Uvod u korištenje R-a

Programskom okruženju za analizu podataka i grafički prikaz
Verzija 2.0.1 (15.11.2004)

W.N. Venables, D.M. Smith
i osnovna grupa suradnika za razvoj R-a

Prevela: Maja Kumbatović
Pregledao i dopunio: Damir Kasum
Urednik: Tarzan Legović

Napomena: Svi primjeri ovdje navedeni mogu se prebaciti u R i izvršiti, jednostavnim naredbama Copy (Ctrl+C) i Paste (Ctrl+V). U prijevodu je korištena uobičajena matematička i računalna terminologija hrvatskog jezika. Da bi se izbjegle nedoumice na pojedinim mjestima je prijevod proširen i odstupa od originala, te su dodani primjeri koji čitaocu mogu olakšati razumjevanje ovog priručnika.

SADRŽAJ

Predgovor

Ovaj uvod u R je napravljen prema originalnim bilješkama koje opisuju S i S-PLUS okruženje napisanim od strane Billa Venablesa i Davida M. Smitha (Insightful Corporation). Izmjenili smo mnoge dijelove da bismo ukazali na razliku između R i S programa, te smo proširili neke dijelove materijala.
Željeli bismo se zahvaliti Billu Venablesu na dozvoli za distribuiranje ove modificirane verzije bilježaka i na njegovoj dosadašnjoj potpori R-u.
Primjedbe i ispravci su uvijek dobro došli. Molim da email prepisku adresirate na R-core@r-project .org.

Preporuke čitatelju

Početnici u korištenju R-a počet će s primjerom rada u R-u (Dodatak A). To bi ih trebalo upoznati sa stilom rada u R-u i, još važnije, dati trenutnu povratnu informaciju o onome što se aktualno događa.
Mnogi korisnici koristit će R okruženje uglavnom zbog njegovih grafičkih mogućnosti. U tome slučaju mogu pročitati Poglavlje 12 [Upotreba grafičkih naredbi], str. 68, o grafičkim mogućnostima, u bilo koje vrijeme, bez potrebe čekanja dok prouče sva prethodna poglavlja.

1 Uvod

1.1 R okruženje

R predstavlja integrirano programsko okruženje za upravljanje podacima, računanje i grafički prikaz. Između ostalog posjeduje

- mogućnost za učinkovito upravljanje podacima i njihovo pohranjivanje
- niz operatora za računanje sa poljima podataka, a posebno matricama
- veliku, koherentnu, integriranu zbirku programskih alata za analizu podataka
- grafičke mogućnosti za analizu podataka te za njihovo prikazivanje direktno na zaslonu računala ili na papiru, te
- dobro razvijen, jednostavan i učinkovit programski jezik koji uključuje uvjetne tvrdnje, petlje, rekurzivne funkcije definirane od strane korisnika te postupke za učitavanje i spremanje podataka. (Većina samih funkcija koje omogućuje sustav je pisana u S jeziku.)

Izraz "okruženje" upotrijebljen je da istakne da je R dobro planiran i konzistentan sustav a ne sustav koji se postepeno dopunjavanja s specifičnim i nefleksibilnim programskim alatima, što je često slučaj kod drugih programa za analizu podataka.
.

1.2 Odgovarajući programska podrška (software) i dokumentacija

R se može promatrati kao implementacija S jezika koji je bio razvijen u Bell laboratorijima od strane Ricka Beckera, Johna Chambersa i Allana Wilksa, te također predstavlja osnovicu S-PLUS sustava.
Razvoj S jezika je opisan u četiri knjige Johna Chambersa i suradnika. Osnovna literatura za R je *Novi S jezik: Programsko okruženje za analize podataka i grafiku* autora Richard A. Becker, John M. Chambers i Allan R. Wilks. Nove svojstva S-a iz 1991. godine (S verzija 3) opisane su u *Statistsički modeli u S-u* izdavača Johna M. Chambersa i Trevora J. Hastie-a. Vidi Dodatak D [Reference], str. 91, za detaljne reference.

Osim navedenoga, dokumentacija za S/S-PLUS može se u principu upotrijebiti za R, imajući na umu razlike između implementcije S-a i R-a. Vidi poglavlje "Koja dokumentacija postoji za R?" u *FAQ o R statističkom sustavu*.

1.3 R i statistika

U ovom uvodu još nije spomenuta *statistika*, a ipak mnogi koriste R kao statistički sustav. Mi radije razmišljamo o R-u kao okruženju u kojem su ugrađene klasične i moderne statističke tehnike. Neke od njih su ugrađene u osnovu R okruženja, ali mnoge se dobavljive kao *paketi.* (Trenutno je razlika između toga stvar povijesnog razvoja). Otprilike 8 paketa dolazi s R-om (tzv. "standardni" paketi) a mnogo više ih je dostupno preko CRAN Internet stranice (via http://cran.r-project.org).
Većina klasičnih statistika i mnoge od najnovijih metodologija dostupne su za korištenje putem R-a ali će se korisnici morati oko toga malo potruditi.
Postoji značajna razlika u filozofiji između S-a (prema tomu i R-a) i drugih glavnih statističkih sustava. U S-u se statistička analiza obično provodi u nizu koraka pohranjujući međurezultate pojedinih koraka u objekte. Naprimjer, SAS i SPSS će dati obilne rezultate iz regresijske ili diskriminantne analize dok će R dati minimalni rezultat i pohraniti rezultate u odgovarajuće objekte za kasnija ispitivanja drugim R funkcijama.

1.4 R i grafički sustavi (window system)

Najpogodniji način korištenja R-a je na grafičkoj radnoj stanici koja pokreće sustav grafičkih prozora. Ovo je uputstvo namijenjeno korisnicima koju posjeduju tu mogućnost Posebno ćemo se povremeno osvrnuti na korištenje R-a u X grafičkom sustavu iako se većina informacija općenito odnosi na bilo koji rad u R okruženju.
Većina korisnika će trebati povremeno koristiti operativni sustav na svom računalu. U ovom uputstvu uglavnom se raspravlja o interakciji s operativnim sustavima na UNIX računalima. Ako se koristite R-om na Microsoft Windowsima trebat ćete napraviti male prilagodbe.
Postavljanje postavki za korištenje svih prilagodljivih svojstava R-a je direktan iako pomalo zamoran postupak i o njemu ovdje više neće biti raspravljano. Korisnici koji budu imali poteškoća morat će se obratiti za pomoć lokalnom stručnjaku ili potražiti pomoć na internet stranicama www.r-project.org.
.

1.5 Interaktivno korištenje R-a

R program daje znak (prompt) kad očekuje naredbe za unos podataka. Poćetna postavka (default) je znak '>', što na UNIX-u može biti isto kao znak ljuske (shell-a), tako da može izgledati kao da se ništa ne događa. Vidjet ćemo, međutim, da je lako promijeniti prompt ako se to želi. Pretpostavimo da je u UNIX-u prompt ljuske '$'.
Kod korištenja R-a na UNIX-u za početak se preporuča sljedeći postupak:

1. Napravite posebni poddirektorij, npr. 'work', za pohranjivanje datoteka s podacima i koji ćete koristiti R za rješavanje određenog problema. To će biti radni direktorij kadgod upotrijebite R za ovaj taj problem.

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

2. Startajte R pomoću naredbe

   ```
   $ R
   ```

   Na ekranu će se pojaviti poruka (zavisi o verziji R-a koja je instalirana):

```
R : Copyright 2004, The R Foundation
    Version 1.9.1 Patched (2004-07-29)
    >
```

3. Ušli ste u R okruženje i možete koristiti R naredbe (vidi kasnije u tekstu)
4. Za izlaženje iz R programa koristi se naredba
5.

```
> q( )
Save workspace image? [y/n/c]:
```

Kod toga će vam biti postavljeno pitanje želite li pohraniti podatke koje ste unijeli u R, tj. cijelu radnu okolinu. Možete odgovoriti sa ***yes***, ***no*** odnosno ***cancel*** (dovoljno je pritisnuti od jednog slova) da biste pohranili podatke prije napuštanja programa, napustili program bez pohranjivanja podataka odnosno vratili se na rad u R-u. Pohranjeni podaci biti će dostupni u budućem radu u R-u.
Na isti način, ponovo možete pokrenuti R:.

1. Idete u 'work' radni direktorij i startajte program kao i ranije:

```
$ cd work
$ R
```

2. Koristite se R programom, te završavate rad s q( ) naredbom.

Za korištenje R-a u MS Windows-u postupak je u osnovi isti. Kreirajte folder, tj. radni direktorij (iz Windows Explorera). Na glavnoj radnoj površini nađete ikonu R (koja je postavljena prilikom instalacije R-a), zatim pritisnete desnu tipku miša *na ikoni R-a*, te dobijete padajući izbornik. Idite na opciju *Properties*, gdje će vam se otvoriti prozor u kojem će te u polje *Start in* upisati punu putanju (path) do vašeg radnog foldera (nemojte zaboraviti piritsnuti gumbe *Apply* i *OK*). Zatim pokrenite R dvostrukim pritiskom na ikonu.

1.6 Uvod u korištenje R-a

Čitateljima koji žele dobiti osjećaj za korištenje R-a na računalu prije samog početka korištenja preporuča se pročitati uvod u korištenje R-a u Dodatku A [Primjer rada u R okruženju], str. 83.

1.7 Pomoć pri korištenju funkcija i svojstava

R posjeduje sustav pomoći slično *man*-u u UNIX-u. Za dobivanje podrobnijih podataka o bilo kojoj posebno imenovanoj funkciji, npr. solve, koristi se naredba

```
> help(solve)
```

ili

```
> ?solve
```

Na Unix sustavima pomoć će se prikazati u predefiniranom formatu (na MS Windows-u u posebnom prozoru), i koristimo tipku <SPACE> - razmaknicu za pomak kroz tekst (scroll), a pregled završavamo pritskom na tipku <Q>.
Za svojstvo specificirano posebnim znakovima, argument mora biti naveden unutar dvostrukih ili jednostrukih navodnika kao "niz znakova": Isto je potrebno kod nekoliko riječi sintaktičkog značenja uključujući if, for i function.

```
> help( "[[")
```

Jedan oblik navodnika može se upotrijebiti da bi se izbjegao drugi, kao u nizu "It's important". Dogovor je da se preferiraju dvostruki navodnici.
Dostupan je još jedan način pretraživanja pomoći: pojmovno pretraživanje – po zadanim riječima. Na primjer tražimo gdje se nalazi pojam "linear models":

```
> help.search("linear models")
```

Rezultat je:

```
Help files with alias or concept or title matching 'linear
models' using fuzzy matching:
glm.nb(MASS)   Fit a Negative Binomial Generalized
                 Linear Model
lm.gls(MASS)   Fit Linear Models by Generalized
                 Least Squares
loglm(MASS)    Fit Log-Linear Models by Iterative
                 Proportional Scaling
...... .......
```

Dobijemo ime funkcije gdje se koristi zadani pojam (u zagradi je naveden paket gdje se nalazi funkcija) i opis funkcije. daljnja pomoć se može dobiti npr. sa slijedećom naredbom:

```
> help(glm.nb, package="MASS")
```

Kod većine R instalacija pomoć se može dobiti u HTML formatu pokretanjem

```
> help.start( )
```

koji će pokrenuti Web preglednik (obično, Netscape na UNIX-u) što omogućuje pretraživanje stranica pomoću hiperlinkova. Na UNIX-u se daljnja pomoć traži putem HTML baziranog sustava za pomoć. 'Search Engine and Keywords' link na stranici učitan pomoću help.start( ) je posebno koristan jer sadrži listu s mnogo pojmova koja pretražuje kroz dostupne funkcije. To može biti dobar način za brzo usmjeravanje i za shvaćanje opsega mogućnosti koje R nudi.
Help.search naredba omogućuje traženje pomoći na različite načine: pokušajte ?help.search za detalje i primjere.

Primjeri za help topic mogu se pokrenuti sa

```
> example(topic)
```

Windows verzija R-a ima na izbor druge sustave za pomaganje: upotrijebite

```
> ?help
```

za daljne pojedinosti.

1.8 Naredbe u R-u, velika i mala slova, itd.

Tehnički je R *jezik izražavanja (interpreter)* vrlo jednostavne sintakse. On razlikuje velika i mala slova kao većina programa baziranih na UNIX-u, tako su A i a različiti znakovi i predstavljaju različite varijable. Skup znakova koji se mogu upotrebljavati u R nazivima ovisi o operativnom sustavu i o jezičnim postavkama računala (*locale, regional settings*).

Dozvoljeni su svi alfanumerički znakovi i u nekim zemljama to uključuje slova s akcentima plus '.'[1], uz ograničenje da naziv ne može početi s brojkom.
Osnovne naredbe se sastoje od *izraza (expressions)* ili od *pridruživanja (assignments)*. Ako je izraz dan kao naredba, on se ispituje, prikazuje na ekranu, i vrijednost se gubi. Pridruživanjem se također evaluira izraz i prenosi vrijednost u varijablu ali se rezultat automatski ne prikazuje.
Naredbe se odvajaju s pomoću točke-zareza (';') ili s prelaskom u novi red. Osnovne naredbe se mogu zajedno grupirati u jedan složeni izraz s pomoću zagrada ('{' i '}'). *Komentari* se mogu staviti gotovo svugdje[2], počevši od znaka ('#'), sve do kraja reda je komentar.
Ako naredba nije potpuna na kraju reda, R će dati drukčiji prompt:

```
+
```

u drugom i u narednim redovima te nastaviti čitati unos dok naredba nije sintaktički potpuna. Korisnik može promijeniti prompt +. U ovom priručniku ćemo izostavljati prompt za nastavak naredbe i nastaviti u novom redu uvlačenjem teksta.
Rezultat naredbe biti će ispisan na ekranu. Ako je rezultat polje podataka (array), kao npr. vektor ili matrica ispis će biti formatiran i indeksi prvih vrijednosti u redu će biti ispisani u uglatim zagradama. Primjer:

```
> array(0,20)

[1]  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[15] 0 0 0 0 0 0
```

Oznake [1] i [15] označavaju pozicije prvih elemenata u redu. Tj. prva nula u prvom redu je prvi element polja od 20 nula, dok je prva nula u drugom redu na 15 poziciji. Važno je uočiti da ove oznake nisu dio podataka nego samo pomoć prilikom prikaza podataka.
Primjer matrice:

```
> matrix(0,4,4)

     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    0    0    0    0
```

Ispred svakog stupca i retka je napisan indeks stupca odnosno kolone.

1.9 Ponovno pozivanje i ispravljanje prethodnih naredbi

U mnogim verzijama UNIX-a i u MS Windowsima R pruža mehanizam ponovnog pozivanja i ponovnog izvršavanja prethodno unešenih naredbi. Tipke s okomitim strelicama mogu se upotrijebiti za prolaženje naprijed i natrag kroz kroz *povijest naredbi (history)*. Nakon što se neka naredbi na taj način locirala, kursor se može pomicati unutar naredbe korištenjem tipka s vodoravnim strelicama, te se slova mogu brisati s pomoću DEL tipke ili dodavati s drugim tipkama. Više detalja se navodi kasnije: vidi Dodatak C [Editor komandne linije], str. 90.
Mogućnosti ponovnog pozivanja i editiranja u UNIX-u su jako prilagodljive. Kako se to može učiniti možete pročitati u uvodu priručnika za **readline** knjižnicu.

---

[1] Programeri u C-u trebali bi obratiti pozornost da '_' nije dostupno dok '.' jest, te se često upotrebljava za razdvajanje riječi u R-ovim nazivima.
[2] **ne** unutar nizova niti unutar liste argumenata definicije neke funkcije

Alternativno Emacs text editor daje općenitije mehanizme potpore (putem ESS, *Emacs Speaks Statistics*) za interaktivan rad u R-u. Vidi poglavlje "*R i Emacs"* i *FAQ o R statističkom sustavu*.

1.10 Izvršavanje naredbi putem datoteke ili preusmjeravanje rezultata u datoteku

Ako su naredbe pohranjene u nekoj vanjskoj datoteci, recimo 'commands.R' u radnom direktoriju 'work' može ih se izvršavati u bilo koje vrijeme u radu s R-om pomoću naredbe

```
> source("commands.R")
```

U MS Windowsima naredba **source** je također dostupna u izborniku **File**. Funkcija sink

```
> sink("record.lis")
```

preusmjerit će sve rezultate u datoteku 'record.lis'. 'record.lis' je tekst datoteka i možemo ju editirati. S pomoću naredbe

```
> sink( )
```

ponovno će rezultati biti prikazani na ekranu.

1.11 Trajnost podataka i uklanjanje objekata

Veličine koje R kreira i kojima rukuje poznate su kao *objekti*. To mogu biti varijable, polja brojeva, nizovi znakova (stringovi) , funkcije, ili općenitije strukture izgrađene od takvih komponenata.
Za vrijeme rada s računalom u R-u objekti se kreiraju i pohranjuju po nazivu (taj proces će biti raspravljen u sljedećem poglavlju. R-ova naredba

```
> objects( )
```

(alternativno, ls() se može upotrijebiti za prikaz naziva objekata trenutno postojećih u memorijskom prostoru R-a). Skup objekata trenutno pohranjenih u memorijskom prostoru R-a naziva se radna okolina (*workspace)*.
Za uklanjanje objekata iz memorije na raspolaganju je funkcija rm:

```
> rm (x, y, z, ink, junk, temp, foo, bar)
```

Svi objekti kreirani za vrijeme rada u R-u mogu se stalno pohraniti u datoteku za kasniju upotrebu. Na kraju svakog rada u R-u pruža vam se mogućnost pohranjivanja svih trenutno raspoloživih objekata. Ako naznačite da želite pohraniti objekte, objekti se upisuju u datoteku pod nazivom '.Rdata'[3] u direktoriju.
Kad startate R u nekom kasnijem trenutku, R učitava ovu datoteku i svi objekti iz prijašnjeg rada nalaze se u memoriji. Istovremeno se ponovno napuni s time povezana lista prethodno izvršenih naredbi (history). Kod startanja, R uvjek učitava datoteku '.Rdata' koja se nalazi na putanji (path-u) , tj. onu koju 'vidi'. U izborniku postoje opcije za spremanje i čitanje ove datoteke (koju možete spremiti i pod drugim imenom i naknadno učitati).
Preporuča se korištenje odvojenih radnih direktorija za analize koje se provode putem R-a. Čest je slučaj da se u jednoj analizi kreiraju objekti s nazivima x i y. Takvi nazivi obično imaju jedno značenje u kontekstu jedne analize, a mogu imati drugo značenje u nekoj drugoj analizi, te može doći do konfliktne situacije ako se u istom direktoriju provodi nekoliko analiza.

[3] Početna «točka» u nazivu ove datoteke čini ga *nevidljivim* u normalnim izlistima datoteka na UNIX-u.

2 Jednostavne operacije: brojevi i vektori

2.1 Vektori i pridruživanje

R radi sa strukturama podataka. Najjednostavnija takva struktura je brojčani *vektor*, koji se sastoji od skupa (niza) brojeva. Za kreiranje vektora x , koji se sastoji od recimo pet brojeva, npr. 10.4, 5.6, 3.1, 6.4 i 21.7 (**Decimale se označavaju s točkom(.)**. R pod MS Windows-om ne poštuje lokalne postavke - regional settings) upotrijebite R naredbu

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

Ovo je pridruživanje s pomoću *funkcije* c( ) koja u ovom kontekstu može imati bilo koji broj *argumenata*, i čiji je rezultat vektor čije su komponente navedene kao argumenti funkcije c().[1] Možda je točnije reći da funkcija c() kreira vektor. Pojedinačni brojevi se smatraju vektori duljine jedan.
Obratite pozornost na to da se operator pridruživanja ('<-') sastoji od dva znaka '<' ("manje od") i '-' ("minus") stavljenih neposredno jedan pored drugoga i 'pokazuje' na objekt na kojem se pridružuje vrijednost izraza. U većini situacija može se upotrebiti i znak jednakosti (=).

Pridruživanje se također može izvršiti s pomoću funkcije assign( ). Ekvivalentan način da se učini isto pridruživanje poput gornjega je sljedeći:

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

Uobičajeni operator, <-, može se smatrati kao sintaktički prečac (short-cut) gore navedenomu.
Pridruživanje (assignment) se može također načiniti u drugom smjeru, koristeći očitu promjenu u operatoru pridruživanja. Tako se isto pridruživanje može izvršiti pomoću

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

Napomena: Ovdje je opisano nešto, što mi prihvaćamo intuitivno. Mi govorimo o 'vektoru x' i komponentama vektora x. U stvari, mi objektu koji smo definirali pridružujemo naziv x i pristupamo objektu putem tog naziva.

Ako je neki izraz upotrijebljen kao potpuna naredba vrijednost se prikaže na ekranu *i gubi*[3]. Tako, ako bismo upotrijebili naredbu

```
> 1/x
```

pet recipročnih vrijednosti bi se prikazalo na ekranu i izgubilo (jer nismo pridružili varijablu) a vrijednost x-a ostala bi nepromijenjena.
Pridruživanje

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

kreirao bi vektor y sa 11 elemenata, unosa koji se sastoje od dva x-a s nulom u sredini.

---

[1] S argumentima drukčijim od vektorskog tipa, kao što su argumenti načina lista, akcija c( ) je drukčija. Vidi Poglavlje 6.2.1 [Povezivanje ili nizanje lista], str. 28
[3] Zapravo, ona je još uvijek dostupna kao .Last.value prije nego se prijeđe na bilo koji drugi iskaz

2.2 Aritmetika vektora

Vektori se mogu upotrebljavati u aritmetičkim izrazima, u kojemu slučaju se operacije izvode element po element.
Napomena: Objekt koji se u R-u zove vektor nije isti kao vektor definiran u standarnoj matematičkoj terminologiji.
Vektori u istom izrazu ne moraju svi biti iste duljine. Ako nisu iste duljine, vrijednost izraza je vektor čija je duljina duljina najduljeg vektora u izrazu. Kraći vektori u izrazu se ciklički nadopunjavaju (*recycling)* tako da se produže na duljinu najduljeg vektora (u slučaju da to nije moguće, računska operacija se neće izvršiti. Ovo ponašanje ovisi o verziji R-a koju upotrebljavate). Posebno, konstanta se jednostavno ponavlja. Tako se gore navedenim pridruživanjem pomoću naredbe

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

generira novi vektor v duljine 11 sastavljen dodavanjem, element po element, 2(x cilklički proširen na dužinu od 11 elemenata, y ponovljen samo jedanput, i 1 ponovljen 11 puta.

Osnovni aritmetički operatori su uobičajeni +, -, (, / i ^ za dizanje na potenciju. Pored toga, na raspolaganju su sve uobičajene aritmetičke funkcije. log, exp, sin, cos, tan, sqrt, i tako dalje, svi imaju svoje uobičajeno značenje. max i min biraju najveći odnosno najmanji element vektora. range je funkcija čija je vrijednost vektor duljine dva, naime c(min(x), max(x)). length(x) je broj elemenata u x-u, sum(x) daje zbroj elemenata u x-u, a prod(x) njihov umnožak.
Dvije statističke funkcije su mean(x), koja izračunava srednju vrijednost uzorka, koja je jednaka kao sum (x)/length(x), i var(x) koja daje

```
sum((x-mean(x))^2)/(length(x)-1)
```

ili varijancu uzorka. Ako je argument za var() matrica n-sa-p, vrijednost je matrica p-sa-p kovarijance uzorka dobivena tako što se retci promatraju kao neovisni vektori *p*-variatnog uzorka.
sort(x) vraća vektor iste veličine kao što je x sa elementima poredanim u uzlaznom poretku; međutim na raspolaganju su druge fleksibilnije mogućnosti sortiranja (vidi order( ) ili sort.list ( ) koje daju permutaciju da bi se obavilo sortiranje).
Obratite pozornost da max i min odabiru najveću i najmanju vrijednost u njihovim argumentima čak i ako imaju nekoliko vektora. *Paralelne* funkcije maksimuma i minimuma pmax i pmin vraćaju vektor (čija je duljina jednaka njihovom najduljem argumentu), koji sadrži u svakom elementu najveći(najmanji) element u toj poziciji u bilo kojem od unesenih vektora.
Za većinu upotreba korisniku neće biti stalo jesu li "brojevi" u numeričkom vektoru cijeli brojevi, realni ili kompleksni. Interno se računanja obavljaju kao **realni brojevi dvostruke preciznosti** ili kao kompleksni brojevi dvostruke preciznosti ako su podaci koji se unose kompleksni.

Za rad sa kompleksnim brojevima navedite eksplicitno kompleksni dio. Tako će

```
sqrt(-17)
```

dati NaN (Not A Number) i upozorenje, ali će

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

obaviti računanje kao s kompleksnim brojevima (i – imaginarna jedinica).

2.3 Generiranje regularnih nizova

R ima nekoliko naredbi (možemo reći: operatora ili oznaka) za generiranje nizova brojeva. Jedan od najvažnijih je operator : - dvotočka, koji označava niz brojeva. Na primjer 1:30 je vektor c(1, 2,…, 29, 30). **Operator dvotočka ima najveći prioritet prilikom izvršavanja naredbi unutar nekog izraza**, tako je, na primjer 2∗1:15 vektor c(2, 4, …, 28, 30). Uvrstite n <- 10 i usporedite nizove 1:n-1 i 1:(n-1). Konstrukcija 30:1 se može upotrijebiti za generiranje niza natraške.
Funkcija seq( ) je općenitija funkcija za generiranje nizova. Ima pet argumenata, od kojih samo neki mogu biti specificirati. Prva dva argumenta, ako su dana, specificiraju početak i kraj niza, te ako su oni jedina dva dana argumenta, rezultat je isti kao s operatorom dvotočka. To jest seq(2, 10) je isti vektor kao 2:10.
Parametri u seq( ), i u mnogim drugim funkcijama R-a, mogu također biti dati u obliku s nazivom, u kojem slučaju je redoslijed u kojem se pojavljuju nebitan. Prva dva parametra mogu se nazvati from=*value* i to=*value*; tako su seq(1,30), seq(from=1, to=30) i seq(to=30 from=1) svi isto kao 1:30. Iduća dva parametra u seq( ) mogu biti by=*value* i length=*value,* što specificira razliku (step) između dva element, odnosno duljinu niza. Ako niti jedno od navedenoga nije dano, podrazumijeva se predefinirana vrijednost 1 (by=1).
Na primjer

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

generira u s3 vektor c(-5.0, -4.8, -4.6, …, 4.6, 4.8, 5.0). Slično tomu

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

generira isti vektor u s4.
Peti parametar zove se along=*vektor*, koji, ako se upotrebljava, mora biti jedini parametar, te generira niz 1, 2, …, length(*vektor*), ili prazan niz ako je vektor prazan (što može biti slučaj).
Srodna funkcija je rep( ) koja se može upotrijebiti za repliciranje nekog objekta na razne načine.
Najjednostavniji oblik je

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

što će dati pet x-ova od početka do kraja u s5.

2.4 Logički vektori

Kao kod numeričkih vektora, R omogućava rukovanje logičkim veličinama. Elementi logičkih vektora mogu imati vrijednosti TRUE, FALSE i NA (za "not available" - tj nije definirano, vidi niže u tekstu). Prve dvije vrijednosti se često skraćuju kao T odnosno F. Treba imati na umu, međutim, da su T i F samo varijable čiije su vrijednosti TRUE i FALSE unaprijed definirane, ali nisu posebno namjenjene u tu svrhu, te im vrijednost može biti promijenjena od strane korisnika. Zato treba **uvijek** koristiti TRUE i FALSE.
Logički vektori su generirani *uvjetima*. Na primjer

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

kreira temp kao vektor iste duljine kao što je x, sa FALSE vrijednostima za elemente x-a gdje uvjet *nije ispunjen* i sa TRUE vrijednostima za elemente x-a gdje je uvjet ispunjen.
Logički operatori su <, <=, >, >=, == za točnu jednakost i != za nejednakost. Pored toga, ako su c1 i c2 logički izrazi, tada je c1 & c2 njihov presjek (AND), c1 | c2 njihova unija (OR), a !c1 je negacija c1.

Logički vektori se mogu upotrebljavati u računskim operacijama, u kojem slučaju FALSE postaje 0 a TRUE postaje 1. Postoji i treći slučaj koji ćemo sada opisati.

2.5 Nedostajuće vrijednosti

U nekim slučajevima nije moguće u potpunosti znati komponente nekog vektora. Kad neki element ili vrijednost "nije dostupna" ili u statističkom smislu predstavlja "nedostajuću vrijednost", njegovo mjesto u vektoru može se označiti pripisujući mu specijalnu vrijednost NA. Općenito, svaka operacija koja se obavi sa nekom nedefiniranom vrijednošću NA i sama postaje NA. Očito, nemožemo izračunati izraz koji nije definiran.
Funkcija is.na(x) daje logički vektor iste veličine kao x sa vrijednosti TRUE jedino ako je odgovarajući element u x-u NA.

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

Obratite pozornost da je logički izraz x == NA sasvim različit od is.na(x) jer NA nije prava vrijednost nego oznaka (marker) za neku veličinu koja nije dostupna. Tako je x == NA vektor iste duljine kao x, čije su *sve* vrijednosti NA, i kao logički izraz je nepotpun, te prema tome neodlučiv.
Skrećemo pozornost da postoji jedna druga vrsta "nedostajućih" vrijednosti koje su rezultat računskih operacija, tzv. *Not a Number*, NaN, vrijednosti, tj. vrijednosti koje nisu definirane. Primjeri su sljedeći

```
> 0/0
```

ili

```
> Inf – Inf       (kolokvijalno: beskonačno manje beskonačno)
```

koji oba daju NaN jer rezultat nije definiran.
Ukratko, is.na(xx) je TRUE za oboje, NA i za NaN vrijednosti. Da bismo ih razlikovali, is.nan(xx) je TRUE jedino za NaN vrijednosti.

