Wprowadzenie do języka R

Opracowanie: *Ewelina Bednarz*

Wstęp

R-project jest pakietem matematycznym przeznaczonym do zaawansowanych obliczeń statystycznych. Objęty jest licencja GPL, a zatem jest programem całkowicie bezpłatnym i wolnym. Możliwości pakietu R są stosunkowo duże i nie kończą się na samych tylko obliczeniach. Umożliwia on również tworzenie wykresów o bardzo dobrej jakości. Mogą być one zapisywane w formatach EPS, PDF i WMF, co pozwala na łatwe włączenie ich do publikacji naukowych. Jego funkcjonalność uzupełniają dodatkowe biblioteki do konkretnych zastosowań, dostarczane wraz z obszerna dokumentacja.

Program można pobrać ze strony domowej, znajdującej się pod adresem http://www.r-project.org. Znajdują się tam zarówno źródła programu, jak i gotowe skompilowane pakiety z przeznaczeniem na konkretny system operacyjny (w przypadku Windows wraz z instalatorem).

Źródła:

- Ł.Komsta, Wprowadzenie do środowiska R
- http://www.r-project.org

Zaczynamy…

Znak > jest zachętą do wprowadzenia polecenia. Wszystkie operacje w programie R dokonywane są przez wprowadzanie poleceń. Istnieje możliwość otrzymania pomocy na temat określonej funkcji poprzez napisanie jej nazwy poprzedzonej znakiem zapytania, np. ?sin. Możemy również przeszukiwać wszystkie pliki pomocy pod katem wystąpienia określonego słowa, np. help.search("mean").

R może być wykorzystywany jako stosunkowo precyzyjny kalkulator. Wystarczy wprowadzić zadane wyrażenie, a otrzymamy jego wynik.

#Przykłady:

```
1+1
sqrt(30)
log(100) # logarytm naturalny
log(100,10)  # logarytm dziesiętny
pi
sin(30*pi/180) # argumenty funkcji trygonometrycznych podaje się w radianach
1/(2*sqrt(45*89))
16^(0.5)
```

R umożliwia organizację danych w różne struktury, a najprostszą z nich jest wektor. W języku R nie ma prostszych struktur; zatem pojedyncza liczba jest wektorem o długości równej 1. Jednak jedynka w nawiasie nie oznacza długości wektora, lecz numer pierwszej pozycji, która podana jest w danym wierszu. Wektor może zawierać liczby, ciągi znaków (np. podpisy do danych), jak również dane logiczne (prawda lub fałsz).
Na wektorach można wykonywać przeróżne operacje. Ich rezultatem jest również wektor, którego wszystkie dane zostały w odpowiedni sposób przeliczone. Utworzymy wektor zawierający 10 liczb i zapamiętamy go w zmiennej dane. Następnie będziemy starali się wykonać podstawowe operacje statystyczne na tych wynikach.

```
dane = c(10.34,10.87,10.32,9.64,10.39,9.48,10.55,9.36,9.67,10.58)
# funkcja c() łączy podane argumenty w wektor

dane
summary(dane)
mean(dane)
shapiro.test(dane) #test normalności
# wysoka wartość p, więc nie ma podstaw do odrzucenia hipotezy o normalności
#rozkładu tych danych

var(dane) #wariancja
sd(dane) #odchylenie standardowe
```

Operacje na wektorach:

```
1/dane #odwrotności wszystkich liczb w wektorze
sin(dane)  #sinusy wszystkich liczb w wektorze
dane^2 # wszystkie liczby w wektorze do kwadratu

t.test(dane,mu=9)#test t-Studenta dla jednej średniej, równej 9. Wartość p przemawia za
#odrzuceniem hipotezy, że wyniki pochodzą z populacji o takiej średniej.

length(dane)

# listę wszystkich zmiennych przechowywanych w środowisku uzyskuje
# się poleceniem ls()
```

Typowym zadaniem może być dopasowanie przykładowych danych (np. krzywej kalibracyjnej) do modelu liniowego i kwadratowego.
Na początek umieszczamy poszczególne wartości *x* i *y* w odpowiednich wektorach. Następnie zmiennym lin i sq przypisujemy dopasowanie tych danych do modelu liniowego lub kwadratowego. Po wyświetleniu podsumowania (*summary*) każdego z wyników, otrzymujemy tabelę zawierającą obszerne wyniki dopasowania - wartości R2, reszty regresji, estymatory poszczególnych współczynników, ich błędy standardowe, jak również wartość *t* oraz odpowiadającą im wartość *p* dla istotności tych współczynników.

