Programiranje v R-ju
6. Okolja, datoteke, Monte Carlo

Vladimir Batagelj

Univerza v Ljubljani, FMF, Matematika

Finančna matematika
Ljubljana, februar 2009
17. december 2012

Kazalo

Okolja (environment)

Okolje sestavlja zbirka poimenovanih objektov in podatek o nadrejenem (starševskem) okolju. Nadrejeno okolje izvemo s funkcijo `parent.env()`.

R-ovski seji ustreza *osnovno* ali *globalno* okolje `.GlobalEnv`, v katerem so shranjeni vsi objekti, ki jih ustvarimo s prireditvenimi stavki. Dodatna, nadrejena okolja ustvarijo posamezni paketi/knjižnice, ki jih vključimo s funkcijo `require(lib)` ali `library(lib)`.

Vsak klic funkcije ustvari svoje lokalno okolje v okolju, v katerem je bil klic narejen.

Tolmač pri iskanju imen išče, začenši v tekočem okolju, kjer je ime uporabljeno, po verigi nadrejenih okolij. Dobimo je s funkcijo `search()`. Kje se ime nahaja pa izvemo s funkcijo `find(ime)`.

Kadar želimo uporabiti objekt z imenom *ime* iz paketa *pak* uporabimo ime oblike *pak*`::`*ime*.

Pri pripravi paketov lahko ustrezno iskalno zaporedje po uporabljenih paketih zagotovimo z imenskimi prostori (`NAMESPACE`) paketov.

. . . okolja

Okolja lahko ustvarimo tudi sami s funkcijo `new.env()`. Za delo z okolji imamo na voljo naslednje funkcije:
`assign(x,v,envir=e)` – shrani vrednost *v* v okolje *e* pod imenom *x*
`get(x,envir=e)` – vrne objekt z imenom *x* iz okolja *e*
`exists(x,envir=e)` – preveri ali v okolju *e* obstaja objekt z imenom *x*
`remove(list=imena,envir=e)` – odstrani *imena* iz okolja *e*
V funkcijah `get` in `exists` moramo dodati parameter `inherits=FALSE`, če želimo omejiti iskanje samo na okolje *e*.
Seznam objektov v okolju *e* dobimo s funkcijo `ls(e)` ali `objects(e)`.
V povezavi z okolji imamo na voljo posebno obliko prireditvenega stavka
ime <<- *izraz*
ki poišče *ime* v tekočem okolju in mu priredi vrednost *izraza*. Če ga ne najde, ustvari nov objekt s tem imenom v osnovnem okolju. Ta oblika stavka omogoča preživetje izbranih količin uporabljenih v funkcijah.

Okolja – primeri


```
> e1 <- new.env()
> parent.env(e1)
<environment: R_GlobalEnv>
> globalenv()
<environment: R_GlobalEnv>
> e2 <- new.env(parent=e1)
> assign("a",3,envir=e1)
> bla <- function(){a <<- 7}
> find("a")
character(0)
> get("a",envir=e2)
[1] 3
> bla()
> a
[1] 7
```

```
> find("a")
[1] ".GlobalEnv"
> objects(e1)
[1] "a"
> objects(globalenv())
[1] "a"   "bla" "e1"  "e2"
> get("a",envir=e2)
[1] 3
> mget("a", e1, ifnotfound = 0)
$a
[1] 3
> mget("b", e1, ifnotfound = 0)
$b
[1] 0
```

Okolja lahko v R-ju uporabimo tudi kot slovarsko podatkovno strukturo (dictionary), ki nam omogoča učinkovito delo s podatki oblike (ključ, vrednost). Pri tem iskanje omejimo na to okolje z `inherits=FALSE`.