2.6 Znakovni vektori

U R-u se često upotrebljavaju znakovne veličine i znakovni vektori, na primjer kao oznake na grafikonima. Gdje je to potrebno, označuju se kao niz znakova u dvostrukim navodnicima, npr. "x-values", "New iteration results".
Znakovni nizovi (stringovi) se unose upotrebom dvostrukih (") ili jednostrukih navodnika (') ali se prikazuju korištenjem dvostrukih navodnika (ili ponekad bez navodnika). Ovdje se koristi sintaksa C jezika. Upotrebljava se \ kao znak za escape (ASCII 27) prilikom kreiranja tzv. escape nizova (to je niz znakova koji se prilikom štampanja interpretiraju kao naredbe), tako se \\ upisuje i printa kao \\, a unutar dvostrukih navodnika "se upisuje kao \". Postoji čitav niz tih naredbi kao npr.: \n – newline (nova linija), \t - tab (tabulator), \b – backspace.
Znakovni vektori se mogu "zbrajati" (concatenate) u novi vektor pomoću funkcije c( ); njihova upotreba će biti česta.
Funkcija paste( ) uzima proizvoljan broj argumenata i spaja ih jednog po jednog u niz znakova (string). Sve brojke koje su date među argumentima se spajaju u niz znakova na očigledan način, to jest, onakve kakve bi bile da su printane. Argumenti se po default-u odvajaju u rezultatu praznim mjestom za jedan znak, no to se može promijeniti definiranjem separatora, sep=*string*, koji može biti i prazan string (tj. sep = "").
Na primjer

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

pretvara labs u znakovni vektor

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

Obratite posebno pozornost da je ovdje došlo također do ponavljanja elemenata kratkih lista; tako je c("X", "Y") ponovljeno 5 puta da bi odgovaralo nizu 1:10.[4]
Napomena: Ako nemate iskustva u radu sa R-om, vrlo lako vas rezultat može zbuniti, npr.:

naredba	rezultat
> o <- paste("A","B")	[1] "A B"
> o <- paste("A","B",sep="x")	[1] "AxB"
> o <- paste("A","B","C",sep="x")	[1] "AxBxC"
> o <- paste("A","B","C",1:3,sep="x")	[1] "AxBxCx1" "AxBxCx2" "AxBxCx3"

U zadnjoj naredbi treba voditi računa da je 1:3 vektor sa tri komponente

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Ono što R prvo napravi, pretvori znak "A" u

$$\begin{bmatrix} "A" \\ "A" \\ "A" \end{bmatrix}$$

i "B" u

$$\begin{bmatrix} "B" \\ "B" \\ "B" \end{bmatrix}$$

te onda vrši operaciju spajanja odgovarajućih znakova, te je rezultat vektor sa tri komponente, te imamo tri znakovna niza (stringa) kao rezultat. Uvjek treba voditi računa da R prvo sve vektore proširuje do veličine najvećeg vektora, a zatim izvršava zadane opercije.

2.7 Indeksni vektori; odabiranje i modificiranje podskupova nekog skupa podataka

Podskupovi elemenata nekog vektora mogu se odabrati tako da se nazivu vektora doda *indeksni vektor* stavljen u uglatu zagradu. Općenitije, svaki izraz koji određuje neki vektor može imati podskupove svojih elemenata odabrane na sličan način, dodavanjem indeksnog vektora stavljenog u uglatu zagradu neposredno nakon izraza.
Postoje četiri različite vrste indeksnih vektora.

1. **Logički vektor**. U ovom slučaju indeksni vektor mora imati istu duljinu kao vektor iz kojega se odabiru elementi. Vrijednosti koje su TRUE u indeksnom vektoru se odabiru a vrijednosti za FALSE izostavljaju. Na primjer

[4] paste(…, collapse=*ss*) omogućava ubacivanje argumenata u jedan znakovni niz stavljanjem *ss* između; postoji više procedura za operacije s znakovnim nizovima , vidi help za sub i substring.

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

kreira (ili ponovno kreira) objekt y koji će sadržavati vrijednosti koje su definirane, u istom poretku. Obratite pozornost da će ako x ima nedostajućih vrijednosti, y biti kraći od x-a. Također

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

kreira objekt z i smješta u njega vrijednosti vektora x+1 za koji je odgovarajuća vrijednost u x-u ujedno definirana i pozitivna.

2. **Vektor pozitivnih brojčanih veličina.** U ovom slučaju vrijednosti u indeksnom vektoru moraju biti unutar skupa {1, 2, …, length(x)}. Odgovarajući elementi vektora su odabrani i nanizani, *u tom poretku*, u rezultatu. Indeksni vektor može biti bilo koje duljine, i rezultat ima istu duljinu kao indeksni vektor. Na primjer x[6] je šesta komponenta x-a te

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

odabire prvih 10 elemenata x-a (uz pretpostavku da length(x) nije manja od 10). Također

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

daje znakovni vektor duljine 16 koji se sastoji od "x" , "y" , "y" , "x" ponovljeno četiri puta.

3. **Vektor negativnih brojčanih veličina.**Takav indeksni vektor specificira one vrijednosti koje treba *isključiti*, tj ne uzeti ih. Tako

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

daje y-u sve osim prvih pet elemenata x-a.

4. **Vektor znakovnih nizova**. Ova se mogućnost primjenjuje samo kada neki objekt ima svojsvo (atribut) *naziv* za identifikaciju njegovih komponenata. U tome se slučaju podvektor vektora s nazivima može koristiti na jednak način kao oznake s pozitivnom brojkom pod točkom 2.

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

Ako pogledamo vrijednost od objekta lunch, tj. napišemo

```
> lunch
```

rezultat je

```
apple orange
    1      5
```

Prednost je u tomu što je alfanumeričke *nazive* često lakše pamtiti nego *brojčane oznake*. Ova opcija je posebno korisna kada imamo spremnik podataka (tzv. data frames) kao što ćemo vidjeti kasnije.

Indeksirani izraz se često može pojaviti na početku nekog izraza, u kojem se slučaju odnosi *samo na te elemente vektora*. Izraz mora imati oblik vector[*index_vector*] jer upotreba nekog proizvoljnog izraza umjesto naziva vektora ovdje nema mnogo smisla.

Pridruženi vektor mora odgovarati duljini indeksnog vektora, a kad se radi o logičkom indeksnom vektoru, on mora također imati istu duljinu kao vektor kojega indeksira.

Na primjer

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

zamjenjuje bilo koju nedostajuću vrijednost u x-u nulama tako da

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

ima isti učinak kao

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

2.8 Druge vrste objekata

Vektori su najvažnije vrste objekata u R-u ali postoji nekoliko drugih vrsta s kojima ćemo se susresti u kasnijim poglavljima.

- *matrice* ili općenitije polja (*array)* su višedimenzionalne generalizacije vektora. Zapravo one *jesu* vektori koji se mogu indeksirati s dva ili više indeksa i koji će biti prikazani na posebne načine. Vidi Poglavlje 5 [Polja i matrice], str. 22.
- *faktori* omogućuju kompaktne načine rukovanja kategoričkim podacima. Vidi Poglavlje 4 [Uređeni i neuređeni faktori], str. 19.
- *liste* su općenito oblik vektora u kojemu različiti elementi ne moraju biti iste vrste, i koji su i sami često vektori ili liste. Liste omogućuju pogodan način vraćanja rezultata statističkog izračuna. Vidi Poglavlje 6.1 [Liste], str. 33.
- Spremnici podataka (*data frames)* su matricama slične strukture, u kojima stupci mogu biti različitih tipova. Promatrajte spremnik podataka kao obične matrice s jednim retkom po za svaku veličinu koju promatramo ali sa (po mogućnosti) oboje brojčanim i kategoričkim varijablama (tj. možemo imati varijable i matematičke izraze). Mnogi eksperimenti se najbolje opisuju s spremnicima podataka: obrade su kategoričke ali je rezultat brojčan. Vidi Poglavlje 6.3 [Spremnici podataka (data frames)], str. 35.
- *funkcije* su također objekti u R-u koji se mogu pohraniti u memoriji (workspace-u) projekta. To omogućuje jednostavno i pogodno proširivanje R-a. Vidi Poglavlje 10 [Pisanje vlastitih funkcija], str. 48.

3 Objekti, njihovi tipovi i svojstva

3.1 Suštinska (generička) svojstva: tip i duljina

Veličine sa kojima R-a radi tehnički su poznate kao *objekti*. Primjeri su vektori sa realnim i kompleksnim vrijednostima , vektori logičkih vrijednosti i vektori znakovnih nizova. Oni su poznati kao osnovne (atomarne) strukture jer su njihove komponente sve iste vrste, ili istoga tipa , to jest *numeričke*[1]*, kompleksne, logičke* odnosno *znakovne.*

Vrijednosti vektora moraju biti *istoga tipa*. Tako svaki dani vektor mora nedvosmisleno biti *logički, numerički, složeni* ili *znakovni*. Jedini mali izuzetak od ovog pravila je specijalna "vrijednost" označena kao NA za nepostojeće vrijednosti. Obratite pozornost da vektor može biti prazan a da ipak ima definiran tip. Na primjer prazan vektor znakovnog niza se označava kao *character(0),* a prazan numerički vektor kao *numeric(0).*

R također radi s objektima nazvanim *liste*, koji su tipa *list*. To su poredani nizovi objekata koji pojedinačno mogu biti bilo kojega tipa. Za *liste* možemo reći da su "rekurzivne" radije nego da su osnovne strukture jer njihove komponente mogu također biti liste.

Druge rekurzivne strukture su strukture tipa *funkcija* i *izraza (expression)*. **Funkcije su objekti** koji čine dio R-a zajedno sa sličnim funkcijama koje piše korisnik, o kojima ćemo kasnije detaljnije raspravljati. Izrazi kao objekti čine napredan dio R-a o kojemu nećemo

[1] *numeric podrazumjeva dva tipa*, tj. *integer* i *double* precision, kako je objašnjeno u priručniku.

raspravljati u ovim uputama osim indirektno kada raspravljamo o *formulama* koje se koriste kod modeliranja u R-u.
Pod tipom (modom) nekog objekta podrazumijevamo osnovni tip njegovih temeljnih sastavnica. To je poseban slučaj "svojstva" nekoga objekta. Drugo svojstvo svakog objekta je njegova *duljina*. Funkcije mode(*objekt*) i length(*objekt*) mogu se upotrijebiti za pronalaženje tipa i duljine bilo koje definirane strukture[2].
Daljnja svojstva nekog objekta obično se dobiju s pomoću attributes(*object*), vidi Poglavlje 3.3 [Mjenjanje svojstava objekta], str. 19. Zbog toga se *tip* i *duljina* također nazivaju "intrinsic attributes" nekog objekta.
Na primjer, ako je z kompleksni vektor duljine 100, tada će nam funkcija mode(z) vratiti vrijednost "complex" a length(z) je 100.
R nam daje mogućnosti promjena tipa objekta gotovo svugdje gdje se to može smatrati razumnim (a ponekad i gdje nije razumno).
Na primjer , ako definiramo

```
> z <- 0:9
```

mogli bismo staviti

```
> digits <- as.character(z)
```

nakon čega digits postaje znakovni vektor c("0", "1", "2", … , "9").
Naredba

```
> d <- as.integer(digits)
```

pretvara d u numerički vektor.
Sada su d i z isti[3]. Postoji velika zbirka funkcija oblika as.*something*( ) za pretvaranje iz jednog tipa u drugi, ili za dodavanje nekog novog svojstva nekom objektu. Čitatelj može konzultirati različite help file-ove da se s njima upozna.

3.2 Mijenjanje duljine objekta

"prazan" objekt može imati definirani tip. Naredbom

```
> e <- numeric( )
```

kreiramo vektor e kao numerički tip . Slično tomu character( ) je prazan znakovni vektor, itd. Nakon što je kreiran neki objekt bilo koje veličine, mogu mu se dodavati nove komponente jednostavno dajući mu neku indeksnu vrijednost izvan njegovog prethodnog raspona. Tako u

```
> e[3] <- 17
```

e postaje vektor duljine 3, (čije su prve dvije komponente u ovom trenutku obje NA). To se odnosi na bilo koju strukturu uopće, pod uvjetom da je tip dodane komponente (dodanih komponenata) istog tipa kao i objekt.
Ovo automatsko prilagođavanje duljina nekog objekta se često upotrebljava, na primjer u funkciji scan( ) za unos. (Vidi Poglavlje 7.2 [Funkcija scan( )], str. 39.)
Obrnuto, za skračivanje veličine nekog objekta potrebno je samo jedno pridruživanje da se to učini. Tako, ako je alpha neki objekt duljine 10, tada u

[2] Obratite pozornost, međutim, da length(*object*) ne sadrži uvijek suštinski korisnu informaciju, npr. kada je *objekt* funkcija.
[3] Općenito, konverzija (eng. coercion) iz numeričkog tipa u znakovni i ponovno natrag u numerički neće biti točno reverzibilna zbog zaokruživnja u znakovnom prikazu.

```
> alpha <- alpha[2 * 1:5]
```

alpha postaje objekt duljine 5 koji se sastoji od samo bivših komponenata s parnim indeksom. Stari indeksi, naravno, nisu zadržani.

3.3 Mjenjanje svojstava objekta

Funkcija atributes(*object*) daje listu svih nesuštinskih svojstava trenutno definiranih za taj objekt. Funkcija attr(*object, name*) može se upotrijebiti za izbor određenog svojstva. Ove se funkcije rijetko upotrebljavaju, osim u posebnim okolnostima kad se kreira neko novo svojstvo u određenu svrhu, na primjer za povezivanje datuma kreiranja ili nekog operatora s nekim R-ovim objektom. Sama zamisao je, međutim, vrlo značajna.
Potrebno je pažljivo postupati prilikom dodavanja ili brisanja svojstva jer su svojstva sastavni dio sustava objekata koji se koriste u R-u.
Kad se svojstvo stavlja s lijeve strane nekog pridruživanja, može ga se upotrijebiti za pridruživanje novog atributa *objektu* ili za promjenu nekog postojećeg svojstva. Na primjer:

```
> attr(z,"dim") <- c(10,10)
```

R tretira z poput matrice 10 sa10.

3.4 Klasa objekta

Svi objekti u R-u pripadaju nekoj klasi. Funkcija class će nam za svaki objekt ispisati kojoj klasi pripada. Za obične vektore to je upravo tip vektora: "numeric", "logical", character", ali "matrix", "list", "factor" i "data.frame" su isto moguće vrijednosti.
Posebno svojstvo, poznato kao *klasa* objekta, upotrebljava se da bi se omogućilo objektno orijentiran stil (object oriented style) programiranja u R-u.
Na primjer ako neki objekt ima klasu "data.frame", bit će prikazan na određen način, funkcija plot( ) će ga pokazati grafički na određen način, a druge, tzv. generičke funkcije kao što summary( ) reagirat će na njega kao argument na način koji je osjetljiv na njegovu klasu.
Za privremeno uklanjanje efekta klase, upotrijebite funkciju unclass( ). Na primjer ako velićina *winter* ima klasu "data.frame" tada ako napišeno

```
> winter
```

prikazati u data frame obliku, koji je više poput matrice, dok će ga

```
> unclass(winter)
```

prikazati kao običnu listu. Ovu ćete mogućnost trebati samo u dosta posebnim situacijama. Jedna je od takvih situacija upoznavanje s pojmom klase i generičkih funkcija.
O generičkim funkcijama i klasama raspravljat ćemo kasnije u Poglavlju 10.9 [Klase, generičke funkcije i objektna orijentacija], str. 56, no samo u kratkim crtama.

4 Uređeni i neuređeni faktori

Faktor je vektorski objekt koji se upotrijebljava za grupiranje komponenata vektora iste duljine. R poznaje *uređene* i *neuređene* faktore. Dok stvarnu primjenu faktora daju formule u modelima (vidi Poglavlje 11.1.1 [Kontrasti], str. 59).

4.1 Primjer

Uzmimo, na primjer, da imamo uzorak od 30 obračunavatelja poreza iz svih država i terirorija Australije[1] i da je pojedinačna država iz koje potječu označena svojom znakovnom skračenicom :

```
> state <- c("tas", "sa",  "qld",  "nsw", "nsw", "nt",   "wa",  "wa",
             "qld", "vic", "nsw", "vic", "qld", "qld",  "sa",  "tas",
             "sa",  "nt",  "wa",  "vic", "qld", "nsw",  "nsw", "wa",
             "sa",  "act", "nsw", "vic", "vic", "act")
```

Obratite pozornost da u slučaju znakovnog vektora, "uređen" znači svrstan abecednim redom.
Faktor se slično kreira koristeći funkciju factor( ):

```
> statef <- factor(state)
```

Funkcija print( ) rukuje faktorima malo drukčije nego drugim objektima:

```
> statef
 [1] tas sa  qld nsw nsw nt  wa  wa  qld vic nsw vic qld qld sa
[16] tas sa  nt  wa  vic qld nsw nsw wa  sa  act nsw vic vic act
Levels: act nsw nt qld sa tas vic wa
```

Da bismo našli nivoe nekog faktora može se koristiti funkcija levels( ).

```
> levels(statef)
[1] "act" "nsw" "nt" "qld" "sa" "tas" "vic" "wa"
```

4.2 Funkcija tapply( ) i nepravilna polja

Kao nastavak prethodnog primjera, pretpostavimo da imamo prihode istih obračunavatelja poreza u drugom vektoru (u odgovarajuće velikim jedinicama novca)

```
> incomes <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56,
               61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46,
               59, 46, 58, 43)
```

Da bismo izračunali srednju vrijednost prihoda uzorka za svaku državu možemo upotrijebiti posebnu funkciju tapply( ):

```
> incmeans <- tapply(incomes, statef, mean)
```

što daje vektor srednjih vrijednosti s komponentama nazvanim po nivoima

act	nsw	nt	qld	sa	tas	vic	wa
44.500	57.333	55.500	53.600	55.000	60.500	56.000	52.250

[1] Strani čitatelji trebaju znati da Australija ima osam država odnosno teritorija, to jest Teritorij glavnog grada Australije, države New South Wales, the Northern Territory, Queensland, South Australia, Tasmania, Victoria i Western Australia.

Funkcija tapply( ) se upotrebljava kako bi se primijenila neka funkcija, u ovom slučaju mean() – koja računa srednju vrijednost, na svaku grupu komponenata prvog argumenta, u ovom slučaju incomes, definiranu nivoima druge komponente, u ovom slučaju statef[2] kao da su to dvije odvojene vektorske strukture. Rezultat je struktura duljine jednake duljini koju ima atribut nivoa faktora koji sadrži rezultate. Za više pojedinosti čitatelj bi trebao konzultirati R-ovu dokumentaciju.
Pretpostavimo nadalje da želimo izračunati standardnu pogrešku srednje vrijednosti prihoda država. U tu svrhu trebamo napisati R-ovu funkciju za izračun standardne pogreške za bilo koji dani vektor. Budući da postoji ugrađena funkcija var() za izračunavanje varijancije uzorka, takva je funkcija vrlo jednostavna jednolinearna funkcija, specificirana pridruživanjem:

```
> stderr <- funkcija(x) sqrt(var(x)/length(x))
```

(O pisanju funkcija bit će riječi kasnije u Poglavlju 10 [Pisanje vlastitih funkcija], str. 48). Nakon ovog pridruživanja, standardne pogreške se mogu izračunati s pomoću

```
> incster <- tapply(incomes, statef, stderr)
```

a dobivene izračunate vrijednosti su

```
> incster
```

act	nsw	nt	qld	sa	tas	vic	wa
1.5	4.3102	4.5	4.1061	2.7386	0.5	5.244	2.6575

Za vježbu mogli biste naći uobičajene 95% granice pouzdanosti za srednju vrijednost prihoda država. Da biste to učinili možete upotrijebiti tapply( ) još jednom s funkcijom length() da nađete veličine uzorka te s funkcijom qt() da nađete točke postotaka odgovarajućih *t*-distribucija.
Funkciju tapply() može se upotrijebiti i za rukovanje složenijim indeksiranjem nekog vektora po više kriterija, tj. klasa. Na primjer, mogli bismo razdvojiti obračunavatelje poreza ujedno po državama i po spolu, tj prema dva svojstva. Međutim, u ovom jednostavom slučaju (radi se o samo jednoj kategoriji) će se dogoditi sljedeće. Vrijednosti u vektoru su grupirane prema zadanim kriterijima. Funkcija se zatim primjenjuje na svaku od tih grupa pojedinačno. Vrijednost je vektor, indeksiran prema svojstvima.
Ova kombinacija vektora i više kriterija primjer je onoga što se ponekad naziva *nepravilno polje* s obzirom da veličine podklasa mogu biti različite. Kada su veličine podklasa sve jednake, indeksiranje se može napraviti implicitno i mnogo djelotvornije, kao što ćemo vidjeti u narednom poglavlju.

4.3 Uređeni faktori

Nivoi faktora se pohranjuju abecednim redom ili po redu kojim su faktori bili specificirani, ako su specificirani eksplicitno.
Nekada će nivoi imati prirodni poredak kojeg želimo zabilježiti i kojega želimo da naša statistička analiza koristi. Funkcija ordered() kreira tako poredane faktore ali je u drugome jednaka faktoru. Najčešće je jedina razlika između poredanih i neporedanih faktora u tome što se poredani faktori prikazuju pokazujući poredak nivoa, ali su kontrasti generirani za njih metodom prilagođavanja (fitting) na linearne modele različiti.

[2] Obratite pozornost da je tapply( ) također primjenjiva u ovom slučaju kada drugi argument nije faktor, npr. 'tapply(incomes, state)', a to vrijedi za samo nekoliko drugih funkcija jer se argumenti spajaju u faktore kad je to potrebno (korištenjem as.factor( )).

5 Polja i matrice

5.1 Polja

Polje je skup podataka s višestrukim indeksima, npr. polje realnih ili cijelih brojeva, skup logičkih vrijednosti i sl. R ima jednostavne procedure za kreiranje i rukovanje s poljima, a poseban slučaj su matrice (dvodimenzionalno polje sa posebnim svojstvima). Riječi *dimenzija* ne treba davati nikakvo duboko značenje, ona ovdje samo opisuje broj indeksa skupa podataka. Npr.

b_1, b_2, b_3

je jednodimenzionalno polje sa tri elementa, dok je

$b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,1}$ $b_{3,2}$ $b_{3,3}$

dvodimenzionalno polje s 9 elemenata. Uočimo da u ovom slučaju prvi indeks predstavlja redak u kojem s nalazi element, dok drugi indeks predstavlja stupac. Naravno, mogli smo sve elemente napisati u jednom redu (linearno), ali bi tada izgubili jedno svojstvo, koje se upravo očituje u tome da smo podatke grupirali po stupcima (odnosno retcima),

Svakom polju pridružen je jedan vektor s pozitivnim vrijednostima koji opisuje polje – vektor dimenzije polja. Ako je polje k-dimenzionalno onda je duljina vektora *k*. Vrijednosti u dimenzijskom vektoru su gornje granice svakog indeksa. Donje granice su uvijek 1 (tj. prvi element ima indeks 1 – ovo je važno napomenuti jer u matematici počinjemo brojati od broja 1, dok u informatici (programiranje) obično počinjemo s 0).

Vektor se može promatrati u R-u kao polje jedino ako ima definiran dimenzijski vektor - *dim* kao svoje svojstvo. Pretpostavimo, na primjer, da vektor z ima 1500 elemenata. Pridruživanjem

```
> dim(z) <- c(3,5,100)
```

definira svojstvo *dim,* koje nam kaže da je to polje 3 x 5 x 100. Inaće bi ga promatrali kao 1-dimenzionalno polje. Ovim pridruživanjem možemo promijeniti dimenziju polja, npr.:

```
> dim(z) <- c(2,750)
```

je polje 2x750. Kako ćemo definirati veličinu polja zavisi o prirodi podataka koje obrađujemo.

Druge funkcije kao matrix() i array() dostupne su za jednostavnije manipuliranje sa podacima, kako što ćemo vidjeti u Poglavlju 5.4 [Funkcija array( )], str. 25.

Vrijednosti u vektoru s podacima u istom su poretku kao što se radi u FORTRANU, to jest u "column major order", tj. nanizani su po stupcima.
Na primjer ako je dimenzijski vektor za neko polje a: c(3,4,2), tada u a ima $3*4*2=24$ elementa i vektor podaka ih drži u poretku a[1,1,1], a[2,1,1],…, a[2,4,2], a[3,4,2]. Ovdje se želi reći da ako vektor želimo prikazati u linearnom poretku (kao što je i interno zapisan) u memoriji računala) indeksiranje ide prvo po prvom indeksu, zatim drugom i tako dalje. U slučaju gornjeg polja to znači: $b_{1,1}$, $b_{2,1}$, $b_{3,1}$, $b_{1,2}$, $b_{2,2}$, $b_{3,2}$, $b_{1,3}$, $b_{2,3}$, $b_{3,3}$.

5.2 Indeksiranje polja. Potskup polja

Pojedinačni elementi nekog polja mogu se referencirati kako je gore prikazano, dajući naziv polju nakon kojega slijede indeksi u uglatim zagradama, odijeljeni zarezima.
Općenito, sintaksa je slijedeća: ako indeks nije specificiran onda se uzimaju sve vrijednosti tog indeksa, tako a[,,] označava cijelo polje.
U prethodnom primjeru (sa c(3,4,2)), a[2,,] predstavlja polje $4*2$ s dimenzijskim vektorom c(4,2) i vektorom podataka koji sadržava vrijednosti

c(a[2,1,1], a[2,2,1], a[2,3,1], a[2,4,1], a[2,1,2], a[2,2,2], a[2,3,2], a[2,4,2])

Za bilo koje polje, recimo Z, dimenzijski vektor se može eksplicitno referencirati kao dim(Z) (na bilo kojem kraju pridruživanja).

Također, ako je neko polje zadano sa *samo jednim indeksom ili indeksnim vektorom*, tada se upotrebljavaju jedino odgovarajuće vrijednosti podataka; u tom se slučaju dimenzijski vektor zanemaruje. To, međutim, nije slučaj ako jedini indeks nije vektor nego je on sam polje, o čemu se raspravlja u daljnjem tekstu.

5.3 Indeksna polja

Osim indeksnog vektora možemo definirati i indeksno polje kojim možemo definirati vektor s vrijednostima u zadanom polju i koje definira način odabira elemenata.
Primjer matrice pojašnjava taj proces. U slučaju dvostruko indeksiranog polja može biti dana indeksna matrica koja se sastoji od dva stupca i onoliko redaka koliko se želi. Vrijednosti u indeksnoj matrici su indeksi redaka i stupaca dvodimenzionalnog polja.
Pretpostavimo na primjer da imamo polje X dimenzije 4 x 5 i da želimo učiniti sljedeće:

- Ekstrahirati elemente X[1,3], X[2,2] i X[3,1] kao vektorsku strukturu i
- Zamijeniti te elemente polja X sa 0 (nula).

U tom slučaju trebamo polje koje ima 3 x 2 indeksa, kao u sljedećem primjeru.

```
> x <- array(1:20,dim=c(4,5))      # Kreiramo polje  4 x 5.
```

Ako napišemo:

```
> x
```

dobijemo prikaz polja:

```
     [,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
```

Definiramo indeksno polje;

```
> i <- array(c(1:3,3:1),dim=c(3,2))    # Kreiramo indeksno polje 3 x 2.
```

Ako napišemo:

```
> i
```

dobijemo prikaz indeksnog polja:

```
     [,1] [,2]
[,1]    1    3
[,2]    2    2
[,3]    3    1
```

Ako napišemo:

```
> x[i]
```

dobijemo vektor koji je definiran sa indeksnim poljem i

```
[1] 9 6 3
```

Sada, zamjenjujemo elemente definirane sa i sa nulom:

```
> x[i] <- 0
```

Ako sada pogledamo kako izgleda originalno polje

```
> x
```

vidimo da su elementi definirani sa i zamjenjeni s nulama:

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
```

Kao manje trivijalan primjer, pretpostavimo da želimo načiniti nereduciranu dizajn matricu za blok-dizajn definiran faktorima blocks (b levels) i varieties (v levels). Pretpostavimo nadalje da u eksperimentu ima n grafova (tj. skupova podataka). Možemo postupiti kako slijedi:

```
> Xb <- matrix(0, n, b)
> Xv <- matrix(0, n, b)
> ib  <- cbind(1:n, blocks)
> iv  <- cbind(1:n, varieties)
> Xb[ib] <- 1
> Xv[iv] <- 1
> X <- cbind(Xb, Xv)
```

Za konstruiranje matrice incidencije, recimo N, možemo upotrijebiti

```
> N <- crossprod(Xb, Xv)
```

Međutim, jednostavniji direktan način kreiranja ove matrice je upotrebom table( ):

```
> N <- table(blocks, varieties)
```

(Ovaj primjer koristi se pojmovima iz kombinatorijalne teorije, za detalje možete pogledati:
1. D. Veljan: Kombinatorika s teorijom grafova, Školska knjiga, Zagreb, 1989.
2. M. Aigner: Combinatorial Theory, Springer-Verlag, Berlin, 1979.
3. P.J. Cameron, J.H. Van Lint: Graphs, Codes and Designs, Cambridge University Press, Cambridge, 1980.)

5.4 Funkcija array( )

Jednako tako kao što možemo vektorskoj strukturi definirati svojstvo *dim*, polje možemo konstruirati iz vektora s pomoću funkcije *array*, koja ima oblik

```
> Z <- array(data_vector, dim_vector)
```

Na primjer, ako vektor h sadrži 24 brojeva, ili manje, tada bi naredba

```
> Z <- array(h, dim=c(3,4,2))
```

upotrijebila h za konstruiranje polja 3 x 4 x 2 u Z-u. Ako je veličina h točno 24, rezultat je isti kao

```
> dim(Z) <- c(3,4,2)
```

Ako je, međutim, h ima manje od 24 elemenata, njegove se vrijednosti ciklički ponavljaju od početka dok se ne dostigne veličina 24 (vidi Poglavlje 5.4.1 [Mješovita aritmetika s vektorima i poljima. Pravilo cikličkog nadopunjavanja], str. 25). Kao ekstreman ali česti primjer je

```
> Z <- array(0, c(3,4,2))
```

Z postaje polje koji se sastoji od samih nula.
Ovdje dim(Z) predstavlja dimenzijski vektor c(3,4,2) dok Z[1:24] predstavlja vektor podataka kao što je bilo u h, a Z[] ili Z označava cijelo polje..
Polja se mogu upotrebljavati u aritmetičkim izrazima i rezultat je polje nastalo operacijama s po odgovarajućim elementima: **element po element**. *dim* svojstvo operanada, općenito mora biti isto, i rezultat ima istu dimenziju. Tako ako su A, B i C svi slična polja (znači da se mogu izvršiti opercije po elementima i da su dim(A) = dim(B) = dim(C)), tada

```
> D <- 2*A*B + C + 1
```

daje polje D čija je dimenzija ista kao i polja s kojima smo ušli u aritmetičke operacije, a vrijednosti su:

```
D[i,,j] = 2*A[i,,j]*B[i,,j]+C[i,,j]+1
```

za sve indekse i,j. **Važno je uočiti da su pravila za ove operacije različita od računskih operacija s matricama**. Međutim, točno pravilo u pogledu računanja s poljima i vektorima različitih dimenzija treba pažljivije razmotriti.

5.4.1 Mješovita aritmetika s vektorima i poljima. Pravilo cikličkog nadopunjavanja

Računanje element po element za računanja s vektorima i poljima ponekad može biti pomalo kompliciranо. Iz iskustva smo našli sljedeće kao pouzdan vodič.

- Izraz se kontrolira s lijeva na desno.
- Svi kraći vektorski operandi se produljuju cikličkim nadopunjavanjem njihovih vrijednosti dok ne odgovaraju veličini drugih operanada.
- Dok se god susreću kraći vektori i polja, sva polja moraju imati isto *dim* svojstvo, ili će doći do pogreške.
- Bilo koji operand vektora dulji od operanda matrice ili polja dati će pogrešku.
- Ako su u operacijama prisutna polja i ako nije nastala pogreška koja se zapisala u vektor, rezultat je struktura polja sa zajedničkim dim atributom njezinih operanada.

5.5 Vanjski produkt dvaju polja

Važna operacija s poljima je *vanjski produkt*. Ako su a i b dva numerička polja, njihov je vanjski produkt je polje čiji se dimenzijski vektor dobije povezivanjem (concatenating) dimenzijskih vektora polja a i polja b (važan je poredak), a čiji se elementi dobiju formiranjem svih mogućih produkata elemenata od a s elementima od b. Vanjski produkt se označava s %o%:

```
> ab <- a %o% b
```

ili

```
> ab <- outer(a, b, "*")
```

U slučaju običnog množenja, oznaku * možemo izostaviti.
Funkcija množenja može se zamijeniti nekom proizvoljnom funkcijom dvije varijable. Na primjer ako želimo izračunati vrijednosti funkcije $f(x;y) = \cos(y)/(1 + x^2)$ na skupu svih regularnih (tj. da vrijednosti nisu NaN ili NA) vrijednosti parova (x,y) gdje x i y poprimaju sve vrijednosti iz skupa R-ovim vektorima x odnosno y, možemo postupiti na slijedeći način:
Definiramo funkciju f:

```
> f <- function(x, y) cos(y)/(1 + xˆ2)
```

i primjenimo funkciju:

```
> z <- outer(x, y, f)
```

Posebno, vanjski produkt dva običnih vektora je dvodimenzionalno polje (to jest matrica ranga najmanje 1). Obratite pozornost da operator vanjskog produkta nije komutativan. O definiranju vlastitih R funkcija bit će riječi kasnije u Poglavlju 10 [Pisanje vlastitih funkcija], str. 48.

Primjer: Determinante matrica 2 x 2 sa elementima u skupu 0,1,...,9

Kao zgodan primjer, uzmimo determinante matrica 2 x 2 [*a,b*; *c,d*] gdje je svaki element ne-negativni cijeli broj iz skupa {0,1,…,9}.
Želimo naći determinante, *ad* – *bc*, svih mogućih matrica ovoga oblika i grafički prikazati frekvencije s kojom se svaka vrijednost pojavljuje. To je slično kao nalaženje raspodjele vjerojatnosti determinanta ako se svaki broj bira neovisno i uniformno nasumce.
Ovo se to može izvršiti korištenjem funkcije outer( ) dvaput:

```
> d <- outer(0:9, 0:9)
```

d sadrži **sve** produkte xy , gdje su x, y ε {0,1,...9}.

