

# Obsah

<b>I</b>	<b>Jak pracovat s MATLABem</b>	<b>3</b>
<b>II</b>	<b>Výuka jazyka R</b>	<b>4</b>
	Úvod	5
<b>1</b>	<b>První setkání s jazykem R</b>	<b>6</b>
	1.1 Základní ovládání . . . . .	7
	1.2 Nápověda . . . . .	8
	1.3 Workspace (pracovní prostor) . . . . .	9
	1.4 Datové typy objektů . . . . .	11
	1.5 Datové struktury . . . . .	12
<b>2</b>	<b>Vektory</b>	<b>14</b>
	2.1 Základní příkazy, tvorba vektorů . . . . .	15
	2.2 Subvektory . . . . .	18
	2.3 Délka a změna délky vektoru . . . . .	20
	2.4 Faktory . . . . .	22
<b>3</b>	<b>Matice a pole</b>	<b>25</b>
	3.1 Základní příkazy, tvorba matic a polí . . . . .	25
	3.2 Submatice . . . . .	28
	3.3 Funkce pro manipulaci s maticemi . . . . .	30
<b>4</b>	<b>Datové tabulky a seznamy</b>	<b>36</b>
	4.1 Základní příkazy, tvorba datových tabulek a seznamů . . . . .	36
	4.2 Podmnožiny seznamů . . . . .	38
	4.3 Funkce pro manipulaci s datovými tabulkami a seznamy . . . . .	40
<b>5</b>	<b>Konstanty, operátory a matematické výpočty</b>	<b>48</b>
	5.1 Aritmetické operátory . . . . .	49
	5.2 Porovnávací a logické operátory . . . . .	50

5.3	Množinové operátory . . . . .	51
5.4	Matematické funkce . . . . .	52
5.5	Zaokrouhlování . . . . .	55
5.6	Konstanty . . . . .	56
<b>6</b>	<b>Další příkazy v R</b>	<b>59</b>
6.1	Práce s knihovnamy . . . . .	59
6.2	Práce s daty . . . . .	60
6.3	Vlastnosti objektů . . . . .	63
<b>7</b>	<b>Grafika v R</b>	<b>68</b>
7.1	High-level funkce . . . . .	69
7.2	Low-level funkce . . . . .	84
7.3	Funkce <code>par()</code> . . . . .	85
7.4	Další užitečné funkce . . . . .	87
<b>8</b>	<b>Programování v R</b>	<b>94</b>
8.1	Funkce a dávkové soubory . . . . .	94
8.2	Lokální a globální proměnné . . . . .	96
8.3	Podmíněné příkazy . . . . .	98
8.4	Příkazy cyklů . . . . .	100
8.5	Skupiny funkcí <code>apply()</code> . . . . .	101
	<b>Seznam použité literatury</b>	<b>106</b>

# Část I

## Jak pracovat s MATLABem

**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 podpurných 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].



# 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 z paměti, 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 ukázkou 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 ukázkou 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  $\infty$

- 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 – faktorů. 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("navez", 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`.

Naprostο 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")`.

```
> 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.

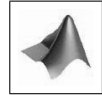
```
> vector(mode="logical", length=3)
[1] FALSE  FALSE  FALSE
> vector(m="character", length=5)
[1] ""     ""     ""     ""     ""
```

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 o délce shodné s délkou vektoru `v`.

Příkaz `seq()` vytváří prázdnou posloupnost.

```
> seq()
[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
> 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 na-

stavení 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  $\{1, 2, \dots, \text{length}(x)\}$ .

```
> 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))]   vek-
tor index může být zadán vektorem hodnot TRUE a FALSE
[1] 2 4 5 7 9 11 13 15
> x[x>7 & x<=12]   vektor index může být dán omezujícími podmínkami – po-
užitím 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
```

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
> u   přesvědčíme se, že vektor u opravdu obsahuje jen původní 3 složky
[1] 5 6 7
```

Pracujme opět s vektorem `u <- 5:12`. Analogickými příkazy k výše uvedenému může být `u[c(1,2,3)]`, `u[1:3]`, `head(u,3)`.

```
> (w <- head(u,3))
[1] 5 6 7
> length(w) <- 5
> w
[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. 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
```

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 pod-
mínku 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  $v_1$  od -10 do 0 s krokem 0.5.
5. Vytvořte vektor hodnot  $v_2$  obsahující sudé hodnoty posloupnosti -12, -11, ..., 11, 12.
  - a) Hodnoty vektoru  $v_2$  zdvojte tak, aby nově vzniklý vektor obsahoval hodnoty -12, ..., 12, -12, ..., 12.
  - b) Hodnoty vektoru  $v_2$  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, ekonomických vědách a statistice. Jejich uplatnění můžeme hledat např. v řešení systémů lineárních rovnic či v aplikaci maticových operací. Tato kapitola proto 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 nebo pole je použít příkazy `matrix()`, popř. `array()`:

`> matrix(u, 4, 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(u, 4, 5, byrow=TRUE)` vytvoří matici 4 x 5, argument `byrow=TRUE` stanovuje, že prvky jsou skládány 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
```

*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.

`> (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

Příkazy `matrix(u, ncol=5)`, `matrix(u, nrow=4)`, `matrix(u, 4)` a `matrix(u, c(4, 5))` jsou ekvivalentní, všechny seskládají po sloupcích vektor `u` do pěti sloupců a čtyř řádků.

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)`.



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("s11",
+ "s12", "s13", "s14", "s15")))
      s11 s12 s13 s14 s15
r1     1  4  9  13  17
r2     2  5  10 14  18
r3     3  6  11 15  19
r4     4  8  12 16  20
```

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

Argument `nazev.vektoru=vektor` slouží k pojmenování jednotlivých řádků (v případě `rbind`) a sloupců (v případě `cbind`):

```
> cbind(sl1=1:3, sl2=4:6)
  sl1 sl2
[1,]  1  4
[2,]  2  5
[3,]  3  6
```

Na rozdíl od MATLABu nelze v jazyce R matici o jednom sloupci získat transpozici 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 `index_3, ...,`

`index_n` na ostatní dimenze. 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

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

### 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
```

Pro maticové násobení se používá operátor `%*%`, oba činitelé musí být odpovídajících rozměrů (vnitřní rozměry obou činitelů musí být shodné). Výjimku tvoří násobení sloupcovým vektorem – může být nahrazen vektorem řádkovým.

```
> A %*% u   sloupcový vektor může být nahrazen vektorem řádkovým
      [,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
```

Příkaz `backsolve(A, b, k, upper.tri, transpose)` rovněž slouží k řešení lineárních rovnic s horní (`upper.tri=TRUE`, implicitně) nebo dolní trojúhelníkovou maticí (`upper.tri=FALSE`). Vektor  $b$  je vektor/matice pravých stran,  $k$  je počet sloupců matice  $A$ , které mají být použity. Pro `transpose=TRUE` řešíme systém  $A'x = b$ , implicitní nastavení je `transpose=FALSE`.

```
> backsolve(A, b, upper.tri=TRUE)  řeší systém lineárních rovnic  $\left( \begin{array}{cc|c} 1 & 1 & 3 \\ 0 & -1 & 1 \end{array} \right)$ 
```

```
  [,1]
[1,]   4
[2,]  -1
```

*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

- 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.
- 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.
- Vypište čtvrtý sloupec matice **A1** jako řádkový i sloupcový vektor.
- 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.
- Pomocí funkcí `rbind` nebo `cbind` vhodně doplňte hodnotami 1 matici **A3** na čtvercovou matici a spočítejte její determinant, stopu a hodnost.
- 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.

- 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("tvrдост", "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 o 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 střeďy).

