průměru
[1]  NA
 Warning message:
In mean.default(factor(1:5)) :
argument is not numeric or logical: returning NA
```

Funkce `factor()` má několik volitelných argumentů (`levels`, `exclude`, `ordered`). Argument `levels` může být použit k definování úrovní (`Levels`) faktoru. Např. můžeme vytvořit faktor i s úrovní, která se mezi daty nevyskytuje:

```
> factor(c(2, 3, 1, NA, 3, 2), levels=1:4)    hodnoty NA nejsou do Levels vy-
pisovány
[1]  2   3   1   <NA>   3   2
 Levels:   1   2   3   4
```

Argument `levels` rovněž může sloužit ke změně pořadí prvků úrovní:

```
> factor(c(2, 3, 1, NA, 3, 2), levels=c(1, 3, 2, 4))
[1]  2   3   1   NA   3   2
 Levels:   1   3   2   4
```

Argument `labels` se používá k definici popisků:

```
> factor(c(0, 1, 1, 0, 1), labels=c("nepritomen", "pritomen"))
[1]  nepritomen   pritomen   pritomen   nepritomen   pritomen
 Levels:   nepritomen   pritomen
> factor(c(0, 1, 1, 0, 1), labels=c("nepritomen", "pritomen"),
+ levels=c(1,0))
[1]  pritomen   nepritomen   nepritomen   pritomen   nepritomen
 Levels:   nepritomen   pritomen
```

Argument `exclude` ignoruje vybrané prvky, tyto prvky jsou nahrazeny hodnotami `NA`

```
> factor(c(2, 3, 1, NA, 3, 2), exclude=3)
[1]  2   <NA>   1   <NA>   <NA>   2
 Levels:   1   2   <NA>
```

`> factor(c(2, 3, 1, NA, 3, 2), exclude=NULL)` argument `exclude=NULL` slouží pro zobrazení hodnoty `NA` v `Levels`

```
[1]  2   3   1   <NA>   3   2
 Levels:   1   2   3   <NA>
```

Argument `ordered=TRUE` slouží k seřazení úrovní faktoru. Jediným rozdílem na výstupu je zobrazení porovnávacího operátoru `<` mezi jednotlivými úrovněmi faktoru. Tuto vlastnost můžeme použít např. při porovnávání jednotlivých prvků.

```
> (velikost <- factor(c(3, 1, 5, 4, 3, 2, 4), labels=c("mravenec",
+ "hlemyzd", "koza", "slon", "zirafa")))
[1]  koza   mravenec   zirafa   slon   koza   hlemyzd   slon
 Levels:   mravenec   hlemyzd   koza   slon   zirafa
> (velikost <- factor(c(3, 1, 5, 4, 3, 2, 4), labels=c("mravenec",
+ "hlemyzd", "koza", "slon", "zirafa"), ordered=TRUE))
[1]  koza   mravenec   zirafa   slon   koza   hlemyzd   slon
 Levels:   mravenec < hlemyzd < koza < slon < zirafa
```

`> velikost >= "koza"` vrací vektor logických hodnot, na pozicích splňujících podmínku jsou hodnoty `TRUE`, na ostatních pozicích `FALSE`

```
[1]  TRUE   FALSE   TRUE   TRUE   TRUE   FALSE   TRUE
```

Příklady k procvičení

1. V cementárně byla měřena hmotnost pytlů cementu, k dispozici jsou naměřené hodnoty:
   50.3, 50.7, 49.2, 50.1, 49.9, 51.1, 49.8, 48.9 a 50.3.
   Poskládejte tyto hodnoty do vektoru `cement` a vyřešte následující úkoly:
   a) Pomocí vhodné funkce zjistěte délku vektoru `cement`.
   b) Vytvořte vektor `cement1`, který bude obsahovat hodnoty na sudých pozicích vektoru `cement` a dále pak jeho poslední tři hodnoty.
   c) Vytvořte vektor `cement2`, který bude náhodnou kombinací pěti hodnot vektoru `cement` s pravděpodobností 0.4 pro první čtyři hodnoty vektoru `cement` a s pravděpodobností 0.2 pro ostatní hodnoty, hodnoty se mohou opakovat.
   d) Vypište ty složky vektoru `cement`, které jsou větší než 50.

2. Náhodně vygenerujte vektor `v` délky 15 s hodnotami 0 a 1. Dále z něj vytvořte faktor `pohlavi`, ve kterém se hodnota 0 bude vypisovat jako "muž"a hodnota 1 jako "žena".

3. Změňte minulé zadání tak, aby se hodnota 0 vypisovala jako "žena"a hodnota 1 jako "muž".

4. Vytvořte vektor hodnot `v1` od -10 do 0 s krokem 0.5.

5. Vytvořte vektor hodnot `v2` obsahující sudé hodnoty posloupnosti -12, -11, . . . , 11, 12.
   a) Hodnoty vektoru `v2` zdvojte tak, aby nově vzniklý vektor obsahoval hodnoty -12, . . . , 12, -12, . . . , 12.
   b) Hodnoty vektoru `v2` zdvojte tak, aby nově vzniklý vektor obsahoval hodnoty -12, -12, . . . , 12, 12.

6. Proveďte losování tahu sportky, tzn. vyberte 7 čísel od 1 do 49 tak, aby se neopakovala.

7. Náhodně vygenerujte 10 hodnot z normálního rozložení se střední hodnotou 5 a rozptylem 2.

Řešení.

1. `cement <- c(50.3, 50.7, 49.2, 50.1, 49.9, 51.1, 49.8, 48.9, 50.3)`
   a) `length(cement)`
   b) `cement1 <- c(cement[seq(from=2, to=length(cement), by=2)], tail(cement, 3))`
   c) `cement2 <- sample(cement, size=5, prob=c(rep(0.4, times=4), rep(0.2, times=length(cement)-4)), replace=TRUE)`
   d) `cement[cement > 50]`

2. `v <- sample(c(0, 1), size=15, replace=TRUE)` nebo `v <- round(runif(min=0, max=1, n=15))`
   `pohlavi <- factor(v, levels=c(0, 1), labels=c("muz", "zena"))`

3. `pohlavi <- factor(v, levels=c(1, 0), labels=c("muz", "zena"))`

4. `v1 <- seq(from=-10, to=0, by=0.5)`

5. `v2 <- seq(from=-12, to=12, by=2)`
   a) `rep(v2, times=2)`
   b) `rep(v2, each=2)`

6. `sample(1:49, size=7)`

7. `rnorm(n=10, mean=5, sd=sqrt(2))`

Kapitola 3

Matice a pole

Základní informace

Matice mají širokou škálu využití nejen v matematice, ale i v technických a ekonomických vědách a statistice. Jejich uplatnění můžeme hledat např. v řešení systémů lineárních rovnic či v aplikaci maticových operací. Tato kapitola seznamuje s možnostmi, jak definovat matice či pole, uvádí množství postupů a funkcí, jak s nimi jednoduše pracovat, v neposlední řadě uvádí příkazy pro řešení systémů lineárních rovnic.

Výstupy z výuky

Studenti

- ovládají příkazy pro vytváření matic a polí,
- umí pracovat se submaticemi,
- dokáží použít funkce pro manipulaci s maticemi,
- rozlišují maticové násobení a násobení po složkách, umí řešit systémy lineárních rovnic.

3.1 Základní příkazy, tvorba matic a polí

Matice je 2-dimenzionální datová struktura, která se skládá z řádků a sloupců. Stejně jako vektory, všechny prvky matice musí být stejného datového typu (numerický, komplexní, logický nebo textový), mohou rovněž obsahovat prvky s hodnotami `NA`, `NaN`, `NULL` nebo `Inf`. Pole je k-dimenzionální struktura, matice je jejím speciálním typem pro $k = 2$.

Základním způsobem k vytvoření matice je použití příkazu `matrix()`:

`matrix(data, nrow, ncol, byrow, dimnames)` funkce pro tvorbu matice

`data`	vektor hodnot, které mají být uspořádány do matice
`nrow`	počet řádků matice, implicitní nastavení `nrow=1`
`ncol`	počet sloupců matice, implicitní nastavení `ncol=1`
`byrow`	implicitní nastavení `byrow=FALSE` značí plnění matice po sloupcích, nastavení `byrow=TRUE` plní matici po řádcích
`dimnames`	argument typu "seznam"(viz Kapitola 4) pro pojmenování řádků a sloupců matice

```
> matrix(data=u, nrow=4, ncol=5)
```
vytvoří matici o 4 řádcích a 5 sloupcích, prvky jsou skládány po sloupcích

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

```
> matrix(data=u, nrow=4, ncol=5, byrow=TRUE)
```
vytvoří matici 4 x 5, argument `byrow=TRUE` zajistí, aby prvky byly do matice seskládány po řádcích

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
```

Názvy řádků a sloupců mohou být specifikovány argumentem `dimnames`. Jedná se o argument typu seznam (pro více podrobností viz Kapitola 4) o dvou složkách – textových vektorech obsahujících názvy jednotlivých řádků a sloupců.

```
> matrix(u, ncol=5, dimnames=list(c("r1", "r2", "r3", "r4"), c("sl1",
+ "sl2", "sl3", "sl4", "sl5")))
   sl1 sl2 sl3 sl4 sl5
r1   1   4   9  13  17
r2   2   5  10  14  18
r3   3   6  11  15  19
r4   4   8  12  16  20
```

Při definování matice je možné jeden z argumentů jejích rozměrů vynechat - jazyk R si chybějící rozměry sám dopočítá. Rovněž není potřeba uvádět názvy argumentů, pokud zachováme přesné pořadí, které je uvedeno v nápovědním systému. Příkazy

```
matrix(u, ncol=5)
matrix(u, nrow=4)
   matrix(u, 4)
matrix(u, c(4, 5))
```

vrací stejné výstupy jako výše uvedený příkaz `matrix(data=u, nrow=4, ncol=5)`, všechny příkazy seskládají vektor `u` po sloupcích do pěti sloupců a čtyř řádků.

Poznámka. V případě, že je vektor `u` kratší než je počet prvků matice, při vytváření matice je uplatněno pravidlo *recycling rule*. Pravidlo spočívá v opakování složek vektoru `u` tak dlouho, dokud jeho délka nedosáhne počtu prvků matice.

Pro definování polí slouží příkaz `array()`, s nímž je práce analogická:

`array(data, dim, dimnames)` funkce pro tvorbu pole
- `data` vektor hodnot, které mají být uspořádány do pole
- `dim` numerický vektor rozměrů pole, implicitní nastavení `dim=length(data)`
- `dimnames` argument typu "seznam"pro pojmenování jednotlivých dimenzí pole

```
> (ar <- array(u, c(2, 5, 2)))   vytvoří pole o rozměrech 2 x 5 x 2
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]   11   13   15   17   19
[2,]   12   14   16   18   20
```

Dalším způsobem, jak vytvořit matice, popř. pole je příkaz `dim()`, který vektor ve svém argumentu uspořádá po sloupcích do pole požadované dimenze.

```
> u <- 1:20
> dim(u) <- c(4, 5)   vektoru u stanoví dimenzi – 4 řádky, 5 sloupců
> u
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Pro zjištění velikosti matice či pole slouží funkce `dim()`. Výstupem je vektor, jehož první složka udává počet řádků, druhá udává počet sloupců a v případě polí odkazují další složky na příslušné dimenze. Pro zjištění počtu řádků či sloupců matic i polí slouží rovněž funkce `nrow()` či `ncol()`.

```
> dim(u)
[1] 4 5
```

```
> dim(ar)
[1]  2   5   2
> nrow(ar)
[1]  2
> ncol(ar)
[1]  5
```

V systému MATLAB lze funkcí `size()` získat rozměry matice, pole i vektoru. Analogickou funkcí pro `size()` je v jazyce R funkce `dim()`, tu ovšem nemůžeme použít pro zjištění délky vektoru. Funkcím `nrow(x)` a `ncol(x)` odpovídají v MATLABu funkce `size(x,1)` a `size(x,2)`.

Dalšími příkazy pro tvorbu matic mohou být `cbind()` nebo `rbind()`, které své argumenty skládají vertikálně (po sloupcích) nebo horizontálně (po řádcích). Argumenty mohou být vektory libovolných délek a/nebo matice s odpovídajícím počtem řádků nebo sloupců.

```
> cbind(1:3, 4:6)
      [,1] [,2]
 [1,]    1    4
 [2,]    2    5
 [3,]    3    6
> cbind(matrix(1:4, c(2, 2)), matrix(c(8, 11), c(2, 1)))
      [,1] [,2] [,3]
 [1,]    1    3    8
 [2,]    2    4   11
> rbind(1:8, 1:5)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    1    2    3    4    5    6    7    8
 [2,]    1    2    3    4    5    1    2    3
Warning message:
In rbind(1:8 1:5) :
number of columns of result is not a multiple of vector length
(arg 2)
```

v případě, že vektory nejsou stejné délky, složky kratšího vektoru se opakují tak dlouho, dokud nedosáhne rozměru delšího vektoru

Funkce `rbind()` a `cbind()` mohou obsahovat volitelný argument tvaru

`nazev.vektoru=vektor`,

který slouží k pojmenování jednotlivých řádků (v případě funkce `rbind`) a sloupců (v případě `cbind`). Tento způsob pojmenování dimenzí funguje pouze pro vstupní vektory, v případě vstupních matic se název ignoruje.

```
> cbind(sl1=1:3, sl2=4:6)
      sl1  sl2
 [1,]   1    4
 [2,]   2    5
 [3,]   3    6
> rbind(u=1:3, A=matrix(1:6, nrow=2))
   [,1]  [,2]  [,3]
 u    1     2     3
      1     3     5
      2     4     6
```

Na rozdíl od MATLABu nelze v jazyce R matici o jednom sloupci získat transpozicí vektoru (funkce `t()`), je třeba použít jeden z příkazů `matrix()`, `dim()` nebo `rbind()`.

3.2 Submatice

K výpisu určité podmnožiny prvků matice či pole můžeme použít hranatých závorek `[]`. Obecně má pro n-rozměrné pole `A` tento příkaz tvar `A[index_1, index_2, ..., index_n]`, kde `index_1`, odkazuje na řádky, `index_2` na sloupce a na ostatní dimenze odkazují `index_3, ..., index_n`. Odkaz na každou dimenzi může být jedním ze čtyř tvarů uvedených v podkapitole 2.2. V případě, že některý z indexů není specifikován, v úvahu je brána celá délka příslušné dimenze.

```
> (A <- matrix(1:20, 4))
      [,1]  [,2]  [,3]  [,4]  [,5]
 [1,]    1     5     9    13    17
 [2,]    2     6    10    14    18
 [3,]    3     7    11    15    19
 [4,]    4     8    12    16    20
> A[-c(1, 2), c(3, 4, 5)]
      [,1]  [,2]  [,3]
 [1,]   11    15    19
 [2,]   12    16    20
```

Jazyk R se vždy snaží vracet objekty s nejnižší možnou dimenzí. Např. chceme-li vypsat jediný sloupec matice, R jej zobrazí jako řádkový vektor. To ovšem může být v některých případech nežádoucí – toto chování můžeme vypnout argumentem `drop=FALSE`:

```
> A[, 2]
[1]  5   6   7   8
> A[, 2, drop=FALSE]
      [,1]
 [1,]    5
 [2,]    6
 [3,]    7
 [4,]    8
```

3.3 Funkce pro manipulaci s maticemi

`nrow(), ncol()`	počet řádků a sloupců pole
`dim()`	dimenze pole
`t()`	transpozice matice
`diag(A)`	vypíše diagonálu matice `A`
`diag(v)`	vytvoří diagonální matici se složkami vektoru `v` na diagonále
`diag(k)`	pro každé přirozené číslo `k` vygeneruje jednotkovou matici rozměrů $k \times k$
`lower.tri()`, `upper.tri()`	výstupem je matice logických hodnot stejných rozměrů jako matice v argumentu. Hodnoty `TRUE` odpovídají prvkům v dolní/horní trojúhelníkové matici. Implicitní nastavení `diag=FALSE` nezahrnuje diagonálu.
`det()`	determinant matice
`eigen()`	výstupem je seznam o dvou položkách - `values` (vlastní čísla) a `vectors` (vlastní vektory). V případě, že nás zajímají pouze vlastní čísla, popř. pouze vlastní vektory matice, použijeme operátoru `$`: `eigen()$values`, popř. `eigen()$vectors`.
`qr()`	QR rozklad. Jedním z výstupů je i hodnost matice, chceme-li zjistit pouze hodnost matice, můžeme použít příkaz `qr()$rank`
`svd()`	singulární rozklad trojúhelníkové matice
`norm()`	norma matice. Je potřeba nainstalovat a načíst podpůrný balíček `Matrix` pomocí příkazů `install.packages("Matrix")` a `library(Matrix)`. Volitelnými argumenty můžeme zvolit typ normy: `"O"`, `"o"` nebo `"1"` pro maximální součty ve sloupcích, implicitní nastavení, `"I"` nebo `"i"` pro maximální součty v řádcích, `"F"` nebo `"f"` pro Frobeniovu normu (euklidovská norma vektor, když je na vstupní matici nahlíženo jako na vektor)
`sum(diag())`	stopa matice

```
> (A <- matrix(c(3, 2, -1, 0), 2))
      [,1]  [,2]
 [1,]    3    -1
 [2,]    2     0
> (v <- diag(A))
[1]  3   0
> diag(v)
      [,1]  [,2]
 [1,]    3     0
 [2,]    0     0
> diag(2)
      [,1]  [,2]
 [1,]    1     0
 [2,]    0     1
> lower.tri(A)
       [,1]   [,2]
 [1,]  FALSE  FALSE
 [2,]   TRUE  FALSE
> A[lower.tri(A)] <- 0
      [,1]  [,2]
 [1,]    3    -1
 [2,]    0     0
> eigen(A)
$values
[1]  2   1
$vectors
           [,1]       [,2]
 [1,]  0.7071068  0.4472136
 [2,]  0.7071068  0.8944272
> eigen(A)$values
[1]  2   1
> qr(A)$rank
[1]  2
> norm(A, "1"); norm(A, "i"); norm(A, "f")
[1]  5
[1]  4
[1]  3.741657
> sum(diag(A))
[1]  3
```

horní trojúhelníková matice

Násobení matic

Pro násobení matic po složkách se používá operátor `*`, navíc musí mít násobené matice souhlasné rozměry. Pro součin matice a vektoru (v libovolném pořadí) se uplatňuje pravidlo *recycling rule*, tzn. jednotlivé složky matice jsou po sloupcích postupně násobeny složkami vektoru.

```
> u <- c(1, 0, 0)
> v <- c(1, 0, 0, 0, 1)
> A <- matrix(c(1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1), ncol=3)
> A
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    1    1
[4,]    1    1    1
> u * A   použití pravidla recycling rule
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    0    0
[3,]    0    1    0
[4,]    1    0    0
> v * A   použití pravidla recycling rule. Výstup navíc obsahuje varovné hlášení, neboť počet prvků matice A není dělitelný délkou vektoru v
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
[4,]    0    0    0
Warning message:
In v * A : longer object length is not a multiple of shorter object
length
```

Poznámka. Jazyk R nerozlišuje při násobení pomocí operátoru `*` pořadí matice a vektoru. Na následujících příkladech se můžeme přesvědčit, že operace `A * u` dává stejný výstup jako `u * A` a operace `A * v` dává stejný výstup jako `v * A`.

```
> A * u   použití pravidla recycling rule
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    0    0
[3,]    0    1    0
[4,]    1    0    0
```

```
> A * v
```
použití pravidla *recycling rule*. Výstup navíc obsahuje varovné hlášení, neboť počet prvků matice `A` není dělitelný délkou vektoru `v`

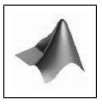
```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
[4,]    0    0    0
Warning message:
In A * v : longer object length is not a multiple of shorter object
length
```

Pro maticové násobení se používá operátor `%*%`, oba činitelé musí být odpovídajících rozměrů (vnitřní rozměry obou činitelů musí být shodné).

```
> A %*% u
     [,1]
[1,]    1
[2,]    0
[3,]    0
[4,]    1
> B <- matrix(c(0, 1, 1, 1, 0, 1, 0, 1), ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    0    1    0    0
[2,]    1    1    1    1
> A %*% B
```
nesouhlasné rozměry matic

```
Error in A %*% B : non-conformable arguments
> B %*% A
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    2    3    2
```

Zatímco v systému MATLAB operaci násobení po složkách zajišťuje operátor `.*`, v jazyce R je to `*`. Operátor `*` v systému MATLAB označuje maticové násobení, v jazyce R je to operátor `%*%`.



Řešení lineárních rovnic a inverze

Řešení lineárních rovnic je inverzní operací k maticovému násobení

```
> b <- A %*% x
```

`A` značí čtvercovou matici koeficientů pro lineární systém, `b` je vektor nebo matice pravé strany. Vektor/matice `x` je řešením systému lineárních rovnic $Ax = b$, které získáme příkazem `solve(A, b)`. V lineární algebře řešení formálně dostaneme $x = A^{-1}b$, kde A^{-1} značí matici inverzní k matici `A`. Matici inverzní můžeme v R spočítat pomocí `solve(A)`.