```
> fr <- table(outer(d, d, "-"))
```

fr sadrži sve elemente oblik xy – zt, gdje su x, y, z, t ε {0,1,...9}. Sada možemo nacrtati graf:

```
> plot(as.numeric(names(fr)), fr, type="h", xlab="Determinant", ylab="Frequency")
```

Obratite pozornost na pretvaranje names atributa tablice učestalosti u numerički atribut kako bi se ponovno dobio raspon determinantnih vrijednosti. Očigledan način tretiranja ovog problema sa *for* petljama, o kojemu će se raspravljati u Poglavlju 9 [Grupiranje, petlje i uvjetno izvršavanje], str. 47, vrlo je neučinkovit i nepraktičan.
Također je, možda iznenađujuće, otprilike 1 od 20 takvih matrica singularna.

5.6 Proširenje operacije transponiranja na polja

Funkcija aperm(a, perm) vrši permutaciju elemenata polja a permutirajući indekse. Argument *perm* mora biti permutacija brojeva {1,…,*k*}, gdje je *k* broj indeksa od a. Rezultat funkcije je polje iste veličine kao a, s time da je stara dimenzija putem perm[j] postala nova j-th dimenzija. Najlakši način da se zamisli ova operacija je primjena na matricama. Ako je A matrica, (to jest, polje s dvostrukim indeksom), onda je B dobiven s pomoću

```
> B <- aperm(A, c(2,1))
```

transponirana matrica od A. Za matrice postoji funkcija t( ) koja vrši transponiranje, tako smo mogli upotrijebiti B <- t(A).
Jedan primjer (vidjeti u R-u ako upišete ?aperm):

```
> x  <- array(1:24, 2:4)
```

x je polje sa elementima od 1 do 24. Prema defenciji, 2:4 je 2,3,4, te je x trodimenzionalno polje. Tj. prvi indeks ima vrijednosti 1,2, drugi indeks ima vrijednosti 1,2,3, dok treći indeks ima vrijednosti 1,2,3,4. Ako u R-u ispišemo x na ekranu dobijemo:

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

Ovo prikazivanje 3 dim. polja je na prvi pogled malo neobično, ali to je stvar navike. Ako pogledamo što prikazuje prva grupa podataka, onda vidimo da je treći indeks 1. prvi indeks ide od 1 do 2, dok drugi indeks od 1 do 3. Tj.

$a_{1,1,1} = 1$ $a_{1,2,1} = 3$ $a_{1,3,1} = 5$

$a_{2,1,1} = 2$ $a_{2,2,1} = 4$ $a_{2,3,1} = 6$

I tako dalje, po trećem indeksu koji ide do 4. Ovdje se treba prisjetiti predhodne napomene, da funkcija *array* popunjava polje prema "column major order".

Ako sada primjenimo:

```
> xt <- aperm(x, c(2,1,3))
```

dobijemo polje u kojem su prva dva indeksa zamjenila mjesto, Ako prikažemo polje xt, dobijemo:

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

, , 2
     [,1] [,2]
[1,]    7    8
[2,]    9   10
[3,]   11   12

, , 3
     [,1] [,2]
[1,]   13   14
[2,]   15   16
[3,]   17   18

, , 4
     [,1] [,2]
[1,]   19   20
[2,]   21   22
[3,]   23   24
```

Vidimo, da prvi indeks ide od 1 do 3, drugi od 1 do 2, dok je treći indeks ostao isti.

5.7 Matrice

Do sada smo pod riječju matrica podrazumjevali 2 dim. polje, tj. polje sa 2 indeksa. Međutim, matrice su tako važan poseban slučaj da iziskuju posebnu raspravu. R posjeduje mnogo operatora i funkcija samo za matrice. Naime, u matematici je matrica dobro definirana struktura (pogledati neki udžbenik iz linearne algebre). Najednostavnije, možemo reći da su matrice 2 dimenzionalna polja (array) realnih ili kompleksnih brojeva sa posebno definiranim matematičkim operacijama. Na primjer t(X) je funkcija za transponiranje matrice, kako je prije spomenuto. Funkcije nrow(A) i ncol(A) daju broj redaka odnosno broj stupaca u matrici A.

5.7.1 Množenje matrica

Za množenje matrica koristi se operator %∗%. Matrica n x 1 ili 1 x n može se, naravno, upotrijebiti kao *n*-vektor ako je u kontekstu takav vektor odgovarajući. Obratno, vektori koji se nađu u izrazima za množenje matrica automatski se, ako je to moguće, promatraju kao vektor - redak ili vektor- stupac.
Ako su, na primjer, A i B kvadratne matrice iste veličine, tada je

```
> A*B
```

matrica produkata element po element , a

```
> A %*% B
```

je produkt matrica. Ako je x vektor, tada je

```
> x %*% A %*% x
```

kvadratna forma (vidi: udžbenik iz linearne algebre).[1]
Funkcija crossprod() formira "unakrsne produkte", što znači da je crossprod(X, y) isto kao t(X) %*% y ali je operacija mnogo djelotvornija. Ako je drugi argument u crossprod( ) izostavljen, uzima se da je isti kao prvi.
Značenje diag() ovisi o njegovom argumentu. diag(v), gdje je v vektor, daje dijagonalnu matricu s elementima vektora kao dijagonalnim elementima. S druge strane diag(M), gdje je M matrica, daje vektor s glavnim dijagonalnim elementima u M-u. To je ista konvencija kao diag() u MATLAB-u. Također, pomalo zbunjujuće, ako je k jedna brojčana vrijednost, tada je diag(k) jedinična matrica dimenzije k x k.

5.7.2 Linearne jednadžbe i inverzija

Kod rješavanja sustava linearnih jednadžbi javlja se problem obrnute računske operacije od množenja matrica. Kada su nakon

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

dati samo A i b, vektor x je rješenje toga sustava linearnih jednadžbi. U R-u

```
> solve(A,b)
```

rješava sustav, vraćajući x (uz određeni gubitak u točnosti). Obratite pozornost da je u linearnoj algebri, formalno $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, gdje $\mathbf{A}^{-1}$ označava *inverznu matricu* od **A**, koje se može izračunati s pomoću

```
> solve(A)
```

ali je to u rijetkim slučajevima potrebno. Numerički je neefikasno i nestabilno računati x <- solve(A) %*% b umjesto solve(A,b).
Kvadratičnu formu $\mathbf{x}'\mathbf{A}^{-1}\mathbf{x}$ koja se upotrebljava u računanjima s više slučajnih varijabla , trebalo bi se računati s nečim poput x %*% solve(A,x) radije nego računanjem inverzne matrice od A.
Za podsjetnik, jedan elementarni primjer: Imamo sustav linearnih jednadžbi sa 4 nepoznanice:

$$\begin{aligned} 10x + 18y + 9z + 8t &= -5 \\ 4x + 8y + 4z + 3t &= -2 \\ 5x + 12y + 7z + 4t &= 1 \\ x + 3y + 2z + t &= 4 \end{aligned}$$

[1] Obratite pozornost da je x %*% x dvosmisleno, jer može značiti bilo $\mathbf{x}'\mathbf{x}$ ili $\mathbf{x}\mathbf{x}'$, gdje **x** ima oblik stupca. U takvim slučajevima manja matrica je implicitno prihvaćena interpretacija, tako da je u ovom slučaju rezultat skalar $\mathbf{x}'\mathbf{x}$. Matrica $\mathbf{x}\mathbf{x}'$ može se izračunati bilo s pomoću cbind(x) %*% x ili s pomoću x %*% rbind(x), jer je rezultat rbind( ) ili cbind( ) uvijek matrica.

Ako ovaj sustav napišemo u vektorskom obliku dobijemo

$$\mathbf{A}\%*\%\mathbf{x} = \mathbf{b}$$

gdje je

$$\mathbf{A} = \begin{bmatrix} 10 & 18 & 9 & 8 \\ 4 & 8 & 4 & 3 \\ 5 & 12 & 7 & 4 \\ 1 & 3 & 2 & 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -5 \\ -2 \\ 1 \\ 4 \end{bmatrix}$$

Idemo sada riješiti ovaj sustav jednadžbi:

1. Definiramo matricu **A**:

```
> A <- c(10,4,5,1,18,8,12,3,9,4,7,2,8,3,4,1)
```

Obratite pažnju kojim redosljedom su napisani brojevi ("column major order").

2. Definiramo dimenziju matrice:

```
> dim(A) <- c(4,4)
```

3. Definiramo matricu **b**:

```
> b <- c(-5,-2,1,4)
```

4. Definiramo dimenziju:

```
> dim(b) = c(4,1)
```

5. Izračunamo **x**:

```
> x <- solve(A,b)
```

Ako prikažemo x dobijemo rješenja gornjeg sustava:

```
     [,1]
[1,] -28
[2,]  12
[3,] -13
[4,]  22
```

tj. x = -28, y = 12, z = -13, t = 22.

5.7.3 Svojstvene vrijednosti i svojstveni vektori

Funkcija eigen(Sm) računa svojstvene vrijednosti i svojstvene vektore matrice Sm. Rezultat ove funkcije je lista s dvije komponente koje sadrže svojstvene vrijednosti (values) i svojstvene vektore (vectors). Pridruživanje

```
> ev <- eigen(Sm)
```

će pridružiti ovu listu varijabli ev. Tada je ev$val vektor svojstvenih vrijednosti Sm-a, a ev$vec je matrica odgovarajućih svojstvenih vektora. Da smo samo trebali svojstvene vrijednosti mogli smo upotrijebiti direktno pridruživanje:

```
> evals <- eigen(Sm)$values
```

evals sada sadrži vektor svojstvenih vrijednosti dok je druga komponenta odbačena. Ako napišemo

```
> eigen(Sm)
```

tj. kao naredbu, obe komponente s prikažu na ekranu , sa njihovim nazivima.

5.7.4 Dekompozicija singularnih vrijednosti i determinante

Funkcija svd(M) uzima kao argument matricu M i računa singularnu dekompoziciju matrice M. Ova dekompozicija se sastoji od matrice ortonormalnih stupaca U s istim prostorom stupaca kao M, druge matrice ortonormalnih stupaca V s prostorom stupaca kao što je prostor redaka M-a, i dijagonalne matrice spozitivnim elementima D tako da je M = U %∗% D %∗% t(V). D je zapravo vraćen kao vektor dijagonalnih elemenata. Rezultat svd(M) je zapravo lista triju komponenata nazvanih d, u i v, čije je značenje očito.
Ako je M u stvari kvadratna, nije teško vidjeti da

```
> absdetM <- prod(svd(M)$d)
```

izračunava apsolutnu vrijednost determinante M-a. Ako je taj izračun često potreban s mnogo matrica mogao bi se definirati kao R-ova funkcija

```
> absdet <- function(M) prod(svd(M)$d)
```

nakon čega možemo upotrijebiti absdet( ) kao samo još jednu funkciju R-a. Još jedan trivijalan ali moguće koristan primjer, možda biste željeli napisati neku funkciju, recimo tr( ), za računanje *traga* kvadratne matrice. [Savjet: Nema potrebe za eksplicitnim korištenjem petlje. Pogledajte ponovno funkciju diag( ).]

5.7.5 Metoda najmanjih kvadrata i QR dekompozicija

Funkcija lsfit() vraća listu koja daje rezultate u postupku prilagođavanja (fitting) metodom najmanjih kvadrata.
Pridruživanje kao što je

```
> ans <- lsfit(X, y)
```

daje rezultate prilagođavanja metodom najmanjih kvadrata, kod čega je y vektor s opservacijama a X dizajn matrica. Pogledajte pomoć (u R-u: ?ime_naredbe) za podrobnije podatke, a također i sve vezano uz funkciju ls.diag( ) u vezi regresije. Obratite pozornost da je ukupni srednjak (grand mean) automatski uključen te ga ne treba eksplicitno uključivati kao stupac X-a.
Druga tijesno povezana funkcija je funkcija qr( ) i njoj srodne funkcije. Uzmite u obzir sljedeća pridruživanja

```
> Xplus <- qr(X)
> b <- qr.coef(Xplus, y)
> fit <- qr.fitted(Xplus, y)
```

```
> res <- qr.resid(Xplus, y)
```

One izračunavaju pravokutnu projekciju y-a na raspon X-a u prilagođavanju, projekciju na pravokutni komplement u res-u i vektor koeficijent za projekciju u b-u, to jest, b je u biti rezultat MATLAB-ovog "backslash" operatora.
Ne pretpostavlja se da X ima raspon cijeloga stupca. Suvišci će se otkriti i odmah odstraniti.
Ovaj način je starija alternativa računanja metode najmanjih kvadrata. Iako je još uvijek korisna u nekim slučajevima, ona se danas uglavnom zamjenjuje mogućnostima statističkih modela, o čemu će se raspravljati u Poglavlju 11 [Statistički modeli u R-u], str. 57.

5.8 Kreiranje matrica pomoću funkcija cbind( ) i rbind( )

Kako smo već prije vidjeli, matrice se mogu graditi od drugih vektora i matrica s pomoću funkcija cbind( ) i rbind( ). U grubim crtama, cbind( ) kreira novu matricu povezujući zajedno matrice horizontalno, u stupcima, dok rbind( ) kreira matrice vertikalno, u redcima.
U pridruživanju

```
> X <- cbind(arg_1, arg_2, arg_3,…)
```

argumenti za cbind( ) moraju biti vektori bilo koje duljine ili matrice iste veličine stupaca, to jest s istim brojem redaka. Rezultat je matrica s nanizanim argumentima *arg_1*, *arg_2*,… koji formiraju stupce.
Ako su neki od argumenata za cbind( ) vektori, oni mogu biti kraći od veličine stupca bilo koje prisutne matrice, u kojem se slučaju ciklički proširuju da odgovaraju veličini stupca matrice (ili duljini najduljeg vektora ako nisu date matrice).
Funkcija rbind( ) obavlja takvu operaciju za retke. U tome se slučaju svi argumenti vektora, moguće ciklički produljeni, uzimaju naravno kao vektori retka.
Pretpostavimo da X1 i X2 imaju isti broj redaka. Za njihovo spajanje s pomoću stupaca u matricu X, zajedno s početnim stupcem 1s možemo upotrijebiti

```
> X <- cbind(1, X1, X2)
```

Rezultat od rbind( ) ili cbind( ) uvijek ima status matrice. Prema tomu su cbind(x) i rbind(x) možda najjednostavniji načini da se vektor x može eksplicitno tretirati kao matrica stupaca odnosno matrica redaka.

5.9 Funkcija povezivanja, c( ), s poljima

Trebalo bi imati na umu da funkcije povezivanja cbind( ) i rbind( ) poštuju svojstvo dimenzije (dim) , dok ih osnovna c( ) funkcija ne poštuje, nego radije čisti numeričke objekte od svih *dim* i *dimnames* svojstava. To je ponekad korisno.
Službeni način da se neki polje vrati natrag u jednostavan vektorski objekt je upotrebom as.vector( )

```
> vec <- as.vector(X)
```

Međutim, sličan rezultat se može postići upotrebom c( ) sa samo jednim argumentom:

```
> vec <- c(X)
```

Postoje male razlike između ta dva načina, ali je konačno izbor između njih uglavnom stvar stila (prvi se način, međutim, preferira).

5.10 Tablice frekvencija iz faktora

Podsjetimo se da neki faktor definira podjelu u grupe. Slično tomu, jedan par faktora definira dvosmjernu unakrsnu klasifikaciju, i tako dalje. Funkcija table( ) omogućuje računanje tablica frekvencija iz faktora jednake duljine. Ako imamo argumente *k* faktora, rezultat je k-struko polje frekvencije.
Pretpostavimo, na primjer, da je statef faktor (primjer iz 4.1.) koji daje šifru države za svaki element u vektoru s podacima. Pridruživanje

```
> statefr <- table(statef)
```

daje u statefr tablicu učestalosti za svaku državu u uzorku. Učestalosti su poredane i obilježene svojstvom nivoa faktora. Ovaj jednostavan slučaj je ekvivalentan, ali mnogo pogodniji od

```
> statefr <- tapply(statef, statef, length)
```

Pretpostavimo nadalje da je incomef kategorija koja daje pogodno definiran "income class" za svaki unos u vektor s podacima, na primjer s pomoću funkcije cut( ):

```
> factor(cut(incomes, breaks = 35+10*(0:7))) -> incomef
```

Zatim za izračun dvosmjerne tablice učestalosti koristimo:

```
> table(incomef,statef)
```

	statef							
incomef	act	nsw	nt	qld	sa	tas	vic	wa
(35,45]	1	1	0	1	0	0	1	0
(45,55]	1	1	1	1	2	0	1	3
(55,65]	0	3	1	3	2	2	2	1
(65,75]	0	1	0	0	0	0	1	0

Proširenje na tablice učestalosti s većim brojem smjerova je neposredno.

6 Liste i spremnici podataka (data frames)

6.1 Liste

R-ova *lista* je objekt koji se sastoji od niza objekata koje zovemo *komponente objekta* (liste). Nema posebne potrebe da komponente budu istoga tipa ili vrste, te se, na primjer neka lista može sastojati od numeričkog vektora, logičke vrijednosti, matrice, složenog vektora, znakovnog polja, funkcije, i tako dalje. Navodimo jednostavan primjer kako se radi lista:

```
> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

Komponente su uvijek *numerirane* te ih se uvijek može navesti kao takve. Tako, ako je Lst naziv liste s četiri komponente, komponente se mogu pojedinačno navoditi kao Lst[[1]], Lst[[2]], Lst[[3]] i Lst[[4]]. Ako je, nadalje, Lst[[4]] polje koji ima vektor kao indeks, tada je Lst[[4]][1] prvi podatak.
Ako je Lst neka lista, tada funkcija length(Lst) daje broj komponenata najviše razine koje lista sadrži.

Komponente lista mogu također imati *nazive*, te se u tom slučaju komponentu može navesti dajući naziv komponente kao znakovni niz u dvostrukim uglatim zagradama umjesto brojke ili, što je pogodnije, s pomoću izraza koji ima oblik

> *name*$*component_name*

To je vrlo korisna konvencija jer omogućava lakše dolaženje do tražene komponente ako zaboravite njezin broj.
Tako je u jednostavnom primjeru gore navedenom:

Lst$name isto kao Lst[[1]] i daje nam "Fred",
Lst$wife je isto kao Lst[[2]] i daje nam "Mary", a
Lst$child.ages[1] je isto kao Lst[[4)]] [1] i predstavlja brojku 4.

Pored toga, nazivi komponenata liste se mogu također korisiti u dvostrukim uglatim zagradama, t.j. Lst[["name"]] je isto kao Lst$name. To je naročito korisno kada se naziv komponente koju treba ekstrahirati pohranjuje u drugoj varijabli kao što je slučaj u

> x <- "name"; Lst[[x]]

Veoma je važno razlikovati Lst[[1]] od Lst[1]. '[[…]]' je operator koji se upotrebljava za odabir jednog elementa dok je '[…]' generalni operator za indekse. Tako je ovaj prvi *prvi objekt u listi* Lst, te ako lista ima naziv, naziv *nije uključen*. Drugi je *podlista liste* Lst *koja se sastoji samo od prvog unosa*. *Ako se radi o listi koja ima naziv, naziv se prenosi na podlistu*.
Nazivi komponenata mogu se kratiti do najmanjeg broja slova potrebnog da ih jedinstveno identificira. Tako se Lst$coefficients može minimalno specificirati kao Lst$coe, a Lst$covariance kao Lst$cov.
Vektor s nazivima je zapravo jednostavno svojstvo liste kao i bilo koji drugi, te se s njime može postupati kao s takvim. Također se i drugim strukturama može, naravno, slično dati svojstvo s *imenima*.

6.2 Konstruiranje i modificiranje lista

Nove liste se mogu formirati od postojećih objekata s pomoću funkcije list( ). Pridruživanje oblika

> Lst <- list(*name_1*=*object_1*, …, *name_m*=*object_m*)

uspostavlja listu Lst od *m* komponenata korištenjem *object_1, …, object_m* za komponente i dajući im nazive kako su specificirani nazivima argumenta, (koji se mogu slobodno birati). Ako se ti nazivi izostavljaju, komponente su samo numerirane. **Komponente korištene za formiranje liste *se kopiraju* kod formiranja nove liste**, što ne utječe na originale.
Liste, kao i bilo koji objekt sa indeksima, mogu se proširivati specificiranjem dodatnih komponenata. Na primjer

> Lst[5] <- list(matrix=Mat)

6.2.1 Povezivanje lista

Kada se funkciji povezivanja c( ) daju argumenti koji su liste, rezultat je objekt koji je također tipa liste, i čije su komponente komponente lista u argumentu spojenih zajedno u niz.

> list.ABC <- c(list.A, list.B, list.C)

Podsjetimo se da je u slučaju vektorskih objekata kao argumenata funkcija povezivanja slično spojila zajedno sve argumente u jednu vektorsku strukturu. U ovom slučaju sva druga svojstva, kao npr. dim - dimenzija, se odbacuju.

6.3 Spremnici podataka (data frames)

Spremnik podataka je lista koja ima klasu "data.frame". Postoji restrikcija za liste koje se mogu pretvoriti u spremnik podataka, naime

- Sve komponente moraju biti vektori (numerički, znakovni ili logički), faktori, brojčane matrice, liste, ili drugi data frameovi.
- Matrice, liste i spremnici podataka daju onoliko varijabla novome spremniku koliko imaju stupaca, elemenata odnosno varijabli.
- Numerički vektori, logički vektori i faktori uključuju se onako kakvi jesu, a znakovni vektori se spajaju u faktore, čiji su nivoi jedinstvene vrijednosti koje se pojavljuju u vektoru.
- Vektorske strukture koje se pojavljuju kao varijable spremnika moraju sve imati *istu duljinu*, a matrice moraju sve imati istu *veličinu retka*.

Spremnici podataka se u mnogome može promatrati kao matricu sa stupcima koji mogu biti različitih tipova i svojstava. Spremnik podatak može biti prikazan u obliku matrice, s retcima i stupcima ekstrahiranim prema konvencijama o indeksiranju matrica.

6.3.1 Izrada spremnika podataka

Objekti koji zadovoljavaju restrikcijama postavljenim za stupce (komponente) spremnika mogu se upotrijebiti za konstruiranje spremnika s pomoću funkcije data.frame:

```
> accountants <- data.frame(home=statef, loot=income, shot=incomef)
```

Lista čije komponente zadovoljavaju restrikcijama za spremnik može se pretvoriti u spremnik podataka upotrebom funkcije as.data.frame( ).

Najjednostavniji način konstruiranja spremnika podataka od početka je upotrebom funkcije read.table( ) za čitanje jednog cijelog spremnika iz vanjske datoteke. O tome se dalje raspravlja u Poglavlju 7 [Čitanje podataka iz datoteka], str. 37.

6.3.2 attach() i detach()

Oznaka $, kao na primjer accountants$statef, za komponente liste nije uvijek jako pogodna. Korisno bi bilo učiniti na neki način komponente neke liste ili spremnika privremeno vidljivima kao varijable pod njihovim nazivom komponenata, ne morajući svaki puta eksplicitno navoditi naziv liste.

Funkcija attach( ), jednako kao što ima naziv direktorija kao argument, može također imati i spremnik podataka. Pretpostavimo da je lentils spremnik s tri varijable lentils$u, lentils$v, lentis$w. Funkcija

```
> attach(lentils)
```

stavlja spremnik podataka na poziciju 2 pretražne putanje te, pod uvjetom da na poziciji 1 nema varijabla u, v ili w, one su dostupne kao varijable iz spremnika same po sebi. Ovdje pridruživanja poput

```
> u <- v+w
```

ne zamjenjuje komponentu u spremniku, nego je radije maskira jednom drugom varijablom u u radnom direktoriju na poziciji 1 pretražnog puta. Za vršenje permanentne promjene u samom spremniku, najjednostavniji je način ponovno se vratiti na $ oznaku:

```
> lentils$u <- v+w
```

Međutim, nova vrijednost komponente u nije vidljiva dok se spremnik ne odvoji i ponovno ne priključi.
Za odvajanje spremnika podataka koristi se funkcija

```
> detach( )
```

Točnije, ovaj iskaz odvaja iz pretražne putanje cjelinu koja se trenutno nalazi na poziciji 2. Tako u ovom kontekstu varijable u, v i w više ne bi bile vidljive osim pod notacijom za listu kao lentils$u i tako dalje. Cjeline na pozicijama u pretražnom putu višim od 2 mogu se odvojiti davanjem njihovog broja detach-u, no mnogo je sigurnije uvijek koristiti naziv, na primjer s pomoću detach(lentils) ili detach("lentils")

NAPOMENA: U sadašnjoj verziji R-a, pretražna putanja može sadržavati najviše 20 stavki. Izbjegavajte priključivanje istoga spremnika više od jedanput. Uvijek odvojite spremnik čim ste završiti s korištenjem njegovih varijabla.

NAPOMENA: U sadašnjoj verziji R-a, liste i spremnici podataka se mogu jedino priključivati na poziciji 2 ili višoj. Nije moguće direktno prirdruživanje u neku priključenu listu ili spremnik (tako da su u određenoj mjeri statični).

6.3.3 Rad s spremnicima podataka

Korisna konvencija koja omogućava nesmetan rad s mnogo različitih problema zajedno u istom radnom direktoriju sastoji se u sljedećem

- grupirajte zajedno sve varijable za bilo koji dobro definiran i odvojen problem u spremnik podataka pod odgovarajuće informativnim nazivom;

- prilikom rada na nekom problemu priključite odgovarajuće spremnike na poziciju 2, i koristite radni direktorij na nivou 1 za trenutno radne veličine i privremene varijable;

- prije završetka rada na nekom problemu, dodajte spremniku podataka korištenjem $ sintakse za pridruživanje sve varijable koje želite zadržati za buduće korištenje, i zatim odvojite spremnik podataka sa funkcijom detach( );

- na kraju odstranite sve nepotrebne varijable iz radnog direktorija i držite ga čistim što je više moguće od preostalih privremenih varijabla.

Na takav način je prilično jednostavno u istom direktoriju raditi s mnogo problema, od kojih svi na primjer imaju varijable nazvane x, y i z.

6.3.4 Priključivanje proizvoljnih lista

attach( ) je generička funkcija koja omogućava da se na pretražni putanju mogu priključiti ne samo direktoriji i spremnici podataka, nego također druge klase objekta. Posebno, bilo koji objekt tipa "list" može se priključiti na isti način:

```
> attach(any.old.list)
```

Sve što je bilo priključeno može se odvojiti korištenjem funkcije detach(), uz upotrebu pozicijskog broja ili, što je bolje, uz upotrebu naziva.

6.3.5 Rukovanje s pretražnom putanjom

Funkcija *search* pokazuje pretražnu putanju na kojem se trenutno nalazite, te je stoga vrlo koristan način čuvanja traga o tome koji su spremnici podataka i liste (i paketi) bili priključeni i odvojeni. Ona početno daje

```
> search( )

[1] ".GlobalEnv""Autoloads"    "package:base"
```

gdje je .GlobalEnv workspace.[1]
Nakon što je lentils priključen dobivamo

```
> search( )

[1] ".GlobalEnv"        "lentils"         "Autoloads"          "package:base"

> ls(2)

[1] "u" "v" "w"
```

i kao što vidimo, ls (ili objects()) se može upotrijebiti za ispitivanje sadržaja bilo koje pozicije na pretražnoj putanji.
Na kraju odvajamo spremnik podataka i potvrđujemo da je uklonjen iz pretražnog puta.