Datovou tabulku vytvoříme příkazem `data.frame()`. Argumenty `nazev_1=vektor_1`, `nazev_2=vektor_2`, ... specifikujeme názvy sloupců (proměnných) a jejich hodnoty, názvy sloupců jsou nepovinné, stačí zadat pouze hodnoty. 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`, ...

```
> (1 <- 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 dimenze tabulky `dat`, 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 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 specifikujeme jejím číslem. K výpisu více složek můžeme použít operátor `:` nebo funkci `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 dvojitéh 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 uvozkách 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", "utory", "streda"))
> tab[1]
      cervena
pondeli      1
utory        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 nastalo 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 tvaru `seznam$nova_slozka <- objekt` nebo `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.

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

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

K odstranění řádků můžeme použít i operátor [,].

```
> (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.

Alternativou k funkcím `cbind()` a `rbind()` může být funkce `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 kte-
rého 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 tabulek dat se používá funkce `order()`, která vrací vektor indexů vzestupně (implicitně) nebo sestupně uspořádaných prvků. Pro více informací o funkci `order` viz odstavec 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),] seřazuje podle sloupce
k_dispozici, v případě vícenásobných hodnot řadí 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
> knihy[order(knihy$k_dispozici, -knihy$pocet),] pokud některý ze sloupců
tabulky tvoří numerický vektor, pro sestupné uspořádání tohoto sloupce můžeme pou-
žít operátoru -. Rovněž si můžeme všimnout, že v případě nejednoznačných podmínek
na seřazení (řádek č. 1 a 4) dostává přednost řádek s nižším pořadovým číslem, u názvů
řádků se postupuje podle abecedního uspořádání.
      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 nové 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

## KAPITOLA 4. DATOVÉ TABULKY A SEZNAMY

---

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)
```

## KAPITOLA 4. DATOVÉ TABULKY A SEZNAMY

---

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", "Plechý", "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 / 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

|                                         |                                                                                                                                                                                                                                              |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>all(relace)</code>                | testuje, zda jsou všechny složky <code>relace</code> pravdivé                                                                                                                                                                                |
| <code>any(relace)</code>                | testuje, zda je alespoň jedna složka <code>relace</code> pravdivá                                                                                                                                                                            |
| <code>which(relace)</code>              | vrací pořadí těch složek <code>relace</code> , které jsou pravdivé (popř. které splňují danou podmínku). V případě polí můžeme použít argument <code>arr.ind=T</code> pro výpis hodnot v podobě čísel řádků a sloupců, popř. dalších dimenzí |
| <code>x %in% y, is.element(x, y)</code> | testuje, zda je <code>x</code> podmnožinou množiny <code>y</code> , vrací logické hodnoty                                                                                                                                                    |
| <code>intersect(mnoziny)</code>         | průnik množin                                                                                                                                                                                                                                |
| <code>union(mnoziny)</code>             | sjednocení množin                                                                                                                                                                                                                            |
| <code>setdiff(x, y)</code>              | rozdíl množin, vrací ty prvky vektoru <code>x</code> , které nejsou obsaženy v <code>y</code>                                                                                                                                                |

```
> x <- 1:10
> y <- 2:9
> z <- -5:5
> all(z)      testuje, zda jsou všechny prvky vektoru z nenulové
[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 číslo  $n$  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 funkce `sum(x)` a `prod(x)` vrací součet, resp. součin všech prvků.

MATLAB aplikuje funkce `sum()` a `prod()` na matice 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:

|                        |                                                                                                                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ceiling()</code> | zaokrouhlení k nejbližšímu většímu celému číslu                                                                                                                                        |
| <code>floor()</code>   | zaokrouhlení k nejbližšímu menšímu celému číslu                                                                                                                                        |
| <code>trunc()</code>   | zaokrouhlení směrem k 0, celá část daného čísla                                                                                                                                        |
| <code>round()</code>   | zaokrouhlení k nejbližšímu celému číslu, parametrem <code>digits=pocet</code> volíme počet desetinných míst, na jaký má být zaokrouhlení provedeno (implicitně <code>digits=0</code> ) |
| <code>signif()</code>  | zaokrouhlení na určitý počet platných cifer (parametr <code>digits</code> , 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` ...  $\pi$ , 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  $v_1$  s hodnotami 1, 3, 4,  $v_2$  jako posloupnost délky 5 s počáteční hodnotou -6 a krokem 2 a  $v_3$  s prvními třemi složkami vektoru  $v_2$ .
  - a) Proveďte součet, rozdíl, součin a podíl vektorů  $v_1$  a  $v_3$ .
  - b) Vysvětlete výsledky, pokud operace z předchozího zadání použijete na vektory  $v_1$  a  $v_2$ .
  - c) Uveďte celou část a zbytek po dělení vektoru  $v_1$  vektorem  $v_3$ .
  - d) Uveďte pozice vektoru  $v_1$ , jejichž hodnoty jsou různé od hodnot vektoru  $v_3$  na odpovídajících si pozicích.
  - e) Uveďte počet prvků vektoru  $v_2$ , jehož hodnoty jsou menší než -3 nebo rovny 1.
  - f) Spočítejte součin nenulových hodnot vektoru  $v_2$ .
  - g) Zjistěte, zda je alespoň jeden prvek vektoru  $v_1$  nulový.
2. Ve třech laboratořích byly analyzovány různé vlastnosti kávy: obsah vody ( $x_1$ ), hmotnost zrn ( $x_2$ ), pH ( $x_3$ ), tuky ( $x_4$ ), kofein ( $x_5$ ), obsah minerálů ( $x_6$ ) a extrakt ( $x_7$ ). V první laboratoři `lab1` byly pozorovány vlastnosti  $x_1$ ,  $x_2$ ,  $x_3$ ,  $x_6$ , ve druhé



## KAPITOLA 5. KONSTANTY, OPERÁTORY A MATEMATICKÉ VÝPOČTY

---

laboratoři lab2 vlastnosti  $x_1$ ,  $x_3$ ,  $x_4$ ,  $x_5$  a ve třetí laboratoři lab3 vlastnosti  $x_1$ ,  $x_2$  a  $x_5$ . Zjistěte, které vlastnosti

- byly analyzovány ve všech třech laboratořích,
- byly analyzovány v první i druhé laboratoři, ale ne ve třetí,
- nebyly analyzovány ani jednou z laboratoří,
- byly analyzovány alespoň jednou z laboratoří.

3. Je dána matice  $A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 1 \\ 8 & -27 & 1 \end{pmatrix}$ . Spočítejte její spektrální poloměr, tj. největší vlastní číslo v absolutní hodnotě.

- Kolika způsoby můžeme z krabičky 20 kuliček vybrat právě 5? Kolika způsoby bychom mohli vybrat 5 kuliček, záleželo by-li na jejich pořadí?
- Spočítejte směrnici tečny ke grafu funkce, víte-li, že daná tečna svírá s kladnou poloosou  $x$  úhel  $60^\circ$ .
- Náhodně vygenerujte vektor 10 celých čísel z intervalu  $[-2, 3]$ .
- Náhodně vygenerujte vektor  $v_4$  šesti hodnot z intervalu  $[1, 4]$  a zaokrouhlete jej na:
  - 3 desetinná místa,
  - 3 platné cifry,
  - směrem k nule.
- Vypište posledních 5 velkých písmen abecedy.
- Vypište anglické názvy měsíců, jejichž pořadí odpovídá násobkům čísla 3.