```
> (A <- matrix(c(1, 3, 1, -1), 2))
     [,1]  [,2]
[1,]    1     1
[2,]    3    -1
> (b <- matrix(c(3, 1), 2))
     [,1]
[1,]    3
[2,]    1
> solve(A, b)
     [,1]
[1,]    1
[2,]    2
> solve(A) %*% b
     [,1]
[1,]    1
[2,]    2
```

K řešení systémů lineárních rovnic slouží i funkce `backsolve()`:

`backsolve(A, b, k, upper.tri, transpose)` funkce pro řešení systémů lineárních rovnic s trojúhelníkovou maticí soustavy

`A`	matice soustavy
`b`	vektor/matice pravé strany
`k`	počet sloupců matice `A` a řádků `b`, který má být při výpočtu použit
`upper.tri`	systém s horní trojúhelníkovou maticí soustavy odpovídá implicitnímu nastavení `upper.tri=TRUE`, nastavení `upper.tri=FALSE` odpovídá systému s dolní trojúhelníkovou maticí
`transpose`	nastavením `transpose=TRUE` řešíme systém $A'x = b$, implicitní nastavení `transpose=FALSE`

```
> backsolve(A, b, upper.tri=TRUE)
     [,1]
[1,]    4
[2,]   -1
```

řeší systém lineárních rovnic $\left(\begin{array}{cc|c} 1 & 1 & 3 \\ 0 & -1 & 1 \end{array}\right)$

Poznámka. Stejně jako u maticového násobení, ani u funkcí `solve` a `backsolve` není nezbytně nutné zadávat vektory pravé strany jako sloupcové vektory. Pro vstupní řádkový vektor bude výstupem řádkový vektor.

Příklady k procvičení

1. Vytvořte matici `A1` s rozměry 3 x 5, jejíž první řádek bude tvořen posloupností -1, 1, 3, . . . , druhý řádek budou tvořit hodnoty 3 a poslední řádek ekvidistantní posloupnost s minimální hodnotou -2 a maximální hodnotou 2.

2. Vytvořte matici `A2` s rozměry 3 x 5, jejíž první sloupec bude tvořen náhodnými čísly z rovnoměrného rozložení na intervalu $[0, 5]$, druhý a třetí sloupec bude tvořit posloupnost s maximální hodnotou 3.5 a krokem 0.2 a poslední dva sloupce budou obsahovat hodnotu `NA`.

3. Vypište čtvrtý sloupec matice `A1` jako řádkový i sloupcový vektor.

4. Vypište prvky prvního a druhého řádku a všech lichých sloupců matice `A1`. Uložte je do matice `A3` a vhodným příkazem zjistěte její rozměry, použijte i funkci pro zjištění počtu řádků a sloupců zvlášť.

5. Pomocí funkcí `rbind` nebo `cbind` vhodně doplňte hodnotami `1` matici `A3` na čtvercovou matici a spočítejte její determinant, stopu a hodnost.

6. Byla pozorována kvalita říční vody. Z údajů v následující tabulce

		tvrdost	vodivost	koncentrace draslíku
lokalita 1	měření 1	19.4	0.832	1.9
	měření 2	18.9	0.84	2.4
lokalita 2	měření 1	21.2	0.826	2.1
	měření 2	17	0.479	1.6

   vytvořte pole `kvalita` s rozměry 2 x 3 x 2, dimenze, řádky i sloupce pojmenujte.

7. Vytvořte matici `A`, vektory `c` a `d`:
   $A = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}$, $c = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $d = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
   a matici `B` o rozměrech 3 x 6 s hodnotami rovnými 1.
   a) Řešte systém lineárních rovnic $Ax = c$.
   b) Vysvětlete výstupy příkazů `A * B`, `A %*% B`, `A * c`, `A * d`, `c * A`, `d * A` `A %*% c` and `A %*% d`
   c) Vhodnými příkazy vytvořte k matici `A` dolní trojúhelníkovou matici `C` a pro vektor pravé strany `e` $= \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}$ řešte soustavu lineárních rovnic $Cx = e$ s dolní trojúhelníkovou maticí `C`.

Řešení.

1. `A1 <- matrix(c(seq(from=-1, by=2, length=5), rep(3,5), seq(from=-2, to=2, length=5)), nrow=3, byrow=T)`

2. `A2 <- matrix(c(runif(n=3, min=0, max=5), seq(to=3.5, by=0.2, length=6), rep(NA, 6)), nrow=3)`

3. `A1[,4]`
   `A1[,4, drop=FALSE]`

4. `A3 <- A1[c(1,2), seq(from=1, by=2, to=ncol(A1))]`
   `dim(A3)`, `nrow(A3)`, `ncol(A3)`

5. `A3 <- rbind(A3, rep(1, ncol(A3)))`
   `det(A3)`, `sum(diag(A3))`, `qr(A3)$rank`

6. `kvalita <- array(c(19.4, 18.9, 0.832, 0.84, 1.9, 2.4, 21.2, 17, 0.826, 0.479, 2.1, 1.6), dim=c(2, 3, 2), dimnames=list(c("mereni1", "mereni2"), c("tvrdost", "vodivost", "koncentraceK"), c("lokalita1", "lokalita2")))`

7. `A <- matrix(c(1, -1, 0, 0, 1, -1, 1, 1, 1), 3)`, `B <- matrix(1, nrow=3, ncol=6)`, `c <- matrix(1:3, ncol=1)`, `d <- matrix(1:2, ncol=1)`
   a) `x <- solve(A, c)`
   b) `A * B`, `A * c`, `A * d` ... násobení po složkách (neodpovídající rozměry),
   `c * A`, `d * A` ... násobení po složkách, uplatněno pravidlo *recycling rule*
   `A %*% B`, `A %*% c`, `A %*% d` ... maticové násobení,
   c) `e <- matrix(c(3, 0, 1), nrow=3)`, `backsolve(A, e, upper.tri=FALSE)`
   nebo pomocí konstrukce dolní trojúhelníkové matice `C`:
   `C <- A`, `C[!lower.tri(A,1)] <- 0`, `solve(C, e)`

Kapitola 4

Datové tabulky a seznamy

Základní informace

Datové tabulky jsou datové struktury sloužící k uchování souboru dat, které nachází své uplatnění především při statistickém zpracování dat. Jedná se tabulku dat, jejíž sloupce představují proměnné (pozorované znaky) často různých datových typů a řádky představují jednotlivá pozorování.
Dalšími využívanými datovými strukturami jsou seznamy. Jedná se o objekty, které dokáží uchovávat množství informací různých datových struktur. Jejich využití spočívá ve smysluplném uspořádání např. vstupních a výstupních informací provedené statistické analýzy do jediného objektu.
V této kapitole se naučíme vytvářet datové tabulky a seznamy a pracovat s nimi.

Výstupy z výuky

Studenti

- se seznámí s datovými tabulkami a seznamy, znají rozdíly mezi nimi a umí je vytvářet,
- umí několika způsoby vytvářet podmnožiny datových tabulek a seznamů,
- dokáží používat funkce pro manipulaci s datovými tabulkami a seznamy.

4.1 Základní příkazy, tvorba datových tabulek a seznamů

Datová tabulka je 2-dimenzionální struktura, která slouží k uchování souboru dat. Soubor dat se skládá z množiny proměnných (sloupce), které jsou pozorovány na množství

případů (řádky). Jednotlivé sloupce mohou být různých datových typů, ale prvky každého sloupce musí být stejného datového typu. Počet případů musí být pro každou proměnnou stejný.

Seznam je nejobecnější datová struktura v R, která seskupuje několik (různých) objektů do objektu většího rozsahu. Jedná se o datovou strukturu skládající se z posloupnosti objektů, kterým se říká složky. Každá složka může obsahovat objekt jakéhokoliv datového typu. Seznam tedy může obsahovat vektory různých datových typů a délek, matice, pole, datové tabulky, funkce a/nebo jiné seznamy. Proto jsou seznamy vhodnými výstupy nejrůznějších výpočtů.

Rovněž je důležité si uvědomit, že datová tabulka je speciálním případem seznamu. Nejedná se o nic jiného, než seznam, jehož složky jsou vektory stejné délky a odpovídající pozice vyjadřují stejné případy (např. výskyt aut červené a modré barvy během středy).

Datovou tabulku vytvoříme příkazem `data.frame()`. Pomocí přiřazení názvů vektorům v seznamu argumentů `nazev_1=vektor_1, nazev_2=vektor_2, ...` můžeme specifikovat názvy sloupců (proměnných) a jejich hodnoty. Názvy sloupců jsou nepovinné, stačí zadat pouze hodnoty - v tomto případě jsou názvy proměnných zvoleny implicitně na základě pravidel stanovených jazykem R. Argument `row.names` (implicitní nastavení `NULL` případy čísluje) specifikuje názvy případů. Argument `check.names` s implicitním nastavením `TRUE` kontroluje, zda jsou názvy proměnných syntakticky správné a zda se neopakují, v případě duplikací se stará o jejich přejmenování.

```
> data.frame(obor=factor(c(1, 0, 0, 1, 1), labels=c("OM", "MAEK")),
+ body=c(18, 13, 15, 20, 15))
    obor  body
 1  MAEK    18
 2    OM    13
 3    OM    15
 4  MAEK    20
 5  MAEK    15
> data.frame(obor=factor(c(1, 0, 0, 1, 1), labels=c("OM", "MAEK")),
+ body=c(18, 13, 15, 20, 15), row.names=c("Petr", "Pavel", "Jirina",
+ "Adela", "Matej"))
        obor  body
   Petr  MAEK    18
  Pavel    OM    13
 Jirina    OM    15
  Adela  MAEK    20
  Matej  MAEK    15
```

```
> data.frame(a=c(1,2), a=c(T,F), check.names=T)
  a    a.1
1 1   TRUE
2 2  FALSE
> data.frame(a=c(1,2), a=c(T,F), check.names=F)
  a     a
1 1  TRUE
2 2 FALSE
```

Funkce `list()` slouží k vytvoření seznamu. Stejně jako u funkce `data.frame()` mohou být i složky seznamu pojmenovány pomocí argumentů `nazev_1=slozka_1, nazev_2=slozka_2, ...`.

```
> (l <- list(barva=c("cervena", "modra", "bila"), data.frame(Petr=
+ sample(5, replace=T), Pavel=1:5, row.names=c("po","ut", "st", "ct",
+ "pa"))))
$barva
[1] "cervena" "modra"  "bila"

[[3]]
   Petr Pavel
po    1     1
ut    3     2
st    3     3
ct    5     4
pa    3     5
```

Funkce `dim()`, `names()` a `contents()` slouží k výpisu vlastností datové tabulky. Funkce `dim()` vypisuje rozměry datové tabulky, funkce `names()` zobrazuje názvy proměnných. Funkce `contents()` vrací vnitřní strukturu datové tabulky. Funkce je obsažena v balíčku `Hmisc`, který není ve standardní distribuci, je třeba jej proto doinstalovat příkazem `install.packages("Hmisc")` a načíst příkazem `library(Hmisc)`.

K výpisu vlastností seznamu můžeme použít funkci `names()`, která vrací názvy složek seznamu. Funkce `dim()` a `contents()` u seznamu použít nemůžeme, můžeme je ovšem nahradit funkcemi `length()`, která vrací počet složek seznamu, a `str()`, která vypisuje vnitřní strukturu seznamu.

4.2 Podmnožiny datových tabulek a seznamů

K vypsání podmnožiny seznamu můžeme použít jednoduchých `[]` nebo dvojitých `[[]]` hranatých závorek. Jednoduchými závorkami uvádíme, které složky seznamu chceme získat. Jestliže jednotlivé složky seznamu nejsou pojmenovány, požadovanou složku spe-

cifikujeme jejím číslem. K výpisu více složek můžeme použít operátoru : nebo funkce `c()`. Jestliže jsou složky seznamu pojmenovány, požadované prvky specifikujeme jejich názvy v uvozovkách. Podmnožina seznamu vytvořená pomocí jednoduchých hranatých závorek je opět typu seznam.

Naopak, příkaz pro vytváření podmnožiny pomocí dvojitých hranatých závorek vrací objekt takového typu, jakým byl při definování seznamu. V tomto případě je na každou složku odkazováno jednotlivě, nepoužívá se proto operátor : ani funkce `c()`. Stejně jako u jednoduchých hranatých závorek, na každou složku je odkazováno jejím číslem, má-li požadovaná složka název, můžeme na ni odkazovat jejím názvem v uvozovkách nebo můžeme použít operátoru `$`.

```
> l <- list(barva=c("cervena", "modra", "bila"), matrix(1:4, 2),
+ data.frame(Petr=sample(5, replace=T), Pavel=1:5, row.names=c("po",
+ "ut", "st", "ct", "pa")))
> l["barva"]
$barva
[1]  "cervena"  "modra"  "bila"
> typeof(l["barva"])   příkaz typeof() určí typ svého argumentu
[1]  "list"
> l[[2]]
      [,1]  [,2]
 [1,]    1     3
 [2,]    2     4
> typeof(l[[2]])
[1]  "integer"
> typeof(l$barva)
[1]  "character"
```

Protože datové tabulky jsou speciálním případem seznamů, řádky a/nebo sloupce tabulky dat mohou být získány pomocí `[]`, `[[]]` a/nebo operátoru `$`. Datové tabulky mohou být považovány i za zobecněné matice, k vytvoření podmnožiny můžeme proto použít `[,]`.

```
> tab <- data.frame(cervena=c(1,2,3), modra=c(3,6,0), bila=c(2,5,1),
+ row.names=c("pondeli", "utery", "streda"))
> tab[1]
         cervena
 pondeli       1
 utery         2
 streda        3
> typeof(tab[1])
[1]  "list"
```

```
> tab[[1]]   ekvivalentní příkaz příkazům tab[["cervena"]] a tab$cervena
[1]  1   2   3
> typeof(tab[[1]]); typeof(tab[["cervena"]]; typeof(tab$cervena))
[1]  "double"
[1]  "double"
[1]  "double"
> tab["utery", "bila"]
[1]  5
```

K výběru podmnožiny datové tabulky slouží i příkaz `subset(x, )`. Argument `x` specifikuje název datové tabulky, z níž podmnožinu vybíráme. Argument `subset` specifikuje řádky vyhovující dané podmínce, přičemž hodnoty `NA` jsou brány jako `FALSE`. Argument `select` specifikuje sloupce, které chceme vypsat, můžeme použít funkce `c()`, operátoru `:` i operátoru `-` pro vynechání složek.

```
> subset(tab, subset=cervena==3, select=c(modra, bila))
        modra  bila
 streda     0     1
```

Funkce `subset()` vždy vrací tabulku dat, i když má jen jeden řádek nebo sloupec. K tomu, aby vrátila jen jednoduchý vektor, musíme za vlastní definici podmnožiny datové tabulky použít operátor `$`, za nímž následuje název sloupce:

```
> tab["pondeli", "cervena"]
[1]  1
> subset(tab, subset=cervena==1, select=cervena)
         cervena
 pondeli       1
> subset(tab, subset=cervena==1, select=cervena)$cervena
[1]  1
```

4.3 Funkce pro manipulaci s datovými tabulkami a seznamy

Přidání dalších sloupců

Prvním způsobem, jak do datové tabulky přidat nový sloupec s hodnotami, je příkaz tvaru

`datová_tabulka$nazev_noveho_sloupce <- hodnoty`.

Druhým způsobem je provést přiřazení pomocí funkce `data.frame()` (bez přiřazení zobrazuje nové hodnoty pouze dočasně).

```
> (knihy <- data.frame(nazev=c("Dekameron", "Maj", "Temno", "Bidnici",
+ "Babicka"), autor=c("Boccaccio", "Macha", "Jirasek", "Hugo",
+ "Nemcova")))
       nazev      autor
 1  Dekameron  Boccaccio
 2        Maj      Macha
 3      Temno    Jirasek
 4    Bidnici       Hugo
 5    Babicka    Nemcova
> knihy$pocet <- c(3, 6, 4, 3, 5)
> knihy
       nazev      autor  pocet
 1  Dekameron  Boccaccio      3
 2        Maj      Macha      6
 3      Temno    Jirasek      4
 4    Bidnici       Hugo      3
 5    Babicka    Nemcova      5
> (knihy <- data.frame(knihy, k_dispozici=c(F, F, T, F, T)))
       nazev      autor  pocet  k_dispozici
 1  Dekameron  Boccaccio      3        FALSE
 2        Maj      Macha      6        FALSE
 3      Temno    Jirasek      4         TRUE
 4    Bidnici       Hugo      3        FALSE
 5    Babicka    Nemcova      5         TRUE
```

Funkce `transform()` pouze tiskne aktuální datovou tabulku, nepřidává nastálo novou proměnnou (v opačném případě musíme provést přiřazení).

```
> transform(knihy, rok=c(1971, 1997, 1983, 2003, 1992))
```

přidá sloupec `rok` s danými hodnotami, proměnná `knihy` ovšem zůstane nezměněna

```
       nazev      autor  pocet  k_dispozici   rok
 1  Dekameron  Boccaccio      3        FALSE  1971
 2        Maj      Macha      6        FALSE  1997
 3      Temno    Jirasek      4         TRUE  1983
 4    Bidnici       Hugo      3        FALSE  2003
 5    Babicka    Nemcova      5         TRUE  1992
```

Pro přidání dalších složek do seznamu lze pomocí operátoru přiřazení, a to jednou z možností

```
seznam$nova_slozka <- objekt
seznam[["nova_slozka"]] <- objekt
seznam["nova_slozka"] <- objekt.
```

```
> (vyzkum <- list(n=28, lokalita="Brno"))
$n
[1]  28

$lokalita
[1]  "Brno"

> vyzkum$obdobi <- 2004:2012
> vyzkum[["jakost"]] <- factor(c(0, 1, 1, 1, 0, 0, 1, 0, 1, 1), labels=
+ c("1", "2"))
> vyzkum
$n
[1]  28

$lokalita
[1]  "Brno"

$obdobi
[1]  2004   2005   2006   2007   2008   2009   2010   2011   2012

$jakost
[1]  1   2   2   2   1   1   2   1   2   2

Levels: 1 2
```

Odstraňování řádků a sloupců

Každý sloupec můžeme z datové tabulky odstranit použitím operátoru `$` nebo `[]` nastavením na hodnotu `NULL`. Dalším způsobem je použití operátoru `[,]` k výběru podmnožiny datové tabulky a přepsání původní proměnné.

```
> knihy
       nazev      autor  pocet  k_dispozici
 1  Dekameron  Boccaccio      3        FALSE
 2        Maj      Macha      6        FALSE
 3      Temno    Jirasek      4         TRUE
 4    Bidnici       Hugo      3        FALSE
 5    Babicka    Nemcova      5         TRUE
```

`> knihy$pocet <- NULL` analogický příkaz: `knihy["pocet"] <- NULL`. O tom, že sloupce byly opravdu odstraněny, se můžeme přesvědčit výpisem názvů sloupců (funkce `names()`)

```
> names(knihy)
[1]  "nazev"  "autor"  "k_dispozici"
```

```
> (knihy <- knihy[1:3, c("nazev", "k_dispozici")])

       nazev  k_dispozici
 1  Dekameron        FALSE
 2        Maj        FALSE
 3      Temno         TRUE
```

Slučování

Ke sloučení datových tabulek můžeme použít funkce `cbind()` nebo `rbind()`, které je sloučí po sloupcích nebo řádcích. Při použití funkce `cbind()` musí mít přidávané sloupce stejný počet řádků jako v již existující datové tabulce. Rovněž je vhodné se přesvědčit, že přidávané sloupce mají stejné pořadí řádků. Při použití funkce `rbind()` je třeba dodržovat stejný počet sloupců a jejich shodné názvy.

Poznámka. Při spojování datové tabulky a vektoru s různými rozměry dochází k chybovému hlášení, ovšem při spojování matice a vektoru o různých rozměrech je operace provedena použitím pravidla *recycling rule* a doplněna o varovné hlášení.

```
> zbozi1 <- data.frame(ID=c("X1", "X2", "X3"), cena=c(99, 109, 99),
+ sklad=c(T, T, F))
   ID  cena  sklad
 1  X1    99   TRUE
 2  X2   109   TRUE
 3  X3    99  FALSE
> zbozi2 <- data.frame(ID=c("X4", "X5"), cena=c(99, 89), sklad=c(F, T))
   ID  cena  sklad
 1  X4    99  FALSE
 2  X5    89   TRUE
> rbind(zbozi1, zbozi2)
   ID  cena  sklad
 1  X1    99   TRUE
 2  X2   109   TRUE
 3  X3    99  FALSE
 4  X4    99  FALSE
 5  X5    89   TRUE
> zbozi3 <- data.frame(ID=c("X4", "X5"), sklad=c(F, T), cena=c(99, 89))
   ID  sklad  cena
 1  X4  FALSE    99
 2  X5   TRUE    89
> rbind(zbozi1, zbozi3)   tabulky mají odlišné názvy sloupců, nelze je spojit
Error in match.names(clabs, names(xi)) :
  names do not match previous names
```

```
> zbozi4 <- data.frame(ID=c("X1", "X2", "X4"), barva=c("cerna", "bila",
+ "bila"))
   ID  barva
1  X1  cerna
2  X2   bila
3  X4   bila
> cbind(zbozi1, zbozi4)
```

tabulky mají stejný počet řádků, dojde proto k jejich spojení (ovšem bez ohledu na to, že je v každé z nich uveden sloupec `ID` s různými hodnotami. Pokud bychom chtěli tomu chování předejít, musíme použít funkci `merge()`.)

```
   ID  cena  sklad  ID  barva
1  X1    99   TRUE  X1  cerna
2  X2   109   TRUE  X2   bila
3  X3    99  FALSE  X4   bila
```

Alternativou k funkcím `cbind()` a `rbind()` může být již zmíněná funkce `merge()`.

`merge(x, y, by, by.x, by.y, all, all.x, all.y)` sloučí datové tabulky `x` a `y`

`by`, `by.x`, `by.y` specifikují ty názvy sloupců, podle kterých mají být tabulky sloučeny. V případě stejného názvu sloupců použijeme argument `by`. Jsou-li názvy sloupců ke sloučení různé, specifikujeme je pomocí argumentů `by.x` a `by.y`. V případě více podmínek na spojení podmínky uvádíme ve formě vektoru typu řetězec.

`all`, `all.x` a `all.y` specifikace těch řádků, které se mají objevit na výstupu. Implicitní nastavení `all=FALSE` vrací pouze řádky z průniku obou tabulek, `all=TRUE` vrací řádky ze sjednocení obou tabulek. `all.x=TRUE` vrací všechny řádky tabulky `x`, analogicky je tomu u `all.y=TRUE`

Poznámka. Při použití některého z argumentů `all=TRUE`, `all.x=TRUE` nebo `all.y=TRUE` mají „volná“ místa, která vznikla spojením, hodnotu `NA`.

```
> (tab1 <- data.frame(auto=c("fiat", "opel", "skoda", "bmw"), barva=
+ c("seda", "cervena", "cerna", "modra"), rok=c(2003, 1999, 2008,
+ 2004)))
   auto    barva   rok
1  fiat     seda  2003
2  opel  cervena  1999
3 skoda    cerna  2008
4   bmw    modra  2004
> (tab2 <- data.frame(znacka=c("saab", "bmw", "audi"), majitel=c("muz",
+ "zena", "zena")))
  znacka  majitel
1   saab      muz
2    bmw     zena
3   audi     zena
```

```
> merge(tab1, tab2)
```

protože tabulky neobsahují stejný název sloupce, podle kterého by se sloučení mělo řídit, R vytvoří kartézský součin obou tabulek (tzn. ke každému řádku `tab1` se připojí každý řádek `tab2`)

```
      auto    barva   rok  znacka  majitel
 1    fiat     seda  2003    saab      muz
 2    opel  cervena  1999    saab      muz
 3   skoda    cerna  2008    saab      muz
 4     bmw    modra  2004    saab      muz
 5    fiat     seda  2003     bmw     zena
 6    opel  cervena  1999     bmw     zena
 7   skoda    cerna  2008     bmw     zena
 8     bmw    modra  2004     bmw     zena
 9    fiat     seda  2003    audi     zena
 10   opel  cervena  1999    audi     zena
 11  skoda    cerna  2008    audi     zena
 12    bmw    modra  2004    audi     zena
```

```
> merge(tab1, tab2, by.x="auto", by.y="znacka")
```

spojení na základě výskytu `bmw` v obou sloupcích `auto` i `znacka`

```
    auto  barva   rok  majitel
 1   bmw  modra  2004     zena
```

```
> merge(tab1, tab2, by.x="auto", by.y="znacka", all.x=T)
```

argument `all.x=T` zajistil, aby výstup obsahoval všechny hodnoty z `tab1`

```
     auto    barva   rok  majitel
 1    bmw    modra  2004     zena
 2   fiat     seda  2003     <NA>
 3   opel  cervena  1999     <NA>
 4  skoda    cerna  2008     <NA>
```

Řazení

K seřazení dat v datové tabulce se používá funkce `order()`, která vrací vektor indexů vzestupně (implicitní nastavení `decrerasing=FALSE`) nebo sestupně (nastavení `decreasing=TRUE`) uspořádaných prvků. Více informací o funkci `order` je uvedeno v odstavci 5.4.

```
> knihy
       nazev      autor  pocet  k_dispozici
 1  Dekameron  Boccaccio      3        FALSE
 2        Maj      Macha      6        FALSE
 3      Temno    Jirasek      4         TRUE
 4    Bidnici       Hugo      3        FALSE
 5    Babicka    Nemcova      5         TRUE
```

```
> knihy[order(knihy$k_dispozici, knihy$nazev),]
```
seřazení podle sloupce `k_dispozici`, v případě více stejných hodnot řazení podle sloupce `nazev`

```
       nazev      autor  pocet  k_dispozici
4    Bidnici       Hugo      3        FALSE
1  Dekameron  Boccaccio      3        FALSE
2        Maj      Macha      6        FALSE
5    Babicka    Nemcova      5         TRUE
3      Temno    Jirasek      4         TRUE
```

Pokud některý ze sloupců tabulky tvoří numerický vektor, pro sestupné uspořádání této proměnné můžeme na místo argumentu `decreasing=TRUE` použít operátoru `-`. V případě nejednoznačných podmínek na seřazení (řádky se stejnými hodnotami pro seřazení) dostává na výpisu přednost řádek s nižším pořadovým číslem, u názvů řádků se postupuje podle abecedního uspořádání.