```
> detach("lentils")
> search( )

[1] ".GlobalEnv"     "Autoloads"    "package:base"
```

7 Čitanje podataka iz datoteka

Veliki objekti s podacima obično će se čitati kao vrijednosti iz vanjskih datoteka radije nego da se unose za vrijeme rada u R-u. R-ove mogučnosti za unošenje podataka su jednostavne, sa prilično striktnim, čak dosta nefleksibilnim zahtjevima. R-ovi dizajneri pretpostavljaju da ćete moći modificirati datoteke koje unosite s pomoću drugih alata kao što su editori ili Perl[1] kako biste zadovoljili zahtjeve R-a. Uglavnom je to vrlo jednostavno.
Varijable ćemo uglavnom imati u spremnicima podataka, što se preporuča, cijeli spremnik se može učitati direktno s pomoću funkcije read.table(). Također postoji primitivnija funkcija za unos, scan( ), koja se može direktno pozvati.
Za podrobnije informacije o unošenju (importiranje) podataka u R kao i o iznošenju (eksport) podataka pogledajte priručnik o *unošenju podataka u R te njihovom iznošenju*.

7.1 Funkcija read.table( )

[1] Vidi on-line help za autoload što se tiče značenja drugog izraza.
[1] Kod rada u UNIX-u mogu se koristiti alati Sed ili Awk.

Da bi se jedan cijeli spremnik podataka učitao direktno, potrebno je da vanjska datoteka ima poseban oblik.

- Prvi red u datoteci treba sadržavati *naziv* za svaku varijablu u spremniku.
- Svaki dodatni red u datoteci ima za prvu stavku *oznaku retka* i vrijednosti za svaku varijablu.

Ako datoteka ima u svom prvom redu jednu stavku manje nego u drugom, ovakvo uređenje se podrazumijeva. Tako bi prvih nekoliko redova datoteke koju treba učitati kao spremnik podataka moglo izgledati na sljedeći način.

Input file form with names and row labels:

	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

...

Po definiciji se numerički podaci (osim oznaka redaka) čitaju kao numeričke varijable a varijable koje nisu numeričke, kao što je Cent.heat u ovome primjeru, kao faktori. To se po potrebi može promijeniti.
Funkcija read.table( ) može se tada upotrijebiti za čitanje spremnika direktno

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

Često ćete možda htjeti izostaviti direktno uključivanje oznaka redaka i upotrijebiti oznake po definiciji. U tome se slučaju iz datoteke može izostaviti stupac oznake redaka kako se vidi iz sljedećeg.

Input file form without row labels:

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

...

Spremnik se tada može pročitati kao

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

gdje opcija header=TRUE specificira da je prvi red red s nazivima, i da prema tome, kako implicira oblik datoteke, nisu dane eksplicitne oznake redaka. Ako imamo, u prvim redovima komentare, možemo ih izbaciti (tj. ne učitati) sa opcijom skip=BrojRedovaZaPreskočiti. Za potpunu sintaksu upotrebe ove naredbe treba u R-u koristiti pomoć: ?read.table.

7.2 Funkcija scan( )

Pretpostavimo da su vektori podataka jednake duljine i da ih treba učitati paralelno. Pretpostavimo, nadalje, da imamo tri vektora, prvi vektor tipa karakter a ostala dva numeričkog tipa, a da je datoteka 'input.dat'. Prvi je korak upotrijebiti scan( ) da se tri vektora učitaju kao lista, na sljedeći način

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

Drugi argument je prazna struktura koja samo definira tipove tri vektora koja treba očitati. Rezultat, sadržan u inp, je lista čije su komponente tri očitana vektora. Za odvajanje podataka u tri odvojena vektora upotrijebite pridruživanje poput

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

Pogodnije je da prazna lista ima komponente s nazivima, u kojemu slučaju se oni mogu upotrijebiti za pristup učitanim vektorima. Na primjer

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

Ako želite pristupiti varijablama odvojeno možete ih ponovno pridružiiti:

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

ili listu možete priključiti u poziciju 2 pretražne putanje (vidi Poglavlje 6.3.4 [Priključivanje proizvoljnih lista], str. 36).

Ako je drugi argument jedna vrijednost a ne lista, učitava se jedan vektor, čije sve komponente moraju biti istoga tipa kao što je prazna struktura.

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

Postoje razrađenije procedure za učitavanje koje su detaljizirane u priručniku.

7.3 Pristup ugrađenim skupovima podataka

R ima oko 100 skupova podataka a ostali su skupovi dostupni u paketima (uključujući standardne pakete koji dolaze s R-om). Da biste vidjeli listu skupova podataka koji su vam trenutno dostupni upotrijebite

```
>data( )
```

Od verzije 2.0.0. R-a svim skupovima koji dolaze s R-om može se pristupiti direktno – navođenjem naziva. Međutim, mnogi paketi još upotrebljavaju staru konvenciju u kojoj se podacima pristupa naredbom data(), na primjer

```
>data(infert)
```

i to se još može koristiti i kod standardnih paketa. U većini slučajeva time će se učitati R-ov objekt istoga naziva. Međutim, u nekoliko slučajeva će se učitati više objekata, pa treba pogledati on-line help da vidite što možete očekivati.

7.3.1 Učitavanje podataka iz drugih R-ovih paketa

Za pristup podacima iz nekog drugog paketa upotrijebite argument package, na primjer

```
>data(package="nls")
>data(Puromycin, package="nls")
```

Ako je paket priključen preko library, njegovi skupovi podataka se automatski uključuju u traženje, tako da će

```
>library(nls)
>data( )
>data(Puromycin)
```

izlistati sve skupove podataka u svim trenutno priključenim paketima (barem **base** i **nls**) i zatim učitati Puromycin iz prvog paketa u kojemu se takav skup nalazi.
Paketi koje načini sam korisnik mogu biti bogat izvor skupova podataka.

7.4 Uređivanje podataka

Kada se edit pozove, a parametar je spremnik podataka ili matrica, edit pruža posebno okruženje za editiranje – prikazuje podatke u tablici ("spreadsheet-like"). To je korisno za male izmjene nakon što su podaci učitani. Naredba

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

omogućit će vam editiranje skup podataka xold, a po završetku editiranja izmijenjeni se objekt pridružuje objektu xnew-u.

Upotrijebite

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

za unošenje novih podataka putem tabele.

8 Razdiobe vjerojatnosti

8.1 R kao skup statističkih tablica

Jedna je od pogodnosti korištenja R-a što daje obiman skup statističkih tablica. Daju se funkcije za procjenu funkcije kumulativne distribucije $P(X \leq x)$, funkcije vjerojatnosti gustoće i funkcije kvantila (s time da je q najmanji x tako da je $P(X \leq x) > q$), i za simuliranje iz distribucije.

Raspodjela	R naziv	dodatni argumenti
beta	beta	shape1, shape2, ncp
binomna	binom	size, prob
Cauchyjeva	cauchy	location, scale
Hi-kvadratna	chisq	df, ncp
eksponencijalna	exp	rate
F	f	df1, df1, ncp

gama	gamma	shape, scale
geometrijska	geom	prob
hipergeometrijska	hyper	m, n, k
log-normalna	lnorm	meanlog, sdlog
logistička	logis	location, scale
negativna binomna	nbinom	size, prob
normalna	norm	mean, sd
Poissonova	pois	lambda
Studentova	t	df, ncp
uniformna	unif	min, max
Weibullova	weibull	shape, scale
Wilcoxonova	wilcox	m, n

Stavite ispred gore navedenih naziva prefiks ‘d’ za density = gustoća, prefiks ‘p’ za CDF (Common Data Form) - uobičajeni oblik podataka, ‘q’ za funkciju kvantila i ‘r’ za simulaciju slučajnih odstupanja. Prvi argument je x za d*xxx*, q za p*xxx*, p za q*xxx* i n za r*xxx* (osim kod rhyper and rwilcox, gdje je nn). Parametar necentralnosti ncp je trenutno jedini dostupan za CDF-ove i nekoliko drugih funkcija: vidi help za detaljne informacije.
Funcije p*xxx* i q*xxx* sve imaju logičke argumente lower.tail i log.p, a funkcije d*xxx* imaju log. To omogućuje, npr., dobivanje kumulativne (ili "integrirane") funkcije rizika (*hazard* funkcije), $H(t) = - \log(1 – F(t))$, s pomoću

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

ili mnogo točnijih log-vjerojatnosti (s pomoću d*xxx*(…, log=TRUE)), direktno.
Pored toga, postoje funkcije ptukey and qtukey za raspodjelu studentiziranog raspona uzoraka iz normalne raspodjele.
Evo nekih primjera

```
> ## 2-strana p-vrijednost za t raspodjelu (distribuciju)
> 2*pt(-2.43, df = 13)

> ## točka gornjih 1% podataka za F(2, 7) raspodjelu
> qf(0.99, 2, 7)
```

8.2 Ispitivanje raspodjele skupa podataka

Sa skupom podataka koji ima jednu slučajnu varijablu (univariate), raspodjelu možemo ispitati na velik broj načina. Najjednostavniji je način ispitati brojeve. Dva blago različita sažetka se dobiju sa summary() i fivenum(), a pregled brojeva se dobije sa stem ("stem and leaf" grafikonom, to je graf najbliži smislu korjena, grana i lista).
(Ovdje ćemo koristiti podatke iz skupa podataka *faithful*. To su podaci koji se odnose na duljinu među erupcijama i duljinu erupcije Old Faithful gejzira u nacionalnom parku Yellowstone u državi Wyoming, USA. Ako napišemo help(faithful), dobiti ćemo opis problema i podataka. Ako napišemo *faithful* dobiti ćemo ispis podataka na ekran).

```
> data(faithful)

> 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] 1.60000  2.1585  4.0000  4.4585  5.1000
```

(Daje nam pet Tukey-evih brojeva(minimum, doljnji kvartil, medijan, gornji kvartil, maksimum) koji se kasnije koristi za Box plot.)

```
> stem(eruptions)

  Decimalna točka je za 1 brojku(brojke) u lijevo od |

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 | 000000335778888800223355557778
42 | 033355557788002333335555777 78
44 | 022223355577800000000233333577 8888
46 | 0000233357700000023578
48 | 00000022335800333
50 | 0370
```

Stem-and-leaf grafikon je poput histograma, a R posjeduje funkciju hist za grafičko prikazivanje histograma.

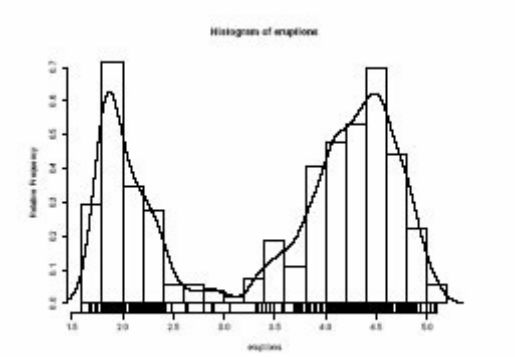
```
> hist(eruptions)

## učini intervale manjima, nacrtaj gustoću

> hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)

> rug(eruptions) # dodaj podatke na histogram
```

Elegantniji grafovi gustoće mogu se načiniti s pomoću density, a mi smo dodali liniju koju je stvorila funkcija density u ovom primjeru. Širina pojasa bw je izabrana s pomoću pokušaja i pogreške jer predefinirana vrijednost (default) daje previše poravnanja (to je obično slučaj kod "zanimljivih" gustoća). (Automatizirane metode s izborom širine pojasa su uključene u paketima **MASS** i **KernSmooth**.)

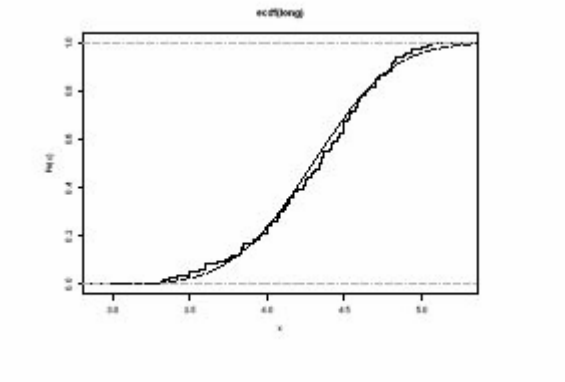


Možemo napraviti grafikon funkcije empirijske kumulativne raspodjele s pomoću funkcije ecdf u standardnom paketu **stepfun**.

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

Ova raspodjela je očito daleko od bilo koje standardne raspodjele. Kako bi bilo da uzmemo dio raspodjele desnoj strani, recimo erupcije dulje od 3 minute? Prilagodimo normalnoj raspodjeli i prekrijemo sa prilagođenom kumulativnom funkcijom raspodjele.

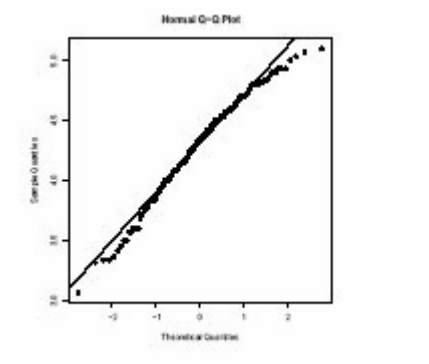
```
> long <- eruptions[eruptions > 3]
> plot(ecdf(long), do.points=FALSE, verticals=TRUE)
> x <- seq(3, 5.4, 0.01)
> lines(x, pnorm(x, mean=mean(long), sd=sqrt(var(long))), lty=3)
```



Grafikoni kvantil-kvantil (Q-Q) mogu pomoći da to pažljivije ispitamo.

```
> par(pty="s")
qqnorm(long); qqline(long)
```

koje pokazuje dobru podudarnost ali kraći desni rep nego što bi se očekivalo za normalnu raspodjelu. Izvršimo usporedbu sa simuliranim podacima iz neke *t* raspodjele



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

koja će pokazivati dulje repove nego što se očekuje za normalni uzorak.
Možemo učiniti Q-Q grafikon u odnosu na izvornu raspodjelu s pomoću

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

Konačno, možda bismo željeli formalniji test slaganja s normalnom razdiobom. Paket **ctest** omogućuje Shapiro-Wilk test

```
> library(ctest)
> shapiro.test(long)

        Shapiro-Wilk test normaliteta

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

i Kolmogorov-Smirnov test

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

        Kolmogorov-Smirnov test za jedan uzorak

data:   long
D = 0.0661, p-value = 0.4284
alternativna pretpostavka: je dvostrana
```

(Obratite pozornost da ovdje teorija distribucije nije važeća, jer smo procijenili parametre normalne raspodjele iz istoga uzorka.)

8.3 Testovi sa jednim i sa dva uzorka

Do sada smo uspoređivali jedan uzorak s normalnom raspodjelom. Više uobičajena je operacija usporedbe aspekata dva uzorka. Obratite pozornost da su u R-u svi "klasični" testovi uključujući dolje opisane testove u paketu **stats**, za koje može biti potrebno učitati eksplicitno upotrebom naredbe library(stats) kako bi testovi bili dostupni.

Razmotrimo sljedeće skupove podataka o latentnoj toplini fuzije leda (*cal/gm*) iz Rice (1995, str. 490)

Metoda 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
Metoda B: 80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97

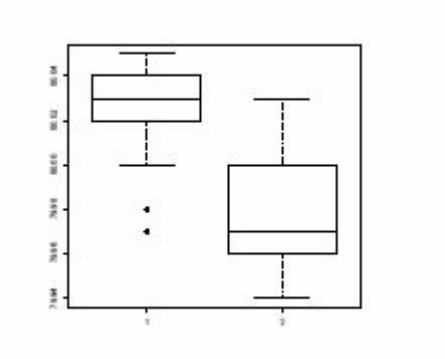
Boxplots – grafikoni omogućuju jednostavnu grafičku usporedbu ova dva uzorka.

```
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 – grafikon (A, B)
koji pokazuje da prva grupa teži višim vrijednostima nego druga grupa.



Za ispitivanje jednakosti srednjih vrijednosti ova dva primjera, možemo upotrijebiti t-test s pomoću:

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

        Welchov t-test s dva uzorka

data: A i B
t = 3.2499, df = 12.027, p-value = 0.00694
alternativna pretpostavka: prava razlika srednjih vrijednosti nije jednaka 0
95-postotni interval povjerenja:
 0.01385526 0.07018320
procjene uzoraka:
srednja vrijednost x-a srednja vrijednost y-a
     80.2077                79.97875
```

što pokazuje znatnu razliku, pod pretpostavkom normaliteta. Po defaultu R-ova funkcija ne pretpostavlja jednakost varijancija u dva uzorka (za razliku od slične t.test funkcije S-PLUS-a). Možemo upotrijebiti F-test za testiranje jednakosti varijanca, pod uvjetom da su dva uzorka uzeta od normalnih populacija.

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

    F-test za usporedbu dvije varijancije

data: A i B
F = 0.5837, num df = 12, denom df = 7, p-value = 0.3938
alternativna pretpostavka: pravi omjer varijanca nije jednak 1
95-postotni interval povjerenja:
 0.1251097 2.1052687
procjene uzoraka:
omjer varijancija
      0.5837405
```

što ne pokazuje značajnu razliku, tako da možemo upotrijebiti klasični *t*-test koji pretpostavlja jednakost varijancija.

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

        t-test za dva uzorka

data: A i B
t = 3.4722, df = 19, p-value = 0.002551
alternativna pretpostavka: prava razlika srednjih vrijednosti nije jednaka 0
95-postotni interval povjerenja:
 0.01669058 0.06734788
procjena uzoraka:
srednja vrijednost x-a srednja vrijednost y-a
     80.2077                79.97875
```

Svi ovi testovi pretpostavljaju normalnost dva uzorka. Wilcoxon (ili Mann-Whitney) test za dva uzorka pretpostavlja jedino da postoji zajednička neprekidna raspodjela.

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

        Wilcoxonov rank sum test s korekcijom kontinuiteta

data: A i B
W = 89, p-value = 0.007497
alternativna pretpostavka: pravi mu nije jednak 0
Upozorenje:
Ne može se izračunati točnu p-vrijednost s vezama u: wilcox.test(A, B)
```

Obratite pozornost na upozorenje: u svakom uzorku postoji nekoliko veza, što snažno upućuje da su ti podaci iz diskretne raspodjele (vjerojatno uslijed zaokruživanja).
Postoji nekoliko načina grafičkog uspoređivanja dva uzorka. Već smo vidjeli par boxplots grafikona. Sljedeće

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

pokazat će dva empirijska CDF-a, a qqplot će načiniti Q-Q grafikon dva uzorka. Kolmogorov-Smirnov test ima maksimalnu vertikalnu udaljenost između dva ecdf-a pod pretpostavkom zajedničke kontinuirane raspodjele:

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

        Kolmogorov-Smirnov test za dva uzorka

data:  A i B
D = 0.5962, p-value = 0.05919
alternativna pretpostavka: dvostrano

Poruka upozorenja:
ne može se izračunati ispravne p-vrijednosti s vezama u: ks.test(A, B)
```

9 Grupiranje, petlje i uvjetno izvršavanje

9.1 Grupiranje izraza

R je jezik izražavanja u smislu da je njegova jedina vrsta naredbe funkcija ili izraz koja vraća rezultat. Izraz je čak je i pridruživanje, čiji je rezultat dodijeljena mu vrijednost, i može se koristiti gdje god se može koristiti bilo koji izraz; naročito su mogući višestruka pridruživanja.

Naredbe se mogu zajedno grupirati u vitičastim zagradama, {*expr_1*;…; *expr_m*}, u tom slučaju je **rezultat grupe rezultat posljednjega izraza u grupi koji se izvršava**. Budući je takva grupa također izraz, i ona sama može, na primjer, biti stavljena u zagrade i upotrijebljena kao dio nekog još većeg izraza, i tako dalje.

9.2 Kontrolni iskazi

9.2.1 Uvjetno izvršavanje: *if* iskaz

Jezik raspolaže s uvjetnom strukturom oblika

> if (*expr_1*) *expr_2* else *expr_3*

gdje izraz *expr_1* mora imati za rezultat neku logičku vrijednost (TRUE ili FALSE) i tada je rezultat cijelog izraza očigledan. Tj. ako expr_1 daje TRUE izvršava se expr_2 inaće se izvršava expr_3. Ako expr_1 vrati NA (Not Available) sustav će javiti pogrešku.
Operatori (tzv. "short-circuit") && i || se često upotrebljavaju u expr_1 iskazu. Dok se & i | primjenjuju na vektore && i || se primjenjuju na vektore duljine jedan, i jedino, ako treba, određuju njihov drugi argument.
Postoji vektorska verzija strukture if/else , funkcija ifelse. Ona ima oblik ifelse(condition, a, b) i vraća vektor koji ima duljinu njegovog najduljeg argumenta, s elementom a[i] ako je condition[i] ispunjen odnosno s elementom b[i] ako uvjet nije ispunjen.

9.2.2 Strukture za ponavljano izvršavanje – *for* petlja, *repeat* i *while*

Struktura izraza za *for* petlju ima oblik

> for (*name* in *expr_1*) *expr_2*

gdje je *name* kontrolna varijabla. *expr_1* je vektorski izraz, (često niz poput 1:20), a *expr_2* je često grupirani izraz sa svojim pod-izrazima pisanim kao dummy *name*. *expr_2* se ponavljano izvršava dok *name* prolazi kroz vrijednosti u vektorskom izrazu *expr_1.*
Kao primjer, pretpostavimo da je ind vektor indikatora klase i da želimo načiniti posebne grafove y-a u zavisnosti od x unutar klasa. Jedna je mogućnost upotrijebiti *coplot( )* o kojemu će biti riječi kasnije, koji će dati niz grafova odgovarajućih svakome nivou faktora. Drugi je način staviti sve grafove u jedan prikaz, kako slijedi:

```
> xc <- split(x, ind)

> yc <- split(y, ind)

> for (i in 1:length(yc)) {
      plot(xc[[i]], yc[[i]]) ;
```

```
    abline(lsfit(xc[[i]], yc[[i]]))
}
```

(Obratite pozornost na funkciju split() koja daje listu vektora dobivenih cijepanjem nekog većeg vektora prema klasama specificiranim faktorom. To je korisna funkcija koja se uglavnom koristi vezano uz prikaz grafova tipa "boxplots". Pogledajte help za više pojedinosti.)

***UPOZORENJE**: for()* petlja se u R kôdu upotrebljava mnogo rjeđe nego kod jezika koji se kompiliraju. Kôd koji uzima pregled 'cijelog objekta' će u R-u biti je jasniji i brži.
Druge mogućnosti za korištenje petlji su iskaz

> repeat *expr*

i iskaz

> while (*condition*) *expr*

Naredba *break* se može upotrijebiti za završetak bilo koje petlje,. To je jedini način da se završi *repeat* .
Naredba *next* se može upotrijebiti da se prekine jedan određeni ciklus i da se skoči na sljedeći.
Kontrolni iskazi se najčešće upotrebljavaju vezano uz *funkcije* o kojima se raspravlja u Poglavlju 10 [Pisanje vlastitih funkcija], str. 48, gdje se navode dodatni primjeri.

10 Pisanje vlastitih funkcija

Kako smo do sada neformalno vidjeli, R jezik dozvoljava korisniku **kreiranje objekata tipa funkcija**. To su prave R-ove funkcije koje su pohranjene u posebnom internom obliku i mogu se upotrebljavati u drugim izrazima. Tijekom toga postupka R jezik dobiva sve više na snazi i spretnosti, pa je učenje pisanja korisnih funkcija jedan od glavnih načina ugodnog i produktivnog korištenja R-a.
Treba naglasiti da je većina funkcija koje se dobivaju kao dio R-ovog sustava, na primjer mean( ), var( ), postscript( ) i tako dalje, sama pisana u R-u te se ne razlikuje bitno od funkcija koje pišu korisnici.
Funkcija se definira pridruživanjem oblika

> *name* <- function(*arg_1, arg_2, …*) *expression*

expression je R-ov izraz, (obično neki grupirani izraz), koji upotrebljava argumente, arg_i, da bi se izračunala neka vrijednost. Vrijednost izraza je vrijednost koju nam daje funkcija.
Pozivanje funkcije tada obično ima oblik *name* (*expr_1, expr_2*, …) i može se upotrijebiti svugdje gdje je pozivanje funkcije legitimno.

10.1 Jednostavni primjeri

Kao prvi primjer, uzmimo funkciju za izračunavanje *t*-statistike za dva uzorka, pokazujući postupke korak po korak. To je, naravno, umjetan primjer, jer ima drugih, jednostavnijih načina za postizanje istoga rezultata.
Funkcija se definira kako slijedi:

```
> twosam <- function(y1, y2) {
    n1  <- length(y1) ; n2  <- length(y2)
    yb1 <- mean(y1) ;  yb2 <- mean(y2)
    s1  <- var(y1) ;   s2  <- var(y2)
    s <-  ((n1 – 1)*s1 + (n2-1)*s2)/(n1+n2-2)
    tst <- yb1 – yb2)/sqrt(s2*(1/n1 + 1/n2))
    tst
  }
```

Kad je ova funkcija definirana možete izvršiti *t*-testove za dva uzorka s pomoću poziva kao što je

```
> tstat <- twosam(data$male, data$female) ; tstat
```

Kao drugi primjer, uzmimo funkciju za direktno oponašanje MATHLAB "backslash" naredbe, koja vraća koeficijente ortogonalne projekcije vektora *y* na prostor stupca matrice, *X*. (To se obično naziva procjena najmanjih kvadrata koeficijenata regresije.) To se obično radi s pomoću funkcije qr( ); međutim, ponekad je malo riskantno upotrijebiti funkciju direktno, tako da vrijedi upotrijebiti jednostavnu funkciju kao što je sljedeća zbog sigurnosti upotrebe.
Tako, sa zadanim vektorom *y*, koji je *n* x 1, i matricom *X*, koja je *n* x *p*, *X y* se definira kao $(X'X)^{-}X'y$, gdje je $(X'X)^{-}$ generalizirana inverzija od *X'X*.