*Řešení.*

- ```
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)
```

## KAPITOLA 5. KONSTANTY, OPERÁTORY A MATEMATICKÉ VÝPOČTY

---

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 podpurných 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 externími, vlastními datovými soubory. Pro práci s takovými soubory je potřeba je umět načíst, součástí této kapitoly jsou funkce pro 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 knihovnamy, 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 knihovnamy

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 načítání dat

`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
```

## KAPITOLA 6. DALŠÍ PŘÍKAZY V R

---

```
[1] "2" "4" "6" "1" "3" "5"
```

Další takovou funkcí je `read.table(file, header, sep, dec, row.names, col.names)`, která načítá data do tabulky dat (`data.frame`)

`file, sep, dec` viz funkce `scan()`

`header` logická hodnota, zda je v datech obsažena hlavička

`row.names` textový řetězec názvů řádků nebo číslo odkazující na sloupec názvů

`col.names` vektor názvů sloupců

```
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("s11", "s12",
+ "s13", "s14", "s15", "s16"))
```

 přestože poslední sloupec obsahuje názvy řádků, argument `col.names` musí obsahovat jeho název i tohoto sloupce ("`s16`"), i když nebude vytištěn

```
  s11 s12 s13 s14 s15
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
```

```
> load(file="dunaj.dat")
```

 načtení dat ze souboru `dunaj.dat` (data znázorňující kolísání [m<sup>3</sup>/s] Dunaje během roku, [8])

```
> dunaj
```

```
[1] 1987 1728 1862 2083 2143 2187 2588 2224 2001
[10] 1767 1460 1444
```

Pro usnadnění práce s daty jazyk R obsahuje okolo sta vestavěných datových souborů (v balíčku `datasets`). Funkce `data()` zobrazí seznam dostupných datových sou-

borů. K zobrazení vybraných souborů slouží příkaz `data(nazev, package)`<sup>1</sup>, 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
```

### Ukládání dat

K ukládání objektů `object` do souboru specifikovaného argumentem `file=""` slouží funkce `save(object, file="")`. Objekt může být následně načten pomocí funkce `load`.

```
> l <- list(v=1:5, mat=matrix(2:7, 3))   pro ukládání více objektů současně je
vhodné seskupit tyto objekty do seznamu
> save(l, file="seznam.txt")
> nacteni <- load("seznam.txt")   načtení uloženého souboru seznam.txt, který
obsahuje proměnnou l
> nacteni
[1] "1"
> l
$v
```

---

<sup>1</sup>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.

```
[1] 1 2 3 4 5
```

```
$mat
      [,1] [,2]
[1,]    2    5
[2,]    3    6
[3,]    4    7
```

Posloupnost jednotlivých příkazů i s jejich výstupy můžeme uložit v menu *File* → *Save to File...* Historii příkazů můžeme uložit v menu *File* → *Save History* nebo příkazem `savehistory(file=".RHistory")`, příkazem `loadhistory(file=".RHistory")` ji následně můžeme načíst. V případě, že otevíráme dříve uložený soubor, historie příkazů je načtena automaticky s ním.

## 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)

```
> (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



## KAPITOLA 6. DALŠÍ PŘÍKAZY V R

---

```
> 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 jednotlivých barev během 4 časových období."
nastavení komentáře k objektu tab
```

```
> comment(tab)
zobrazení komentáře
```

```
[1] "Pocty aut jednotlivých barev během 4 časových období."
```

`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("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) Převedte 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`. Vypisem 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 nepřehledné 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 nejrozumnější 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, přepíše 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:

<code>axes</code>	implicitní hodnota <code>TRUE</code> , nastavení na hodnotu <code>FALSE</code> potlačuje vykreslování os
<code>log</code>	nastavením na hodnoty <code>log="x"</code> , <code>log="y"</code> , <code>log="xy"</code> budou mít vybrané osy logaritmické měřítko

<code>main</code>	textový řetězec pro název grafu, je umístěn nad grafem
<code>sub</code>	textový řetězec, je umístěn pod grafem a psán menším fontem
<code>type</code>	typ výstupního grafu <code>type="p"</code> vykresluje samostatné body (implicitní nastavení) <code>type="l"</code> linie <code>type="b"</code> přerušované linie s body <code>type="c"</code> přerušované linie bez bodů <code>type="o"</code> body navzájem spojené liniemi <code>type="h"</code> vertikální linie <code>type="s"</code> schodovitý graf s první linií horizontální <code>type="S"</code> schodovitý graf s první linií vertikální <code>type="n"</code> žádné vykreslování dat, vykresleny jsou pouze osy, rozsah souřadného systému závisí na datech
<code>xlab, ylab</code>	textové řetězce pro názvy os $x$ a $y$ , tyto argumenty mění implicitně zadané názvy os při volání high-level funkcí
<code>xlim, ylim</code>	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 příloženém 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  $y/x$

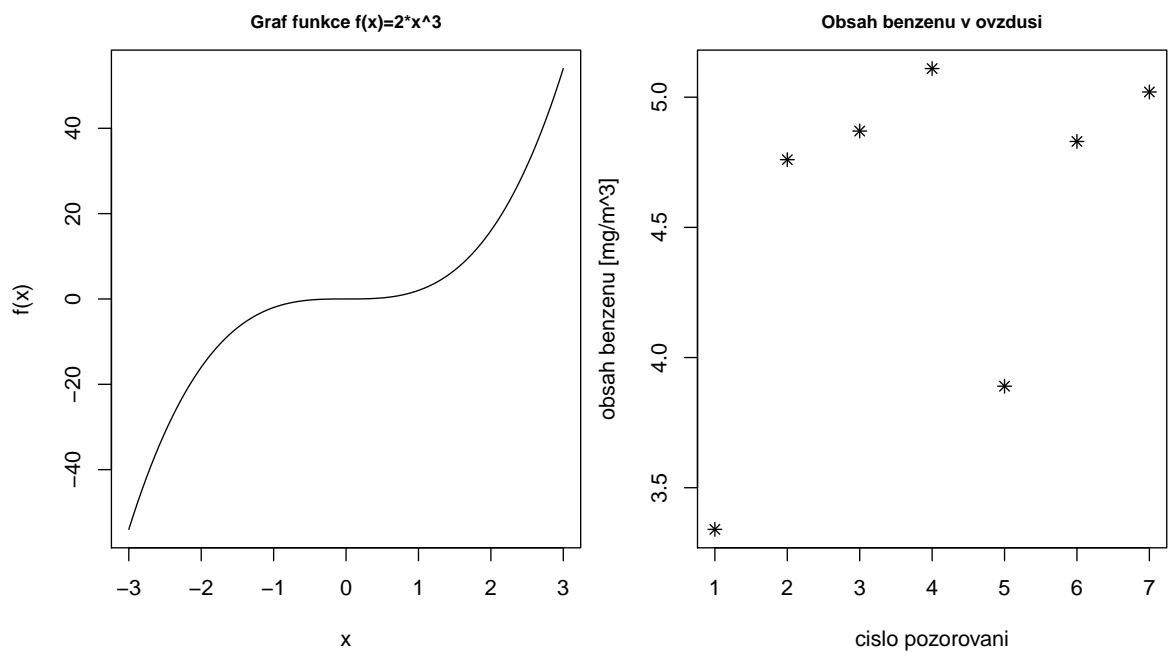
## KAPITOLA 7. GRAFIKA V R

---