Okolja – primeri


```
> stevec
Error: object "stevec" not found
> stej <- function(){
  if (!exists("stevec")) stevec <<- 0;
  stevec <<- stevec+1; stevec}
> stej()
[1] 1
> stej()
[1] 2
> stej()
[1] 3
> stej()
[1] 4
> stevec
[1] 4
> stej()
[1] 5
> stevec
[1] 5
```

Branje in izpis

Branje in izpis lahko poleg na datotekah uporabljamo tudi na drugih tokovih zlogov. Vrsto toka opišemo s funkcijami `file`, `gzfile`, `bzfile`, `url`, `socket`, ...
Tok moramo pred uporabo odpreti s funkcijo `open(tok,določila)`. Določila so niz znakov sestavljen iz treh delov ("r","w","a")(,"+")(,"t","b"). Tako na primer določilo "w+b" pomeni, da nameravamo na dvojiško datoteko pisati in z nje tudi brati. Tok zapremo (sprostimo) s funkcijo `close(tok)`.
Za branje so na voljo funkcije `readline`, `readLines`, `scan`, `load`, `source`, `readChar`, `readBin`, `read.table`, ...
Podobno so za pisanje so na voljo funkcije `print`, `cat`, `save`, `dump`, `write`, `writeChar`, `writeBin`, `write.table`, ...
`flush`, `seek`, ...

Branje in izpis – datoteke


```
naprej <- function() {
  repeat {
    odg <- readline("Naprej (Da/Ne)? ")
    if (substr(tolower(odg),1,1) == "d") {cat("Nadaljujmo!\n"); break}
  }
}
> naprej()
Naprej (Da/Ne)? n
Naprej (Da/Ne)?
Naprej (Da/Ne)? g
Naprej (Da/Ne)? D
Nadaljujmo!
niz <-
("n,skupina,spol,s1,s2,s3,s4
1,1,ž,1,1,5,1\n2,2,ž,2,1,4,1\n3,1,ž,2,2,4,3\n4,2,ž,3,1, ,3
5,1,m,4,5,2,4\n6,2,m,5,4,5,5\n7,1,m,5,3,4,4\n8,2,m,4,5,5,5")
> t <- read.table(textConnection(niz),header=TRUE,sep=",",row.names="n")
> t
  skupina spol s1 s2 s3 s4
1       1    ž  1  1  5  1
2       2    ž  2  1  4  1
3       1    ž  2  2  4  3
4       2    ž  3  1 NA  3
...
8       2    m  4  5  5  5
> c <- textConnection(niz)
> s <- read.table(c,header=TRUE,sep=",",row.names="n",colClasses=
+ c("integer","integer","character","NULL","NULL","integer","integer"))
> s
  skupina spol s3 s4
1       1    ž  5  1
2       2    ž  4  1
...
```