```
> bslash <- function(X, y) {
   X <- qr(X)
   qr.coef(X, y)
  }
```

Nakon što je taj objekt kreiran, on ostaje permanentan, kao i svi objekti, i može se upotrijebiti u iskazima poput

```
> regcoeff <- bslash(Xmat, yvar)
```

i tako dalje.
Klasična R-ova funkcija lsfit( ) dobro obavlja taj posao, i više od toga[1]. Ona izračunava naizmjence funkcije qr( ) i qr.coef( ) na ponešto neintuitivan način. Prema tomu, vrijedno je izdvojiti upravo taj dio u funkciju jednostavnu za rukovanje, ako će se često upotrebljavati. U tome slučaju, možemo od nje načiniti binarni operator za matrice zbog još pogodnije upotrebe.

10.2 Definiranje novih binarnih operatora

Da smo funkciji bslash( ) dali drukčiji naziv, to jest jedan naziv oblika

%*anything*%

mogli smo je upotrijebiti kao *binarni operator* u izrazima radije nego u obliku za funkciju. Pretpostavimo, na primjer, da izaberemo ! za unutrašnji znak. Tada bi definiranje funkcije počelo sa

```
> "%!%" <- function(X, y) {…}
```

(Obratite pozornost na upotrebu navodnih znakova.) Funkcija bi tada mogli biti upotrijebljena kao X%!%y. (Backslash simbol sam po sebi nije pogodan izbor jer u ovom kontekstu predstavlja posebne probleme.)
Operator za množenje matrice, %*%, i operator vanjskog produkta matrice, %o%, su drugi primjeri binarnih operatora definiranih na ovaj način.

[1] Pogledajte također metode opisane u Poglavlju 11 [Satistički modeli u R-u], str. 52.

10.3 Imenovani argumenti i predefinirane vrijednosti (default)

Kao što smo prije zabilježili u Poglavlju 2.3 Generiranje regularnih nizova str. 13, ako su argumenti pozvanih funkcija dati u obliku "*name=object*", mogu biti navedeni bilo kojim poretkom. Nadalje, niz argumenata može početi u neimenovanom, pozicionalnom obliku, i specificirati imenovane argumente nakon pozicionalnih argumenata.
Tako, ako imamo funkciju fun1 definiranu s pomoću

```
> fun1 <- function(data, data.frame, graph, limit) {
  [izostavljeno tijelo funkcije]
}
```

tada se funkcija može pozvati na nekoliko načina, na primjer

```
> ans <- fun1(d, df, TRUE, 20)
> ans <- fun1(d, df, graph=TRUE, limit=20)
> ans <- fun1(data=d, limit=20, graph=TRUE, data.frame=df)
```

koji su svi ekvivalentni.
U mnogim slučajevima argumentima se mogu dati opće odgovarajuće predefinirane vrijednosti, u kojemu slučaju ih se može potpuno izostaviti kod pozivanja kad su e vrijednosti odgovarajuće. Na primjer, da je fun1 definiran kao

```
> fun1 <- function(data, data.frame, graph=TRUE, limit=20) {…}
```

mogao bi biti pozvan kao

```
> ans <- fun1(d, df)
```

koji je sada ekvivalentan sa tri gore navedena slučaja, ili kao

```
> ans <- fun1(d, df, limit=10)
```

koji mijenja jednu od predefiniranih vrijednosti.
Važno je obratiti pozornost da predefinirane vrijenosti (default-i) mogu biti proizvoljni izrazi koji čak uključuju druge argumente iste funkcije; oni nisu ograničeni da budu konstante, kao što je bio slučaj u ovom našem jednostavnom primjeru.

10.4 Argument '…'

Drugi je česti zahtjev da se omogući prenošenje argumenata jedne funkcije drugoj funkciji. Na primjer, mnoge grafičke funkcije upotrebljavaju funkciju par( ) a funkcije poput plot( ) omogućuju korisniku prijenos grafičkih parametara na par( ) za kontrolu grafičkog rezultata. (Vidi Poglavlje 12.4.1 [Permanentne promjene: funkcija par( )], str. 74, za podrobnije detalje o funkciji par( ). To se može učiniti uključujući jedan dodatni argument funkcije, doslovno '…', koji se zatim može prenostiti dalje. Navodimo primjer

```
fun1 <- function(data, data.frame, graph=TRUE, limit=20, …) {
  [... izostavljene tvrdnje...]
  if (graph)
    par(pch="*", …)
  [...izostavljene tvrdnje...]
}
```

10.5 Pridruživanje unutar funkcija

Obratite pozornost da je ***svako obično pridruživanje koji se nalazi unutar funkcije je lokalno i privremeno, te se gubi nakon što se izađe iz funkcije***. Tako pridruživanje X <- qr(X) ne utječe na vrijednost argumenta u programu koji se poziva.
Da bi mogao u potpunosti razumjeti pravila koja upravljaju područjem R-ovih pridruživanja (assignment) čitatelj treba biti upoznat s pojmom "evaluation *frame"*. To je pomalo napredna ali ne i teška tema, i o tome ovdje neće biti više raspravljano.
Ako se namjerava imati globalne i premanentna pridruživanja unutar neke funkcije, može se upotrijebiti ili operator "superassignment-a", <<-, ili funkciju assign( ). Za detalje pogledajte help dokument. Korisnici S-PLUS-a trebaju biti svijesni da <<- ima u R-u drukčije značenje.
O tome se dalje raspravlja u Poglavlju 10.7 [Djelokrug i vidljivost varijabli], str. 53.

10.6 Napredniji primjeri

10.6.1 Faktori učinkovitosti u blok dizajnima

Kao kompletniji primjer funkcije, razmotrimo nalaženje faktora učinkovitosti za blok dizajn. (O nekim aspektima ovog problema je već bilo riječi u Poglavlju 5.3 [Indeksna polja], str. 23.)
Blok dizajn je definiran sa dva faktora, recimo blocks (b nivoi) i varieties (v nivoi). Ako su R i K, *replikati* od v sa v i b sa b, odnosno matrice *veličine bloka*, i ako je N matrica incidencije b sa v, tada su faktori učinkovitosti definirani kao svojstvene vrijednosti matrice

$$E = I_v - R^{-½} N'K^{-1} NR^{-½} = I_v - A'A,$$

gdje je $A = K^{-½}NR^{-½}$. Jedan način pisanja funkcije je sljedeći

```
> bdeff <- function(blocks, varieties) {
    blocks <- as.factor(blocks)                  # minor safety move
    b <- length(levels(blocks))
    varieties <- as.factor(varieties)            # minor safety move
    v <- length(levels(varieties))
    K <- as.vector(table(blocks))                # remove dim attr
    R <- as.vector(table(varieties))             # remove dim attr
    N <- table (blocks, varieties)
    A <- 1/sqrt(K) * N * rep(1/sqrt(R), rep(b, v))
    sv <- svd(A)
    list(eff=1 – sv$dˆ2, blockcv=sv$u, varietycv=sv$v)
}
```

U ovom slučaju, numerički je malo bolje raditi s rutinama za dekompoziciju singularne vrijednosti, nego s rutinama za svojstvene vrijednosti.
Rezultat funkcije je lista koja daje ne samo faktore učinkovitosti kao prvu komponentu, nego također kanoničke kontraste block i variety, jer oni ponekad daju dodatnu korisnu kvalitativnu informaciju.

10.6.2 Ispuštanje svih naziva u prikazu polja

Za prikaz velikih matrica ili polja, često ih je korisno prikazati u zbijenom obliku bez naziva ili brojeva u poljima. Uklanjanjem *dimnames* atributa neće se postići takav učinak, nego se radije polju mora dati neki *dimnames* atribut koji se sastoji od praznih nizova. Na primjer za prikaz matrice, X

```
> temp <- X
> dimnames(temp) <- list(rep("", nrow(X)), rep("", ncol (X))
> temp; rm(temp)
```

Isto se može mnogo spretnije učiniti s pomoću funkcije no.dimnames( ), kao što je prikazano niže u tekstu. Ovo također pokazuje kako neke učinkovite i korisne korisničke funkcije mogu biti sasvim kratke.

```
no.dimnames <- function(a) {
  ## Remove all dimension names from an array for compact printing.
  d <- list( )
   l <- 0
   for(i in dim(a)) {
     d[[1 <- 1 + 1]] <- rep("", i)
   }
   dimnames(a) <- d
   a
}
```

Kad je ova funkcija definirana, polje se može printati u zgusnutom formatu s pomoću

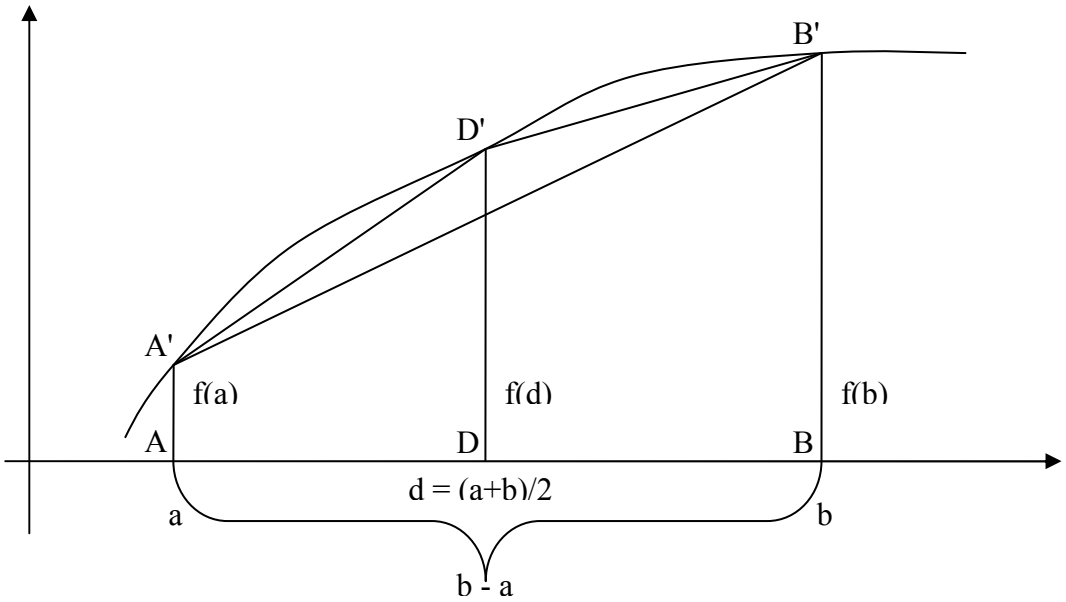
```
> no.dimnames(X)
```

Ovo je posebno korisno kod velikih polja s cjelim brojevima, gdje nas zapravo zanimaju više uzorci nego vrijednosti.

10.6.3 Rekurzivna numerička integracija

Funkcije mogu biti rekurzivne te one same mogu definirati funkcije unutar sebe. Obratite pozornost, međutim, da takve funkcije, ili varijable, nisu naslijeđene od pozvanih funkcija na višim nivoima što bi bio slučaj da se nalaze na pretražnom putu.

Niže je naveden pojednostavljeni primjer jednodimenzionalne numeričke integracije. Integrand se ocjenjuje na krajevima intervala i u sredini. Želimo izračunati integral funkcije f na intervalu [a,b]. Ideja je slijedeća. Prvo izračunamo površinu trapeza (a0) čiji su vrhovi točke A,B,B',A'. Nakon toga na sredini intervala [a,b], tj. u točki D povučemo vertikalnu liniju do grafa funkcije i dobijemo točku D'. Ako tu točku spojimo sa točkama A' i B' dobijemo dva trapeza. Trapez A,D,D',A' (čija je površina a1) i trapez D,B,B'D' (sa površinom a2). Pogledamo razliku površina između prvog trapeza i sume druga dva i ako je u granicama očekivanja (eps) onda je postupak gotov, a ako nije rekurzivno ponavljamo računanje na oba manja trapeza. Rezultat je prilagodljiv proces integracije koji je koristan u područjima gdje je integrand daleko od linearnosti. Ova funkcija je jedino kompetitivna s drugim algoritmima kada je integrand glatka krivulja koji je vrlo teško procijeniti.



Također se ovaj primjer djelomično navodi kao mala zagonetka u programiranju u R-u.

```
area <- function(f, a, b, eps = 1.0e-06, lim = 10) {
  fun1 <- function(f, a, b, fa, fb, a0, eps, lim, fun) {
    ## funkcija 'fun1' je vidljiva samo unutar 'area'
    d <- (a + b)/2
    h <- (b - a)/4
    fd <- f(d)
    a1<- h * (fa + fd)
    a2<- h * (fd + fb)
    if(abs(a0 – a1 – a2) < eps || lim = = 0)
      return(a1 + a2)
    else {
      return(fun(f, a, d, fa, fd, a1, eps, lim – 1, fun) +
          fun(f, d, b, fd, fb, a2, eps, lim – 1, fun))
    }
  }
  fa <- f(a)
  fb <- f(b)
  a0 <- ((fa + fb) * (b – a))/2
  fun1(f, a, b, fa, fb, a0, eps, lim, fun1)
}
```

10.7 Djelokrug i vidljivost varijabli

Rasprava u ovom poglavlju je više tehničke prirode nego u drugim djelovima ovog dokumenta. U njemu se, međutim, podrobno obrađuju glavne razlike između S-PLUS i R-a.
Simboli koji se upotrebljavaju u okviru neke funkcije mogu se podijeliti u tri klase; formalni parametri, lokalne varijable i slobodne varijable. Formalni parametri neke funkcije su parametri koji se nalaze u listi argumenata te funkcije. Njihove vrijednosti se određuju procesom vezivanja stvarnih argumenata funkcije na formalne parametre. Lokalne varijable su varijable čije se vrijednosti izračunavaju unutar funkcije. Varijable koje nisu formalni parametri niti lokalne varijable nazivaju se slobodne varijable. Slobodne varijable postaju lokalne ako su pridružene. Uzmimo definiranje sljedeće funkcije.

```
f <- function(x) {
  y <- 2*x
  print(x)
  print(y)
  print(z)
}
```

U ovoj je funkciji x formalni parametar, y je lokalna varijabla, a z slobodna varijabla.
U R-u se vezivanje slobodnih varijabla rješava na način da se prvo pogleda u okruženje u kojemu je funkcija kreirana. To se naziva *lexical scope*. Prvo definiramo funkciju nazvanu cube

.

```
cube <- function(n) {
   sq <- function( ) n*n
   n*sq( )
}
```

Varijabla n u funkciji sq nije argument te funkcije. Prema tomu, ona je slobodna varijabla, te treba upotrijebiti pravila vidljivosti da se utvrdi vrijednost koju bi joj se pridružilo. Po statičkoj definiciji (S-PLUS) to je globalna varijabla. Po leksičkoj definicji (R) vrijednost je parametar funkcije cube jer je to aktivno povezivanje varijable n u vrijeme definiranja funkcije sq. Razlika između pridruživanja u R-u i pridruživanja u S-PLUS je u tome što S-PLUS traži globalnu varijablu nazvanu n dok R prvo traži varijablu nazvanu n u okruženju kreiranom kada se funkcija cube izvršava.

```
## izvršavanje funkcije u S-u
S> cube(2)
Error in sq( ) : Object "n" not found
Dumped
S> n <- 3
S> cube(2)
[1] 18

## ista funkcija izvršena u R-u
R> cube(2)
[1] 8
```

Leksička vidljivost se može također upotrijebiti da se funkcijama dade promjenjivo stanje (*mutable state*). U sljedećem ćemo primjeru pokazati kako se R može upotrijebiti za imitiranje bankovnog računa. Pravovaljani bankovni račun treba imati stanje računa (balance), funkciju za podizanje sredstava (withdrawals), funkciju za ulaganje i funkciju za utvrđivanje aktualnog stanja računa. To postižemo kreiranjem tri funkcije unutar funkcije account i zatim vraćajući listu koja ih sadrži. Kad se pozove account on uzima numerički argument total i vraća listu koja sadrži navedene tri funkcije. Budući da su te funkcije definirane u okruženju koje sadrži total, one će imati pristup vrijednosti totala.
Posebni operator za pridruživanje, <<-, upotrebljava se za promjenu vrijednosti pridodane totalu. Ovaj operator gleda natrag u okolna okruženja za okruženje koje sadrži simbol total, i kada nađe takvo okruženje zamjenjuje vrijednost, u tom okruženju sa vrijednošću na desnoj strani. Ako se globalno ili okruženje najvišeg nivoa dostigne bez nalaženja simbola total, onda se ta varijabla kreira i pridodaje tom okruženju. Za većinu korisnika <<- kreira globalnu varijablu i prirodaje joj vrijednost na desnoj strani[2]. Jedino kada se <<- upotrebljava u funkciji

[2] Na neki način to imitira ponašanje u S-PLUS jer ovaj operator u S-PLUS-u uvijek kreira globalnu varijablu ili pridodaje globalnoj varijabli neku vrijednost.

koja je vraćena kao vrijednost neke druge funkcije ponašat će se na poseban način, niže opisan.

```
open.account <- function(total) {
    list(
        deposit = function(amount) {
            if(amount <= 0)
            stop("Deposits must be positive!\n")
            total <<- total + amount
            cat(amount, "deposited. Your balance is", total, "\n\n")
        },
        withdraw = function(amount) {
            if(amount > total)
            stop("You don't have that much money!\n")
            total <<- total - amount
            cat(amount, "withdrawn. Your balance is", total, "\n\n")
        },
        balance = function() {
        cat("Your balance is", total, "\n\n")
        }
    )
}

ross <- open.account(100)
robert <- open.account(200)
ross$withdraw(30)
ross$balance( )
robert$balance( )

ross$deposit(50)
ross$balance( )
ross$withdraw(500)
```

10.8 Prilagođavanje okruženja

Korisnici mogu prilagoditi okruženje na nekoliko raznih načina. Postoji datoteka za inicijaliziranje stranice i svaki direktorij može imati svoju posebnu datoteku za inicijaliziranje. Konačno, mogu se upotrijebiti posebne funkcije .First i .Last.
Lociranje datoteke za inicijaliziranje stranice je uzeto iz vrijednosti varijable R_PROFILE. Ako ta varijabla nije definirana, upotrebljava se datoteka 'Rprofile.site' u R home poddirektoriju 'etc' (Unix !). Ta datoteka bi trebala sadržavati naredbe koje želite izvršavati svaki put kad startate R na svom računalu. Druga, personalni profile datoteka nazvana '.Rprofile'[3], može se smjestiti u bilo koji direktorij. Ako se R pozove u tome direktoriju tada će ta datoteka biti učitana i izvršena. Ta datoteka daje pojedinačnim korisnicima kontrolu nad njihovim radnim prostorom i omogućuje različite postupke započinjanja u raznim radnim direktorijima. Ako se u startup direktoriju ne nađe niti jedna '.Rprofile' datoteka, tada R traži takvu datoteku u korisnikovom home direktoriju i upotrijebi ga (ako postoji).

Bilo koja funkcija nazvana .First( ) u bilo kojem od dvije profile datoteke ili u '.RData' slici ima poseban status. Ona se automatski izvršava na početku rada u R-u na računalu i može se upotrijebiti za inicijaliziranje okruženja. Na primjer, definiranje u niže navedenom primjeru

[3] Tako da se ne vidi u UNIX-u.

mijenja prompt u $ i definira razne druge korisne stvari koje se mogu uzeti kao sigurne u preostalom tijeku rada na računalu.
Tako je slijed u kojem se datoteke izvršavaju ‘Rprofile.site’, ‘.Rprofile’, ‘.RData’ i zatim .First( ). Definiranje u kasnijim datotekama će maskirati definiranja u ranijim datotekama.
Primjer datoteke koja sa prva izvršava (očito, ovdje se govori o Unix okruženju):

```
> .First <- function( ) {
   options(prompt="$ ", continue="+\t")        # $ is je  prompt
   options(digits=5, length=999)               # format brojeva
   x11( )                                      # grafika
   par(pch = "+")                              # simbol za crtanje točaka kod grafova
   source(file.path(Sys.getenv("HOME"), "R", "mystuff.R")
                                               # korisnikove funkcije
   library(MASS )                              # pridruži paket MASS
}
```

Slično tomu se funkcija .Last( ), ako je definirana, izvršava na samom kraju rada sa R-om. Navodimo sljedeći primjer

```
> .Last <- function( ) {
   graphics.off( )                  # mala sigurnosna mjera
   cat(paste(date( ), "\nAdios\n"))  # Idemo na ručak?
}
```

10.9 Klase, generičke funkcije i objektna orijentacija

Klasa nekog objekta određuje kako će objekt biti tretiran od strane *generičkih* funkcija. Obrnuto tomu, generička funkcija obavlja zadaću ili aktivnost na njenim argumentima *koja je specifična za klasu samoga argumenta*. Ako argument nema atribut klase ili ako ima klasu koju specifično ne opskrbljuje dotična generička funkcija, uvijek stoji na raspolaganju *predefinirana aktivnost*.
Jedan primjer će to pojasniti. Mehanizam klase nudi korisniku mogućnost dizajniranja i pisanja generičkih funkcija za posebne svrhe. Među drugim generičkim funkcijama su plot( ) za grafičko prikazivanje objekata, summary( ) za sumiranje analiza različitih vrsta, i anova( ) za uspoređivanje statističkih modela.
Broj generičkih funkcija koje mogu tretirati neku klasu na specifičan način može biti dosta velik. Na primjer, funkcije koje mogu biti pogodne na neki način za objekte klase "data.frame" uključuju

```
[     [[<-      any      as.matrix
[<-    model    plot     summary
```

Trenutno kompletnu listu može se dobiti s pomoću funkcije methods( )

```
> methods(class="data.frame")
```

Obrnuto, broj klasa kojima neka generička funkcija može rukovati također može biti dosta velik. Na primjer, funkcija plot( ) ima predefiniranu metodu i varijante za objekte klasa "data.frame", "density", "factor" i još više njih. Kompletnu listu se, ponovno, može dobiti s pomoću funkcije methods( ):

```
> methods(plots)
```

Čitatelj se upućuje da pogleda službene reference za kompletnu raspravu o ovom mehanizmu.

11 Statistički modeli u R-u

U ovome poglavlju se polazi od pretpostavke da čitatelj posjeduje određeno znanje iz statističke metodologije, pogotovo što se tiče analize regresije i analize varijance. Kasnije polazimo od malo ambicioznijih pretpostavki, naime, da čitatelj ima neka saznanja o generaliziranim linearnim modelima i nelinearnoj regresiji.
Zahtjevi za prilagođavanje statističkim modelima su relativno dobro definirani da omoguće konstruiranje općenitih alata koji se upotrebljavaju u širokom spektru problema.
R daje međusobno spojeni niz alata koje čine prilagođavanje statističkim modelima veoma jednostavnim. Kako smo napomenuli u uvodu, osnovni ulaz je minimalan, a za pojedinosti je potrebno pozvati ekstraktorske funkcije (navedi poglavlje).

11.1 Definiranje statističkih modela; formule

Osnovni statistički model je model linearne regresije s homogenim (homoscedastic) pogreškama (jednakim pogreškama u y za bilo koji x)

$$y_i = \sum_{j=0}^{p} \beta_j x_{ij} + e_i \ , \qquad e_i \sim \text{NID}\ (0,\sigma^2), \qquad i = 1,\ldots, n$$

Ili u matričnoj terminologiji

$$y = \boldsymbol{X}\beta + e$$

gdje je *y* zavisni vektor, ***X*** je *modelna matrica* ili *dizajn matrica* koja ima stupce $x_0,x_1,\ldots,x_p$, kao određujuće varijable. Vrlo često će x_0 biti stupac koji definira neki izraz za odsječak.

Primjeri

Prije formalne specifikacije, nekoliko primjera može pokazati o čemu se radi.
Pretpostavimo da su y, x, x0, x1, x2,… brojčane varijable, da je **X** matrica a A, B, C,… faktori. Niže navedene formule na lijevoj strani specificiraju statističke modele opisane na desnoj strani.

y ~ x
y ~ 1 + x — Oboje podrazumijevaju isti model jednostavne linearne regresije *y*-a na *x*-u. Prvi oblik ima implicitni izraz, a drugi eksplicitni.

y ~ 0 + x
y ~ -1 + x
y ~ x – 1 — Jednostavna linearna regresija *y*-a na *x*-u kroz ishodišre (to jest bez člana koji oynačuje presjecište).

log(y) ~ x1 + x2
Višestruka regresija transformirane varijable, log(*y*), na *x*1 i *x*2 (s implicitnim presjecištem).

y ~ poly(x,2)
y ~ 1 + x + I(x^2)
Polinomna regresija *y*-a na *x*-u drugoga stupnja. Prvi oblik koristi pravokutne polinome, a drugi eksplicitne potencije, kao osnovu.

y ~ X + poly(x,2)
Višestruka regresija *y*-a s modelnom matricom koja se sastoji od matrice *X*

kao i polinomnih izraza u *x*-u do drugoga stupnja.

y ~ A — Jedna klasifikacijska analiza modela varijance *y*-a, s klasama određenim s pomoću A.

y ~ A + x — Jedna klasifikacijska analiza modela kovarijance *y*-a, s klasama određenim s pomoću A, i sa slučajnom kovarijablom *x*.

y ~ A*B
y ~ A + B + A:B
y ~ B%in%A
y ~ A/B — Dvofaktorski neaditivni model *y*-a na A i B. Prva dva oblika specificiraju istu unakrsnu klasifikaciju, a druga dva oblika specificiraju istu vezanu klasifikaciju. U apstraktnom smislu sva četiri oblika specificiraju isti modelni podprostor.

y ~ (A + B + C)^2
y ~ A*B*C – A:B:C
Eksperiment s tri faktora ali s modelom koji sadrži glavne učinke i interakcije samo dva faktora. Obje formule specificiraju isti model.

y ~ A * x
y ~ A/x
y ~ A/(1 + x) – 1
Odvojeni modeli jednostavne linearne regresije *y*-a na *x*-u unutar nivoa od *A*, s različitim kodovima. Posljednji oblik daje eksplicitne procjene onoliko različitih intercepata i nagiba koliko *A* ima nivoa.

y ~ A*B + Error(C)
Eksperiment sa dva tretirajuća faktora, *A* i *B*, i slojevima pogreške koje određuje faktor *C*. Primjer je eksperiment rastavljenog grafikona, s cijelim grafikonom (pa prema tome i grafikonima na koje se rastavlja) određenim s pomoću faktora *C*.

Operator ~ se upotrebljava za definiranje *formule za model* u R-u. Za obični linearni model oblik je sljedeći

response ~ *op_1 term_1 op_2 term_2 op_3 term_3*…

gdje je

response — vektor ili matrica, (ili izraz koji ocjenjuje vektor ili matricu) koji definira varijablu/varijable odgovora.
op_i — je operator, bilo + ili -, koji implicira uključenost ili isključenost izraza u model, (prvi je prema želji).
term_i — je jedno od sljedećeg
- vektor ili matrica, ili 1,
- faktor, ili
- *izraz u obliku formule* koji se sastoji od faktora, vektora ili matrica povezanih s pomoću *operatora formule*.

U svim slučajevima svaki izraz definira kolekciju stupaca koje treba dodati ili odstraniti iz modelne matrice. Brojka 1 stoji umjesto stupca presjecišta te je predefinirano uključena u modelnu matricu osim ako se izričito ne ukloni.

Operatori formule su po učinku slični Wilkinsonovom i Rogersovom sustavu znakova koji se upotrebljavaju u programima kao što su Glim i Genstat. Neizbježna je promjena što operator '.' postaje ':' jer je točka u R-u punovrijedni znakovni naziv.
Sustav znakova je sažet kako slijedi (temeljeno na Chambers & Hastie, 1992, p.29):

Y ~ M	*Y* se modelira kao *M*.
M_1 + M_2	Uključuje *M_1* i *M_2*.
M_1 - M_2	Uključuje *M_1* izostavljajući izraze u *M_2*.
M_1 : *M_2*	Tenzorski produkt *M_1* i *M_2.*
M_1 %in% M_2	Slično kao *M_1*:*M_2,* ali s drukčijim kodiranjem.
*M_1 * M_2*	*M_1 + M_2 + M_1*:*M_2.*
M_1 / M_2	*M_1 + M_2%in% M_1.*
M^n	Svi izrazi u *M* zajedno s "interakcijama" sve do *n*-tog reda
I(*M*)	Izolira *M*. Unutar *M*-a svi operatori imaju svoje normalno aritmetičko značenje i takav izraz se pojavljuje u modelnoj matrici.

Obratite pozornost da unutar zagrada koje obično zatvaraju argumente funkcije svi operatori imaju svoje normalno aritmetičko značenje. Funkcija I() je funkcija identiteta koja se upotrebljava jedino da omogući terminima u modelnoj formuli da budu definirani upotrebom aritmetičkih operatora.
Osobito obratite pozornost da modelne formule specificiraju *stupce modelne matrice*, a specifikacija parametara je implicitna. To nije slučaj u drugim kontekstima, na primjer kod specificiranja nelinearnih modela.

11.1.1 Kontrasti

Potrebno je barem osnovno poznavanje načina kako modelne formule specificiraju stupce modelne matrice. To je lako ako imamo neprekidne varijable jer svaka od njih daje jedan stupac u modelnoj matrici (a presjecište (intercept) će dati stupac varijabla ako je uključen u model).
Što je u slučaju faktora A sa *k*-nivoa? Odgovor na to pitanje je različit za neporedane i poredane faktore. Za *neporedane* faktore, *k* – 1 stupci se generiraju za indikatore drugog, …, *k*-tog nivoa faktora. (Tako implicitna parametrizacija treba odgovor na svakom nivou staviti u kontrast prema odgovoru na prvom nivou.) Kod *poredanih* faktora, *k* – 1 stupci su ortogonalni polinomi na 1, …,*k*, izostavljajući konstantni izraz.
Iako je odgovor već sada složen, to nije sve. Prvo, ako se član presjecišta izostavi u modelu koji sadržava faktorski član, prvi takav član se kodira u *k* stupce, dajući indikatore za sve nivoe. Drugo, cijelo ponašanje se može promijeniti uspostavljanjem opcija za kontraste. Po pretpostavci (defaultu) se te opcije u R-u uspostavljaju kako slijedi

```
options(contrasts = c("contr.treatment", "contr.poly"))
```

Glavni razlog što to spominjemo je zbog toga što R i S imaju različite predefinirane vrijednosti za neporedane faktore jer S upotrebljava Helmert kontraste. Tako, ako vaše rezultate trebate usporediti s rezultatima priručnika ili rada kolega koji su koristili S-PLUS, trebate staviti:

```
options(contrasts = c("contr.helmert", "contr.poly"))
```

Ta je razlika namjerno načinjena jer se kontrasti tretmana (R-ova predefinirana vrijednost) smatraju jednostavnijima za interpretiranje od strane početnika.
To još uvijek nije sve, jer se shema kontrasta koja će se upotrebljavati može namiještati za svaki izraz u modelu s pomoću funkcija contrasts i C.

Još nismo razmotrili interakcijske članove: oni generiraju produkte stupaca unesene za izraze od kojih se stupci sastoje.
Iako su pojedinosti složene, formule modela u R-u će generirati modele koje bi stručnjak statističar očekivao, pod uvjetom da je marginalnost sačuvana. Na primjer, usklađivanje modela s interakcijom ali ne s odgovarajućim glavnim učincima uglavnom će dati iznenađujuće rezultate, i isključivo je predmet za stručnjake.

11.2 Linearni modeli

Osnovna funkcija za usklađivanje običnih višestrukih modela je lm(), a zadnja verzija poziva je sljedeća:

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

Na primjer

> fm2 <- im(y ˜ x1 + x2, data = production)

bi pristajalo modelu višestruke regresije *y*-a na *x*1 i *x*2 (s implicitnim izrazom za presjecište). Važan, ali tehnički proizvoljan, parametar data = production specificira da bilo koja varijabla potrebna za konstruiranje modela treba doći najprije iz production *data.frame-a. To vrijedi bez obzira je li data frame* production *priključen na pretražni put ili nije.*

11.3 Generičke funkcije za ekstrahiranje informacija o modelu

Vrijednost od lm() je prilagođeni modelni objekt; tehnički je to lista rezultata klase "lm". Informacija o prilagođenom modelu se može prikazati, ekstrahirati, ucrtati itd s pomoću generičkih funkcija koje se orijentiraju na objekte klase "lm". To su funkcije

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

Niže u tekstu se navodi kratak opis najčešće upotrebljavanih generičkih funkcija.

anova(*object_1, object_2*)
: Uspoređuje podmodel s nekim vanjskim modelom i producira analizu tablice varijance.

coefficients(*object*)
: Ekstrahira koeficijent regresije (matricu).
Kratki oblik: coef(*object*).

deviance(*object*)
: Rezidualni zbroj kvadrata, po potrebi ponderiran.

formula(*object*)
: Ekstrahira formulu modela.

plot(*object*)
: Producira četiri crteža, pokazujući ostatke (reziduals), prilagođene vrijednosti i dijagnoze.

predict(*object,* newdata=*data.frame*)
: Dobiveni *data frame* mora imati varijable specificirane s istim oznakama kao original. Vrijednost je vektor ili matrica predviđenih vrijednosti koje odgovaraju vrijednostima varijabli u *data.frame-u*.

print(*object*)
: Prikazuje konciznu verziju objekta. Najčešće se upotrebljava implicitno.

residuals(*object*)
Ekstrahira (matricu) ostataka, po potrebi ponderiranih.
Krati oblik: resid(*object*).

step(*object*)
Odabire pogodan model dodavanjem ili ispuštanjem izraza uz poštivanje hijerarhije. Model s najvećom vrijednosti AIC (Akaike's Information Criterion) pronađenom stepenastim traženjem dobiva se natrag.

summary(*object*)
Prikazuje opsežni sažetak rezultata analize regresije.

11.4 Analiza varijance i usporedba modela

Funkcija za prilagođavanje modela, aov(formula, data=*data.frame*), djeluje na najjednostavnijem nivou na vrlo sličan način kao funkcija lm(), i primjenjuje se većina generičkih funkcija navedenih u tablici u Poglavlju 11.3 [Generičke funkcije za ekstrahiranje informacija o modelu], str. 60.
Treba napomenuti da osim navedenoga, funkcija aov() omogućuje analizu modela s višestrukim slojevima pogreške kao što su eksperimenti s razdvojenim crtežima, ili balansirani nekompletni blok dizajni s povratom izmeđublokovske informacije. Formula modela:

response ~ *mean.formula* + Error(*strata.formula*)

specificira višeslojni eksperiment sa slojevima pogreške definiranim s pomoću *strata.formule*. U najjednostavnijem slučaju *strata.formula* je jednostavno faktor onda kada definira eksperiment sa dva sloja, naime sloj između i sloj unutar nivoa faktora.
Na primjer, sa faktorima svih određujućih varijabla, formula modela kakva je u