```
> plot(function(x) 2*x^3, xlim=c(-3, 3), ylab="f(x)", main="Graf funkce  
+ f(x)=2*x^3")
```

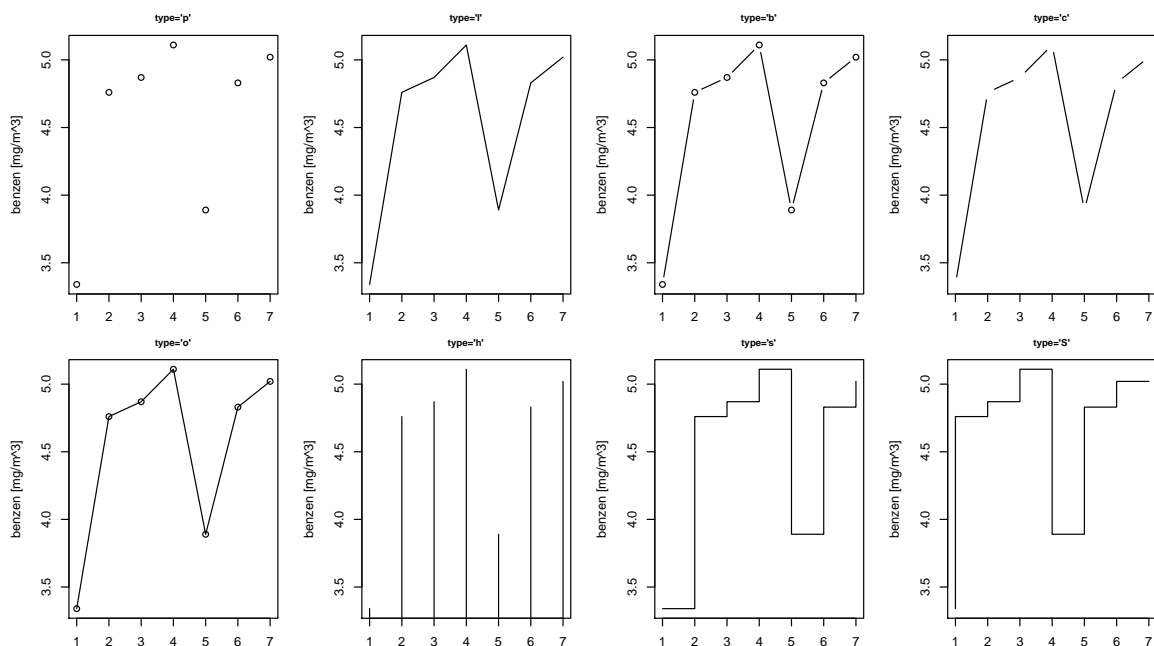
```
> benzen <- c(3.34, 4.76, 4.87, 5.11, 3.89, 4.83, 5.02) sedm měření pro  
kontrolu obsahu benzenu v ovzduší [mg/m^3], [8]
```

```
> 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 `width` 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 matici

`horiz` implicitní hodnota `horiz=FALSE` vykresluje obdélníky vertikálně, argument `horiz=TRUE` horizontálně

`density` hodnota nebo numerický vektor nastavující hustotu šrafování sloupců

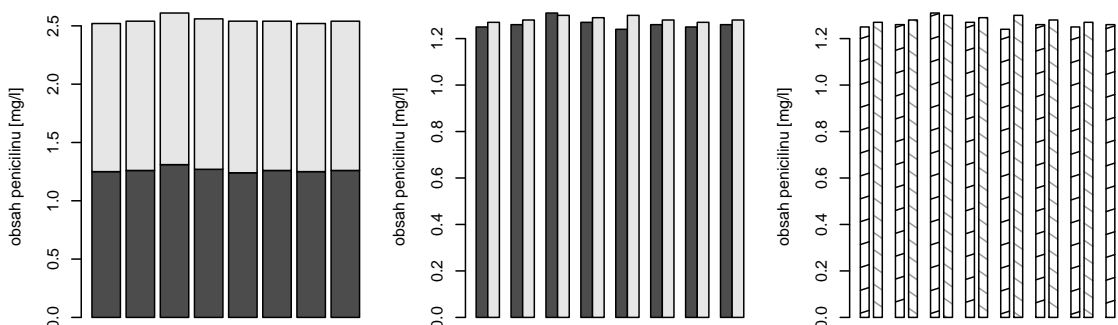
`angle` úhel pro sklon šrafování



## KAPITOLA 7. GRAFIKA V R

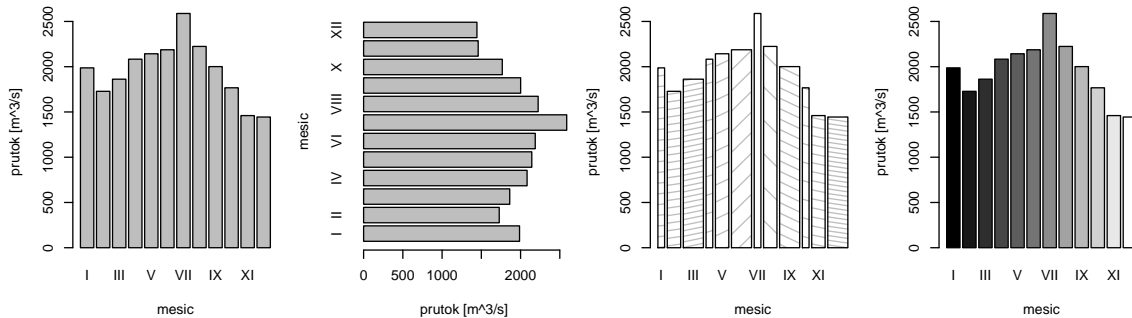
---

```
> 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 [m3/s] během roku, [8])
> barplot(dunaj, names.arg=c("I", "II", "III", "IV", "V", "VI", "VII",
+ "VIII", "IX", "X", "XI", "XII"), ylab="prutok [m3/s]",
+ xlab="mesic")
> barplot(dunaj, horiz=T, names.arg=c("I", "II", "III", "IV", "V",
+ "VI", "VII", "VIII", "IX", "X", "XI", "XII"), ylab="prutok
+ [m3/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
+ [m3/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 [m3/s]", xlab="mesic")
```

Obr. 7.4. Kolísání průtoku Dunaje během roku [ $\text{m}^3/\text{s}$ ]

- `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

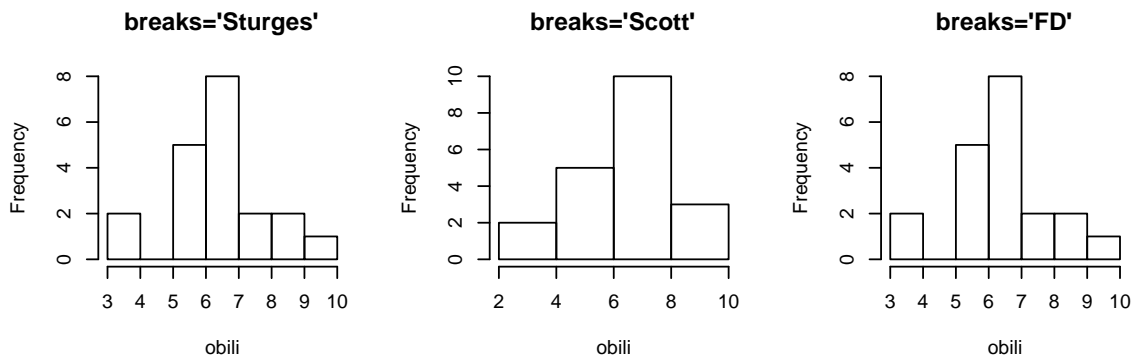
`intensities` shodné s `density`

`mids` středy intervalů

`xname` název objektu v argumentu

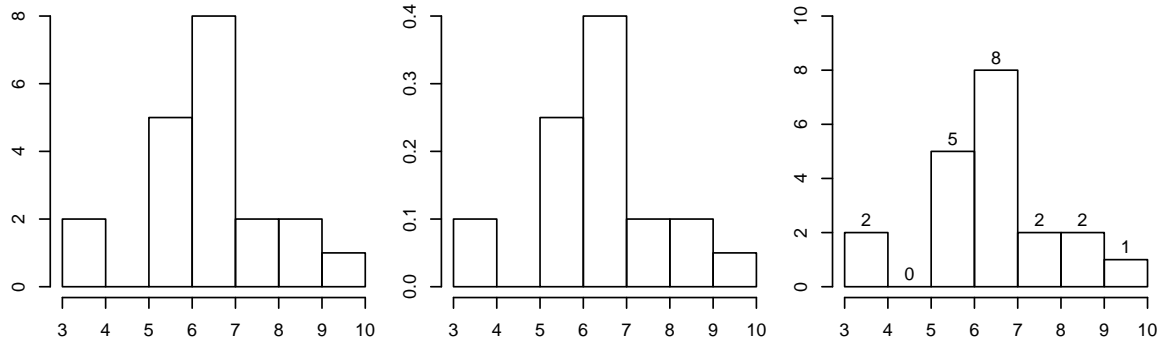
`equidist` logická hodnota stejné délky všech intervalů

- ```
> load(file="obili.dat")    načtení souboru dat (sklizňová ztráta obilí [g/m³], [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/m<sup>3</sup>], argumenty šířky intervalů

