Ján Buša a Ladislav Ševčovič

Open source systém na spracovanie údajov

Sadzba programom pdfTEX

Obsah

Úvod

Open source program **R** je komplexný systém na manipuláciu s údajmi, ich spracovanie, analýzu a následné grafické zobrazenie. **R** je implementáciou jazyka **S**, ktorý bol vyvinutý v Bellovych laboratóriách Rickom Beckerom, Johnom Chambersom a Allanom Willksom a ktorý slúži ako základ systému S-PLUS.

Základná distribúcia systému **R** je udržiavaná malou skupinou štatistikov známou pod menom **R** Development Core Team. Obrovské množstvo dodatočnej funkcionality je implementované formou balíkov, vytvorených veľkou skupinou dobrovoľníkov.

Cieľom tejto príručky je poskytnutie základných informácií o systéme **R**, práci s ním, ako aj o riešení viacerých praktických úloh s využitím funkcií programu **R**. Podrobnejšie informácie nájdete na oficiálnej domovskej stránke projektu **R** `http://www.R-project.org`, prípadne v literatúre uvedenej v zozname na konci príručky.

Košice 2007

1 Predbežné informácie

1.1 Prostredie R

R je integrovaná súprava softvérových nástrojov umožňujúca manipuláciu, numerické spracovanie a grafické zobrazenie dát. Okrem iného poskytuje:

- možnosti efektívne manipulovať s dátami a ukladať ich;
- súbor operátorov na prácu s poliami, napríklad s maticami;
- veľkú prepojenú kolekciu prostriedkov na analýzu dát;
- grafické možnosti na dátovú analýzu a zobrazenie výsledkov priamo na počítači alebo po vytlačení;
- kvalitne vyvinutý, jednoduchý a efektívny programovací jazyk (nazývaný „**S**“) obsahujúci rozhodovacie operátory, cykly, rekurzívne funkcie definované používateľmi, vstupné a výstupné procedúry. (V skutočnosti je väčšina funkcií napísaných v jazyku **S**.)

Označenie „prostredie“ má charakterizovať systém **R** skôr ako plne prepojený systém a nie ako postupne dopĺňanú zbierku špeciálnych nepružných nástrojov, ako sa to často stáva v prípade iných programov určených na analýzu dát.

R je výborný prostriedok na rozvoj nových metód interaktívnej analýzy údajov. Jeho vývoj bol veľmi dynamický a je rozšírený o veľké množstvo balíkov. Napriek tomu má väčšina programov, napísaných v systéme **R** na riešenie určitého problému analýzy údajov, nevyhnutne dočasný charakter.

1.2 R a štatistika

V prostredí **R** je možné riešiť aj mnoho klasických i moderných štatistických úloh. Niektoré sú priamo súčasťou základného prostredia **R**, viaceré sú k dispozícii ako balíky. Spolu s programom **R** je dodaných okolo 25 balíkov (nazývaných „štandardné“ alebo „odporúčané“ balíky), mnohé ďalšie sa dajú získať z archívu CRAN (napr. z `http://CRAN.R-project.org` alebo inde).

Je preto potrebné, aby bol používateľ pripravený na drobnú námahu pri vyhľadávaní niektorých štatistických procedúr, ak nie sú priamo v základnom prostredí **R**.

Existuje jeden dôležitý rozdiel medzi filozofiou **S** (a teda aj **R**) a filozofiou iných známych štatistických systémov. Štatistická analýza sa v systéme **S** obyčajne vykonáva ako postupnosť krokov, pri ktorej sa medzivýsledky ukladajú do objektov. Zatiaľ čo systémy SAS a SPSS vydajú rozsiahly výstup ako výsledok regresnej alebo diskriminantnej analýzy, výstup programu **R** je minimálny a priebežné výsledky sú uložené do objektov na prípadné využitie ďalšími funkciami **R**.

1.3 Interaktívne použitie programu R

Pri použití programu **R** sa zobrazí výzva (anglicky prompt), keď **R** očakáva na vstupe príkazy. Prednastavená výzva má tvar ‚>', môžeme ju zmeniť príkazom `options`, napríklad na ‚R>␣' príkazom `options(prompt="R> ")`.

V OS Linux (ak predpokladáme „prompt" ‚$') môžete začať pracovať nasledujúcim spôsobom:

1. Vytvorte zvláštny priečinok, napríklad `work`, v ktorom budete ukladať súbory, ktore použijete na riešenie úlohy programom **R**. Tento bude pracovným adresárom pre túto úlohu, môžete sa do neho dostať príkazom `cd` – *change directory*.

   ```
   $ mkdir work
   $ cd work
   ```

2. Spustťe program **R** príkazom

   ```
   $ R
   ```

3. Teraz už môžete zadávať príkazy programu **R**. Príkaz na ukončenie činnosti programu **R** je

   ```
   > q()
   ```

   V tomto momente sa program **R** opýta, či si prajete uložiť údaje. Uložené údaje budú potom k dispozícii pri ďalšom spustení programu **R**.

4. Nasledujúce použitie programu je jednoduché. Nastavte `work` ako pracovný priečinok a spustťe program **R**:

   ```
   $ cd work
   $ R
   ```

   Po ukončení práce uzavrite program **R** príkazom `q()`.

Pri použití programu **R** pod OS Windows je postup principiálne rovnaký. Po vytvorení pracovného priečinka môžete tento nastaviť ako aktuálny pri spustení v položke Štart odkazu na program **R** a program spustiť dvojklikom na ikonku. Adresár môžete zmeniť aj prostredníctvom položky File/Change dir ... po spustení programu **R**.

2 Jednoduché manipulácie s číslami a s vektormi

2.1 Vektory a priradenie

R pracuje s pomenovanými dátovými štruktúrami. Najjednoduchšou štruktúrou je číselný vektor, ktorý predstavuje usporiadanú postupnosť čísel. Napríklad, na zadanie vektora s názvom `x`, pozostávajúceho z piatich čísel, konkrétne 10.4, 5.6, 3.1, 6.4 a 21.7, použite príkaz

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

Toto je operátor priradenia pomocou funkcie `c()`, ktorá v tomto prípade môže obsahovať ľubovoľný počet *vektorových* argumentov. Jej hodnotou je vektor, ktorý vznikne spojením jej argumentov od začiatku do konca.[1]

Číslo, ktoré sa vyskytuje vo výraze sa chápe ako vektor dĺžky 1.

Poznamenajme, že operátor priradenia (‘`<-`’), pozostáva z dvoch znakov ‘`<`’ („menší ako“) a ‘`-`’ („mínus“) nasledujúcich bezprostredne za sebou a „ukazuje“ na objekt, ktorému sa priraďuje hodnota výrazu. Vo väčšine prípadov je možné alternatívne použiť operátor ‘`=`’. Priradenie sa môže vykonať tiež použitím funkcie `assign()`. Rovnaký výsledok ako vyššie dosiahneme zadaním:

```
> assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))
```

Bežný operátor `<-` je možné chápať ako syntaktickú skratku príkazu `assign()`.

Priradenie je možné vykonať aj v opačnom smere použitím zrejmej zmeny v operátore priradenia, teda zadaním

```
> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x
```

Ak zadáme len výraz (bez priradenia), hodnota sa vytlačí a stratí[2]. Teda po použití príkazu

```
> 1/x
```

bude na obrazovke vytlačených 5 prevrátených hodnôt jednotlivých zložiek vektora `x` (hodnota `x` sa, prirodzene, nezmení).

Ďalším priradením

```
> y <- c(x, 0, x)
```

bude vytvorený vektor `y` s 11-imi zložkami, pozostávajúci z dvoch kópií vektora `x`, medzi ktorými je uprostred umiestnená 0.

[1] Ak použijeme iný typ argumentov ako vektorový, napríklad argumenty typu zoznam, výsledok použitia funkcie `c()` bude úplne iný.

[2] V skutočnosti, pred použitím ďalšieho príkazu, je ešte stále k dispozícii ako hodnota `.Last.value`.

2.2 Vektorová aritmetika

Vektory sa dajú používať v aritmetických výrazoch. V tom prípade sa operácie vykonávajú po zložkách. Vektory použité v určitom výraze nemusia mať rovnakú dĺžku. Ak sa vo výraze vyskytnú vektory rôznej dĺžky, výsledok bude mať rovnakú dĺžku ako *najdlhší* vektor. Zložky kratších vektorov sa *cyklicky* dopĺňajú tak, aby tieto vektory dosiahli maximálnu dĺžku. Ak sa vo výraze nachádza konštanta, jednoducho sa opakuje. Teda po predchádzajúcich priradeniach príkazom

```
> v <- 2*x + y + 1
```

vytvoríme nový vektor `v` dĺžky 11, ktorého zložky sa vytvoria sčítaním odpovedajúcich zložiek vektora `2*x` opakovaného 2.2-krát[3], vektora `y` a čísla 1 opakovaného 11-krát.