```
> fm <- aov(yield ~ v + n*p*k + Error(farms/blocks), data=farm.data)
```

tipično bi bila upotrijebljena za opis eksperimenta sa srednjim modelom v + n*p*k i tri sloja pogreške, naime među farmama, unutar farmi, među blokovima i unutar blokova.

11.4.1 ANOVA tablice

Obratite pozornost, da se analiza tablica (ili tablice) varijance upotrebljava za niz prilagođenih modela. Prikazani zbrojevi kvadrata su smanjenji u odnosu na rezidualne zbrojeve kvadrata, što proizlazi iz uvrštavanja određeneog izraza u modelu na određeno mjesto u nizu. Odatle će jedino za ortogonalne eksperimente poredak uvrštavanja biti bez važnosti.
Za višeslojne eksperimente najprije se projicira odgovor na slojeve pogreške, ponovno u niz, i da se usrednjeni model uskladi sa svakom projekcijom. Za više pojedinosti pogledajte Chambers & Hastie (1992).
Mnogo fleksibilnija alternativa od predefinirane pune ANOVA tablice je uspoređivanje direktno dva ili više modela s pomoću funkcije anova( )

```
> anova(fitted.model.1, fitted.model.2, …)
```

Dobijemo prikaz ANOVA tablice koja pokazuje razlike između usklađenih modela kada su usklađeni u nizu. Usklađeni modeli koji se uspoređuju obično bi bili, naravno, hijerarhijski niz. To ne daje drukčiju informaciju default-u, ali ju čini lakšom za razumijevanje i kontrolu.

11.5 Ažuriranje usklađenih modela

Funkcija update() je u mnogo pogodnija funkcija koja omogućuje usklađivanje modela koji se razlikuje od prethodno usklađenoga modela obično u samo nekoliko dodanih ili uklonjenih izraza. Oblik te funkcije je

> *new.model* <- update(*old.model*, *new.formula*)

U *new.formula* posebna oznak koja se sastoji samo od jedne točke '.' može se upotrijebiti da zamjeni "odgovarajući dio formule staroga modela". Na primjer

```
> fm05 <- 1m(y ~ x1 + x2 + x3 + x4 + x5, data = production)
> fm6    <- update(fm05, . ~ . + x6)
> smf6 <- update(fm6, sqrt(.) ~ .)
```

bi uskladilo višestruku regresiju pet slučajnih varijabli s varijablama (pretpostavljeno) iz data frame production, također bi uskladilo dodatni model koji uključuje šestu regresijsku varijablu, i uklopilo u model drugi korijen.
Obratite pozornost posebno da se, u slučaju ako je data=argument specificiran u izvornom pozivanju funkcije za usklađivanje modela, ta informacija prenosi dalje preko objekta prilagođenog modela na update( ) i njegove istoimenike (allies).
Naziv '.' se može također koristiti u drugim kontekstima ali s malo drukčijim značenjem. Na primjer

```
> fmfull <- 1m(y ~ . , data = production)
```

bi uskladilo model s odgovorom y i svim ostalim regresijskim varijablama u data frame-u "production".
Druge funkcije za istraživanje inkrementalnih nizova modela su add1( ), drop1( ) i step( ). Nazivi tih funkcija govore o njihovoj namjeni, ali za potpunu informaciju pogledajte *on-line help.*

11.6 Generalizirani linearni modeli

Generalizirano linearno modeliranje predstavlja razvoj linearnih modela kako bi se distribucije i transformacije nenormalnog odgovora prilagodile linearnosti na uredan i izravan način. Generalizirani linearni model može se opisati u terminima sljedećih pretpostavki:
● Postoji odgovor *y*, koji je od interesa, te poticajne varijable x_1 i x_2, … , čije vrijednosti utječu na distribuciju varijable y.
● Poticajne varijable utječu na distribuciju *y*-a kroz *samo jednu linearnu funkciju*. Ta linearna funkcija se naziva *linearni prediktor*, i obično se piše

$$\eta = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p,$$

prema tome x_i nema utjecaja na distribuciju *y*-a ako je $\beta_i = 0$, i jedino u tom slučaju.
● Distribucija *y*-a ima oblik

$$f_Y(y;\mu,\varphi) = \exp\left[\frac{A}{\varphi}\{y\lambda(\mu) - \gamma(\lambda(\mu))\} + \tau(y,\varphi)\right]$$

gdje je φ *parametar skale* (možda poznat), i koji je konstantan za sva opažanja, *A* je prethodna težina, pretpostavljena kao poznata ali koja može varirati s opažanjima, a μ je srednja vrijednost varijable *y*. Tako se pretpostavlja da je distribucija varijable *y* određena njegovom sredinom a također možda i parametrom skale.

● Sredina, μ, je poravnata invertibilna funkcija linearnog prediktora:

$$\mu = m(\eta), \eta = m^{-1}(\mu) = \ell(\mu)$$

i ta inverzna funkcija, $\ell(\ )$, se naziva *link funkcija*.
Ove pretpostavke su dostatno labave da uključe široku klasu modela korisnih u statističkoj praksi, a opet dostatno čvrste da dozvole razvoj unificirane metodologije procjene i zaključivanja, barem približno. Čitatelj se upućuje na bilo koju referencu suvremene literature iz tog područja za dobivanje potpunih podataka, na primjer McCullagh & Nelder (1989) ili Dobson (1990).

11.6.1 Familije generaliziranih linearnih modela

Klasa generaliziranih linearnih modela kojima se rukuje s pomoću R-ovih metoda uključuje gausovu, binomnu, binomnu, inverznu gausovu i gama distribuciju slučajne varijable, te također modele *kvazi-vjerojatnosti* gdje distribucija nije eksplicitno specificirana. U ovom drugom slučaju, funkcija varijance mora biti specificirana kao funkcija sredine, ali u drugim slučajevima je ova funkcija implicirana u distribuciji.
Svaka distribucija dopušta različite funkcije povezivanja za spajanje sredine s linearnim prediktorom. One funkcije koje su automatski dostupne prikazane su u niže navedenoj tablici.

Family name	Link functions
binomial	logit, probit, cloglog
gaussian	identity
Gamma	identity, inverse, log
inverse.gaussian	1/muˆ2
poisson	identity, log, sqrt
quasi	logit, probit, cloglog, identity, inverse, log, 1/muˆ2, sqrt

Kombinacija distribucija, funkcije povezivanja i raznih drugih informacija potrebnih za provođenje vježbe modeliranja naziva se *family* – *porodica* generaliziranog linearnog modela.

11.6.2 Funkcija glm( )

Budući da distribucija ovisne varijable ovisi o poticajnim varijablama kroz *samo* jednu linearnu funkciju, jednaki mehanizam koji je bio upotrijebljen za linearne modele može se također upotrijebiti za specificiranje linearnog dijela generaliziranog modela. Porodica mora biti specificirana na drukčiji način.
R-ova funkcija za usklađivanje generaliziranog linearnog modela je glm( ) koja ima oblik

> *fitted.model* <- glm(*formula*, family=*family.generator*, data=*data.frame*)

Jedina nova karakteristika je *family.generator*, koji je instrument s pomoću kojega se porodica opisuje. To je naziv funkcije koja generira listu funkcija i izraza koji zajedno definiraju i kontroliraju model i proces procjene. Iako na prvi pogled može izgledati malo složeno, upotreba te funkcije je prilično jednostavna.
Nazivi standardnih raspoloživih generatora familija modela su navedeni pod "Nazivi familije" u tablici u Poglavlju 11.6.1 [Familije generaliziranih linearnih modela], str. 63. Gdje postoji izbor linkova, naziv linka se može također dobiti s nazivom familije, u zagradama kao parametar. U slučaju kvazi familije, funkcija varijance se može također specificirati na ovaj način.
Neki primjeri pojašnjavaju taj proces.

Gaussova familija

Pozivom poput

```
> fm <- glm(y ~ x1 + x2, family = gaussian, data = sales)
```

postiže se isti rezultat kao s pomoću

```
> fm <- lm(y ~ x1+x2, data=sales)
```

ali mnogo manje učinkovito. Obratite pozornost da Gaussova familija nema automatski izbor linkova, pa nije dozvoljen niti jedan parametar. Ako problem iziskuje Gaussovu familiju s nestandardnim linkom, to se obično može postići putem kvazi familije, kako ćemo vidjeti kasnije.

Binomna familija

Razmotrimo mali, umjetni primjer, iz Silvery (1970).
Na otoku Kalythos u Egejskom moru muško stanovništvo ima prirođenu očnu bolest, čiji se simptomi s godinama života sve jače manifestiraju. Uzorci muškog stanovništva otoka razne životne dobi ispitani su na sljepoću te su zabilježeni rezultati ispitivanja. Podaci su prikazani dolje u tekstu

Godine starosti:	20	35	45	55	70
Broj testiranih:	50	50	50	50	50
Broj slijepih:	6	17	26	37	44

Problem s kojim se ovdje susrećemo je uskladiti oba, logistički i probit model, s ovim podacima te procijeniti za svaki model LD50, to jest životnu dob kod koje su izgledi sljepoće za muškog stanovnika 50%.
Ako je *y* broj slijepih u životnoj dobi *x*, a *n* broj ispitanika, oba modela imaju oblik

$$y \sim \mathrm{B}(n, F(\beta_0 + \beta_1 x))$$

gdje je funkcija standardne normalne distribucije za probit model $F(z) = \Phi(z)$, a za logit model (default) $F(z) = e^z/(1 + e^z)$. U oba slučaja LD50 je

$$\mathrm{LD50} = \beta_0/\beta_1$$

to jest, točka u kojoj je argument funkcije distribucije nula.
Prvi je korak urediti podatke kao *data frame*

```
> kalythos <- data.frame(x = c(20,35,45,55,70), n = rep(50,5),
                         y = c(6,17,26,37,44))
```

Za usklađivanje binomnog modela korištenjem glm( ) postoje dvije mogućnosti odgovora:
● Ako je odgovor *vektor*, pretpostavlja se da sadržava *binarne* podatke, pa tako mora biti 0/1 vektor.
● Ako je odgovor *matrica s dva stupca*, pretpostavlja se da prvi stupac sadržava broj uspjeha u ispitivanju, a drugi stupac broj promašaja.
Ovdje trebamo ovu drugu konvenciju, tako da našem data frame-u dodajemo matricu:

```
> kalythos$Ymat <- cbind(kalythos$y, kalythos$n – kalythos$y)
```

Za usklađivanje modela koristimo

```
> fmp <- glm(Ymat ~ x, family = binomial(link=probit), data = kalythos)
> fm1 <- glm(Ymat ~ x, family = binomial, data = kalythos)
```

Budući je logit link default, parametar se može izostaviti kod drugog pozivanja. Da bismo vidjeli rezultate svakog usklađivanja možemo upotrijebiti

```
> summary(fmp)
> summary(fm1)
```
Oba modela su (i previše) dobro usklađena. Da bismo našli LD50 procjenu možemo upotrijebiti jednostavnu funkciju:
```
> ld50 <- function(b) -b[1]/ b[2]
> ldp <- ld50(coef(fmp)); 1dl <- ld50(coef(fml)); c(ldp, ldl)
```
Stvarne procjene iz tih podataka su 43.663 godine odnosno 43.601 godina.

Poissonovi modeli

Kod Poissonove familije predefinirani link je log, a u praksi se ova familija uglavnom koristi za prilagođavanje surogatnih Poissonovih nelinearnih modela s podacima učestalosti, čija je stvarna distribucija često multinomna. Ovo je velika i važna tematika o kojoj ovdje nećemo više raspravljati. Ona čak predstavlja glavni dio upotrebe sveukupnih ne-gausovih generaliziranih modela.
Ponekad se istinski Poissonov podatak pojavi u praksi, i u prošlosti je često bio analiziran kao gausovski podatak nakon log transformacije ili nakon transformacije drugog korijena. Kao elegantna alternativa ovog drugog slučaja, Poissonov generalizirani linearni model može se prilagoditi kao u niže navedenom primjeru:

```
> fmod <- glm(y ~ A + B + x, family = poisson(link=sqrt),
      data = worm.counts)
```

Modeli kvazi-vjerojatnosti

Za sve porodice, varijanca odgovora će ovisiti o sredini i imat će parametar skale kao multiplikator. Oblik ovisnosti varijancije o sredini je karakteristika distribucije odgovora; na primjer za Poissonovu distribuciju Var[y] = μ.
Za procjenu i zaključak o kvazi-vjerojatnosti nije specificirana precizna distribucija odgovora nego radije samo funkcija povezivanja i oblik funkcije varijance jer ovisi o sredini. Budući procjena kvazi-vjerojatnosti koristi formalno identične tehnike kao Gaussova distribucija, ova porodica daje način prilagođavanja Gaussovih modela s nestandardnim funkcijama povezivanja ili funkcijama varijance slučajno.
Na primjer, razmotrimo prilagođavanje nelinearne regresije

$$y = \frac{\theta_1 z_1}{z_2} + e$$

koje se može alternativno napisati kao

$$y = \frac{1}{\beta_1 \chi_1 + \beta_2 \chi_2} + e$$

gdje je $x_1 = z_2/z_1$, $\chi_2 = -1/\chi_1$, $\beta_1 = 1/\theta_1$ a $\beta_2 = \theta_2/\theta_1$. Uz pretpostavku da se načini odgovarajući *data frame*, mogli bismo prilagoditi ovu nelinearnu regresiju kao
```
> nlfit <- glm(y ~ x1 + x2 – 1,
      family = quasi(link=inverse, variance=constant),
      data = biochem)
```
Čitatelj se upućuje da po potrebi konzultira priručnik i *help* document za dalje informacije.

11.7 Nelinearna metoda najmanjih kvadrata i modeli maksimalne vjerojatnosti

Određeni oblici nelinearnog modela mogu se prilagoditi s pomoću Generaliziranih Linearnih Modela (glm()). No, u većini slučajeva moramo pristupiti problemu prilagođavanja nelinearne krivulje kao problemu nelinearne optimizacije. Rutina nelinearne optimizacije u R-u je nlm( ), koja zamjenjuje ms( ) i nlmin( ) u S-PLUS-u. Tražimo vrijednosti parametra koje minimiziraju neki indeks nedostatka prilagodbe, i nlm( ) to obavlja iterativnom probom raznih vrijednosti parametra. Za razliku od linearne regresije, na primjer, nema garancije da će postupak konvergirati u zadovoljavajuće procjene. nlm( ) iziskuje početna nagađanja o tome koje vrijednosti parametra bi trebalo probati, i konvergencija bitno ovisi o kvaliteti početnih nagađanja.

11.7.1 Najmanji kvadrati

Jedan način prilagođavanja nelinearnog modela je minimiziranjem zbroja kvadrata pogrešaka (SSE) ili ostataka. Ova metoda ima smisla ako su opažene pogreške mogle s vjerojatnošću nastati iz normalne distribucije.

Navodimo primjer iz Bates & Watts (1988), str. 51. Podaci su sljedeći:

```
    > 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 koji treba prilagoditi je:

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

Da bi izvršili usklađivanje trebamo početne procjene parametara. Jedan način nalaženja razumnih početnih vrijednosti je da se podaci nacrtaju, pogode neke vrijednosti parametara i navede modelna krivulja korištenjem tih vrijednosti.

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

Mogli bismo učiniti i bolje, ali ove početne vrijednosti 200 i .1 izgledaju odgovarajuće. Sada izvršimo prilagođavanje:

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

Nakon prilagođavanja, out$minimum je SSE, a out$estimates su procjene najmanjih kvadrata parametara. Da bismo dobili približne standardne pogreške (SE) procjena činimo sljedeće:

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

Brojka 2 u gornjem redu predstavlja broj parametara. 95%-tni interval povjerenja bio bi procjena parametra ± 1.96 SE. Možemo dodadi usklađivanje najmanjih kvadrata na novom grafikonu :

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

Standardni paket **nls** daje mnogo šire mogućnosti za prilagođavanje nelinearnih modela s pomoću najmanjih kvadrata. Model koji smo upravo prilagodili je model Michaelis-Menten, tako možemo upotrijebiti:

```
> 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
```

11.7.2 Maksimalna vjerojatnost

Maksimalna vjerojatnost je metoda prilagođavanja nelinearnog modela koja se primjenjuje čak i onda kad pogreške nisu raspodjeljene normalno. Metoda nalazi vrijednosti parametara koje maksimiziraju log vjerojatnost, odnosno ekvivalentno tome, one vrijednosti koje minimiziraju negativnu log-vjerojatnost. Ovdje navodimo primjer iz Dobson (1990), str. 108-111. Ovaj primjer prilagođava logistički model s podacima doze i odgovora, a također bi se jasno mogao prilagoditi upotrebom glm( ). Podaci su sljedeći:

```
> x <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113,
      1.8369, 1.8610, 1.8839)
> y <- c(  6, 13, 18, 28, 52, 53, 61, 60)
> n <- c(59, 60, 62, 56, 63, 59, 62, 60)
```

Negativna log-vjerojatnost koju treba minimizirati je:

```
> fn <- function(p)
   sum( - (y*(p[1]+ p[2]*x) - n*log(1+exp(p[1]+ p[2]*x))
        + log(choose(n, y)) ))
```

Odabiremo razumne početne vrijednosti i vršimo prilagođavanje:

```
> out <- nlm(fn, p = c(-50,20), hessian = TRUE)
```

Nakon prilagođavanja out$minimum je negativna log-vjerojatnost, a out$estimates su procjene maksimalne vjerojatnosti parametara. Da bismo dobili približne SEs procjena radimo sljedeće:

```
> sqrt(diag(solve(out$hessian)))
```

95%-tni interval povjerenja bio bi procjena parametra ± 1.96 SE.

11.8 Neki nestandardni modeli

Završavamo ovo poglavlje samo kratkim spominjanjem nekih od drugih R-ovih metoda koje su dostupne za posebne probleme regresije i analize podataka.

● **Mješoviti modeli**. Paket **nlme**, koji je doprinos korisnika, daje funkcije lme( ) i nlme( ) za linearne i nelinearne modele s mješovitim učincima, to jest za linearnu i nelinearnu regresiju u kojoj neki od koeficijenata odgovaraju slučajnim učincima. Te funkcije otežavaju upotrebu formula za specificiranje modela.

● **Lokalno aproksimirajuće regresije**. Funkcija loess( ) prilagođava neku neparametarsku regresiju s pomoću lokalno ponderirane regresije. Takve su regresije korisne za isticanje trenda u nesređenim podacima ili za reduciranje podataka kako bi se dao uvid u veliki set podataka.
Funkcija loess je standardni paket **modreg**, zajedno s kôdom za projicirani slijed regresije.

● **Robustna regresija**. Dostupno je nekoliko funkcija za prilagođavanje regresijskih modela na način koji je otporan na utjecaj ekstremnih izdvojenih vrijednosti u podacima. Funkcija lqs u standardnom paketu istoga naziva daje najsuvremenije algoritme za visokorezistentne prilagodbe. Manje rezistentne ali statistički djelotvornije metode su dostupne u paketima koji su doprinos korisnika, na primjer funkcija rlm u paketu **MASS**.

● **Aditivni modeli**. Cilj ove tehnike je konstruiranje funkcije za regresiju iz poravnanih aditivnih funkcija određujućih varijabla, obično po jedne za svaku određujuću varijablu. Funkcije avas i ace u paketu **acepack** i funkcije bruto i mars u paketu **mda** daju neke primjere tih tehnika u paketima koji su doprinos korisnika R-u.

● **Modeli bazirani na grananju (Tree based models).** Umjesto traženja eksplicitnog globalnog linearnog modela za predikciju ili interpretaciju, (tree based modeli – modeli bazirani na grananju teže razdvajanju podataka, rekurzivno, u kritičnim točkama određujućih varijabla, kako bi se podaci na kraju podijelili u grupe koje su što više homogene unutar sebe a što više heterogene među sobom. Rezultati često vode do stjecanja uvida kojma druge metode analize podataka ne obiluju. Modeli su ponovno specificirani u obliku običnog linearnog modela. Funkcija za prilagođavanje modela je tree( ) ali se mnoge druge generičke funkcije kao što su plot( ) i text( ) dobro prilagođuju grafičkom prikazivanju rezulata usklađivanja modela baziranog na grananju.
Modeli bazirani na grananju dostupni su u R-u *putem* paketa **rpart** i **tree** koji su doprinos korisnika.

12 Upotreba grafičkih naredbi

Grafičke mogučnosti su važna i izuzetno raznovrsna komponenta R-ovog okruženja. Moguće ih je upotrijebiti za prikaz mnogo različitih statističkih grafova kao i za građenje potpuno novih vrsta grafova.
Grafičke mogučnosti se mogu upotrebljavati na interaktivan i na grupni (batch) način ali je u većini slučajeva interaktivna upotreba produktivnija. Interaktivna upotreba je također lagana jer R prilikom startanja inicira grafički podsustav (*device driver)* koji otvara poseban *grafićki prozor* za prikaz interaktivnih grafika. Iako se to odvija automatski, korisno je znati da se u UNIX-u koristi naredba X11( ), u Windowsu naredba windows( ) a u MacOS 8/9 naredba macintosh( ) za otvaranje grafičkog prozora.

Kad je jednom podsustav pokrenut, R-ove naredbe za grafove mogu se koristiti za kreiranje različitih grafičkih prikaza.
Naredbe za kreiranje grafova se dijele u tri osnovne grupe:
● Funkcije za grafike **visoke razine** kreiraju novi grafikon na grafičkom podsustavu, moguće s osima, oznakama, naslovima i tako dalje.
● Funkcije za grafike **niske razine** dodaju više informacija na postojeći grafikon kao što su dodatne točke, linije i oznake.
● Funkcije za **interaktivne** grafike omogućuju da se interaktivno doda informacija na postojeći grafikon ili da se iz njega ekstrahira informacija, s pomoću točkastog uređaja kao što je miš.

Pored toga, R vodi listu *grafičkih parametara* s kojima se mogu prilagođavati vlastiti grafovi. Ovaj priručnik opisuje samo osnovne naredbe. Poseban grafički podsustav nalazi se u paketu **grid** i nadopunjuje se sa osnovnim paketom naredbi. Ovaj podsustav je vrlo moćan, ali treba uložiti više truda da bi ga se moglo upotrebljavati. Također se preporuča paket **lattice** koji se nadograđuje na grid paket. Ta kombinacija paketa omogućava nam tzv. "multi-panel" grafove kao što se to može raditi u **Trellis** sustavu koji je dio S jezika.

12.1 Naredbe za grafiku visoke razine

Funkcije za grafike visoke razine su dizajnirane za generiranje kompletnog grafa od podataka koji se daju funkciji kao argumenti. Po potrebi se automatski generiraju osi, oznake i naslovi (osim ako ne zahtijevate drukčije). Naredbe za grafiku visoke razine uvijek kreiraju novi grafikon, brišući po potrebi onaj već postojeći.

12.1.1 Funkcija plot( )

Jedna od R-ovih najčešće korištenih funkcija za grafove je funkcija plot( ). To je *generička* funkcija: vrsta grafa koji se kreira ovisi o vrsti ili *klasi* prvoga argumenta.

plot(*x, y*)
plot(*xy*) Ako su *x* i *y* vektori, plot(*x, y*) rezultat je točkasti graf (scatterplot) *y*-a u zavisnosti o *x*-u. Isti se učinak može proizvesti dajući jedan argument kao listu koja sadrži dva elementa *x* i *y* ili kao matricu s dva stupca.
plot(*x*) Ako je *x* vremenski niz (time series), proizvodi graf vremenskih nizova, ako je *x* numerički vektor, dobiva se graf vrijednosti vektora u zavisnosti od indeksa u vektoru, a ako je *x* vektor s kompleksnim brojevima, dobiva se graf imaginarnog dijela komleksnog broja u odnosu na realni dio.
plot(*f*)
plot(*f, y*)
f je faktor, *y* je numerički vektor. Prvi oblik generira graf *f*-a u obliku "bar"; drugi oblik generira tzv. "boxplot" *y*-a za svaki nivo *f*-a.
plot*(df)*
plot*(~ expr)*
plot*(y ~ expr)*
df je spremnik podataka, *y* je bilo koji objekt, *expr* je lista naziva objekata odvojenih s pomoću '+' (npr. a+b+c). Prva dva oblika generiraju distribucijske grafove varijabla u spremniku (prvi oblik) odnosno distribucijske grafove broja objekata s nazivima (drugi oblik). Treći oblik daje graf y-a u odnosu na svaki objekt imenovan u *expr*.

12.1.2 Prikazivanje višestrukih podataka

R omogućuje dvije vrlo korisne funkcije za prezentiranje višestrukih podataka. Ako je X numerička matrica ili spremnik podataka, naredba

```
> pairs(X)
```

proizvodi točkaste grafove matrica parova, varijabli koje su definirane stupcima X-a, to jest, svaki stupac X-a je prikazan u zavisnosti od svakog drugog stupca X-a, i rezultat je *n*(*n*-1) grafova čije vrijednosti na koordinatnim osima su određene elementima matrice.

Ako imamo tri ili četiri varijable možda će nam naredba *coplot* dati bolje pojašnjenje. Ako su *a* i *b* numerički vektori, a *c* numerički vektor ili faktor (svi jednake duljine), tada naredba

```
> coplot(a ~ b | c)
```

daje niz grafova, gdje je prikazana zavisnot *a* u odnosu na *b* za dane vrijednosti *c*-a. Ako je c faktor, to jednostavno znači da je a prikazan u odnosu na b za svaku nivo c-a. Kada je c numerčki vektor, on je podijeljen na nekoliko uvjetnih intervala i za svaki je interval a ucrtan u odnosu na b za vrijednosti c-a unutar toga intervala. Broj i pozicija intervala može se kontrolirati sa given.values = argument to coplot( ) – funkcija co.intervals( ) je korisna za odabiranje intervala. Možete također upotrijebiti dvije *dane* varijable korištenjem naredbe poput

```
> coplot(a ~ b | c + d)
```

koja daje grafove a u odnosu na b za svaki zajednički uvjetni interval od c i d.

Funkcija coplot( ) i pairs( ) obje uzimaju argument panel= koji se može upotrijebiti za prilagođavanje vrste grafa koji se pojavljuje u svakom panelu. Za dobivanje točkastog grafa predefinirana vrijednost je points(), ali dajući neku drugu funkciju grafike niskoga nivoa za dva vektora x i y kao vrijednost za panel= možete proizvesti bilo koju vrstu grafa koju želite. Primjer panel funkcije korisne za upotrebu funkcije coplot je funkcija panel.smooth( ).

12.1.3 Grafički prikaz

Druge funkcije za grafiku visoke razine služe za kreiranje različitih vrsta grafikona. Evo nekoliko primjera:

qqnorm(x)
qqline(x)
qqplot(x, y)

Grafovi raspodjela-usporedba. Prvi oblik crta numerički vektor x u odnosu na očekivanu normalnu raspodjelu (grafikon normalne raspodjele), a drugi oblik dodaje ravnu liniju takvom grafikonu crtanjem linije kroz kvartile raspodjele i podataka. Treći oblik crta kvantile x-a u odnosu na kvantile y-a radi uspredbe njihovih pripadnih raspodjela.

hist(x)
hist(x, nclass=*n*)
hist(x, breaks=*b*, …)

Daje histogram numeričkog vektora x. Obično se odabire neki razuman broj klasa ali se može preporučiti nclass=argument. Alternativno se prijelomne točke mogu točno

specificirati sa breaks=argument. Ako je dat argument probability=TRUE, y koordinata predstavljaja relativnu frekveniciju umjesto broja događaja.

dotchart(x, …)

Konstruira točkasti graf za podatke u x-u. Na grafu os *y* daje nazive (oznake) podataka u x-u dok os *x* daje njihovu vrijednost. Na primjer, ovim grafom moguć je lagan vizualni prikaz svih unesenih podataka s vrijednostima koje leže u specificiranim rasponima.

image(x, y, z, …)
contour(x, y, z, …)
persp(x, y, z, …)

Grafovi triju varijabli. Naredba image crta pravokutnu rešetku koristeći različite boje za vrijednost z-a, naredba contour crta linije kontura za vrijednost z-a, a naredba persp crta 3D plohu.

12.1.4 Argumenti za funkcije grafike visoke razine

Funkcijama grafike visoke razine može se pridodati nekoliko argumenata kako slijedi:

add=TRUE Funkcija djeluje kao funkcija grafike niske razine, nadodajući graf na već postojeći (samo neke funkcije).

axes=FALSE Ne crta koordinatne osi – korisno za dodavanje vlastitih prilagođenih osi s pomoću funkcije axis( ). Pre definirana vrijednost je axes=TRUE, znači uključi osi.

log="x"
log="y"
log="xy" Os *x*, os *y* ili obje su logaritamske. To vrijedi za mnoge, ali ne za sve vrste grafova.

type= Type= argument kontrolira vrstu nastalog grafa, kako slijedi:
type="p" Crta pojedinačne točke (predefinirana vrijednost)
type="l" Crta linije
type="b" Crta točke povezane linijama (obadvije)
type="o" Crtajte točke precrtane linijama
type="h" Crtajte okomite linije od točaka do osi nula (*high-density*)
type="s"
type="S" Graf stepenaste funkcije. U prvom obliku točku definira vrh vertikale, u drugom obliku je definira dno vertikale.
type="n" Ne crta se. Međutim, osi su još uvijek nacrtane (predefinirana vrijednost) i načinjen je sustav koordinata u skladu s podacima. Idealno za kreiranje grafova nakon kojih slijedi korištenje funkcija grafike niske razine.

xlab=*string*
ylab=*string*

Oznake na osima *x* i *y*. Koristite ove argumente za promjenu predefiniranih vrijednosti, obično se upotrebljavaju nazivi objekata korišteni u pozivanju funkcije za grafike visoke razine.

main=*string*

Naslov crteža, smješten na vrhu grafa i pisan velikim fontom slova.

sub=*string*

Podnaslov, smješten neposredno ispod *x*-osi i pisan manjim fontom slova.

