Įvadas į duomenų statistinio apdorojimo paketą „R“

W. N. Venablesas, D. M. Smitas ir „R“ paketo kūrimo komanda

Žinynas parengtas versijai 4.0.3 (2020-10-10).

Turinys

Pratarmė

Šis įvadas į duomenų statistinio aprodojimo paketą „R“ yra kilęs iš originalaus užrašų rinkinio, apie kitas duomenų statistinio apdorojimo aplinkas „S“ ir „S-Plus“, kurias 1990 – 1992 metais aprašė Bilas Venablesas ir Deividas M. Smitas Adelaidės universitete. Žinyno autoriai padarė keletą nedidelių pakeitimų, kad atsispindėtų duomenų statistinio apdorojimo programų „R“ ir „S“ skirtumai, ir išplėtė dalį medžiagos.

Norėtume nuoširdžiai padėkoti Bilui Venablesui (ir Deividui Smitui) už suteiktą leidimą tokiu būdu platinti šią modifikuotą versiją ir už tai, kad jis yra duomenų statistinio apdorojimo paketo „R“ rėmėjas nuo pat pradžių.

Komentarai ir pasiūlymai visada laukiami. Prašome susisiekti su kūrėjais elektroninio pašto adresu r-core@r-project.org.

Pasiūlymai skaitytojui

Daugelis vartotojų pradeda naudoti duomenų statistinio apdorojimo paketą „R“ dėl jo grafinių galimybių. Skaitykite 12 skyrių, kurį galima skaityti beveik bet kuriuo metu ir nereikia laukti, kol visi ankstesniai skyriai bus perskaityti.

1 Įvadas

1.1 „R“ aplinka

„R“ yra integruotas programinės įrangos priemonių rinkinys, skirtas manipuliuoti duomenimis, juos apskaičiuoti ir grafiškai atvaizduoti. Savybės:

- efektyvi duomenų tvarkymo ir saugojimo priemonė,
- didelis, nuoseklus, integruotas tarpinių duomenų analizės priemonių rinkinys,
- grafinė duomenų analizė ir atvaizdavimo galimybės tiesiogiai kompiuteryje arba spausdintinėje kopijoje,
- gerai išplėtota, paprasta ir efektyvi programavimo kalba (vadinama „S“), kurią sudaro sąlyginiai sakiniai, ciklai, vartotojo apibrėžtos rekursinės funkcijos ir įvesties bei išvesties priemonės.

Sąvoka „aplinka“ yra skirta apibūdinti ją kaip visiškai suplanuotą ir nuoseklią sistemą, o ne kaip papildomą labai specifinių ir nelanksčių įrankių pasirinkimą, kaip dažnai būna su kita duomenų analizės programine įranga.

„R“ yra labai svarbi priemonė naujai kuriamiems interaktyvių duomenų analizės metodams. Ji greitai vystėsi ir buvo išplėsta dideliu paketų rinkiniu. Tačiau dauguma programų, parašytų „R“ kalba, iš esmės yra efemeriškos, parašytos vienam duomenų analizės vienetui.

1.2 Susijusi programinė įranga ir dokumentacija

„R“ gali būti laikomas „S“ kalbos įgyvendinimu, kurį „Bell“ laboratorijose sukūrė Rickas Beckeris, Johnas Chambersas ir Allanas Wilksas, ir kuri, taip pat, yra „S-Plus“ sistemų pagrindas.

Yra keletas knygų, kuriose aprašoma, kaip „R“ naudoti duomenų analizei ir statistikai, o „S“/„S-Plus“ dokumentacija paprastai gali būti naudojama kartu su „R“, nepamirštant skirtumų tarp diegimų.

1.3 „R“ ir statistika

Daugelis žmonių naudoja „R“ kaip statistikos sistemą. Mums labiau patinka galvoti apie aplinką, kurioje buvo įdiegta daugybė klasikinių ir modernių statistikos metodų. Keletas iš jų yra įdiegti į pagrindinę „R“ aplinką, tačiau daugelis jų pasiekiami kaip paketai. Yra 25 paketai, kurie suderinti su „R“ (vadinami „standartiniais“ ir „rekomenduojamais“ paketais), o daug daugiau galima jų rasti CRAN interneto svetainėje https://CRAN.r-project.org.

Daugiausia klasikinės statistikos ir daugumos naujausių metodikų galima naudoti su „R“, tačiau vartotojams gali reikėti būti pasirengusiems šiek tiek padirbėti, kad ją rastų.

Yra svarbus „S“ (taigi ir „R“) ir kitų pagrindinių statistinių sistemų filosofijos skirtumas. „S“ statistinė analizė paprastai atliekama kaip žingsnių seka, o tarpiniai rezultatai saugomi objektuose. Taigi, kai „SAS“ ir „SPSS“ programos duos gausų regresinės ar diskriminantinės analizės rezultatą, „R“ pateiks minimalų rezultatą ir rezultatus laikys tinkamame objekte, kad vėliau galėtų tęsti kitas „R“ funkcijas.

1.4 „R“ ir langų sistema

Patogiausias būdas naudoti „R“ yra darbo vietoje, kurioje veikia langų sistema. Šis žinynas skirtas vartotojams, kurie turi šią galimybę. Visų pirma, mes kartais remsimės „R“ naudojimu X langų sistemoje, nors didžioji dalis to, kas pasakyta, paprastai taikoma bet kokiam „R“ aplinkos diegimui.

Daugeliui vartotojų karts nuo karto reikės tiesiogiai bendrauti su savo kompiuterio operacine sistema. Šiame vadove mes daugiausia aptariame sąveiką kompiuteriuose su UNIX operacine sistema. Jei „R“ naudojate operacinėje sistemoje „Windows“ ar „MacOS“, turėsite atlikti keletą nedidelių pakeitimų.

1.5 Interatyvus „R“ naudojimas

Kai naudojate „R“ programą, ji tikisi įvesties komandų. Numatytoji raginimo eilutė yra ‘>’, kuri UNIX sistemoje gali būti tokia pat, kaip apvalkalo eilutė, ir todėl gali pasirodyti, kad nieko

nevyksta. Tačiau, kaip matysime, jei norite, lengvai galite pakeisti į kitą „R“ raginimo eilutę. Darysime prielaidą, kad UNIX apvalkalo eilutė yra „$“.

Pirmą kartą naudojant „R“ programą operacinėje sistemoje UNIX, siūloma tokia procedūra:

1. Sukurkite atskirą pakatalogį, tarkime `work`, kad galėtumėte laikyti duomenų failus, kuriems apdoroti naudosite „R“ paketą. Tai bus darbinis katalogas, kai naudosite „R“ tam tikrai problemai spręsti.

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

2. Paleiskite „R“ programą su komanda

   ```
   $ R
   ```

3. Šiuo atveju gali būti vykdomos „R“ komandos (pamatysite vėliau).
4. Norėdami išeiti iš „R“ programos, naudokite komandą:

   ```
   > q()
   ```

Šiuo metu Jūsų paklaus, ar norite išsaugoti duomenis iš savo „R“ sesijos. Kai kuriose sistemose tai atvers dialogo langą, o kitose gausite teksto raginimą, į kurį galėsite atsakyti „taip“, „ne“ arba „atšaukti“ (tai padarys vienos raidės santrumpa), kad išsaugotumėte duomenis prieš užbaigiant darbą, išeitumėte neišsaugodami arba norėdami grįžti į „R“ sesiją. Išsaugoti duomenys bus prieinami būsimose „R“ sesijose.

Kitos „R“ sesijos yra paprastos.

1. Padarykite `work` darbiniu aplanku ir paleiskite programą kaip anksčiau:

   ```
   $ cd work
   $ R
   ```

2. Nutraukite „R“ programą naudodami komandą `q()` sesijos pabaigoje.

Norint naudoti „R“ operacinėje sistemoje „Windows“, iš esmės reikalinga ta pati procedūra. Sukurkite aplanką, kuris bus kaip darbinis katalogas ir nustatykite tai „R“ šaukinio `Start In` lauke. Tada du kartus spustelėdami piktogramą paleiskite „R“ paketą.

1.6 Pagalbos informacija apie funkcijas

„R“ turi integruotą pagalbos priemonę, panašią į UNIX priemonę `man`. Norėdami gauti daugiau informacijos apie konkrečią funkciją, pavyzdžiui `solve`, komanda yra tokia:

```
> help(solve)
```

Alternatyvi komanda yra:

```
> ?solve
```

Funkcijai, kurią nurodo specialieji ženklai, argumentas turi būti įterptas į dvigubas ar atskiras kabutes, paverčiant jį „simbolių eilute“: Tai taip pat būtina keliems žodžiams, turintiems sintaksinę reikšmę, įskaitant `if, for` ir `function.`

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

Bet kurios kabutės gali būti naudojamos kitai formai išvengti. Mūsų praktika yra tokia, kad pirmenybė teikiama dviguboms kabutėms.

Daugelyje „R“ diegimų visa pagalba HTML formatu galima naudojantis komanda

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

Ši komanda paleis interneto naršyklę, leidžiančią naršyti pagalbos puslapiuose naudojant hipersaitus. UNIX operacinėje sistemoje, vėlesnės pagalbos užklausos siunčiamos į HTML pagrįstą pagalbos sistemą. Nuoroda „Search Engine and Keywords“ puslapyje, kuri įkelta naudojantis `help.start()`, yra ypač naudinga, nes joje yra aukšto lygio sąvokų sąrašas, kuriame ieškoma pagal galimas funkcijas. Tai gali būti puikus būdas greitai gauti informaciją ir suprasti, ką „R“ duomenų statistinio apdorojimo paketas gali pasiūlyti.

`help.search` komanda leidžia ieškoti pagalbos įvairiais būdais. Pavyzdžiui,

```
> ??solve
```

Norėdami gauti daugiau informacijos ir daugiau pavyzdžių bandykite *`?help.search`*

Pavyzdžiai gali būti rangami ir naudojant komandą

```
> example(topic)
```

„Windows“ operacinėje sistemoje „R“ paketo versijos turi kitas pasirenkamas pagalbos sistemas. Norėdami gauti daugiau informacijos naudokite

```
> ?help
```

1.7 „R“ komandos, raidžių lygio paisymas ir kt.

Techniškai „R“ yra išraiškos kalba, turinti labai paprastą sintaksę. Šioje kalboje skiriamos didžiosios ir mažosios raidės, kaip ir daugumoje UNIX pagrindu sukurtų paketų, todėl didžioji raidė *A* ir mažoji raidė *a* yra skirtingi simboliai ir būtų susiję su skirtingais kintamaisiais. Simbolių rinkinys, kurį galima naudoti „R“ pavadinimuose, priklauso nuo operacinės sistemos ir šalies, kurioje įdiegtas „R“ paketas (kokia lokalė techniškai naudojama). Paprastai leidžiami visi raidiniai ir skaitmeniniai simboliai[1] (o kai kuriose šalyse tai yra kirčiuotos raidės) bei „.“ ir „_“, su apribojimu, kad pavadinimas turi prasidėti „.“ arba raide. Jeigu prasideda „.“, tai antras simbolis negali būti skaitmuo. Pavadinimų ilgis nėra ribojamas.

Elementarias komandas sudaro arba išraiškos, arba priskyrimai. Jei išraiška yra pateikiama kaip komanda, ji yra įvertinama, atspausdinama (nebent specialiai padaryta nematoma) ir reikšmė prarandama. Priskyrimas taip pat įvertina išraišką ir perduoda reikšmę kintamajam, tačiau rezultatas nėra automatiškai atspausdinamas.

Komandos yra atskiriamos kabliataškiu „;“, arba nauja linija. Elementarios komandos gali būti sujungtos į vieną junginį pasinaudojus riestiniais skliaustais „{ }“. Komentarus galima pateikti beveik bet kur[2] pradedant grotelių ženklu „#“. Tokiu atveju visas tekstas iki eilutės pabaigos yra komentaras.

Jei komanda neužsibaigta eilutės pabaigoje, tai „R“ pagal nutylėjimą pateiks skirtingą raginimo eilutę (+) antroje ir paskesnėse eilutėse bei toliau skaitys įvestį, kol komanda bus sintaksiškai baigta.

Šį raginimą vartotojas gali pakeisti. Paprastai praleisime pratęsimo raginimą ir nurodysime tęsimą paprasčiausiomis įtraukomis.

[1] Perkeliamame „R“ kode (įskaitant ir tuos, kurie naudojami „R“ pakete) gali būti naudojami tik A–Z, a–z, 0–9.

[2] tik ne eilutės viduje, nei funkcijos apibrėžimo argumentų sąraše.

Komandų eilutės, įvestos konsolėje, yra apribotos[3] iki maždaug 4095 baitų (ne simbolių).

1.8 Ankstesnių komandų atkūrimas ir redagavimas

Daugelyje operacinių sistemų UNIX ir „Windows“ versijų „R“ pateikia ankstesnių komandų prisiminimo ir pakartotinio jų vykdymo mechanizmą. Vertikalūs rodyklių klavišai klaviatūroje gali būti naudojami norint slinkti pirmyn ir atgal per komandų istoriją. Kai tokiu būdu komanda randama, žymeklis gali būti perkeltas į komandos vidų naudojant horizontalius rodyklių klavišus, o nereikalingi simboliai panaikinami pašalinimo klavišu arba įtraukiami su kitais klavišais.

1.9 Komandų vykdymas iš failo arba nukreipimas išvesties į failą

Jei komandos[4] yra saugomos išoriniame faile, veikiančiame kataloge naudokite `commands.R` darbiniame kataoge `work`. Tai gali būti vykdoma bet kuriuo metu „R“ sesijoje su komanda

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

„Windows“ operacinėje sistemoje pasinaudojus funkcija `sink`,

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

visos paskesnės išvestys iš konsolės bus nukreiptos į išorinį failą. Komanda, kuri vėl tai atkuria konsolėje yra

```
> sink()
```

1.10 Duomenų pastovumas ir objektų pašalinimas

Esybės, kurias sukuria „R“ ir jomis manipuliuoja, yra žinomi kaip objektai. Tai gali būti kintamieji, skaičių masyvai, simbolių eilutės, funkcijos arba iš tokių komponentų sudarytos bendrosios struktūros.

[3] kai kurios konsolės neleis įkelti daugiau eilučių. Kai kurios pašalins eilučių perteklių arba panaudos kaip kitos eilutės pradžią.

[4] neriboto ilgio.

„R“ sesijos metu objektai yra sukuriami ir saugomi pagal pavadinimą (šį procesą aptarsime kitame skyriuje). Naudojama „R“ komanda

```
> objects()
```

(alternatyvi komanda `ls()`) gali būti naudojama daugumos objektų pavadinimų, kurie šiuo metu saugomi „R“, rodymui. Šiuo metu saugoma objektų kolekcija vadinama darbo sritimi.

Norėdami pašalinti objektus naudokite funkciją `rm`:

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

Visus „R“ sesijos metu sukurtus objektus galima visam laikui išsaugoti faile, kad būtų galima naudoti būsimose „R“ sesijose. Kiekvienos „R“ sesijos pabaigoje jums suteikiama galimybė išsaugoti visus esamus objektus. Jei nurodote, kad norite tai padaryti, objektai įrašomi į failą, vadinamą `.RData`[5] ir esantį dabartiniame kataloge, o sesijoje naudojamos komandų eilutės įrašomos į failą, vadinamą `.Rhistory`.

Kai vėliau „R“ pradedamas iš to paties katalogo, jis perkelia darbo sritį iš šio failo. Tuo pačiu metu iš naujo įkeliama susijusių komandų istorija. Rekomenduojama, kad atliekant analizę su „R“ būtų naudojami atskiri darbo katalogai. Gana įprasta, kad analizės metu sukuriami objektai, kurių vardai yra *x* ir *y*. Tokie pavadinimai, kaip šie, dažnai būna prasmingi atliekant vieną analizę, tačiau gali būti gana sunku nuspręsti, kokie jie gali būti, kai kelios analizės buvo atliktos tame pačiame kataloge.

[5] Priekyje esantis taškas padaro failą nematomą įprastuose UNIX failų sąrašuose ir numatytuosiuose GUI failų sąrašuose „MacOS“ ir „Windows“.

2 Paprastos manipuliacijos. Skaičiai ir vektoriai

2.1 Vektoriai ir priskyrimai

„R“ dirba su duomenų struktūromis. Paprasčiausia tokia struktūra yra skaitinis vektorius, kuris yra vienas objektas, susidedantis ir rikiuoto skaitmenų rinkinio. Norėdami nustatyti vektorių, tarkime `x`, kurį sudaro penki skaičiai, būtent `10.4, 5.6, 3.1, 6.4` ir `21.7`, naudokite „R“ komandą

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

Tai yra priskyrimo sakinys naudojant funkciją `c()`, kuri šiame kontekste gali paimti pasirinktą skaičių vektoriaus argumentų ir kurios reikšmė yra vektorius, kuris gautas susiejant jo argumentus nuo pradžios iki galo[6].

Skaičius, kuris atsiranda išraiškoje, yra laikomas vieno ilgio vektoriu. Atkreipkite dėmesį, kad priskyrimo operatorius „`<-`“, kurį sudaro du simboliai „<“ („mažiau nei“) ir „−“ („minus“) turi būti griežtai vienas šalia kito ir nukreipia į objektą, kuris gaus išraiškos reikšmę. Daugelyje situacijų operatorius „=“ gali būti naudojamas kaip alternatyva.

Priskyrimą taip pat galima atlikti naudojantis funkcija `assign()`. Lygiavertis būdas atlikti tą patį priskyrimą, kaip aprašyta aukščiau, yra:

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

Įprastas operatorius „`<-`“ gali būti laikomas sintaksiniu trumpiniu. Paskyrimai taip pat gali būti atliekami kita kryptimi, naudojant akivaizdų priskyrimo operatoriaus pasikeitimą. Taigi tą pačią užduotį galima atlikti naudojant

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

Jei išraiška naudojama kaip visa komanda, tai reikšmė išspausdinama ir prarandama[7]. Taigi mes turėtume naudoti komandą

[6] Su kitais nei vektorinio tipo argumentais, pavyzdžiui, sąrašo argumentais, `c()` veiksmas yra gana skirtingas.

[7] Tiesą sakant, ji vis dar prieinama kaip `.Last.value` prieš vykdant kitus veiksmus.

```
> 1/x
```

ir penkių reikšmių ekvivalentas bus atspausdintas terminale (o `x` reikšmė, žinoma, nepakis).

Sekantis priskyrimas

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

sukurtų vektorių `y` su 11 įrašų, sudarytų iš dviejų `x` egzempliorių ir nuliu viduryje.

2.2 Vektorinė aritmetika

Vektoriai gali būti naudojami aritmetinėse išraiškose, tokiu atveju operacijos atliekamos kiekvienam elementui. Vektoriai, kurie yra toje pačioje išraiškoje, nebūtinai turi būti vienodo ilgio. Jei jie nėra vienodo ilgio, išraiškos reikšmė yra vektorius, kurio ilgis yra toks pat, kaip ilgiausio vektoriaus, kuris yra išraiškoje. Trumpesni išraiškos vektoriai yra perdirbami tiek dažnai, kiek reikia (galbūt nežymiai), kol jie atitinka ilgiausio vektoriaus ilgį. Visų pirma konstanta yra tiesiog kartojama. Taigi, su aukščiau nurodytais priskyrimais naudojama komanda

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

generuoja naują vektorių `v`, kurio ilgis 11, ir jis sukonstruotas sudedant elementą po elemento, `2*x` pakartojamas 2 kartus, `y` pakartojamas tik kartą ir 1 pakartojamas 11 kartų.

Elementarūs aritmetiniai operatoriai yra įprasti `+`, `-`, `*`, `/` ir `^` . Be to, yra visos įprastos aritmetinės funkcijos. Tokios fukcijos kaip: `log`, `exp`, `sin`, `cos`, `tan`, `sqrt`, ir t. t., turi savo įprastą reikšmę. `max` ir `min` atitinkamai parenka didžiausius ir mažiausius vektoriaus elementus. `range` yra funkcija, kurios reikšmė yra dviejų ilgių vektorius, būtent `c(min(x), max(x))`. `length(x)` yra elementų skaičius esantis `x`, `sum(x)` pateikia bendrą elementų, esančių `x` sumą, ir `prod(x)` jų produktą.

Dvi statistinės funkcijos yra `mean(x)`, kuri apskaičiuoja imties vidurkį, ir yra tokia pat kaip `sum(x)/length(x)`, ir `var(x)` kuri suteikia

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

arba imties dispersiją. Jei argumentas į `var()` yra [n × p] dydžio matrica, tai reikšmė yra [p × p] imties kovariacijos matrica, kuri gaunama įvertinus eilutes kaip nepriklausomus p-mačio imties vektorius.

`sort(x)` grąžina tokio paties dydžio vektorių kaip `x` su elementais, išdėstytais didėjančia tvarka; tačiau yra ir kitų lankstesnių rikiavimo galimybių (žiūrėti `order()` arba `sort.list()`, kurie sukuria permutaciją rikiavimui atlikti).

Pastaba, kad `max` ir `min` parenka didžiausias ir mažiausias reikšmes savo argumentuose, net jei joms būtų duoti keli vektoriai. Lygiagrečios maksimalios ir minimalios funkcijos `pmax` ir `pmin` grąžina vektorių (kurio ilgis lygus ilgiausiam jo argumentui), kurio kiekviename elemente yra didžiausias (mažiausias) elementas toje vietoje bet kuriame iš įvestų vektorių.

Daugeliu atvejų vartotojui nerūpės ar skaičiai, esantys skaitiniame vektoriuje, yra sveikieji, realieji ar kompleksiniai. Vidiniai skaičiavimai atliekami kaip dvigubi tikslūs realieji skaičiai arba dvigubi tikslūs kompleksiniai skaičiai, jei įvesties duomenys yra kompleksiniai.

Norėdami dirbti su kompleksiniais skaičiais, pateikite aiškią kompleksinę dalį. Taigi, `sqrt(-17)` pateiks `NaN` rezultatą ir įspėjimą, bet `sqrt(-17+0i)` atliks kompleksinio skaičiaus skaičiavimus.

2.3 Reguliarių sekų generavimas

„R“ turi daugybę galimybių generuoti dažniausiai naudojamas skaičių sekas. Pavyzdžiui, `1:30` yra vektorius `c(1, 2, ..., 29, 30)`. Dvitaškio operatorius turi didelę reikšmę išraiškoje, taigi, pvz., `2*1:15` yra vektorius `c(2, 4, ..., 28, 30)`. Konstrukcija `30:1` gali būti naudojama generuoti atbulinę seką.

Funkcija `seq()` yra bendresnė sekų generavimo priemonė. Ji turi penkis argumentus, iš kurių tik keli gali būti nurodyti viename iškvietime. Pirmieji du argumentai, jei jie pateikiami, nurodo sekos pradžią ir pabaigą, o jei tai yra vieninteliai du argumentai, rezultatas yra toks pats kaip naudojant dvitaškio operatorių. Todėl `seq(2,10)` yra tas pats vektorius kaip `2:10`.