Toto je dosť neobvyklý spôsob. Skúste si premyslieť, ako to funguje, napríklad po zadaní nasledujúcich príkazov:

```
> u=c(1,2,3); w=c(u,0,0,u); 2*u+w
[1] 3 6 9 2 4 7 4 7
Warning message:
   longer object length
   is not a multiple of shorter object length in: 2 * u + w
```

V OS Windows sa objavilo aj upozornenie, že dĺžka vektora `w` – 8 – nie je celým násobkom dĺžky vektora `u` – 3.

Základné aritmtické operátory sú bežne používané `+`, `-`, `*`, `/` a `^` sa používa na umocňovanie. Navyše sú k dispozícii všetky elementárne aritmetické funkcie: `log`, `exp`, `sin`, `cos`, `tan`, `sqrt` atď. majú obvyklý význam. Funkcie `max` a `min` vyberú najväčšiu resp. najmenšiu zložku vektora. Funkcia `range` vráti vektor dĺžky 2, konkrétne `c(min(x), max(x))`. Funkcia `length(x)` určí počet zložiek vektora `x`, `sum(x)` vráti súčet zložiek vektora `x`, `prod(x)` vypočíta ich súčin. K dispozícii sú aj dve štatistické funkcie – `mean(x)` vypočíta výberový priemer, ktorý sa rovná `sum(x)/length(x)` a `var(x)` dáva `sum((x-mean(x))^2)/(length(x)-1)`, čiže nevychýlený výberový rozptyl. Ak je argumentom funkcie `var()` matica typu $n \times p$, jej hodnotou bude výberová kovariančná matica typu $p \times p$, pričom riadky vstupnej matice sa chápu ako nezávislé výberové vektory rozmeru p.

Funkcia `sort(x)` vráti vektor rovnakej dĺžky ako vektor `x` so zložkami usporiadanými v narastajúcom poradí; avšak existujú aj flexibilnejšie prostriedky na usporadúvanie (napríklad `order()` alebo `sort.list()`). Všimnime si, že `max` a `min` vyberú najväčšiu a najmenšiu hodnotu svojich argumentov dokonca aj vtedy, ak je zadaných niekoľko vektorov. Paralelné funkcie maximum a minimum – `pmax` a `pmin` – vrátia vektor (jeho dĺžka je

[3] $5 \times 2.2 = 11$.

rovnaká, ako najväčšia dĺžka vstupného argumentu) obsahujúci na každej pozícii najväčší resp. najmenší prvok na odpovedajúcej pozícii vo všetkých vstupných vektoroch. Vo väčšine prípadov sa používateľ nemusí starať o to, či sú „čísla“ celé, reálne alebo dokonca komplexné. Vnútorne sa operácie vykonávajú v dvojnásobnej presnosti reálnych čísel, alebo v dvojnásobnej presnosti komplexných čísel, ak sú vstupné údaje komplexné.

Na prácu s komplexnými číslami zadajte *explicitne* imaginárnu zložku. Napríklad

```
sqrt(-17)
```

vráti hodnotu `NaN` a varovanie, ale

```
sqrt(-17+0i)
```

uskutoční výpočet v množine komplexných čísel.

2.3 Generovanie rovnomerných postupností

R má niekoľko prostriedkov na generovanie často používaných číselných postupností. Napríklad `1:30` je vektor `c(1, 2, ..., 29, 30)`. Operátor ,:' (dvojbodka) má vo výrazoch najvyššiu prioritu, teda, napríklad `2*1:15` je vektor `c(2, 4, ..., 28, 30)`. Zadajte `n <- 10` a porovnajte postupnosti `1:n-1` a `1:(n-1)`.

Konštrukcia `30:1` sa dá použiť na vygenerovanie klesajúcej postupnosti.

Funkcia `seq()` je všeobecnejší prostriedok na vytváranie postupností. Má 5 argumentov, pričom pri jednotlivom použití môžu byť použité len niektoré z nich. Prvé dva argumenty, ak sú zadané, určujú začiatok a koniec postupnosti a v prípade, že sú zadané len tieto dva argumenty, výsledok bude rovnaký ako pri použití operátora `:`. Teda `seq(2,10)` je taký istý vektor ako `2:10`.

Parametre funkcie `seq()` a mnohých ďalších funkcií **R**, môžu byť zadané aj v tvare priradenia hodnôt vnútorným premenným, kedy nezáleží na poradí ich zadania. Prvé dva parametre môžu byť označené `from=value` a `to=value`; teda `seq(1,30)`, `seq(from=1, to=30)` a `seq(to=30, from=1)` definujú rovnaké vektory ako `1:30`. Nasledujúce dva parametre príkazu `seq()` môžu byť označené `by=value` resp. `length=value`, čím určíme veľkosť kroku resp. celkovú dĺžku postupnosti. Ak nie je daná žiadna z týchto hodnôt, predpokladá sa implicitne nastavená hodnota `by=1`.

Napríklad

```
> seq(-5, 5, by=.2) -> s3
```

priradí premennej `s3` vektor `c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)`. Podobne

```
> s4 <- seq(length=51, from=-5, by=.2)
```

priradí rovnaký vektor premennej `s4`.

Piaty parameter sa definuje zadaním `along=vector`, ktorý môže byť použitý len osamotene, pričom sa vytvorí postupnosť $1, 2, \ldots,$ `length(vector)` alebo prázdna postupnosť, ak je daný vektor prázdny.

Na zopakovanie zložiek nejakého objektu je možné rôznymi komplikovanými spôsobmi použiť funkciu `rep()`. Najjednoduchší tvar je

```
> s5 <- rep(x, times=5)
```

ktorý priradí 5 kópií vektora `x` od začiatku do konca premennej `s5`. Ďalšia užitočná verzia

```
> s6 <- rep(x, each=5)
```

zopakuje každú zložku vektora `x` 5-krát postupne pre všetky zložky.

2.4 Logické vektory

R umožňuje prácu s logickými hodnotami rovnako ako s aritmetickými. Zložky logického vektora môžu nadobúdať hodnoty `TRUE` – pravda, `FALSE` – lož a `NA` (z anglického „not available“ – neznáma, viď nižšie). Prvé dve hodnoty sa často skracujú na `T` resp. `F`. Uvedomte si však, že `T` a `F` sú len premenné, ktoré sú implicitne nastavené na `TRUE` a `FALSE`, ale nie sú kľúčovými slovami, a teda môžu byť prepísané používateľom. Preto je lepšie používať vždy `TRUE` a `FALSE`. Logické vektory sú vytvárané logickými podmienkami. Napríklad

```
> temp <- x > 13
```

určí vektor `temp` rovnakej dĺžky ako vektor `x` s hodnotami `FALSE` odpovedajúcim zložkám vektora `x`, ktoré nespĺňajú podmienku (prvé štyri) a `TRUE` v pozíciách položiek, spĺňajúcich podmienku (posledná zložka).

Logické operátory sú `<`, `<=`, `>`, `>=`, `==` pre presnú rovnosť a `!=` pre nerovnosť. Navyše, ak sú `c1` a `c2` logické výrazy, tak `c1 & c2` je ich prienik (konjunkcia, logický súčin „and“), `c1 | c2` je ich zjednotenie (disjunkcia, logický súčet „or“) a `!c1` je negácia výrazu `c1`. Logické vektory môžu byť použité aj v obyčajnej aritmetike, v tom prípade sú transformované na číselné vektory – z `FALSE` sa stane 0 a z `TRUE` vznikne 1. Avšak existujú situácie, keď logické vektory a ich číselné analógy nie sú ekvivalentné, viď, napríklad, nasledujúci oddiel.

2.5 Chýbajúce hodnoty

V niektorých prípadoch nemusia byť známe všetky zložky vektora. Ak je zložka alebo hodnota „neznáma“ alebo „chýbajúca“ hodnota v štatistickom zmysle, môže byť jej miesto vnútri vektora rezervované priradením zvláštnej hodnoty `NA`. Vo všeobecnosti je výsledkom každej operácie obsahujúcej hodnotu `NA` tiež hodnota `NA`. Dôvodom na použitie tohto pravidla je jednoducho fakt, že ak nie operácia úplne definovaná, výsledok tiež nemôže byť známy a teda nie je k dispozícii.

Funkcia `is.na(x)` vráti logický vektor rovnakého rozmeru ako vektor `x` s hodnotou `TRUE` práve na tých pozíciách, ktorým odpovedajúce zložky vektora `x` sú `NA`.

```
> z <- c(1:3,NA);  ind <- is.na(z)
```

Poznamenajme, že logický výraz `x == NA` sa líši od `is.na(x)`, pretože `NA` nie je v skutočnosti hodnota ale len značka hodnoty, ktorá nie je k dispozícii. Teda `x == NA` je vektor rovnakrej dĺžky ako `x`, ktorého všetky hodnoty sú `NA`, keďže samotný logický výraz je neúplný a teda neurčitý.