```
> knihy[order(knihy$k_dispozici, -knihy$pocet),]
```
vzestupné seřazení proměnné `k_dispozici` a sestupné seřazení proměnné `knihy`

```
       nazev      autor  pocet  k_dispozici
2        Maj      Macha      6        FALSE
1  Dekameron  Boccaccio      3        FALSE
4    Bidnici       Hugo      3        FALSE
5    Babicka    Nemcova      5         TRUE
3      Temno    Jirasek      4         TRUE
```

Příklady k procvičení

1. Následující tabulka popisuje 10 pacientů s podezřením na krátkozrakost (myopii). Tabulku přepište do systému R jako datovou tabulku `myopie1`, hodnoty pro pohlaví a myopii rodičů zadávejte jako faktor.

ID	pohlaví	délka studie	myopie rodičů
101	muž	3	ano
102	žena	5	ne
103	žena	2	ne
104	muž	4	ano
105	žena	4	ano
106	muž	3	ano
107	muž	2	ne
108	muž	1	ano
109	žena	7	ano
110	muž	5	ano

2. Přidáním nového sloupce pro délku oční bulvy `delka` s hodnotami 20.4, 22.7, 21.3, 24.56, 20.9, 21.8, 23.5, 23.9, 19.9, 22.6 vytvořte z tabulky `myopie1` tabulku `myopie`.
   a) Z tabulky `myopie` odstraňte všechny sudé řádky.
   b) Z tabulky `myopie` odstraňte sloupec "myopie rodičů". Zjistěte rozměry nově vzniklé tabulky.
   c) Vypište "ID"a "pohlaví"pacientů z tabulky `myopie1` s délkou studie větší než čtyři roky.
   d) Vypište tabulku `myopie` seřazenou podle délky studie pacientů, popř. dle délky oční bulvy.

3. Následující tabulku definujte jako datovou tabulku `myopie2`:

ID	barva očí	věk
101	modrá	8
102	zelená	11
103	hnědá	9
104	hnědá	14
105	modrá	9
106	šedá	13
107	zelená	9
108	modrá	8
109	modrá	12
110	hnědá	13

   Poté slučte tabulky `myopie1` a `myopie2` na základě "ID". Vyzkoušejte sloučení i pomocí funkce `cbind`.

4. Vytvořte seznam `CR` se složkami
   "země": Čechy, Morava, Slezsko,
   "města": Praha, Brno, Ostrava, Plzeň, Olomouc,
   "pohoří":

pohoří	vrchol	nadmořská výška
Krkonoše	Sněžka	1602
Jeseníky	Praděd	1492
Šumava	Plechý	1378
Beskydy	Lysá hora	1323
Krušné hory	Klínovec	1244

   a) Vypište druhou složku seznamu `CR` ve formě seznamu. Rovněž zjistěte, jakého je tato složka datového typu.

b) Přidejte novou složku "sousedé": Německo, Rakousko, Slovensko, Polsko.
c) Smažte složku "země"a pomocí vhodné funkce zjistěte počet složek seznamu `CR`.
d) Spočítejte průměrnou nadmořskou výšku pro uvedené vrcholy. (Pro výpočet průměru použijte funkci `mean()`).

Řešení.

1. `myopie1 <- data.frame(ID=101:110, pohlavi=factor(c(0, 1, 1, 0, 1, 0, 0, 0, 1, 0), labels=c("muz", "zena"), levels=c(0, 1)), delka.studie=c(3, 5, 2, 4, 4, 3, 2, 1, 7, 5), myopie.rodic=factor(c(1, 0, 0, 1, 1, 1, 0, 1, 1, 1), labels=c("ne", "ano"), levels=c(0, 1)))`

2. `myopie <- data.frame(myopie1, delka=c(20.4, 22.7, 21.3, 24.56, 20.9, 21.8, 23.5, 23.9, 19.9, 22.6))`
   a) `myopie <- myopie[-seq(from=2, by=2, to=nrow(myopie)),]`
   b) `myopie[,"myopie.rodic"] <- NULL`, `dim(myopie)`
   c) `subset(myopie1, select=c(ID, pohlavi), subset=delka.studie>4)`
   d) `myopie[order(myopie$delka.studie, myopie$delka),]`

3. `myopie2 <- data.frame(ID=101:110, barva.oci=c("modra", "zelena", "hneda", "hneda", "modra", "seda", "zelena", "modra", "modra", "hneda"), vek=c(8, 11, 9, 14, 9, 13, 9, 8, 12, 13))`
   `merge(myopie1, myopie2, by="ID")`
   `cbind(myopie1, myopie2)`

4. `CR <- list(zeme=c("Cechy", "Morava", "Slezsko"), mesta=c("Praha", "Brno", "Ostrava", "Plzen", "Olomouc"), pohori=data.frame(pohori= c("Krkonose", "Jeseniky", "Sumava", "Beskydy", "Krusne hory"), vrchol=c("Snezka", "Praded", "Plechy", "Lysa hora", "Klinovec"), nadm.vyska=c(1602, 1492, 1378, 1323, 1244)))`
   a) `CR[2]` nebo `CR["mesta"]`, `typeof(CR[2])`
   `CR[[2]]` nebo `CR[["mesta"]]` nebo `CR$mesta`, `typeof(CR[[2]])`
   b) `CR$sousede <- c("Nemecko", "Rakousko", "Slovensko", "Polsko")` nebo `CR[["sousede"]] <- c("Nemecko", "Rakousko", "Slovensko", "Polsko")`
   c) `CR$zeme <- NULL` nebo `CR["zeme"] <- NULL`
   `length(CR)`
   d) `mean((CR$pohori))[,3]`

Kapitola 5

Konstanty, operátory a matematické výpočty

Základní informace

Kapitola je věnována aritmetickým, porovnávacím, logickým a množinovým operátorům, jejichž znalost je základním předpokladem programovacích schopností. Dále jsou uvedeny matematické funkce, základní statistické funkce a funkce pro zaokrouhlování. Někdy je pro zjednodušení práce velmi výhodné pracovat s vestavěnými konstantami, které jsou uvedeny na konci kapitoly.

Výstupy z výuky

Studenti

- ovládají aritmetické operátory, zopakují si rozdíly mezi maticovým násobením a násobením po složkách,
- používají porovnávací a logické operátory,
- se seznámí s množinovými operátory,
- znají a umí vhodně použít matematické funkce,
- ovládají funkce pro zaokrouhlování a dokáží využívat zabudovaných konstant.

5.1 Aritmetické operátory

`+`	sčítání
`-`	odčítání
`*`	násobení
`/`	dělení
`^`	umocňování
`%*%`	maticové násobení
`%%`	zbytek po celočíselném dělení (modulo)
`%/%`	celá část z celočíselného dělení
`t()`	transpozice matice nebo datové tabulky

Transpozicí (řádkového) vektoru je v jazyce R stále (řádkový) vektor. Transpozicí řádkového vektoru je v systému MATLAB sloupcový vektor.

```
> a <- c(3, 5, 7, 8); b <- c(1, 2, 3); c <- c(4, 1, 8)
> a + b
[1]  4   7   10   9
 Warning message:
In a + b : longer object length is not a multiple of shorter object
length
```

Protože vektor `b` je menší délky než vektor `a`, je potřeba jeho délku zvětšit o jednu pozici. (Pravidlo pro postupné opakování složek do požadované délky se nazývá *recycling rule*.) Poslední složka výstupního vektoru je tedy součtem 4. složky vektoru `a` a 1. složky vektoru `b`. Na stejném principu fungují i všechny ostatní aritmetické operátory.

MATLAB by v tomto případě hlásil chybu a výpočet by neprovedl, protože sčítance nejsou stejné dimenze.

```
> "a"+ "b"
Error in "a"+ "b": non-numeric argument to binary operator
```

Binární operátory můžeme použít pouze na numerické argumenty. Na rozdíl od R, v MATLABu je můžeme použít i na textové řetězce, které jsou převedeny na odpovídající kód v ASCII tabulce a následně provedena příslušná operace.

```
> b <- 1:3
> c <- c(4, 1, 8)
> b / c
[1]  0.250  2.000  0.375
> b %/% c
[1]  0  2  0
> b %% c
[1]  1  0  3
```

5.2 Porovnávací a logické operátory

Porovnávací operátory slouží k porovnávání odpovídajících si složek vektorů. Na výstupu dostáváme vektor logických hodnot `TRUE` a `FALSE` délky nejdelšího vektoru na vstupu. Hodnoty `TRUE` obsazují ty pozice, které splňují danou podmínku, ostatní pozice jsou vyplněny hodnotami `FALSE`.

`==`	rovno
`!=`	není rovno
`<, <=`	menší, menší nebo rovno
`>, >=`	větší, větší nebo rovno
`&`	logické a
`|`	logické nebo
`!`	negace

```
> a <- c(3, 5, 7)
> b <- c(1, 2, 3)
> c <- 1:4
> a <= b
[1]  FALSE  FALSE  FALSE
> a >= 3 & b <= 2
[1]  TRUE  TRUE  FALSE
> !(a == 5 | a == b)
[1]  TRUE  FALSE  TRUE
```

Při porovnání vektorů o různých délkách je uplatněno pravidlo *recycling rule*:

```
> b == c
[1]  TRUE  TRUE  TRUE  FALSE
 Warning message:
In b == c : longer object length is not a multiple of shorter object
length
```

5.3 Množinové operátory

`all(relace)`	testuje, zda jsou všechny složky `relace` pravdivé
`any(relace)`	testuje, zda je alespoň jedna složka `relace` pravdivá
`which(relace)`	vrací pořadí těch složek `relace`, které jsou pravdivé (popř. které splňují danou podmínku). V případě polí můžeme použít argument `arr.ind=T` pro výpis hodnot v podobě čísel řádků a sloupců, popř. dalších dimenzí
`x %in% y`, `is.element(x, y)`	testuje, zda je `x` podmnožinou množiny `y`, vrací logické hodnoty
`intersect(x, y)`	průnik množin `x` a `y`
`union(x, y)`	sjednocení množin `x` a `y`
`setdiff(x, y)`	rozdíl množin, vrací ty prvky vektoru `x`, které nejsou obsaženy v `y`

```
> x <- 1:10
> y <- 2:9
> z <- -5:5
> all(z)    testuje, zda nabývají všechny prvky vektoru z hodnoty TRUE, v případě
numerických vektorů odpovídají hodnotě TRUE nenulové hodnoty
[1]  FALSE
> all(z > -10)    testuje, zda jsou všechny prvky vektoru z větší než −10
[1]  TRUE
> any(z)
[1]  TRUE
> all(y) >= x[5]
[1]  FALSE
> any(x) == any(y)
[1]  TRUE
> m <- matrix(2:10, c(3, 3))
      [,1]  [,2]  [,3]
 [1,]    2     5     8
 [2,]    3     6     9
 [3,]    4     7    10
> which(m > 8 | m == 3)
[1]  2   8   9
> which(m > 8 | m == 3, arr.ind=T)
      row  col
 [1,]   2    1
 [2,]   2    3
 [3,]   3    3
```

```
> x %in% y
[1]  FALSE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE
[10]  FALSE
> is.element(y, x)
[1]  TRUE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE
> union(x[1:5], y[c(7, 8)])
[1]  1   2   3   4   5   8   9
> intersect(x[1:5], y[c(1, 2, 3)])
[1]  2   3   4
> setdiff(x, y)
[1]  1   10
> setdiff(y, x)
integer(0)
```

5.4 Matematické funkce

Předpokládejme, že objekt `x` je numerický, komplexní, logický vektor nebo pole, operace jsou prováděny po složkách.

`abs(x)`, `sqrt(x)` absolutní hodnota a druhá odmocnina objektu `x`
`sign(x)` signum objektu `x`

Logaritmické a exponenciální funkce:
`log(x)`, `log10(x)`, `log2(x)` přirozený logaritmus, logaritmus se základem 10 a 2
`log(x, base)` logaritmus se základem `base`
`exp(x)` exponenciální funkce `x`

Trigonometrické a hyperbolické funkce:
`cos(x)`, `sin(x)`, `tan(x)`, `cosh(x)`, `sinh(x)`, `tanh(x)`

Inverzní trigonometrické a hyperbolické funkce:
`acos(x)`, `asin(x)`, `atan(x)`, `acosh(x)`, `asinh(x)`, `atanh(x)`

`gamma(x)` gamma funkce pro kladná reálná čísla `x`
`choose(n, k)` kombinační číslo $\binom{n}{k}$ pro každé reálné `n` a přirozené číslo `k`, `n`$\geq$`k`
`factorial(x)` faktoriál `x`

`max(x)`, `min(x)`, `sum(x)`, `prod(x)` vrací maximální a minimální prvek, součet a součin prvků argumentu `x`. Pro objekty typu matice nebo pole uvedené funkce vrací na

výstup jedinou hodnotu ze všech prvků. Pokud budeme požadovat provedení operací po složkách, je potřeba použít některou z funkcí skupiny `apply()`, více o nich naleznete v odstavci 8.5.

MATLAB aplikuje funkce `max()`, `min()`, `sum()` a `prod()` na matice nikoliv po prvcích, ale po sloupcích.

`cummax(x), cummin(x), cumsum(x), cumprod(x)` vrací vektor, jehož složkami jsou maximum, minimum, kumulativní součet a součin prvků argumentu `x`

```
> a <- c(1, 2, 3, 5, 8, 2, 4, 1, 2, 2)
> cumsum(a)
[1]  1   3   6   11   19   21   25   26   28   30
> cummin(a)   vždy nerostoucí posloupnost prvků
[1]  1   1   1   1   1   1   1   1   1   1
> cummax(a)   vždy neklesající posloupnost prvků
[1]  1   2   3   5   8   8   8   8   8   8
```

`range(x)` vektor obsahující minimum a maximum objektu `x`, `range(x)` je ekvivalentní příkazu `c(min(x), max(x))`
`mean(x), median(x)` průměr a medián objektu `x`
`sd(x)` směrodatná odchylka objektu `x`, v případě, že `x` je matice, `sd(x)` vrací směrodatnou odchylku jejích sloupců
`var(x), cov(x, y), cor(x, y)` rozptyl `x`, kovariance a korelace vektorů `x`, `y`, v případě, že `x`, `y` jsou matice, kovariance a korelace jsou počítány mezi sloupci `x` a `y`
`quantile(x)` generická funkce, vrací minimum, dolní kvartil, medián, horní kvartil a maximum objektu `x`

```
> c(min(a), max(a)); range(a)
[1]  1   8
[1]  1   8
> mean(a)
[1]  3
> var(a)
[1]  4.666667
> quantile(a)
   0%   25%   50%   75%  100%
 1.00  2.00  2.00  3.75  8.00
```

`pmax(x, y, z, ...), pmin(x, y, z, ...)` objekt maximálních/minimálních prvků na odpovídajících si pozicích

```
> b <- 1:10
> pmin(a, b)
[1]  1   2   3   4   5   2   4   1   2   2
```

Pro pole o 2 a více dimenzích s numerickými, komplexními nebo logickými hodnotami nebo pro datové tabulky můžeme použít následující funkce, které vrací průměry, součty a rozptyly po sloupcích či řádcích:
`colMeans(x)`, `colSums(x)`, `colVars(x)`
`rowMeans(x)`, `rowSums(x)`, `rowVars(x)`

```
> (e <- matrix(a,5))
      [,1]  [,2]
 [1,]    1     2
 [2,]    2     4
 [3,]    3     1
 [4,]    5     2
 [5,]    8     2
> rowSums(e)
[1]  3   6   4   7   10
```

`sort(x, decreasing, na.last, index.return)` seřazení objektu `x`
`decreasing=F` vzestupné pořadí (implicitní nastavení), `decreasing=T` sestupné pořadí,
`na.last=NA` zajišťuje vynechání hodnot `NA` (implicitní nastavení), `na.last=T` zajišťuje, aby hodnoty `NA` byly řazeny na konec, `na.last=F` řadí hodnoty `NA` na začátek
`index.return=T` vypíše i původní pořadí hodnot

`order(x, decreasing, na.last)` vypíše indexy seřazených hodnot
`na.last=T` řadí hodnoty `NA` na konec (implicitní nastavení), `na.last=F` řadí hodnoty `NA` na začátek, `na.last=NA` hodnoty `NA` vynechává

`rank(x, na.last, ties.method)` vypíše pořadí jednotlivých hodnot odpovídajících vzestupně seřazenému vektoru `x`
`na.last` stejně jako u funkce `order`
`ties.method` nabývá jedné z hodnot `c("first", "random", "average", "max", "min")` a používá se pro specifikaci řazení shodných hodnot. `"first"` řadí vzestupně podle pozice v řadě, `"random"` řadí náhodně, `"average"` podle průměrného pořadí (implicitní nastavení) a `"min"`/`"max"` podle hodnoty minimálního/maximálního pořadí

`rev(x)` převrácení pořadí hodnot vektoru

```
> a <- c(1, 2, 3, 5, 8, NA, 2, 4, 1, 2, 2)
> sort(a)   hodnoty NA automaticky vynechává
[1]  1   1   2   2   2   2   3   4   5   8
> sort(a, na.last=F)
[1]  NA   1   1   2   2   2   2   3   4   5   8
```

```
> order(a)
[1]  1   9   2   7   10   11   3   8   4   5   6
> rank(a, ties.method="average")
[1]  1.5   4.5   7.0   9.0   10.0   11.0   4.5   8.0   1.5   4.5   4.5
> rev(a)
[1]  2   2   1   4   2   NA   8   5   3   2   1
```

5.5 Zaokrouhlování

Pro zaokrouhlování se používají následující funkce:

`ceiling()`	zaokrouhlení k nejbližšímu většímu celému číslu
`floor()`	zaokrouhlení k nejbližšímu menšímu celému číslu
`trunc()`	zaokrouhlení směrem k 0, celá část daného čísla
`round()`	zaokrouhlení k nejbližšímu celému číslu, parametrem `digits=pocet` volíme počet desetinných míst, na jaký má být zaokrouhlení provedeno (implicitně `digits=0`)
`signif()`	zaokrouhlení na určitý počet platných cifer (parametr `digits`, zbytek doplní nulami)

```
> ceiling(c(3.468575, -3.468575))
[1]  4   -3
> floor(c(3.468575, -3.468575))
[1]  3   -4
> trunc(c(3.468575, -3.468575))
[1]  3   -3
> round(c(3.468575, -3.468575), digits=3)
[1]  3.469   -3.469
> signif(c(3.468575,-3.468575), digits=3)
[1]  3.47   -3.47
```

Poznámka. Zaokrouhlování čísla 5: pokud následují za číslicí 5 jen nuly, číslo je zaokrouhleno směrem dolů, pokud následuje jakákoliv jiná číslice, číslo je zaokrouhleno nahoru.

```
> round(7.125, digits=2)
[1]  7.12
> round(7.12501, digits=2)
[1]  7.13
```

5.6 Konstanty

Jazyk R má zabudovány konstanty, z nichž nejpoužívanější jsou:
`pi` ... π, Ludolfovo číslo
`exp(1)` ... e, základ přirozeného logaritmu, Eulerovo číslo
`i` ... i, komplexní jednotka
`.Last.value` ... proměnná obsahující poslední hodnotu, jež nebyla přiřazena do žádné proměnné explicitně
`letters` ... malá písmena abecedy
`LETTERS` ... velká písmena abecedy
`month.name` ... anglické názvy měsíců
`month.abb` ... zkratky anglických názvů měsíců

```
> 3
[1]  3
> .Last.value
[1]  3
> LETTERS[9:14]
[1]  "I"  "J"  "K"  "L"  "M"  "N"
> month.name[c(7, 10)]
[1]  "July"  "October"
```

Příklady k procvičení

1. Definujte vektory `v1` s hodnotami 1, 3, 4, `v2` jako posloupnost délky 5 s počáteční hodnotou -6 a krokem 2 a `v3` s prvními třemi složkami vektoru `v2`.
   a) Proveďte součet, rozdíl, součin a podíl vektorů `v1` a `v3`.
   b) Vysvětlete výsledky, pokud operace z předchozího zadání použijete na vektory `v1` a `v2`.
   c) Uveďte celou část a zbytek po dělení vektoru `v1` vektorem `v3`.
   d) Uveďte pozice vektoru `v1`, jejichž hodnoty jsou různé od hodnot vektoru `v3` na odpovídajících si pozicích.
   e) Uveďte počet prvků vektoru `v2`, jehož hodnoty jsou menší než -3 nebo rovny 1.
   f) Spočítejte součin nenulových hodnot vektoru `v2`.
   g) Zjistěte, zda je alespoň jeden prvek vektoru `v1` nulový.

2. Ve třech laboratořích byly analyzovány různé vlastnosti kávy: obsah vody (`x1`), hmotnost zrn (`x2`), pH (`x3`), tuky (`x4`), kofein (`x5`), obsah minerálů (`x6`) a extrakt (`x7`). V první laboratoři `lab1` byly pozorovány vlastnosti `x1`, `x2`, `x3`, `x6`, ve druhé laboratoři `lab2` vlastnosti `x1`, `x3`, `x4`, `x5` a ve třetí laboratoři `lab3` vlastnosti `x1`, `x2` a `x5`. Zjistěte, které vlastnosti

a) byly analyzovány ve všech třech laboratořích,
b) byly analyzovány v první i druhé laboratoři, ale ne ve třetí,
c) nebyly analyzovány ani jednou z laboratoří,
d) byly analyzovány alespoň jednou z laboratoří.

3. Je dána matice $\texttt{A} = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 1 \\ 8 & -27 & 1 \end{pmatrix}$. Spočítejte její spektrální poloměr, tj. největší vlastní číslo v absolutní hodnotě.

4. Kolika způsoby můžeme z krabičky 20 kuliček vybrat právě 5? Kolika způsoby bychom mohli vybrat 5 kuliček, záleželo by-li na jejich pořadí?

5. Spočítejte směrnici tečny ke grafu funkce, víte-li, že daná tečna svírá s kladnou poloosou x úhel $60\,^\circ$.

6. Náhodně vygenerujte vektor 10 celých čísel z intervalu $[-2, 3]$.

7. Náhodně vygenerujte vektor `v4` šesti hodnot z intervalu $[1, 4]$ a zaokrouhlete jej na:
a) 3 desetinná místa,
b) 3 platné cifry,
c) směrem k nule.

8. Vypište posledních 5 velkých písmen abecedy.

9. Vypište anglické názvy měsíců, jejichž pořadí odpovídá násobkům čísla 3.

Řešení.

1. `v1 <- c(1, 3, 4)`, `v2 <- seq(length=4, from=-2, by=2)`, `v3 <- v2[1:3]` nebo `v3 <- head(v2, 3)`
a) `v1 + v3`, `v1 - v3`, `v1 * v3`, `v1 / v3`
b) `v1 + v2`, `v1 - v2`, `v1 * v2`, `v1 / v2`... na kratší vektor použito pravidlo *recycling rule*
c) `v1 %/% v3`, `v1 %% v3`
d) `which(v1 != v3)`
e) `sum(v2 < -3 | v2 == 1)`
f) `prod(v2[v2 != 0])`
g) `any(v1 == 0)`

2. `lab1 <- c("x1", "x2", "x3", "x6")`,
`lab2 <- c("x1", "x3", "x4", "x5")`,
`lab3 <- c("x1", "x2", "x5")`

a) `intersect(intersect(lab1, lab2), lab3)`
b) `setdiff(intersect(lab1, lab2), lab3)`
c) `setdiff(c("x1", "x2", "x3", "x4", "x5", "x6"), union(union(lab1, lab2), lab3))`
d) `union(union(lab1, lab2), lab3))`

3. `A <- matrix(c(2, 1, 8, 1, -1, -27, 3, 1, 1), 3)`
   `max(abs(eigen(A)$values))`

4. `choose(20, 5)`, `factorial(20)/factorial(15)`

5. `tan(pi/3)`

6. `round(runif(n=10, min=-2, max=3))` nebo
   `sample(x=-2:3, size=10, rep=T)`

7. `v4 <- runif(n=6, min=1, max=4)`
   a) `round(v4, 3)`
   b) `signif(v4, 3)`
   c) `trunc(v4)`

8. `tail(LETTERS, 5)`

9. `month.name[seq(from=3, by=3, to=12)]`

Kapitola 6

Další příkazy v R

Základní informace

Jazyk R obsahuje velké množství funkcí, některé z nich jsou součástí standardní distribuce, mnohé jsou součástí podpůrných balíčků. Pro využití i těchto funkcí je velmi důležité umět pracovat s balíčky – umět je nainstalovat a načíst.
Pro statistickou analýzu dat je rovněž důležité umět pracovat nejen s vestavěnými datovými soubory, ale rovněž s vlastními (externími) datovými soubory. Pro práci s takovými soubory je potřeba je umět načíst, součástí této kapitoly jsou funkce pro ukládání dat a načítání externích datových souborů.
Poslední část kapitoly tvoří seznam funkcí, který uživatelům usnadňuje jednoduše a přehledně z datových struktur čerpat základní informace.

Výstupy z výuky

Studenti

- ovládají práci s knihovnami, umí je nainstalovat a načíst,
- dokáží použít různé funkce pro načítání dat,
- umí ukládat objekty do souboru,
- znají funkce pro výpis vlastností objektů.

6.1 Práce s knihovnami

Ne všechny funkce jsou přístupné ze základních knihoven, jsou umístěny v dodatečných balíčcích. Výpis aktuálně nainstalovaných knihoven můžeme získat příkazem `library()`.

Příkaz `search()` vyhledá přiinstalované knihovny.