Argumentai į `seq()`, ir daugeliui kitų „R“ funkcijų, taip pat gali būti suteikti įvardyta forma, tokiu atveju jų pasirodymo eiliškumas neturi reikšmės. Pirmieji du argumentai gali būti įvardyti `from=value` ir `to=value`; tokiu būdu `seq(1,30)`, `seq(from=1, to=30)` ir `seq(to=30, from=1)` visi yra tokie patys kaip `1:30`. Kiti du argumentai į `seq()` gali būti įvardyti `by=value` ir `length=value`, kurie atitinkamai nurodo žingsnio dydį ir ilgį. Jei nė vienas iš jų nėra nurodytas, laikoma, kad numatytoji reikšmė yra `by=1`.

Pavyzdžiui,

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

į `s3` generuoja vektorių `c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)`. Panašiai

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

generuoja tą patį vektorių į `s4`.

Galima įvardinti penktąjį argumentą `along=vector`, kuris paprastai naudojamas kaip vienintelis argumentas kuriant seką `1, 2, ..., length(vector)`, arba tuščią seką, jei vektorius tuščias.

Susijusi funkcija yra `rep()`, kurią galima panaudoti atkartojant objektą įvairiais sudėtingais būdais. Paprasčiausia forma yra

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

kuriame bus penki `x` egzemplioriai ištisiniame `s5`. Dar viena versija

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

kuris kartoja kiekvieną *x* elementą penkis kartus prieš pereidamas prie kito.

2.4 Loginiai vektoriai

Kaip ir su skaitiniais vektoriais, „R“ leidžia manipuliuoti loginiais dydžiais. Loginio vektoriaus elementai gali turėti reikšmes `TRUE`, `FALSE`, ir `NA` (trūkstama reikšmė). Pirmieji du dažnai atitinkamai sutrumpinami kaip `T` ir `F`. Tačiau atminkite, kad `T` ir `F` yra tik kintamieji, kurie

pagal numatytuosius nustatymus yra TRUE ir FALSE, tačiau nėra rezervuoti žodžiai, todėl vartotojas juos gali perrašyti. Taigi, jūs visada turėtumėte naudoti TRUE ir FALSE.

Loginius vektorius generuoja sąlygos. Pavyzdžiui,

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

nustato temp tokio paties ilgio kaip x vektorius, kurio reikšmės FALSE atitinka x elementus, kai nesilaikoma sąlygos, ir TRUE, priešingu atveju. Loginiai operatoriai yra <, <=, >, >=, == tiksliai lygybei ir != nelygybei. Jei c1 ir c2 yra loginės išraiškos, tada c1 & c2 yra jų sankirta („ir“), c1 | c2 yra jų sąjunga („arba“), ir !c1 yra neigiamas c1.

Loginiai vektoriai gali būti naudojami įprastoje aritmetikoje, tokiu atveju jie yra verčiami į skaitinius vektorius, FALSE tampa 0, o TRUE tampa 1. Tačiau yra situacijų, kai loginiai vektoriai ir jų priverstiniai skaitiniai ekvivalentai nėra lygiaverčiai.

2.5 Trūkstamos reikšmės

Kai kuriais atvejais vektoriaus komponentai gali būti nevisiškai žinomi. Kai statistine prasme elementas ar reikšmė yra „trūkstama reikšmė“, vieta vektoriuje gali būti rezervuota jam priskiriant specialiąją reikšmę NA. Apskritai bet kokia operacija esanti NA tampa NA. Ši taisyklė motyvuojama paprasčiausiai tuo, kad jei operacijos specifikacija yra neišsami, rezultatas negali būti žinomas ir todėl nėra prieinamas.

Funkcija is.na(x) pateikia tokio paties dydžio kaip x loginį vektorių, kurio reikšmė TRUE, tada ir tik tada, jei atitinkamas x elementas yra NA.

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

Atkreipkite dėmesį, kad loginė išraiška x == NA yra visiškai kitokia nei is.na(x) kadangi NA iš tikrųjų nėra reikšmė, bet kiekio, kurio nėra, žymuo. Taigi, x == NA yra vektorius, kurio ilgis toks pat kaip x, ir visos reikšmės yra NA, nes pati loginė išraiška yra neišsami ir todėl nenusakoma. Atkreipkite dėmesį, kad yra ir antros rūšies „trūkstamos“ reikšmės, kurios

gaunamos skaičiuojant, ir vadinamos „negalima skaitinė reikšmė“ (`NaN`). Abu pateikti pavyzdžiai

```
> 0/0
```

arba

```
> Inf - Inf
```

grąžina `NaN`, nes gauto rezultato negalima protingai apibrėžti. Apibendrinant `is.na(xx)` yra `TRUE` tiek `NA`, tiek `NaN` reikšmėms. Trūkstamos reikšmės kartais spausdinamos kaip `<NA>`, kai simbolių vektoriai spausdinami be kabučių.

2.6 Tekstinis vektorius

Tekstiniai vektoriai yra dažnai naudojami „R“ programoje, pavyzdžiui, kaip brėžinių etiketės. Prireikus jie yra žymimi simbolių seka, apribota kabutėmis, pvz., `"x-values"`, `"Nauji rezultatai"`.

Simbolių eilutės yra įvedamos naudojant atitinkamai kabutes (`"`) arba apostrofą (`'`), bet spausdinamos naudojant kabutes (arba kartais be kabučių). Tekstinis vektorius gali būti sujungtas į vektorių naudojant `c()` funkciją.

Funkcija `paste()` pasirenka argumentų skaičių ir susieja juos vieną po kito į simbolių eilutes. Bet kokie skaičiai, pateikti tarp argumentų, yra aiškiai įterpiami į simbolių eilutes, tai yra lygiai taip pat, jei jie būtų atspausdinti.

Pagal nutylėjimą argumentai rezultatuose yra atskirti vienu tuščiu simboliu, bet tai gali būti pakeista įvardytu argumentu `sep=string`, kuris pakeis tai į `string`, tikėtina tuščiu.

Pavyzdžiui,

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

komanda `labs` paverčia simbolių vektoriais

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

Ilgus sąrašus galime sutrumpinti, todėl `c("X", "Y")` yra kartojamas 5 kartus kad atitiktų

seką `1:10`.

`paste(..., collapse=ss)` sujungia argumentus į vieno simbolio eilutę, įterpdamas ss tarpe, pvz., `ss <- "|"`. Yra daugiau įrankių, skirtų manipuliuoti simboliais.

2.7 Duomenų rinkinio poaibių parinkimas ir keitimas

Vektoriaus elementų poaibiai gali būti parenkami prie vektoriaus pavadinimo laužtiniuose skliaustuose pridedant indekso vektorių. Apskritai bet kokia išraiška, kuris yra vertinama kaip vektorius, gali turėti savo elementų poaibius, kurie yra parenkami, kai iškart po išraiškos laužtiniuose skliaustuose pridedamas indekso vektorius.

Tokie indekso vektoriai gali būti naudojami bet kuriame iš šių keturių skirtingų tipų vektorių:

1. **Loginis vektorius.** Šiuo atveju indekso vektorius perdirbamas į tokį patį ilgį kaip ir vektorius, iš kurio turi būti parenkami elementai. Yra pasirenkamos reikšmės, kurios indekso vektoriuje atitinkanka `TRUE`, o reikšmės, kurios atitinkanka `FALSE`, yra praleidžiamos. Pavyzdžiui,

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

   sukuria (arba sukuria iš naujo) objektą `y`, kuriame bus netrūkstamų `x` reikšmių, išdėstytų ta pačia tvarka. Atminkite, kad jei `x` reikšmių trūksta, tai `y` bus trumpesnė nei `x`. Ir

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

   sukuria objektą `z` bei įdeda į jį vektoriaus `x+1` reikšmes, kurių atitinkama *x* reikšmė nebuvo trūkstama ir buvo teigiama.

2. **Teigiamų sveikųjų skaičių vektorius**. Šiuo atveju indekso vektoriaus reikšmės turi būti aibėje `{1, 2, . . . , length(x)}`. Atitinkami vektoriaus elementai yra parinkti ir sujungiami tokia tvarka kaip ir rezultatuose. Indekso vektorius gali būti bet kokio ilgio, o rezultatas yra tokio paties ilgio kaip indekso vektorius. Pavyzdžiui, `x[6]` yra šeštas `x` komponentas ir

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

pasirenka 10 pirmųjų `x` elementų (darant prielaidą, kad `length(x)` nėra mažiau už 10). Taip pat, mažai tikėtinas atvejis, kad

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

sukuria 16 ilgio simbolių vektorių, susidedantį iš keturis kartus kartojamų `"x"`, `"y"`, `"y"`, `"x"`.

3. **Neigiamų sveikųjų skaičių vektorius.** Toks indekso vektorius nurodo reikšmes, kurios turi būti pašalinamos, o ne įtraukiamos. Taigi,

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

pateikia visus `y`, išskyrus pirmuosius penkis `x` elementus.

4. **Simbolių eilučių vektorius.** Ši galimybė taikoma tik tada, kai objektas turi pavadinimų atributą, kad galėtų identifikuoti jo komponentus.

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

Privalumas yra tas, kad tekstinius pavadinimus dažnai lengviau įsimename nei skaitinius indeksus. Ši parinktis yra ypač naudinga naudojant duomenų sistemas.

Indeksuota išraiška taip pat gali pasirodyti priskyrimo gavimo pabaigoje; tokiu atveju priskyrimo operacija atliekama tik tiems vektoriaus elementams. Išraiška turi būti tokios formos `vector[index_vector]` ir pasirinktinė išraiška vietoje vektoriaus pavadinimo čia neturi daug prasmės. Pavyzdžiui,

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

pakeičia trūkstamas `x` reikšmes nuliais ir

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

turi tą patį poveikį kaip

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

3 Objektai, jų tipai ir atributai

3.1 Vidiniai požymiai: tipas ir ilgis

„R“ programoje esybės yra apibrėžiamos kaip objetaktai, tai gali būti vektoriai, apie kuriuos jau rašėme praeitame skyriuje. Pavyzdžiui, skaitinių (realiųjų), kompleksinių, loginių reikšmių vektoriai. Jie yra žinomi kaip „atominės“ struktūros, nes visi jų komponentai yra to paties tipo.

„R“ taip pat veikia su objektais, kurie vadinami sąrašais. Tai yra rikiuotos objektų sekos, kurios atskirai gali būti bet kokio tipo. Sąrašai yra žinomi kaip „rekursinės“, o ne atominės struktūros, nes jų komponentai patys gali būti sąrašai.

Kitos rekursinės struktūros yra funkcijos ir išraiškos. Funkcijos yra objektai, sudarantys „R“ sistemos dalį, kartu su panašiomis vartotojo parašytomis funkcijomis, kurias išsamiai aptarsime vėliau. Išraiškos kaip objektai sudaro išplėstinę „R“ dalį, kuri nebus aptariama šiame vadove, išskyrus netiesiogiai, kai mes aptariame formules, naudojamas modeliuojant „R“.

Objekto tipu nusakomas pagrindinis jo sudedamųjų dalių tipas. Kita kiekvieno objekto savybė yra jo ilgis. Funkcijos `mode(object)` ir `length(object)` gali būti naudojamos bet kurios apibrėžtos struktūros tipui ir ilgiui išsiaiškinti[8].

Tolesnes objekto savybes paprastai teikia `attributes(object)`. Dėl to tipas ir ilgis taip pat vadinami „vidiniais objekto atributais“. Pavyzdžiui, jei `z` yra kompleksinis vektorius, kurio ilgis 100, tada išraiškoje `mode(z)` simbolių eilutė bus „kompleksinė“, o `length(z)` bus `100`.

Pavyzdžiui,

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

mes galėtume įdėti

[8] Tačiau atminkite, kad `length(object)` ne visada pateikia naudingą informaciją, pvz., kai objektas yra funkcija.

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

po kurio skaitmenys yra tekstinis vektorius `c("0", "1", "2", ..., "9")`. Tolesnis pavertimas, arba tipo pakeitimas, vėl atkuria skaitinį vektorių:

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

Dabar `d` ir `z` yra tokie patys.[9] Yra gausus formos `as.something()` funkcijų rinkinys pavertimui iš vieno tipo į kitą, arba suteikiant objektui kitus atributus, kurių jis dar neturi.

3.2 Objekto ilgio keitimas

„Tuščias" objektas vis tiek gali turėti tipą. Pavyzdžiui,

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

sukuria `e`, kaip skaitinio tipo tuščią vektoriaus struktūrą. Panašiai `character()` yra tuščias tekstinis vektorius ir pan. Sukūrus bet kokio dydžio objektą, prie jo gali būti pridedami nauji komponentai, tiesiog suteikiant jam indekso reikšmę už ankstesnio diapazono ribų.

Taigi,

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

dabar `e` yra vektorius, kurio ilgis 3 (pirmieji du komponentai šiuo metu yra `NA`). Tai galioja bet kuriai struktūrai, jei papildomo(-ų) komponento(-ų) tipai pirmiausia atitinka objekto tipą.

Šis automatinis objekto ilgio reguliavimas yra naudojamas dažnai, pavyzdžiui įvesties funkcijoje `scan()`.

Priešingai, norint sutrumpinti objekto dydį, reikia tik priskyrimo padaryti tai. Vadinasi, jei `alpha` yra objektas, kurio ilgis 10, tada ši komandos eilutė

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

padaro objektą, kurio ilgis bus 5 ir kurį sudaro tik buvę komponentai su lyginiais indeksais. Senieji indeksai, žinoma, neišsaugomi. Tada mes galime išlaikyti tik pirmąsias tris reikšmes

[9] Pavertimas iš skaitmenų į simbolius ir grįžimas į tą patį tipą gali būti atliktas netiksliai, nes simbolių vaizdavime gali būti padarytos klaidos.

```
> length(alpha) <- 3
```

ir tuo pačiu būdu vektoriai gali būti išplėsti (praleidžiant reikšmes).

3.3 Atributų gavimas ir nustatymas

Funkcija `attributes(object)` grąžina visų šiuo metu tam objektui apibrėžtų neesminių požymių sąrašą. Funkcija `attr(object, name)` gali būti naudojama norint pasirinkti konkretų atributą. Šios funkcijos naudojamos retai, išskyrus gana ypatingas aplinkybes, kai tam tikram konkrečiam tikslui sukuriamas naujas atributas, pavyzdžiui, susieti sukūrimo datą arba operatorių su „R“ objektu.

Priskiriant ar šalinant atributus reikia būti atsargesniems, nes jie yra neatsiejama objektų sistemos, kuri naudojama „R“, dalis.

Pavyzdžiui,

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

leidžia „R“ traktuoti `z` taip, lyg tai būtų [10x10] dydžio matrica.

3.4 Objekto klasė

Visi objektai, kurie yra sukurti „R“, turi klasę, kurią nurodo funkcijų klasė. Paprastų vektorių atveju tai yra tipas, pavyzdžiui, `"numeric"` („skaitinis“), `"logical"` („loginis“), `"character"` („tekstinis“) arba `"list"` („sąrašas“), tačiau `"matrix"` („matrica“), `"array"` („masyvas“), `"factor"` („faktorius“) ir `"data.frame"` („duomenų sistema“) yra kitos galimos reikšmės.

Specialus atributas, kuris žinomas kaip objekto klasė, yra naudojamas tam, kad būtų galima taikyti objektinį programavimą „R“ pakete. Pavyzdžiui, jei objektas turi klasę `"data.frame"`, jis bus atspausdintas tam tikru būdu, ir funkcija `plot()` tam tikru būdu parodys tai grafiškai. Tada kitos, vadinamosios bendrosios fukcijos, tokios kaip `summary()`, reaguos į jį kaip argumentą, kuris būdingas šiai klasei.

Norėdami laikinai pašalinti klasės padarinius, naudokite funkciją `unclass()`. Pavyzdžiui, jei `winter` turi klasę `"data.frame"` tuomet

```
> winter
```

atspausdins duomenų sistemos pavidalu, kuri labiau panaši į matricą, kadangi

```
> unclass(winter)
```

atsispausdins kaip paprastą sąrašą.

4 Rikiuoti ir nerikiuoti faktoriai

Faktorius yra vektorinis objektas, naudojamas nurodyti to paties ilgio kitų vektorių komponentų klasifikaciją (grupavimą). Kitaip tariant, tai „R“ duomenų tipas, turintis iš anksto apibrėžtą galimų skirtingų reikšmių (kategorijų) skaičių.

4.1 Konkretus pavyzdys

Tarkime, kad mes turime uždavinį, kurio sąlygoje pateikiami 30 mokesčių apskaitininkų iš aštuonių Australijos valstijų duomenys. Jų individualūs duomenys apibrėžiami tekstiniu vektoriumi:

```
> 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")
```

Atminkite, kad tekstinio vektoriaus atveju „rikiuotas“ reiškia, kad surikiuota abėcėlės tvarka.

Faktorius panašiai sukuriamas naudojant funkciją `factor()`:

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

Funkcija `print()` tvarko faktorius šiek tiek skirtingai nei kitus objektus:

```
> 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
```

Norint sužinoti faktoriaus lygius, galima naudoti funkciją `levels()`

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

4.2 Funkcija `tapply()` ir nesulygiuotas masyvas

Tęskime ankstesnį pavyzdį, ir tarkime, kad turime tų pačių mokesčių apskaitininkų pajamas kitame vektoriuje

```
> 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)
```

Norėdami apskaičiuoti imties vidurkį, dabar galime naudoti specialiąją funkciją `tapply()`:

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

Funkcija `tapply()` naudojama funkcijai taikyti, čia `mean()` kiekvienai pirmojo argumento komponentų grupei ir `incomes` yra apibrėžtas antrojo komponento lygiais, čia `statef` tarsi būtų atskiros vektorinės struktūros.

Gautas vidurkių rezultatas pateikiamas vektoriumi su komponentais, kurie paženklinti pagal lygius:

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

Rezultatas yra struktūra tokio pat ilgio, kaip ir rezultatus turinčio faktoriaus lygių atributo. Tarkime, kad mums reikėjo apskaičiuoti pajamų vidurkio standartines paklaidas. Norėdami tai padaryti, turime parašyti „R“ funkciją, kad apskaičiuotume standartinę bet kurio vektoriaus paklaidą.

Kadangi imties dispersijai apskaičiuoti yra sukurta funkcija `var()`, tokia funkcija yra labai paprastai nurodoma priskyrimu:

```
> stdError <- function(x) sqrt(var(x)/length(x))
```

Po šios priskyrimo standartinės paklaidos apskaičiuojamos

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

ir tada apskaičiuotos vertės yra

```
> 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
```

Jums gali rūpėti rasti pajamų vidurkio 95% pasikliautiną intervalą. Norėdami tai padaryti, dar kartą naudokite `tapply()` su funkcija `length()`, kad rasti imties dydžius ir funkciją `qt()`, kad rasti atitinkamo *t*-skirstinio procentinius taškus.

Funkcija `tapply()` taip pat gali būti naudojama sudėtingesniam vektoriaus indeksavimui pagal kelias kategorijas. Pvz., galbūt norėtume padalinti mokesčių apskaitininkus tiek pagal valstijas, tiek pagal jų lytį. Reikšmės, esančios vektoriuje, yra surenkamos į grupes, pagal atitinkančius atskirus įrašus faktoriuje. Tada funkcija taikoma kiekvienai iš šių grupių atskirai. Reikšmė yra funkcijos rezultatų, kurie paženklinti faktoriaus atributu `level`, vektorius.

Vektoriaus ir ženklinimo faktoriaus derinys yra pavyzdys to, kas kartais vadinama nesulygiuotu masyvu, nes poklasių dydžiai gali būti netaisyklingi. Kai poklasių dydžiai yra vienodi, indeksavimas gali būti atliekamas netiesiogiai ir daug efektyviau.

4.3 Rikiuoti faktoriai

Faktorių lygiai yra saugomi abėcėlės tvarka arba tokia tvarka, kuria jie buvo nurodyti saugoti `factor`, jei tai buvo aiškiai nurodyta.

Kartais lygiai turės natūralų išrikiavimą, kurį norime įrašyti ir norime, kad būtų panaudota statistinėje analizėje. Funkcija `ordered()` sukuria tokius rikiuotus faktorius, bet kitu atveju jie yra identiški `factor`. Daugeliu atvejų vienintelis skirtumas tarp rikiuotų ir nerikiuotų faktorių yra tas, kad pirmieji yra atspausdinti, nurodant lygių išdėstymą, bet kontrastai, kurie generuojami taikant tiesinius modelius, yra skirtingi.

5 Masyvai ir matricos

5.1 Masyvai

Masyvas, tai duomenų struktūra, sudaryta iš to paties duomenų tipo komponentų – elementų. Elementas įvardijamas masyvo vardu ir indeksu, nurodančiu konkretų elementą masyve. „R“ suteikia paprastas galimybes kurti ir valdyti masyvus, ypač specialius matricų atvejus.

Dimensijų vektorius yra neneigiamų skaičių vektorius. Jei jo ilgis yra `k`, tada masyvas yra k-dimensijų, pvz. matrica yra 2 dimensijų masyvas. Dimensijos indeksuojamos nuo vieno iki reikšmių, nurodytų dimensijų vektoriuje.

„R“ gali naudoti vektorių kaip masyvą tik tuo atveju, jei jis turi dimensijų vektorių kaip `dim` atributą. Tarkime, kad `z` yra 1500 elementų vektorius.

Priskyrimas

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

suteikia `dim` atributą ir tai leidžia jį traktuoti kaip $3 \times 5 \times 100$ masyvą

Kitos funkcijos tokios kaip `matrix()` ir `array()` yra prieinamos paprastesniems ir natūraliau atrodantiems priskyrimams.

Pavyzdžiui, jei masyvas `a` yra dimensijų vektorius, kur `c(3,4,2)`, tada `a` turi 3 x 4 x 2 = 24 įrašus, o duomenų vektorius juos laiko eilės tvarka `a[1,1,1], a[2,1,1],... , a[2,4,2], a[3,4,2]`.

Masyvai gali būti vienmačiai, tokie masyvai paprastai traktuojami taip pat, kaip vektoriai (įskaitant ir spausdinant), tačiau išimtys gali sukelti painiavą.

5.2 Masyvo indeksavimas

Atskiri masyvo elementai gali būti susiejami nurodant masyvo pavadinimą, po kurio laužtiniuose skliaustuose, atskirtais kableliais, pateikiami indeksai.

Apskritai, masyvo poskyriai gali būti patikslinti pateikiant indeksų vektorių seką vietoje indeksų; tačiau jei bet kuriai rodyklės pozicijai pateikiamas tuščias indeksų vektorius, imamas visas to indekso diapazonas.