Najprostsza regresja liniowa:

```
#dopasowanie przykładowych danych do modelu liniowego i kwadratowego:
x = c(1,2,3,4,5,6)
y = c(101,204,297,407,500,610)

lin = lm(y~x)  #dopasowanie do modelu liniowego
summary(lin)

sq = lm(y~x+I(x^2)) #dopasowanie do modelu kwadratowego
summary(sq)
```

Funkcja summary wywołana na tym obiekcie przedstawia kolejno wartości reszt (lub, w przypadku większej ich liczby, wartości skrajne, medianę i kwartyle), estymatory nachylenia prostej (slope) i przecięcia z osią y (intercept). Dla każdego z estymatorów podany jest błąd standardowy oraz odpowiadające mu wartości t i p dla jego istotności. Podano również wartości testu F na istotność samej korelacji pomiędzy zmiennymi (hipoteza H0: wszystkie współczynniki przy zmiennych są równe 0 – a_1=a_2=...=a_k=0)

Wektory c.d.

```
1:30
30:1 # sekwencja malejąca
n = 10
1:n-1 # dwukropek ma priorytet, zatem otrzymamy 1:10, pomniejszone o 1
1:(n-1) # a teraz sekwencja 1:9
seq(along=dane) # sekwencja od 1 do długości zmiennej dane
rep(1:5,5) # powtarzamy 1:5 pięć razy
rep(1:5,each=5) # powtarzamy każdy z elementów 5 razy
rep(1:5,length.out=43) # wektor o długości 43, składający się z powtórzeń 1:5

a = seq(-1,1,length=10) # generujemy sekwencje
a
a[2] # drugi element
a[-2] # wszystkie oprócz drugiego
a[c(1,5)] # pierwszy i piaty
a[-c(1,5)] # wszystkie oprócz nich
a > 0 # wektor logiczny - które większe od zera
a[a>0] # ten sam wektor jako indeks, czyli wypisze te liczby
a == 1 # inny wektor logiczny - uwaga na podwójną równość

a = seq(-1,2,length=15)
a
mean(a) # średnia z danych
a/mean(a) # każdą liczbę podziel przez ich średnią
a*c(1,2) # co drugą liczbę pomnóż przez 2 - ostrzeżenie że wektor dłuższy nie jest
#wielokrotnością krótszego

b = a * c(1,2) # co drugi element mnożymy przez 2
b
a-b # różnice między elementami
b[c(1,3,5)] = c(10,11,12) # wstawiamy 10,11,12 na 1,3 oraz 5 pozycje
b
a[1:5] = 0 # zerujemy 5 pierwszych elementów
a
a[8] = b[8] # 8 element wektora a jest równy 8 elementowi b
a
a = 1/a # wektor a zawiera odwrotności dotychczasowych wartości
a
# Inf oznacza nieskończoność
# NaN oznacza wartość nieokreślona
```

```
a=c(6,2,4,8,2,6,8,9,3,5,6)
b=c(2,8,0,4,3,6,1,5,4,8,4)
sum(a)  # suma elementów a
sum(a>3) # ile elementów jest większych od 3?
sum(a[a>3]) # zsumujmy te elementy!
pmin(a,b)  # wartości minimalne
pmax(a,b)  # wartości maksymalne
length(a)
c=c(a,b)  # a teraz łączymy wektory!
c
sort(c)  # sortowanie
which(c>3) # które elementy są większe niż 3?
range(c) # jaki jest zakres (min i max?)
cummin(c) # najmniejsza wartość dotychczasowa
cummax(c) # największa wartość dotychczasowa
diff(c) # różnice miedzy kolejnymi wartościami
```