```
> search()
 [1]  ".GlobalEnv"         "package:stats"  "package:graphics"
 [4]  "package:grDevices"  "package:utils"  "package:datasets"
 [7]  "package:methods"    "Autoloads"      "package:base"
```

Mnohdy je zapotřebí přiinstalovat další balíčky: záložka *Packages* → *Install Package(s) ...* nebo příkazem `install.packages("nazev_balicku")`. Před zahájením práce s přiinstalovanými balíčky je třeba je načíst příkazem `library(nazev_balicku)`, až teprve v tomto okamžiku je balíček připraven k používání. O tom se znovu můžeme přesvědčit:

```
> library(Matrix)
Loading required package : lattice
> search()
  [1]  ".GlobalEnv"     "package:Matrix"    "package:lattice"
  [4]  "package:stats"  "package:graphics"  "package:grDevices"
  [7]  "package:utils"  "package:datasets"  "package:methods"
 [10]  "Autoloads"      "package:base"
```

6.2 Práce s daty

Funkce pro ukládání dat
V prostředí R existuje několik funkcí pro ukládání dat. Uživatel by si měl nejdříve rozmyslet, k jakému účelu bude dále data používat - od toho se totiž bude odvíjet výběr správné funkce pro uložení dat.

Univerzální funkcí pro ukládání dat je funkce `save()`. Funkce se používá pro uložení libovolného objektu (nebo více objektů) do souboru nejčastěji s koncovkou `.RData` a `.dat` (interní formáty pro uchování dat v R), resp. do binárního souboru s příponou `.txt`. Takto uložené soubory mohou být následně načteny pomocí funkce `load()`.

`save(..., file)` funkce pro ukládání dat
`...` názvy ukládaných objektů
`file` textový řetězec specifikující název souboru pro uložení dat

```
> v <- 1:5
> mat <- matrix(2:7, 3)
> save(v, mat, file="seznam1.Rdata")
```
uložení objektů `v` a `mat`

```
> l <- list(v=1:5, mat=matrix(2:7, 3))
```
pro ukládání více objektů současně je mnohdy užitečné seskupit tyto objekty do seznamu a uložit je jako jeden objekt

```
> save(l, file="seznam2.RData")
> save(v, file="v.txt")
```
uložení objektu `v` do binárního souboru `v.txt`

Další využívanou funkcí pro ukládání dat je funkce `write.table()`, která slouží především k ukládání matic a datových tabulek. Nejtypičtějšími formáty pro ukládání jsou `.txt` a `.csv`, soubory obou formátů jsou jednoduše zobrazitelné v libovolném textovém editoru. Takto uložená data mohou být v prostředí R načtena pomocí funkce `read.table()`.

`write.table(x, file, sep, eol, dec, row.names, col.names)` funkce pro uložení matice nebo datové tabulky

`x`	název ukládaného objektu
`file`	textový řetězec specifikující název souboru pro uložení dat
`sep`	oddělovač sloupců, implicitní nastavení `sep=""`
`eol`	znak pro zalomení řádků, implicitní nastavení `eol="\n"`
`dec`	znak pro desetinnou čárku, implicitní nastavení `dec="."`
`row.names`	logická hodnota indikující, zda mají být uloženy i názvy řádků, resp. vektor názvů případů
`col.names`	vektor názvů proměnných

```
> krajeCR <- data.frame(nazev=c("Hlavni mesto Praha", "Stredocesky",
+ "Jihocesky", "Plzensky", "Karlovarsky", "Ustecky", "Liberecky",
+ "Kralovehradecky", "Pardubicky", "Olomoucky", "Moravskoslezsky",
+ "Jihomoravsky", "Zlinsky", "Vysocina"),
+ mesto=c("Praha", "Praha", "Ceske Budejovice", "Plzen", "Karlovy "
+ "Vary", "Usti nad Labem", "Liberec", "Hradec Kralove", "Pardubice"
+ "Olomouc", "Ostrava", "Brno", "Zlin", "Jihlava"),
+ SPZ = c("A", "S", "C", "P", "K", "U", "L", "H", "E", "M", "T", "B",
+ "Z", "J"))
```

```
> krajeCR
                  nazev              mesto  SPZ
 1   Hlavni mesto Praha              Praha    A
 2          Stredocesky              Praha    S
 3            Jihocesky  Ceske Budejovice    C
 4             Plzensky              Plzen    P
 5          Karlovarsky       Karlovy Vary    K
 6              Ustecky     Usti nad Labem    U
 7            Liberecky            Liberec    L
 8      Kralovehradecky     Hradec Kralove    H
 9           Pardubicky          Pardubice    E
 10           Olomoucky            Olomouc    M
 11     Moravskoslezsky            Ostrava    T
 12        Jihomoravsky               Brno    B
 13             Zlinsky               Zlin    Z
 14            Vysocina             Jihlava    J
> write.table(x=krajeCR, file="krajeCR.csv")
> write.table(x=krajeCR, file="krajeCR.txt")
```

uložení objektu `krajeCR` do souboru `krajeCR.csv`

uložení objektu `krajeCR` do souboru `krajeCR.txt`

Posloupnost jednotlivých příkazů i s jejich výstupy můžeme uložit v menu *File* → *Save to File...*. Historii příkazů lze uložit v menu *File* → *Save History* nebo příkazem `savehistory(file=".RHistory")`, příkazem `loadhistory(file=".RHistory")` ji můžeme následně načíst. V případě, že otevíráme dříve uložený soubor, historie příkazů je načtena automaticky s ním.

Funkce pro načítání dat

V jazyce R existuje několik funkcí pro načítání dat. Rozlišujeme funkce pro načítání dat ze schránky, externích datových souborů či objektů vytvořených (a uložených) přímo v prostředí R. Zaměříme se na nejpoužívanější funkce a uvedeme jejich použití.

K načítání objektů vytvořených v prostředí R (soubory s příponou `.Rdata` a `.dat`), resp. souborů s příponou `.txt` slouží funkce `load()`.

`load(file, verbose)` pro načítání dat

`file`	textový řetězec specifikující název souboru pro načtení dat
`verbose`	nastavení `TRUE` na obrazovku vytiskne názvy načítaných objektů, implicitní nastavení `verbose=FALSE`

```
> data1 <- load(file="v.txt", verbose=TRUE)
Loading objects:
  v
```

načtení souboru `v.txt`

```
> data1
[1]  "v"
> v
[1]  1   2   3   4   5
> data2 <- load(file="seznam1.RData", verbose=TRUE)
```

načtení datového souboru `seznam1.RData`, který obsahuje proměnné `v` a `mat`

```
Loading objects:
 v
 mat
> data2
[1]  "v"  "mat"
> v
[1]  1   2   3   4   5
> mat
      [,1]  [,2]
 [1,]    2     5
 [2,]    3     6
 [3,]    4     7
> load(file="dunaj.dat")
```

načtení dat ze souboru `dunaj.dat` (data znázorňující kolísání [m^3/s] Dunaje během roku, [8])

```
> dunaj
[1]  1987   1728   1862   2083   2143   2187   2588   2224   2001
[10]  1767   1460   1444
```

Pro načítání tabulek (především s příponou `.txt`) slouží funkce `read.table()`. K načítání datových tabulek ze souborů s příponou `.csv` lze použít i výše zmíněná funkce `read.table()`, pro pohodlnější načítání jsou však v prostředí R implementovány funkce `read.csv()` a `read.csv2()`. Obě tyto funkce se jen nepatrně liší ve svých vstupních argumentech, pro všechny tři funkce si ukážeme syntaxi a rozdíly v jejich použití. Podstatné je rovněž zmínit, že data načtená těmito funkcemi jsou ukládána jako objekty v podobě datové tabulky.

`read.table(file, header, sep, dec, row.names, col.names, nrows, skip, check.names, ...)` funkce pro načítání datových tabulek

`file`	textový řetězec uvádějící cestu k souboru pro načtení
`header`	logická hodnota pro názvy proměnných v prvním řádku, implicitní nastavení `header=FALSE`
`sep`	oddělovač sloupců
`dec`	znak pro desetinnou čárku
`row.names`	vektor názvů řádků
`col.names`	vektor názvů sloupců
`nrows`	maximální počet řádků, který má být načten
`skip`	počet řádků, který má být přeskočen před samotným načtením
`check.names`	implicitní nastavení `chcek.names=TRUE` zajišťuje syntaktickou správnost proměnných

```
> tab1 <- read.table(file="krajeCR.txt", header=TRUE)
> head(tab1, n=5)
```

pro zkrácení výpisu zobrazíme pouze prvních pět řádků tabulky

```
                nazev             mesto  SPZ
1  Hlavni mesto Praha             Praha    A
2         Stredocesky             Praha    S
3           Jihocesky  Ceske Budejovice    C
4            Plzensky             Plzen    P
5         Karlovarsky      Karlovy Vary    K
```

```
> tab2 <- read.table(file="krajeCR.csv", header=TRUE, sep="")
> head(tab2, n=5)
                nazev             mesto  SPZ
1  Hlavni mesto Praha             Praha    A
2         Stredocesky             Praha    S
3           Jihocesky  Ceske Budejovice    C
4            Plzensky             Plzen    P
5         Karlovarsky      Karlovy Vary    K
```

`read.csv(file, header, sep, dec, ...)` funkce pro načítání datových tabulek z `.csv` souboru

`file`	textový řetězec uvádějící cestu k souboru pro načtení
`header`	logická hodnota pro názvy proměnných v prvním řádku, implicitní nastavení `header=TRUE`
`sep`	oddělovač sloupců, implicitní nastavení `sep=","`
`dec`	znak pro desetinnou čárku, implicitní nastavení `dec="."`

`read.csv2(file, header, sep, dec, ...)` funkce pro načítání datových tabulek z `.csv` souboru

`file`	textový řetězec uvádějící cestu k souboru pro načtení
`header`	logická hodnota pro názvy proměnných v prvním řádku, implicitní nastavení `header=TRUE`
`sep`	oddělovač sloupců, implicitní nastavení `sep=";"`
`dec`	znak pro desetinnou čárku, implicitní nastavení `dec=","`

```
> tab3 <- read.csv(file="krajeCR.csv", sep="")
> head(tab3, n=5)
                 nazev             mesto  SPZ
1  Hlavni mesto Praha             Praha    A
2         Stredocesky             Praha    S
3           Jihocesky  Ceske Budejovice    C
4            Plzensky             Plzen    P
5         Karlovarsky      Karlovy Vary    K
```

`> lab1 <- read.csv2(file="laborator.csv")` soubor `laborator.csv` obsahuje údaje o procentuálním zastoupení měřené látky ve dvou odlišných laboratořích

```
> lab1
   laborator  obsah.latky
1          1         69.2
2          1         57.1
3          1         62.8
4          2         63.5
5          2         59.7
6          2         60.2
> lab2 <- read.csv(file="laborator.csv", sep=";", dec=",")
> lab2
   laborator  obsah.latky
1          1         69.2
2          1         57.1
3          1         62.8
4          2         63.5
5          2         59.7
6          2         60.2
```

Pro načítání dat ze schránky lze použít funkce `scan()`, pro načítání datových tabulek ze schránky funkce `read.table()` s argumentam `file="clipboard"`.

`scan(file, what, sep, dec, nmax)` je funkce pro načtení vektoru nebo seznamu z konzole nebo souboru. Popis argumentů:

`file`	textový řetězec uvádějící cestu k souboru, při načítání ze schránky `file="clipboard"`
`what`	typ načítané hodnoty (numerické, komplexní, ...)
`sep`	znak, kterým jsou odděleny jednotlivé načítané položky, např. `sep=","`
`dec`	znak pro desetinnou čárku
`nmax`	maximální počet hodnot, který má být načten

Postup pro zobrazení hodnot uložených ve schránce:
1. zkopírovat příslušná data, např. 2; 4; 6; 1; 3; 5
2. zavolat funkci `scan()`

```
> x <- scan(file="clipboard", sep=";")
```

Po jejím zavolání se objeví oznámení:

```
Read 6 items
> x
[1]  2   4   6   1   3   5
> x <- scan(file="clipboard", sep=";", what=character())
Read 6 items
> x
[1]  "2"  "4"  "6"  "1"  "3"  "5"
```

1, 2, 3, 4, 5
3, 2, 1, 1, 2

```
> read.table(file="clipboard", sep=",")
   V1  V2  V3  V4  V5
 1  1   2   3   4   5
 2  3   2   1   1   2
```

1 2 3 4 5 r1
3 2 1 1 2 r2

```
> read.table(file="clipboard", row.names=6, col.names=c("sl1", "sl2",
+ "sl3", "sl4", "sl5", "sl6"))
```

přestože poslední sloupec obsahuje názvy řádků, argument `col.names` musí obsahovat jeho název i tohoto sloupce (`"sl6"`), i když nebude vytištěn

```
     sl1  sl2  sl3  sl4  sl5
 r1    1    2    3    4    5
 r2    3    2    1    1    2
```

sl1 sl2 sl3 sl4 sl5 sl6
1 2 3 4 5 r1
3 2 1 1 2 r2

```
> read.table(file="clipboard", row.names=6, header=T)
    sl1  sl2  sl3  sl4  sl5
r1    1    2    3    4    5
r2    3    2    1    1    2
```

V praxi se často setkáváme s datovými formáty ve formátu `.xls` či `.xlsx`, o nichž v textu zatím nebyla žádná zmínka. Pro načítání dat těchto formátů sice existují funkce v dalších balíčcích, práce s nimi je ovšem natolik obtížná, že je doporučováno nejprve data převést do `.csv` souboru a ten poté načíst.

Poznámka. Pro načítání `.txt` souborů je doporučován následující postup:

1. zobrazení souboru v libovolném prohlížeči textových dokumentů,
2. pokud je soubor čitelný, pro načtení dat je vhodné použití funkce `scan()`, příp. `read.table()` při načítání datových tabulek.

Alternativou rovněž může být zobrazení `.txt` souboru přímo v R pomocí funkce `readLines()`. Pokud je soubor v „čitelném“ formátu, výpis bude čitelný i v R a k načtení dat je možno použít funkci `read.table()`. Pokud je soubor v binárním formátu, výpis je nečitelný (nedává smysl) a k načtení dat je třeba použít funkci `load()`.

```
> readLines("v.txt")    v (binárním) souboru je uložen vektor v, k uložení byl použit
příkaz save(v, file="v.txt")
[1]  "RDX2""X"  ""  ""
 Warning messages:
1: In readLines("v.txt") : line 3 appears to contain an embedded nul
2: In readLines("v.txt") : line 4 appears to contain an embedded nul
3: In readLines("v.txt") : incomplete final line found on 'v.txt'
> load("v.txt")
> v
[1]  1   2   3   4   5
```

```
> readLines("krajeCR.txt")
```

v souboru je uložena datová tabulka `krajeCR`, k uložení byl použit příkaz `write.table(krajeCR, file="krajeCR.txt")`, výpis je čitelný (dává smysl)

```
 [1]  "\"nazev\"\"mesto\"\"SPZ \""
 [2]  "\"1\"\"Hlavni mesto Praha\"\"Praha\"\"A\""
 [3]  "\"2\"\"Stredocesky\"\"Praha\"\"S\""
 [4]  "\"3\"\"Jihocesky\"\"Ceske Budejovice\"\"C\""
 [5]  "\"4\"\"Plzensky\"\"Plzen\"\"P\""
 [6]  "\"5\"\"Karlovarsky\"\"Karlovy Vary\"\"K\""
 [7]  "\"6\"\"Ustecky\"\"Usti nad Labem\"\"U\""
 [8]  "\"7\"\"Liberecky\"\"Liberec\"\"L\""
 [9]  "\"8\"\"Kralovehradecky\"\"Hradec Kralove\"\"H\""
[10]  "\"9\"\"Pardubicky\"\"Pardubice\"\"E\""
[11]  "\"10\"\"Olomoucky\"\"Olomouc\"\"M\""
[12]  "\"11\"\"Moravskoslezsky\"\"Ostrava\"\"T\""
[13]  "\"12\"\"Jihomoravsky\"\"Brno\"\"B\""
[14]  "\"13\"\"Zlinsky\"\"Zlin\"\"Z\""
[15]  "\"14\"\"Vysocina\"\"Jihlava\"\"J\""
> krajeCR <- read.table("krajeCR.txt")
> krajeCR
> head(krajeCR, n=5)
                  nazev             mesto  SPZ
1   Hlavni mesto Praha             Praha    A
2          Stredocesky             Praha    S
3            Jihocesky  Ceske Budejovice    C
4             Plzensky             Plzen    P
5          Karlovarsky      Karlovy Vary    K
6              Ustecky    Usti nad Labem    U
7            Liberecky           Liberec    L
8      Kralovehradecky    Hradec Kralove    H
9           Pardubicky         Pardubice    E
10           Olomoucky           Olomouc    M
11     Moravskoslezsky           Ostrava    T
12        Jihomoravsky              Brno    B
13             Zlinsky              Zlin    Z
14           Vysocina           Jihlava    J
```

Pro usnadnění práce s daty jazyk R obsahuje okolo sta vestavěných datových souborů (v balíčku `datasets`). Funkce `data()` zobrazí seznam dostupných datových souborů. K zobrazení vybraných souborů slouží příkaz `data(nazev, package)`[1], argument `package` slouží ke specifikaci balíčku, ve kterém se data nachází.

Dalšími užitečnými funkcemi jsou `attach()` a `detach()`.
Funkce `attach()` má použití zejména u seznamů a datových tabulek, kde umožňuje vypisování jejich složek přímo, tzn. bez uvedení názvu objektu. Argumentem funkce je název objektu, u kterého chceme výše uvedené provést.

```
> tabulka <- data.frame(id=c(1:6), skupina=c(1, 2, 2, 1, 2, 1),
+ hodnota=runif(6, 2, 4))
> id
Error : object 'id' not found
> attach(tabulka)
> id
[1]  1   2   3   4   5   6
```

Opakem k funkci `attach` je funkce `detach`, která naopak znemožní, aby byly složky volány pouze svým názvem, nikoliv uvedením i názvu objektu.

```
> detach(tabulka)
> id
Error : object 'id' not found
> tabulka$id
[1]  1   2   3   4   5   6
```

6.3 Vlastnosti objektů

Se základními vlastnostmi objektů jsme se již seznámili v předešlých kapitolách. Zde si uvedeme některé další funkce, které nám o vnitřní struktuře objektů vypoví mnohem více.

`summary(object)` jedná se o tzv. generickou funkci (funkce, která si nejprve zjistí, jakého typu je její parametr, podle něj pak vypisuje celkový přehled: pro vektory vrací základní charakteristiky polohy, pro faktory vrací četnosti jednotlivých složek)

[1] Od verze 2.0.0 jsou všechny datové soubory přístupné přímo zavoláním jejich názvu. Mnohé balíčky ovšem stále používají dřívější způsob volání pomocí `data(nazev)`, jenž může být stále využíván i dnes.

```
> (a <- c(1, 2, 3, 5, 8, 2, 4, 1, 2, 2))
[1]  1   2   3   5   8   2   4   1   2   2
> summary(a)
```

v případě číselných vektorů funkce `summary()` vrací minimum, 1. kvartil, medián, průměr, 3. kvartil a maximum

```
   Min.  1st Qu.  Median  Mean  3rd Qu.  Max.
   1.00     2.00    2.00  3.00     3.75  8.00
> (b <- c(1+3i, 7-1i, NA))
[1]  1+3i   7-1i   NA
> summary(b)
   Length    Class     Mode
        3  complex  complex
> (c <- c(T, T, F, T, F))
[1]  TRUE   TRUE   FALSE   TRUE   FALSE
> summary(c)
     Mode  FALSE  TRUE  NA's
   logical      2     3     0
> (d <- c("k", "l", "l", "m"))
[1]  "k"  "l"  "l"  "m"
> summary(d)
   Length      Class       Mode
        4  character  character
> (e <- matrix(a, c(5, 2)))
        [,1]  [,2]
 [1,]     1     2
 [2,]     2     4
 [3,]     3     1
 [4,]     5     2
 [5,]     8     2
> summary(e)
```

vypíše přehled pro každý sloupec

```
       V1              V2
   Min.   :1.0   Min.   :1.0
   1st Qu.:2.0   1st Qu.:2.0
   Median :3.0   Median :2.0
   Mean   :3.8   Mean   :2.2
   3rd Qu.:5.0   3rd Qu.:2.0
   Max.   :8.0   Max.   :4.0
> (f <- factor(a))
[1]  1   2   3   5   8   2   4   1   2   2
 Levels: 1   2   3   4   5   8
> summary(f)
```

vypisuje četnosti jednotlivých úrovní faktoru (`Levels`)

```
1   2   3   4   5   8
2   4   1   1   1   1
```

```
> (tab <- data.frame(cervena=c(15, 14, 12, 12), seda=c(13, 17, 10, 9),
+ zelena=c(7, 9, 8, 6)))
   cervena  seda  zelena
1       15    13       7
2       14    17       9
3       12    10       8
4       12     9       6
> summary(tab)   opět vypisuje přehled pro každý sloupec zvlášť
   cervena          seda          zelena
Min.   :12.00  Min.   : 9.00  Min.   :6.00
1st Qu.:12.00  1st Qu.: 9.75  1st Qu.:6.75
Median :13.00  Median :11.50  Median :7.50
Mean   :13.25  Mean   :12.25  Mean   :7.50
3rd Qu.:14.25  3rd Qu.:14.00  3rd Qu.:8.25
Max.   :15.00  Max.   :17.77  Max.   :9.00
```

`str(object)` přehledně vypíše podrobnou vnitřní strukturu objektu, je alternativou k funkci `summary()` s tím rozdílem, že výpis provádí do řádku

```
> str(a)
num   [1:10]   1   2   3   5   8   2   4   1   2   2
> str(b)
cplx   [1:3]   1+3i   7-1i   NA
> str(c)
logi   [1:5]   TRUE   TRUE   FALSE   TRUE   FALSE
> str(d)
chr   [1:4]   "k"  "l"  "l"  "m"
> str(e)
num   [1:5, 1:2]   1   2   3   5   8   2   4   1   2   2
> str(f)
Factor w/   6 levels "1"  "2"  "3"  "4",..:  1   2   3   5   8   2
4   1   2   2
> str(tab)
'data.frame':   4 obs. of 3 variables
 $ cervena: num   15   14   12   12
 $ seda   : num   13   17   10   9
 $ zelena : num   7   9   8   6
```

`comment(object)` nastaví nebo vypíše komentář k danému objektu

```
> comment(tab) <- "Pocty aut jednotlivych barev behem 4 casovych
+ obdobi."   nastavení komentáře k objektu tab
```

```
> comment(tab)    zobrazení komentáře
[1]  "Pocty aut jednotlivych barev behem 4 casovych obdobi."
```

`attributes(object)` vypisuje všechny atributy objektu. K výpisu nebo nastavení konkrétní vlastnosti objektu slouží příkaz `attr(object, name)`, kde parametr `name` udává název zjišťované vlastnosti.

```
> (ar <- array(1:8, c(2, 2, 2)))
,  , 1
     [,1]  [,2]
[1,]    1     2
[2,]    2     4
,  , 2
     [,1]  [,2]
[1,]    5     7
[2,]    6     8
> attributes(ar)
$dim
[1]  2   2   2
> attributes(f)
$levels
[1]  "1"  "2"  "3"  "4"  "5"  "8"

$class
[1]  "factor"
> attributes(tab)
$names
[1]  "cervena"  "seda"  "zelena"

$row.names
[1]  1   2   3   4

$class
[1]  "data.frame"

$comment
[1]  "Pocty aut jednotlivych barev behem 4 casovych obdobi."
> attr(tab, "names")
[1]  "cervena"  "seda"  "zelena"
```

Funkce `as.something()` umožňuje změny datových typů všude, kde je to smysluplné.