Pavyzdžiui, pasinaudojus prieš tai buvusiu pavyzdžiu, `a[2, , ]` yra 4×2 masyvas, turintis dimensijų vektorių `c(4,2)` ir duomenų vektorių, kuris turi tokia tvarka išdėstytas reikšmes

```
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])
```

`a[ , , ]` žymi visą masyvą, kuris yra toks pats, kaip visiškai pašalinus indeksus ir naudojant vieną patį `a`.

Bet kokiam masyvui, tarkime `Z`, dimensijų vektorius gali būti aiškiai nurodytas kaip `dim(Z)` (bet kurioje priskyrimo pusėje).

Be to, jei masyvo pavadinimas pateikiamas tik su vienu indeksu ar indeksų vektoriumi, tai naudojamos tik atitinkamos duomenų vektoriaus reikšmės, tokiu atveju dimensijų vektorius yra ignoruojamas. Tačiau tai netaikoma tuo atveju, jei pavienis indeksas nėra vektorius, o yra masyvas, kaip mes aptarsime toliau.

5.3 Indeksų matricos

Taip pat kaip indekso vektorius bet kurioje indekso vietoje, matrica gali būti naudojama su viena indekso matrica, norint priskirti dydžių vektorių netaisyklingam elementų rinkiniui masyve, arba išgauti netaisyklingą rinkinį kaip vektorių.

Matricos pavyzdys paaiškina procesą. Dvigubai indeksuojamo masyvo atveju indekso matrica gali susidaryti iš dviejų stulpelių ir tiek eilučių, kiek norima. Indeksų matricos įrašai yra dvigubai indeksuoto masyvo eilučių ir stulpelių indeksai.

Tarkime, pavyzdžiui, turime 4×5 masyvą `X` ir mes norime atlikti šiuos veiksmus:

- ištraukti elementus `X[1,3], X[2,2]` ir `X[3,1]` kaip vektorinę struktūrą, ir

- pakeisti šiuos įrašus `X` masyve nuliais.

Šiuo atveju mums reikia 3 × 2 indeksų masyvo, kaip pateiktame pavyzdyje.

```
> x <- array(1:20, dim=c(4,5)) # Sukurkite 4 x 5 masyvą.
> x
     [,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
> i <- array(c(1:3,3:1), dim=c(3,2))
> i                              # i yra 3 x 2 indeksų masyvas.
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> x[i]                           # Ištraukite tuos elementus
[1]9 6 3
> x[i] <- 0                      # Pakeiskite elementus nuliais
> x
     [,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
>
```

Indeksų matricose neigiami indeksai neleidžiami. `NA` ir nulio reikšmės yra leidžiamos. Indekso matricos, kurioje yra nulis, eilutės yra ignoruojamos, o eilutės, kuriose yra `NA`, rezultatas būna `NA`.

5.4 Funkcija array()

Suteikiant vektoriaus struktūrai `dim` atributą, masyvus iš vektorių galima sudaryti naudojant masyvo funkciją, kuri turi formą

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

Pavyzdžiui, jei vektorių `h` sudaro 24 ar mažiau skaičių, tada komanda

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

naudotų `h`, norėdama nustatyti 3 x 4 x 2 masyvą, esantį `Z`. Jei `h` dydis yra tiksliai 24, rezultatas yra tas pats kaip

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

Tačiau jei `h` yra trumpesnis nei 24, jo reikšmės iš naujo perdirbamos, kad jis padidėtų iki 24 skaičių, bet `dim(h) <- c(3,4,2)` reikštų klaidą dėl ilgio neatitikimo.

Kaip kraštutinis, bet dažnas pavyzdys

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

padaro `Z` visų nulių masyvą.

Šiuo atveju `dim(Z)` reiškia dimensijų vektorių `c(3,4,2)`, ir `Z[1:24]` reiškia duomenų vektoriui, koks jis buvo `h`, ir `Z[]` su tuščiais indeksais arba `Z` be indekso reiškia visą masyvą kaip masyvą.

Masyvai gali būti naudojami aritmetinėse išraiškose, o rezultatas yra masyvas, suformuotas atliekant kiekvieno elemento operacijas duomenų vektoriuje. Operandų atributai `dim` paprastai turi būti vienodi, ir tai tampa rezultato dimensijų vektoriu. Todėl, jei `A`, `B` ir `C` visi panašūs masyvai, tada

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

sukuria `D` panašiu masyvu, kai jo duomenų vektorius yra duotų kiekvienam elementui skirtų operacijų rezultatas. Tačiau reikia šiek tiek atidžiau apsvarstyti tikslią taisyklę, susijusią su mišraus masyvo ir vektorių skaičiavimais.

5.4.1 Mišraus vektoriaus ir masyvo aritmetika

Tiksli taisyklė, turinti įtakos mišriam elementų skaičiavimui su vektoriais ir masyvais, yra šiek tiek keista ir sunkiai randama nuorodose. Keletas patarimų:

- Išraiška turi būti nuskaityta iš kairės į dešinę.
- Bet kokie trumpi vektoriniai operandai praplėčiami perdirbant jų reikšmes, kol jie atitinka kitų operandų dydį.
- Kol susiduriama tik su trumpais vektoriais ir masyvais, visi masyvai turi turėti tą patį `dim` atributą arba įvyksta klaida.

- Bet kuris vektoriaus operandas, ilgesnis už matricos ar masyvo operandą, sukuria klaidą.
- Jei yra masyvo struktūros ir vektoriui nebuvo nustatyta jokių klaidų arba pavertimų, tai rezultatas yra masyvo struktūra, turinti bendrą masyvo operandų atributą `dim`.

5.5 Išorinis dviejų masyvų produktas

Svarbi masyvų operacija yra išorinis produktas. Jei `a` ir `b` yra du skaitiniai masyvai, jų išorinis produktas yra masyvas, kurio dimensijų vektorius gaunamas sujungiant jų dviejų dimensijų vektorius (tvarka svarbi), ir kurio duomenų vektorius gaunamas suformuojant visus galimus duomenų vektoriaus `a` elementų produktus su `b` elementais. Išorinį produktą formuoja specialus operatorius `%o%`:

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

Alternatyva yra

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

Daugybos funkcija gali būti pakeista pasirinkta dviejų kintamųjų funkcija. Pavyzdžiui, jei norėtume įvertinti funkciją $f(x;y) = \cos(y)/(1+x^2)$ per įprastą reikšmių lentelę su x ir y koordinatėmis, kurios „R“ programoje atitinkamai apibrėžtos vektoriais `x` ir `y`, galėtume elgtis taip:

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

Visų pirma dviejų įprastų vektorių išorinis produktas yra dvigubai indeksuotas masyvas (yra matrica, kurios rangas ne didesnis kaip 1). Atkreipkite dėmesį, kad išorinio produkto operatorius, žinoma, nėra komutatyvus.

5.6 Apibendrintas masyvo perkėlimas

Funkcija `aperm(a, perm)` gali būti naudojamas masyvui `a` pakeisti. Argumentas `perm` turi būti sveikųjų skaičių permutacija *{1, ... ,k}*, kur *k* yra indeksų skaičius esantis `a`. Funkcijos rezultatas yra masyvas tokio paties dydžio kaip `a`, tačiau su senomis dimensijomis, kurias

suteikė `perm[j]` tapdama naujaja *j*-tąja dimensija. Lengviausias būdas galvoti apie šią operaciją yra apibendrinti matricų perkėlimą. Iš tikrųjų, jei `A` yra matrica, tada `B` pateiktas

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

yra tik `A` perkėlimas. Šiuo ypatingu atveju yra galima paprastesnė funkcija `t()`, taigi mes galėjome panaudoti `B <- t(A)`.

5.7 Matricų galimybės

Kaip pažymėta pirmiau, matrica yra tik masyvas su dviem indeksais. Tačiau tai yra toks svarbus ypatingas atvejis, kurį reikia atskirai aptarti. „R“ yra daugybė operatorių ir funkcijų, prieinamų tik matricoms. Pavyzdžiui, `t(X)` yra matricos perkėlimo funkcija. Funkcijos `nrow(A)` ir `ncol(A)` atitinkamai nurodo eilučių ir stulpelių skaičių `A` matricoje.

5.7.1 Matricų daugyba

Operatorius `%*%` yra naudojamas matricų daugybai. [n × 1] arba [1 × n] dydžio matrica, žinoma, gali būti naudojama kaip *n*-vektorius, jei tai tinka kontekste.

Atvirkščiai vektoriai, atsirandantys matricos daugybos išraiškose, yra automatiškai paaukštinami arba į eilučių, arba į stulpelių vektorius, atsižvelgiant į tai, kuris variantas yra nuoseklus.

Jei, pavyzdžiui, `A` ir `B` yra to paties dydžio kvadratinės matricos, tada

```
> A*B
```

yra elementas po elemento produkto matrica ir

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

yra matricos produktas. Jei `x` yra vektorius, tada

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

yra kvadratinė forma. Funkcija `crossprod()` formuoja „kryžminius produktus“, tai reiškia, kad `crossprod(X, y)` yra tas pats kaip `t(X) %*% y`, tačiau operacija yra efektyvesnė. Jei antrasis argumentas į `crossprod()` yra praleistas, laikoma, kad jis sutampa su pirmuoju.

`Diag()` reikšmė priklauso nuo jo argumento. Funkcija `diag(v)`, kur `v` yra vektorius, pateikia diagonalinę matricą, kurios įstrižainės įrašai yra vektoriaus elementai.

Iš kitos pusės `diag(M)`, kur `M` yra matrica, pateikia šios matricos įrašų pagrindinių įstrižainių vektorių.

5.7.2 Tiesinės lygtys ir inversija

Tiesinių lygčių sprendimas yra atvirkštinis matricos dauginimas. Kai po komandos

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

pateikiami tik `A` ir `b`, tai vektorius `x` yra tos tiesinės lygčių sistemos sprendimas. „R“ programoje komanda

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

išsprendžia sistemą, grąžindama `x` (iki tam tikro tikslumo praradimo). Atkreipkite dėmesį, kad tiesinėje algebroje formaliai $x = A^{-1}b$, kur A^{-1} žymi atvirkštinę `A`, kurią galima apskaičiuoti pasinaudojus `solve(A)`, bet retai taip reikia.

Skaitmenine prasme, neefektyvu ir potencialiai nestabilu apskaičiuoti `x <- solve(A) %*% b` vietoj `solve(A,b)`.

Kvadratinė forma $x^TA^{-1}x$ kuri yra naudojama daugiamačiuose skaičiavimuose, turėtų būti apskaičiuojama maždaug taip `x %*% solve(A,x)`, o ne skaičiuojant atvirkštinę `A`.

5.7.3 Matricos skaidymas singuliariomis reikšmėmis ir determinantai

Funkcija `svd(M)` ima pasirenkamą matricos `M` argumentą ir skaičiuoja matricos `M` skaidymą singuliariomis reikšmėmis. Jį sudaro ortonormalinių `U` stulpelių matrica su ta pačia stulpelio erdve kaip ir `M`, antroji ortonormalinių `V` stulpelių matrica, kurios stulpelio erdvė yra `M` eilutės erdvė ir teigiamų įrašų įstrižainė matrica tokia, kad `M = U %*% D %*% t(V)`. `D` iš tikrųjų grąžinamas kaip įstrižinių elementų vektorius. `svd(M)` rezultatas iš tikrųjų yra trijų komponentų, pavadintų `d`, `u` ir `v` sąrašas su aiškiomis reikšmėmis.

Jei `M` iš tikrųjų yra kvadratas, tada nėra sunku pastebėti, kad

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

apskaičiuoja `M` determinanto absoliučiąją reikšmę. Jei šio skaičiavimo prireiktų dažnai naudojant įvairias matricas, tai galima būtų apibūdinti kaip „R“ funkciją

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

po to galėtume naudoti `absdet()` kaip dar vieną „R“ funkciją. Kaip dar vieną trivialų, bet potencialiai naudingą pavyzdį, galbūt norėtumėte apsvarstyti funkcijos `tr()` rašymą, norint apskaičiuoti kvadratinės matricos pėdsaką.

5.8 Blokinių matricų formavimas

Matricas galime sudaryti iš kitų vektorių ir matricų pasinaudoję funkcijomis `cbind()` ir `rbind()`. `cbind()` formuoja matricas sujungdamos matricas horizontaliai arba stulpeliais, o funkcija `rbind()` formuoja matricas sujungdamos matricas vertikaliai arba eilutėmis.

Priskyrime

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

argumentas į `cbind()` turi būti bet kokio ilgio vektoriai, arba to paties stulpelio dydžio matricos, tai yra tas pats eilučių skaičius. Rezultatas yra matricos su susietais argumentais `arg 1`, `arg 2`, `. . .`, kurie formuoja stulpelius.

Jei kai kurie argumentai į `cbind()` yra vektoriai, jie gali būti trumpesni nei bet kurių esamų matricų stulpelio dydis, tokiu atveju jie cikliškai išplečiami, kad atitiktų matricos stulpelio dydį (arba ilgiausio vektoriaus ilgį, jei matricos nepateiktos).

Funkcija `rbind()` atlieka atitinkamą eilutėms skirtą operaciją. Šiuo atveju, bet kokie vektoriniai argumentai, galbūt cikliškai išplėsti, žinoma, laikomi eilučių vektoriais.

Tarkime, `X1` ir `X2` turi tą patį eilučių skaičių. Norėdami sujungti juos pagal stulpelius į `X` matricą kartu su pradiniu stulpeliu, galime naudoti

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

`rbind()` ir `cbind()` rezultatas visada turi matricos būseną. Taigi `cbind(x)` ir `rbind(x)` yra galbūt paprasčiausi būdai, aiškiai leidžiantys vektorių *x* atitinkamai traktuoti kaip stulpelį ar eilutės matricą.

5.9 Susiejimo funkcija `c()`

Reikėtų pažymėti, kad kadangi `cbind()` ir `rbind()` yra sujungimo funkcijos susijusios su `dim` atributais, pagrindinė `c()` funkcija to nedaro, o išvalo visų `dim` ir `dimnames` atributų skaitinius objektus.

Oficialus būdas priversti masyvą grįžti į paprastą vektorinį objektą yra naudoti `as.vector()`

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

Tačiau panašų rezultatą galima pasiekti naudojant funkciją `c()` ir pateikiant tik vieną argumentą, tiesiog dėl šio šalutinio poveikio:

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

Tarp šių dviejų būdų yra nežymūs skirtumai, tačiau galiausiai pasirinkimas tarp jų priklauso nuo stiliaus.

5.10 Dažnių lentelės iš faktorių

Prisiminkite, kad faktorius apibrėžia skaidymą į grupes. Panašiai faktorių pora apibrėžia dviejų krypčių kryžminį klasifikavimą ir pan. Funkcija `table()` leidžia apskaičiuoti dažnio lenteles iš vienodo ilgio faktorių. Jei yra *k* faktoriaus argumentai, tai rezultatas yra *k*-krypčių pasikartojimų masyvas.

Tarkime, kad `statef` yra faktorius, suteikiantis būsenos kodą kiekvienam įrašui duomenų vektoriuje. Priskyrimas

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

pateikia kiekvienos būsenos imtyje dažnių lentelę. Dažniai yra surikiuoti ir paženklinti faktoriaus lygių atributu. Šis paprastas atvejis yra lygiavertis, bet patogesnis nei

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

Be to, tarkime, kad `incomef` yra faktorius, suteikiantis tinkamai apibrėžtą klasę kiekvienam duomenų vektoriaus įrašui. Pavyzdžiui, su funkcija `cut()`:

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

Tada apskaičiuokite dvikryptę dažnių lentelę:

```
> 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
```

6 Sąrašai ir duomenų sistemos

6.1 Sąrašai

Programoje „R“ sąrašas yra objektas, kurį sudaro surikiuotas objektų, kurie vadinami jo komponentais, rinkinys. Nereikia, kad komponentai būtų to paties tipo, pavyzdžiui, sąrašą gali sudaryti skaitinis vektorius, loginė reikšmė, matrica, kompleksinis vektorius, simbolių masyvas, funkcija ir t.t. Čia yra paprastas pavyzdys, kaip sudaryti sąrašą:

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

Komponentai visada sunumeruojami ir visada gali būti nurodyti. Taigi, jei `Lst` yra sąrašo su keturiais komponentais pavadinimas, tai komponentai gali būti atskirai vadinami `Lst[[1]]`, `Lst[[2]]`, `Lst[[3]]` ir `Lst[[4]]`. Jei toliau, `Lst[[4]]` yra vektoriaus indeksuotas masyvas tada `Lst[[4]][1]` yra pirmasis jo įrašas.

Jei `Lst` yra sąrašas, tada funkcija `length(Lst)` nurodo turimų (aukščiausio lygio) komponentų skaičių.

Sąrašų komponentai taip pat gali būti pavadinti, o tokiu atveju komponentas gali būti nurodytas pateikiant komponento pavadinimą kaip simbolių eilutę vietoj skaičiaus dvigubuose skliaustuose arba pateikiant tokios formos išraišką

```
> name$component_name
```

Tai labai naudinga praktika, nes pamiršus skaičių lengviau gauti reikiamą komponentą. Taigi, pateikiame lengvus pavyzdžius:

- `Lst$name` yra tas pats kaip `Lst[[1]]` ir eilutė yra `"Fred"`,
- `Lst$wife` yra tas pats kaip `Lst[[2]]` ir eilutė yra `"Mary"`,
- `Lst$child.ages[1]` yra tas pats kaip `Lst[[4]][1]` ir skaičius yra `4`.

Be to, sąrašo komponentų pavadinimus taip pat galima naudoti dvigubuose skliausteliuose, pvz., `Lst[["name"]]` yra tas pats kaip `Lst$name`. Tai ypač naudinga, kai išgaunamo komponento pavadinimas saugomas kitame kintamajame kaip

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

Labai svarbu atskirti `Lst[[1]]` nuo `Lst[1]`. `'[[...]]'` yra operatorius, naudojamas pasirinkti vieną elementą, kadangi `'[...]'` yra bendras indeksuojamas operatorius. Taigi ankstesnis yra pirmasis objektas sąraše `Lst`, o jei yra įvardytas sąrašas, tai pavadinimas neįtrauktas. Pastarasis yra sąrašo pogrupis `Lst`, kurį sudaro tik pirmasis įrašas. Jei tai įvardytas sąrašas, pavadinimai perkeliami į pogrupį.

Komponentų pavadinimai gali būti sutrumpinti iki minimalaus raidžių skaičiaus, reikalingo norint juos unikaliai identifikuoti. Taigi, `Lst$coefficients` gali būti nurodytas kaip `Lst$coe` ir `Lst$covariance` kaip `Lst$cov`.

Pavadinimų vektorius iš tikrųjų yra tiesiog sąrašo atributas, kaip ir bet kuris kitas, ir gali būti laikomas tokiu.

6.2 Sąrašų sudarymas ir modifikavimas

Iš esamų objektų naujus sąrašus gali sudaryti funkcija `list()`. Priskyrimas

```
> Lst <- list(name_1=object_1, ..., name_m=object_m)
```

sukuria `m` komponentų sąrašą `Lst` naudojant `object_1,..., object_m` komponentams ir suteikiant jiems pavadinimus, nurodytus argumentų pavadinimuose, kuriuos galima laisvai pasirinkti.

Jei šie pavadinimai praleisti, komponentai sunumeruojami. Sudarant naują sąrašą, sąrašo sudarymui naudojami komponentai nukopijuojami, o originalams tai nedaroma. Sąrašus, kaip ir bet kurį indeksuojamą objektą, galima išplėsti nurodant papildomus komponentus. Pavyzdžiui

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

6.2.1 Sąrašų sujungimas

Kai sujungimo funkcijoje `c()` kaip argumentai pateikiami sąrašai, tai rezultatas taip pat yra sąrašo tipo objetas, kurio komponentai yra argumentų sąrašai, kurie sujungti iš eilės.

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

Prisiminkite, kad su vektoriniais objektais kaip argumentais sujungimo funkcija panašiai sujungia visus argumentus į vieną vektorinę struktūrą. Tokiu atveju visi kiti atributai, tokie kaip `dim` atributai, yra atmetami.

6.3 Duomenų sistemos

Duomenų sistema yra sąrašas su klase `"data.frame"`. Sąrašams, iš kurių gali būti sudaromos duomenų sistemos, yra apribojimų:

1. komponentai turi būti vektoriai (skaitiniai, simboliai arba loginiai), faktoriai, skaitinės matricos, sąrašai ar kitos duomenų sistemos;
2. matricos, sąrašai ir duomenų sistemos pateikia kuo daugiau kintamųjų naujai duomenų sistemai, nes atitinkamai jie turi stulpelius, elementus arba kintamuosius.
3. skaitiniai vektoriai, loginiai operatoriai ir faktoriai yra įtraukiami tokie, kokie yra, ir pagal nutylėjimą, simbolių vektoriai yra priversti būti faktoriais, kurių lygiai yra unikalios reikšmės, rodomos vektoriuje.
4. vektorinės struktūros, rodomos kaip duomenų sistemos kintamieji, turi būti vienodo ilgio, o visos matricinės struktūros turi turėti tą patį eilutės dydį.

Duomenų sistema daugeliu atvejų gali būti laikoma matrica, kurios stulpeliai gali būti skirtingų tipų ir atributų. Jis gali būti rodomas matricos forma, o jo eilutės ir stulpeliai ištraukiami naudojant matricos indeksavimo tvarką.

6.3.1 Duomenų sistemų kūrimas

Objektai, tenkinantys duomenų sistemos stulpeliams (komponentams) nustatytus apribojimus, gali būti naudojami formuojant juos naudojant funkciją `data.frame`:

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

Sąrašas, kurio komponentai atitinka duomenų sistemos apribojimus, gali būti verčiamas į duomenų sistemą, naudojant funkciją `as.data.frame()`

Paprasčiausias būdas sukurti duomenų sistemas nuo nulio yra naudoti funkciją `read.table()` ir nuskaityti visą duomenų rėmelį iš išorinio failo.

6.3.2 `attach()` ir `detach()`

Notacija `$`, tokia kaip `accountants$home`, sąrašo komponentams ne visada yra labai patogi. Naudinga priemonė būtų kažkokiu būdu padaryti sąrašo ar duomenų sistemos komponentus laikinai matomus kaip kintamuosius jų komponento pavadinime, nereikia kiekvieną kartą aiškiai nurodyti sąrašo pavadinimo. Funkcija `attach()` paima duomenų bazę kaip sąrašą ar duomenų sistemą ir laiko savo argumentu. Taigi, tarkime `lentils` yra duomenų sistema su trim kintamaisiais `lentils$u`, `lentils$v`, `lentils$w`. Komanda

```
> attach(lentils)
```

padeda duomenų sistemą į paieškos kelio antrą poziciją su sąlyga, kad kintamųjų `u`, `v` ir `w` nėra pirmoje pozicijoje bei `u`, `v` ir `w` galima naudoti kaip kintamuosius iš duomenų sistemos.