Všimnime si, že existuje ešte druhý druh „chýbajúcich“ (anglicky missing) hodnôt, ktoré sú výsledkom číselných výpočtov. Sú to tzv. `NaN` hodnoty (z anglického Not a Number). Napríklad

```
> 0/0
```

alebo

```
> Inf - Inf
```

vrátia výsledok `NaN`, keďže výsledok nemôže byť zmysluplne definovaný.

Na záver, `is.na(xx)` je `TRUE` v obidvoch prípadoch pre hodnoty `NA` aj `NaN`. Na rozlíšenie týchto prípadov je `is.nan(xx)` hodnota `TRUE` len pre hodnotu `NaN`. Chýbajúce hodnoty sú niekedy vytlačené ako `<NA>`, keď sa vektory znakov (reťazce) tlačia bez úvodzoviek.

2.6 Znakové vektory

Znakové hodnoty alebo znakové vektory sa často používajú v programe **R**, napríklad na popis grafov. Ak ich potrebujeme, vytvoríme ich ako postupnosť znakov ohraničenú úvodzovkami, napr. `"hodnoty x"`, `"Novy vektor"`.

Reťazce sa zadávajú použitím úvodzoviek (`"`) alebo apostrofov (`'`), ale tlačia sa použitím úvodzoviek (alebo v niektorých prípadoch bez úvodzoviek). Používajú únikové postupnosti v štýle jazyka C, kde `\` je únikový symbol, teda `\\` sa zadáva a tlačí ako `\` a vo vnútrii úvodzoviek sa `"` zadáva ako `\"`. Ďalšími užitočnými únikové postupnosti sú `\n` – newline, `\t` – tabulátor a `\b` – „backspace“.

Znakové vektory môžu byť spojené do jedného vektora funkciou `c()`. Funkcia `paste()` použije ľubovoľný počet argumentov a spojí ich po jednom do znakových reťazcov. Všetky čísla medzi argumentmi sa prirodzene transformujú na znakové reťazce, teda rovnakým spôsobom, ako by to bolo, keby boli tlačené. Argumenty sú implicitne oddeľované medzerou, čo však môžeme zmeniť nastavením vnútornej premennej, `sep=string`, ktorý zmení medzeru na reťazec `string`, ktorý može byť aj prázdny.

Napríklad

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

vytvorí `labs` ako znakový vektor

```
c("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")
```

Všimnime si, že aj v tomto prípade sa používa cyklické opakovanie zložiek kratších vektorov; teda `c("X", "Y")` sa opakuje 5-krát aby sa dosiahla dĺžka postupnosti `1:10`.[4]

2.7 Indexové vektory; voľba a úprava podmnožín dátovej množiny

Podmnožiny zložiek vektora môžu byť určené pridaním vektora indexov v hranatých zátvorkách ku názvu vektora. Všeobecnejšie povedané, každý výraz, ktorého hodnota je vektor, môže mať podmnožiny zložiek vybrané pridaním vektora indexov v hranatých zátvorkách bezprostredne za výrazom.

Takéto vektory indexov môžu byť niektorého z nasledujúcich štyroch typov:

1. Logický vektor. V tomto prípade musí mať indexový vektor rovnakú dĺžku ako vektor, ktorého zložky vyberáme. Zvolené sú tie zložky vektora, ktoré odpovedajú hodnote `TRUE` indexového vektora, zložky odpovedajúce hodnote `FALSE` sú vynechané. Napríklad

   ```
   > y <- x[!is.na(x)]
   ```

   vytvorí (alebo prepíše) objekt `y`, ktorý bude obsahovať len definované hodnoty vektora `x`, v pôvodnom poradí. Všimnime si, že ak má vektor `x` chýbajúce alebo nedefinované zložky, bude `y` kratší ako `x`. Podobne

   ```
   > (x+1)[(!is.na(x)) & x>0] -> z
   ```

   vytvorí objekt `z` a umiestni do neho hodnoty vektora `x+1` odpovedajúce definovaným a kladným zložkám vektora `x`.

2. Vektor kladných celočíselných hodnôt. V tomto prípade musia hodnoty indexov patriť do množiny $\{1, 2, \ldots, \texttt{length(x)}\}$. Odpovedajúce zložky vektora sú vybrané a spojené do nového vektora v tomto poradí. Indexový vektor môže mať ľubovoľnú dĺžku a výsledok bude mať rovnakú dĺžku ako vektor indexov. Napríklad `x[6]` je 6. zložka vektora `x` a

   ```
   > x[1:10]
   ```

   vyberie prvých 10 prvkov vektora `x` (tu predpokladáme, že dĺžka `length(x)` je aspoň 10). Podobne

[4] `paste(..., sep=ss)` spojí dva argumeny do jedného znakového reťazca s vložením reťazca `ss` mezdi ne. Na manipuláciu so znakovými reťazcami slúžia ešte ďalšie nástroje, viď pomoc pre `sub` alebo `substring`.

```
> c("x","y")[rep(c(1,2,2,1), times=4)]
```

(možno málo pravdepodobné) vytvorí znakový vektor dĺžky 16 pozostávjúci zo štvoríc znakov `"x"`, `"y"`, `"y"`, `"x"` zopakovaných 4-krát.

3. Vektor záporných celočíselných hodnôt. Takýto indexový vektor definuje hodnoty, ktoré majú byť vylúčené. Teda

```
> y <- x[-(1:5)]
```

priradí do vektora `y` všetky zložky vektora `x` okrem prvých piatich.

4. Vektor znakových reťazcov. Táto možnosť sa dá použiť len v prípade, ak má objekt definované menné atribúty na identifikáciu svojich zložiek. V tom prípade môže byť indexový vektor pozostávajúci z mien jednotlivých zložiek rovnakým spôsobom ako v prípade kladných indexov v bode 2 vyššie.

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> fruit[c("apple","orange")]
 apple orange
     1      5
```

Výhodou je to, že mená sa často ľahšie pamätajú, ako číselné indexy.

Indexovaný výraz sa môže tiež objaviť na strane priraďovanej premennej. V tomto prípade sa operátor vykoná len pre tie zložky vektora, ktoré určuje indexový vektor. Výraz musí mať tvar `vector[index_vector]`, keďže na tomto mieste nemá veľký význam mať namiesto vektora ľubovoľný výraz.

Vektor, ktorý priraďujeme, musí mať rovnakú dĺžku ako indexový vektor a v prípade logického indexového vektora musí mať znova rovnakú dĺžku ako vektor, ktorý indexuje.

Napríklad

```
> x[is.na(x)] <- 0
```

nahradí všetky chýbajúce resp. neurčité hodnoty vektora `x` nulami a príkaz

```
> y[y < 0] <- -y[y < 0]
```

má rovnaký účinok ako

```
> y <- abs(y)
```

2.8 Ďalšie typy objektov

Vektory sú najdôležitejší typ objektov systému **R**, ale existujú viaceré ďalšie, s ktorými sa stretneme neskôr:

- Matrice alebo všeobecnejšie polia sú viacrozmerné zovšeobecnenia vektorov. V skutočnosti sú to vektory indexované pomocou dvoch alebo viacerých indexov a môžu byť vytlačené zvláštnym spôsobom.
- Faktory (anglicky factors) poskytujú zjednodušený spôsob na prácu s kategorickými dátami.
- Zoznamy (anglicky lists) sú všeobecným tvarom vektorov, v ktorých jednotlivé zložky nemusia byť rovnakého typu a ktoré sú často samy vektory alebo zoznamy. Zoznamy poskytujú príjemný spôsob vrátenia výsledkv štatistických výpočtov.
- Dátové rámce sú štruktúry podobné maticiam. Ich stĺpce môžu byť rôznych typov. Môžete si predstaviť dátový rámček ako „dátovu maticu“, s jedným riadkom na pozorovanú jednotku s (možnými) číselnými alebo kategorickými premennými. Mnohé experimenty sú najlepšie popísané dátovými rámčekmi: postupy sú kategorické ale odozva je číselná.
- Funkcie samotné sú tiež objekty **R**, ktoré môžu byť v pracovnom priestore projektu. Toto poskytuje jednoduchý a pohodlný spôsob rozšírenia systémy **R**.

3 Načítanie údajov zo súborov

Veľké dátové objekty je lepšie zadávať ako hodnoty z externého súboru ako zadávať ich počas práce **R** z klávesnice. Vstupné nástroje **R** sú jednoduché a ich požiadavky sú dosť presné a viacmenej nepružné. Existuje jasný predpoklad tvorcov systému **R**, že budete schopní upravovať svoje vstupné súbory pomocou iných prostriedkov, ako sú napríklad textové editory alebo Perl[5], aby ste ich pripravili podľa požiadaviek **R**. Vo všeobecnosti je to veľmi jednoduché.

Ak majú byť premenné uložené najmä do dátových rámčekov, čo veľmi odporúčame, celá dátova tabuľka (rámček) môže byť načítaná priamo funkciou `read.table()`. Existuje aj jednoduchšia vstupná funkcia `scan()`, ktorá môže byť volaná priamo.

Podrobnejšie informácio o importovaní a exportovaní údajov nájdete v príručke **R** Data Import/Export.

3.1 Funkcia `read.table()`

Externý súbor, z ktorého sa priamo načítava tabuľka údajov, má obyčajne špecialny tvar:

- Prvý riadok súboru musí obsahovať názvy všetkých premenných tabuľky.
- Každý ďalší riadok obsahuje ako prvú položku riadkové návestie (anglicky label) a hodnoty všetkých premenných.

Ak má súbor v prvom riadku o jednu položku menej ako v druhom, predpokladá sa, že platia vyššie uvedené predpoklady. Prvých niekoľko riadkov súboru môže teda vyzerať nasledovne:

Vstupný súbor s názvami a označením riadkov:

```
     Price  Floor  Area  Rooms  Age  Cent.heat
01   52.00  111.0   830     5   6.2      no
02   54.75  128.0   710     5   7.5      no
03   57.50  101.0  1000     5   4.2      no
04   57.50  131.0   690     6   8.8      no
05   59.75   93.0   900     5   1.9     yes
...
```

Implicitne sa číselné položky (s výnimkou označení riadkov) načítavajú ako číselné premenné a nečíselné premenné, ako napr. `Cent.heat` v príklade, sa načítavajú ako faktory. V prípade potreby sa to dá zmeniť.

Funkcia `read.table()` sa dá použiť na priame načítanie dátovej tabuľky nasledovne:

[5] Pod OS Linux môžete použiť služby Sed alebo Awk.

```
> HousePrice <- read.table("houses.data")
```

Často chceme vynechať zadanie riadkových návestí a použiť implicitné označenia. V tom prípade je možné vynechať stĺpec riadkových značiek, ako, napríklad, v nasledujúcom príklade:

Vstupný súbor bez označenia riadkov:

```
Price  Floor  Area  Rooms  Age  Cent.heat
52.00  111.0   830     5    6.2     no
54.75  128.0   710     5    7.5     no
57.50  101.0  1000     5    4.2     no
57.50  131.0   690     6    8.8     no
59.75   93.0   900     5    1.9    yes
 ...
```

V tom prípade na načítanie tabuľky údajov použijeme príkaz

```
> HousePrice <- read.table("houses.data", header=TRUE)
```

kde voľba `header=TRUE` určuje, že prvý riadok je riadok hlavičiek (názvov), a teda z tvaru súboru implicitne vyplýva, že nie sú zadané explicitné riadkové návestia.

3.2 Funkcia `scan()`

Predpokladajme teraz, že dátové vektory majú rovnakú dĺžku a majú byť načítané súčasne. Predpokladajme navyše, že máme tri vektory, prvý má formát reťazca a ostatné dva sú číselné. Všetky tri vektory tvoria tri stĺpce súboru `input.dat`. Prvým krokom je použitie príkazu `scan()` na načítanie troch vektorov ako zoznam (anglicky list) nasledovne:

```
> inp <- scan("input.dat", list("",0,0))
```

Druhým argumentom je fiktívny zoznam (a dummy list structure), ktorý určuje formát troch vektorov, ktoré sa majú načítať. Výsledok, uložený v premennej `inp`, je zoznam, ktorého zložkami sú tri načítané vektory. Na oddelenie jednotlivých položiek do troch samostatných vektorov použijeme nasledovné priradenia:

```
> label <- inp[[1]]; x <- inp[[2]]; y <- inp[[3]]
```

Pohodlnejšie je zadať fiktívny zoznam v mennom tvare (s označením názvov stĺpcov). V tom prípade sa názvy dajú použiť na prístup k načítaným vektorom. Napríklad

```
> inp <- scan("input.dat", list(id="", x=0, y=0))
```

Ak chcete ku premenným pristupovať oddelene, môžu byť buď priradené premenným v pracovnom prostredí:

```
> label <- inp$id; x <- inp$x; y <- inp$y
```

alebo zoznam môže byť pridaný na 2. miesto vyhľadávacej cesty.

Ak je druhý argument jednoduchá hodnota a nie zoznam, načíta sa jediný vektor, ktorého všetky zložky musia mať rovnaký formát ako daná fiktívna hodnota:

```
> X <- matrix(scan("light.dat", 0), ncol=5, byrow=TRUE)
```

Existujú aj prepracovanejšie prostriedky na načítanie údajov, sú podrobnejšie popísané v rôznych príručkách.

3.3 Prístup k zabudovaným dátovým množinám

Okolo 100 dátových súborov je dodaných s programom **R** (v balíku `datasets`) a ďalšie sú dostupné v balíkoch (vrátane odporúčaných balíkov dodaných spoluso systémom **R**). Ak chcete vidieť zoznam aktuálne dostupných údajových súborov, zadajte príkaz

```
data()
```

Od verzie **R** 2.0.0 sú všetky dátové súbory dodané spoločne s programom **R** dostupné priamo zadaním názvu (mena). Avšak, viaceré balíky stále používajú predchádzajúce konvencie, podľa ktorých sa tiež načítavajú dáta do **R**, napríklad

```
data(infert)
```

a tieto môžu byť ešte aj teraz použité so štandarnými balíkmi (ako v tomto príklade). Vo väčšine prípadov načíta systém **R** objekt s rovnakým názvom. Avšak, zriedkavo sa načíta niekoľko objektov, preto si pozrite „online help“ pre objekt, aby ste videli, čo môžete očakávať.

3.3.1 Načítanie údajov z iných balíkov R

Na prístup k údajom určitého balíka použite argument `package`, napríklad

```
data(package="rpart")
data(Puromycin, package="datasets")
```

Ak bol nejaký balík pripojený s knižnicou, jeho dátové súbory sú automaticky pridané do položky na vyhľadávanie.

Balíky vytvorené a poskytnuté používateľmi môžu byť bohatým zdrojom dátových súborov.

3.4 Editácia údajov

Po vyvolaní dátovej tabuľky alebo matice, príkaz `edit` vyvolá špeciálne prostredie a editovanie tabuliek. Toto prostredie je užitočné na vykonanie drobných zmien po načítaní údajov. Príkazom

```
> xnew <- edit(xold)
```

umožníme editáciu dátového súboru `xold`, pričom pozmenený objekt bude priradený premennej `xnew`. Ak chcete zmeniť pôvodný dátový súbor `xold`, najjednoduchšou cestou je použitie príkazu `fix(xold)`, čo je ekvivalentné s použitím príkazu `xold <- edit(xold)`.

Použite

```
> xnew <- edit(data.frame())
```

na vytvorenie nových údajov prostredníctvom tabuľkového prostredia.

Tabuľka 1: Rozdelenia pravdepodobnosti programu **R**

Rozdelenie	**R-názov**	**ďalšie argumenty**
beta	`beta`	`shape1, shape2, ncp`
binomické	`binom`	`size, prob`
Cauchyho	`cauchy`	`location, scale`
chí-kvadrát	`chisq`	`df, ncp`
exponenciálne	`exp`	`rate`
F	`f`	`df1, df2, ncp`
gama	`gamma`	`shape, scale`
geometrické	`geom`	`prob`
hypergeometrické	`hyper`	`m, n, k`
log-normálne	`lnorm`	`meanlog, sdlog`
logistické	`logis`	`location, scale`
negatívne	`binomial`	`nbinom size, prob`
normálne	`norm`	`mean, sd`
Poissonove	`pois`	`lambda`
Studentove t	`t`	`t df, ncp`
rovnomerné	`unif`	`min, max`
Weibullove	`weibull`	`shape, scale`
Wilcoxonove	`wilcox`	`m, n`

4 Rozdelenia pravdepodobnosti

4.1 Súbor štatistických tabuliek programu R

Jedným z pohodlných použití programu **R** je poskytnutie obsiahlých súborov štatistických tabuliek. K dispozícii sú funkcie na výpočet distribučnej funkcie (anglicky the cumulative distribution function – CDF) $P(X \leqq x)$, funkcie rozdelenia hustoty pravdepodobnosti (anglicky the probability density function – PDF) a inverznej distribučnej funkcie (anglicky the quantile function) (pre dané q, sa q-kvantilom nazýva najmenšie x také, že $P(X \leqq x) > q$) a na simuláciu výberu vzoriek (generovanie pseudonáhodných čísel) s daným rozdelením.

Predpona názvu je ‚`d`' pre hustotu (anglicky density), ‚`p`' pre CDF, ‚`q`' pre kvantilovú funkciu a ‚`r`' na simuláriu (anglicky random numbers). Typ funkcie je určený názvom `xxx`. Prvý argument je `x` pre `dxxx`, `q` pre `pxxx`, `p` pre `qxxx` a `n` pre `rxxx` (s výnimkou funkcií `rhyper` a `rwilcox`, pre ktoré to je `nn`). Tzv. parameter necentrality (anglicky the non-centrality parameter) `ncp` ešte nie je k dispozícii pre väčšinu funkcií.

Všetky funkcie `pxxx` a `qxxx` majú logické argumenty `lower.tail` a `log.p` a funkcia `dxxx` má len argument `log`. Toto umožňuje, napríklad, priame

získanie tzv. kumulatívnej funkcie rizika (anglicky the cumulative alebo „integrated“ hazard function), $H(t) = -\log(1 - F(t))$ pomocou príkazu