12.2 Naredbe za grafiku niske razine

Ponekad funkcije za grafike visoke razine ne kreiraju onakvu vrstu grafa koju želite. U takvom se slučaju mogu upotrijebiti naredbe za grafiku niske razine radi dodavanja ekstra informacije (kao što su točke, linije ili tekst) na trenutni graf.
Neke od korisnijih funkcija za grafike niske razine su:

points(x, y)
lines(x, y)

Dodaje točke ili linije na trenutno aktivan graf. (svojstvo naredbe plot( ): type=argument može također koristiti kao rgument u tim funkcijama (kao i prdefinirane vrijednosti: "p" – za točke i "l" - za linije)

text(x, y, labels, …)

Dodaje tekst graf u točki s koorinatama x,y. Obično su oznake cijeli brojevi ili znakovi, i u tom slučaju se labels[i] crta u točki (x[i], y[i]). Predefinirana vrijednost je 1:length(x).

Napomena: Ova funkcija se često upotrebljava u nizu

> plot(x, y, type="n"); text(x, y, names)

Parametar type="n" ne prikazuje točke ali kreira osi, a funkcija text( ) daje posebne znakove kako je specificirano u znakovnom vektoru names za točke.

abline(a, b)
abline(h=y)
abline(v=x)
abline(*1m.obj*)

Dodaje liniju na aktivni graf, nagiba b i presjecišta a. h=*y* se može upotrijebiti za specificiranje *y*-koordinata visina gdje horizontalne linije prelaze preko grafikona, dok se v=*x* može slično upotrijebiti za specificiranje *x*-koordinata za vertikalne linije. Također l*m.obj* može biti lista s koeficijentima komponente duljine 2 (kao što je rezultat funkcija za prilagođavanje (fitting) modela), koji su uzeti kao intercept i nagib, tim poretkom.

polygon(x, y …)

Crta poligon definiran uređenim nizom vrhova u (x, y) i osjenčava ga po želji s crtkanim linijama, ili popunjava ako grafički podsustav dopušta sa slikama.

legend(x, y, legend, …)

Dodaje legendu trenutno aktivnom grafu na specificiranoj poziciji. Znakovi, vrste linija, boje itd. koji se ucrtavaju iste su kako i u vektoru oznaka. Također mora postojati najmanje još jedan argument *v* (vektor jednake duljine kao što je duljina legende) s odgovarajućim vrijednostima kako slijedi

legend(, fill=*v*)
Oznaka boje pozadine
legend(, col=*v*)
Oznaka boje za crtanje točaka i linija
legend(, lty=*v*)
Vrsa linije
legend(, lwd=*v*)
Debljina linije
legend(, pch=*v*)
Znakovni niz koji želimo prikazati (vektor znakova)

title(main, sub)

Dodaje glavni naziv na vrh trenutno aktivnog graf pisan velikim fontom slova i (prema izboru) podnaslov sub na dnu pisan manjim fontom slova.

axis(side, …)

Dodaje os trenutnom grafu na strani određenoj prvim argumentom (1 do 4, brojeći u pravcu kazaljke na satu od dna). Ostali argumenti kontroliraju pozicioniranje osi unutar ili pored grafikona, i označavaju pozicije i nazive. Korisno za dodavanje prilagođenih osi nakon što se pozove plot() sa axes=FALSE argument.

Funkcije za grafike niske razine obično iziskuju neku informaciju o pozicioniranju (npr. *x* i *y* koordinate) da bi se odredilo gdje treba smjestiti nove elemente grafikona. Koordinate se daju u smislu korisnikovih koordinata koje su definirane prethodnom naredbom grafa visoke razine i koje se biraju na temelju dobivenih podataka.
Gdje su potrebni *x* i *y* argumenti, također je dovoljno dati jedan argument kao listu s elementima nazvanim x i y. Slično tomu, također je punovaljan unos matrice s dva stupca. Na taj se način funkcije kao što je locator() (vidi niže u tekstu) mogu upotrijebiti za interaktivno specificiranje pozicija na grafu.

12.2.1 Matematičko označavanje

U nekim slučajevima je korisno dodati grafu matematičke simbole i formule. To se u R-u može postići definiranjem nekog *izraza* radije nego pisanjem znakova u text, mtext, axis, ili title. Na primjer, sljedeći kôd crta formulu za funkciju binomne raspodjelu vjerojatnosti:

```
>text(x, y, expression(paste(bgroup("(", atop(n, x),  ")"),pˆx, qˆ{n-x})))
```

Više podataka, uključujući puni izlist dostupnih mogućnosti, može se dobiti u R-u korištenjem naredbi

```
> help(plotmath)
> example(plotmath)
```

12.2.2 Upotreba Hershey fontova u vektoru

Hershey vektorske fontove moguće je specificirati kao fontove koji se koriste za dobivanje tekstova prilikom upotrebe funkcije *text* i *contour*. Tri su razloga za upotrebu Hershey fontova:
● Hershey fontovi daju kvalitetniji izgled, naročito na ekranu računala, ako je tekst rotiran i/ili ako je sitan.
● Hershey fontovi daju određene simbole koji se ne mogu dobiti u standardnim fontovima. Posebno, imaju znakove horoskopa, kartografske simbole i astronomske simbole.
● Hershey fontovi imaju slova ćirilice i japanska slova (Kana i Kanji).
Više podataka, uključujući tablice s Hershey znakovima, može se dobiti u R-u korištenjem naredbi:

```
> help(Hershey)
> example(Hershey)
> help(Japanese)
> example(Japanese)
```

12.3 Interaktivni rad s grafovima

R također posjeduje funkcije koje omogućuju korisnicima ekstrahiranje ili dodavanje informacija na grafu korištenjem miša. Najjednostavnija takva funkcija je funkcija locator() :

locator(n, type)

Čeka dok korisnik odabere lokacije na trenutnom aktivnom grafu korištenjem lijeve tipke miša. To se nastavlja dok se ne izabere n (predefinirana vrijednost 512) točaka ili dok se ne pritisne druga tipka miša (Unix, Windows) ili dok se mišem ne klikne izvan prozora s graovima (Mac). Type argument omogućuje crtanje grafa na odabranim točkama i ima jednak učinak kao kod naredbi za grafiku visoke razine; predefinirana vrijednost znači ne crtanje grafa. locator( ) vraća lokacije odabranih točaka kao listu s dvije komponente x i y.

locator( ) se obično poziva bez argumenata. To je osobito korisno za interaktivno odabiranje pozicija za grafičke elemente kao što su legende ili oznaka kad je teško unaprijed izračunati gdje ih treba smjestiti. Na primjer, za smještanje nekog informativnog teksta pored neke točke grafa može biti korisna naredba

```
> text(locator(1), "Outlier", adj=0)
```

locator( ) neće raditi ako aktivni uređaj, kao što je *postscript* ne radi s mišem.

identify(x, y, labels)
: Omogućava korisniku da istakne bilo koju točku definiranu x-om i y-om (korištenjem lijeve tipke na mišu) crtanjem odgovarajuće komponente oznaka u blizini (ili indeksnog broja točke ako nema oznake). Vraća oznake odabranih točaka kad se pritisne druga tipka (unix, Windows) ili ako se miš klikne izvan prozora za grafike (Mac).

Ponekad želimo radije identificirati određene *točke* na grafu nego njihov položaj. Na primjer, možemo željeti da korisnik odabere neko opažanje koja ga zanima iz grafičkog prikaza i da zatim manipulira tim opažanjem na neki način. Ako imamo broj (*x*, *y*) koordinata u dva numerička vektora x i y, mogli bismo upotrijebiti funkciju identify( ) kako slijedi:

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

Funkcija identify( ) ne obavlja crtanje grafa kao takvo, nego jednostavno omogućava korisniku da pomiče strelicu miša i klikne lijevu tipku miša pokraj neke točke. Točka najbliža strelici miša bit će istaknuta njenim indeksnim brojem (to jest, njenom pozicijom u x/y vektorima) ucrtanim u blizini. Alternativno možete upotrijebiti neki informativni naziv (kao što je name za isticanje upotrebom labels argumenta za identify( ), ili možete potpuno isključiti isticanje s pomoću plot=FALSE argumenta. Kad je postupak završen (vidi gore u tekstu), identify( ) vraća indekse odabranih točaka; te indekse možete upotrijebiti za ekstrahiranje odabranih točaka iz originalnih vektora x i y.

12.4 Korištenje grafičkih parametara

Kod kreiranja grafika, posebno u svrhu prezenacije ili publiciranja, R ne daje uvijek upravo ono što se traži. Možete, međutim, prilagoditi gotovo svaki aspekt prikaza korištenjem grafičkih parametara (*graphics parameters)*. R održava listu velikog broja grafičkih parametara koji kontroliraju, između ostalog, vrste linija, boje, raspored slika i poravnavanje teksta. Svaki grafički parametar ima naziv (kao što je ‘col’, koji kontrolira boje) te vrijednost (broj boje, na primjer).

Za svaki aktivni grafički podsustav se održava posebna lista grafičkih parametara, i svaki podsustav kad ga se starta ima predefinirani skup parametara. Grafički parametri se mogu promijeniti na dva načina: za stalno, utječući na sve grafičke funkcije koje su dostupne tom grafičkom podsustavu; ili privremeno, utječući na samo jedno pozivanje grafičke funkcije.

12.4.1 Permanentne promjene: funkcija par( )

Funkcija par( ) se upotrebljava da bi se pristupilo i modificiralo listu grafičkih parametara za trenutnu aktivni grafički podsustav.

par( )
: Bez argumenata, vraća listu svih grafičkih parametara i njihovih vrijednosti za trenutno aktivni grafički podsustav.

par(c("col", "lty"))
: Sa znakovnim vektorom kao argumentom, vraća samo grafičke parametre s nazivima (ponovno, kao listu.)

par(col=4, lty=2)
: S argumentima koji imaju nazive (ili jednim argumentom liste), mijenja vrijednosti grafičkih parametara s nazivima i vraća originalne vrijednosti parametara kao listu.

Mijenjanje grafičkih parametara korištenjem funkcije par( ) *permanentno* mijenja vrijednost tih parametara, u smislu da će nova vrijednost utjecati na sva buduća pozivanja grafičkih funkcija (u trenutnom aktivnom grafičkom podsustavu). Mijenjanje grafičkih parametara na takav način može se smatrati kao stavljanje predefiniranih vrijednosti (default) za parametre, koje će se upotrebljavati za sve grafičke funkcije osim ako se ne da alternativna vrijednost.
Obratite pozornost da pozivanje par() *uvijek* utječe na globalne vrijednosti grafičkih parametara, čak i onda kada se par( ) poziva iz funkcije. Takvo je ponašanje često nepoželjno – obično želimo promijeniti neke grafičke parametre, napraviti graf, i zatim vratiti originalne vrijednosti bez utjecanja na korisnikov rad u R-u. Početne vrijednosti možete ponovno vratiti tako da pohranite rezultate funkcije par( ) kad vršite promjene i da ponovno vratite početne vrijednosti kad je crtanje grafa završeno.

```
> oldpar <- par(col=4, lty=2)
   … mijenjanje parametara, ctranje grafa …
> par(oldpar)
```

Spremanje svih parametara, te njihovo vračanje možete napraviti sa

```
> oldpar <- par(no.readonly=TRUE)
  … mijenjanje parametara, ctranje grafa …
> par(oldpar)
```

12.4.2 Privremene promjene: argumenti grafičkih funkcija

Grafički parametri se mogu također dodati (gotovo) svim grafičkim funkcijama kao argumenti s nazivima. To ima jednak učinak kao dodavanje argumenata funkciji par() osim što promjene traju samo tijekom poziva funkcije. Na primjer

```
>plot(x, y, pch="+")
```

stvara točkasti graf raspršenja (scatterplot) koristeći znak plus kao znak za crtanje grafa, ne mijenjajući predefinirani znak za crtanje grafa.

12.5 Lista grafičkih parametara

U narednom se poglavlju navode pojedinosti o mnogim opće upotrebljavanim grafičkim parametrima. R-ova help dokumentacija za funkciju par() daje jezgrovitiji sažetak; ovaj je tekst samo detaljizirana alternativa.
Grafički će parametri biti prikazani u sljedećem obliku:
name=*value*

Opis učinka parametra. *name* je naziv parametra, to jest, naziv argumenta koji se koristi u pozivanjima par( ) ili grafičke funkcije. *value* je tipična vrijednost koju možete koristiti kod mijenjanja parametra.

Uočite, da *axes* nije grafički parametar nego argument u nekoliko funkcija za crtanje (pogledajte: *xaxt* i *yaxt*).

12.5.1 Grafički elementi

R-ovi grafovi se sastoje od točaka, linija, teksta i poligona (ispunjena područja). Postoje grafički parametri koji kontroliraju kako se ti *grafički elementi* crtaju, kako slijedi:

pch="+" Znak koji se upotrebljava za crtanje točaka. Predefinirana vrijednost varira ovisno o grafičkom podsustavu, ali je obično ‘o’. Nacrtane točke se pretežno

pojavljuju malo iznad ili ispod prave pozicije, osim ako ne upotrijebite "." kao znak za crtanje grafa, koji daje centrirane točke.

pch=4 Kada je pch dat kao brojka između 0 i uključivo 18 nastaje specijalizirani simbol za crtanje grafova. Da biste vidjeli što simboli predstavljaju, upotrijebite naredbu

```
legend(locator(1), as.character(0:25), pch = 0:25)
```

Od 21 do 25 simboli se ponavljaju, ali to može biti korisno ako se koriste različite boje.

lty=2 Vrste linija. Svi grafički podsustavi ne podržavaju alternativne vrste linija (one usto variraju na onim grafičkim podsustavima koji ih prihvaćaju), međutim linija tipa 1 je uvijek puna linija, a linija tipa 2 i brojeva većih od 2 su točkaste ili crtkane linije odnosno neka kombinacija to dvoje.

lwd=2 Širine linija. Željena širina linija, u višestrukim vrijednostima «standardne» širine linije. Odnosi se na linije osi kao i na linije nacrtane s pomoću lines( ), itd.

col=2 Boje koje se upotrebljavaju za točke, linije, tekst, popunjena područja i slike. Svaki od tih grafičkih elemenata ima listu raspoloživih boja (paleta), i vrijednost tog parametra je indeks te liste.

col.axis
col.lab
col.main
col.sub Za bojanje osi, oznaka, glavnog i pomoćnog naziva.

font=2 Brojka koja specificira koji font upotrijebiti u tekstu. Ako je moguće grafički podsustav urediti tako da brojka 1 odgovara običnom tekstu, brojka 2 podebljanim slovima, brojka 3 kosim slovima, a brojka 4 podebljanim kosim slovima.

font.axis
font.lab
font.main
font.sub Font koji se upotrebljava za označavanje osi, oznake *x*-a i *y*-a, te glavne naslove i podnaslove.

adj=-0.1 Poravnavanje teksta u odnosu na poziciju crtanja grafa. 0 znači poravnati lijevo, 1 poravnati desno, a 0.5 centrirati horizontalno oko pozicije crteža. Stvarna vrijednost predstavlja proporciju teksta lijevo od pozicije crteža, tako vrijednost –0.1 ostavlja prazninu između teksta i pozicije crteža koja odgovara 10% od širine teksta.

cex=1.5 Širina znakova. Vrijednost je željena veličina znakova u tekstu (uključujući crtane znakove u odnosu na predefniranu veličinu teksta).

12.5.2 Koordinatne osi i znakovi za označavanje

Mnogi od R-ovih grafova visoke razine imaju osi, a možete ih sami konstruirati pomoću grafičke funkcije niske razine axis(). Osi imaju tri glavne komponente: linije osi (*axis line) (*stil linije kojega kontrolira grafički parametar lty), *tick marks* (oznake razdiobe koji označavaju razdiobu na jedinice duž linije osi) i oznake jedinica razdiobe (*tick labels)*. Ove se komponente mogu prilagoditi s pomoću sljedećih grafičkih parametara.

lab=c(5, 7, 12)

Prva dva broja su željeni broj intervala na osi *x* odnosno na osi *y*. Treći broj je željena duljina oznaka jedinica razdiobe, u znakovima (uključujući decimalnu točku). Odabir premalene vrijednosti za ovaj parametar može imati za rezultat da se sve oznake zaokruže na isti broj!

las=1 Orijentacija oznaka osi. 0 uvijek znači paralelno s osi, 1 uvijek znači horizontalno, a 2 okomito na os.

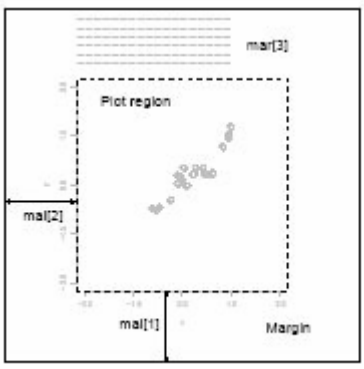
mgp=c(3, 1, 0)
Pozicije komponenata osi. Prva komponenta je udaljenost od oznaka osi do pozicije osi, u tekstovnim linijama. Druga komponenta je udaljenost do oznaka jedinca raspodjele, a posljednja je komponenta udaljenost od pozicije osi do linije osi (obično nula). Brojke pozitivnog predznaka mjere izvan područja grafa, a brojke negativnog predznaka unutar područja grafa.

tck=0.01 Duljina oznake razdiobe, u omjeru veličine područja crteža. Kad je tck malen (manji od 0.5) znakovi na osi *x* odnosno osi *y* su prisilno iste veličine. Vrijednost od 1 daje rešetkaste linije. Negativne vrijednosti oznake izvan područja crteža. Za oznake unutar područja crteža koristite tck=0.01 i mgp=c(1, -1.5, 0).

xaxs="r"
yaxs="i" Stilovi osi za os x odnosno za os y. Kod stila "i" (internal) i stila "r" (predefinirana vrijednost) oznake uvijek leže u rasponu podataka. Međutim , stil "r" ostavlja malo prostora na rubovima. (S ima još definiranih stilova koji nisu još definirani u R-u).

12.5.3 Margine slika

R-ov grafikon je poznat kao slika (figure) i uključuje prostor za crtanje (plot region) okružen marginama (koji mogu sadržavati oznke osi, naslove, itd.) i koji je (obično) ograničen samim osima.
Tipična slika je



Grafički parametri koji kontroliraju oblik slike uključuju:

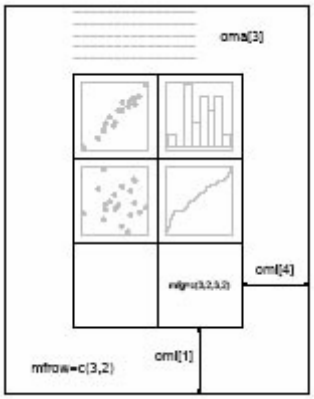
mai=c(1, 0.5, 0.5, 0)
Širine donje, lijeve, gornje odnosno desne margine izmjerene u inčima.

mar=c(4, 2, 2, 1)
Slično kao mai, osim što su mjerne jedinice tekstovne linije.

mar i mai su ekvivalentni u smislu što mijenjanje jednoga od njih mijenja vrijednost drugoga. Predefinirane vrijednosti odabrane za ovaj parametar su često prevelike; desna margina je rijetko kad potrebna, kao i gornja margina ako se ne upotrebljava naslov. Donja margina i lijeva margina moraju biti dostatno široke za smještanje osi i oznaka jedinica razdiobe. Nadalje, predefinirana vrijednost se odabire bez obzira na veličinu i površinu uređaja za prikaz: na primjer, korištenje postscript*()* driver-a s argumentom height=4 rezultirat će dobivanjem grafikona koji je oko 50% margine osim ako se mar ili mai izričito ne postave. Kad se koriste višestruke slike (vidi niže u tekstu) margine se upola smanjuju, međutim to ne mora biti dostatno kada se mnogo slika nalazi na istoj stranici.

12.5.4 Okruženje višestruke slike

R vam dozvoljava kreiranje polje sa *nxm* slika na jednoj stranici. Svaka slika ima svoje vlastite margine, i polje slika je neobvezatno okružen *vanjskom marginom*, kao što je prikazano na sljedećoj slici.



Grafički parametri koji se odose na višestruke slike su sljedeći:

mfcol=c(3, 2)
mfrow=c(2, 4)

Veličina polja s višestrukom slikom. Prva vrijednost je broj redaka; druga vrijednosti je broj stupaca. Jedina je razlika između ova dva parametra u tome što mfcol puni slike po stupcima, dok mfrow puni slike po retcima.

Raspored u u gornjoj slici se mogao kreirati stavljanjem mfrow=c(3,2); slika pokazuje stranicu nakon što su četiri grafikona nacrtana.

mfg=c(2, 2, 3, 2)

Pozicija trenutne slike u okruženju višestruke slike. Prva dva broja su redak i stupac trenutne slike; zadnja dva broja su broj redaka i stupaca u polju s višestrukom slikom. Postavite taj parametar da možete prelaziti iz slike na sliku u polju. Možete čak upotrijebiti vrijednosti za zadnja dva broja koje su različite od *true* vrijednosti za slike nejednake veličine na istoj stranici.

fig=c(4, 9, 1, 4)/10

Pozicija trenutne slike na stranici. Vrijednosti su pozicije lijevog, desnog, donjeg odnosno gornjeg ruba kao postotak stranice mjereno od donjeg lijevog ugla. Vrijednost u primjeru bi se odnosila na desnu sliku u donjem dijelu stranice. Postavite taj parametar za željeno pozicioniranje slika unutar stranice.

oma=c(2, 0, 3, 0)
omi=c(0, 0, 0.8, 0)

Veličina vanjskih margina. Poput mar i mai, prvi oblik mjeri u tekstovnim linijama, a drugi u inčima, počevši od donje margine u pravcu kazaljke na satu.

Vanjske margine su osobito korisne za naslove pisane uzduž stranice, itd. Tekst se može dodati na vanjske margine s pomoću funkcije mtext() s argumentom outer=TRUE. Po predefiniranim vrijednostima, međutim, nema vanjskih margina pa ih morate kreirati eksplicitno korištenjem oma ili omi.
Složenija uređenja višestrukih slika može se dobiti korištenjem funkcija split.screen() i layout( ).

12.6 Pokretački programi (device drivers)

R može generirati grafike (različitih razina kakvoće) na gotovo bilo kojoj vrsti ekrana ili štampača. Prethodno, međutim, R treba dobiti informaciju o kakvom tipu uređaja se radi. To se postiže startanjem *pokretačkog programa*. Svrha ovog programa je konvertirati grafičke instrukcije iz R-a (na primjer "nacrtaj liniju") u oblik koji dotični uređaj može razumjeti.

Ovi programi startaju pozivom funkcije odgovarajučeg pokretačkog programa. Za svaki uređaj postoji jedna takva funkcija: otipkajte help(Devices) da dobijete listu svih. Na primjer, davanjem naredbe

```
> postscript( )
```

postiže se slanje svih budućih grafičkih prikaza na štampač u PostScript formatu. Neki opće upotrebljavani pokretački programi su:

X11( )	Za upotrebu sa X11 window sustavom
postscript( )	Za printanje na PostScript štampačima ili za kreiranje PostScript grafičkih datoteka.
pictex( )	Kreira LaTeX datoteku.

Kad ste završili s radom na uređaju, obavezno isključite pokretački program davanjem naredbe

```
> dev.off( )
```

To osigurava uredan završetak rada uređaja; na primjer ako se radi o štampačima i ploterima to osigurava da svaka stranica bude završena i poslana na uređaj.

12.6.1 PostScript i dokumenti

Dodajući naziv datoteke pokretačkom programu postscript( ), možete pohraniti grafikone u PostScript formatu u datoteku proizvoljnog naziva. Grafikon će biti u nacrtan horizontalno (landscape) osim ako ne date argument horizontal=FALSE, te možete kontrolirati veličinu grafičkog prikaza s pomoću argumenata width i height (grafikon će biti odgovarajuće dimenzioniran da se prilagodi tim dimenzijama). Na primjer, naredba

```
> postscript("file.ps", horizontal=FALSE, height=5, pointsize=10)
```

će kreirati datoteku koji sadržava PostScript kôd za sliku visoku pet inča, koju možda želimo uvrstiti u neki dokument. Važno je obratiti pozornost da će u slučaju ako datoteka navedena u naredbi već postoji, ta datoteka biti izbrisana. Isto će se dogoditi ako je datoteka ranije bila samo kreirana u tijeku aktualnog rada u R-u (current session).

PostScript prikaz se često upotrebljava za uvrštavanje (insert) slike u neki drugi dokument. To najbolje uspijeva kad se proizvede *encapsulated* PostScript datoteka: R uvijek proizvodi prikaz koji se prilagođava, ali jedino označava prikaz kao takav kad se dade argument onefile=FALSE. Ovo neobično označavanje proizlazi iz S-kompatibilnosti: ono zapravo znači da će prikaz biti jedna stranica (koja je dio EPSF specifikacije). Prema tomu, da bismo načinili grafikon za uvršatavanje treba upotrijebiti nešto poput

```
> postscript("plot1.eps", horizontal=FALSE, onefile=FALSE,
        height=8, width=6, pointsize=10)
```

12.6.2 Višestruki grafički podsustavi

U naprednom korištenju R-a često je korisno imati istovremeno aktivno više grafičkih podsustava. Naravno, samo jedan može u danom trenutku primati grafičke naredbe, i taj je uređaj poznat kao *trenutno aktivni grafički podsustav (current device)*. Kad imamo aktivno više grafičkih podsustava, oni su predstavljeni rednim brojem i nazivom koji ga opisuje.

Glavne naredbe koje koristimo kada imamo istovremeno aktivno više grafičkih podsustava:

X11( ) [Unix]
windows( )
win.printer
win.metafile()
[Windows]
quartz( ) [MacOS X]

postscript( )
pdf()
… Svaki novi poziv pokretačkog programa otvara novi grafički podsustav, proširujući tako listu podsustava za još jedan. Taj podsustav postaje trenutno aktivni, i na njega se šalje grafički prikaz. (Neke platforme mogu imati drugačije grafičke podsustave).

dev.list( )
Vraća broj i naziv svih aktivnih grafičkih podsustava. Podsustav na poziciji 1 na listi je uvijek tzv. *null device*, koji uopće ne prima grafičke naredbe.

dev.next( )
dev.prev( )
Vraća broj i naziv grafičkog podsustava koji slijedi nakon trenutno aktivnog odnosno koji mu prethodi.

dev.set(which=*k*)
Može se upotrijebiti da zamijeni trenutno aktivni grafički podsustav sa podsustavom na poziciji *k* sa liste podsustava. Vraća broj i naziv podsustava.

dev.off(*k*)
Isključuje grafički uređaj na poziciji *k* sa liste. Za neke grafičke podsustave, kao za *postscript* , nakon toga će se odmah odštampati datoteka ili ispravno zatvoriti za kasnije štampanje, ovisno o tomu kakve je imao parametre.

dev.copy(device, …, k)
dev.print(device, …, k)
Kreira kopiju grafičkog podsustava k. Device je ovdje pokretački program, kao što je postscript, s ekstra argumentima, po potrebi, specificiranim s pomoću

'…'. dev.print je sličan, ali se kopirani podsustav odmah zatvara, tako da se završne radnje kao što je štampanje odmah izvršavaju.

graphics.off( )
Isključuje sve grafičke podisteme na listi, osim null devicea.

12.7 Dinamičke grafike

R trenutno ne posjeduje ugrađene funkcije za dinamičke grafike, na primjer za rotiranje grafičkih objekata ili za interaktivno osjenčanje grafova. Međutim, velike mogučnosti za dinamičku grafiku dostupne su u sustavu XGobi razvijenom od strane Swayne, Cook i Buja i dostupnom na stranici

http://www.research.att.com/areas/stat/xgobi

i njima je moguć pristup iz R-a putem **xgobi** paketa.
XGobi trenutno radi u X Windows sustavu, u Unix-u ili Windows-u, a R sučelja su dostupna za svaki od njih.
Projekt za razvoj XGobi-ja zove se GGobi i napredak se može vidjeti na stranici

http://www.ggobi.org

13 Paketi

Sve funkcije i podaci u R-u su spremljeni u paketima. Kada je paket učitan njegov sadržaj postaje dostupan. Dva su razloga za ovaj pristup: prvi je efikasnost (učitavanje svih mogućih paketa tražilo bi enormnu količinu memorije i usporilo rad R-a), drugi razlog je da ovaj pristup osigurava nezavisnost u razvoju paketa i izbjegavaju se konfliktne situacije u razvoju koda.
Proces razvoja paketa opisan je u poglavlju "Kreiranje R paketa" u Writing R Extensions (pdf datoteka sezove r_exts.pdf). Ovdje ćemo opisati pakete sa stanovišta korisnika.

Da biste vidjeli spisak instaliranih paketa, napišite naredbu:

```
>library()
```

Instalirani paket nije i odmah dostupan. Paket treba **učitati** da bismo ga mogli koristiti. Naredba, za učitavanje je:

```
>library(ime_paketa)
```

Primjer, učitavanja paketa **boot** (koji sadrži funkcije koje su napisali Davison & Hinkley (1997)):

```
>library(boot)
```

Korisnici koji su povezani na internet mogu koristiti funkciju CRAN.packages() (koja je dostupna u Packages izborniku u Windows-ima i u RAqua grafičkom sučelju) za automatsko ažuriranje i instaliranje paketa.

Da biste vidjeli koje pakete imate učitane u R-u, koristite naredbu:

```
>search()
```

koja će prikazati učitane pakete. Nekoliko učitanih paketa neće biti prisutno u toj listi. O tome možete pročitati u poglavlju 13.3.

Naredbom

```
>help.start()
```

pokrećemo pomoć u HTML formatu. U poglavlju *Reference* --> Package dobijemo listu svih instaliranih paketa, te odabirom paketa dobijemo pomoć za dotični paket. Općenito, za sustavni i cjelovit pregled R-a preporučujemo korištenje HTML oblika pomoći. Ako trebamo, brzi pristup opisu pojedine funkcije ili naredbe najbrži pristup je ?ime_naredbe.