Šiuo atveju toks paskyrimas, kaip

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

nepakeičia duomenų sistemos komponento `u`, veikiau maskuoja jį darbiniame kataloge kitu kintamuoju `u` paieškos kelio primoje pozicijoje. Pačios duomenų sistemos pakeitimas yra paprasčiausias būdas dar kartą panaudoti `$` notaciją:

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

Tačiau nauja komponento `u` reikšmė nematoma tol, kol duomenų sistema bus atsieta ir pridėta dar kartą.

Norėdami atsieti duomenų sistemą, naudokite funkciją

```
> detach()
```

Tiksliau sakant, ši komanda objektą atsieja nuo paieškos kelio, kuris šiuo metu yra antroje pozicijoje. Taigi, dabartiniame kontekste kintamieji `u`, `v` ir `w` nebebus matomi, išskyrus tuos atvejus, kai sąraše yra naudojamos tokios notacijos kaip `lentils$u` ir pan. Objektus, esančius paieškos kelio aukštesnėje nei 2 pozicijoje, galima atsiesti nurodant jų skaičių atsiejimui, bet visada saugiau naudoti pavadinimą, pavyzdžiui, `detach(lentils)` ir `detach("lentils")`

Pastaba: Programoje „R“ sąrašai ir duomenų sistemos gali būti pridedamos tik 2-oje ar aukštesnėje pozicijoje, o tai, kad pridedama yra originalaus objekto kopija. Pridėtas reikšmes galite pakeisti priskirdami, tačiau originalus sąrašas arba duomenų sistema nesikeičia.

6.3.3 Darbas su duomenų sistemomis

Naudingi susitarimai leidžiantys patogiai dirbti su daugeliu skirtingų problemų tame pačiame darbiniame kataloge:

- duomenų rinkinyje tinkamai informatyviu pavadinimu surinkite visus kintamuosius bet kuriai gerai apibrėžtai ir atskirai problemai;
- dirbdami su problema 2 pozicijoje, priskirkite atitinkamą duomenų sistemą, o operacijų kiekiams ir laikiniesiems kintamiesiems naudokite 1 lygio darbinį katalogą;
- prieš užbaigdami problemą, pridėkite visus kintamuosius, kuriuos norite išsaugoti, kad ateityje galėtumėte naudoti duomenų sistemą naudodami priskyrimo formą `$` ir tada `detach();`
- galiausiai pašalinkite visus nepageidaujamus kintamuosius iš darbinio katalogo ir laikykite jį kiek įmanoma švaresnį nuo likusių laikinų kintamųjų.

Tokiu būdu yra gana paprasta dirbti su daugeliu problemų tame pačiame kataloge, pavyzdžiui, visi jie turi kintamuosius įvardytus `x`, `y` ir `z`.

6.3.4 Pasirinktinis sąrašų pridėjimas

`attach()` yra bendroji funkcija, leidžianti prie paieškos kelio pridėti ne tik katalogus ir duomenų sistemas, bet ir kitas objekto klases. Visų pirma, bet koks sąrašo tipo objektas gali būti pridedamas tokiu pat būdu:

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

Viską, kas buvo pridėta, galima atsieti pasinaudojus `detach()`, pagal pozicijos numerį arba, pageidautina, pagal pavadinimą.

6.3.5 Paieškos kelio valdymas

Funkcija *search* rodo dabartinį paieškos kelią, todėl yra labai naudingas būdas sekti, kurios duomenų sistemos ir sąrašai (ir paketai) buvo pridėti ir atskirti. Iš pradžių pateikiama

```
> search()
[1] ".GlobalEnv" "Autoloads"   "package:base"
```

čia `.GlobalEnv` yra darbo sritis.

Po to, kai lentils yra prijungtas mes turime

```
> search()
[1] ".GlobalEnv" "lentils"     "Autoloads"   "package:base"
> ls(2)
[1] "u" "v" "w"
```

ir kaip mes matome, `ls` (arba `objects`) gali būti naudojamas bet kurios paieškos kelio pozicijos turiniui ištirti.

Galiausiai atsiejame duomenų sistemą ir patvirtiname, kad ji buvo pašalinta iš paieškos kelio.

```
> detach("lentils")
> search()
[1] ".GlobalEnv" "Autoloads"   "package:base"
```

7 Duomenų skaitymas iš failų

Dideli duomenų objektai paprastai bus skaitomi kaip reikšmės iš išorinių failų, o ne įvedami klaviatūroje „R“ sesijos metu. „R“ įvesties priemonės yra paprastos, tačiau jų reikalavimai yra gana griežti ir netgi gana nelankstūs. „R“ programos kūrėjai aiškiai suprato, kad jūs galėsite modifikuoti savo įvesties failus naudodami kitas programas, kad atitiktų „R“ reikalavimus.

Jei kintamieji daugiausia turi būti laikomi duomenų sistemose, visą duomenų sistemą galima nuskaityti tiesiogiai naudojant funkciją `read.table()`. Taip pat yra primityvesnė įvesties funkcija `scan()`, kurią galima iškviesti tiesiogiai.

7.1 Funkcija `read.table()`

Norėdami tiesiogiai nuskaityti visą duomenų sistemą, išorinis failas paprastai turės specialią formą:

1. pirmoje failo eilutėje turėtų būti duomenų sistemos kiekvieno kintamojo pavadinimas;
2. kiekviena papildoma failo eilutė pirmiausia turi eilutės žymeną ir kiekvieno kintamojo reikšmes.

Jei failo pirmoje eilutėje yra mažiau elementų nei antroje, laikoma, kad šis susitarimas galioja. Taigi, pirmosios kelios failo eilutės, kurias reikia skaityti kaip duomenų sistemas, gali atrodyti taip:

	Kaina	Aukštas	Plotas	Kambariai	Amžius	Šildymas
01	52.00	111.0	830	5	6.2	ne
02	54.75	128.0	710	5	7.5	ne
03	57.50	101.0	1000	5	4.2	ne
04	57.50	131.0	690	6	8.8	ne
05	59.75	93.0	900	5	1.9	taip
...						

Pagal numatytuosius nustatymus skaitiniai elementai (išskyrus eilutės žymenas) yra skaitomi kaip skaitiniai ir neskaitiniai kintamieji, pavyzdžiui, šiuo atveju kintamasis

„Šildymas“ yra faktorius. Prireikus tai galima pakeisti. Funkcija `read.table()` tada gali būti naudojamas tiesiogiai skaityti duomenų sistemą

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

Dažnai norėsite neįtraukti tiesiogiai eilučių žymenų ir naudoti numatytąsias žymenas. Tokiu atveju failas gali praleisti eilučių žymenų stulpelį, kaip nurodyta toliau.

Kaina	Aukštas	Plotas	Kambariai	Amžius	Šildymas
52.00	111.0	830	5	6.2	ne
54.75	128.0	710	5	7.5	ne
57.50	101.0	1000	5	4.2	ne
57.50	131.0	690	6	8.8	ne
59.75	93.0	900	5	1.9	taip

Tada duomenų sistema gali būti skaitoma kaip

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

kur `header=TRUE` parinktis nurodo, kad pirmoji eilutė yra antraščių eilutė, taigi, remiantis failo forma nėra nurodytos aiškios eilutės žymenos.

7.2 Funkcija `scan()`

Tarkime, kad duomenų vektoriai yra vienodo ilgio ir turi būti skaitomi lygiagrečiai. Be to, tarkime, kad yra trys vektoriai, pirmasis simbolių tipo, o kiti du – skaitinio tipo, ir failas yra `input.dat`. Pirmasis žingsnis yra naudoti `scan()`, norint skaityti tris vektorius kaip sąrašą

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

Antrasis argumentas yra netikro sąrašo struktūra, nustatanti trijų skaitomų vektorių tipą. Rezultatas, laikomas `inp`, yra sąrašas, kurio komponentai yra trys perskaityti vektoriai. Norėdami atskirti duomenų elementus į tris atskirus vektorius, naudokite priskyrimą

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

Patogiau, kad netikrame sąraše gali būti įvardytų komponentų, tokiu atveju pavadinimai gali būti naudojami prieigai prie skaitomų vektorių. Pavyzdžiui,

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

Jei norite kintamuosius pasiekti atskirai, juos galima priskirti kintamiesiems darbinėje srityje:

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

arba sąrašas gali būti pridedamas paieškos kelio 2 pozicijoje.

Jei antrasis argumentas yra viena reikšmė, o ne sąrašas, skaitomas vienas vektorius, kurio visi komponentai turi būti to paties tipo, kaip ir netikroji reikšmė.

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

Yra ir sudėtingesnių įvesties priemonių, kurios išsamiai aprašytos žinynuose.

7.3 Duomenų redagavimas

Kai kreipiamasi į duomenų sistemą ar matricą, `edit` sukuria atskirą į skaičiuoklę panašią aplinką redagavimui. Tai naudinga atliekant nedidelius pakeitimus, kai tik bus perskaitytas duomenų rinkinys. Komanda

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

leis jums redaguoti savo duomenų rinkinį `xold`, ir užbaigus pakeistas objektas yra priskiriamas į `xnew`. Jei norite pakeisti originalų duomenų rinkinį `xold`, paprasčiausias būdas yra naudoti `fix(xold)`, kuris yra lygiavertis `xold <- edit(xold)`.

Naudokite

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

įvesti naujus duomenis per skaičiuoklės sąsają.

8 Tikimybiniai skirstiniai

8.1 „R“ kaip statistinių lentelių rinkinys

Vienas iš patogių „R“ naudojimo būdų yra pateikti išsamų statistinių lentelių rinkinį. Funkcijos numatytos įvertinti suminio pasiskirstymo funkciją $P(X \leq x)$, tikimybės tankio funkciją ir kiekybinę funkciją (duotas q, mažiausias x toks, kad $P(X \leq x) > q$).

Skirstinys	**„R“ pavadinimas**	**Papildomi argumentai**
Beta	`beta`	`shape1, shape2, ncp`
Binominis	`binom`	`size, prob`
Koši	`cauchy`	`location, scale`
Chi-kvadrato	`chisq`	`df, ncp`
Eksponentinis	`exp`	`rate`
Fišerio (F)	`f`	`df1, df2, ncp`
Gama	`gamma`	`shape, scale`
Geometrinis	`geom`	`prob`
Hipergeometrinis	`hyper`	`m, n, k`
Log-normalusis	`lnorm`	`meanlog, sdlog`
Logistinis	`logis`	`location, scale`
Neigiamas binominis	`nbinom`	`size, prob`
Normalusis	`norm`	`mean, sd`
Puasono	`pois`	`lambda`
signed rank	`signrank`	`n`
Stjudento	`t`	`df, ncp`
Tolygusis	`unif`	`min, max`
Veibulo	`weibull`	`shape, scale`
Vilkoksono	`wilcox`	`m, n`

Programoje „R“ skirstinius aprašančių funkcijų pavadinimai turi 2 dalis:

1. funkcijos tipą (norimą skaičiavimą) aprašanti dalis (raidė `d`, `p`, `q` arba `r`);
2. skirstinio tipą aprašanti dalis (trumpasis „R“ skirstinio pavadinimas).

Funkcijos tipą aprašančių „R“ funkcijų pavadinimo dalys:

- `d...()` – tikimybės tankio arba tikimybių (masės) funkcija;

- `p…()` – pasiskirstymo funkcija;
- `q…()` – kvantilių funkcija (atvirkštinė pasiskirstymo funkcijai);
- `r…()` – atsitiktinių reikšmių generavimo funkcija.

`pxxx` ir `qxxx` visos funkcijos turi loginius argumentus `lower.tail` ir `log.p`, bei funkcija `dxxx` turi `log`.

Be to, yra funkcijos `ptukey` ir `qtukey` naudojamos bandymų, iš normaliojo skirstinio, režių pasiskirstymui, o funkcijos `dmultinom` ir `rmultinom` skirtos polinominiam skirstiniui. Kiti paskirstymai yra prieinami paketuose, ypač *SuppDists* (https://CRAN.R-project.org/package=SuppDists).

8.2 Duomenų rinkinio pasiskirstymo nagrinėjimas

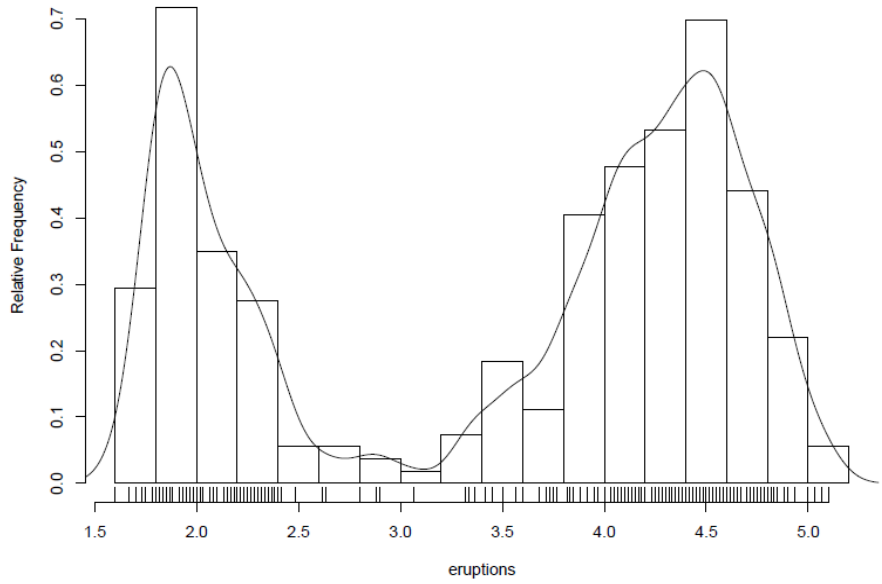
Atsižvelgiant į duomenų rinkinį (vienmatį), mes galime patikrinti jų pasiskirstymą įvairiais būdais. Pateikiamos dvi šiek tiek skirtingos santraukos pasinaudojus `summary` bei `fivenum` ir skaičių rodymas pasinaudojus `stem`.

```
> attach(faithful)
> summary(issiverzimai)
 Min. 1st Qu.  Median   Mean 3rd Qu.    Max.
1.600   2.163   4.000  3.488   4.454   5.100
> fivenum(issiverzimai)
[1] 1.6000 2.1585 4.0000 4.4585 5.1000
> stem(issiverzimai)
Dešimtainis taškas yra 1 skaitmuo į kairę nuo |
  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 8
44 | 022223355577800000000233333577888 8
46 | 00002333577000000235 78
48 | 00000022335800333
50 | 0370
```

„R“ turi funkciją `hist`, kuri skirta norint nubraižyti histogramas (1 pav.).

```
> hist(issiverzimai) # sudaro tankio grafiką
> hist(issiverzimai, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(issiverzimai, bw=0.1))
> rug(issiverzimai) # rodo faktinius duomenų taškus
```



1 pav. Išsiveržimų histograma

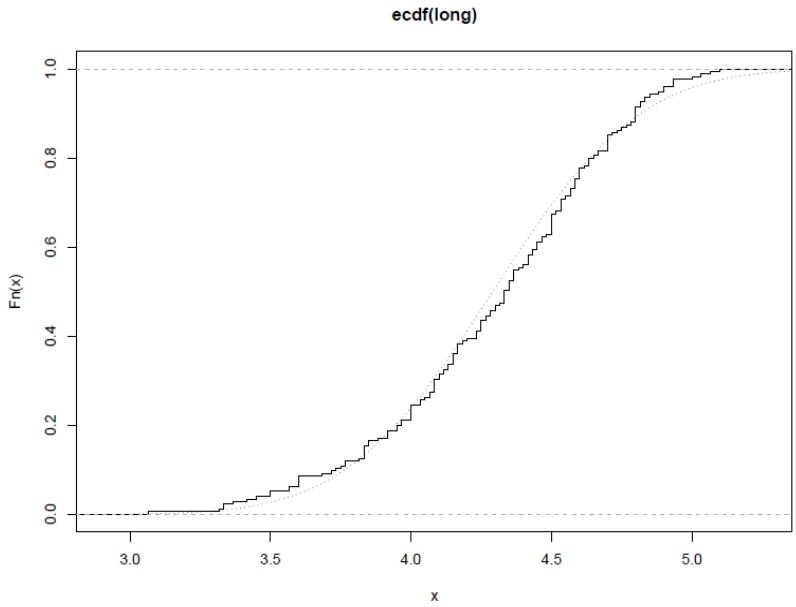
Elegantiškesnius tankio grafikus galima sudaryti pasinaudojus `density`, o šiame pavyzdyje mes pridėjome liniją, kuri sukurta pasinaudojus funkcija `density`. Pralaidumas `bw` buvo pasirinktas pagal bandymų ir klaidų metodą, nes numatytasis nustatymas suteikia per daug glodinimo.

Empirinę sukauptąją paskirstymo funkciją galime nubraižyti naudodami funkciją `ecdf`.

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

Akivaizdu, kad šis paskirstymas nėra standartinis. Tarkime, kad išsiveržimai yra ilgesni nei 3 minutės. Leiskite mums pritaikyti normalų paskirstymą ir pasinaudoti sukaupta pasiskirstymo funkcija (2 pav.).

```
> long <- issiverzimai [issiverzimai > 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)
```



2 pav. Sukauptosios paskirstymo funkcijos grafikas

Kvantilių palyginimo, arba kvantilių-kvantilių (Q-Q) diagrama gali padėti mums atidžiau tai išnagrinėti.

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

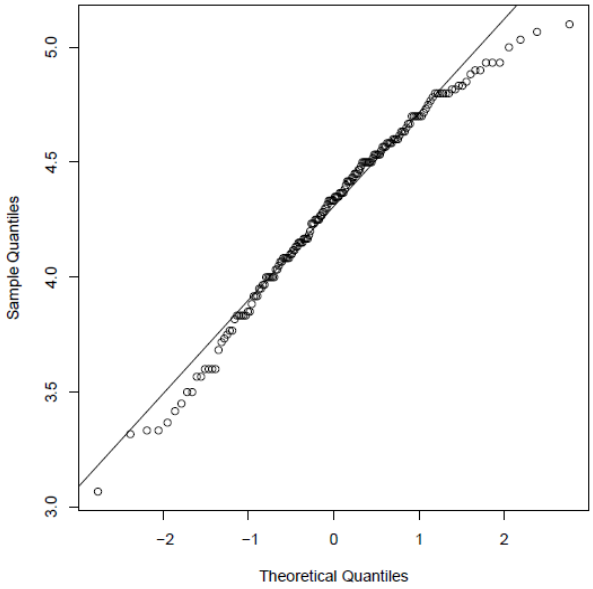
parodo pagrįstą, tačiau trumpesnę dešiniąją uodegą, nei būtų galima tikėtis iš įprasto pasiskirstymo. Palyginkime tai su keletu modeliuotų duomenų iš *t*-skirstinio

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

kurie paprastai (jei tai yra atsitiktinis pavyzdys) rodys ilgesnes uodegas, nei tikėtasi. Mes galime padaryti kvantilių palyginimo diagramą (3 pav.) panaudojus

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

```
qqline(x)
```



3 pav. Kvantilių palyginimo diagrama

„R“ pateikia Šapiro-Vilko testą

```
> shapiro.test(long)

        Shapiro-Wilk normality test

data: long

W = 0.9793, p-value = 0.01052
```

ir Kolmogorovo-Smirnovo testą

```
> ks.test(long, "pnorm", mean = mean(long), sd =
        sqrt(var(long)))
         One-sample Kolmogorov-Smirnov test
data: long
D = 0.0661, p-value = 0.4284
alternative hypothesis: two.sided
```

Atkreipkite dėmesį, kad pasiskirstymo teorija čia negalioja, nes mes įvertinome normalaus pasiskirstymo parametrus iš to paties pavyzdžio.

8.3 Vienos ir dviejų imčių testai

Iki šiol mes palyginome vienos imties normalųjį pasiskirstymą. Daug dažnesnė operacija yra palyginti dviejų imčių aspektus. Atminkite, kad visi klasikiniai testai, įskaitant žemiau aprašytus, yra pakete *stats*, kuris paprastai būna jau įkeltas.

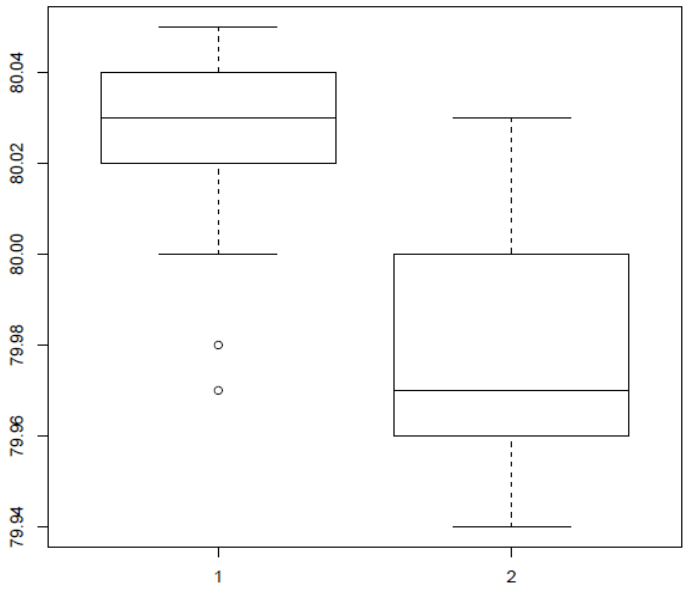
Apsvarstykite šiuos duomenų rinkinius apie ledo sintezės šilumą (cal/gm)

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

Stačiakampė diagrama (4 pav.) pateikia paprastą grafinį dviejų imčių palyginimą

```
A <- scan()
79.98 80.04 80.02 80.04 80.03 80.03 80.04 79.97
80.05 80.03 80.02 80.00 80.02
B <- scan()
80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97
boxplot(A, B)
```

o tai rodo, kad pirmoji grupė linkusi duoti aukštesnių rezultatų nei antroji.



4 pav. Dviejų imčių palyginimo diagrama

Išbandyti dviejų pavyzdžių vidurkių lygybę, galime naudoti neporinį *t*-testą

```
> t.test(A, B)
        Welch Two Sample t-test
data: A and B
```

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

o tai rodo reikšmingą skirtumą, darant prielaidą dėl normalumo.

Pagal nutylėjimą „R“ funkcija nereiškia, kad abiejų pavyzdžių dispersijos yra vienodos. Mes galime naudoti F testą, norėdami patikrinti lygybes dispersijose, su sąlyga, kad abi imtys yra iš normalių populiacijų.

```
> var.test(A, B)
        F test to compare two variances
data: A and B
F = 0.5837, num df = 12, denom df = 7, p-value = 0.3938
alternative hypothesis: true ratio of variances is not
equal to 1 95 percent confidence interval:
 0.1251097 2.1052687
sample estimates:
ratio of variances
        0.5837405
```

kuris nerodo jokio reikšmingo skirtumo, todėl galime naudoti klasikinį *t*-testą, kuriame daroma prielaida, kad dispersijos yra lygios.