```
pxxx(t, ..., lower.tail = FALSE, log.p = TRUE)
```

alebo presnejšie log-vierohodnosť (log-likelihoods) (pomocou použitia funkcie `dxxx(..., log = TRUE)`).

K dispozícii sú navyše funkcie `ptukey` and `qtukey` pre rozdelenie oblasti (the studentized range) výberu s normálnym rozdelením.

Nasleduje niekoľko príkladov.

```
> ## 2-tailed p-value for t distribution
> 2*pt(-2.43, df = 13)
> ## upper 1% point for an F(2, 7) distribution
> qf(0.01, 2, 7, lower.tail = FALSE)
```

4.2 Overenie rozdelenia štatistického súboru

Pre zadaný výberový súbor údajov môžeme overovať jeho rozdelenie viacerými spôsobmi. Najjednoduchším je preveriť čísla. Dva zľahka odlišné sumáre poskytujú funkcie `summary` a `fivenum`, údaje sa dajú znázorniť funkciou `stem` (a „stem and leaf“ plot – kmeň a listy).

```
> attach(faithful)
> summary(eruptions)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.600   2.163   4.000   3.488   4.454   5.100
> fivenum(eruptions)
[1] 1.6000 2.1585 4.0000 4.4585 5.1000
> stem(eruptions)

   The decimal point is 1 digit(s) to the left of the |

   16 | 070355555588
   18 | 000022233333335577777777888822335777888
   20 | 00002223378800035778
   22 | 0002335578023578
   24 | 00228
   26 | 23
   28 | 080
   30 | 7
   32 | 2337
   34 | 250077
   36 | 0000823577
```

```
  38 | 2333335582225577
  40 | 000000335778888800223355577778
  42 | 033355557788002333335555777 8
  44 | 022223355577800000000233333577788888
  46 | 00002333577000000235 78
  48 | 00000022335800333
  50 | 0370
```

Tento graf („kmeň a listy“) sa podobá na histogram, na znázornenie histogramov má **R** funkciu `hist`.

```
> hist(eruptions)
## make the bins smaller, make a plot of density
> hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(eruptions, bw=0.1))
> rug(eruptions) # show the actual data points
```

Krajšie grafy hustoty sa dajú vytvoriť pomocou funkcie `density`, preto sme v tomto príklade pridali riadok vytvorený funkciou `density`. Parameter `bw` (bandwidth – šírka pásu) sme zvolili metódou „pokusov a chýb“, keďže implicitné nastavenie príliš vyhladzovalo priebeh (to sa obyčajne stáva v prípade „zaujímavých“ hustôt). (Existujú aj lepšie metódy výberu parametra `bw`, v tomto príklade nastavenie `bw = "SJ"` dá dobrý výsledok.)

Použitím funkcie `ecdf` môžeme nakresliť graf tzv. empirickej distribučnej funkcie (anglicky the empirical cumulative distribution function – ECDF).

```
> plot(ecdf(eruptions), do.points=FALSE, verticals=TRUE)
```

Toto rozdelenie je zrejme ďaleko od nejakého štandardného rozdelenia. Ako je to s tzv. „pravým režimom“ (the right-hand mode), napríklad s erupciami dlhšími ako 3 minúty? Vyskúšajme aproximáciu pomocou normálneho rozdelenia a nakrelime výslednú funkciu CDF.

```
> long <- eruptions[eruptions > 3]
> plot(ecdf(long), do.points=FALSE, verticals=TRUE)
> x <- seq(3, 5.4, 0.01)
 > lines(x, pnorm(x, mean=mean(long), sd=sqrt(var(long))), lty=3)
```

Precíznejšie to môžeme overiť pomocou tzv. „Quantile-quantile“ (Q-Q) grafu.

```
par(pty="s")       # arrange for a square figure region
qqnorm(long); qqline(long)
```

ktorý ukazuje rozumnú aproximáciu, ale kratší pravý chvost (right tail) ako by sa dalo očakávať pre normálne rozdelenie. Porovnajme to s dátami vytvorenými simuláciou pre *t*-rozdelenie:

```
x <- rt(250, df = 5)
qqnorm(x); qqline(x)
```

ktoré obyčajne vykazuje (ak sa jedná o náhodný výber) dlhší chvost, ako sa očakáva pre normálne rozdeelenie. Nasledujúcim postupom môžeme vytvoriť Q-Q graf voči generovanému rozdeleniu:

```
qqplot(qt(ppoints(250), df = 5), x, xlab = "Q-Q plot for t dsn")
qqline(x)
```

Nakoniec môžeme chcieť vykonať formálnejší test súladu (alebo nesúladu) s normalitou. **R** poskytuje Shapirov-Wilkov test

```
> shapiro.test(long)

        Shapiro-Wilk normality test

data:  long
W = 0.9793, p-value = 0.01052
```

a Kolmogorovov-Smirnovov test

```
> ks.test(long, "pnorm", mean = mean(long), sd = sqrt(var(long)))

        One-sample Kolmogorov-Smirnov test

data:  long
D = 0.0661, p-value = 0.4284
alternative hypothesis: two.sided
```

(Všimnime si, že v tomto prípade neplatí teória rozdelení, pretože sme parametre normálneho rozdelenia odhadovali z tej istej vzorky.)

4.3 Jedno- a dvoj-výberové testy

Doteraz sme porovnávali jeden výber s normálnym rozdelením. Častejšie sa vyskytuje úloha porovnania parametrov dvoch výberov. Poznamenajme, že v programe **R** sú všetky „klasické“ testy vrátane nižšie použitého súčasťou balíku `stats`, ktorý je obyčajne načítaný.

Uvažujme nasledujúci súbor údajov merania skupenského tepla topenia ľadu (cal/gm) od Ricea (1995, p. 490):

```
Method A: 79.98 80.04 80.02 80.04 80.03 80.03 80.04 79.97
          80.05 80.03 80.02 80.00 80.02
Method B: 80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97
```

Jednoduché gragické porovnanie dvoch vzoriek poskytuje funkcia `boxplot`:

```
A <- scan()
79.98 80.04 80.02 80.04 80.03 80.03 80.04 79.97
80.05 80.03 80.02 80.00 80.02

B <- scan()
80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97

boxplot(A, B)
```

Graf svedčí o tom, že prvá skupina dáva vyššie výsledky ako druhá.

Rovnosť stredných hodnôt dvoch výberov môžeme otestovať nepárovým t-testom nasledujúcim spôsobom:

```
> t.test(A, B)

         Welch Two Sample t-test

data:  A and B
t = 3.2499, df = 12.027, p-value = 0.00694
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
0.01385526 0.07018320
sample estimates:
mean of x mean of y
80.02077  79.97875
```

Za predpokladu normality je rozdiel významný. Funkcia **R** implicitne nepredpokladá rovnosť disperií dvoch výberov (na rozdiel od podobnej funkcie `t.test` systému S-PLUS). Na otestovanie rovnosti disperzií môžeme použiť F-test, za predpokladu, že obidva výbery sú robené z populácií s normálnym rozdelením.