```
> u <- 1:10
[1]  1  2  3  4  5  6  7  8  9  10
> v <- as.character(u)   převede numerický vektor na vektor textových hodnot
[1]  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
> w <- as.integer(v)   převede vektor textových hodnot na numerický vektor, vektory u a w jsou stejného typu
[1]  1  2  3  4  5  6  7  8  9  10
```

K převodu objektů z jednoho datového typu na druhý je k dispozici velké množství funkcí typu `as.something()`. Jejich seznam můžeme najít příkazem `methods(as)`.

Příklady k procvičení

1. Do pracovního adresáře načtěte soubor `strom.dat`. Součástí tohoto souboru je objekt `strom`.
   a) Vhodným příkazem zjistěte informace o vnitřní struktuře tohoto objektu.
   b) Zjistěte, zda je pro tento objekt vytvořen komentář. V případě že není, stručný komentář pro něj vytvořte.
   c) Převeďte tento objekt na textovou matici `strom1`.

2. Nainstalujte a načtěte knihovnu `Hmisc`. Na objektu `strom` porovnejte výstup funkce `describe()` z této knihovny s funkcemi `str()` a `attributes()`. Výstup funkce `describe()` uložte do proměnné `descr`.

3. Do proměnné `mereni1` načtěte soubor `mereni.txt` i s názvy řádků a sloupců.

4. Umožněte, aby na jednotlivé sloupce proměnné `mereni1` mohlo být odkazováno přímo.

5. Spojte proměnné `strom`, `strom1`, `descr` a `metoda2` do seznamu `seznam`, ten uložte do souboru `seznam.dat`. Výpisem souborů pracovního adresáře se přesvědčte, že jste tak opravdu učinili.

Řešení.

1. `load("strom.dat")`
   a) `summary(strom)`
   b) `comment(strom) <- "Metody mereni vysky stromu"`
   c) `strom1 <- as.matrix(strom)`

2. `install.packages("Hmisc")`, `library(Hmisc)`
   `descr <- describe(strom)`, `str(strom)`, `attributes(strom)`

3. `mereni1 <- read.table(file="mereni.txt")`

4. `attach(mereni1)`

5. `seznam <- list(strom=strom, strom1=strom1, descr=descr, metoda2=metoda2)`
   `save(seznam, file="seznam.dat")`, `dir()`

Kapitola 7

Grafika v R

Základní informace

Grafika jazyka R patří k jedné z nejvíce propracovaných částí, nabízí nepřeberné množství funkcí pro tvorbu kompletních grafů včetně popisků (high-level grafika) stejně jako funkcí k přidávání nových částí či změně již existujícího vzhledu grafu (low-level grafika).
Cílem kapitoly je uvést nejvíce používané argumenty grafických funkcí, seznámit se s funkcí `par`, jejíž argumenty dokáží zastoupit mnohé low-level funkce, a dalšími funkcemi k tvorbě grafických výstupů.

Výstupy z výuky

Studenti

- znají rozdíly mezi high-level a low-level grafikou,
- se seznámí s několika příkazy pro tvorbu high-level grafiky a umí používat jejich argumenty,
- ovládají funkce pro vykreslení základních typů grafů,
- dokáží vytvářet 3D grafy,
- umí použít funkce pro tvorbu low-level grafiky,
- umí využívat vlastností funkce `par`,
- znají příkazy pro ukládání grafu do souboru, umí pracovat s podgrafy.

Systém R umožňuje zobrazovat širokou škálu grafů. Příkazy k vykreslování grafů jsou rozděleny do tří základních skupin:

- **High-level** funkce vytváří kompletní nový graf s osami, popisky, názvem atd.
- **Low-level** funkce přidávají do již existujícího grafu další informace, např. další body, čáry, popisky.
- **Interaktivní grafika** umožňuje interaktivně pomocí myši přidávat data do již existujícího grafu.

V případě, že jsou tyto grafy nedostačující, můžeme použít dalších balíčků - např. `grid`, `lattice`, `iplots`, `misc3D`, `rgl`, `scatterplot` nebo balíček `maps` obsahující nejrůznější mapy.

7.1 High-level funkce

Všechny grafy jsou nejprve vytvořeny pomocí high-level funkcí, které vytváří "kompletní"graf. Kompletní v tom smyslu, že jsou automaticky vygenerovány osy, popisky a nadpis (pokud sami nenastavíme jinak). High-level funkce vždy vykreslují nový graf, v případě, že již nějaký graf existuje, přepíší jej. Je důležité si uvědomit, že data k vykreslení mohou být různé objekty - podle druhu objektu grafické funkce následně vykreslují graf.

Argumenty k high-level funkcím:

`axes`	implicitní hodnota `TRUE`, nastavení na hodnotu `FALSE` potlačuje vykreslování os
`log`	nastavením na hodnoty `log="x"`, `log="y"`, `log="xy"` budou mít vybrané osy logaritmické měřítko
`main`	textový řetězec pro název grafu, je umístěn nad grafem
`sub`	textový řetězec, je umístěn pod grafem a psán menším fontem
`type`	typ výstupního grafu
	`type="p"` vykresluje samostatné body (implicitní nastavení)
	`type="l"` linie
	`type="b"` přerušované linie s body
	`type="c"` přerušované linie bez bodů
	`type="o"` body navzájem spojené liniemi
	`type="h"` vertikální linie
	`type="s"` schodovitý graf s první linií horizontální
	`type="S"` schodovitý graf s první linií vertikální
	`type="n"` žádné vykreslování dat, vykresleny jsou pouze osy, rozsah souřadného systému závisí na datech

`xlab, ylab` textové řetězce pro názvy os x a y, tyto argumenty mění implicitně zadané názvy os při volání high-level funkcí

`xlim, ylim` 2-prvkový vektor specifikující minimální a maximální hodnotu k vykreslení dané osy

Dále jsou uvedeny nejpoužívanější typy grafů se svými argumenty. Nejedná se o kompletní výčet argumentů, uvedeny jsou pouze ty nejdůležitější. Použitá data jsou uvedena na přiloženém CD.

• `plot(x)` vzhled grafu závisí na povaze vstupních dat. Pro numerický vektor vrací jednoduchý graf s indexy na ose x a hodnotami na ose y, pro matice vrací graf s hodnotami prvního sloupce na ose x a odpovídajícími hodnotami druhého sloupce na ose y (ostatní sloupce jsou ignorovány), pro faktory vykresluje sloupcový graf četností jednotlivých kategorií, pro datové tabulky vykresluje bodový graf závislosti všech proměnných, pro funkce vykresluje hladkou čáru

• `plot(x, y)` jestliže `x` a `y` jsou vektory téže délky, funkce vykreslí bodový graf hodnot `y` na pozicích `x`. Stejného výsledku můžeme docílit nahrazením argumentů `x` a `y` buď seznamem, obsahujícím dvě složky `x` a `y`, nebo maticí o dvou sloupcích (viz příkaz `plot(x)`).

Implicitní vzhled grafu `plot(x)` a `plot(x, y)` můžeme upravit pomocí široké škály volitelných argumentů:

`main, sub, type`

`axes, xlim, ylim, xlab, ylab`

`ann, col, bg, pch, cex, lty, lwd` parametry zadávající výpis názvu a popisků os, barvu, barvu pozadí grafu, znaky pro vykreslení bodů a jejich velikost, typ a tloušťku čar. Více informací v odstavci 7.3.

`frame.plot` logická hodnota implicitně nastavená na `TRUE` graf orámuje

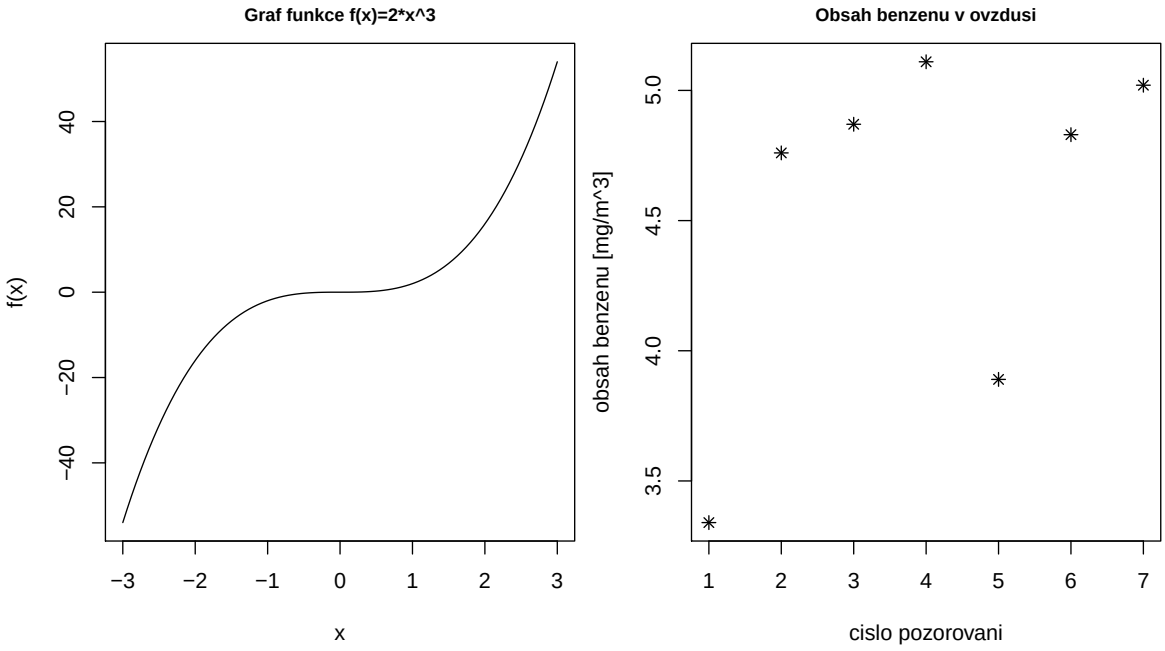
`asp` poměr osy y/x

```
> plot(function(x) 2*x^3, xlim=c(-3, 3), ylab="f(x)", main="Graf funkce
+ f(x)=2*x^3")
```

```
> benzen <- c(3.34, 4.76, 4.87, 5.11, 3.89, 4.83, 5.02)
```

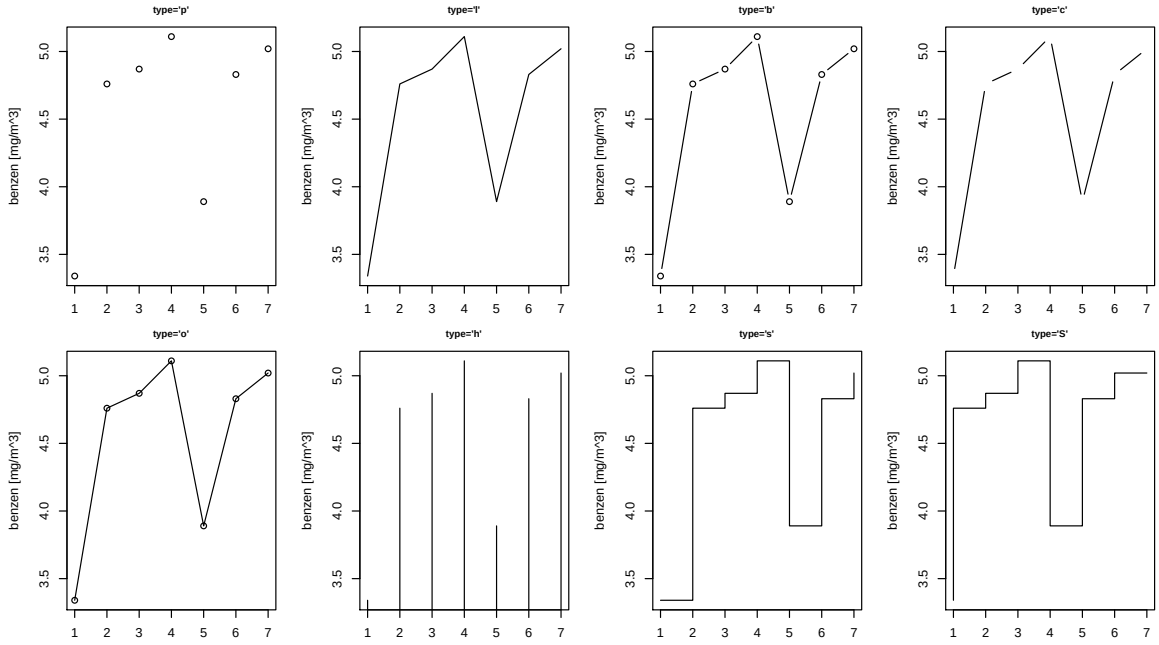
sedm měření pro kontrolu obsahu benzenu v ovzduší [mg/m^3], [8]

```
> plot(benzen, pch=8, xlab="cislo pozorovani", ylab="obsah benzenu
+ [mg/m^3]", main="Obsah benzenu v ovzdusi")
```



Obr. 7.1. Funkce plot()

```
> plot(benzen, type="p", main="type='p'", ylab="benzen [mg/m^3]")
> plot(benzen, type="l", main="type='l'", ylab="benzen [mg/m^3]")
> plot(benzen, type="b", main="type='b'", ylab="benzen [mg/m^3]")
> plot(benzen, type="c", main="type='c'", ylab="benzen [mg/m^3]")
> plot(benzen, type="o", main="type='o'", ylab="benzen [mg/m^3]")
> plot(benzen, type="h", main="type='h'", ylab="benzen [mg/m^3]")
> plot(benzen, type="s", main="type='s'", ylab="benzen [mg/m^3]")
> plot(benzen, type="S", main="type='S'", ylab="benzen [mg/m^3]")
```



Obr. 7.2. Možné varianty argumentu `type`

• `barplot()` sloupcový graf

`axes, main, sub, xlab, ylab, xlim, ylim`

`beside` lze použít pouze v případě vstupního argumentu typu matice. Implicitní hodnota `beside=FALSE` vykreslí více obdélníků nad sebou, nastavení na hodnotu `TRUE` vedle sebe.

`width` vektor hodnot udávajících šířku vykreslovaných obdélníků na ose x. Jestliže vektor `wide` nedosahuje délky rovné počtu složek argumentu, je použito pravidlo *recycling rule*.

`space` velikost místa pro vynechání mezi jednotlivými sloupci (podíl průměrné šířky sloupce), může se jednat o numerickou hodnotu nebo vektor hodnot. Pro vstupní matici a argument `beside=TRUE` může být argument `space` specifikován vektorem dvou hodnot, první vyjadřuje odestup sloupců ve stejné skupině (ve stejném sloupci), druhý vyjadřuje odestup mezi skupinami. Implicitní nastavení je `space=c(0,1)` pro vstupní matici a argument `beside=TRUE`, pro ostatní možnosti `space=0.2`.

`names.arg` vektor názvů ke každému sloupci nebo skupině sloupců

`legend.text` vektor textových řetězců uvádějící názvy v legendě, má smysl jen pro vstupní matice

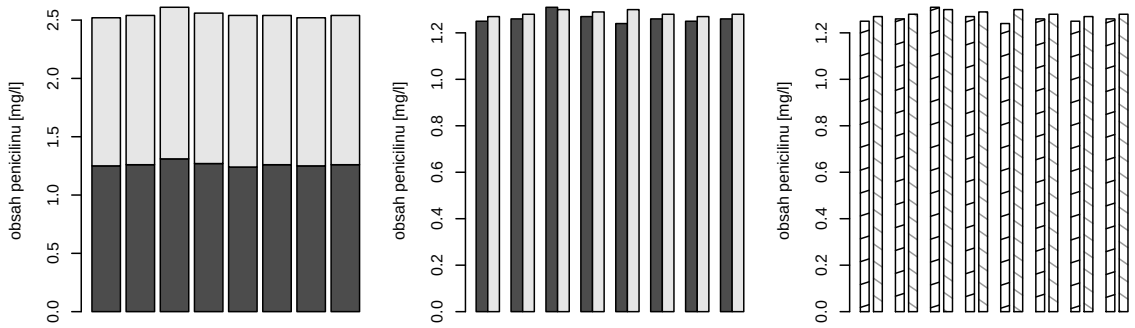
`horiz` implicitní hodnota `horiz=FALSE` vykresluje obdélníky vertikálně, argument `horiz=TRUE` horizonatálně

`density` hodnota nebo numerický vektor nastavující hustotu šrafování sloupců

`angle` úhel pro sklon šrafování

```
> load(file="penicilin.dat")
```
načtení potřebného souboru dat, který musí být uložen v aktuálním pracovním adresáři. Data pro porovnání obsahu penicilinu [mg/l] v krvi dvou pacientů, [8]

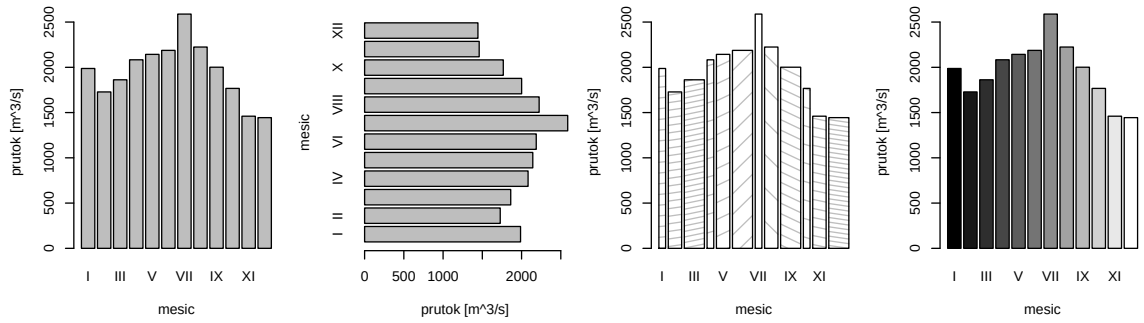
```
> barplot(penicilin, ylab="obsah penicilinu [mg/l]")
> barplot(penicilin, beside=T, ylab="obsah penicilinu [mg/l]")
> barplot(penicilin, beside=T, ylab="obsah penicilinu [mg/l]",
+ space=c(0.5,1.5), density=c(7,18), angle=c(60,105), col=gray(c(0,
+ 0.6)))
```
funkce `gray` slouží k vykreslení různého stupně šedi (více v odstavci 7.3)



Obr. 7.3. Porovnání obsahu penicilinu v krvi dvou pacientů [mg/l]

```
> load(file="dunaj.dat")
```
načtení datového souboru `dunaj.dat` (kolísání průtoku Dunaje [m^3/s] během roku, [8])

```
> barplot(dunaj, names.arg=c("I", "II", "III", "IV", "V", "VI", "VII",
+ "VIII", "IX", "X", "XI", "XII"), ylab="prutok [m^3/s]",
+ xlab="mesic")
> barplot(dunaj, horiz=T, names.arg=c("I", "II", "III", "IV", "V",
+ "VI", "VII", "VIII", "IX", "X", "XI", "XII"), ylab="prutok
+ [m^3/s]", xlab="mesic")
> barplot(dunaj, width=c(1, 2, 3), density=rep(c(5, 8, 13), times=4),
+ angle=seq(from=30, by=10, length=12), names.arg=c("I", "II", "III",
+ "IV", "V", "VI", "VII", "VIII", "IX", "X", "XI", "XII"), ylab="prutok
+ [m^3/s]", xlab="mesic")
> barplot(dunaj, col=gray(seq(from=0, to=1, length=12)), names.arg=
+ c("I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX", "X", "XI",
+ "XII"), ylab="prutok [m^3/s]", xlab="mesic")
```



Obr. 7.4. Kolísání průtoku Dunaje během roku [mˆ3/s]

• `hist()` histogram pro numerický vektor

`breaks` argument šířky intervalů, implicitní nastavení je `breaks="Sturges"`, která šířku intervalů počítá Sturgesovým pravidlem: $\log_2 n + 1$, [18], dalšími možnostmi jsou `breaks="Scott"` (Scottovo pravidlo: $3.5 \cdot \hat{\sigma} \cdot n^{-1/3}$, [18]) nebo `breaks="FD"` (Freedman a Diaconis: $2 \cdot IQR \cdot n^{-1/3}$, [18]), n udává počet hodnot vektoru, $\hat{\sigma}$ je odhadem směrodatné odchylky a IQR značí interkvantilové rozpětí

`freq` implicitní nastavení `freq=TRUE` vykresluje absolutní četnosti, nastavení na hodnotu `freq=FALSE` vykresluje relativní četnosti

`right` pro implicitní hodnotu `right=TRUE` jsou intervaly zprava uzavřené, zleva otevřené, pro hodnotu `right=FALSE` je tomu naopak

`labels` pro hodnotu `labels=TRUE` vypisuje nad sloupce absolutní/relativní četnosti

`density, angle` stejně jako u `barplot()`

`axes, main, sub, xlim, ylim, xlab, ylab`

Funkce `hist()` má k dispozici i výstupní hodnoty:

`breaks` hodnoty hranic intervalů

`counts` absolutní četnosti pro každý interval

`density` hustota pravděpodobnosti pro jednotlivé intervaly

`intensities` shodné s `density`

`mids` středy intervalů

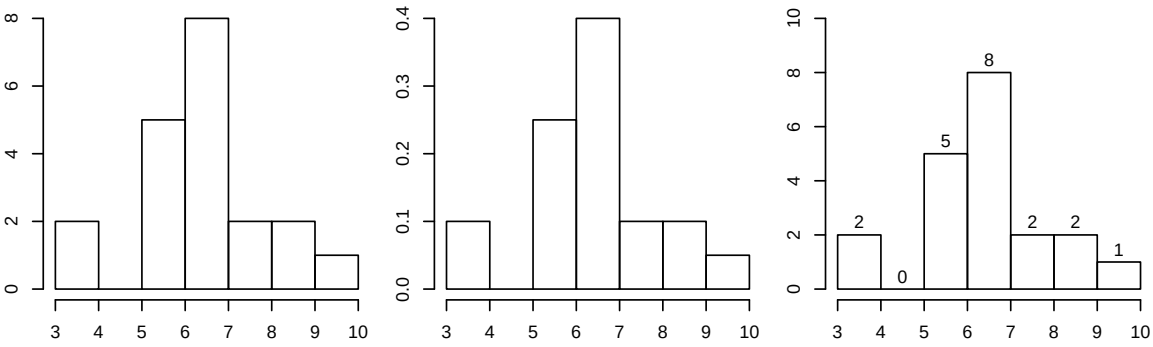
`xname` název objektu v argumentu

`equidist` logická hodnota stejné délky všech intervalů

```
> load(file="obili.dat")    načtení souboru dat (sklizňová ztráta obilí [g/m^3], [8])
> hist(obili, breaks="Sturges", main="breaks='Sturges'")
> hist(obili, breaks="Scott", main="breaks='Scott'")
> hist(obili, breaks="FD", main="breaks='FD'")
```

Obr. 7.5. Sklizňová ztráta obilí [g/mˆ3], argumenty šířky intervalů

```
> hist(obili, main="")
> hist(obili, freq=F, main="")
> hist(obili, labels=T, main="")
```



Obr. 7.6. Sklizňová ztráta obilí [g/mˆ3], zobrazení absolutních a relativních četností

```
> (hist(obili))   výstupní hodnoty
$breaks
[1]  3   4   5   6   7   8   9   10

$counts
[1]  2   0   5   8   2   2   1

$intensities
[1]  0.09999998   0.00000000   0.25000000   0.40000000   0.10000000
+ 0.10000000   0.05000000

$density
[1]  0.09999998   0.00000000   0.25000000   0.40000000   0.10000000
+ 0.10000000   0.05000000

$mids
[1]  3.5   4.5   5.5   6.5   7.5   8.5   9.5

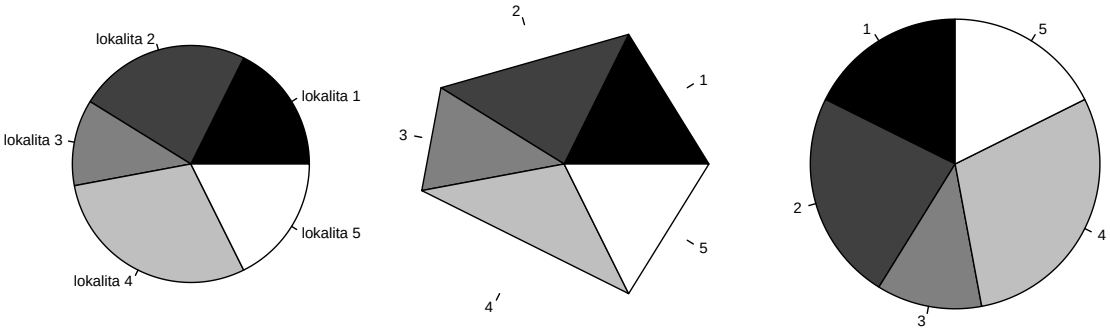
$xname
[1]  "obili"