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

Visi šie testai daro prielaidą, kad abi imtys yra normalios. Dviejų imčių Vilkoksono (arba Mano Vitney) testas tik daro prielaidą dėl bendro tolydžiojo pasiskirstymo pagal nulinę hipotezę.

```
> wilcox.test(A, B)
        Wilcoxon rank sum test with continuity correction
data: A and B
W = 89, p-value = 0.007497
alternative hypothesis: true location shift is not equal to 0
Warning message:
Cannot compute exact p-value with ties in: wilcox.test(A, B)
```

Atkreipkite dėmesį į įspėjimą: kiekvienoje imtyje yra keli ryšiai, o tai aiškiai rodo, kad šie duomenys yra iš diskretaus paskirstymo (tikriausiai dėl apvalinimo).

Yra keli būdai, kaip grafiškai palyginti dvi imtis. Mes jau matėme porą stačiakampių diagramų. Sekančios komandos

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

parodys dvi empirines sukauptąsias paskirstymo funkcijas ir `qqplot` pateiks dviejų imčių kvantilių-kvantilių (Q-Q) diagramą. Kolmogorovo-Smirnovo testas yra maksimalus vertikalus atstumas tarp dviejų `ecdf`, darant prielaidą dėl bendro tolydžiojo pasiskirstymo:

```
> ks.test(A, B)
        Two-sample Kolmogorov-Smirnov test
data: A and B
D = 0.5962, p-value = 0.05919
alternative hypothesis: two-sided
Warning message:
cannot compute correct p-values with ties in: ks.test(A, B)
```

9 Grupavimas, ciklai ir sąlyginis vykdymas

9.1 Grupavimo išraiškos

„R“ yra naudojama išraiškos kalba ta prasme, kad joje vienintelis komandos tipas yra funkcija arba išraiška, kuri grąžina rezultatą. Net priskyrimas yra išraiška, kurios rezultatas yra priskirta reikšmė, ir ji gali būti naudojama visur, kur galima naudoti bet kurią išraišką; ypač kur yra galimi keli priskyrimai.

Komandos gali būti sugrupuotos riestiniuose skliaustuose `{expr_1; ...; expr_m}`, tokiu atveju grupės reikšmė yra paskutinės išraiškos vertintoje grupėje rezultatas. Kadangi tokia grupė taip pat yra išraiška, ji, pavyzdžiui, gali būti įtraukta į skliaustelius ir naudojama kaip dar didesnės išraiškos dalis ir pan.

9.2 Valdymo sakiniai

9.2.1 Sąlyginis vykdymas: sąlyginis sakiniai

Kalba turi sąlyginę konstrukciją, kurios forma yra

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

kur `expr_1` turi įvertinti iki vienos loginės reikšmės ir tada išryškėja visos išraiškos rezultatas.

Trumpojo jungimo operatoriai `&&` ir `||` dažnai naudojami kaip teiginio sąlygos dalis. Kadangi `&` ir `|` pritaikomas vektoriams elementų atžvilgiu, tai `&&` ir `||` taikomas vieno ilgio vektoriams ir, jei reikia, įvertina tik antrąjį jų argumentą.

Vektorizuota konstrukcijos `if/else` versija yra funkcija `ifelse`. Tai turi formą `ifelse(condition, a, b)` ir grąžina to paties ilgio vektorių kaip `condition`, su elementais `a[i]` jei `condition[i]` yra teisinga, kitu atveju grąžina `b[i]` (kur `a` ir `b` yra perdirbami pagal poreikį).

9.2.2 Pakartotinis vykdymas

Taip pat yra naudojama ciklo konstrukcija, kurios forma yra

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

kur `name` yra ciklo kintamasis. `expr_1` yra vektorinė išraiška (dažnai tokia seka kaip `1:20`) ir `expr_2` dažnai yra sugrupuota išraiška su jos subišraškomis (sub-expressions). `expr_2` yra pakartotinai įvertinamas, kai pavadinimas svyruoja per vektoriaus rezultato `expr_1` reikšmes.

Tarkime `ind` yra vektorius, kurio klasė yra indikatoriai ir mes norime sudaryti atskirus brėžinius `y` lyginant su `x` klasėse. Viena iš galimybių yra naudoti `coplot()`, kuri sudarys brėžinių masyvą, atitinkantį kiekvieną faktoriaus lygį. Kitas būdas tai padaryti:

```
> 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]]))
  }
```

Atkreipkite dėmesį į funkciją `split()`, kuri sudaro vektorių sąrašą, ir kuris gaunamas padalijus didesnį vektorių pagal faktoriaus nurodytas klases. Tai yra naudinga funkcija, dažniausiai naudojama kartu su stačiakampėmis diagramomis. Žiūrėkite `help` priemonę, kad gautumėte daugiau informacijos.

Pastaba: `for()` ciklai yra naudojami „R“ programos kode daug rečiau nei kompiliuojamose kalbose.

Kitos kartojimo išraiškos yra sakinys

```
> repeat expr
```

ir sakinys

```
> while (condition) expr
```

Sakinys `break` statement gali būti naudojamas nutraukti bet kurį ciklą, galbūt neįprastai. Tai vienintelis būdas nutraukti `repeat` ciklus. Sakinys `next` gali būti naudojamas nutraukti vieną tam tikrą ciklą ir pereiti prie kito.

10 Savo funkcijų rašymas

„R“ programoje naudojama kalba vartotojui leidžia kurti objektus, kurių tipas yra `function`. Tai yra tikros „R“ funkcijos, kurios saugomos specialia vidine forma ir gali būti naudojamos tolesnėms išraiškoms ir pan. Proceso metu kalba įgauna didžiulę galią, patogumą ir eleganciją, o mokymasis rašyti naudingas funkcijas yra vienas iš pagrindinių būdų, kaip patogiai ir produktyviai naudoti „R“ paketą.

Pabrėžtina, kad dauguma funkcijų yra „R“ paketo dalis. Tokios funkcijos kaip `mean()`, `var()`, `postscript()` ir pan. yra užrašytos „R“ kalba ir todėl iš esmės nesiskiria nuo vartotojo užrašytų funkcijų. Funkcija apibrėžiama priskyrimu, kurios forma yra

```
> name <- function(arg_1, arg_2, ...) expression
```

čia `expression` yra „R“ išraiška (paprastai sugrupuota išraiška), kuri naudojasi argumentais `arg_i` kaip nori apskaičiuoti reikšmę. Išraiškos reiškmė yra reikšmė, kurią grąžina funkcija. Tada kvietimas funkcijai paprastai būna toks `name(expr_1, expr_2, ...)` ir gali atsirasti visur, kur funkcijos iškvietimas yra teisėtas.

10.1 Paprasti pavyzdžiai

Kaip pirmą pavyzdį apsvarstykite funkciją, kuri padėtų apskaičiuoti dviejų imčių *t*-statistiką, rodančią visus veiksmus. Žinoma, tai dirbtinis pavyzdys, nes yra ir kitų, paprastesnių būdų, kaip pasiekti tą patį tikslą.

Funkcija apibrėžiama taip:

```
> 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(s*(1/n1 + 1/n2))
    tst
```

```
}
```

Apibrėžę šią funkciją, galite atlikti du *t*-testus, naudodami tokį iškvietimą kaip

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

Kaip antrą pavyzdį, išnagrinėkime funkciją, tiesiogiai mėgdžiojančią „Matlab“ kairinio kampo komandą, kuri grąžina vektoriaus *y* ortogonalios projekcijos į (ant – onto) matricos `X` stulpelio erdvę koeficientus. Paprastai tai būtų daroma su funkcija `qr()`, tačiau kartais tai naudoti yra šiek tiek sudėtinga, o norint saugiai naudotis, verta atlikti paprastas funkcijas, tokias kaip šios.

Duotas [n × 1] vektorius *y* ir [n × p] dydžio matrica X, tada X *y* yra apibrėžiamas kaip $(X^T X)^{-} X^T y$, kur $(X^T X)^{-}$ yra apibendrinta atvirkštinė X′X .

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

Sukūrus šį objektą, jis gali būti naudojamas tokiuose teiginiuose kaip

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

Klasikinė „R“ funkcija `lsfit()` daro šį darbą gana gerai ir dar daugiau. Naudoja funkcijas `qr()` ir `qr.coef()` aukščiau pateiktu šiek tiek priešingu būdu, kad atliktumėte šią skaičiavimo dalį. Taigi tikriausiai yra naudinga, jei ši dalis yra atskirta paprasto naudojimo funkcijoje, jei ji bus dažnai naudojama. Jei taip, galbūt norėsime, kad jis taptų dvejetainiu matricos operatoriumi, kad jį būtų dar patogiau naudoti.

10.2 Naujų dvejetainių operatorių apibrėžimas

Jei būtume davę funkcijai `bslash()` kitą pavadinimą, būtent vieną iš formos

```
%anything%
```

ji galėjo būti naudojamas kaip dvejetainis operatorius išraiškose, o ne funkcijos formoje.

Tarkime, kad mes pasirenkame `!` vidiniam ženklui. Tada funkcijos apibrėžimas prasidėtų taip:

```
> "%!%" <- function(X, y) { ... }
```

Tada funkcija galėtų būti naudojama kaip `X%!%y`. Pats kairinio kampo simbolis nėra patogus pasirinkimas, nes šiame kontekste jis sukelia specialių problemų.

Matricos daugybos operatorius `%*%` ir and išorinis produkto matricos operatorius `%o%` yra kiti tokiu būdu apibrėžtų dvejetainių operatorių pavyzdžiai.

10.3 Įvardyti argumentai ir nutylėjimai

Jei argumentai iškviestoms funkcijoms pateikiami forma „`name = object`“, jie gali būti duodami bet kokia tvarka. Be to, argumentų seka gali prasidėti neįvardytoje pozicinėje formoje ir nurodyti įvardytus argumentus po pozicinių argumentų.

Taigi, jei yra funkcija `fun1`, kuri yra apibrėžta

```
> fun1 <- function(data, data.frame, graph,
    limit) { [function body omitted]
  }
```

tada funkciją galima panaudoti keliais būdais, pavyzdžiui

```
> 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)
```

ir visi šie būdai yra lygiaverčiai.

Daugeliu atvejų argumentams gali būti suteikiamos dažniausiai tinkamos numatytosios reikšmės; tokiu atveju jos gali būti praleistos iškvietime, kai numatytosios reikšmės yra tinkamos. Pavyzdžiui, jei funkcija `fun1` būtų apibrėžta kaip

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

ji galėtų būti iškviesta kaip

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

kuri dabar prilygsta trims aukščiau paminėtiems atvejams, arba kaip

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

kuri keičia vieną iš numatytųjų.

Svarbu pažymėti, kad nutylėjimai gali būti pasirinktos išraiškos, netgi susijusios su kitais tos pačios funkcijos argumentais; jie neapsiriboja konstantomis, kaip mūsų paprastame pavyzdyje.

10.4 '...' argumentas

Kitas dažnas reikalavimas – leisti vienai funkcijai perduoti argumentų parametrus kitai. Pavyzdžiui, daugelis grafikos funkcijų naudoja funkciją `par()` ir tokią funkciją kaip `plot()`, kuri leidžia vartotojui perduoti grafinius parametrus funkcijai `par()`, kad galėtų kontroliuoti išvestį. Tai galima padaryti įtraukiant papildomą funkcijos argumentą `'...'`, kuris vėliau gali būti perduotas. Pateikiamas pavyzdys.

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

Retesniais atvejais funkcijai reikės kreiptis į `'...'` komponentus. Išraiška `list(...)` įvertina visus tokius argumentus ir grąžina juos į įvardytą sąrašą, o `..1, ..2,` ir t.t., įvertina juos vienu kartu, su `'..n'` grąžina n-tąjį nesuderintą argumentą.

10.5 Priskyrimai funkcijose

Atminkite, kad visi įprasti priskyrimai, atliekami funkcijoje yra vietiniai ir laikini, jie prarandami išėjus iš funkcijos. Taigi priskyrimas `X <- qr(X)` nedaro įtakos argumento reikšmei iškvietimo programoje.

Norėdami visiškai suprasti „R“ priskyrimų apimtį reglamentuojančias taisykles, skaitytojas turi būti susipažinęs su vertinimo sistemos sąvoka. Tai šiek tiek pažengusi, nors ir sunkiai suprantama tema, kuri čia nėra nagrinėjama.

Jei funkcijai numatyti visuotiniai ir nuolatiniai priskyrimai, tada gali būti naudojamas operatorius `<<-` arba funkcija `assign()`.

10.6 Pažangesni pavyzdžiai

10.6.1 Visų vardų praleidimas spausdintame masyve

Spausdinimo tikslais, naudojant dideles matricas ar masyvus, dažnai naudinga juos atspausdinti bloko forma be masyvo pavadinimų ar skaičių. Pašalinus `dimnames` atributą, šio efekto nebus pasiekta, greičiau masyvui turi būti suteiktas `dimnames` atributas, kurį sudaro tuščios eilutės.

Pavyzdžiui, norint atspausdinti matricą `X`

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

Tai daug patogiau padaryti naudojant funkciją `no.dimnames()`, kaip parodyta žemiau, norint pasiekti tą patį rezultatą. Tai taip pat iliustruoja, kaip kai kurios veiksmingos ir naudingos vartotojo funkcijos gali būti gana trumpos.

```
no.dimnames <- function(a) {
    ## Kompaktiškam spausdinimui iš masyvo pašalinkite visus matmenų pavadinimus.
    d <- list()
    l <- 0
    for(i in dim(a)) {
      d[[l <- l + 1]] <- rep("", i)
    }
    dimnames(a) <-
    d a
  }
```

Apibrėžus šią funkciją, masyvas gali būti atspausdintas artimu formatu, naudojant

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

Tai ypač naudinga dideliems sveikųjų skaičių masyvams, kur modeliai yra tikrai įdomesni, nei tik reikšmės.

10.6.2 Rekursyvi skaitinė integracija

Funkcijos gali būti rekursyvios ir pačios gali apibrėžti funkcijas savyje. Tačiau atkreipkite dėmesį, kad tokios funkcijos (ar iš tikrųjų kintamieji) nepaveldimos iškvietimo funkcijų aukštesnėse vertinimo sistemoje, kaip būtų, jei jos būtų paieškos kelyje.

Žemiau pateiktame pavyzdyje parodytas paprastas vienos dimensijos skaitinės integracijos atlikimo būdas. Neatimama dalis vertinama intervalo galiniuose taškuose ir viduryje. Jei vienos plokštės (one-panel) trapecijos taisyklės atsakymas yra pakankamai artimas dviem plokštėm (two panel), tada pastaroji grąžinama kaip reikšmė. Priešingu atveju tas pats procesas yra rekursyviai taikomas kiekvienoje panel. Rezultatas yra adaptyvus integracijos procesas, kuris sutelkia funkcijų vertinimą regionuose, kur neatimama dalis yra labiausiai nutolusi nuo tiesinės. Vis dėlto yra didelė pridėtinė vertė, ir funkcija konkuruoja tik su kitais algoritmais, kai neatimama dalis yra sklandi ir labai sunkiai įvertinama.

Šis pavyzdys taip pat pateikiamas kaip maža „R“ programavimo dėlionė.

```
area <- function(f, a, b, eps = 1.0e-06, lim = 10) {
  fun1 <- function(f, a, b, fa, fb, a0, eps, lim, fun) {
    ## funkcija ‘fun1’ yra matoma tik viduje ‘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 Taikymo sritis

Šiame skyriuje diskusija yra šiek tiek labiau techninė nei kitose šio dokumento dalyse. Simboliai, kurie atsiranda funkcijos formose, gali būti suskirstyti į tris klases: formalieji parametrai, vietiniai ir laisvieji kintamieji. Formalūs funkcijos parametrai yra tie, kurie pateikiami funkcijos argumentų sąraše. Jų reikšmės nustatomos atsižvelgiant į faktinių funkcijos argumentų susiejimą su formaliaisiais parametrais. Vietiniai kintamieji yra tie, kurių reikšmės nustatomos įvertinant išraiškas funkcijų formose. Kintamieji, kurie nėra formalūs parametrai ar vietiniai kintamieji, vadinami laisvaisiais kintamaisiais. Laisvieji kintamieji tampa vietiniais kintamaisiais, jei jiems priskiriami. Apsvarstykite šį funkcijos apibrėžimą.

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

Šioje funkcijoje `x` yra formalusis parametras, `y` yra vietinis kintamasis, o `z` yra laisvasis kintamasis. „R“ programoje laisvųjų kintamųjų susiejimai išsprendžiami pirmiausia pažiūrėjus į aplinką, kurioje buvo sukurta funkcija. Tai vadinama leksine sritimi. Pirmiausia apibrėžiame funkciją, vadinamą „`cube`“.

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

Funkcijos `sq` kintamasis `n` nėra tos funkcijos argumentas. Todėl tai yra laisvas kintamasis, ir norint nustatyti reikšmę, kuri turi būti susieta, turi būti taikomos apimties nustatymo taisyklės. Pagal leksinę sritį tai yra funkcijos parametras, nes tai yra aktyvus kintamojo `n` ryšys tuo metu, kai buvo apibrėžta funkcija `sq`.

Leksinė sritis taip pat gali būti naudojama funkcijoms pakeisti. Šiame pavyzdyje parodysime, kaip „R“ programa gali būti naudojama imituoti banko sąskaitą. Veikiančioje

banko sąskaitoje turi būti likutis arba visa suma, išėmimų funkcija, indėlių įnešimo funkcija ir esamo likučio nustatymo funkcija. Mes to pasiekiame sukurdami tris funkcijas, kurios yra `account` ir grąžindami sąrašą su jomis. Kai yra iškviečiama `account`, ji užima skaitinį argumentą `total` ir pateikia sąrašą, kuriame yra trys funkcijos. Kadangi šios funkcijos yra apibrėžtos aplinkoje, kurioje yra `total` argumentas, jos turės prieigą prie jos reikšmių.

Specialaus priskyrimo operatorius `<<-` yra naudojamas pakeisti su reikšmę susietą su `total`. Šis operatorius uždaroje aplinkoje žvelgia į aplinką, kurioje yra simbolis `total`, o radęs tokią aplinką, toje aplinkoje esanti reikšmė pakeičia dešinės pusės reikšmę. Jei pasiekiama globalioji arba aukščiausio lygio aplinka, nerandant simbolio `total` tada tas kintamasis sukuriamas ir priskiriamas ten. Dauguma vartotojų naudoja operatorių `<<-` ir sukuria globalų kintamąjį bei priskiria jam dešinės pusės reikšmę. Tik tada kai operatorius `<<-` buvo naudojamas funkcijoje, kuri buvo grąžinta kaip kitos funkcijos reikšmė, tai įvyks ypatingas čia aprašytas elgesys.

```
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 Aplinkos pritaikymas

Vartotojai gali pritaikyti savo aplinką keliais skirtingais būdais. Yra svetainės inicializacijos failas ir kiekvienas katalogas gali turėti savo specialųjį inicializacijos failą. Galiausiai, gali būti naudojamos specialiosios funkcijos `.First` ir `.Last`.

Vietos inicijavimo failo vieta imama iš aplinkos kintamojo `R_PROFILE` reikšmės. Jei tas kintamasis nenustatytas, failas `Rprofile.site` yra naudojamas „R“ namų pakatalogyje `etc`. Šiame faile turėtų būti komandos, kurias norite vykdyti kiekvieną kartą, kai jūsų sistemoje paleidžiamas „R“. Antrasis asmeninis profilio failas pavadinimu `.Rprofile` gali būti dedamas į bet kurį katalogą. Jei tame aplanke bus iškviestas „R“, tada šis failas bus gaunamas. Šis failas suteikia individualiems vartotojams galimybę valdyti savo darbo sritį ir leidžia skirtingas paleidimo procedūras skirtinguose darbo kataloguose. Jei nerandamas failas `.Rprofile` paleidimo kataloge, tada „R“ ieško `.Rprofile` failo vartotojo namų kataloge ir naudoja jį (jei toks yra). Jei aplinkos kintamasis `R_PROFILE_USER` yra nustatytas, vietoj `.Rprofile` failų naudojamas failas, į kurį jis nurodo.

Bet kuri funkcija pavadinta `.First()` bet kuriame iš dviejų profilio failų arba `.RData` paveikslėlyje turi ypatingą būseną. Jis automatiškai atliekamas „R“ sesijos pradžioje ir gali būti naudojamas aplinkai suformuoti. Pavyzdžiui, pateiktame pavyzdyje pateiktas apibrėžimas keičia raginimą į `$` ir nustato įvairius kitus naudingus dalykus, kuriuos vėliau galima laikyti savaime suprantamu dalyku likusioje sesijos dalyje.

Taigi failų vykdymo seka yra: `Rprofile.site`, `.RData` ir tada `.First()`. Vėlesnių failų apibrėžimas užmaskuos ankstesnių failų apibrėžimus.

```
> .First <- function() {
  options(prompt="$ ", continue="+\t")      # $ yra raginimas
  options(digits=5, length=999) # pasirinktiniai numeriai ir spausdinimas
```

```
  x11()                                  # grafikams
  par(pch = "+")                         # braižymo simbolis
  source(file.path(Sys.getenv("HOME"), "R", "mystuff.R"))
                                         # mano asmeninės funkcijos
  library(MASS)                          # pridėti paketą
}
```

Panašiai, jei apibrėžta funkcija `.Last()`, tai ji (paprastai) vykdoma pačioje sesijos pabaigoje. Pavyzdys duotas žemiau:

```
> .Last <- function() {
  graphics.off()                         # nedidelė saugos priemonė.
  cat(paste(date(),"\nAdios\n"))         # Ar jau laikas pietums?
}
```

10.9 Klasės, bendrinės funkcijos ir objekto orientacija

Objekto klasė nustato, kaip jis bus traktuojamas vadinamųjų bendrinių funkcijų. Kitaip tariant, bendrinė funkcija atlieka užduotį ar veiksmą pagal savo argumentus, būdingus pačiai argumento klasei. Jei argumentui trūksta kokio klasės atributo arba jei jo klasei netaikoma konkreti nagrinėjama bendrinė funkcija, visada pateikiamas numatytasis veiksmas.

Klasių mechanizmas suteikia vartotojui galimybę projektuoti ir rašyti bendrines funkcijas specialiems tikslams. Tarp kitų bendrinių funkcijų yra: `plot()` skirtas objektams pavaizduoti grafiškai; `summary()` įvairių rūšių analizėms apibendrinti; `anova()` statistinių modelių palyginimui.