```
> hist(obili, main="")
> hist(obili, freq=F, main="")
> hist(obili, labels=T, main="")
```



Obr. 7.6. Sklizňová ztráta obilí [g/m<sup>3</sup>], zobrazení absolutních a relativních četností

```
> (hist(obili))  výstupní hodnoty
$breaks
[1] 3 4 5 6 7 8 9 10
$count
[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
```

```

$mid
[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"

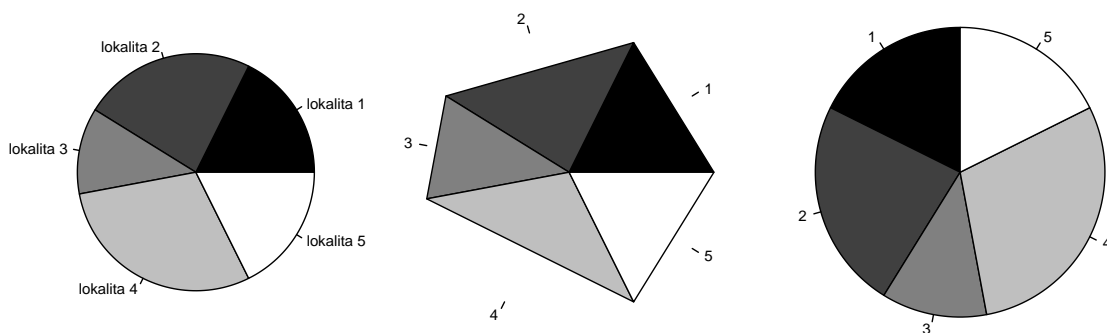
```

- `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]

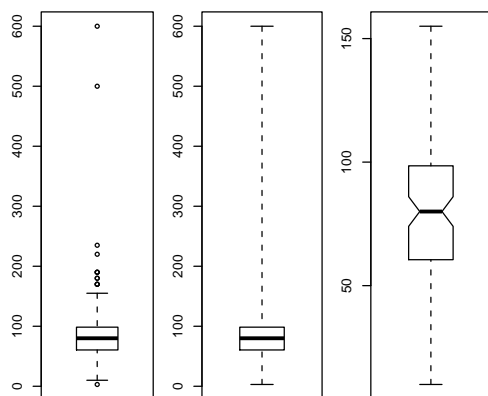
- `boxplot()` krabicový graf
    - `range` vymezuje rozpětí "fousků" grafu, pro kladné `range` se "fousky" rozpínají do vzdálenosti `range · IQR` (interkvartilová odchylka), pro `range=0` se rozpínají od minimální po maximální hodnotu, implicitní nastavení `range=1.5`
    - `notch` implicitní nastavení `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".<sup>1</sup>
  - `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)
```

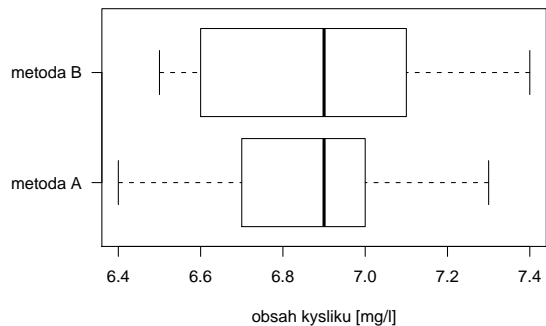
```
> load(file="kyslik.dat")     načtení datového souboru (porovnání dvou metod sta-
novení obsahu kyslíku ve vodě [mg/l], [8])
> boxplot(kyslik, names=c("metoda A", "metoda B"), horizontal=TRUE,
+ las=1, xlab="obsah kyslíku [mg/l]")
```

---

<sup>1</sup>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.



Obr. 7.8. Obsah dusičnanů ve studniční vodě [mg/l]



Obr. 7.9. Porovnání dvou metod stanovení obsahu kyslíku ve vodě [mg/l]

```
> (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)

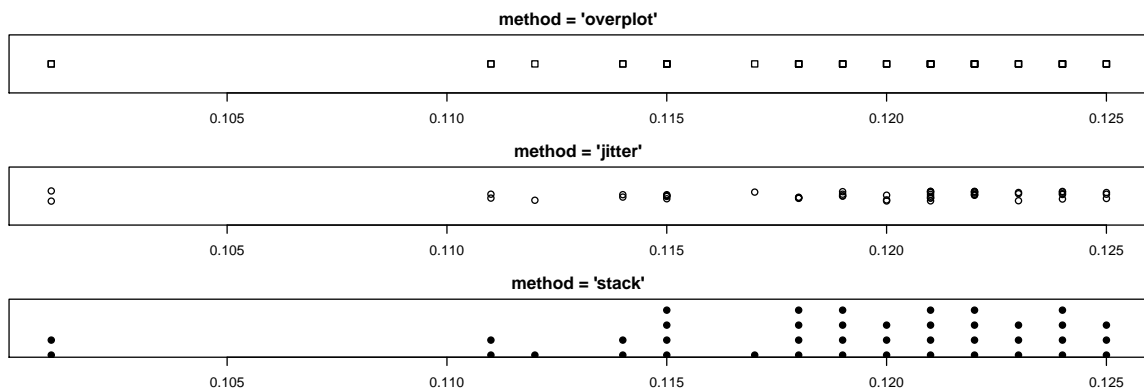
`at` numerický vektor udávající pozici pro vykreslení bodů

> `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 číslíc  
`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

## KAPITOLA 7. GRAFIKA V R

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))
```

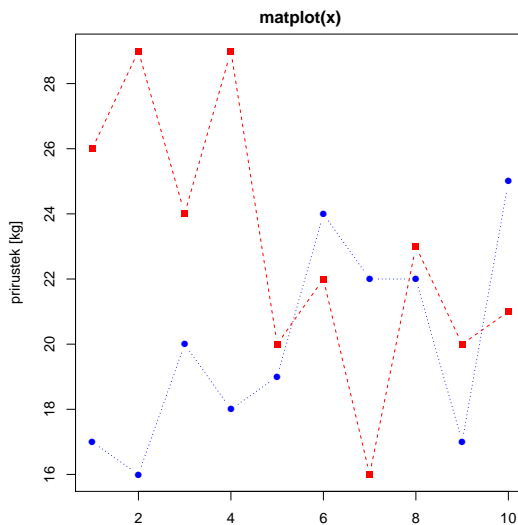
```
> legend(7, 29, c("smes 1", "smes 2"), lty=c(2,3), pch=c(15, 16),  
+ col=c(2,4)) funkce pro zobrazení legendy, více viz odstavec 7.2
```

```
> time <- seq(from=1962, to=1978, length=17)
```