```
> var.test(A, B)

         F test to compare two variances

data:  A and B
F = 0.5837, num df = 12, denom df =  7, p-value = 0.3938
alternative hypothesis: true ratio of variances is not equal to 1
```

```
95 percent confidence interval:
 0.1251097 2.1052687
sample estimates:
ratio of variances
         0.5837405
```

Test nevykazuje významné rozdiely, preto môžeme použiť klasický t-test predpokladajúci rovnosť disperzií:

```
> t.test(A, B, var.equal=TRUE)

         Two Sample t-test

data:  A and B
t = 3.4722, df = 19, p-value = 0.002551
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.01669058 0.06734788
sample estimates:
mean of x mean of y
 80.02077  79.97875
```

Všetky tieto testy predpokladajú normalitu obidvoch výberov. Dvojvýberový Wilcoxonov (alebo Mannov-Whitneyov) test predpokladá pre nulovú hypotézu len *spoločné spojité rozdelenie*:

```
> wilcox.test(A, B)

         Wilcoxon rank sum test with continuity correction

data:  A and B
W = 89, p-value = 0.007497
alternative hypothesis: true location shift is not equal to 0

Warning message:
Cannot compute exact p-value with ties in: wilcox.test(A, B)
```

V každej vzorke sú náznaky toho, že tieto údaje sú z diskrétneho rozdelenia (možno pre zaokrúhľovanie).

To je niekoľko spôsobov grafického porovnania dvoch vzoriek. Práve sme videli pár „boxplot“-ov. Nasledujúce príklady

```
> plot(ecdf(A), do.points=FALSE, verticals=TRUE, xlim=range(A, B))
> plot(ecdf(B), do.points=FALSE, verticals=TRUE, add=TRUE)
```

ukážu dve empirické CDF a funkcia `qqplot` vyrobí Q-Q graf dvoch vzoriek. Test Kolmogorova-Smirnova overuje maximálnu vertikálnu odchýlku medzi dvomi funkciami ECDF za predpokladu spoločného spojitého rozdelenia:

```
> ks.test(A, B)

        Two-sample Kolmogorov-Smirnov test

data:  A and B
D = 0.5962, p-value = 0.05919
alternative hypothesis: two-sided

Warning message:
cannot compute correct p-values with ties in: ks.test(A, B)
```

5 Štatistické modely v R

Podkladom pre zavedenie štatistického modelu je lineárny regresný model s nezávislými homoskedastickými chybami e_i, ktorý môžeme zapísať v tvare

$$y_i = \sum_{j=0}^{p} \beta_j x_{ij} + e_i, \quad i = 1, \ldots, n, \tag{1}$$

kde e_i vyhovujú podmienke NID$(0, \sigma^2)$.[6] V maticovom tvare bude rovnica (1) vyzerať takto

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{e}, \tag{2}$$

kde $\boldsymbol{y}$ je vektor závisle premennej alebo tiež odozva, $\boldsymbol{\beta}$ je vektor neznámych parametrov, $\boldsymbol{X}$ je modelová matica alebo matica funkčných hodnôt[7] so stĺpcami x_0, x_1, ..., x_p, ktoré určujú nezávisle premenné. Často je stĺpec x_0 jednotkový a definuje hodnoty „priesečníkov s osou y“ (an *intercept* term, ďalej budeme používať výraz „priesečník“).

Príklady

Pred zavedením formálnych špecifikácií uvedieme niekoľko jednoduchých užitočných príkladov. Predpokladajme, že `y, x, x0, x1, x2, ...` sú numerické premenné, `X` je matica a `A, B, C, ...` sú činitele (faktory), potom nasledujúcimi vzťahmi môžeme špecifikovať niektoré štatistické modely: Operátor ~ je používaný na definovanie *predpisu modelu* v programe **R**. Tvar pre všeobecný lineárny model je

```
response ~ op_1 term_1 op_2 term_2 op_3 term_3 ...,
```

kde `response` je vektor alebo matica, ktorá definuje rozsah premennej(ných), `op_i` je operátor + alebo − vyjadrujúci včlenenie alebo vylúčenie výrazu `term_i` do modelu, `term_i` môže byť vektor, matica alebo `1`, ľubovolný činiteľ alebo vyjadrenie, ktoré obsahuje činitele, vektory alebo matice spojené operátorovým vyjadrením. Viac sa dočítate v príručke k programu **R** (VENABLES ET AL., 2006).

5.1 Lineárne modely

Základnou funkciou na fitovanie všeobecných mnohonásobných modelov je funkcia `lm()`, jej volanie v príkazovom riadku je nasledovné:

```
> fitted.model <- lm(formula, data = data.frame)
```

Napríklad príkazom

[6] NID – Normally and Independently Distributed (statistics)

[7] Niekedy ju označujú ako maticu hodnôt pozorovaní nezávisle premenných.

`y ~ x`
`y ~ 1 + x` obidva príklady vyjadrujú jednoduchý lineárny regresný model $y(x)$. V prvom je hodnota „priesečníka“ implicitná, v druhom je explicitne určená hodnota 1.

`y ~ 0 + x`
`y ~ -1 + x`
`y ~ x - 1` toto sú jednoduché modely lineárnej regresie $y(x)$, ktoré prechádzajú počiatkom.

`log(y) ~ x1 + x2` multilineárna regresia s transformovanou premennou $\log(y)$ a implicitnou hodnotou „priesečníka“.

`y ~ poly(x,2)`
`y ~ 1 + x + I(x^2)` polynomiálna regresia $y(x)$ druhého stupňa. Prvý tvar používa ortogonálne polynómy a druhý používa explicitné umocnenie ako základ.

`y ~ X + poly(x,2)` mnohonásobná regresia y s maticovým modelom, ktorý obsahuje maticu X a polynomiálne hodnoty x druhého stupňa.

`y ~ A` jednoduchá klasifikácia analýzy rozptylu (analysis of variance) modelu y s triedou určenou faktorom A.

`y ~ A + x` jednoduchá klasifikácia analýzy kovariancie (analysis of covariance) modelu y s triedou určenou faktorom A a s kovariovanými x.

`y ~ A * x`

`y ~ A/x`

`y ~ A/(1 + x) - 1` separované jednoduché lineárne regresné modely $y(x)$ v zmysle úrovne A s rozdielnym kódovaním. Posledný tvar produkuje explicitné odhady pretože množstvo hodnôt „priesečníkov“ a smerníc je určených stupňom A.

```
> fm2 <- lm(y ~ x1 + x2, data = production)
```

budeme fitovať mnohonásobný regresný model $y(x_1, x_2)$ s implicitnými hodnotami „priesečníkov“.

Dôležitý avšak technický voliteľný parameter `data = production` stanovuje, že premenné potrebné na koštrukciu modelu budú najprv načítané zo súboru dát `production`. V tomto prípade bez ohľadu na to či je prístup k súboru `production` zapísaný do vyhľadavacej cesty (search path) alebo nie.

5.2 Funkcie na získanie informácií o modely

Hodnoty funkcie `lm()` sú výsledkom fitovaného modelu; technicky výpis výsledkov je triedou `"lm"`, ktorá obsahuje tieto položky

`add1`	`coef`	`effects`	`kappa`	`predict`	`residuals`
`alias`	`deviance`	`family`	`labels`	`print`	`step`
`anova`	`drop1`	`formula`	`plot`	`proj`	`summary`

Stručne vysvetlíme význam najpoužívanejších:

`anova(object_1, object_2)` porovnáva submodel s vonkajším modelom a vytvára tabuľku analýzy rozptylu.

`coefficients(object)` vyberá regresné koeficienty (maticu), skrátená forma má tvar `coef(object)`.

`deviance(object)` reziduálna suma štvorcov odchýlok, je vahovaná ak sú váhy priradené.

`formula(object)` vyberá formulu modelu.

`plot(object)` vytvára štyri grafy, ktoré zobrazia rezíduá, fitované hodnoty a niektoré diagnostiky.

`print(object)` vytlačí sa stručná verzia objektu. Často sa používa implicitne.

`residuals(object)` vyberá maticu rezíduí, je vahovaná ak sú váhy priradené, skrátená forma má tvar `resid(object)`.

`step(object)` vyberá vhodný model zvyšovaním alebo znižovaním výrazov a uchováva ich hierarchiu. Model s najmenšou hodnotou AIC (Akaikeho informačné kritérium) sa zobrazuje postupným hľadaním s návratom.

`summary(object)` vytlačí sa obsiahlý súhrn výsledkov regresnej analýzy.

`vcov(object)` vráti variančnú a kovariančnú maticu hlavných parametrov fitovaného modelu.

5.3 Nelineárna metóda najmenších štvorcov

Istou formou nelineárneho modelu vhodného na fitovanie je všeobecný lineárny model `glm()`, ale vo väčšine prípadoch volíme niektorý postup

nelineárnej optimalizácie. Program **R** ponúka niekoľko modulov na nelineárnu optimalizáciu ako sú `optim()`, `nlm()` a (z verzie **R** 2.2.0) `nlminb()`, ktoré sú funkciami podobné modulom `ms()` a `nlminb()` z programu S-Plus.

Hodnoty parametrov sú vyhľadávané iteráciou minimalizovaním niektorého ukazovateľa kvality fitovania. Na rozdiel od lineárnej regresie, napríklad, nie je zaručené, že procedúra bude konvergovať k uspokojivým odhadom.

Všetky metódy vyžadujú začiatočné odhady hľadaných parametrov a konvergencia zavisí od kvality týchto štartovacích hodnôt.

5.3.1 Príklad nelineárnej metódy najmenších štvorcov

Jednou z možností fitovania nelineárneho modelu je minimalizácia sumy štvorcov odchýlok (SSE – sum of the squared errors) alebo rezíduí. Táto metóda má zmysel vtedy, keď pozorované chyby majú normálne rozdelenie. Uvedieme príklad z knihy Batesa a Wattsa (1988). Dáta sú nasledovné:

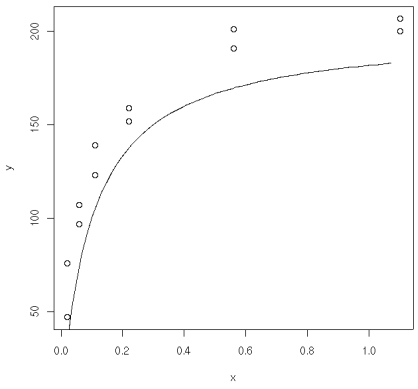
```
> x <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56,
         0.56, 1.10, 1.10)
> y <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)
```

Model, ktorý budeme fitovať zapíšeme v tvare:

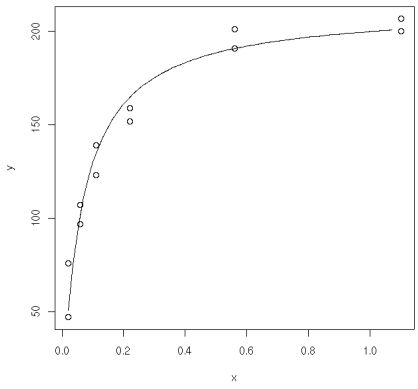
```
> fn <- function(p) sum((y - (p[1] * x)/(p[2] + x))^2)
```

Na to, aby fitovanie prebehlo v poriadku, potrebujeme zadať začiatočné odhady hľadaných parametrov. Jedna z možností nájsť ich vhodné hodnoty je zobrazenie dát, odhadnúť z nich hodnoty niektorých parametrov a potom vykresliť krivku modelu s týmito hodnotami.