Regularni izrazi

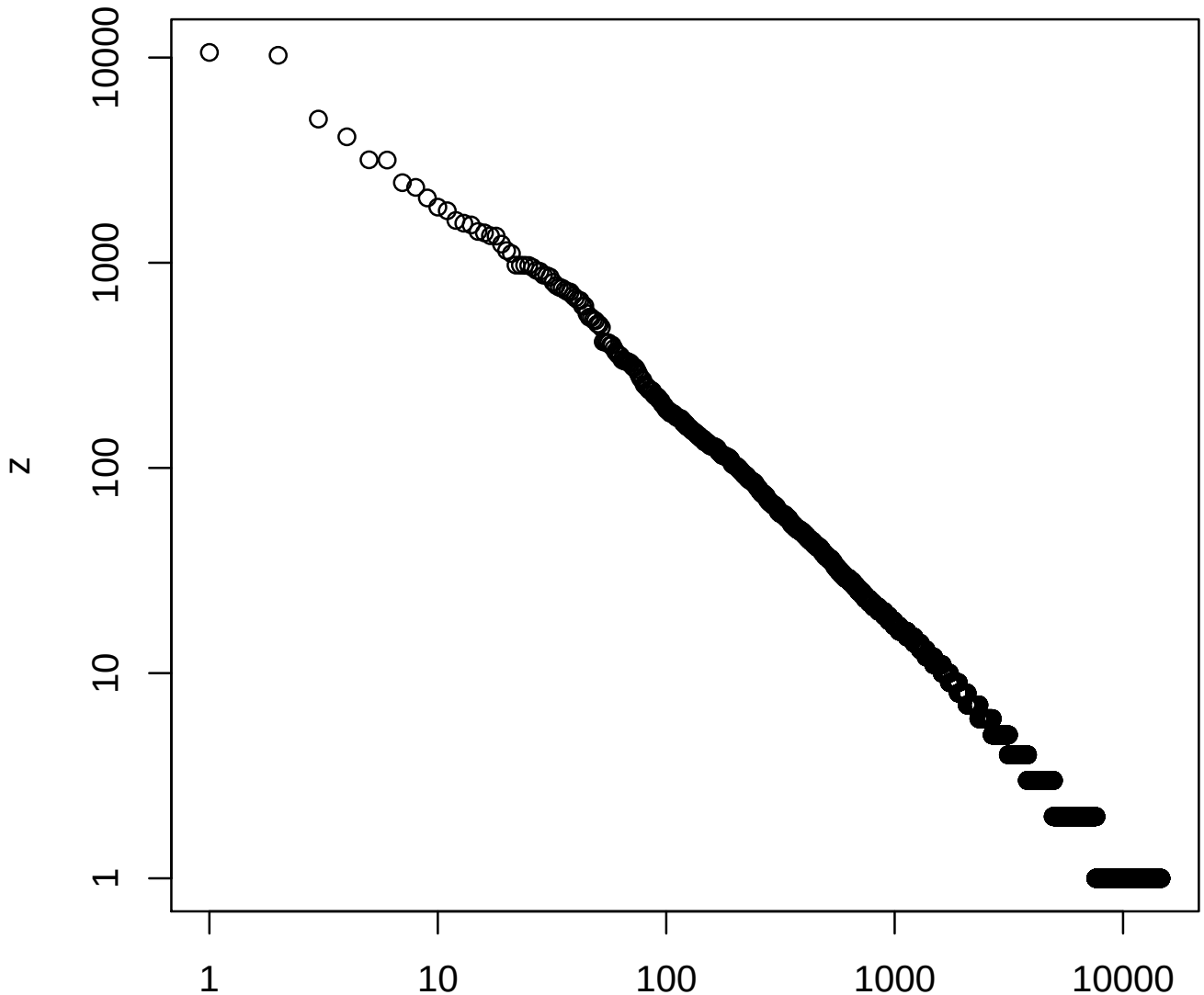
R podpira tudi delo z *regularnimi izrazi* – vzorci, s katerimi opišemo množice nizev. Za podrobnosti glejte `help(regex)`.
Regularni izraz *reg* lahko uporabimo v naslenjih funkcijah:
`grep(`*reg*`,`*v*`)`, `sub(`*reg*`,`*niz*`,`*v*`)`, `gsub(`*reg*`,`*niz*`,`*v*`)`, `regexpr(`*reg*`,`*v*`)`, `gregexpr(`*reg*`,`*v*`)`, `strsplit(`*v*`,`*reg*`)`, `glob2rx(`*fpat*`)`, ...
Tako v naslednjem primeru regularni izraz `"[[:punct:]]+|[[:space:]]+"` določa množico vseh nizov, ki so sestavljeni iz vsaj enega ločila ali vsaj enega 'belega' znaka.

1001 noč – štetje besed v besedilu

Na straneh projekta Gutenberg je mogoče najti čez 10 tisoč prosto dostopnih del (knjig), pretežno v angleščini. Izbrali smo prvi del "1001 noči", ga prebrali po vrsticah, odstanili 'ovitek', prešteli pogostost besed, jih uredili v padajočem vrstnem redu in narisali v dvojno logaritemski lestvici – dobimo skoraj premico (Zipfov zakon).

```
> page <- "http://www.gutenberg.org/dirs/etext02/11001108.txt"
> stran <- readLines(con<-url(page)); close(con)
> i <- grep("\\*\\*\\* START OF THIS PROJECT GUTENBERG EBOOK",stran,ignore.case=TRUE)
> i
[1] 41
> length(stran)
[1] 17634
> j <- grep("End of the Project Gutenberg EBook",stran,ignore.case=TRUE)[1]
> j
[1] 17239
> knjiga <- stran[(i+1):(j-1)]
> separator <- "[[:punct:]]+|[[:space:]]+"
> cleni <- unlist(strsplit(knjiga,separator))
> besede <- tolower(cleni[nchar(cleni)>0])
> length(besede)
[1] 178136
> t <- table(besede)
> z <- rev(sort(t))
> z[1:10]
besede
  the   and    of    to     a     i    in    he    my   his
10596 10259  5015  4111  3176  3167  2457  2330  2069  1867
> length(z)
[1] 14624
> plot(z,log="xy")
```