$equidist
[1]  TRUE

$attr(,"class")
[1]  "histogram"
```

- `pie()` koláčový graf pro vektor nezáporných celých čísel
  `labels` vektor textových hodnot vyjadřující názvy jednotlivých dílů grafu
  `edges` numerická hodnota udávající počet vrcholů polygonu
  `clockwise` logická hodnota vykreslení po (`TRUE`) nebo proti (`FALSE`, implicitní nastavení) směru hodinových ručiček
  `init.angle` počáteční úhel pro natočení grafu
  `density, angle`

```
> kralici <- c(3, 4, 2, 5, 3)
> pie(kralici, angle=30, labels=c("lokalita
+ 1", "lokalita 2", "lokalita 3", "lokalita 4", "lokalita 5"),
+ col=gray(seq(from=0, to=1, length=5)))
> pie(kralici, edges=5, col=gray(seq(from=0, to=1, length=5)))
> pie(kralici, clockwise=F, init.angle=90, col=gray(seq(from=0, to=1,
+ length=5)))
```



Obr. 7.7. Počet chycených králíků v jednotlivých lokalitách lesa, [8]

• `boxplot()` krabicový graf

`range` vymezuje rozpětí "fousků"grafu, pro kladné `range` se "fousky"rozpínají do vzdálenosti `range` $\cdot IQR$ (interkvartilová odchylka), pro `range=0` se rozpínají od minimální po maximální hodnotu, implicitní nastavení `range=1.5`

`notch` implicitní nastavení `notch=FALSE`, pro `notch=TRUE` zobrazuje výřezy po stranách "krabice"(výřezy odpovídají hodnotám $\frac{\pm 1.58 \cdot IQR}{\sqrt{n}}$)

`outline` implicitní nastavení `outline=TRUE` zobrazuje odlehlé hodnoty, hodnota `FALSE` odlehlé hodnoty nezobrazuje

`names` vektor textových řetězců pro názvy jednotlivých krabicových grafů

`horizontal` pro hodnotu `horizontal=TRUE` vykresluje grafy horizontálně

`pars` seznam parametrů k volbě vzhledu krabicového grafu (velikosti "krabic", hodnoty "fousků"a extrémů, typy, šířky a barvy čar, velikosti bodů atd.), více pomocí příkazu `help(bxp)`

Příkazem `(boxplot())` získáme výstupní hodnoty:

`stats` matice, jejíž sloupce obsahují informaci o hodnotách dolního "fousu", dolní části krabice, mediánu, horní části krabice a horním "fousu".[1]

`n` vektor počtu pozorování každé skupiny

`conf` matice, jejíž sloupce obsahují horní a dolní extrémy výřezů (notch)

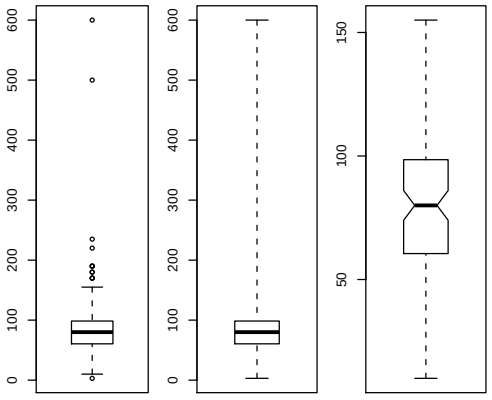
`out` odlehlé hodnoty

`group` vektor stejné délky jako `out`, jehož prvky uvádějí, do které skupiny patří odlehlé hodnoty

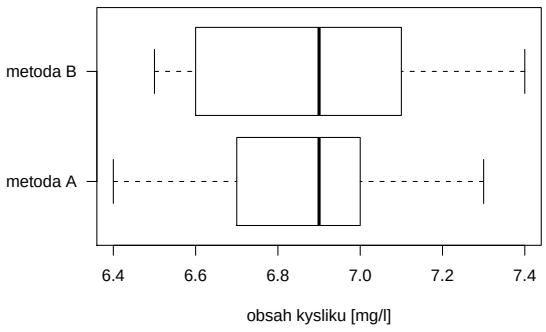
`names` vektor názvů skupin

```
> load(file="dusicnany.dat")
```
načtení datového souboru (obsah dusičnanů ve studniční vodě, [8])
```
> boxplot(dusicnany)
> boxplot(dusicnany, range=0)
> boxplot(dusicnany, notch=TRUE, outline=FALSE)

> load(file="kyslik.dat")
```
načtení datového souboru (porovnání dvou metod stanovení obsahu kyslíku ve vodě [mg/l], [8])
```
> boxplot(kyslik, names=c("metoda A", "metoda B"), horizontal=TRUE,
+ las=1, xlab="obsah kysliku [mg/l]")
```



Obr. 7.8. Obsah dusičnanů ve studniční vodě [mg/l]



Obr. 7.9. Porovnání dvou metod stanovení obsahu kyslíku ve vodě [mg/l]

[1] Záměrně jsou uvedeny pojmy jako dolní "fous", dolní část krabice, horní část krabice, horní "fous", protože implicitní nastavení (dolní hradba, dolní kvartil, horní kvartil, horní hradba) může být uživatelem změněno pomocí argumentů `range` a `pars` na jiné hodnoty.

```
> (boxplot(kyslik))    výstupní hodnoty
$stats
       [,1]  [,2]
 [1,]   6.4   6.5
 [2,]   6.7   6.6
 [3,]   6.9   6.9
 [4,]   7.0   7.1
 [5,]   7.3   7.4

$n
[1]  30   30

$conf
          [,1]      [,2]
 [1,]  6.81346  6.755766
 [2,]  6.98654  7.044234

$out
numeric(0)

$group
numeric(0)

$names
[1]  "metodaA"  "metodaB"
```

• `stripchart()` pro numerické vektory, seznamy a tabulky dat vykresluje diagram rozptýlení, používá se pro jednoduché zobrazení dat

`method` způsob zobrazení shodných dat, implicitní nastavení `method="overplot"` způsobuje překrývání shodných dat, `method="jitter"` zobrazuje náhodně kolem osy `y` a `method="stack"` skládá nad sebe

`jitter` v případě volby `method="jitter"` můžeme argumentem `jitter` nastavit šířku vykreslování kolem osy `y`

`offset` v případě volby `method="stack"` argument udává rozestup mezi jednotlivými vykreslenými body

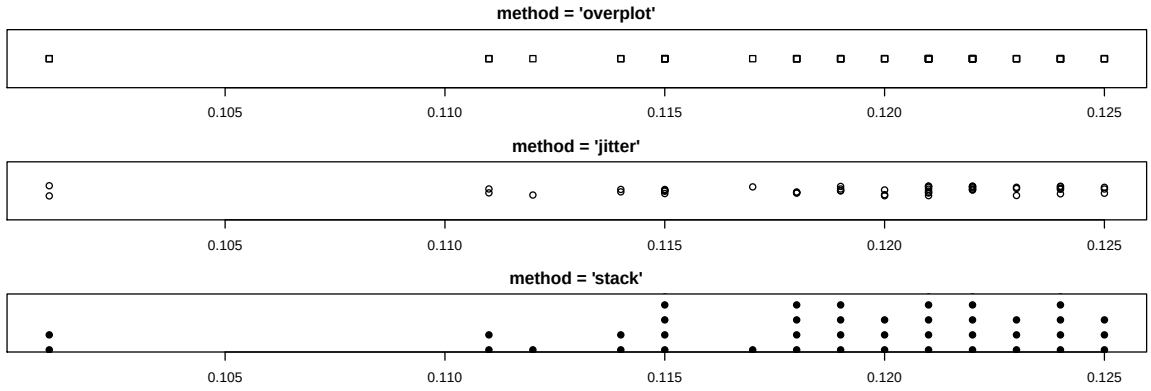
`vertical` pro hodnotu `vertical=TRUE` vykresluje vertikálně

`group.names` textový řetězec názvů skupin zobrazovaný po bocích grafu (při vykreslení více grafů)

`at` numerický vektor udávající pozici pro vykreslení bodů

```
> load(file="cekani.dat")
```
načtení datového souboru (doba čekání na zákazníka [min], [8])

```
> stripchart(cekani, method="overplot", main="method = 'overplot'",
+ cex=1)
> stripchart(cekani, method="jitter", main="method = 'jitter'", cex=1,
+ pch=1)
> stripchart(cekani, method="stack", main="method = 'stack'", cex=1.5,
+ pch=20, offset=0.4, at=0)
```
argument `cex` udává velikost znaků, `pch` typ znázorňovaných bodů, více viz odstavec 7.3



Obr. 7.10. Doba čekání na zákazníka [min]

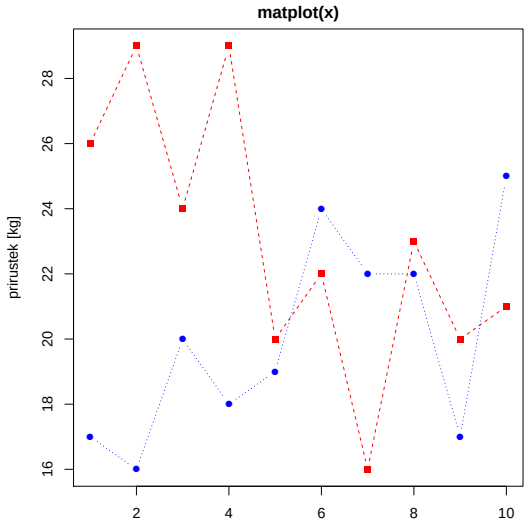
• `matplot()` vykreslení více datových řad do jednoho grafu. Vzhled grafu lze měnit níže uvedenými argumenty, implicitní vykreslení je pomocí číslic
`matplot(x)` pro vstupní matici `x` tvoří každou datovou řadu jeden sloupec matice
`matplot(x, y)` pro dvě vstupní matice stejných rozměrů udávají shodné pozice souřadnice bodů, datové řady opět tvoří sloupce matic. Lze vykreslit i pro vstupní vektor `x` stejné délky jako počet řádků matice `y`

`type, main, xlab, ylab, xlim, ylim`
`lty, lwd, pch, col, cex` viz odstavec 7.3

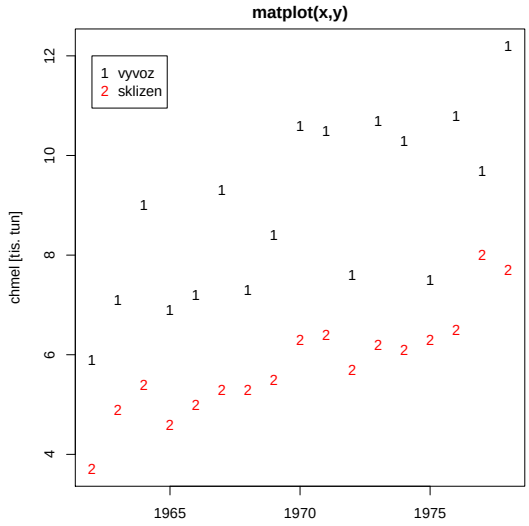
```
> load(file="smes.dat")
```
načtení datového souboru (vliv krmné směsi na přírůstek zvířat, [8])

```
> matplot(smes, main="matplot(x)", ylab="prirustek [kg]", type=c("b",
+ "b"), lty=c(2,3), pch=c(15, 16), col=c(2,4))
> legend(7, 29, c("smes 1", "smes 2"), lty=c(2,3), pch=c(15, 16),
+ col=c(2,4))
```
funkce pro zobrazení legendy, více viz odstavec 7.2

```
> time <- seq(from=1962, to=1978, length=17)
> load(file="chmel.dat")
```
načtení datového souboru (vývoz a sklizeň chmele v ČSSR v letech 1962 až 1978, [8])
```
> matplot(time, chmel, main="matplot(x,y)", ylab="chmel [tis. tun]")
> legend(time[1], 12, c("vyvoz", "sklizen"), pch=c("1","2"), col=1:2)
```



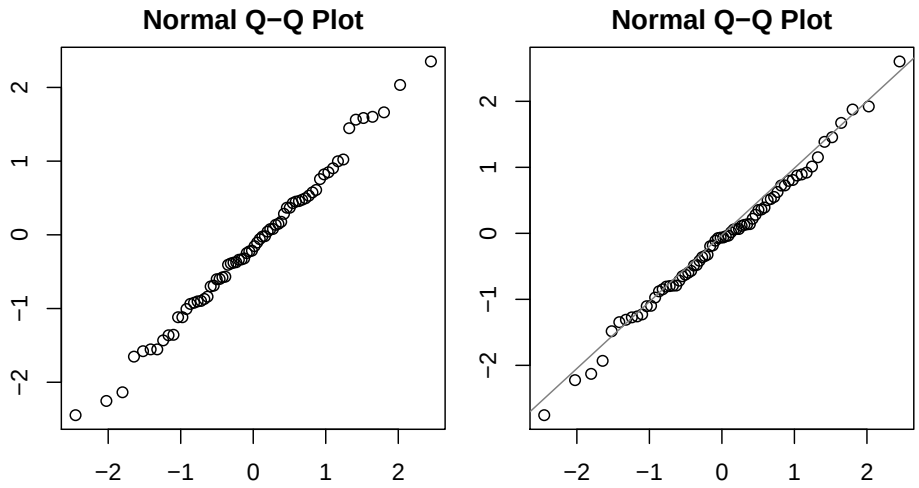
Obr. 7.11. Vliv krmné směsi na přírůstek zvířat [kg], [8]



Obr. 7.12. Vývoz a sklizeň chmele v ČSSR v letech 1962 až 1978

- `qqnorm(), qqline()` grafy pro ověřování normality dat. `qqnorm()` vykresluje normální kvantil-kvantilový graf (Q-Q plot), `qqline()` navíc přidá do již existujícího grafu přímku odpovídající normálnímu rozložení, jedná se o low-lewel funkci

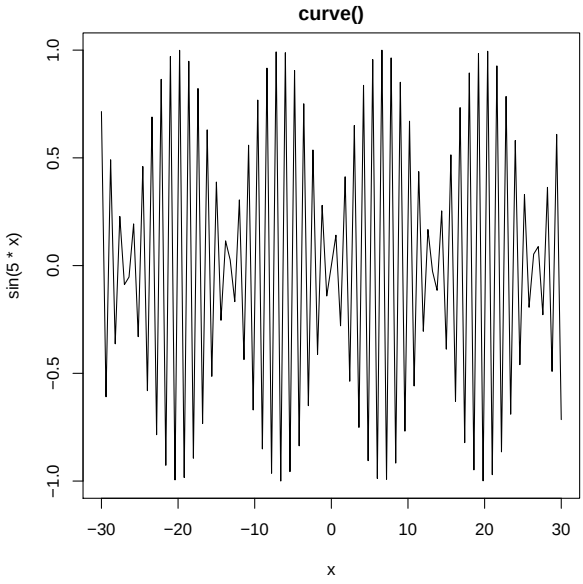
```
> qqnorm(rnorm(70))
> qqnorm(rnorm(70))
> qqline(rnorm(70), col=gray(0.5))
```
funkce `gray` slouží k vykreslení různé stupně šedi (více v odstavci 7.3)



Obr. 7.13. Q-Q plot pro výběr z normálního rozložení

- `curve()` vykreslí křivku
  `from, to` interval pro vykreslení funkce

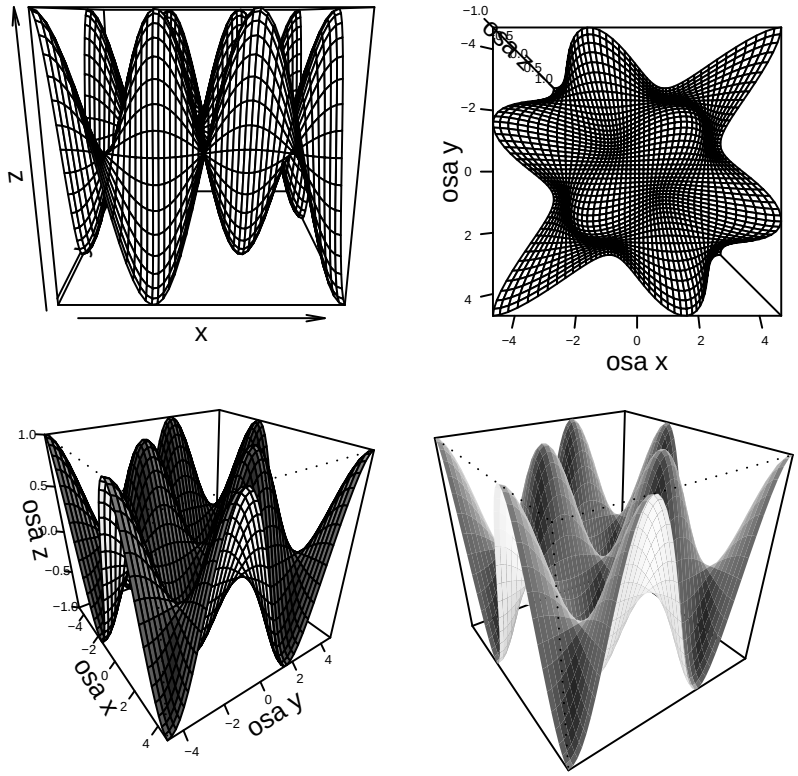
```
> curve(sin(5*x), from=-30, to=30, main="curve()")
```



Obr. 7.14. Funkce `curve()`

• `persp(x, y, z)` slouží k vykreslení 3D grafu
`xlim, ylim, zlim` intervaly pro vykreslování hodnot na osách x, y a z
`main, sub, col`
`theta` úhel pro otočení grafu v horizontálním smyslu
`phi` úhel pro otočení grafu ve vertikálním smyslu
`shade` stínování
`ticktype` typ os, `ticktype="simple"` (implicitní nastavení) vykreslí pouze šipky souběžné s osami grafu, šipky indikují směr nárůstu hodnot, `ticktype="detailed"` zobrazuje i měřítko os (počet značek na ose udává argument `nticks`)
`border` implicitní nastavení `NULL` vykresluje křivky, barvu vykreslovaných křivek můžeme měnit pomocí argumentu `col` nebo funkce `gray()`, nastavení `border=NA` vykreslení křivek zabraňuje

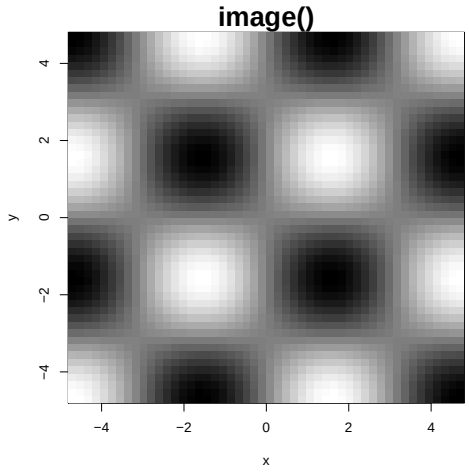
```
> persp(x, y, z)
> persp(x, y, z, ticktype="detailed", nticks=5, phi=270, xlab="osa x",
+ ylab="osa y", zlab="osa z")
> persp(x, y, z, ticktype="detailed", nticks=5, phi=30, theta=55,
+ shade=0.5, xlab="osa x", ylab="osa y", zlab="osa z")
> persp(x, y, z, shade=0.5, phi=30, theta=55, border=NA, axes=F)
```



Obr. 7.15. Funkce `persp()`

• `image()` vytvoří mřížku s obdélníky různého stupně šedi, slouží k zobrazení 3-dimenzionálních nebo prostorových dat

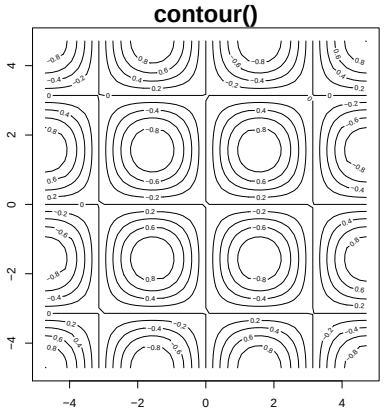
```
> x <- seq(-1.5*pi, 1.5*pi, length=50)
> y <- seq(-1.5*pi, 1.5*pi, length=50)
> z <- sin(x) %*% t(sin(y))
> image(x, y, z, col=gray(seq(from=0,to=1,length=80)), main="image()")
```



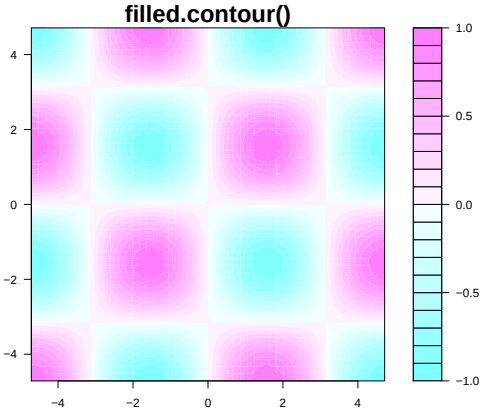
Obr. 7.16. Funkce `image()`

Funkce `contour()` a `filled.contour()` slouží k podobnému zobrazení jako funkce `image()`.