```
> plot(x, y)
> xfit <- seq(.02, 1.1, .05)
> yfit <- 200 * xfit/(0.1 + xfit)
> lines(spline(xfit, yfit))
```



Obr. 1: Zobrazenie dát a fitovanej krivky so štartovacími odhadmi parametrov 200 a 0.1



Obr. 2: Zobrazenie dát a fitovanej krivky s optimálnymi odhadnutými parametrami

Mohli sme to urobiť aj lepšie, ale štartovacie hodnoty 200 a 0.1 sa ukazujú byť primerané. Teraz začneme s fitovaním:

```
> out <- nlm(fn, p = c(200, 0.1), hessian = TRUE)
```

Po ukončení fitovania položka `out$minimum` je SSE a položka `out$estimate` vyjadruje odhady parametrov metódou najmenších štvorcov. Na získanie štandardných chýb (SE) apoximovaných odhadov musíme previesť nasledujúcu operáciu:

```
> sqrt(diag(2*out$minimum/(length(y) - 2) * solve(out$hessian)))
```

Číslo 2 v predchádzajúcom riadku reprezentuje počet parametrov. Konfidenčný interval pre 95% dáva pre odhadované parametre hodnoty ± 1.96 SE. Teraz môžeme zobraziť výsledky v novom grafe, pozri obrázok 2:

```
> plot(x, y)
> xfit <- seq(.02, 1.1, .05)
> yfit <- 212.68384222 * xfit/(0.06412146 + xfit)
> lines(spline(xfit, yfit))
```

Štandardný balík `stats` poskytuje rozšírené možnosti na fitovanie nelineárnych modelov metódou najmenších štvorcov. Model, ktorý práve fitujeme je Michaelisov-Mentenov model, pre ktorý použijeme príkazy:

```
> df <- data.frame(x=x, y=y)
> fit <- nls(y ~ SSmicmen(x, Vm, K), df)
> fit
Nonlinear regression model
model: y ~ SSmicmen(x, Vm, K)
data: df
```

```
      Vm     K
212.68370711 0.06412123
residual sum-of-squares: 1195.449
> summary(fit)
Formula: y ~ SSmicmen(x, Vm, K)
Parameters:
Estimate Std. Error t value Pr(>|t|)
Vm 2.127e+02 6.947e+00 30.615 3.24e-11
K 6.412e-02 8.281e-03 7.743 1.57e-05
Residual standard error: 10.93 on 10 degrees of freedom
Correlation of Parameter Estimates:
      Vm
K 0.7651
```

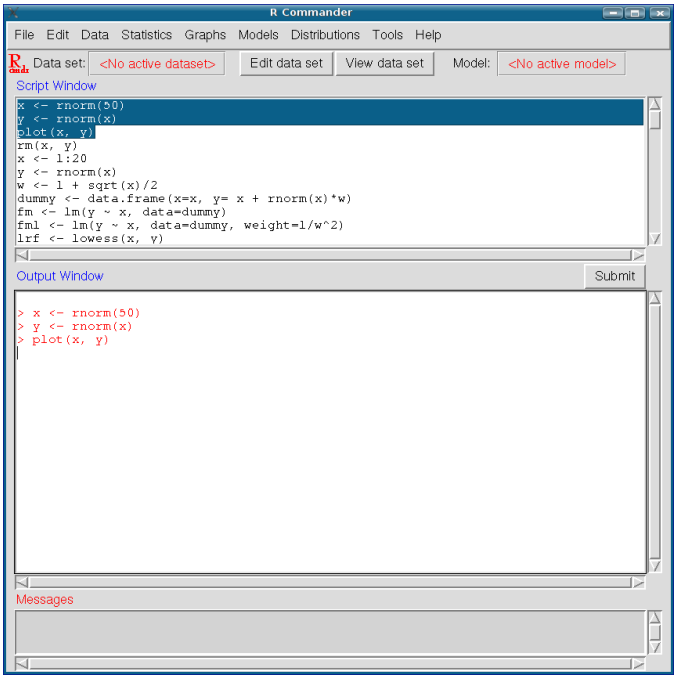
6 Ukážkové lekcie

6.1 R Commander

Súčasťou programu **R** je modul `Rcmdr`, ktorý je v podstate grafický (GUI) uživatelský interfejs a umožňuje pohdlnú prácu s programom. V prostredí UNIX sa program **R** spúšťa z príkazového riadku odoslaním príkazu `R`. Po štarte programu **R** sa `RCommander` aktivuje načítanim knižnice `Rcmdr` zápisom a odoslaním príkazu

```
> library(Rcmdr)
```

Po načítaní knižnice sa nám otvorí okno programu (obrázok 3). Podrobný prehľad a opis modulu najdete v článku J. Foxa (1988).



Obr. 3: Pracovné prostredie grafického interfejsu `Rcmdr`

6.2 Vytvorenie, diagnostika a zobrazenie dát

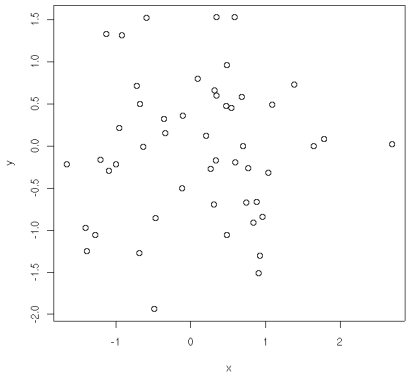
Začneme príkazmi, ktoré vygenerujú dva pseudonáhodné vektory x a y

```
> x <- rnorm(50)
> y <- rnorm(x)
```

vykreslíme ich príkazom

```
> plot(x, y)
```

výsledok znázorňuje obrázok 4.



Obr. 4: Priebeh pseudonáhodných vektorov x a y

Príkazom

```
> ls()
[1] "x" "y"
```

sa môžeme presvedčiť či sú objekty načítané do pracovného priestoru programu **R**. V prípade, keď už vytvorené objekty nebudeme potrebovať, môžeme ich vymazať príkazom

```
> rm(x, y)
> ls()
character(0)
```

Vytvorme nový objekt s váhovým vektorom štandardných odchýlok a uložíme ho do dvojstĺpcovej tabuľky, ktorú si napokon zobrazíme

```
> x <- 1:6 # x = (1, 2, ..., 6)
> w <- 1 + sqrt(x)/2
> dummy <- data.frame(x = x, y = x + rnorm(x)*w)
> dummy

  x          y
1 1 -0.1501478
2 2  2.4534085
3 3  3.7375891
4 4  4.0030544
5 5  4.6228321
6 6  6.9000943
```

Fitovanie a zobrazenie výsledkou analýzy dát $y(x)$ jednoduchou lineárnou regresiou vykonáme príkazmi