Bendrinių funkcijų, kurios gali apibūdinti klasę konkrečiu būdu, skaičius gali būti gana didelis. Pavyzdžiui, funkcijos, kurios gali tilpti į kai kuriuos objektus, priklausančius `"data.frame"` klasei, yra:

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

Šiuo metu išsamų sąrašą galima gauti naudojantis funkcija `methods()`:

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

Klasių, kurias gali atlikti bendroji funkcija, skaičius taip pat gali būti gana didelis. Pavyzdžiui, funkcija `plot()` turi numatytąjį metodą ir klasių `"data.frame"`, `"density"`, `"factor"` objektų variantus. Pilną sąrašą vėl galite gauti naudodamiesi funkcija `methods()`:

```
> methods(plot)
```

Daugelio bendrinių funkcijų funkcijos forma yra gana trumpa, pavyzdžiui

```
> coef
function (object, ...)
UseMethod("coef")
```

`UseMethod` buvimas rodo, kad tai yra bendroji funkcija. Norėdami sužinoti, kokius metodus galima naudoti, galime naudoti `methods()`

```
 > methods(coef)
 [1] coef.aov*   coef.Arima*      coef.default*     coef.listof*
 [5] coef.nls*   coef.summary.nls*
```

Nematomos funkcijos pažymėtos žvaigždute. Šiame pavyzdyje yra šeši metodai, kurių nė vieno negalima pamatyti įvedant jo pavadinimą. Tai galime perskaityti bet kuriuo iš šių punktų

```
> getAnywhere("coef.aov")
A single object matching 'coef.aov' was found
It was found in the following places
  registered S3 method for coef from namespace stats
  namespace:stats
with value
function (object, ...)
{
    z <- object$coef
    z[!is.na(z)]
}
> getS3method("coef", "aov")
> function (object, ...)
{
    z <- object$coef
```

```
    z[!is.na(z)]
}
```

Funkcija pavadinta `gen.cl` bus iškviesta bendrine funkcija *gen* klasei `cl`, todėl nepavadinkite funkcijų šiuo stiliumi, nebent jos būtų skirtos metodams.

11 Statistiniai modeliai

Šiame skyriuje daroma prielaida, kad skaitytojas yra šiek tiek susipažinęs su statistine metodika, ypač su regresine analize ir dispersine analize. Reikalavimai pritaikyti statistinius modelius yra pakankamai tiksliai apibrėžti, kad būtų galima sudaryti bendrąsias priemones, taikytinas daugelyje problemų.

„R“ siūlo komplektą priemonių, kurios labai palengvina statistinių modelių pritaikymą. Kaip minime įvade, pagrindinė išvestis yra minimali, todėl reikia paprašyti išsamesnės informacijos iškvietus funkcijas.

11.1 Statistinių modelių apibrėžimas

Statistinio modelio šablonas yra tiesinės regresijos modelis su nepriklausomomis homoscedastic klaidomis

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

Matricine prasme tai būtų parašyta

$$y = X\beta + e$$

kur y yra atsako vektorius, X yra modelio matrica arba projektavimo matrica ir turi stulpelius $x_0, x_1, \ldots, x_p$, nusakančius kintamuosius. Labai dažnai x_0 bus stulpelis, apibrėžiantis laisvąjį narį.

Pavyzdžiai

Prieš pateikdami formalią specifikaciją, keletas pavyzdžių gali būti naudingi nustatant vaizdą.

Tarkime, kad `y, x, x0, x1, x2, ...` yra skaitiniai kintamieji, `X` yra matrica ir `A, B, C, ...` yra faktoriai.

Toliau pateikiama lentelė (1 lentelė), kurioje nurodytos formulės ir jas aprašantys statistiniai modeliai.

1 lentelė. Statistinių modelių formulės

`y ~ x` `y ~ 1 + x`	Abi formulės nusako tą patį paprastą tiesinės regresijos modelį. Pirmoji formulė turi numanomą laisvąjį narį, antroje formulėje jis yra nurodytas.

`y ~ 0 + x`
`y ~ -1 + x`
`y ~ x - 1`
	Paprasta tiesinė regresija (be laisvojo nario).

`log(y) ~ x1 + x2`	Transformuoto kintamojo dauginė regresija (su numanomu laisvuoju nariu).
`y ~ poly(x,2)` `y ~ 1 + x + I(x^2)`	Polinominė regresija. Pirmojoje formulės formoje naudojami ortogonalieji polinomai. Antroje naujodamas laipsnis.
`y ~ X + poly(x,2)`	Dauginė regresija *y* su modelio matrica, kurią sudaro X matrica ir taip pat polinominiai nariai.
`y ~ A`	*y* dispersijos modelio klasifikacijos analizė, su klasėmis, kurias nustatė A.
`y ~ A + x`	*y* kovariacijos modelio klasifikacijos analizė, su klasėmis, kurias nustatė A ir su kovariatoriumi *x*.

`y ~ A*B`
`y ~ A + B + A:B`
`y ~ B %in% A`
`y ~ A/B`
	Dviejų faktorių y neadityvus modelis į A ir B. Pirmieji du nurodo tą pačią kryžminę klasifikaciją, o antrieji du nurodo tą pačią įdėtąją klasifikaciją. Abstrakčiai kalbant, visos keturios nurodo to paties modelio poerdvį.

`y ~ (A + B + C)^2` `y ~ A*B*C - A:B:C`	Trijų faktorių eksperimentas, tačiau su modeliu, kuriame yra pagrindiniai efektai ir tik dviejų faktorių sąveika. Abi formulės nurodo tą patį modelį.

`y ~ A * x`
`y ~ A/x`
`y ~ A/(1 + x) - 1`
	Paprasti *y* tiesiniai regresijos modeliai, esantys *x* per A lygius, su skirtingais kodavimais. Paskutinėje formoje pateikiami aiškūs skirtingų atkarpos ilgių ir nuolydžių įvertinimai, kiek jų yra A lygių.

`y ~ A*B + Error(C)`	Eksperimentas su dviem gydymo veiksniais – A ir B bei su paklaidos sluoksniais, kuriuos nustato C faktorius.

Operatorius ~ naudojamas apibrėžti modelio formulę „R“ pakete. Įprasto tiesinio modelio forma yra

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

čia

`response` yra vektorius arba matrica (arba išraiška vertinama pagal vektorių ar matricą) apibrėžiantys atsako kintamąjį (-uosius). `op_i` yra operatorius, arba + arba −, reiškiantis nario įtraukimą ar neįtraukimą į modelį (pirmasis neprivalomas).

`term_i` yra:

- vektoriaus arba matricos išraiška, arba 1,
- faktorius
- formulės išraiška, susidedanti iš faktorių, vektorių ar matricų, sujungtų formulės operatorių.

Visais atvejais kiekvienas narys apibūdina stulpelių rinkinį, kuris turi būti pridėtas arba pašalintas iš modelio matricos. 1 reiškia laisvąjį stulpelį ir pagal nutylėjimą yra įtrauktas į modelio matricą, nebent jis aiškiai pašalintas.

Formulės operatorių žymėjimas apibendrintai pateikiamas 2 lentelėje.

2 lentelė. Operatorių žymėjimas

`Y ~ M`	Y yra sumodeliuotas kaip M.
`M_1 + M_2`	Įtraukia M_1 ir M_2.
`M_1 - M_2`	Įtraukia M_1 paliekant M_2 narius.
`M_1 : M_2`	M_1 ir M_2 tenzorinis produktas. Jei abu nariai yra faktoriai, tai poklasiai yra faktoriai.

`M_1 %in% M_2`	Panašus į `M_1:M_2`, bet su kitokiu kodavimu.
`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`	Visi nariai, esantys M kartu su „sąveikomims“ iki eilės *n*.
`I(M)`	Izoliuoja M. M viduje esantys visi operatoriai turi normalią aritmetinę reikšmę, ir šis narys pasirodo modelio matricoje. Atminkite, kad skliausteliuose, kuriuose paprastai pateikiami funkcijos argumentai, visi operatoriai turi įprastą aritmetinę reikšmę, Funkcija `I()` yra tapatybės funkcija, naudojama tam, kad modelių formulėse būtų galima apibrėžti narius, naudojant aritmetinius operatorius.

Ypač atkreipkite dėmesį, kad modelio formulės nurodo modelio matricos stulpelius, parametrų specifikacija yra netiesioginė. Kitose situacijose to nėra, pavyzdžiui, nurodant netiesinius modelius.

11.1.1 Kontrastai

Mums reikia bent kažkokios idėjos, kaip modelio formulėse nurodomi modelio matricos stulpeliai. Tai lengva, jei turime tolydžiuosius kintamuosius, nes kiekvienas pateikia vieną modelio matricos stulpelį (o laisvasis pateiks stulpelį vienetų, jei jie bus įtraukti į modelį).

O kaip *k*-lygio fatorius A? Atsakymas skiriasi dėl rikiuotų ir nerikiuotų faktorių. Nerikiuotiems faktoriams $k - 1$ stulpeliai sudaromi antriems rodikliams, . . . , *k*-tasis faktoriaus lygis. Taigi, numanomas parametrizavimas turi kontrastuoti kiekvieno lygio atsaką su pirmuoju. Rikiuotiems faktoriams $k - 1$ stulpeliai yra otogomalieji polinomai pagal 1, ..., *k*, ir praleidus konstantą.

Jei modelyje, kuris turi faktorinių narių yra praleidžiamas laisvasis narys, pirmasis toks narys yra užkoduojamas kaip *k* stulpeliai, kuriuose pateikiami visų lygių rodikliai. Toliau, visas

`contrasts` elgesys gali būti pakeistas `options` nustatymuose. Numatytasis „R“ nustatymas yra

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

Mes dar nebaigėme, nes kontrasto schemą, kurią reikia naudoti, kiekvienam modelio nariui galima nustatyti naudojant funkcijas `contrasts` ir `C`. Mes dar nesvarstėme sąveikos narių: jie sukuria stulpelių produktus, pateiktus jų komponentų nariams.

Nors išsami informacija yra sudėtinga, „R“ statistiniame pakete modelio formulės paprastai sugeneruoja modelius, kurių tikisi statistas ekspertas, su sąlyga, kad bus išlaikytas skirtumas. Pavyzdžiui, modelio, turinčio sąveiką, pritaikymas, bet ne atitinkami pagrindiniai efektai, iš esmės duos stebinančių rezultatų ir yra skirtas tik ekspertams.

11.2 Tiesiniai modeliai

Pagrindinė įprastų dauginių modelių funkcija yra `lm()`, ir supaprastinta iškvietimo versija yra tokia:

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

Pavyzdžiui,

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

būtų tinkamas dauginės regresijos modeliui *y* pagal *x1* ir *x2* (su numanomu laisvuoju nariu).

Svarbus (bet techniškai neprivalomas) parametras `data = production` nurodo, kad visi kintamieji, reikalingi modeliui sukurti, turėtų būti kilę pirmiausia iš duomenų sistemos `production`. Taip yra nepriklausomai nuo to, ar duomenų sistema `production` buvo pridėta paieškos kelyje ar ne.

11.3 Bendrinės funkcijos modeliui išgauti

Funkcijos `lm()` reikšmė yra pritaikytas modelio objektas, techniškai tai klasės `"lm"` rezultatų sąrašas. Tada informaciją apie pritaikytą modelį galima parodyti, išskleisti, nubraižyti ir pan., naudojant bendrines funkcijas, kurios orientuojasi į klasės `"lm"` objektus. Jos apima:

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

Trumpas dažniausiai naudojamų bendrinių funkcijų aprašymas pateiktas 3 lentelėje.

3 lentelė. Bendrinės funkcijos

`anova(object_1, object_2)`	Palygina submodelį (submodel) su išoriniu modeliu ir atlieka dispersinės lentelės analizę.
`coef(object)`	Ištraukia regresijos koeficientą (matricą). Ilga forma: `coefficients(object)`.
`deviance(object)`	Kvadratų sumos liekana, jei reikia, įvertinta.
`formula(object)`	Ištraukia modelio formulę.
`plot(object)`	Sukuria keturis brėžinius, kuriuose nurodomos liekanos, suderintos reikšmės ir tam tikra diagnostika.
`predict(object, newdata=data.frame)`	Pateiktoje duomenų sistemoje turi būti kintamųjų, nurodytų tomis pačiomis žymėmis kaip ir originale. Reikšmė yra numatytų reikšmių vektorius arba matrica, atitinkanti nustatytas kintamojo reikšmes, esančias data.frame.
`print(object)`	Atspausdina glaustą objekto versiją. Dažniausiai naudojamas netiesiogiai.
`residuals(object)`	Ištraukia liekanų matricą. Vertinamą kaip tinkamą. Trumpa forma: `resid(object)`.
`step(object)`	Pasirenka tinkamą modelį pridėdami arba atsisakydami narių ir išsaugodami hierarchijas.
`summary(object)`	Atspausdina išsamią regresinės analizės rezultatų santrauką.

`vcov(object)`	Grąžina pritaikyto modelio objekto pagrindinių parametrų dispersijos ir kovariacijos matricą.

11.4 Dispersijos analizė ir modelio palyginimas

Modelio pritaikymo funkcija `aov(formula, data=data.frame)` veikia paprasčiausiu lygiu, labai panašiai kaip funkcija `lm()`, ir pritaiko daugumą bendrųjų funkcijų, išvardytų 11.3 skyriaus lentelėje.

Pažymėtina, kad papildomai `aov()` leidžia analizuoti modelius su keliais klaidų sluoksniais, tokiais kaip eksperimentai su padalijimais, arba subalansuoti neišsamūs blokų projektai, atkuriant tarpblokų informaciją. Modelio formulė

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

nurodo kelių pakopų eksperimentą su klaidų sluoksniais, apibrėžtais `strata.formula`.

Paprasčiausiu atveju strata.formula yra tiesiog faktorius, kai jis apibūdina dviejų sluoksnių eksperimentą, būtent tarp faktorių lygių ir pačiuose lygiuose.

Atsižvelgiant į visus lemiančius kintamųjų faktorius, pavyzdinė formulė:

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

paprastai būtų naudojamas apibūdinti eksperimentą su vidurkio modeliu `v + n*p*k` ir tris klaidų sluoksnius.

11.4.1 ANOVA lentelės

Taip pat atkreipkite dėmesį, kad dispersijos lentelės (arba lentelių) analizė yra pritaikytų modelių seka. Pateiktos kvadratų sumos yra likusių kvadratų sumų sumažėjimas, atsirandantis dėl to nario įtraukimo į modelį toje sekos vietoje. Taigi tik ortogonaliems eksperimentams įtraukimo tvarka nebus reikšminga. Daugelio pakopų eksperimentams pirmiausia reikia pateikti atsakymą į klaidų sluoksnius, ir vėl paeiliui pritaikyti vidurkio modelį kiekvienai projekcijai.

Lankstesnė alternatyva numatytai pilnai ANOVA lentelei yra palyginti du ar daugiau modelių tiesiogiai naudojant funkciją `anova()`.

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

Tada rezultatas yra ANOVA lentelė, rodanti skirtumus tarp pritaikytų modelių, kai jie taikomi iš eilės. Pritaikyti modeliai paprastai palyginami būtų hierarchinė seka. Tai nesuteikia skirtingos informacijos apie numatytuosius nustatymus, o palengvina supratimą ir valdymą.

11.5 Tinkamų modelių atnaujinimas

Funkcija `update()` iš esmės yra patogi funkcija, leidžianti pritaikyti modelį, kuris skiriasi nuo anksčiau pritaikymo modelio tik keliais papildomais ar pašalintais nariais. Forma yra tokia:

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

Formulėje `new.formula` specialusis pavadinimas, susidedantis iš laikotarpio „.“, tik, gali būti naudojamas stovėti už „atitinkamą seno modelio formulės dalį“. Pavyzdžiui,

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

būtų tinkama penkioms kintamosioms regresijoms su kintamaisiais (tikėtina) iš duomenų sistemos gamybos, tinka papildomam modeliui, apimančiam šeštąjį regresijos kintamąjį, ir tinka modeliui, kurio atsakymui buvo pritaikyta kvadratinės šaknies transformacija.

Ypač atkreipkite dėmesį, jei argumentas `data=` yra nurodytas pradiniame kvietime į modelio pritaikymo funkciją, ši informacija perduodama per pritaikytą modelio objektą į `update()`.

Pavadinimas „`.`“ taip pat gali būti naudojamas kituose kontekstuose, tik turint šiek tiek kitokią prasmę. Pavyzdžiui,

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

atitiktų modelį su atsaku *y* ir regresoriaus kintamaisiais ir visais kitais kintamaisiais duomenų sistemoje production.

Kitos funkcijos, skirtos ištirti pamatines modelių sekas yra `add1()`, `drop1()` ir `step()`. Jų pavadinimai tiksliai parodo jų paskirtį, tačiau išsamią informaciją rasite internetinėje žinyne.

11.6 Apibendrinti tiesiniai modeliai

Apibendrintas tiesinis modeliavimas tiesinių modelių kūrimas. Apibendrintą tiesinį modelį galima apibūdinti tokia prielaidų seka:

- Yra atsakas *y* ir stimulo kintamieji x_1, x_2, ... , kurių reikšmės daro įtaką atsako pasiskirstymui.
- Stimulo kintamieji daro įtaką *y* pasiskirstymui tik per vieną tiesinę funkciją. Ši tiesinė funkcija vadinama tiesiniu prediktoriumi ir paprastai rašoma

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

x_i neturi įtakos *y* paskirstymui tada ir tada, jei $\beta_i = 0$.

- *y* pasiskirstymas yra formos

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

kur φ yra mastelio parametras (galbūt žinomas) ir yra pastovus visų stebėjimų metu, A žymi ankstesnį svorį, kuris, kaip manoma, yra žinomas, bet gali kisti atsižvelgiant į stebėjimus, ir yra *y* vidurkis. Taigi daroma prielaida, kad *y* pasiskirstymą lemia ir jo vidurkis, ir galbūt mastelio parametras.

- Vidurkis μ yra sklandi nekintama tiesino prediktoriaus funkcija

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

ir ši atvirkštinė funkcija ℓ () yra vadinama susiejimo funkcija.

Šios prielaidos yra pakankamai laisvos, kad apimtų plačią statistikos praktikoje naudingų modelių klasę, tačiau yra pakankamai griežtos, kad bent jau apytiksliai būtų galima sukurti vieningą įvertinimo ir išvadų metodiką.

11.6.1 Šeimos

Atsako pasiskirstymo, susiejimo funkcijos ir įvairios kitos informacijos, reikalingos modeliavimo pratyboms, derinys vadinamas apibendrinta tiesinio modelio šeima.

Kiekvienas atsako pasiskirstymas suteikia daugybę susiejimo funkcijų, kad vidurkis būtų susietas su tiesiniu prediktoriumi. Automatiškai pasiekiami duomenys pateikiami 4 lentelėje.

4 lentelė. Šeimų sąrašas

Šeimos pavadinimas	Susiejimo funkcija
`binomial`	`logit, probit, log, cloglog`
`gaussian`	`identity, log, inverse`
`Gamma`	`identity, inverse, log`
`inverse.gaussian`	`1/mu^2, identity, inverse, log,`
`poisson`	`identity, log, sqrt`
`quasi`	`logit, probit, cloglog, identity, inverse, log, 1/mu^2, sqrt`

11.6.2 Funkcija `glm()`

Kadangi atsako pasiskirstymas priklauso nuo stimulo kintamųjų tik per vieną tiesinę funkciją, apibendrinto modelio tiesinei daliai nurodyti vis tiek gali būti naudojamas tas pats mechanizmas, kuris buvo naudojamas tiesiniams modeliams. Šeima turi būti apibrėžta kitaip. „R“ pakete naudojam funkcija `glm()`, kad atitiktų apibendrintą tiesinį modelį. Jos forma:

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

Vienintelė nauja funkcija yra family.generator, kuris yra instrumentas, kuriuo apibūdinama šeima. Tai funkcijos pavadinimas, sukuriantis funkcijų ir išraiškų, kartu apibrėžiančių ir valdančių modelį bei įvertinimo procesą, sąrašą. Nors iš pirmo žvilgsnio tai gali atrodyti šiek tiek sudėtinga, jo naudojimas yra gana paprastas. Kai kurie žemiau pateikti pavyzdžiai paaiškina procesą.

Šeima: `gaussian`

Iškvietimas toks kaip

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

pasiekia tą patį rezultatą kaip

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

bet daug mažiau efektyviai. Atkreipkite dėmesį, kad Gauso šeimai nėra automatiškai suteikiama galimybė pasirinkti nuorodas, todėl jokie parametrai neleidžiami. Jei norint išspręsti problemą reikalinga nestandartinių ryšių Gauso šeima, tai dažniausiai galima pasiekti pasitelkiant Kvazi šeimą, kaip pamatysime vėliau.

Šeima: `binomial`

Nagrinėkime pavyzdį. Vienoje Egėjo jūros saloje Kalythos vyrai kenčia nuo įgimtos akių ligos, kurios padariniai labiau išryškėja senstant. Buvo tiriami įvairaus amžiaus saloje gyvenančių vyrų pavyzdžiai dėl aklumo, ir užfiksuoti rezultatai. Duomenys pateikiami žemiau:

Amžius:	20	35	45	55	70
Vyrų skaičius:	50	50	50	50	50
Aklų vyrų skaičius	6	17	26	37	44

Problema yra suderinti šiuos duomenis su logistiniais ir Probit modeliais, ir įvertinti LD50 kiekvienam modeliui. LD50 yra vyro amžius, kai aklumo tikimybė yra 50%.

Jei *y* yra aklų vyrų, kurių amžius *x*, skaičius ir *n* yra testuotų vyrų skaičius, abu modeliai turi formą

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

kur Probit modelio atveju $F(z) = \Phi(z)$ yra standartinė normaliojo paskirstymo funkcija ir Logit modelio atveju (numatytasis) $F(z) = e^z/(1 + e^z)$. Abiem atvejais LD50 yra

$$LD50 = -\beta_0/\beta_1$$

tai yra, taškas, kuriame paskirstymo funkcijos argumentas yra lygus nuliui. Pirmasis žingsnis yra nustatyti duomenis kaip duomenų sistemą

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

Pritaikius binominį modelį naudojant funkciją `glm()` yra trys atsako galimybės:

- Jei atsakas yra vektorius, laikoma, kad jis turi dvejetainius duomenis, taigi turi būti 0 = 1 vektorius.
- Jei atsakas yra dviejų stulpelių matrica, laikoma kad pirmame stulpelyje nurodomas bandymų pasisekimų skaičius, o antrame – nesėkmių skaičius.
- Jei atsakas yra faktorius, pirmasis jo lygis laikomas nesėkme (0), o visi kiti lygiai – „sėkme“ (1).

Čia mums reikia antrosios iš šių sąlygų, todėl mes įtraukiame matricą į savo duomenų sistemą:

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

Norint pritaikyti modelius, mes naudojame

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

Kadangi Logit susiejimas yra numatytasis, antrojo iškvietimo metu parametras gali būti praleistas. Norėdami pamatyti kiekvieno pritaikymo rezultatus, mes galėtime naudotis

```
> summary(fmp)
> summary(fml)
```

Abu modeliai tinka deramai. Norint rasti LD50 įvertį, mes galime naudoti paprastą funkciją:

```
> ld50 <- function(b) -b[1]/b[2]
> ldp <- ld50(coef(fmp)); ldl <- ld50(coef(fml)); c(ldp, ldl)
```

Faktiniai šių duomenų įverčiai atitinkamai yra 43.663 ir 43.601 metai.