```
> contour(x, y, z, main="contour()")
> filled.contour(x, y, z, main="filled.contour()")
```



Obr. 7.17. Funkce `contour()`



Obr. 7.18. Funkce `filled.contour()`

Velký výběr dalších 3D grafů nabízí balíčky `misc3d`, `scatterplot3d` a `lattice`.

• `symbols()` na dané souřadnice vykreslí symboly - kružnice, čtverce, trojúhelníky, hvězdičky, více informací v nápovědě: `help(symbols)`

7.2 Low-level funkce

Někdy se stává, že grafické funkce nevykreslují přesný typ grafu, jaký bychom si přáli. V těchto případech je dobré použít tzv. low-level funkce. Pomocí nich vytváříme celý graf po samostatných částech přidáním dalších informací (body, čáry, text, ...) do již existujícího grafu.

Nejpoužívanější funkce pro tvorbu low-level grafiky (pro význam argumentů `bg`, `cex`, `col`, `lty`, `lwd` a `pch` viz odstavec 7.3):
• `points(x, y, type, pch, col, bg, cex, lwd, ...)` dle nastavení argumentů vykreslí body daných tvarů a barev na souřadnice `x`, `y`
• `lines(x, y, type, lty, lwd, ...)` vykreslí lomenou čáru mezi body danými souřadnicemi `x` a `y`
• `segments(x0, y0, x1, y1, col, lty, lwd, ...)` vykreslí úsečky mezi dvěma body o souřadnicích [`x0`, `y0`], [`x1`, `y1`]
• `abline(a, b, ...)` vykreslí přímku se směrnicí `b` a průsečíkem s osou y `a`
• `abline(h, ...)` vykreslí vodorovné úsečky přes celou šířku grafu, `h` udává hodnoty na ose y
• `abline(v, ...)` vykreslí svislé úsečky, `v` udává hodnoty na ose x
• `rect(xleft, ybottom, xright, ytop, col, ...)` vykreslí obdélník, jehož vrcholy jsou dány souřadnicemi `xleft`, `ybottom`, `xright` a `ytop`
• `polygon(x, y, col, ...)` vykreslí mnohoúhelník(y) s vrcholy danými body o souřadnicích `x` a `y`,
• `arrows(x0, y0, x1, y1, length, angle, code, col, ...)` vykreslí šipky směřující z bodu o souřadnicích [`x0`, `y0`] do bodu o souřadnicích [`x1`, `y1`], délku a úhel hlavy šipky uvádí argumenty `length` a `angle`, styl vykreslení šipky udává `code` (`0` - bez šipky, `1` - šipka na začátku, `2` - šipka na konci, `3` - šipka na obou stranách)
• `rug(x, ...)` "rohož"s rozestupem, slouží k vykreslení hustoty bodů
• `legend(x, y, legend, ...)` na souřadnice `x` a `y` vytiskne legendu (vektor textových řetězců). Funkce má velké množství volitelných argumentů (viz `help(legend)`)
• `title(main, sub, xlab, ylab, ...)` vypíše název, podtitulek, popisky os
• `grid(nx, ny)` vykreslí mřížku, `nx` a `ny` udávají počet vertikálních a horizontálních čar, implicitní nastavení k vykreslování mřížky (`col="lightgray", lty="dotted", lwd=par("lwd")`) lze samozřejmě libovolně měnit

• `axis(side, at, labels, tick, lty, lwd, col, ...)` vykreslí osu, umožňuje nastavit značky a popisky libovolně pro jednotlivé osy
  - `side` přirozené číslo specifikující stranu grafu, na které má být osa vykreslena (1 - dole, 2 - vlevo, 3 - nahoře, 4 - vpravo)
  - `at` body, ve kterých mají být vykreslovány značky pro měřítko
  - `labels` názvy pro jednotlivé značky
  - `tick` logická hodnota pro vykreslování značek
• `text(x, y, labels, ...)` vypíše textový řetězec `labels` do grafu na pozici [`x`, `y`]
• `mtext(text, side, ...)` vypíše textový řetězec `text` na okraj grafu (`side`)

7.3 Funkce `par()`

Funkce `par()` slouží k nastavení a změnám parametrů aktuálního grafu. Nastavení parametrů pomocí funkce `par()` mění hodnoty parametrů nastálo – ve všech dalších voláních libovolné grafické funkce až do doby zavření grafického okna nebo nastavení parametrů na nové hodnoty. S otevřením nového grafického okna jsou všechna předchozí nastavení ignorována.

Text
- `adj` zarovnání textu pro název grafu a popisky os, 0 - zarovnat vlevo, 1 - zarovnat vpravo, 0.5 - zarovnat horizontálně na střed
- `ann` implicitní hodnota `ann=TRUE` vypisuje anotace (název a popisky os), hodnota `FALSE` je potlačuje
- `cex` relativní velikost znaků (1 - normální velikost, 2 - dvojnásobná), `cex.main` nastaví velikost znaků pro název grafu, `cex.sub` pro podtitulek, `cex.axis`, `cex.lab` pro velikost písma popisků a názvů os

Barva
- `bg` barva pozadí grafu
- `col` barva bodů, obrysů bodů nebo spojnice. Barva může být vyjádřena přirozeným číslem nebo textovým řetězcem. Pro barvu os použijeme příkaz `col.axis`, pro popisky os `col.lab`, pro titulek a podtitulek grafu `col.main` a `col.sub`. Základní barvy jsou uvedeny v Tab. 7.1, podrobnější seznam barev o 657 položkách dostaneme příkazem `colors()`, funkcí `gray()` s argumenty od 0 do 1 získáme různé stupně šedi

barva	numerická hodnota	textový řetězec
bílá	0	"white"
černá	1	"black"
červená	2	"red"
zelená	3	"green"
modrá	4	"blue"
modrozelená	5	"cyan"
fialová	6	"magenta"
žlutá	7	"yellow"
šedá	8	"gray"

Tab. 7.1. Barvy bodů

Graf

`pty` textový řetězec specifikující tvar grafické oblasti, `"s"` čtvercová, `"m"` maximální

`mfcol` 2-prvkový vektor tvaru `c(pocet_radku, pocet_sloupcu)` rozdělující graf na odpovídající počet podgrafů, jednotlivé grafy jsou řazeny po sloupcích

`mfrow` stejně jako `mfcol`, grafy jsou řazeny po řádcích

`new` implicitní nastavení `FALSE`, nastavení na hodnotu `TRUE` umožňuje přikreslení dalšího grafu pomocí high-level funkce do již existujícího grafu

Osy

`las` numerická hodnota pro otočení popisků os:
0 - x vodorovně, y svisle
1 - x i y vodorovně
2 - x svisle, y vodorovně
3 - x i y svisle

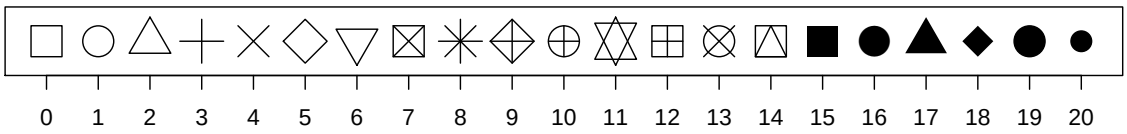
Body, čáry

`lty` numerická hodnota udávající typ čáry, jednotlivé typy s odpovídající hodnotou jsou uvedeny na Obr. 7.19

`lwd` numerická hodnota pro šířku čáry, standardní `lwd=1`

`pch` numerická hodnota pro znak vykreslovaných bodů, jednotlivé znaky s hodnotami `1` – `20` jsou uvedeny na Obr. 7.26. Pro hodnoty `21` – `25` jsou vykresleny znaky s hodnotou `1`, `0`, `5`, `2`, `6` (v tomto pořadí) s volitelnou barvou výplně `bg` a barvou hranice `col`. Hodnoty `26` – `31` nejsou definovány, hodnotám `32` – `255` odpovídají znaky ASCII tabulky. Je možno použít i jakékoliv textové řetězce, vypisuje se však vždy jen první znak.

Obr. 7.19. Typy čar (argument `lty`)



Obr. 7.20. Typy bodů (argument `pch`)

7.4 Další užitečné funkce

K otevření dalšího grafického okna slouží příkaz `dev.new()` nebo `windows()` (pouze v systému Windows). Příkaz `dev.new()` můžeme použít i s numerickým argumentem `which`, prostřednictvím něhož lze příkazy `dev.set(which)` a `dev.off(which)` dané okno aktivovat, popř. zavřít. Příkaz `graphics.off()` zavře všechna grafická okna.
Ukládat grafy lze v menu *File* → *Save as* výběrem požadovaného formátu. Alternativou je použít některého z příkazů:

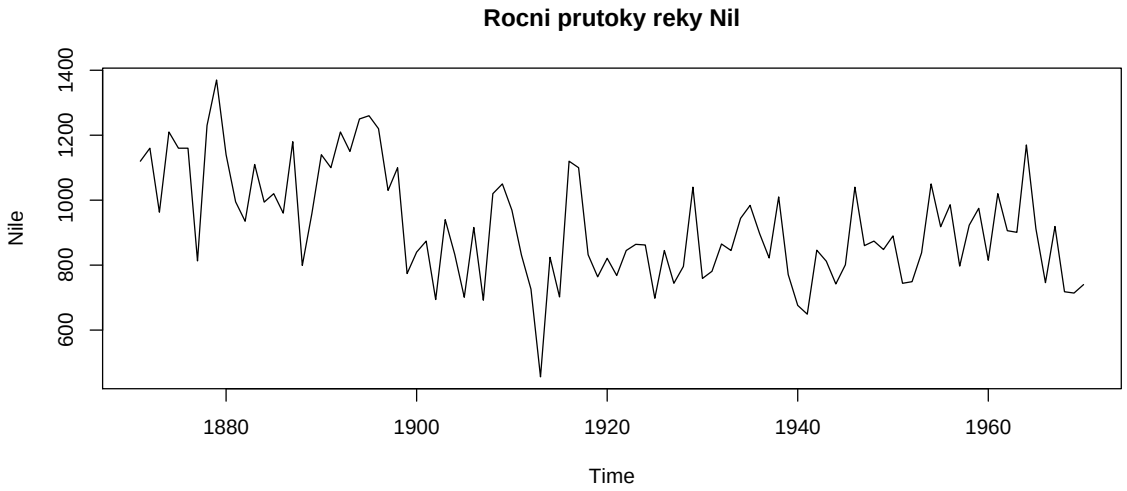
```
pdf(file, width, height)
postscript(file, width, height)
png(file, width, height)
jpeg(file, width, height)
```

`file` textový řetězec pro název souboru
`width` šířka grafu v palcích
`height` výška grafu v palcích

```
> pdf(file="Nil.pdf")
> plot(Nile, main="Rocni prutoky reky Nil")    vykreslení objektu Nile (ve-
stavěná proměnná)
> dev.off()
null device
          1
```

Příkazem `recordPlot()` lze grafy ukládat i jako objekty. Zpětně lze takto uložený graf zobrazit (a upravovat) příkazem `replayPlot()` nebo zavoláním názvu objektu.

```
> plot(Nile, main="Rocni prutoky reky Nil")
> Nil <- recordPlot()    uložení grafu do proměnné Nil
> replayPlot(Nil)    zobrazení grafu
```



Obr. 7.21. Roční průtoky řeky Nil

Funkce `layout()` umožňuje vytvářet složený graf obsahující několik grafů různých rozměrů, přičemž mohou zabírat i více řádků či sloupců.

Jediným povinným argumentem funkce je matice, jejíž počet řádků a sloupců a jejich číselné hodnoty určují pořadí zaplňování jednotlivých "boxů"grafy. Příkaz `layout(matrix(c(1, 0, 3, 2), 2, 2, byrow=TRUE))` vytvoří graf se čtyřmi boxy, přičemž pozice $[1, 1]$ bude obsazena jako první, dále budou obsazovány pozice $[2, 2]$ a $[2, 1]$, pro nulovou hodnotu zůstává pozice prázdná (pozice $[1, 2]$).

Argumenty `heights` a `widths` jsou používány ke specifikaci výšek a šířek boxů. Šablonu s jednotlivými boxy si můžeme prohlédnout příkazem `layout.show(n)` (Obr. 7.22), kde `n` udává počet vykreslovaných grafů.

```
> layout(matrix(c(1, 0, 3, 2), 2, byrow=T), widths=c(3, 1),
+ heights=c(1, 3))
> layout.show(3)
```

(Obr. 7.22)

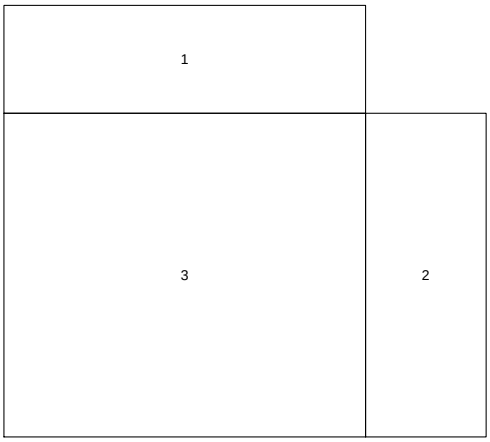
```
> load(file="pary.dat")
```

načtení datového souboru (závislost výšky 169 manželských párů, [8]). Soubor obsahuje proměnnou `pary` typu seznam se složkami `muz`, `zena`, `muzhist` a `zenahist`

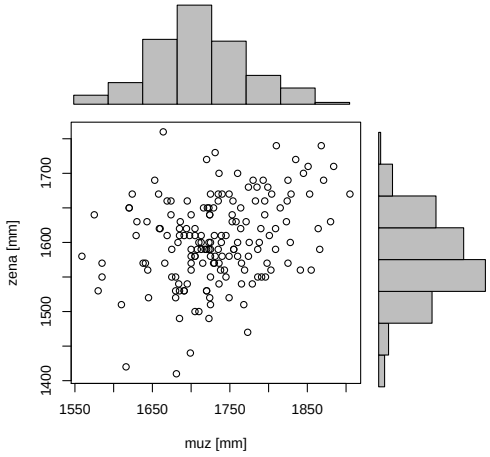
```
> attach(pary)
> layout(matrix(c(1, 0, 3, 2), 2, byrow=T), widths=c(3, 1),
+ heights=c(1, 3))
> par(mar=c(0, 3.5, 1, 1))
```

funkce `mar` slouží pro nastavení okrajů grafu

```
> barplot(muzhist, axes=F, space=0)
> par(mar=c(3.5, 0, 1, 1))
> barplot(zenahist, axes=F, space=0, horiz=T)
> par(mar=c(4.3, 4, 1, 1))
> plot(muz, zena, xlab="muz [mm]", ylab="zena [mm]")
```



Obr. 7.22. Šablona s boxy



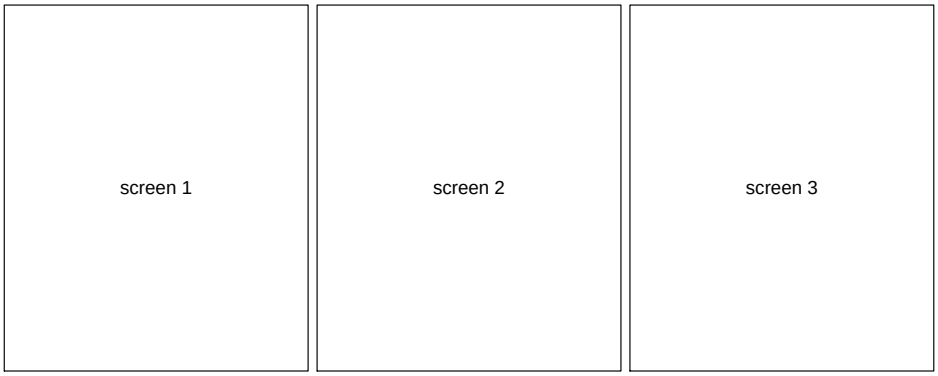
Obr. 7.23. Závislost výšky 169 manželských párů

Funkce `split.screen()` slouží k dělení grafického prostředí na libovolný počet částí. Např. příkazem `split.screen(c(2,2))` rozdělíme prostředí na čtyři části, na něž můžeme odkazovat příkazy `screen(1)`, . . . , `screen(4)` a pomocí těchto příkazů dané podgrafy vytvářet nebo měnit. Každá část přitom může být znovu rozdělena pomocí funkce `split.screen(c(,), screen)`.

Poznámka. Rozdělení grafu na jednotlivé podgrafy můžeme dosáhnout i pomocí argumentů `mfcol` a `mfrow` funkce `par()` (viz odstavec 7.3).

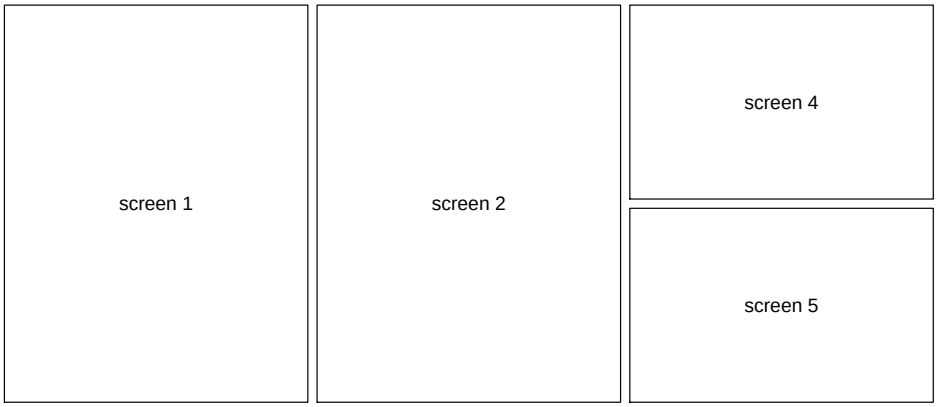
MATLAB používá k rozdělení grafu na více částí funkci `subplot`.

```
> load(file="etnika.dat")
```
načtení datového souboru (počet souhlasných reakcí na deset otázek zástupcům čtyř etnik, [8])
```
> popisx <- rownames(etnika)
> split.screen(c(1,3))
```
vytvoří prázdné grafické okno, navíc je výstupem i vektor hodnot odkazující na jednotlivé podgrafy, šablona okna s boxy viz Obr. 7.24
```
[1] 1 2 3
```



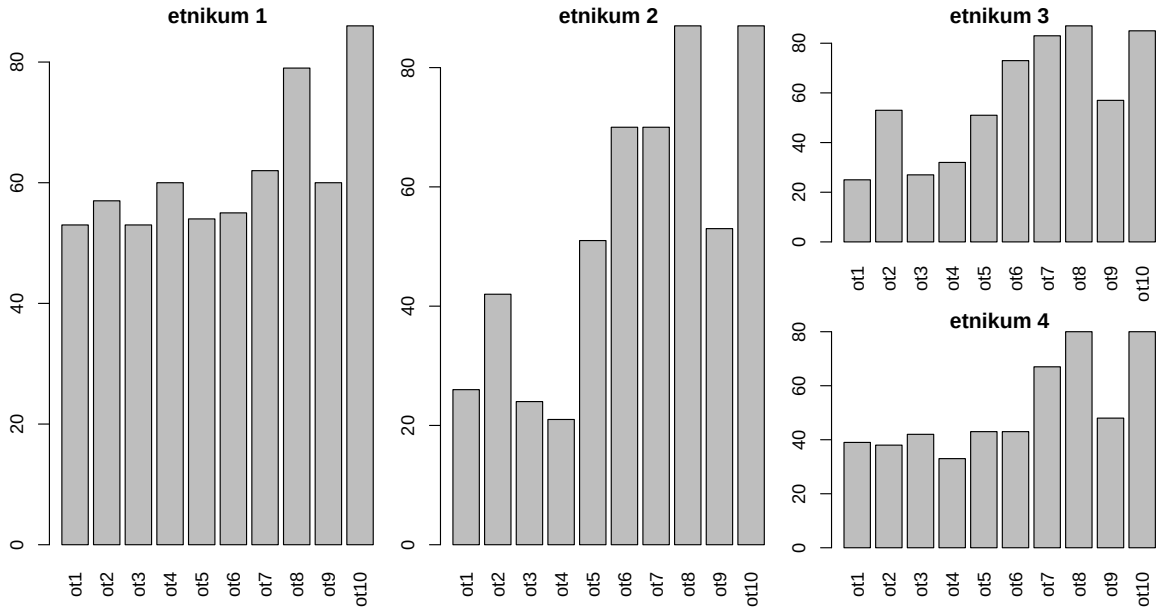
Obr. 7.24. Šablona okna s boxy pro příkaz `split.screen(c(1,3))`

```
> screen(1)
```
aktivace levého boxu pro vykreslení podgrafu
```
> barplot(etnika[,1], main="etnikum 1", names.arg=popisx, las=3)
> screen(2)
> barplot(etnika[,2], main="etnikum 2", names.arg=popisx, las=3)
> split.screen(c(2,1), screen=3)
```
rozdělení pravého boxu na 2 další boxy, aktuální šablona s boxy viz Obr. 7.25
```
[1] 4 5
```



Obr. 7.25. Šablona okna s boxy pro příkaz `split.screen(c(2,1), screen=3)`

```
> screen(4)
> barplot(etnika[,3], main="etnikum 3", names.arg=popisx, las=3)
> screen(5)
> barplot(etnika[,4], main="etnikum 4", names.arg=popisx, las=3)
```



Obr. 7.26. Počet souhlasných reakcí na deset otázek zástupcům čtyř etnik.

Interaktivní funkce `locator()` umožňuje vybrat pozici pro umístění grafických prvků (např. legendy, popisků os, . . .). Funkce `locator()` čeká, než uživatel zvolí levým tlačítkem myši místo, následně vypíše jeho souřadnice. Argument `n` udává počet bodů, u kterých chceme tímto způsobem zjistit souřadnice.

Příklady k procvičení

1. Do jednoho grafu zakreslete na intervalu $[0, 3\pi]$ graf funkcí $\sin x$ (červeně) a $\cos x$ (modře). Graf vhodně otitulkujte a pomocí funkce `locator()` obě funkce popište.

2. Vestavěná proměnná `InsectSprays` popisuje účinnost sprejů proti hmyzu a zahrnuje dvě složky – `count` a `spray`. Vykreslete histogram absolutních četností pro počet jedinců hmyzu daného pozorování (složka `count`).

3. Pro stejnou proměnnou jako v úkolu 2 vykreslete boxplot. Dále zjistěte, zda se data řídí normálním rozložením. Svou odpověď podložte vhodným grafem.

4. Do dvou grafů vedle sebe zobrazte graf funkce $\sin(x)\cos(x+y)$ pro x, y z intervalu $[0, 5]$. Do levého grafu zobrazte prostorový graf, do pravého pouze jeho vrstevnice. Grafy popište.

5. Načtěte soubor `vyskomer.dat`, ve kterém je uloženo 15 kontrolních měření dvěma různými výškoměry. Do jednoho grafu zakreslete průběh měření pro oba výškoměry následovně:

   - průběh pro `vyskomer1` modrými čtverečky,
   - průběh pro `vyskomer2` zelenými hvězdičkami,
   - graf opatřete názvem, popisky os a legendou.

   Graf uložte do souboru "vyskomer.pdf".

Řešení.

1. `x <- seq(from=0, to=3*pi, length=100)`, `y <- matrix(c(sin(x),cos(x)), ncol=2)`
   `matplot(x, y, col=c(2,4), type="l", main="Grafy funkci sin(x) a cos(x)", xlab="osa x", ylab="osa y")` nebo
   `plot(x, sin(x), col=2, type="l", main="Grafy funkci sin(x) a cos(x)", xlab="osa x", ylab="osa y")`, `points(x, cos(x), col=4, type="l")` nebo
   `plot(x, sin(x), col=2, type="l", main="Grafy funkci sin(x) a cos(x)", xlab="osa x", ylab="osa y")`, `lines(x, cos(x), col=4, type="l")`
   `souradnice <- locator(2)`
   `text(souradnice, labels=c("sin(x)", "cos(x)"))`

2. `hist(InsectSprays$count)`

3. `boxplot(InsectSprays$count)`
   `qqnorm(InsectSprays$count)`, `qqline(InsectSprays$count)`

4. `x <- seq(from=0, to=5, length=100)`, `y <- x`,
   `z <- sin(x) %*% t(cos(x+y))`
   `layout(matrix(1:2, nrow=1))`
   `persp(x, y, z, main="Funkce 'persp'")`
   `contour(x, y, z, main="Graf 'countour'")` nebo
   `split.screen(c(1,2))`
   `screen(1)`, `persp(x, y, z, main="Funkce 'persp'")`
   `screen(2)`, `contour(x, y, z, main="Graf 'countour'")`

5. `load(file="vyskomer.dat")`
   `pdf(file="vyskomer.pdf")`
   `matplot(1:15, vyskomer, col=c("blue", "green"), lty=c(1,2),`
   `pch=c(15, 8))`
   `legend(11, 34, c("vyskomer1", "vyskomer2"), col=c("blue","green"),`
   `pch=c(15, 8))`
   `dev.off()`

Kapitola 8

Programování v R

Základní informace

Jazyk R je snadno rozšiřitelný o vlastní funkce či dávkové soubory. Pro zvládnutí tvorby vlastních funkcí je třeba porozumět rozdílům mezi lokálními a globálními proměnnými a ovládat podmíněné příkazy a příkazy cyklů. Použití těchto příkazů může ovšem být u některých datových struktur zdlouhavé či výpočetně neefektivní, těmo situacím je věnována poslední část kapitoly popisující skupiny funkcí `apply`.

Výstupy z výuky

Studenti

- dokáží vysvětlit rozdíl mezi funkcemi a dávkovými soubory,
- rozlišují lokální a globální proměnné,
- ovládají různé podmíněné příkazy,
- umí použít příkazy cyklů,
- jsou schopni vhodně aplikovat funkci `apply`.

8.1 Funkce a dávkové soubory

Programy v jazyce R, které si uživatel může sám vytvořit, lze rozdělit do dvou skupin - dávkové soubory a funkce. V následujících odstavcích si uvedeme hlavní rozdíly mezi nimi a způsoby, jak lze tyto programy vytvořit.

Hlavním rozdílem funkcí a dávkových souborů je ten, že objekty definované uvnitř funkce nejsou k dispozici v pracovním prostoru. U dávkových souborů nezáleží na tom, z jakého adresáře jsou volány, objekty definované uvnitř dávky jsou v pracovním prostoru stále k dispozici. Dalším rozdílem je ten, že funkce pracují se vstupními proměnnými, dávkové soubory nikoliv.

Funkce
Každá funkce v R, ať vestavěná nebo definovaná uživatelem, je tvaru

`nazev <- function(argumenty){telo}`,

kde `nazev` udává název funkce, `argumenty` je čárkami oddělená posloupnost vstupních argumentů a `telo` obsahuje posloupnost příkazů, které musí být uzavřeny ve složených závorkách { } (pouze v případě jediného příkazu mohou být složené závorky vynechány).

Funkce lze vytvářet přímo v příkazové řádce, při tvorbě složitějších zdrojových kódů je vhodnější místo příkazové řádky používat editory. Jazyk R má k dispozici zabudovaný vlastní editor, který najdeme v menu *File* → *New script*. Možné je rovněž používat editory jako PSPad nebo WinEdt s číslovanými řádky, kontrolou syntaxe apod. Takto vytvořené funkce ukládáme do souboru s koncovkou `.R`.

V případě, že je funkce vytvořena přímo v příkazové řádce, spouští se zavoláním svého názvu `nazev` s argumenty uvedenými v kulatých závorkách. V případě, že je vytvořena v textovém editoru, je třeba nejdříve funkcí `source()` načíst soubor, ve kterém je uložena. Argumentem funkce `source()` je textový řetězec obsahující celý název souboru (i s koncovkou `.R`) nebo cestu k němu. Následně se spouští zavoláním názvu funkce `nazev` s argumenty uvedenými v kulatých závorkách.

Při volání funkce bez názvů argumentů musí být jednotlivé argumenty uvedeny přesně v tom pořadí, v jakém byly definovány. V opačném případě můžeme použít přiřazení `nazev_argumentu=hodnota`, popř. názvy argumentů můžeme zkracovat, jestliže zkratka nemůže znamenat žádný jiný argument. Hodnoty argumentů mohou mít definovány své implicitní hodnoty – ty lze příkazem `nazev_argumentu=implicitni_hodnota` nastavit v seznamu argumentů při definování funkce. Při volání funkce pak tyto argumenty mohou být vynechány a bude jim přidělena hodnota `implicitni_hodnota`.

Výstupní hodnoty získáme pomocí funkce `return()`, pouze k výpisu hodnot se používá funkce `print()`. V případě jediného výstupního objektu je tento objekt uváděn do argumentu funkce `return()`, v případě více výstupních objektů musí být seskupeny do seznamu.

Jazyk R rovněž umožňuje definování nových binárních operátorů (např. `%in%`). K odlišení od funkcí musí být jejich název ve tvaru `"%nazev%"`.