Faktor (factor) - specjalna struktura, przechowująca (oprócz szeregu danych) informacje o powtórzeniach takich samych wartości oraz o zbiorze unikalnych wartości tego ciągu.

```
ankieta=c("T","N","T","T","X","N","T","X","N","T","N","T","T","N"
,"X","N","T","N","T","T")
factor(ankieta)
table(ankieta) # ile jakich odpowiedzi? zestawienie
inne=c(1,2,3,2,3,4,5,2,4,32,4,8,9,76,5,6,5,6,5,8,7,6)
sort(inne)
factor(inne)
levels(factor(inne))  # jakie unikalne wartości?

ankieta2=rev(ankieta)  # zestawienie dwuwymiarowe
table(ankieta, ankieta2)

inne
table(inne)

wiek=c(20,23,45,21,67,34,52,31,59,38,19,44,64,18,40,50,32,31,18,2
0)
length(wiek)
wiek2 = cut(wiek,c(0,20,40,50,100))
wiek2
table(wiek2) # ile osób w każdym przedziale?
```

Tablice

```
tbl=1:20
dim(tbl)=c(4,5) # wektor staje się tablicą o wymiarach 4,5
tbl
tbl[1]
tbl[12] # cały czas można się odwoływać do poszczególnych elementów!
tbl[,1] # pierwsza kolumna
tbl[1,] # pierwszy rząd
tbl[2,2]# drugi rząd i druga kolumna

dim(tbl) # wyświetlamy wymiary
dim(tbl) = c(2,10) # zmieniamy wymiary!
tbl
tbl[,6] # a teraz szósta kolumna

i = array(c(1:4,4:1),dim=c(4,2)) # tworzymy tablice współrzędnych
i
```

Funkcja outer tworzy z dwóch tablic znacznie większą tablice wielowymiarowa. Wymiary tej tablicy są połączeniem wektorów wymiarów dwóch tablic, zaś jej zawartość stanowią wszystkie możliwe kombinacje iloczynów (lub innych operacji) pomiędzy elementami.

```
outer(1:10,1:10) # tabliczka mnożenia

outer(1:10,1:10,"/") # operator jako trzeci argument- „tabliczka dzielenia”
# argumentem outer może być też nazwa funkcji operującej na dwóch zmiennych.

tbl
aperm(tbl,c(2,1))#przegrupowanie tabeli-drugi argument to permutacja wymiarów tabeli
#c(2,1) oznacza, ze drugi wymiar staje się pierwszym, a pierwszy drugim
# w tym przypadku tworzy transpozycje tabeli
```

Funkcje cbind i rbind formują tablice z podanych wektorów, poprzez umieszczenie ich rzędami lub kolumnami w nowo tworzonej tabeli. W ten sposób można serie danych łatwo scalić w tabele. Odwrotnie, każdą tabele można przekształcić na wektor funkcją as.vector.

```
cbind(1:5,6:10,11:15) # kolumnami
rbind(1:5,6:10,11:15) #wierszami
as.vector(tbl)
```

Lista (list) jest uporządkowanym zbiorem elementów różnego typu. Do jej utworzenia służy polecenie list. Wywołane z listą liczb, tworzy listę jednoelementowych wektorów

```
s = list(1,2)
s  # element listy (indeks) w podwójnych nawiasach kwadratowych
s =
list(wektor=c(1,2,3),srednia=2,tablica=array(data=c(1,2,3,4),dim=
c(2,2)))
s
s$wektor # odniesienie do elementu "wektor"
s$wektor[1]  # odniesienie do pierwszego elementu tej zmiennej
s[[1]][1]  # to samo, lecz z wykorzystaniem indeksu
s$tablica[,1] # pierwsza kolumna elementu "tablica"
```