```
> fm <- lm(y ~ x, data=dummy)
> summary(fm)

Call:
lm(formula = y ~ x, data = dummy)
Residuals:
      1       2       3       4       5       6
-0.7428  0.6600  0.7435 -0.1918 -0.7727  0.3038
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.6080     0.7007  -0.868  0.43452
x             1.2007     0.1799   6.673  0.00262 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.7527 on 4 degrees of freedom
Multiple R-Squared: 0.9176,     Adjusted R-squared: 0.897
F-statistic: 44.53 on 1 and 4 DF,  p-value: 0.002621
```

Keďže poznáme štandardné odchýlky môžeme vykonať váhovanú regresiu

```
> fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)
> summary(fm1)

Call:
lm(formula = y ~ x, data = dummy, weights = 1/w^2)
Residuals:
       1        2        3        4        5        6
-0.41564  0.43031  0.41438 -0.10340 -0.39307  0.08954
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.7722     0.6482  -1.191  0.29941
x             1.2455     0.1860   6.696  0.00259 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.4192 on 4 degrees of freedom
Multiple R-Squared: 0.9181,     Adjusted R-squared: 0.8976
F-statistic: 44.83 on 1 and 4 DF,  p-value: 0.002588
```

Príkazom

```
attach(dummy)
```

učiníme stĺpce dátoveho súboru viditeľné ako premenné. Na vytvorenie neparametrickej lokálnej regresnej funkcie použijeme operáciu

```
> lrf <- lowess(x, y)
```

a napokon zobrazíme dáta

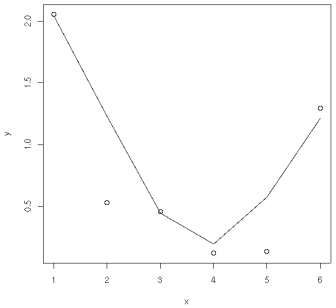
```
> plot(x, y)
```

a pridáme do grafu priebeh lokálnej regresie

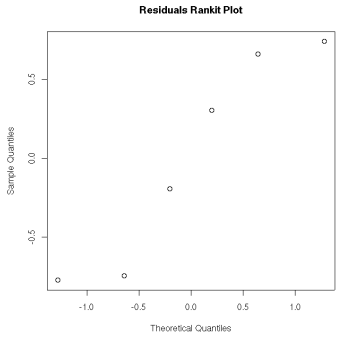
```
> lines(x, lrf$y)
```

Výsledok je znázornený na obrázku 5. Na opis ďalších vlastností náhodnej premennej, ako šikmosť (nesúmernosť) a špicatosť používame momentové charakteristiky. Vynesením teoretických a výberových kvantilov priamo z dát proti sebe do jedného grafu sa získa Q-Q graf[8], ktorý je základným grafickým nástrojom na diagnostiku rozdelenia dát, predovšetkým normality. Obrázku 6 znázorňuje priebeh takého vyhodnotenia dát na šiknosť, špicatosť a „vybočujúce“ body (outliers), získali sme ho príkazom

```
qqnorm(resid(fm), main="Residuals Rankit Plot")
```



Obr. 5: Zobrazenie dát $y(x)$ a krivky vytvorenej lokálnou regresnou funkciou `lrf`

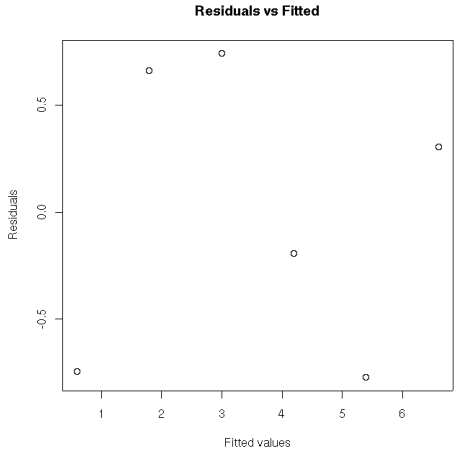


Obr. 6: Q-Q graf – výberová versus teoretická kvantilová funkcia

Na štandardnú diagnostiku regresie používame grafickú kontrolu heteroskedastivity rezíduí. Graf možeme vytvoriť zobrazením fitovaných hodnôt a rezíduí proti sebe (pozri obrázok 7)

```
plot(fitted(fm), resid(fm),
     xlab="Fitted values",
     ylab="Residuals",
     main="Residuals vs Fitted")
```

[8] Z tvaru Q-Q grafu sa dá spoľahlivo posúdiť symetria, normalita, špicatosť, prítomnosť „vybočujúcich“ dát a homogenita výberu.



Obr. 7: Kontrola heteroskedastivity rezíduí pre dáta `dummy`

6.3 Vrstevnicový graf a obrázok s farebnou mapou

Vrstevnicový graf zobrazuje vrstevnice nejakej plochy. V krátkosti uvedieme princíp vytvorenia takého grafu. Na vykreslenie vrstevníc potrebujeme maticu, ktorá reprezentuje hodnoty závisle premennej nejakej plochy. Pomocou funkcie `outer` sa vytvorí zmienená matica. Najprv však pripravíme vektorové dáta x a y rovnakých rozmerov s dĺžkou 50 bodov

```
> x <- seq(-pi, pi, len=50)
> y <- x
```

Príkazom

```
> f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
```

teda vytvoríme štvorcovú maticu `f(x, y)` s hodnotami danými funkciou $f = \cos(y)/(1 + x^2)$. Teraz už môžeme zobraziť mapu vrstevníc funkcie f s dvadsiatimi vrstvami

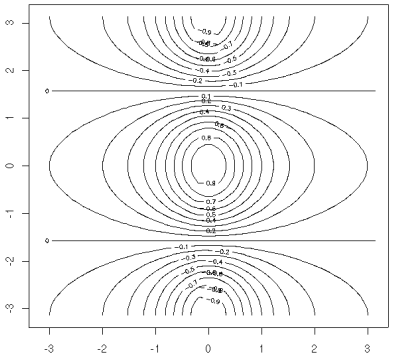
```
> contour(x, y, f, nlevels=20, add=TRUE)
```

Výsledok je znázornený na obrázku 8. Graf môžeme aj vyfarbiť a dať tak vrstvám farebnú hĺbku

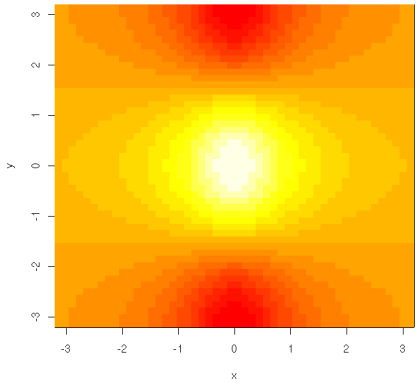
```
> image(x, y, f, col=heat.colors(20))
```

Takto upravený vrstevnicový graf ilustruje obrázok 9.[9]

[9] Ďalšie voľby sú `terrain.colors` a `topo.colors` – vyskúšajte ich.



Obr. 8: Vrstevnicový graf funkcie $f = \cos(y)/(1+x^2)$ s pridaním popiskov



Obr. 9: Vrstevnicový graf s vyplnenými farebnými plochami s farebnou hĺbkou 20

Záver

Sme presvedčení, že program **R** patrí medzi špičkové systémy na štatistické spracovanie, analýzu dát a ich grafické zobrazenie. Veríme, že aj táto príručka prispeje k rozšíreniu jeho používateľov predovšetkým v akademickom prostredí.

Sme si vedomí, že sme pri jej písaní zďaleka nevyčerpali všetky možnosti, ktoré program poskytuje a veríme, že napriek tomu bude príručka užitočná pre širokú komunitu začínajúcu pracovať v prostredí systému **R**. Dúfame, že náš počin nájde kladnú odozvu čitateľskej obce.

Uvítame všetky Vaše pripomienky a návrhy, posielajte ich láskavo na adresy `Jan.Busa@tuke.sk` a `Ladislav.Sevcovic@tuke.sk`.

Literatúra

D. M. Bates, D. G. Watts 1988. *Nonlinear Regression Analysis and its Applications*. John Wiley & Sons, New York, p. 51

J. Fox 2005. *The R Commander: A Basic-Statistics Graphical User Interface to R*. In: Journal of Statistical Software, September 2005, Volume 14, Issue 9, online na `http://www.jstatsoft.org/`

J. A. Rice 1995. *Mathematical Statistics and Data Analysis*. Second edition. Duxbury Press, Belmont, CA

W. N. Venables, D. M. Smith, and the R Development Core Team 2006. *An Introduction to R*, ISBN 3-900051-12-7, online na `http://www.r-project.org/`