1001 noč – Zipfov zakon



Metoda Monte Carlo

Metode *Monte Carlo* je skupno ime za vse metode, ki pri reševanju problemov uporabljajo slučajnost.
Zamisel, da bi pri reševanju problema uporabili slučajnost je razmeroma stara. Tako je na primer že v 18. stoletju francoski prirodoslovec Buffon opisal, kako z metanjem igle oceniti število π. Svojo pravo vrednost pa je metoda Monte Carlo pokazala šele po letu 1943. Tega leta sta jo J. von Neumann in S. Ulam s sodelavci na Los Alamos Scientific Laboratories prvič s pridom uporabila pri reševanju nekih fizikalnih problemov pri izgradnji atomske bombe, ki jih niso uspeli analitično dognati. S tem je končno zapustila ropotarnico kratkočasne (zabavne) matematike.

Uveljavitev metode Monte Carlo je tesno povezana s pojavitvijo računalnikov, saj so šele ti zagotovili potrebne računske zmogljivosti.

. . . metoda Monte Carlo

Pri postopkih Monte Carlo običajno določimo približke za količine, ki nas zanimajo, tako da 'poskus' večkrat ponovimo pri različnih slučajnih pogojih. Kot približek za posamezno količino vzamemo povprečje vrednosti te količine v posameznih poskusih. O natančnosti približka izvemo iz standardnega odklona vrednosti.
Teoretično osnovo tega pristopa si sposodimo iz verjetnostnega računa. To sta izrek Kolmogorova (krepki zakon velikih števil):

> *Za to, da bi povprečje neodvisnih izvedb (realizacij) slučajne spremenljivke X konvergiralo z verjetnostjo 1 (gotovostjo) k njeni pričakovani vrednosti je potrebno in zadostno, da pričakovana vrednost obstaja.*

in centralni limitni izrek:

$$P(\frac{1}{n}\sum_{k=1}^{n}(x_k - \mathrm{E}X) < \frac{t\mathrm{D}X}{\sqrt{n}}) \stackrel{n\to\infty}{\longrightarrow} \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{t} e^{-\frac{z^2}{2}}\,dz$$

Metoda Monte Carlo – Zakon kvadratnega korena

Centralni limitni izrek omogoča oceno napake: z dano verjetnostjo p velja

$$|\mathrm{E}X - \frac{1}{n}\sum_{k=1}^{n} x_k| \leq \frac{\gamma \mathrm{D}X}{\sqrt{n}}$$

kjer je γ odvisna od p-ja.
Torej je, če povzamemo,

$$|\mathrm{E}X - \overline{X}| \leq \frac{c}{\sqrt{n}}$$

kjer je c neka od problema in izbrane verjetnosti p odvisna konstanta. Tej zvezi pravimo *zakon kvadratnega korena*. Če obstajajo končni momenti, je mogoče ocene napake izpeljati tudi za probleme, kjer realizacije niso neodvisne.

Metoda Monte Carlo – Posledice zakona kvadratnega korena

Iz zakona kvadratnega korena vidimo, da se z večanjem števila poskusov približek $\mu(n) = \overline{X}$ poljubno približa vrednosti μ. Toda, kako hitro? Poglejmo, koliko poskusov moramo narediti, da bomo izboljšali rezultat za eno decimalko! V ta namen naj bo ε napaka in n število poskusov pri slabši oceni; N pa število poskusov pri izboljšani oceni. Po zakonu kvadratnega korena je:

$$\varepsilon \approx \frac{c}{\sqrt{n}} \qquad \text{in} \qquad \frac{\varepsilon}{10} \approx \frac{c}{\sqrt{N}} \qquad \text{oziroma} \qquad N \approx 100n$$

kar pomeni, da moramo za vsako novo decimalko vložiti 100 krat toliko dela kot za prejšnje. Zakon kvadratnega korena je potemtakem hkrati temeljni kamen metode Monte Carlo in njena osnovna omejitev, ki ne dopušča, da bi 'izostrili' približke za več kot 2 ali 3 mesta.