```
> kvadr <- function(a, b, c){
+ obsah <- 2*(a*b + b*c + c*a)
+ objem <- a*b*c
+ return(list(obsah=obsah, objem=objem))}
> vystup <- kvadr(1, 2, 3)
> vystup
$obsah
[1]  22

$objem
[1]  6
```

Dávkové soubory

Jedná se o posloupnost příkazů, která může využívat již definovaných objektů z pracovního prostoru. Zpravidla se vytváří v libovolném textovém editoru, název souboru musí mít příponu `.R` (např. `davka.R`). Dávkové soubory se spouští funkcí `source` s názvem dávkového souboru (nebo cestou k němu) uvedeným jako textový řetězec (např. `source("davka.R")`).

```
> a <- 1; b <- 2; c <- 3
> source("kvadr.R")   spuštění dávky kvadr.R
> obsah
[1]  2
> objem
[1]  6
```

Násilné ukončení běžící dávky nebo funkce se provádí příkazem `Q`.

V průběhu funkce nebo dávky se občas může stát, že se potřebujeme dostat zpět do pracovního prostoru, např. vypracování zadaných úkolů (viz cvičení) a následně dále pokračovat v běhu funkce či dávky. To lze příkazem `browser()`, k odlišení funkce či dávky a prostředí pracovního prostoru se prompt změní na `Browse[1]>`. Od této chvíle se nacházíme v pracovním prostoru, kde můžeme zadávat libovolné příkazy. Zpět se dostaneme příkazem `c` nebo klávesou ENTER.

MATLAB místo příkazu `browser()` používá `keyboard`, objevuje se prompt `K>>` a k návratu zpět do dávkového souboru používá příkaz `return`.

8.2 Lokální a globální proměnné

V jazyce R není nezbytně nutné deklarovat proměnné uvnitř funkce. Při vyhodnocování funkce R používá pravidlo *lexical scoping*, které rozhodne, zda je objekt lokální nebo globální proměnnou.

```
> funkce1 <- function(){print(x)}
> x <- 3
> funkce1()
```

Objekt `x` není definován uvnitř funkce `funkce1()`, proto R hledá v uzavřeném[1] prostředí objekt s názvem `x` a vytiskne jeho hodnotu. V případě, že by objekt `x` nebyl nalezen ani v globálním prostředí, zobrazilo by se chybové hlášení `Error in print(x) : object 'x' not found` a vyhodnocování funkce by tímto bylo u konce.

```
[1]  3
```

MATLAB by v případě, kdy objekt `x` není definován uvnitř funkce `funkce1()`, dával chybové hlášení, protože proměnná `x` je považována za lokální v prostředí, ve kterém byla definována (v tomto případě ve workspace).

```
> funkce2 <- function(){x <- 1; print(x)}
> x <- 3
> funkce2()
[1]  1
> x
```

Jestliže je `x` použito jako název objektu uvnitř funkce, hodnota `x` v globálním prostředí nebude změněna.

```
[1]  3
```

Chceme-li mít k dispozici i v globálním prostředí přiřazení provedená uvnitř funkce, musíme použít operátoru `<<-` nebo funkce `assign` s argumentem `envir=.GlobalEnv`.

```
> funkce3 <- function(){x <- 2; print(x)}
> x
```

přesvědčíme se, že objekt `x` není definován

```
Error: object 'x' not found
> funkce3()
[1]  2
> x
```

objekt `x` opravdu není v globálním prostředí definován, je definován pouze v prostředí funkce `funkce3`

```
Error: object 'x' not found
```

Rozdíl při použití operátoru pro globální přiřazení:

```
> funkce4 <- function(){assign("x", 2, envir=.GlobalEnv); print(x)}
> x
Error: object 'x' not found
> funkce4()
[1]  2
> x
[1]  2
```

8.3 Podmíněné příkazy

Do této chvíle jsme sestavovali programy, které vyhodnocovaly pouze jednoduché příkazy nebo jejich posloupnosti uzavřené ve složených závorkách. Často ovšem potřebujeme provést sérii příkazů až na základě výsledku nějaké podmínky/nějakých podmínek.

`if (podminka) {prikaz}`
provede `prikaz` v případě, že `podminka` je vyhodnocena jako `TRUE`. V případě vyhodnocení podmínky jako `FALSE` se žádné příkazy neprovedou.

```
> x <- 5
> if (x > 3) {"Podminka splnena."}
[1]  "Podminka splnena."
```

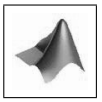
`if (podminka) {prikaz1}`
`else {prikaz2}`
V případě vyhodnocení podmínky `podminka` jako `TRUE` je proveden `prikaz1`, v opačném případě je proveden `prikaz2`.

```
> if (x > 3) {"Podminka splnena."}
+ else {"Podminka nesplnena."}
[1]  "Podminka splnena."
```

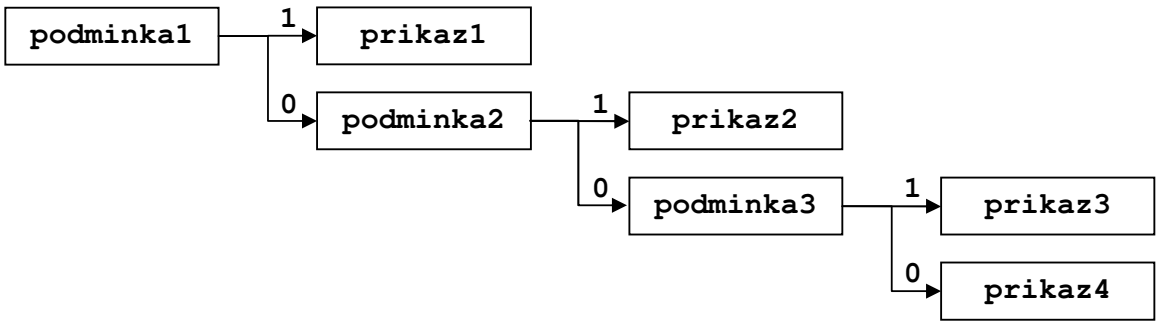
Podmínka vrací buď hodnotu `TRUE` nebo `FALSE`. Jestliže podmínka vrací logický vektor o více než jednom prvku, jako výsledek vyhodnocení podmínky je ovšem brán pouze jeho první prvek. Součástí výstupu je i hlášení:

```
> x <- c(-2, 0, 3, 1)
> if (x > 0) {"vetsi nez 0"}
+ else {"mensi nebo rovno 0"}
[1]  "mensi nebo rovno 0"
 Warning message:
In if (x > 0) { :
the condition has length > 1 and only the first element will be used
```

MATLAB by v tomto případě vyhodnotil podmínku vracející vektor o hodnotách `TRUE` jako `TRUE`, v případě jediné hodnoty `FALSE` jako `FALSE`.



Alternativou k příkazu `if - else` může být příkaz
`ifelse(podminka, prikaz1, prikaz2)`,
která v případě podmínky vyhodnocené jako `TRUE` provede příkaz `prikaz1`, v opačném případě provede `prikaz2`.



Obr. 8.1. Diagram vysvětlující příkaz `if - else if - else`

```
> ifelse(x > 3, "Podminka splnena.", "Podminka nesplnena.")
[1]  "Podminka splnena."
```

```
if (podminka1) {prikaz1}
else if (podminka2) {prikaz2}
else if (podminka3) {prikaz3}
else {prikaz4}
```

Vysvětlení těchto podmíněných příkazů je zachyceno na Obr. 8.1.

```
> x <- 3
> if (x == 1) {"cervena - stat!"}
+ else if (x == 2) {"oranzova - pripravit se"}
+ else if (x == 3) {"zelena - muzeme jet"}
+ else {"zadna jina barva na semaforu neexistuje"}
[1]  "zelena - muzeme jet"
```

Příkaz
`switch(vyraz, prikazy, prikazy_jine)`
slouží také k větvení programu. Jestliže `vyraz` je celé číslo, na výstupu je vrácena hodnota odpovídajícího prvku `prikazy`. Pokud nedojde ke spojení hodnoty `vyraz` s prvkem `prikazy`, funkce vrací hodnotu `NULL`.
Pro textový řetězec musí být název `vyraz` použit ke spojení s příslušným prvkem posloupnosti `prikazy`. Pokud nedojde ke spojení s některým z prvků posloupnosti `prikazy`, můžeme definovat `prikazy_jine`, které budou v tomto případě provedeny. `prikazy_jine` můžeme definovat pouze pro `vyraz` typu textový řetězec.
Syntaxe příkazu bude názornější z následujících příkladů:

```
> switch(2, "vyborne", "velmi dobre", "dobre", "uspokojive",
+ "dostatecne", "nedostatecne")
[1]  "velmi dobre"
> switch(7, "vyborne", "velmi dobre", "dobre", "uspokojive",
+ "dostatecne", "nedostatecne")
[1]  NULL
> switch("vyborne", vyborne="znamka A", velmi_dobre="znamka
+ B", dobre="znamka C", uspokojive="znamka D", dostatecne="znamka E",
+ neuspokojive="znamka F", "znamka neexistuje")
[1]  "znamka A"
> switch("chvalitebne", vyborne="znamka A",
+ velmi_dobre="znamka B", dobre="znamka C", uspokojive="znamka D",
+ dostatecne="znamka E", neuspokojive="znamka F", "znamka neexistuje")
[1]  "znamka neexistuje"
```

8.4 Příkazy cyklů

Někdy potřebujeme opakovaně vyhodnocovat příkaz nebo skupinu příkazů. Je třeba si dát pozor na správnou definici konce cyklu, v opačném případě může dojít k zacyklení. Mezi nejznámější příkazy cyklů patří funkce `for`, `while` a `repeat`.

`for (promenna in vyraz) {prikaz}`
Provede sérii příkazů `prikaz`, `promenna` označuje indexační proměnnou, `vyraz` může být vektor nebo seznam, pro jehož každou složku je `prikaz` vyhodnocen. Cyklus `for` je vhodné používat tehdy, když předem víme, kolik opakování chceme provést.

```
> for (i in 1:5) {cat("pruchod cyklem", i, "\n")}
pruchod cyklem 1
pruchod cyklem 2
pruchod cyklem 3
pruchod cyklem 4
pruchod cyklem 5
```

[2] funkce `cat()` zřetězí a vytiskne své argumenty

Někdy ovšem předem nevíme, kolikrát bude třeba cyklus opakovat, a počet opakování závisí na splnění určité podmínky. V těchto případech použijeme cyklus `while`:
`while (podminka) {prikaz}`
Stejně jako u výrazu `if`, `podminka` je výraz, který po vyhodnocení vrací logickou hodnotu `TRUE` nebo `FALSE`. V případě splnění podmínky `podminka` cyklus provede `prikaz`.

```
> i <- 1
> while (i %in% c(1:5)) {print(i); i <- i + 1}
[1]  1
[1]  2
[1]  3
[1]  4
[1]  5
```

Cyklus `repeat` opakuje vyhodnocování příkazů `prikaz` do té doby, dokud nedosáhne konce. Pro ukončení se používá příkaz `break`, který je součástí posloupnosti příkazů `prikaz`. Syntaxe vypadá následovně:
`repeat {prikaz}`

Poznámka. Příkaz `break` lze použít i pro ukončení cyklů `for` a `while`.

```
> i <- 0
> repeat{
+ print(i)
+ i <- i + 1
+ if (i == 5) {break}}
[1]  0
[1]  1
[1]  2
[1]  3
[1]  4
```

8.5 Skupiny funkcí `apply()`

Do této skupiny patří funkce `apply()`, `lapply()`, `sapply()` a `tapply()`. Společnou vlastností všech těchto funkcí je schopnost aplikovat funkci na vybrané části struktury bez použití cyklu, což má výhody v přehlednosti i rychlosti výpočtu.

Funkce `apply()` se používá pro vektory, matice a pole. Funkce požaduje tři argumenty: název pole a jeho dimenze, na kterých má být provedena požadovaná operace. Její název je třetím argumentem funkce. Jestliže u matic druhý argument nabývá hodnoty 1, operace se provádí po řádcích, v případě hodnoty 2 po sloupcích, u polí vyšší hodnoty odkazují na další dimenze.

```
> ar <- array(1:8, c(2,2,2))
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
> apply(ar, 1, sum)
[1] 16 20
> apply(ar, 2, sum)
[1] 14 22
> apply(ar, 3, sum)
[1] 10 26
```

Obě funkce `lapply()` i `sapply()` provádí výpočet určité funkce v jednotlivých složkách seznamu jako prvního argumentu. Druhým argumentem je název funkce, která má být provedena. Tyto příkazy se liší pouze výstupem - zatímco `lapply()` vrací seznam o stejném počtu složek jako vstupní seznam, funkce `sapply()` se snaží výsledek zjednodušit do vektoru nebo matice.

```
> (l <- list(1:5, 5:14))
[[1]]
[1] 1 2 3 4 5

[[2]]
[1]  5  6  7  8  9 10 11 12 13 14
> lapply(l, range)
[[1]]
[1] 1 5

[[2]]
[1]  5 14
> sapply(l, range)
     [,1] [,2]
[1,]    1    5
[2,]    5   14
```

Funkce `tapply()` aplikuje požadovanou funkci na roztříděná data v tabulce dat. Argument `INDEX` specifikuje seznam položek pro roztřídění.

```
> (tabulka <- data.frame(id=c(1:6), skupina=c(1, 2, 2, 1, 2, 1),
+ hodnota=runif(6, 2, 4)))
  id skupina   hodnota
1  1       1 3.238305
2  2       2 2.502754
3  3       2 3.939131
4  4       1 3.524773
5  5       2 3.731300
6  6       1 3.836787
> tapply(tabulka$hodnota, INDEX=tabulka$skupina, mean)
       1        2
3.533288 3.391061
```

Příklady k procvičení

1. Vytvořte dávku `odd.R`, uvnitř ní definujte vektor `v` s hodnotami 1, 3, 4, 5, 6, 7, 10 a výstupní proměnnou `pocet`, která vrátí počet lichých čísel vektoru `v`.

2. Vytvořte funkci `oddf.R`, která pro libovolný vstupní vektor či vrátí počet lichých čísel. Zkontrolujte, zda vstupní objekt obsahuje pouze numerické hodnoty a zda jsou všechny tyto hodnoty celá čísla.

3. Vytvořte funkci `mattovec.R`, která pro vstupní matici vypíše vektor jejích po sloupcích seřazených prvků. Zajistěte, aby pro diagonální matice byla vypisována pouze diagonála a pro dolní/horní trojúhelníkové matice pouze dolní/horní část.

4. Vytvořte funkci `product.R`, která pro dvě vstupní celočíselné hodnoty `n` a `k` vrátí jejich součin, aniž by byla použita operace násobení. Povolena je pouze operace sčítání.

5. Pro usnadnění práce rybářům vytvořte funkci `ryby.R`, která pro vstupní vektor představující hmotnosti postupně vylovených ryb rybáře upozorní, kdy je třeba dávat další ryby do nové kádě. Nosnost každé kádě je 20 kg. Kolik kádí budou rybáři pro svůj úlovek potřebovat (výstupní proměnná `kad`)? Ošetřete, aby váhy vylovených ryb byla pouze kladná reálná čísla.
   Funkci vyzkoušejte na náhodně vygenerovaném vektoru hmotností 25 ryb s hmotnostmi mezi 1–5 kg, hmotnosti zaokrouhlete na jedno desetinné místo.

6. Soubor `les.dat` obsahuje informace o šířce kmene 45 stromů ve dvou lokalitách lesa. Použijte vhodné příkazy pro zjištění průměrné, minimální a maximální šířky kmene pro každou z lokalit.

Řešení.

1. `v <- c(1, 3, 4, 5, 6, 7, 10)`

   ```
   sude <- v %% 2
   pocet <- sum(sude)

   cat('Pocet lichych cisel: ', pocet, '\n')
   ```
   ... dávku uložit s názvem `odd.R`, načítání a spouštění: `source("odd.R")`

2. `oddf <- function(x){`

   ```
   if (is.numeric(x) & is.real(x)){
     pocet <- sum(x %% 2)
   }

   return(pocet)
   }
   ```
   ... funkci uložit s názvem `oddf.R`,
   načítání funkce: `source("odd.R")`,
   spouštění funkce: `pocet <- oddf(v)`

3. `mattovec <- function(A){`

   ```
   if (sum(A - diag(diag(A))) == 0){
     vec <- diag(A)
   }
   else if (sum(A[!lower.tri(A, diag=T)]) == 0){
     vec <- A[lower.tri(A, diag=T)]
   }
   else if (sum(A[!upper.tri(A, diag=T)]) == 0){
     vec <- A[upper.tri(A, diag=T)]
   }
   else vec <- as.vector(A)

   return(vec)
   }
   ```
   `source("mattovec.R")`, `vec <- mattovec(matrix(1:4, 2))`

4. `product <- function(n, k){`

   ```
   if (round(n)-n != 0 | round(k)-k != 0){
     stop('Nejde o celociselne hodnoty!')
   }
   ```

```
else if (n==0 | k==0){
soucin <- 0
}
else {soucin <- 0
for (i in 1:k){
  soucin <- soucin + n
}}

return(soucin)
}
source("product.R"), soucin <- product(2, 3)
```

5.
```
ryby <- function(x){

kad <- 0

if (all(is.real(x) & x>0)){
  while (sum(x) >= 50){
    index <- max(which(cumsum(x) <= 20))
    x <- x[index:length(x)]
    kad <- kad + 1
  }
}
else stop("Nejedna se o vektor kladnych realnych cisel!")

return(kad)
}
source("ryby.R")
r <- round(runif(min=1, max=5, n=25), digits=1)
kad <- ryby(r)
```

6.
```
load("les.dat")
tapply(les$sirka, INDEX=les$lokalita, mean)
tapply(les$sirka, INDEX=les$lokalita, min)
tapply(les$sirka, INDEX=les$lokalita, max)
```

Seznam použité literatury

[1] BÍNA, V., KOMÁREK, A., KOMÁRKOVÁ, L. *Jak na jazyk R: instalace a základní příkazy.* [online] 2006. 18 s. [cit. květen 2010]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~komarek/Rko/Rmanual2.pdf>

[2] CRAWLEY, M. J. *Statistics: An Introduction Using R, Vectors And Logical Arithmetic.* [online] 30 s. [cit. květen 2010]. Dostupné z WWW: <http://osiris.sunderland.ac.uk/~cs0her/Statistics/Crawley/R2Calculations.pdf>

[3] DROZD, P. *Cvičení z biostatistiky: Základy práce se softwarem R.* [online] Ostrava: 2007. 111 s. ISBN 978-80-7368-433-4. [cit. květen 2010]. Dostupné z WWW: <http://cran.r-project.org/doc/contrib/CviceniR1.pdf>

[4] GUNDERSEN, V. B. *R for MATLAB Users* [online]. 2007 [cit. květen 2010]. Dostupné z WWW: <http://mathesaurus.sourceforge.net/octave-r.html>

[5] HIEBELER, D. *MATLAB/R Reference.* [online] 52 s. [cit. květen 2010]. Dostupné z WWW: <http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf>

[6] KLEIBER, Christian, ZEILEIS, Achim. *Applied Econometrics with R.* New York: Springer 2008. 221 s. ISBN 978-0-387-77316-2

[7] KOLÁČEK, Jan, ZELINKA, Jiří. *Jak pracovat s MATLABem.* [online] 40 s. [cit. květen 2010]. Dostupné z WWW: <http://www.math.muni.cz/~kolacek/vyuka/vypsyst/navod.pdf>

[8] MELOUN, Milan, MILITKÝ, Jiří. *Kompendium statistického zpracování dat: Metody a řešené úlohy včetně CD.* 1. vydání. Praha: Academia, 2002. 764 s. ISBN 80-200-1008-4

[9] PARADIS, Emmanuel. *R for Beginners.* [online] 2002. 58 s. [cit. květen 2010]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~kulich/vyuka/Rdoc/rdebuts_en.pdf>

[10] R Developement Core Team. *Writing R Extensions*. [online] 2009. 155 s. ISBN 3-900051-11-9 [cit. květen 2010]. Dostupné z WWW: <http://cran.r-project.org/doc/manuals/R-exts.pdf>

[11] SCOTT, Theresa A. *An Introduction to R*. [online]. 2004. 52 s. [cit. květen 2010]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~kulich/vyuka/Rdoc/RLectureTScott.pdf>

[12] SCOTT, Theresa A. *An Introduction To The Fundamentals & Functionality Of The R Programming Language: Part I: An Overview*. [online] 69 s. [cit. květen 2010]. Dostupné z WWW: <http://biostat.mc.vanderbilt.edu/wiki/pub/Main/TheresaScott/Scott.IntroToR.I.pdf>

[13] SCOTT, Theresa A. *An Introduction To The Fundamentals & Functionality Of The R Programming Language: Part II: The Nuts and Bolts*. [online] 101 s. [cit. květen 2010]. Dostupné z WWW: <http://biostat.mc.vanderbilt.edu/wiki/pub/Main/TheresaScott/Scott.IntroToR.II.pdf>

[14] SCOTT, Theresa A. *An Introduction To The Fundamentals & Functionality Of The R Programming Language: Section I: Some Language Essentials*. [online] 44 s. [cit. květen 2010]. Dostupné z WWW: <http://www.webpages.uidaho.edu/~brian/R_Scott_intro.pdf>

[15] SPECTOR, Phil. *Data Manipulation with R*. New York: Springer, 2008. 147 s. ISBN 978-0-387-74730-9

[16] VAVRČÍK, H. *Statistická analýza dat v aplikaci R*. [online][cit. květen 2010]. Dostupné z WWW: <http://wood.mendelu.cz/cz/sections/FEM/?q=book/export/html/49>

[17] VENABLES, W. K., SMITH, D. M. and the R Development Core Team. *An Introduction to R*.[online] 100 s. ISBN 3-900051-12-7. [cit. květen 2010]. Dostupné z WWW: <http://cran.r-project.org/doc/manuals/R-intro.pdf>

[18] VENABLES, W. N., RIPLEY, B. D. *Modern Applied Statistics with S-Plus*. 2. upravené vydání. New York: Springer-Verlag, 1994. 462 s. ISBN 0-384-94350-1

[19] *The R Project for Statistical Computing*. [online][cit. květen 2010]. Dostupné z WWW: <http://r-project.org/>

---

[20] The University Of Tennessee. [online][cit. květen 2010]. Dostupné z WWW: <http://www.utm.edu/departments/cens/engineering/Images/matlab_logo_000.gif>