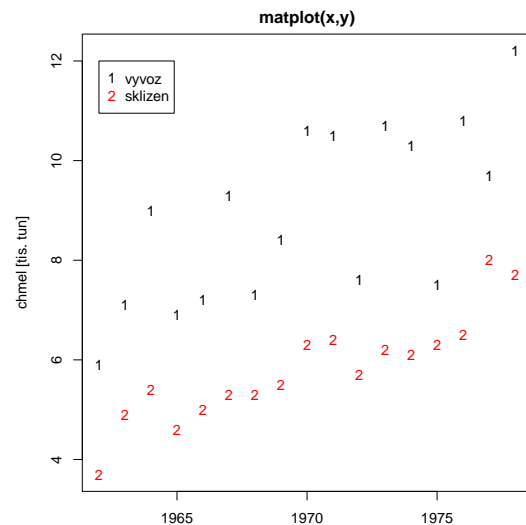
```
> load(file="chmel.dat") načtení datového souboru (vývoz a sklizeň chmele v ČSSR  
v letech 1962 až 1978, [8])
```

```
> 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

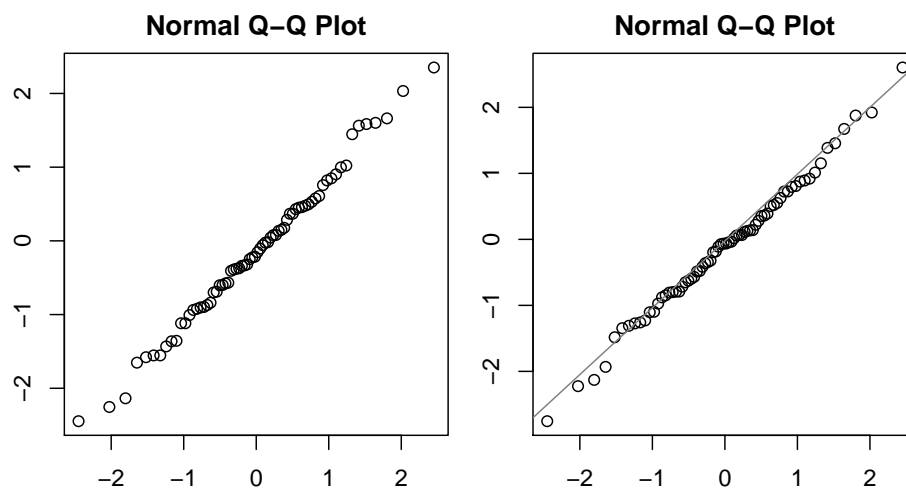
• `qqnorm()`, `qqline()` grafy pro ověřování normality dat. `qqnorm()` vykresluje normální kvantil-kvantilový graf (Q-Q plot), `qqline()` navíc přidá do již existujícího grafu přímku odpovídající normálnímu rozložení, jedná se o low-level funkci

```
> qqnorm(rnorm(70))
```

```
> qqnorm(rnorm(70))
```



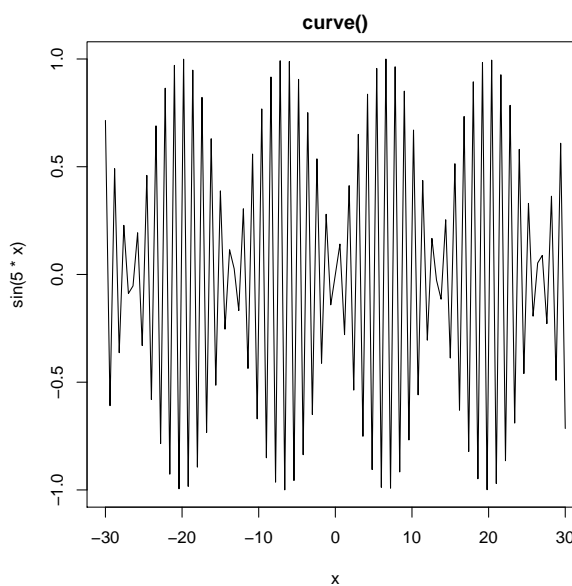
> `qqline(rnorm(70), col=gray(0.5))` funkce `gray` slouží k vykreslení různé stupně šedi (více v odstavci 7.3)



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()")`

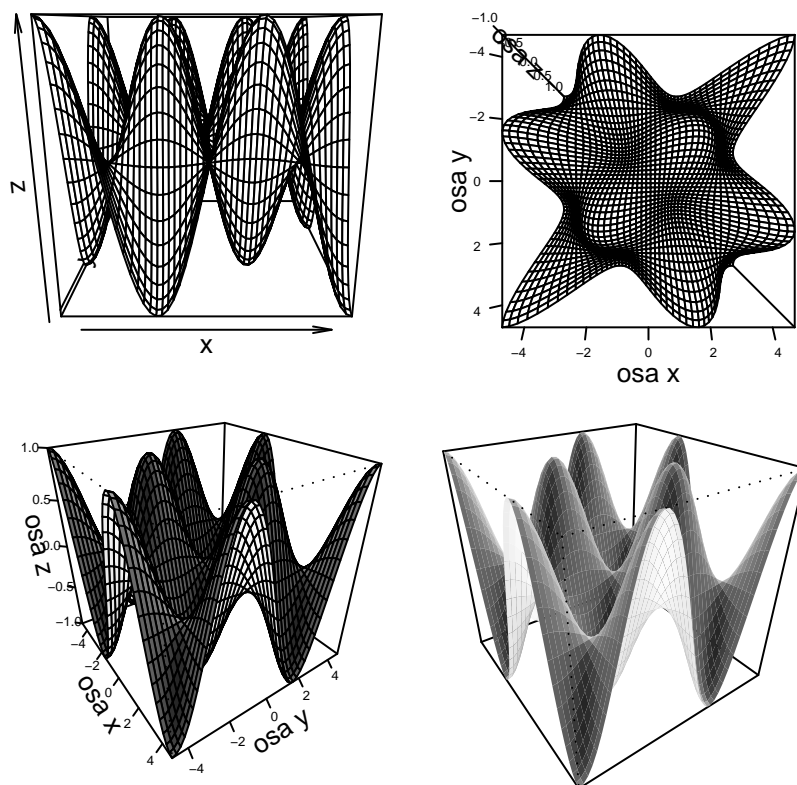


Obr. 7.14. Funkce `curve()`

## KAPITOLA 7. GRAFIKA V R

---

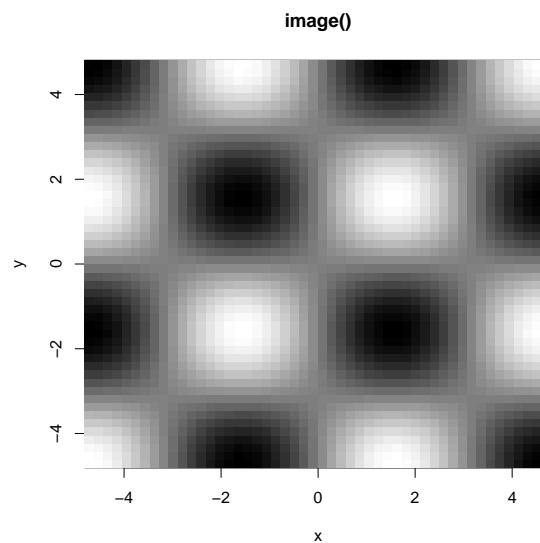
- `persp(x, y, z)` slouží k vykreslení 3D grafu
    - `xlim, ylim, zlim` intervaly pro vykreslování hodnot na osách  $x$ ,  $y$  a  $z$
    - `main, sub, col`
    - `theta` úhel pro otočení grafu v horizontálním smyslu
    - `phi` úhel pro otočení grafu ve vertikálním smyslu
    - `shade` stínování
    - `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

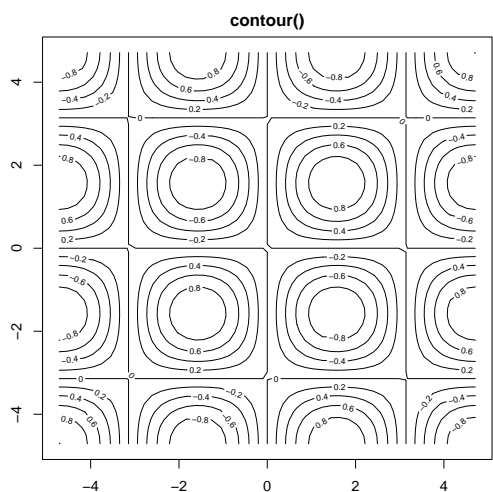
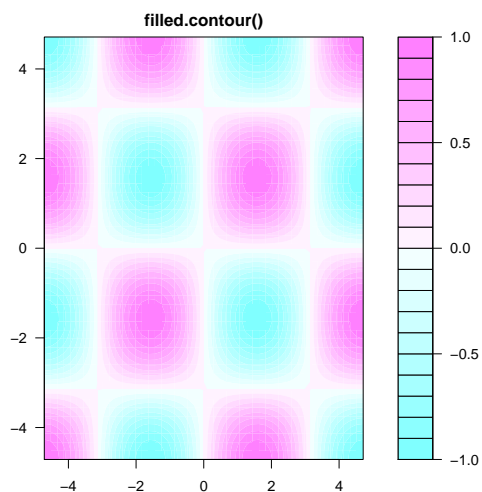
```
> x <- seq(-1.5*pi, 1.5*pi, length=50)
> y <- seq(-1.5*pi, 1.5*pi, length=50)
> z <- sin(x) %*% t(sin(y))
> image(x, y, z, col=gray(seq(from=0, to=1, length=80)), main="image()")
```



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 viz `help(symbols)`

Další tvary pro vykreslení: `rect()` (vykresluje obdélníkovou oblast), `polygon()` (mnohoúhelník), `arrows()` (šipky), `rug()` ("rohož" s rozestupem).

## 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 přímkou se směrnici `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  $x$
- `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 popisek 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  
**mfc** 2-prvkový vektor tvaru `c(pocet_radku, pocet_sloupcu)` rozděluje graf na odpovídající počet podgrafů, jednotlivé grafy jsou řazeny po sloupcích  
**mfrow** stejně jako `mfc`, 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

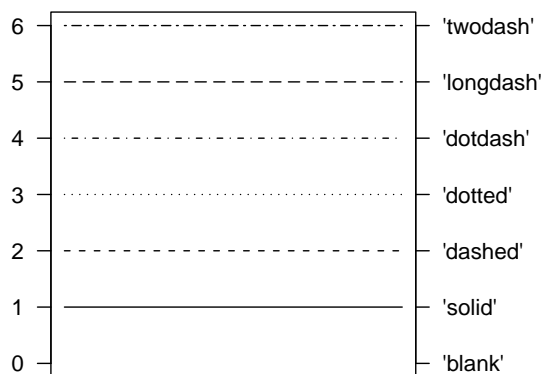
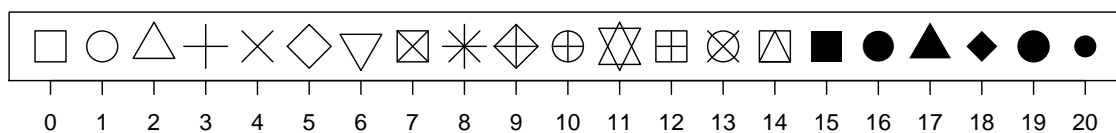
**las** numerická hodnota pro otočení popisků os:  
0 -  $x$  vodorovně,  $y$  svisle  
1 -  $x$  i  $y$  vodorovně  
2 -  $x$  svisle,  $y$  vodorovně  
3 -  $x$  i  $y$  svisle

**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.

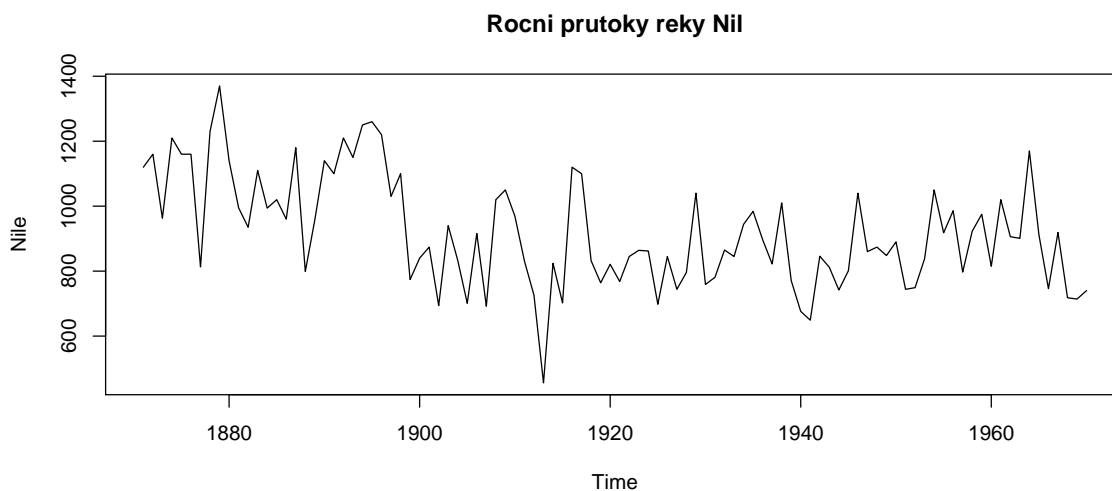
Ukládat grafy lze v menu *File* → *Save as* výběrem požadovaného formátu. Alternativou je použít některého z příkazů:

```
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 (vesta-
věná 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

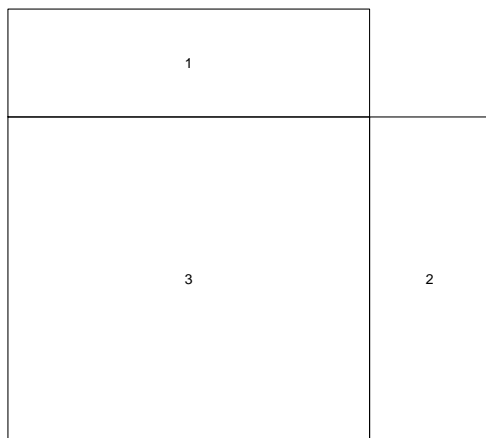


Funkce `layout()` umožňuje vytvářet složený graf obsahující několik grafů různých rozměrů, přičemž mohou zabírat i více řádků či sloupců.

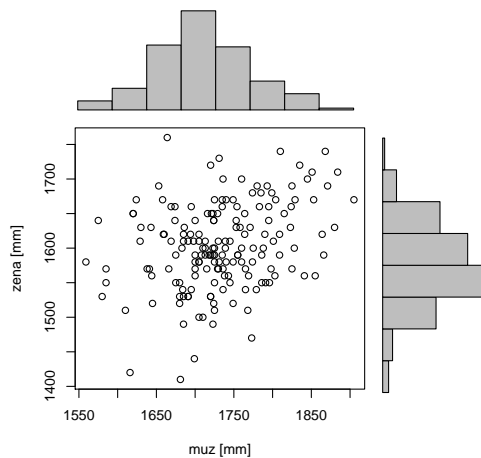
Jediným povinným argumentem funkce je matice, jejíž počet řádků a sloupců a jejich číselné hodnoty určují pořadí zaplňování jednotlivých "boxů" grafy. Příkaz `layout(matrix(c(1, 0, 3, 2), 2, 2, byrow=TRUE))` vytvoří graf se čtyřmi boxy, přičemž pozice `[1, 1]` bude obsazena jako první, dále budou obsazovány 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 manžel-
ských 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

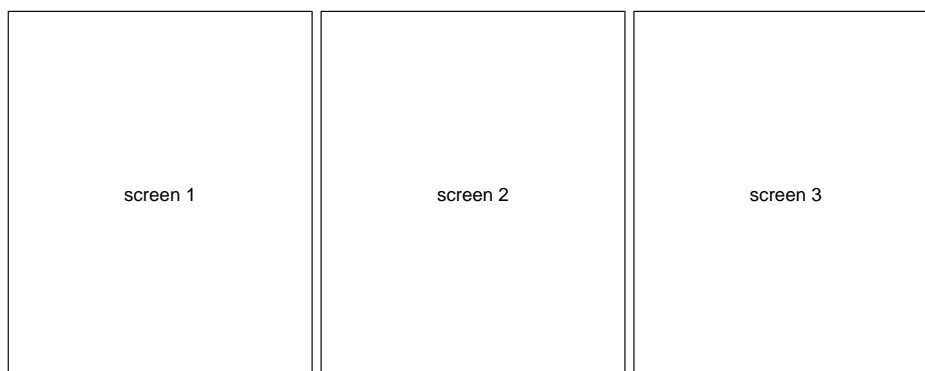
Funkce `split.screen()` slouží k dělení grafického prostředí na libovolný počet částí. Např. příkazem `split.screen(c(2,2))` rozdělíme prostředí na čtyři části, na něž můžeme odkazovat příkazy `screen(1), …, screen(4)` a pomocí těchto příkazů dané podgrafy vytvářet nebo měnit. Každá část přitom může být znovu rozdělena pomocí funkce `split.screen(c(,), screen)`.