13.1 Standardni paketi

Standardni (ili osnovni) paketi promatraju se kao dio R-ovog koda. Oni sadrže osnovne funkcije koje omogučuju R-u da funkcionira i skupove podataka, standardne statističke i grafičke funkcije opisane u ovom priručniku. One moraju biti automatski dostupne u bilo kojoj instalaciji R-a. Pogledajte poglavlje "R paketi" u R FAQ za potpunu listu.

13.2 Dodatni paketi i CRAN

Postoje stotine dodatnih paketa za R pisanih od strane mnogih autora. Neki od njih implementiraju specijalne statističke metode, druge daju pristup do raznih podataka ili opreme (hardware-a) ili samo predstavljaju udžbenike za učenje. Mnogi su dostupni na stranicama CRAN-a (http://CRAN.R-project.org/ ili alternativnim mjestima – mirror sites), dok se drugi nalaze u raznim arhivama kao npr. *Bioconductor* (http://www.bioconductor.org/). R FAQ sadrži listu zadnjih promjena, ali se kolekcija dostupnih paketa mijenja često.

13.3 Namespaces

Paketi mogu imati NAMESPACES i sada svi osnovni i preporučeni paketi imaju osim paketa *datasets*. Namespaces je dio programskog koda u paketu gdje se definira koji objekti, funkcije i varijable su vidljive kada se paket učita, što se postiže funkcijom export(), odnosno koje paket učitava iz drugih paketa za internu upotrebu, za što se koristi funkcija import() . Namespaces ima trostruku ulogu: dozvoljava piscu paketa da sakrije funkcije i podatke koje su samo za internu upotrebu, što štiti funkcije od nepravilnog rada kada korisnik (ili pisac paketa) slučajno pozove funkciju koja sa istim imenom postoji u nekom drugom paketu. Također sa Namespaces je definiran mehanizam kako se pozivaju funkcije iz drugih paketa. Na primjer, funkcija t() je funkcija transponiranja u R-u, ali korisnik može definirati funkciju s istim imenom t. Namespaces razrješavaju konflikt da kada pozovete korisnikovu funkciju t da se ne izvrši R-ova funkcija t.

Postoje dva operatora koji rade sa namespaces. **Prvi je operator :: (dvostuke dvotočke)**. U gornjem primjeru R-ova funkcija bi se pozivala sa base::t , budući je funkcija transponiranja t() definirana u paketu *base*. Samo funkcije koje su eksportirane iz paketa (tj. koje su deklarirane da se vide) mogu se pozivati na ovaj način.

Operator ::: (trostruke dvotočke) može se vidjeti na par mjesta u R-u. Ovaj operator djeluje kao :: (dvostruke dvotočke) ali također dozvoljava pristup skrivenim objektima u paketu. Korisnicima se preporuča funkcija *getAnywhere()* za pretraživanje po više paketa istovremeno.

Paketi su često zavisni, tj. funkcije u jednom paketu pozivaju funkcije u drugom paketu. To znači kada se neki paket učitava biti će učitani svi zavisni paketi. Gornji operatori također vrše učitavanje paketa ako nije učitan da bi mogli naći traženu funkciju. Ako su paketi s namespaces učitani automatski (tj. učitavanje je uzrokovao neki drugi paket) neće biti vidljivi ako pozovemo naredbu *search()*.

• Dodatak A Primjer rada u R okruženju

Sljedeći primjer rada u R okruženju se navodi radi upoznavanja s nekim karakteristikama R okruženja kroz njihovo korištenje. Mnoge karakteristike ovog sustava izgledat će u početku nepoznate i zbunjujuće, ali samo kroz kratko vrijeme.

Prijavite se na sustav, startajte vaš window sustav. Također bi u vašem radnom direktoriju trebali imati datoteku 'morley.tab'. Ako je nemate, obratite se lokalnom ekspertu (ili ju sami kopirajte iz 'base/data' poddirektorija predefinirane u R-ove putanje do njegovih biblioteka). Ako ste ju našli, nastavite dalje.

$ R — Startajte R na odgovarajuć način za vašu platformu.
R-ov program počinje sa porukom.
(U ovom primjeru nećemo pisati prompt da se izbjegne zabuna.)

help.start( ) — Startajte HTML sučelje na on-line pomoći (korištenjem web pretraživača dostupnog na vašem računalu). Kratko ispitajte karakteristike ove mogućnosti s pomoću miša. Načinite ikonu za pomoć i idite dalje.

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

Generira dva pseudo-nasumična normalna vektora *x*- i *y*- koordinate.

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

Crta točke u ravnini. Grafički prozor se automatski pojavljuje.

```
ls( )
```

Pogledajte koji se R-ovi objekti sada nalaze u R-ovom radnom prostoru.

```
rm(x, y)
```

Odstranite objekte koji više nisu potrebni. (Počistite).

```
x <- 1:20
```

Daje x = (1,2,…,20).

```
w <- 1 + sqrt(x)/2
```

Težinski vektor standardnih odstupanja.

```
dummy <- data.frame(x=x, y=x, + rnorm(x)*w)
dummy
```

Kreirajte spremnik podataka od dva stupca, stupac *x* i stupac *y*, i pogledajte u njega.

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

Prilagodite jednostavnu linearnu regresiju *y*-a na *x*-u i pogledajte analizu.

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

Budući da su nam poznata standardna odstupanja, možemo načiniti ponderiranu regresiju.

```
attach(dummy)
```

Načinite stupce u data frame-u vidljivima kao varijable.

lrf <- lowess(x, y)
Načinite neparametarsku funkciju lokalne regresije.

plot(x, y)
Standardni crtež točaka.

lines(x, lrf$y)
Dodajte lokalnu regresiju.

abline(0, 1, lty=3)
Istinska regresijska linija: (intercept 0, nagib 1).

abline(coef(fm))
Neponderirana regresijska linija.

abline(coef(fm1), col = "red")
Ponderirana regresijska linija.

detach ( ) Odstranite data frame iz pretražnog puta.

plot(fitted(fm), resid(fm),
xlab="Fitted values",
ylab="Residuals",
main="Residuals vs Fitted"
Standardni grafikon dijagnostičke regresije da se provjeri homogenost varijance. Da li ga vidite?

qqnorm(resid(fm), main="Residuals Rankit Plot")
Grafikon normalnih pogodaka za provjeru nesimetričnsti, zakrivljenost i izdvojene vrijednosti (outliers).

rm(fm, fm1, lrf, x, dummy)
Ponovno počistite.

U tekstu koji slijedi prikazat će se podaci iz klasičnog eksperimenta Michaelsona i Morleya za mjerenje brzine svjetlosti.

file.show("morley.tab")
Neobvezatno. Pogledajte datoteku.
mm <- read.table("morley.tab")
mm Čitajte Michaelson i Morley podatke kao data frame i pogledajte u data frame. Postoji pet eksperimenata (stupac Expt), svaki od njih ima 20 serija (stupac Run), a s1 je zabilježena brzina svjetlosti, odgovarajuće kodirana.

mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
Promijenite Expt i Run u faktore.

attach(mm)
Učinite data frame vidljivim na poziciji 2 (predefinirano).

plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
Usporedite pet eksperimenata s običnim grafikonima kućicama.

```
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
```
Analizirajte kao slučajan blok, s ‘runs’ i ‘experiments’ kao faktorima.

```
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
```
Prilagodite podmodel izostavljajući ‘runs’ i usporedite korištenjem formalne analize varijancije.

```
detach( )
rm(fm, fm0)
```
Počistite prije nego krenete dalje.

Sada ćemo pogledati još neke grafičke karakteristike: grafikone kontura i slika.

```
x <- seq(-pi, pi, len=50)
y <- x
```
x je vektor od 50 vrijednosti jednake prostranosti u $-\pi \le x \le \pi$. *y* je isto tako.

```
f <- outer(x, y, function(x, y) cos(y)/(1 + xˆ2))
```
f je kvadratna matrica, s retcima i stupcima indeksiranim sa *x* odnosno *y,* vrijednosti funkcije $\cos(y)/(1 + x^2)$.

```
oldpar <- par(no.readonly = TRUE)
par(pty="s")
```
Pohranite parametre ucrtavanja i stavite grafički prikaz na "square".

```
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
```
Načinite konturnu mapu za *f*: dodajte još linija za više detalja.

```
fa <- (f-t(f))/2
```
fa je "asimetrični dio" *f*-a. (t( ) je operacija transponiranja).

```
contour(x, y, fa, nlevels=15)
```
Načinite grafikon konture, …

```
par(oldpar)
```
… i vratite stare grafičke parametre.

```
image(y, y, f)
image(y, y, fa)
```
Načinite nekoliko grafikona slike visoke gustoće (od koje možete ako želite dobiti ispis na papiru),…

```
objects( ); rm(x, y, f, fa)
```
… i počistite prije nego nastavite.

R može također izvoditi računske operacije sa kompleksnim brojevima.

```
th <- seq(-pi, pi, len=10)
z <- exp(1i*th)
```
1i se koristi za kompleksni broj *i* - imaginarna jedinica.

```
par(pty="s")
plot(z)
```

Crtanje grafikona kompleksnih argumenata znači crtanje imaginarnih u odnosu na realne dijelove. To bi trebao biti krug.

w <- rnorm(100) + rnorm(100)*1i

Pretpostavimo da želimo uzorkovati točke unutar jediničnog kruga. Jedna metoda bi bila uzeti složene brojeve sa standardnim normalnim realnim i imaginarnim dijelovima…

w <- ifelse(Mod(w) > 1, 1/w, w)

… i preslikati bilo koje točke izvan kruga na njima recipročne.

plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(z)

Sve točke su unutar jediničnog kruga ali distribucija nije jednolika.

w <- sqrt(runif(100))*exp(2*pi*runif(100)*1i)
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(z)

Druga metoda koristi jednoliku distribuciju. Točke bi sada trebale izgledati ravnomjernije razmještene preko kruga.

rm(th, w, z)

Opet počistite.

q( ) Izađite iz R-ovog programa. Postavit će vam se pitanje želite li pohraniti R-ov radni prostor i, u slučaju ovakvog istraživačkog rada na kompjutoru, vjerojatno ga ne želite pohraniti.

• Dodatak B Pozivanje R-a

▪ B.1 Pozivanje R-a sa komandne linije

Kada radimo u UNIXU ili komandom modu u WINDOWSU naredba R se može koristiti za pokretanje glavnog R-ovog programa s raznolikim opcijama u obliku

R [*opcije*] [<ulazna_datoteka] [>izlazna_datoteka],

ili, putem R CMD sučelja, kao omotač za razne R-ove alate (npr. za procesiranje datoteka u format R-ovog dokumenta ili za rukovanje dodatnim paketima) koji nisu zamišljeni za "direktno" pozivanje.

Pod UNIX-om treba osigurati da varijabla okruženja TMPDIR nije definirana ili da pokazuje na mjesto gdje se mogu kreirati privremenene datoteke i direktoriji.
Većina opcija kontrolira što se događa na početku i na kraju rada s R-om. Mehanizam za početak rada je sljedeći (pogledajte također on-line help za ‘Startup’ zbog podrobnijih podataka):

● Osim ako nije zadan ‘--no-environ’ R traži korisnikove i R-ove sistemske datoteke za definiranje varijabli okruženja. Naziv sustavske datoteke je naziv na koji pokazuje varijabla okruženja R_ENVIRON; ako ona nije definirana koristi se ‘$R_HOME/etc/Renviron.site’ (ako postoji). Korisnikove datoteke koje se traži su ‘.Renviron’ u trenutnom ili u korisnikovom "home" direktoriju (tim poretkom). Te bi datoteke trebale sadržavati linije oblika ‘*name*=*value*’. (Pogledajte help(Startup) za precizan opis.) Varijable koje možda želite definirati uključuju R_PAPERSIZE (predefiniranu veličinu papira), R_PRINTCMD

(predefiniran naredba za štampanje) i R_LIBS (specificira listu R-ovih putanja za traženje dodatnih paketa (add-on packages).

● R zatim traži općenitu parametarsku datoteku (site-wide startup profile) osim ako nije zadana opcija '--no-site-file'. Naziv te datoteke se uzima iz vrijednosti R_PROFILE varijable okruženja. Ako ta varijabla nije definirana koristi se '$R_HOME/etc/Rprofile.site' ako postoji.

● Nakon toga, osim ako nije dano '--no-init-file' R traži file pod nazivom '.Rprofile' u trenutnom direktoriju ili u korisnikovom direktoriju (tim poretkom) i izvršava ga.

● Također učitava zadnju postojeću radnu okolinu pohranjenu u '.RData' (osim ako nije specificirano '--no-restore' ili '--no-restore-data').

● Konačno, ako postoji funkcija .First, ona se izvršava. Ova funkcija (jednako kao i funkcija .Last na kraju rada s R-om) može se definirati u odgovarajućem početnom profilu, ili može biti u '.RData'.

Pored toga, postoje opcije za kontrolu raspoložive memorije za rad R-a (pogledajte on-line help za 'Memory' za podrobnije informacije). Korisnicima obično to neće biti potrebno osim u slučaju ako žele ograničiti količinu memorije kojom se R koristi.

R prihvaća sljedeće parametre u komandnoj liniji.

'--help'
'-h' Prikazuje kratku poruku pomoći.
'--version' Prikazuje verziju R-a.
'RHOME' Daje putanju do R-ovog osnovnog direktorija. U ovom direktoriju nalazi se sve u vezi R-a (izvršne datoteke, paketi, dokumentacija, itd) osim skripti za pokretanje R-a i sustavskih datoteka za pomoć (man stranice).
'--save'
'--no-save'
Kontrolira da li skupove podataka treba ili ne treba pohraniti na završetku rada s R-om. Ako nije dan niti jedan od ova dva parametra, korisnika će se kod završetka rada s R-om (*q*( )) pitati što od toga želi; kod *batch* načina jedno od ovo dvoje mora biti specificirano.
'--no-environ'
Ne čita niti jednu korisničku datoteku za definiranje varijabli okruženja.
'--no-site-file'
Ne čita opću parametarsku datotekul kod pokretanja.
'--no-init-file'
Ne čita korisnikovu parametarsku datoteku kod pokretanja.
'--restore'
'--no-restore'
'--no-restore-data'
Kontrolira da li pohranjena radna okolina (datoteka '.RData' u direktoriju u kojem je R pokrenut) treba biti učitana kod pokretanja R-a. Predefinirana vrijedost je vračanje ('--restore'), dok '--no-restore' podrazumijeva sve specifične '--no-restore-*' opcije.
'--no-restore-history'
Kontrolira da li datoteka povjesti naredbi (history file) (uobičajeno, datoteka '.Rhistory' u direktoriju u kojem je R pokrenut ali se može promjeniti definiranjem R_HISTFILE varijable) treba biti učitan kod pokretanja. Predefinirana vrijednost je učitavanje.
'--vanilla'

Kombinira ‘--no-save’, ‘--no-environ’, ‘--no-site-file’, ‘--no-init-file’ i ‘--no-restore’.

‘--no-readline’
(Samo UNIX) Isključuje editiranje putem komandne linije upotrebom **readline**. To je korisno kad se R pokreće iz Emacs-a pomoću ESS ("Emacs Speaks Statistics") paketa. Pogledajte Dodatak C [Editor komandne linije], str. 90, za podrobnije informacije.

'--ess'
(Samo Windows) Dozvoljava upotrebu Rterm sa *R-inferior-mode* u ESS-u.

‘--min-vsize=*N*’
‘--max-vsize=*N*’
Specificira minimalnu ili maksimalnu količinu memorije (tzv. "heap") koju R rezervira za dinamičke objekte, gdje je *N* broj bytova. Ovdje *N* mora biti cijeli broj ili cijeli broj koja završava sa ‘G’, ‘M’, ‘K’ ili ‘k’ , gdje je ‘Giga’ (2^30), ‘Mega’ (2^20), (kompjuterski) ‘Kilo’ (2^10) ili regularni ‘kilo’ (1000).

‘--min-nsize=*N*’
‘--max-nsize=*N*’
Specificira količinu memorije upotrijebljenu za objekte fiksne veličine stavljanjem broja "cons cells" na *N*. Pogledajte raniju opciju za detalje o *N*-u. Cons cell ima 28 bytova na 32-bit računalima i obično 56 bytova na 64-bit računalima.

‘--max-ppsize=N’
Specificira maksimalnu veličinu "pointer protection stack"-a kao N lokacija. Predefinirana vrijedonst je 10000, ali može biti povečana za potrebe velikih i kompliciranih računa. Momentalno, največa dozvoljena vrijednost je 100000.

‘--max-mem-size=N’
(Samo Windows) Specificira maksimalnu količinu memorije koju R upotrebljava i za R objekte i za radnu okolinu. Predefinirana vrijednost je manja vrijednost od 1024 Mb ili fizičke veličine memorije i mora biti najmanje 16 Mb.

‘--quiet’
‘--silent’
‘--q’
Ne prikazuj početnu poruku o autorskom pravu i pozdravnu poruku.

‘--slave’ Učiniti da R što tiše radi. Ova je opcija zamišljena za programe koji koriste R za računanje rezultata koji su im potrebni.

‘--verbose’ Prikaži više informacija o progresu procesa, posebno postavite R-ovu *verbose* opciju na TRUE. R-ov kôd koristi ovu opciju za kontrolu prikaza dijagnostičkih poruka.

‘--debugger=*name*’
‘-d *name*’ (Samo UNIX) Pokrenite R kroz debuger *name*. Obratite pozornost da se u ovom slučaju daljnji parametri komandne linije ne uvažavaju, te umjesto toga parametre treba zadavati kroz debuger.

‘--gui=*type*’
‘-g *type*’ (Samo UNIX) Koristite *type* kao sučelje grafičkog korisnika (obratite pozornost da to također uključuje interaktivne grafike). Trenutno su moguće vrijednosti za *type* ‘X11’ (predefinirano), 'Tcl/Tk', 'Tk', ‘gnome’ pod uvjetom da je GNOME instaliran na sustavu, i ‘none’.

‘--args’
Ovaj parametar ne radi ništa i označava da tekst iza može biti zanemaren. To je korisno ako želimo dobiti vrijednosti sa naredbom commandArgs().

Obratite pozornost da se unos i izlaz mogu preusmjeriti na uobičajen način (korištenjem ‘<’ i ‘>’). Upozorenja i poruke greški mogu se preusmeriti *stderr* kanal, osim na Windows 9X/ME.

Naredba R CMD omogućuje pozivanje različitih alata koji su korisni u konjunkciji s R-om, ali nisu namijenjeni za "direktno" pozivanje. Opći oblik je

R CMD *command args*

gdje je *command* naziv alata, a *args* argumenti koji mu se pridjeljuju.

Trenutno su dostupni sljedeći alati.

BATCH	Radite s R-om u batch načinu.
COMPILE	(Samo Unix) Kompiliranje datoteka za korištenje s R-om.
SHLIB	Kreiranje (build) dijeljene biblioteke koja se dinamički učitava.
INSTALL	Instaliranje dodatnih paketa.
REMOVE	Odstranjivanje dodatnih paketa.
build	Kreiranje dodatnih paketa.
check	Provjerite dodatne pakete.
LINK	Linker za kreiranje izvršnih programa.
Rprof	Post-procesno R-ovo profiliranje datoteka.
Rdconv	Konvertiranje Rd formata u razne druge formate, uključujući HTML, Nroff, LaTEX, običan tekst, i format S-dokumentacije.
Rd2dvi	Konvertirajte Rd format u DVI/PDF.
Rd2txt	Konvertirajte Rd format u tekst.
Sd2Rd	Konvertirajte S-dokumentaciju u Rd format.

Upotrebite

R CMD *command* --help

za dobivanje informacije o upotrebi svakog od alata dostupnih putem R CMD sučelja.

- **B.2 Pozivanje R-a u Windows sustavu**

Postoje dva načina pokretanja R-a u Windows sustavu. U prozoru kao terminal (tj. pokretanje cmd.exe, command.com). što je opisano u predhodnom poglavlju, pozivajući R.exe ili Rterm.exe. Ovo je više namjenjeno kada radimo u *batch modu.* Za interaktivni rad preporuča se GUI verzija (Rgui.exe).
Postupak pokretanja R-a u Windows sustavu je vrlo sličan postupku u UNIX-u, ali pojam "home directory" mora biti pojašnjen i ne mora uvjek biti definiran na Windows sustavu. Ako je definirana varijabla okruženja R_USER ona sadrži putanju do "home" direktorija. Ako je definirana varijabla okruženja HOME ona također može sadržavati putanju do "home" direktorija. Ako to nije slučaj, nakon ovih varijabli (koje korisnik definira sam) R pretražuje sistemske varijable okruženja i pokušava naći "home" direktorij. Prvo pokušava naći windows-ov "personal" direktorij (obično, C:\Documents and Settings\username\My Documents u Windows XP sustavu). Ako to ne uspije, onda traži da li su sistemske vrijable okruženja HOMEDRIVE i HOMEPATH definirane (koje su obično definirane na Windows-u NT/2000/XP) i one definiraju "home" direktorij. Ako ništa od toga ne uspije onda se uzima direktorij gdje je R pokrenut.

Varijable okruženja se zadaju u obliku "*name*=*value*" parovi na kraju komandne linije prilikom pozivanja Rgui.exe.
Slijedeći parametri su dostupni prilikom pozivanja Rgui.exe
‘--mdi’
‘--sdi’
‘--no-mdi’

Kontrolira da li će Rgui raditi kao MDI program (predefinirana vrijednost, s višestrukim malim prozorima unutar jednog glavnog prozora) ili kao SDI aplikacija (tj. s višestrukim nezavisnim prozorima na radnoj površini).

'--debug' Aktivira opciju *Break to debugger* opciju u izborniku Rgui-a i pokreće program za kotrolu rada (debugger) prilikom izvršavanja naredbi.

U Windows sustavu s R CMD možete specificirati vlastite *.bat ili *.exe datoteke umjesto da upotrebljavate u R ugrađene naredbe. Možete definirati slijedeće varijable okruženja: R_HOME, R_VERSION, R_CMD, R_OSTYPE, PATH, PERL5LIB i TEXINPUTS.
Na primjer, ako imate u svojoj putanji definiran program *latex.exe*, tada možete napisati

R CMD latex.exe mydoc

Ova naredba će pokrenuti LaTeX sa mydoc.tex datotekom sa putanjom do R-ovog *share/texmf* makroa koji je dodan u TEXINPUTS.

▪ B.3 Pozivanje R-a u Mac OS X

Dva su načina za pokretanje R-a pod Mac OS X operacionom sustavu. Pomoću Terminal.app prozora pozivajući R na isti način kao što smo opisali u predhodnom poglavlju. Drugi način je u grafičkom sučelju (GUI-u) pokretanjem R.app koji se po definiciji instalira u *Application* direktorij. To je standardna MAC OS X aplikacija.
Procedure za pokretanje R-a su slične onima a UNIX sustavima. "home" direktorij je unutar R.framework-a, ali pokretačke procedure i aktivni radni direktorij su postavljene kao korisnikov direktorij osim ako nije drugačije definirano u *Preferences* prozoru do kojeg dolazimu u grafičkom sučelju.

• Dodatak C Editor komandne linije

▪ C.1 Uvodne napomene

Kad je GNU **readline** library dostupan u vrijeme konfiguriranja R-a za kompilaciju u UNIX-u, koristi se ugrađeni editor komandne linije koji omogućuje pozivanje, editiranje i ponovno izvršavanje prethodnih naredbi. Napomena: ovaj dodatak se **ne odnosi** na GNOME sučelje u UNIX-u nego samo na standardno sučelje komandne linije.
Može se isključiti (korisno za upotrebu sa ESS[1]) korištenjem startup opcije '--no-readline'.
Windows verzije R-a imaju nešto jednostavnije editiranje komandne linije: pogledajte 'Console' u 'Help' menu GUI-a, i file 'README.Rterm' za editiranje komandne linije u Rterm.exe.
Kad se R koristi s mogućnostima **readline**, dostupne su dolje opisane funkcije.
Mnoge od tih funkcija koriste bilo Control ili Meta znakove. Control znakovi, kao *Control-m*, dobiju se tako da se drži pritisnutu tipku <CTRL> dok se pritišće tipka <m> , i pišu se kao *C-m* u daljnjem tekstu. Meta znakovi (na MS Windows sustavu to je obično tipka <ALT>), kao *Meta-b*, upisuju se tako da se drži pritisnuto <META> dok se pritišće tipka <b> , i u nastavku ćemo ih pisati kao *M-b*. Ako vaš terminal nema tipku za META , svejedno možete tipkati Meta znakove s pomoću nizova od dva znaka počevši sa <*ESC*>. Tako, da biste upisali *M-b* možete otipkati <ESC> <b>. Nizovi ESC znakova su također dozvoljeni na terminalima sa pravim Meta tipkama. Obratite pozornost da je razlika između upotrebe velikih i malih slova značajna za Meta znakove.

[1] Paket 'Emacs Speaks Statistics'; pogledajte URL http://ess.stat.wisc.edu/

- **C.2 Akcije editiranja**

R program čuva povijest naredbi koje upisujete, uključujući i one koje su napisane pogrešno, i naredbe iz povijesti naredbi se mogu ponovno pozvati, mijenjati po potrebi i ponovno zadavati kao nove naredbe. U Emacs stilu editiranja komandne linije bilo koje obično tipkanje u ovoj fazi editiranja uzrokuje da se znakovi uvrste u naredbu koju editirate, premještajući sve znakove desno od kursora. U *vi* editoru, način ubacivanja (insert) znakova starta sa <*M*> - <*i*> ili <*M*> - <*a*>, znakovi se tipkaju i ubacivanje se završi pritiskom na tipku <ESC>.
Pritiskanje tipke <RET> (negdje je označena kao <ENTER>) u bilo koje vrijeme uzrokuje ponovno izvršavanje naredbe.
Druge akcije editiranja su sažete u sljedećoj tablici.

- **C.3 Sažetak naredbi editora komandne linije**

Ponovno pozivanje naredbe i vertikalno kretanje

C-p	Idite na prethodnu naredbu (prema natrag u povijesti).
C-n	Idite na sljedeću naredbu (prema naprijed u povijesti).
C-r text	Nađite posljednju naredbu koja sadrži *tekstovni* niz.

Na većini terminala možete također koristiti tipke sa strelicama gore i dolje umjesto *C-p* odnosno *C-n*.

Horizontalno kretanje kursora

C-a	Idite na početak naredbe.
C-e	Idite na završetak reda.
M-b	Idite jednu riječ natrag.
M-f	Idite jednu riječ naprijed.
C-b	Idite jedan znak natrag.
C-f	Idite jedan znak naprijed.

Na većini terminala možete također koristiti tipke sa strelicama lijevo i desno umjesto *C-b* odnosno *C-f*.

Editiranje i ponovno zadavanje naredbe

text	Umetnite *text* na mjestu gdje se nalazi kursor.
C-f text	Dodajte *text* nakon kursora.
DEL	Izostavite prethodni znak (lijevo od kursora).
C-d	Izostavite znak ispod kursora.
M-d	Izostavite ostatak riječi ispod kursora i "pohranite u memoriju".
C-k	Izostavite od kursora do kraja naredbe i "pohranite u memoriju".
C-y	Uvrstite (yank) posljednji "pohranjeni" tekst ovdje.
C-t	Zamijenite znak ispod kursora sa znakom koji slijedi.
M-l	Promijenite ostatak riječi u mala slova.
M-c	Promijenite ostatak riječi u velika slova.
RET	Ponovno zadajte naredbu R-u. Konačni RET završava niz editiranja komandne linije.

• Dodatak D Reference

D. M. Bates and D. G.Watts (1988), Nonlinear Regression Analysis and Its Applications. John Wiley & Sons, New York.
Richard A. Becker, John M. Chambers and Allan R. Wilks (1988), The New S Language. Chapman & Hall, New York. This book is often called the “Blue Book”.
John M. Chambers and Trevor J. Hastie eds. (1992), Statistical Models in S. Chapman

& Hall, New York. This is also called the “White Book”.
Annette J. Dobson (1990), An Introduction to Generalized Linear Models, Chapman and Hall, London.
Peter McCullagh and John A. Nelder (1989), Generalized Linear Models. Second edition, Chapman and Hall, London.
John A. Rice (1995), Mathematical Statistics and Data Analysis. Second edition.
Duxbury Press, Belmont, CA.
S. D. Silvey (1970), Statistical Inference. Penguin, London.

• Dodatak E O prijevodu

Informatička tehnologija (IT) obiluje mnoštvom stručnih izraza i kratica. Profesionalci iz tog područja, u svakodnevnom govoru upotrebljavaju uglavnom engleske izraze (možda 10% je hrvatsko nazivlje). Problem nastaje kada morate napisati ono što govorite. Iako postoji hrvatsko nazivlje (i veoma dobar englesko-hrvatski riječnik stručnog nazivlja) ima riječi koje izazivaju nedoumicu i zahtjevaju opis da bi se shvatio nihov smisao. Posebno ako to što pišete trebaju čitati ljudi čija je osnovna naobrazba iz nekog drugog područja. Primjer takve riječi, koja se često koristi u ovom priručniku je engleska riječ *array*. Ljudi iz IT područja prevesti će je kao *polje*, i oni točno znaju što pod tim misle. Za poljodjelca postoji *polje kukuruza* ili *polje pšenice*, za matematičara riječ *polje* ima jedno drugo definirano značenje određene matematičke strukture, za fizičara postoji *električno polje*, *magnetsko polje*, *polje sila* i sl. U IT području riječ *polje* označava skup elemenata kod kojeg za svaki element znamo tko su mu prehodnici i koji elementi dolaze iza njega (nasljednici) – matematičari bi rekli da je to uređen skup. Primjer *jednodimenzionalnog polja* je svaki niz elemenata: 6,4,8,3,2 , gdje znamo da je 6 prvi element, 4 drugi, 8 treći, ... Primjer *dvodimenzionalnog polja* brojeva:

6	4	8	3	2
5	3	7	9	1
9	1	4	3	2

Rekli bismo da je ovo polje *dimenzije 3* x 5, tj. da ima tri reda i pet stupaca. Svaki element ima jedinstveno definiranu poziciju (npr. broj sedam ima poziciju (2,3) – tj. prvi broj označava redak, a drugi broj označava poziciju u tom retku (stupac).
Primjer 2: (a_{ij}), $i \le 3$, $j \le 5$, što se može napisati kao

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

Polje može biti *višedimenzionalno* tj. može ovisiti o više indeksa.