Puasono modelis

Su Puasono šeima numatytoji susiejimo funkcija yra `log`, ir praktiškai pagrindinis šios šeimos panaudojimas yra pritaikyti pakaitinius Puasono loginius-tiesinius modelius dažnio duomenims, kurių tikrasis pasiskirstymas dažnai būna daugialypis. Tai yra didelis ir svarbus

dalykas, kurio toliau čia neaptarsime. Tai netgi sudaro didžiąją dalį ne Gauso apibendrintų modelių naudojimo.

Kartais tikrai Puasono duomenys atsiranda praktikoje ir praeityje jie buvo dažnai analizuojami kaip Gauso duomenys po logaritminės ar kvadratinės šaknies transformacijos. Kaip pastarojo alternatyva, gali būti taikomas Puasono apibendrintas tiesinis modelis, kaip šiame pavyzdyje:

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

Kvazi-tikėtinumo modeliai

Visoms šeimoms atsako dispersija priklausys nuo vidurkio ir skalės parametras bus daugiklis. Dispersijos priklausomybės nuo vidurkio forma yra atsako pasiskirstymo charakteristika. Pavyzdžiui, Puasono pasiskirstymas $Var[y] = \mu$.

Kvazi-tikėtinumo įverčiui ir išvedimui tikslus atsakų pasiskirstymas nenurodytas, veikiau tik susiejimo funkcija ir dispersijos funkcijos forma, nes ji priklauso nuo vidurkio. Kadangi kvazi-tikėtinumo įvertis naudoja formaliai identiškus metodus tiems, kurie skirstomi pagal Gauso pasiskirstymą, ši šeima suteikia galimybę Gauso modelius pritaikyti nestandartinėms susiejimo funkcijoms arba dispersinėms funkcijoms. Pavyzdžiui, apsvarstykite netiesinės regresijos pritaikymą

$$y = \frac{\theta_1 z_1}{z_2 - \theta_2} + e$$

kuris gali būti parašytas kaip

$$y = \frac{1}{\beta_1 x_1 + \beta_2 x_2} + e$$

where $x_1 = \frac{z_2}{z_1},\ x_2 = -\frac{1}{z_1},\ \beta_1 = \frac{1}{\theta_1}, \beta_2 = \theta_2/\theta_1$.

Tarkime, tinkama duomenų sistema turi būti nustatyta, kad galėtume pritaikyti šią netiesinę regresiją kaip

```
> nlfit <- glm(y ~ x1 + x2 - 1,
```

```
family = quasi(link=inverse, variance=constant),
data = biochem)
```

11.7 Netiesiniai mažieji kvadratai ir didžiausio tikėtinumo modeliai

Kai kurias netiesinių modelių formas galima pritaikyti pagal apibendrintus teisinius modelius (`glm()`). Bet daugeliu atvejų turime kreiptis į netiesinės kreivės pritaikymo problemą kaip į netiesinės optimizacijos problemą. Mes ieškome parametrų riekšmių, kurios sumažina tam tikrą netinkamumo indeksą, ir tai darome išbandydami įvairias parametrų reikšmes pakartotinai. Skirtingai nei, pavyzdžiui, tiesinė regresija, nėra jokios garantijos, kad procedūra sutaps esant patenkinamiems įverčiams. Visiems metodams reikia pirminių spėjimų apie tai, kokias parametrų reikšmes bandyti, o konvergencija gali labai priklausyti nuo pradinių reikšmių kokybės.

11.7.1 Mažieji kvadratai

Vienas iš būdų netiesiniam modeliui pritaikyti yra sumažinant kvadratinių paklaidų sumą ar liekanas. Šis metodas yra prasmingas, jei pastebėtos paklaidos galėjo kilti iš normaliojo paskirstymo. Pavyzdžiui duomenys yra:

```
> 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)
```

Tinkamumo kriterijus, kurį reikia sumažinti, yra:

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

Norint atlikti pritaikymą, reikia parametrų pradinių įverčių. Vienas iš būdų rasti prasmingas pradines reikšmes yra nubraižyti duomenis, atspėti kai kurias parametrų reikšmes ir panaudoti šių reikšmių modelio kreivę.

```
> plot(x, y)
> xfit <- seq(.02, 1.1, .05)
```

```
> yfit <- 200 * xfit/(0.1 + xfit)
> lines(spline(xfit, yfit))
```

Galėtume padaryti geriau, tačiau šios pradinės 200 ir 0,1 reikšmės atrodo tinkamos. Dabar atlikite tinkamus veiksmus:

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

Po pritaikymo `out$minimum` yra kvadratinių paklaidų suma, `out$estimate` yra parametrų mažiausių kvadratų įverčiai. Norėdami gauti apytiksles standartines įverčių paklaidas, mes naudosime:

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

Aukščiau esančioje eilutėje atimtas skaitmuo 2 žymi parametrų skaičių. 95% pasikliautinasis intervalas būtų parametro įvertis ± 1,96 standartinė paklaida. Galima užrašyti mažiausius kvadratus, kurie tinka naujame brėžinyje:

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

Standartinis paketas *stats* suteikia daug daugiau galimybių netiesinius modelius pritaikyti mažiausiais kvadratais. Modelis, kurį ką tik pritaikėme yra Michaelis-Menten modelis, todėl galime naudoti

```
> 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 Didžiausias tikėtinumas

Didžiausias tikėtinumas yra netiesinio modelio derinimo metodas, kuris taikomas net tada, kai paklaidos nėra normalios. Metodas nustato parametrų reikšmes kurios maksimaliai padidina logaritminį tikėtinumą, arba analogiškai, kurie sumažina neigiamą logaritminį tikėtinumą. Šis pavyzdys tinka logistinio modelio duomenims, kuriems aiškiai taip pat galėtų tikti funkcija `glm()`. Duomenys yra:

```
> 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)
```

Neigiamas logaritminis tikėtinumas, kurį reikia sumažinti yra:

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

Mes pasirenkame prasmingas pradines reikšmes ir atliekame tinkamus veiksmus:

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

Po pritaikymo, `out$minimum` yra neigiamas logaritminis tikėtinumas ir `out$estimate` yra parametrų didžiausio tikėtinumo paklaida. Norėdami gauti apytiksles standartines paklaida, mes atliekame:

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

11.8 Keletas nestandartinių modelių

Mes baigiame šį skyrių tik trumpai paminėdami kai kurias kitas „R“ paketo galimybes, skirtas ypatingoms regresijos ir duomenų analizės problemoms spręsti.

- **Mišrūs modeliai**. Rekomenduotas paketas *nlme*, kurį galima parsisiųsti iš https://CRAN.R-project.org/package=nlme, suteikia funkcijas `lme()` ir `nlme()` tiesiniams ir netiesiniams mišraus efekto modeliams, tai yra tiesinė ir netiesinė regresija,

kurioje kai kurie koeficientai atitinka atsitiktinius efektus. Šioms funkcijoms atlikti labai reikalinga formulė modeliams nurodyti.

- **Lokalios aproksimacijos regresijos**. Funkcija `loess()` tinka neparametrinei regresijai, naudojant vietinę svertinę regresiją. Tokios regresijos yra naudingos norint išryškinti netvarkingų duomenų tendenciją arba norint sumažinti duomenis, kad būtų galima atlikti šiek tiek įžvalgų dideliame duomenų rinkinyje. Funkcija `loess` yra standartiniame pakete *stats,* kartu su projekcijos sekimo regresijos kodu.
- **Atsparioji regresija**. Yra keletas funkcijų, leidžiančių pritaikyti regresijos modelius tokiu būdu, kuris būtų atsparus kraštutinių pašalinių duomenų poveikiui. Funkcija `lqs` rekomenduojamame pakete *MASS* (https://CRAN.R-project.org/package=MASS) suteikia moderniausius algoritmus. Tame pačiame pakete yra ir mažiau atsparūs, bet statistiškai efektyvesni metodai, pavyzdžiui, funkcija `rlm`.
- **Papildomi modeliai**. Šiuo metodu siekiama sukonstruoti regresijos funkciją iš determinuojančių kintamųjų lygiaverčių priedų funkcijų, dažniausiai vieną kiekvienam determinuojančiam kintamajam. Funkcijos `avas` ir `ace` yra pakete *acepack* (https://CRAN.R-project.org/package=acepack) ir funkcijos `bruto` ir `mars`, esančios pakete *mda* (https://CRAN.R-project.org/package=mda) pateikia keletą šių metodų pavyzdžių.
- **Medžio struktūra grįsti modeliai**. Užuot ieškoję aiškaus globalaus tiesinio modelio numatymo ar interpretavimo, medžio struktūra grįsti modeliai siekia, kad duomenys turėtų būti išskaidyti į dvi dalis. Rezultatai dažnai leidžia suprasti, kad kiti duomenų analizės metodai nėra naudingi. Modeliai vėl nurodomi įprasta tiesinio modelio forma. Modelio pritaikymo funkcija `tree()`, bet daugelis kitų bendrinių funkcijų, tokių kaip `plot()` ir `text()` yra gerai pritaikyti medžiai paremtų modelio rezultatų pateikimui grafiniu būdu. Medžio struktūra grįsti modeliai yra prieinami naudojant paketus *rpart* ir

tree, kurie atitinkamai gali būti parsisiunčiami iš https//CRAN.R-project.org/package=rpart ir https://CRAN.R-project.org/package=tree.

12 Grafinės procedūros

Grafinės galimybės yra svarbūs ir nepaprastai universalūs „R“ statistinio paketo aplinkos komponentai. Šias galimybes galima naudoti įvairiausių statistinių grafikų rodymui ar sukurti visiškai naujų tipų grafikus.

Grafikos priemonės (stulpelinės diagramos, stačiakampės diagramos, kvantilių grafikai, histogramos, tankio grafikai) gali būti naudojami tiek interaktyviame, tiek paketiniame režime, tačiau dažniausiai interaktyvus naudojimas yra produktyvesnis. Interaktyvus naudojimas taip pat yra lengvas, nes paleidimo metu „R“ inicijuoja grafikos priemonės tvarkyklę, kuri atidaro specialų grafikos langą interaktyvios grafikos rodymui. Nors tai daroma automatiškai, gali būti naudinga žinoti, kad UNIX sistemoje yra naudojama komanda `X11()`, komanda `windows()` operacinėje sistemoje Windows ir `quartz()` komanda MacOS sistemoje. Visada galima atidaryti naują priemonę pasinaudojus komanda `dev.new()`.

Paleidus priemonės tvarkyklę, „R“ braižymo komandos gali būti naudojamos norint sukurti įvairius grafinius langus ir visiškai naujas langų rūšis. Braižymo komandos yra suskirstytos į tris pagrindines grupes:

- **Aukšto lygio** braižymo funkcijos grafikos priemonėje sukuria naują brėžinį galbūt su ašimis, etiketėmis, pavadinimais ir pan.
- **Žemo lygio** braižymo funkcijos prideda daugiau informacijos prie jau esamo brėžinio, pavyzdžiui, papildomų taškų, linijų ir etikečių.
- **Interaktyvios** grafikos funkcijos suteikia galimybę interaktyviai pridėti informaciją prie esamo brėžinio arba iš jo gauti informaciją, naudojant žymėjimo prietaisą, pavyzdžiui, pelę.

Be to, „R“ palaiko sąrašą grafinių parametrų, kuriais galima manipuliuoti, kad būtų galima pritaikyti brėžinius.

12.1 Aukšto lygio braižymo funkcijos

Aukšto lygio braižymo funkcijos yra suprojektuotos generuoti visą duomenų, perduotų kaip funkcijos argumentai, brėžinį. Jei reikia, ašys, etiketės ir pavadinimai sugeneruojami automatiškai (nebent jūs prašote kitaip). Aukšto lygio braižymo komandos visada pradeda naują brėžinį, prireikus ištrindamos dabartinį brėžinį.

12.1.1 Funkcija `plot()`

Viena iš dažniausiai „R“ pakete naudojamų braižymo funkcijų yra `plot()`. Tai yra bendrinė funkcija, kai pateikto brėžinio tipas priklauso nuo pirmojo argumento tipo arba klasės.

5 lentelė. Funkcijos `plot()` galimybės

`plot(x,y)` `plot(xy)`	Jei *x* ir *y* yra vektoriai, `plot(x,y)` sukuria *y* prieš *x* sklaidos diagramą. Tas pats efektas gali būti pasiektas pateikus vieną argumentą (antra forma) kaip dviejų elementų `x` ir `y` sąrašą arba dviejų stulpelių matricą.
`plot(x)`	Jei *x* yra laiko eilutės, taip gaunamas laiko eilučių grafikas. Jei *x* yra skaitinis vektorius, jis sukuria vektoriaus reikšmių diagramą pagal jų indeksą vektoriuje. Jei *x* yra kompleksinis vektorius, jis sukuria įsivaizduojamų ir realių vektorinių elementų dalių brėžinį.
`plot(f)` `plot(f, y)`	*f* yra faktoriaus objektas, *y* yra skaitinis vektoriu. Pirmoji forma sukuria *f* juostos brėžinį; antroji forma sukuria *y* stačiakampė diagramą kiekvienam *f* lygiui.

`plot(df)`
`plot(~ expr)`
`plot(y ~ expr)`
	`df` yra duomenų sistema, *y* yra bet koks objektas, `expr` yra objektų pavadinimų sąrašas, atskirtas ženklu „+“ (pvz., `a + b + c`). Pirmosios dvi formos sukuria kintamųjų duomenų sistemoje paskirstymo brėžinius (pirmoji forma) arba įvardytų objektų skaičiaus paskirstymo brėžinius (antroji forma). Trečioji forma braižo *y* prieš kiekvieną objektą, įvardytą `expr`.

12.1.2 Daugiamačių duomenų vaizdavimas

„R“ suteikia dvi labai naudingas funkcijas, skirtas daugiamačiams duomenims vaizduoti. Jei `X` yra skaitinė matrica arba duomenų sistema, tai komanda

```
> pairs(X)
```

sukuria sklaidos diagramos matricą (`X` stulpeliais apibrėžtų kintamųjų), tai yra, kiekvienas X stulpelis yra nubraižytas kiekviename kitame X stulpelyje ir gaunami `n(n-1)` brėžiniai yra išdėstyti matricoje, o brėžinių skalės yra pastovios virš matricos eilučių ir stulpelių.

Kai trys arba keturi kintamieji yra įtraukti, tai `coplot` gali būti labiau informatyvesnis. Jei `a` ir `b` yra skaitinis vektorius ir `c` yra skaitinis vektorius arba faktoriaus objektas (visi vienodo ilgio), tada komanda

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

sukuria `a` prieš `b` nurodytoms `c` reikšmėms sklaidos diagramų skaičių. Jei `c` yra faktorius, tai paprasčiausiai reiškia, kad `a` yra nubraižytas prieš `b` kiekvienam `c` lygiui. Kai `c` yra skaitinis, jis yra padalintas į keletą formuojamųjų intervalų ir kiekvienas intervaluas `a` yra nubraižytas prieš `b` reikšmėms `c` intervale. Intervalų skaičių ir vietą galima valdyti naudojant argumentą `given.values=` į `coplot()`, tada fukcija `co.intervals()` yra naudinga norint pasirinkti intervalus. Taip pat, galite naudoti du duotus kintamuosius su tokia komanda kaip

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

kuri sukuria `a` prieš `b` kiekvienam jungtiniam c ir d poveikio intervalui sklaidos diagramas.

Funkcijos `coplot()` ir `pairs()` abi priima argumentą `panel=` , kurį galima naudoti pritaikant kiekviename skydelyje rodomą brėžinio tipą. Numatytoji funkcija `points()` naudojama sukurti skaidos diagramą, bet pateikiant dar kokį nors dviejų `x` ir `y` vektorių, kaip `panel=` reikšmės, žemo lygio grafikos funkciją galite sukurti bet kokio tipo brėžinį.

12.1.3 Grafikų rodymas

Kitos aukšto lygio braižymo funkcijos sukuria skirtingų tipų brėžinius (6 lentelė).

6 lentelė. Aukšto lygio braižymo funkcijos

`qqnorm(x)`
`qqline(x)`
`qqplot(x, y)`
	Paskirstymo – palyginimo brėžiniai. Pirmoji forma nubraižo skaitinį vektorių `x` prieš tikėtinas Normaliosios eilės reikšmes, o antrasis prideda tiesę prie tokio grafiko brėždamas liniją per paskirstymo ir duomenų kvartilius. Trečioji forma nubraižo `x` kvantiles palyginti su `y`, kad būtų galima palyginti jų atitinkamą pasiskirstymą.

`hist(x)`
`hist(x, nclass=n)`
`hist(x, breaks=b, ...)`
	Sukuria skaitinio vektoriaus `x` histogramą. Paprastai pasirenkamas protingas klasių skaičius, tačiau rekomendacija gali būti teikiama kartu argumentu `nclass=`. Arba lūžio taškus galima tiksliai nurodyti naudojant argumentą `breaks=`. Jei yra duotas argumentas `probability=TRUE`, stulpeliai žymi santykinius dažnius, padalintus iš pločio, o ne skaičių.

`dotchart(x, ...)`	Sukuria duomenų, esančių `x`, taškinę diagramą. Taškinėje diagramoje *y* ašis pateikia duomenų, esančių `x`, žymėjimą, o *x* ašis – jų vertę. Pavyzdžiui, tai leidžia lengvai vizualiai pasirinkti visus duomenų įrašus, kurių reikšmės yra nurodytuose diapazonuose.

`image(x, y, z, ...)`
`contour(x, y, z, ...)`
`persp(x, y, z, ...)`
	Trijų kintamųjų brėžiniai. Funkcija `image` nubrėžia stačiakampių tinklelį, naudodamas skirtingas spalvas, kad būtų parodyta `z` reikšmė, funkcija `contour` brėžia linijas, kad parodytų `z` reikšmę, ir funkcija `persp` nupiešia 3D paviršių.

12.1.4 Aukšto lygio braižymo funkcijų argumentai

Yra keletas argumentų, kuriuos galima perduoti aukšto lygio braižymo funkcijoms (7 lentelė).

7 lentelė. Aukšto lygio braižymo funkcijų argumentai

`add=TRUE`	Priverčia šią funkciją veikti kaip žemo lygio grafikos funkciją, ant brėžinio uždedant esamą brėžinį (tik kai kurios funkcijos).
`axes=FALSE`	Stabdo ašių generavimą, naudinga pridedant savo pasirinktines ašis su funkcija `axis()`. Numatytasis `axes=TRUE` reiškia ašis.

log="x" log="y" log="xy"	x,y arba abi ašys yra logaritminės. Tai bus naudinga daugeliui, bet ne visų tipų brėžiniams.
type=	Argumentas type= kontroliuoja sukurto brėžinio tipą taip: type="p" nubraižo atskirus taškus (numatytasis) type="l" nubraižo linijas type="b" brėžinio taškai sujungti linijomis type="o" brėžinio taškai uždengti linijomis type="h" nubraižo vertikalias linijas nuo taškų iki nulinės ašies (didelis tankis) type="s", type="S" Žingsnio funkcija. Pirmoje formoje vertikalės viršutinė dalis nurodo tašką; antroje formoje – apatinė dalis. type="n" Jokio braižymo. Tačiau ašys vis dar brėžiamos (pagal numatytuosius nustatymus), o koordinačių sistema nustatoma pagal duomenis. Idealiai tinka kurti brėžinius su tolesnėmis žemo lygio grafikos funkcijomis.
xlab=string ylab=string	x ir y ašių etiketės. Naudokite šiuos argumentus, kad pakeistumėte numatytąsias etiketes, paprastai objektų, naudojamų kvietime į aukšto lygio braižymo funkciją, pavadinimus.
main=string	Paveikslo pavadinimas, užrašytas brėžinio viršuje dideliu šriftu.
sub=string	Antraštė, dedama šiek tiek žemiau x ašies mažesniu šriftu.

12.2 Žemo lygio braižymo komandos

Kartais aukšto lygio braižymo funkcijos nesukuria tiksliai tokio brėžinio, kokio norite. Tokiu atveju, norint pridėti papildomos informacijos (pvz., taškų, linijų ar teksto) prie esamo brėžinio, galima naudoti žemo lygio braižymo komandas. Keletas naudingesnių žemo lygio braižymo komandų pateikiama 8 lentelėje.

8 lentelė. Žemo lygio braižymo komandos.

`points(x, y)` `lines(x, y)`	Prideda taškus arba sujungtas linijas prie esamo brėžinio. Funkcijos `plot()` argumentas `type=` taip pat gali būti perduotas šioms funkcijoms.
`text(x, y, labels, ...)`	Įtraukia tekstą į brėžinį taškuose, kuriuos nurodo `x,y`. Paprastai etiketės yra sveikasis skaičius arba simbolių vektorius, tuo atveju `labels[i]` yra braižomos taške `(x[i],y[i])`. Numatytasis yra `1:length(x)`. Pastaba, kad ši funkcija dažnai naudojama sekoje `plot(x, y, type="n"); text(x, y, names)` Grafikos paramentras `type="n"` slopina taškus, bet nustato ašis. Funkcija `text()` tiekia specialiuosius simbolius, kuriuos nurodo simbolių vektoriaus taškų pavadinimai.

`abline(a, b)`
`abline(h=y)`
`abline(v=x)`
`abline(lm.obj)`
	Esančiame brėžinyje prideda liniją, kurios krypties koeficientas yra `b` ir atkarpa `a`. Antrojoje formoje `h=y` gali būti naudojamas apibrėžti horizontaliųjų linijų, einančių per brėžinį, y koordinates. Trečiojoje formoje `v=x` gali būti naudojamas apibrėžti vertikalių linijų x koordinatėms. Taip pat `lm.obj` gali būti sąrašas su `coefficients` komponentu, kurio ilgis 2 (pavyzdžiui, modelio pritaikymo funkcijų rezultatas), kurie laikomi atkarpa ir kryptimi nurodyta tvarka.

`polygon(x, y, ...)`	Piešia daugiakampį, apibrėžtą pagal rikiuotas viršūnes, esančias (x, y) ir (pasirinktinai) nuspalvina su brūkšninėmis linijomis arba užpildo, jei grafikos įrenginys leidžia užpildyti brėžinį.
`legend(x, y, legend, ...)`	Prideda legendą dabartiniame grafike nurodytoje vietoje. Braižomi simboliai, linijų stiliai, spalvos ir kt., yra identifikuojami su etiketėmis simbolių vektoriuje `legend`. Bent jau vienas kitas argumentas `v` (to paties ilgio vektorius `legend`) su atitinkamomis braižymo vieneto reikšmėmis taip pat turi būti pateikiamas taip:

	`legend( ,fill=v)` spalvos užpildymo laukams `legend( ,col=v)` Spalvos, kuriomis bus brėžiami taškai ar linijos `legend( ,lty=v)` linijos stilius `legend( ,lwd=v)` linijos plotis `legend( ,pch=v)` simbolių brėžinys (simbolių vektorius)
`title(main, sub)`	Prideda `main` pavadinimą prie dabartinio brėžinio viršaus dideliu šriftu ir (pasirinktinai) antraštę apačioje mažesniu šriftu.
`axis(side, ...)`	Prideda ašį prie esamo brėžinio toje pusėje, kurią pateikė pirmasis argumentas (nuo 1 iki 4, skaičiuojant pagal laikrodžio rodyklę iš apačios). Kiti argumentai kontroliuoja ašies išdėstymą brėžinyje ar šalia jo, taip pat pažymi vietas ir etiketes. Naudinga pridedant pasirinktines ašis po `plot()` iškvietimo su argymentu `axes=FALSE`.

Žemo lygio braižymo funkcijoms paprastai reikalinga tam tikra padėties nustatymo informacija (pvz., x ir y koordinatės), kad būtų galima nustatyti, kur dėti naujus grafiko elementus. Koordinatės pateikiamos atsižvelgiant į vartotojo koordinates, kurias nusako ankstesnė aukšto lygio grafikos komanda ir kurios parenkamos pagal pateiktus duomenis.

Kai reikia `x` ir `y` argumentų, taip pat pakanka pateikti vieną argumentą, kuris yra sąrašas elementų, pavadintų `x` ir `y`. Taip pat tinkama įvestis yra dviejų stulpelių matrica. Tokiu būdu veikia tokia funkcija kaip `locator()` ir gali būti naudojama interaktyviai nurodyti brėžinio vietas.

12.2.1 Matematinė anotacija

Kai kuriais atvejais naudinga į grafiką įtraukti matematinius simbolius ir formules. Tai galima pasiekti „R“ pakete nurodant išraišką, o ne naudojant simbolių eilutes bet kuriame iš `text`, `mtext`, `axis` arba `title`. Pavyzdžiui, ši komanda nubrėžia Binominės skirstinio funkcijos formulę:

```
> text(x, y, expression(paste(bgroup("(", atop(n, x), ")"), p^x,
q^{n-x})))
```

Daugiau informacijos, įskaitant išsamų galimų naudoti „R“ pakete funkcijų sąrašą, galite gauti naudodami komandas:

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

12.3 Sąveika su grafika

„R“ taip pat teikia funkcijas, leidžiančias vartotojui pele išgauti ar papildyti informaciją grafike. Paprasčiausias iš šių funkcijų yra `locator()`.

`locator()` paprastai kviečiama be argumentų. Tai ypač naudinga interaktyviai parenkant grafinių elementų, tokių kaip legendos ar etiketės, vietas, kai sunku iš anksto apskaičiuoti, kur turėtų būti grafikas. Pavyzdžiui, norint įdėti šiek tiek informatyvaus teksto šalia atokesnio taško, reikia naudoti komandą

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

`locator()` bus ignoruojamas, jei dabartinis įrenginys, toks kaip `postscript`, nepalaiko interaktyvaus žymėjimo.

Komanda `locator(n, type)` laukia, kol vartotojas kairiuoju pelės mygtuku pasirinks dabartinio brėžinio vietas. Tai tęsiasi kol `n` (numatyti 512) taškų buvo pasirinkta arba paspaustas kitas pelės mygtukas. Argumentas `type` leidžia brėžti pasirinktuose taškuose ir turi tą patį efektą kaip ir aukšto lygio grafikos komandos, pagal nutylėjimą neatliekamas braižymas. `locator()` grąžina taškų, pasirinktų kaip sąrašas su dviem komponentais `x` ir `y`, vietas.

Komanda `identify(x, y, labels)` leidžia vartotojui paryškinti bet kurį iš taškų, apibrėžtų `x` ir `y` (naudojant kairįjį pelės mygtuką), nubraižant atitinkamą etikečių komponentą netoliese (arba taško indekso numerį, jei etikečių nėra). Grąžina pasirinktų taškų indeksus, kai paspaudžiamas kitas mygtukas.

Kartais norime nustatyti konkrečius brėžinio taškus, o ne jų pozicijas. Pvz.,galime paprašyti, kad vartotojas iš grafinio lango parinktų kokį nors dominantį stebėjimą ir tam tikru būdu

manipuliuotų tuo stebėjimu. Duotos (x; y) koordinatės dviejuose skaitiniuose vektoriuose `x` ir `y`, galime naudoti funkciją `identify()` kaip nurodyta žemiau:

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

Funkcija `identify()` neatlieka jokio braižymo, bet tiesiog leidžia vartotojui perkelti pelės žymeklį ir spustelėti kairįjį pelės mygtuką šalia taško. Jei šalia pelės žymeklio yra taškas, jis bus pažymėtas šalia esančiu jo rodyklės numeriu (jo vieta `x` arba `y` vektoriuose). Arba galite naudoti tam tikrą informacinę eilutę kaip paryškinimą naudodami etikečių argumentą į `identify()`, arba visai išjungti žymėjimą su argumentu `plot = FALSE`. Kai procesas nutraukiamas, `identify()` grąžina pasirinktų taškų indeksus. Šiuos indeksus galite panaudoti pasirinktiems taškams iš originalių vektorių `x` ir `y` išgauti.

12.4 Grafikos parametrų naudojimas

Kuriant grafiką, ypač pristatymo ar publikavimo tikslais, „R“ numatytosios reikšmės ne visada sukuria tiksliai tai, ko reikia. Tačiau, naudodami grafikos parametrus, galite tinkinti beveik kiekvieną brėžinio aspektą. „R“ palaiko daugybę grafikos parametrų, kurie, be kita ko, kontroliuoja tokius dalykus kaip linijos stilius, spalvos, figūros išdėstymas, teksto lygiavimas. Kiekvienas grafikos parametras turi pavadinimą, tokį kaip pavyzdžiui, `col`, kuris kontroliuoja spalvą ir reikšmę (spalvos numeris)

Kiekvienam aktyviai priemonei palaikomas atskiras grafikos parametrų sąrašas, o inicijuojant kiekviena priemonė turi numatytąjį parametrų rinkinį. Grafikos parametrus galima nustatyti dviem būdais: arba visam laikui, paveikiant visas grafikos funkcijas, kuriomis pasiekiama dabartinė priemonė; arba laikinai, paveikiant tik vieną grafikos funkcijos iškvietimą.

12.4.1 Ilgalaikiai pakeitimai: funkcija `par()`

Funkcija `par()` naudojamas prieiti prie esamos grafinės priemonės grafikos parametrų sąrašo ir jį modifikuoti. Funkcija `par()` be argumentų grąžina visų esamos priemonės grafikos parametrų ir jų reikšmių sąrašą.

Funkcija `par(c("col", "lty"))` su simbolių vektoriaus argumentu grąžina tik nurodytus grafikos parametrus (vėlgi kaip sąrašą).

Funkcija `par(col=4, lty=2)` su nurodytais argumentais (arba vienu sąrašo argumentu) nustato nurodytų grafikos parametrų reikšmes ir grąžina originalias parametrų reikšmes kaip sąrašą.

Grafikos parametrų nustatymas naudojant funkciją `par()` visam laikui keičia parametrų reikšmę ta prasme, kad nauja reikšmė paveiks visus būsimus kvietimus į grafikos funkcijas (dabartinėje priemonėje). Galite įsivaizduoti, kaip tokiu būdu nustatyti grafikos parametrus kaip parametrų „numatytųjų“ reikšmių nustatymą, kurie bus naudojami visoms grafikos funkcijoms, nebent būtų nurodyta kita reikšmė.

Atminkite, kad kvietimas į `par()` visada turi įtakos visuotinėms grafikos parametrų reikšmėms, net kai `par()` yra kviečiamas iš funkcijos. Tai dažnai yra nepageidaujamas elgesys, paprastai norime nustatyti kai kuriuos grafikos parametrus, atlikti tam tikrus brėžinius ir atkurti pradines reikšmes, kad nepaveiktume vartotojo „R“ seanso. Pradines reikšmes galite atkurti išsaugodami `par()` rezultatą atlikdami pakeitimus ir atkurdami pradines reikšmes, kai braižymas baigtas.

```
> oldpar <- par(col=4, lty=2)
  . . . plotting commands . . .
> par(oldpar)
```

Norėdami išsaugoti ir atkurti visus grafinius parametrus [10] naudokite

```
> oldpar <- par(no.readonly=TRUE)
  . . . plotting commands . . .
> par(oldpar)
```

[10] Kai kurie grafikos parametrai, tokie kaip dabartinės priemonės dydis, yra skirti tik informacijai.

12.4.2 Laikini pakeitimai: argumentai į grafikos funkcijas

Grafikos parametrai taip pat gali būti perduodami beveik bet kuriai grafikos funkcijai kaip įvardyti argumentai. Tai turi tokį patį poveikį kaip ir argumentų perdavimas į funkciją `par()`, išskyrus tai, kad pakeitimai galioja tik funkcijos iškvietimo metu. Pavyzdžiui,

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

sukuria sklaidos diagramą naudojant pliuso ženklą kaip braižymo ženklą, nekeičiant numatytojo braižymo simbolio būsimiems brėžiniams. Deja, tai nėra įgyvendinama visiškai nuosekliai, todėl kartais reikia nustatyti ir atstatyti grafikos parametrus naudojant `par()`.

12.5 Grafikos parametrų sąrašas

Tolesniuose skyriuose aprašomi daugelis dažniausiai naudojamų grafikos parametrų. Grafikos parametrai bus pateikti tokia forma: `name=value`. Čia `name` yra parametro pavadinimas, tai yra, argumento pavadinimas, naudojamas iškviečiant `par()` arba grafikos funkcijas. `value` yra tipinė reikšmė, kurią galite naudoti nustatydami parametrą. Atminkite, kad ašys nėra grafikos parametras, o kelių brėžinių metodų argumentas.

12.5.1 Grafikos elementai

„R“ brėžiniai sudaryti iš taškų, linijų, teksto ir daugiakampių (užpildytų sričių.) Egzistuoja grafiniai parametrai, kurie valdo šių grafinių elementų brėžinį.

9 lentelė. Grafikos parametrai

`pch="+"`	Simbolis, kuris naudojamas žymint taškus. Numatytoji reikšmė skiriasi atsižvelgiant į grafikos tvarkykles, tačiau dažniausiai tai yra „`o`“. Nubraižyti taškai paprastai atrodo šiek tiek aukščiau arba žemiau tinkamos padėties, nebent naudojate „`.`“ kaip braižymo simbolį, kuris sukuria centre esančius taškus.
`pch=4`	Kai `pch` yra duotas yra pateiktas kaip sveikas skaičius nuo 0 iki 25 imtinai, sukuriamas specializuotas braižymo simbolis. Norėdami pamatyti, kas yra simboliai, naudokite komandą `> legend(locator(1), as.character(0:25), pch = 0:25)`

	Gali pasirodyti, kad nuo 21 iki 25 kartojasi ankstesni simboliai, tačiau juos galima spalvinti skirtingais būdais. Papildomai, `pch` gali būti simbolis arba skaičius intervale `32:255`, vaizduojantis dabartinio šrifto simbolį.
`lty=2`	Linijų tipai. Alternatyvūs linijų stiliai nepalaikomi visose grafikos priemonėse (ir skiriasi nuo tų, kurie veikia), tačiau pirmas (`1`) linijos tipas visada yra vientisa linija, nulinis (`0`) linijos tipas visada yra nematomas, o antras (`2`) ir naujesni linijos tipai yra taškinė ar punktyrinė linija arba tam tikras jų derinys.
`lwd=2`	Pageidaujamas linijų plotis. Įtakoja ašių linijas, taip pat linijas, brėžtas su `lines()` ir pan. Ne visos priemonės tai palaiko, o kai kurie naudojasi pločio apribojimais.
`col=2`	Spalvos, naudojamos taškams, linijoms, tekstui, užpildytoms sritims ir vaizdams. Skaičius iš esamos paletės arba įvardyta spalva.

`col.axis`
`col.lab`
`col.main`
`col.sub`
	Spalva, naudojama atitinkamai ašių anotacijai, *x* ir *y* etiketėms, pagrindiniams pavadinimams ir antraštėms.

`font=2`	Sveikasis skaičius, nurodantis, kokį šriftą naudoti tekstui. Jei įmanoma, įrenginių tvarkyklės išdėstomos taip, kad `1` atitiktų paprastą tekstą, `2` – paryškintą tekstą, `3` – tekstą kursyvu, `4` – paryškintą tekstą kursyvu ir `5` – simbolio šriftą (kuriame yra graikiškos raidės).

`font.axis`
`font.lab`
`font.main`
`font.sub`
	Šriftas, kuris naudojamas atitinkamai ašių anotacijai, *x* ir *y* etiketėms, pagrindiniams pavadinimams ir antraštėms.

`adj=-0.1`	Teksto lygiuotė atsižvelgiant į brėžinio padėtį. `0` reiškia kairįjį lygiavimą, `1` reiškia dešinįjį lygiavimą ir `0.5` reiškia horizontalųjį centravimą ties braižymo padėtimi. Reikšmė `-0.1` palieka 10% teksto pločio tarpą tarp teksto ir brėžinio padėties.
`cex=1.5`	Simbolio išplėtimas. Reikšmė yra norimas teksto simbolių dydis (įskaitant brėžinių simbolius), palyginti su numatytuoju teksto dydžiu.

`cex.axis`
`cex.lab`
`cex.main`
`cex.sub`
	Simbolio išplėtimas, kuris naudojamas atitinkamai ašių anotacijai, *x* ir *y* etiketėms, pagrindiniams pavadinimams ir antraštėms.

12.5.2 Ašys ir padalos žymės

Daugelis „R“ paketo aukšto lygio brėžinių turi ašis, bei taip pat, ašis galite susikurti patys, naudodamiesi žemo lygio grafikos funkcija `axis()`. Ašys susideda iš trijų pagrindinių komponentų: ašies linijos (linijos stilius kontroliuojamas grafikos parametru `lty`), padalos žymės (kurios pažymi vieneto padalijimą išilgai ašies linijos) ir padalos etiketės (kurie žymi vienetus.) Šiuos komponentus galima pritaikyti naudojant sekančius grafikos parametrus (10 lentelė).

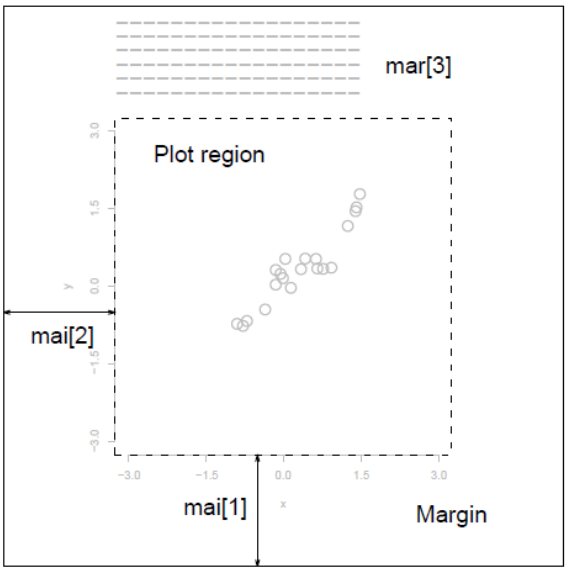
10 lentelė. Ašims taikomi grafikos parametrai

`lab=c(5, 7, 12)`	Pirmieji du skaičiai yra pageidaujamas pažymėtų intervalų skaičius atitinkamai *x* ir *y* ašyse. Trečiasis skaičius yra norimas ašių etikečių ilgis, išreikštas simboliais (įskaitant dešimtainį tašką.) Pasirinkus per mažą šio parametro vertę, visos varnelės etiketės gali būti suapvalintos iki to paties skaičiaus!
`las=1`	Ašių etikečių orientacija. `0` reiškia visada lygiagrečią ašiai, `1` reiškia visada horizontalią, `2` reiškia visada statmeną ašiai.
`mgp=c(3, 1, 0)`	Ašies komponentų padėtis. Pirmasis komponentas yra atstumas nuo ašies etiketės iki ašies padėties teksto eilutėse. Antrasis komponentas yra atstumas iki pažymėtų etikečių, o galutinis komponentas yra atstumas nuo ašies padėties iki ašies linijos (paprastai lygus nuliui). Teigiami skaičiai matuojami už brėžinio srities, neigiami skaičiai – viduje.
`tck=0.01`	Padalos žymių ilgis, kaip brėžimo srities dydžio trupmena. Kai `tck` yra mažas (mažesnis nei `0.5`), *x* ir *y* ašių padalų žymės turi būti tokio paties dydžio. Kai reikšmė yra 1, tai suteikia tinklelio linijas. Neigiamos reikšmės suteikia padalos žymes už brėžinio srities ribų.

	Naudokite `tck=0.01` ir `mgp=c(1,-1.5,0)` vidinėms padalos žymėms.
`xaxs="r"` `yaxs="i"`	Ašių stiliai atitinkamai *x* ir *y* ašims. Naudojant stilius „`i`“ (vidinis) ir „`r`“ (numatytasis), padalos žymės visada patenka į duomenų diapazoną, tačiau stilius „`r`“ kraštuose palieka nedaug vietos.

12.5.3 Paveikslų paraštės

Vienas brėžinys, esantis pakete „R“, yra žinomas kaip paveikslas ir apima brėžinių plotą, apsuptą paraštėmis (gali būti ašių etiketės, pavadinimai ir t.t.) ir (paprastai) apribotas pačių ašių (5 pav.).



5 pav. Paveikslo pavyzdys

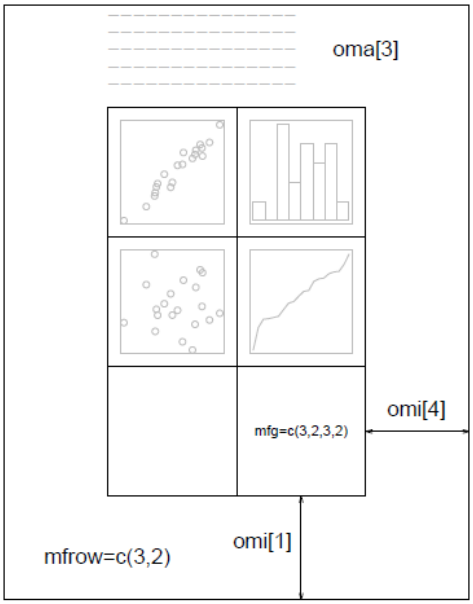
Grafikos parametrus, kontroliuojančius figūros išdėstymą, sudaro komanda `mai=c(1, 0.5, 0.5, 0)`, kur apatinės, kairiosios, viršutinės ir dešinės paraštės pločiai, matuojami coliais. Komanda `mar=c(4, 2, 2, 1)` yra panaši į `mai`, išskyrus, kad matavimo vienetas yra teksto eilutės.

`mar` ir `mai` yra lygiaverčiai ta prasme, kad nustatant vieną keičiama kito reikšmė. Numatytosios šio parametro reikšmės dažnai būna per didelės. Dešinės pusės paraštė retai reikalinga, o viršutinės paraštės nėra, jei nenaudojamas joks pavadinimas. Apatinė ir kairioji paraštės turi būti pakankamai didelės, kad tilptų ašis ir pažymėtų etiketes. Be to, numatytasis pasirinkimas pasirenkamas neatsižvelgiant į priemonės paviršiaus dydį. Pavyzdžiui, naudojant

`postscript()` tvarkyklę su argumentu `height=4` argument bus gautas brėžinys, kurio paraštės yra apie 50%, nebent `mar` arba `mai` nėra aiškiai nustatyti. Kai naudojamos kelios figūros (žr. žemiau), paraštės yra sumažintos, tačiau to gali nepakakti, kai daug figūrų tame pačiame puslapyje.

12.5.4 Kelių figūrų aplinka

„R“ leidžia viename puslapyje susikurti n × m figūrų masyvą. Kiekviena figūra turi savo paraštes, o figūrų masyvas yra pasirinktinai apsuptas išorine paraštė, kaip parodyta 6 paveiksle.



6 pav. Figūrų masyvo pavyzdys

Su keliomis figūromis susiję grafiniai parametrai yra šie:

11 lentelė. Kelių figūrų masyvo nustatymas

`mfcol=c(3, 2)` `mfrow=c(2, 4)`	Nustatykite kelių figūrų masyvo dydį. Pirmoji reikšmė yra eilučių skaičius; antrasis yra stulpelių skaičius. Vienintelis skirtumas tarp šių dviejų parametrų yra tas, kad nustačius `mfcol`, skaičiai turi būti užpildomi stulpeliais; `mfrow` – užpildomi eilutėmis. Išdėstymas galėjo būti sukurtas nustatant `mfrow=c(3,2)`. Nustačius bet kurį iš jų, galima sumažinti bazinį simbolių ir teksto dydį (kontroliuojamas `par("cex")`). Išdėstyme, kuriame yra tiksliai dvi eilutės ir stulpeliai, bazinis dydis

	sumažinamas koeficientu 0.83. Jei yra trys ar daugiau eilučių ar stulpelių, mažinimo koeficientas yra 0.66.
`mfg=c(2, 2, 3, 2)`	Dabartinės figūros padėtis kelių figūrų aplinkoje. Pirmieji du skaičiai yra esamo paveikslo eilutė ir stulpelis. Paskutinieji du skaičiai yra kelių figūrų masyvo eilučių ir stulpelių skaičius. Nustatykite šį parametrą, jei norite peršokti tarp figūrų masyve. Jūs netgi galite naudoti skirtingas paskutinių dviejų skaičių reikšmes, nei tikrąsias nevienodo dydžio figūras tame pačiame puslapyje.
`fig=c(4, 9, 1, 4)/10`	Dabartinės figūros padėtis puslapyje. Reikšmės yra kairiojo, dešiniojo, apatinio ir viršutinio kraštų padėtys, išreikštos procentais nuo puslapio, matuojamo iš apatinio kairiojo kampo. Pavyzdinė reikšmė būtų figūra, esanti puslapio apačioje dešinėje. Nustatykite šį parametrą norėdami patys išdėstyti figūras puslapyje. Jei norite pridėti figūrą dabartiniame puslapyje, naudokite `new=TRUE`.
`oma=c(2, 0, 3, 0)` `omi=c(0, 0, 0.8, 0)`	Išorinių paraščių dydis. Kaip `mar` ir `mai`, pirmasis matuojamas teksto eilutėmis, o antrasis – coliais, pradedant nuo apatinės paraštės ir dirbant pagal laikrodžio rodyklę.

Išorinės paraštės yra ypač naudingos puslapių pavadinimams ir kt. Tekstas gali būti pridėtas prie išorinių paraščių su funkcija `mtext()` ir argumentu `outer=TRUE`. Pagal numatytuosius nustatymus išorinių paraščių nėra, todėl jūs turite jas aiškiai sukurti naudodami `oma` arba `omi`.

Sudėtingesnis kelių figūrų išdėstymas gali būti sukurtas pasinaudojus funkcijomis `split.screen()` ir `layout()`, taip pat pasinaudojus paketais *grid* ir *lattice* (https://CRAN.R-project.org/ package=lattice) .