*Poznámka.* Rozdělení grafu na jednotlivé podgrafy můžeme dosáhnout i pomocí argumentů `mfc` a `mfr` funkce `par()` (viz odstavec 7.3).

MATLAB používá k rozdělení grafu na více částí funkci `subplot`.

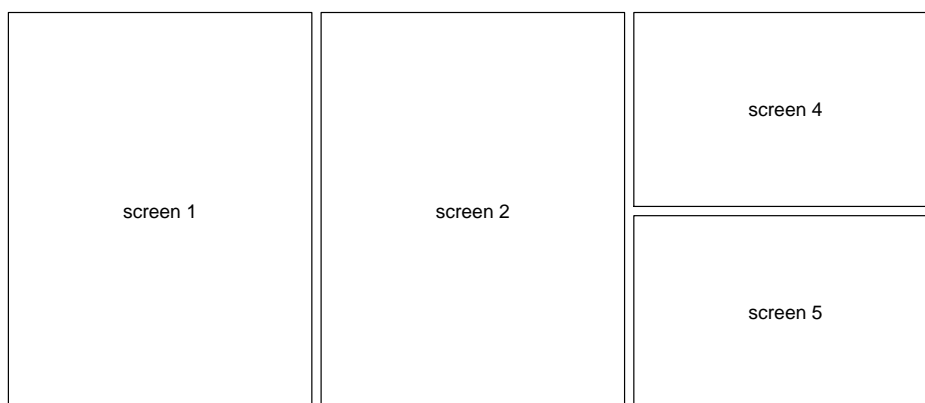


```
> load(file="etnika.dat")   načtení datového souboru (počet souhlasných reakcí
na deset otázek zástupcům čtyř etnik, [8])
> popisx <- rownames(etnika)
> split.screen(c(1,3))     vytvoří prázdné grafické okno, navíc je výstupem i vektor
hodnot odkazující na jednotlivé podgrafy, šablona okna s boxy viz Obr. 7.24
[1] 1 2 3
```



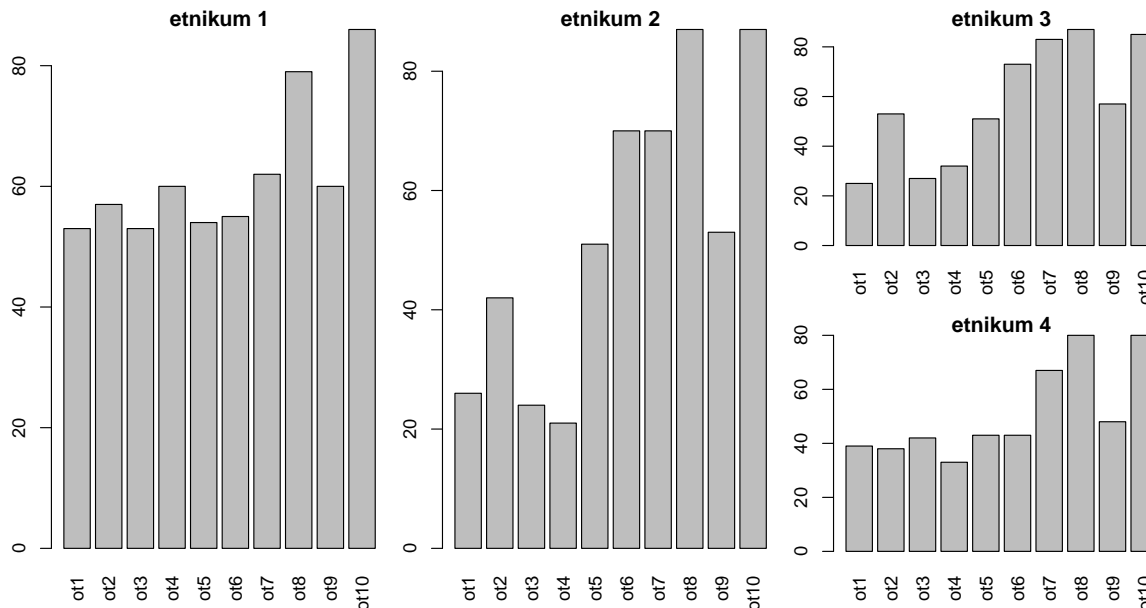
Obr. 7.24. Šablona okna s boxy pro příkaz `split.screen(c(1,3))`

```
> 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, popiseků 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 otitulujte 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(soradnice, 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, těmto 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 závkách { } (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 závkách. 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 závkách.

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 uzavřeném1 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
```

---

<sup>1</sup>To, že jde o uzavřené prostředí, je důležité. V našem příkladu jsou dvě prostředí: globální a prostředí funkce `funkce1()`. Kdyby zde byla tři a více vnořených prostředí, hledání objektu by probíhalo postupně od daného prostředí, přes jednotlivá uzavřená prostředí až ke globálnímu.

[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 potřebyjeme 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`.

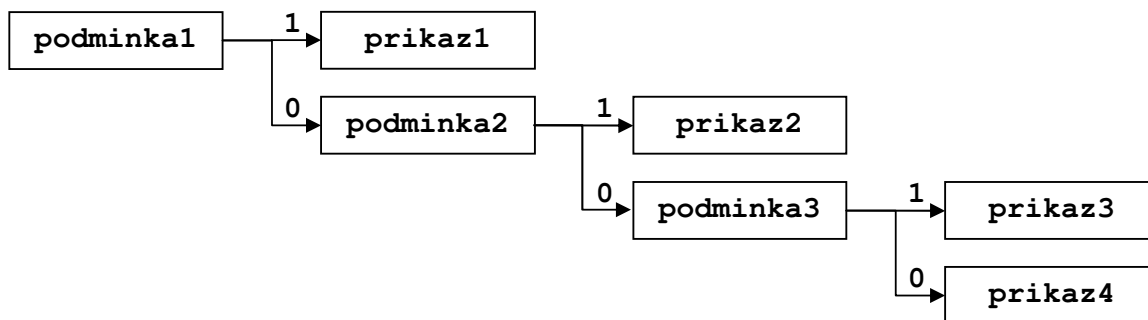
```
> if (x > 3) {"Podminka splnena."}
+ else {"Podminka nesplnena."}
[1] "Podminka splnena."
```

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) {"vetsti 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`.





Obr. 8.1. Diagram vysvětlující příkaz if - else if - else

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`.

```
> 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")}2   funkce cat() zřetězí a
vytiskne své argumenty
pruchod cyklem 1
pruchod cyklem 2
pruchod cyklem 3
pruchod cyklem 4
pruchod cyklem 5
```

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}
```

---

<sup>2</sup>výraz `"\n"` způsobí přechod na nový řádek

Stejně jako u výrazu `if`, podmínka je výraz, který po vyhodnocení vrací logickou hodnotu `TRUE` nebo `FALSE`. V případě splnění podmínky podmínka cyklus provede příkaz.

```
> 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 rozříděná data v tabulce `dat`. Argument `INDEX` specifikuje seznam položek pro rozřídění.

```
> (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("oddf.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 celociselné 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("Nejedná se o vektor kladných reálných čísel!")

  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 řešení 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](http://www.karlin.mff.cuni.cz/~kulich/vyuka/Rdoc/rdebuts_en.pdf)>

## SEZNAM POUŽITÉ LITERATURY

---

- [10] R Development 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](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](http://www.utm.edu/departments/cens/engineering/Images/matlab_logo_000.gif)>