Metode Monte Carlo so zelo močno orodje saj večkrat omogočajo dobiti zadovoljive ocene tudi za probleme, ki jim naše znanje (še) ni doraslo.

Metoda Monte Carlo – Sandokan

Uporabimo metodo Monte Carlo še pri iskanju odgovora na naslednje vprašanje:
V vsakem zavitku čokolade se nahaja ena slikica. Za to da bi napolnili album moramo zbrati n različnih slikic. Koliko čokolad moramo v povprečju kupiti, da bo album poln?
Zamisel postopka Monte Carlo je preprosta: Slučajno kupujemo slikice in jih štejemo, dokler ne zberemo vseh različnih. To večkrat ponovimo. Povprečje kupljenih slikic v posameznem poskusu je iskani odgovor.

Sandokan MP3, Kabir Bedi, besedilo in slike

Metoda Monte Carlo – Sandokan – teoretična rešitev

V program je za primerjavo vgrajen tudi teoretični odgovor. Označimo z $N(n, m)$ pričakovano število slikic, ki jih moramo še kupiti, da bomo napolnili album za n slikic, če nam manjka še m slikic. Velja naslednja rekurzivna zveza

$$N(n, m) = \frac{n}{m} + N(n, m-1) \qquad \text{in} \qquad N(n, 0) = 0$$

iz katere dobimo

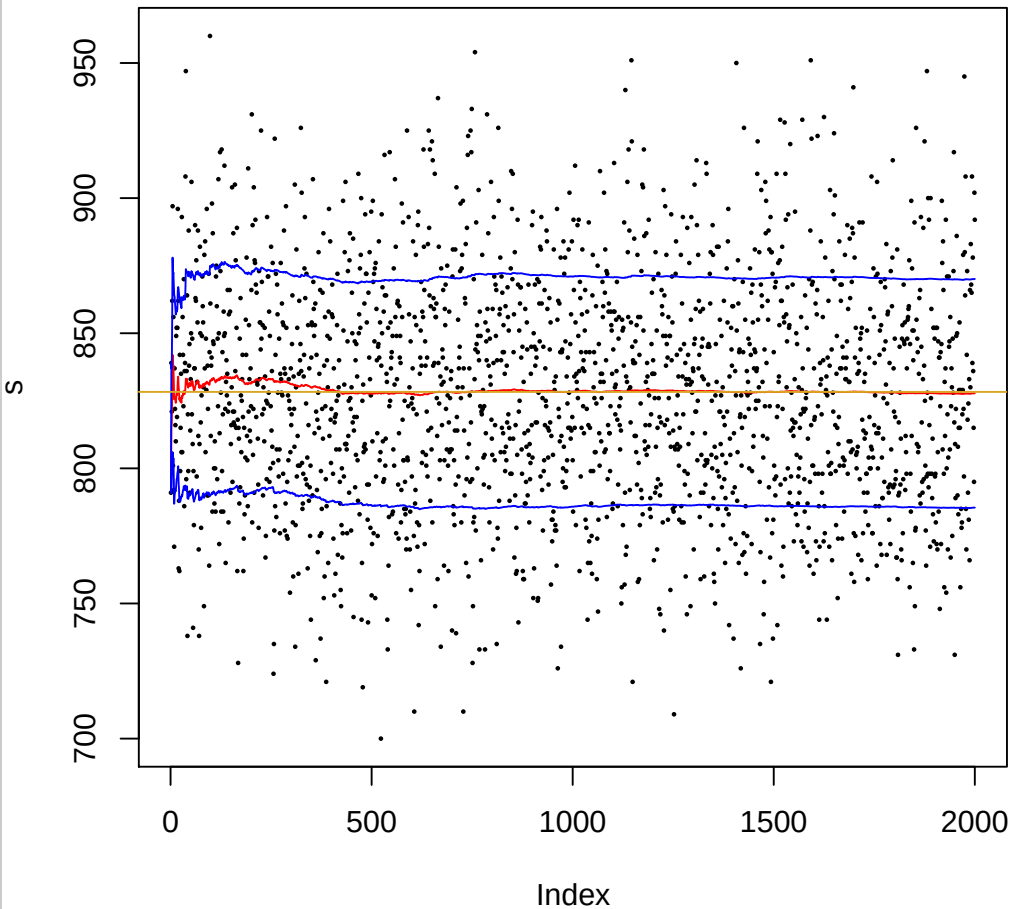
$$N(n, m) = n.(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m}) \approx n.(\ln(m + \frac{1}{2}) + \gamma)$$

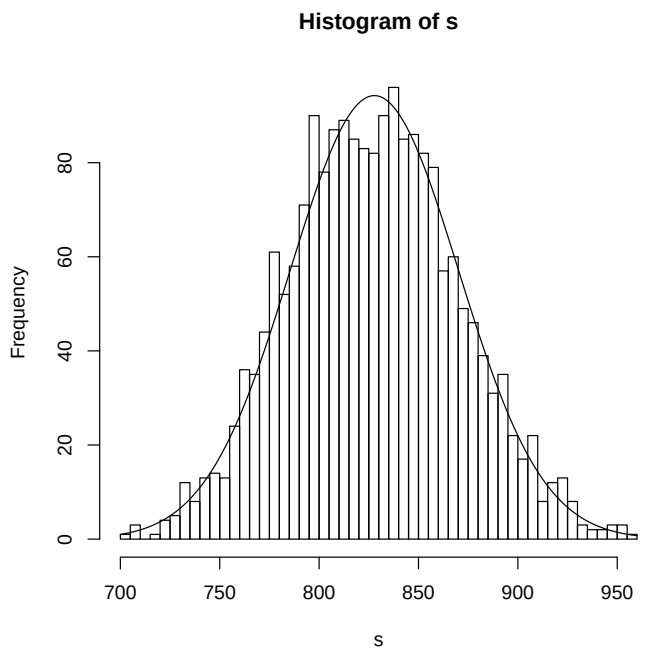
kjer je $\gamma = 0.57721\,56649$ Eulerjeva konstanta.
Program omogoča tudi ocenjevanje povprečnega števila kupljenih slikic za to, da bi zbrali m različnih slikic. Teoretični odgovor na to vprašanje je $N(n, n) - N(n, n-m)$.

Metoda Monte Carlo – . . . Sandokan v Rju


```
dice <- function(n) return(1+trunc(n*runif(1)))
sandokanT <- function(n,m=n) return(n*sum(1/((n-m+1):n)))
slika1 <- function(t,s,...){
  n <- length(s)
  sredina <- cumsum(s)/1:n
  odklon <- sqrt(cumsum(s^2)/1:n - sredina^2)
  plot(s,pch=16,cex=0.2,...)
  lines(1:n,sredina+odklon,col="blue"); lines(1:n,sredina-odklon,col="blue")
  lines(1:n,sredina,col="red"); abline(h=t,col="goldenrod")
}
slika2 <- function(s,breaks=50,...){
  n <- length(s); q <- min(s):max(s); t <- 5*n*dnorm(q,mean(s),sqrt(var(s)))
  hist(s,breaks=breaks,...); lines(q,t,col="red",lw=2)
}
poskus <- function(n,m){
  album <- rep(TRUE,n); razlicnih <- 0; slikic <-0
  while (razlicnih < m) {
    k <- dice(n); slikic <- slikic + 1
    if (album[k]) {album[k] <- FALSE; razlicnih <- razlicnih + 1}
  }
  return(slikic)
}
sandokan <- function(n,m=n,r=100){
  s <- replicate(r,poskus(n,m))
  slika1(sandokanT(400,350),s,ylim=c(750,900))
  odg <- readline("Naprej? "); slika2(s)
}
sandokan(400,350,2000)
```

Metoda Monte Carlo – . . . Sandokan / slika





Kaj manjka?

- Objektno programiranje – S3 in S4.
- Priprava lastnih paketov, dokumentacija.
- Knjižnice/paketi za področje financ. Zgleda, da je najcelovitejša podpora za finančne analize zbrana v paketih zbirke `Rmetrics` (wiki). Poleg tega pa utegneje biti zanimivi še paketi `backtest`, `FinTS`, `financial`, `quantmod`, `tseries`, `portfolio`, `portfolioSim`, `tawny`, ... poglejte še na R/finance. Za tiste, ki poznate SAS ali SPSS, bo zanimiva primerjava z R.