Pakiet R pozwala na łatwe ładowanie danych z zewnętrznych plików.
Przykładowy plik nazywa się „tabela.txt” i ma postać:
Lp Wiek Odpowiedz
1 18 T
2 27 N
3 41 N
4 19 N
5 68 T
Aby tę tabelę wczytać, należy ustawić katalog roboczy na katalog, w którym zapisany jest ten plik, a następnie wykonać poniższą instrukcję.

```
#opcja header - plik tekstowy zawiera w pierwszym wierszu nazwy kolumn
dane = read.table("tabela.txt",header=TRUE)
dane
dane$Lp
dane[[1]]
dane[,2]
dane[2,] # jeden z wierszy

attach(dane) # "przyłączanie" – zmienne istnieją w środowisku "bezpośrednio”
Lp
Wiek
Odpowiedz
detach(dane) # "rozłączenie"
Wiek  # po odłączeniu zmienna nie jest dostępna bezpośrednio,
dane$Wiek # tylko pośrednio
```

Pakiet R posiada wbudowane algorytmy pozwalające na obliczanie gęstości, dystrybuanty i kwantyli najczęściej stosowanych rozkładów. Może również pracować jako precyzyjny generator liczb losowych. Standardowo dostępne są następujące rozkłady: beta, binom, cauchy, chisq, exp, f, gamma, geom, hyper, lnorm, logis, nbinom, norm, pois, t, unif, weibull, wilcox.

Poprzedzając nazwę rozkładu litera d uzyskujemy funkcje gęstości rozkładu. Analogicznie poprzedzając nazwę litera p uzyskujemy wartości dystrybuanty. Generator liczb losowych dostępny jest przy poprzedzeniu nazwy litera r. Funkcje te pozwalają na traktowanie pakietu R jako zestawu bardzo dokładnych tablic statystycznych.

#Przykłady:
```
dnorm(0) # gęstość rozkładu normalnego w zerze
rnorm(30,50,5)  # generator liczb losowych z danego rozkładu

#funkcja sample generuje wektor danych wylosowanych z innego wektora.
sample(1:6,10,replace=T) # symuluje 10 rzutów kostka (losowanie ze zbioru 1:6)
```

Wykresy

Wykresy słupkowe - polecenie barplot
```
#par(mfrow=c(2,1)) # polecenie dzielące okno wykresów na dwie części (dwa wiersze)

tv = abs(round(rnorm(100,4,1.5)))
komp = abs(round(rnorm(100,6,2)))
tv
komp

barplot(tv) #najprostszy wykres 100 słupków odpowiadających danym
barplot(sort(tv)) #wykres posortowanych danych, pozwalający na wizualną ocenę

barplot(table(tv,tv)) #ilość osób oglądających telewizje godzinę, dwie etc.
barplot(table(komp,komp),col=rainbow(10),legend.text=T)
#taki sam wykres dla czasu spędzonego przed komp.
#rainbow - lista kolorów

#inne funkcje tworzące kolory: heat.colors, terrain.colors, topo.colors, cm.colors.
#Wektory szarości uzyskujemy przez funkcję grey, np. grey(0:10/10) to
#10 kolorów od szarego do białego. Można też wyspecyfikować swoją listę kolorów,
#np. col=c("red","orange","yellow","green","blue",violet").
#Pełna lista nazw kolorów wyświetlana jest komendą colors().
```

Wykresy kołowe - komenda pie, z analogiczną składnią.

```
pie(table(tv)) #można precyzować kolory, podpisy poszczególnych wycinków etc.
```

Histogramy

```
x=rbinom(10000, 500, 1/2)
hist(x)
hist(x, breaks=40, probability=T) #narzucamy liczbę „przedziałów” histogramu.
x=sort(x)
lines(x, dnorm(x,mean(x), sd(x)))
```

#Przykład – błądzenie losowe - (yn0, yn1), yn0=cumsum(y), yn1=cumsum(x)

```
y=runif(500)
y=round(y)
y=2*y-1
x=runif(500)
x=round(x)
x=2*x-1
y0=cumsum(y)
x0=cumsum(x)
plot(x0, y0,col=0)
for(i in 2:500){
    lines(x0[i-1:i], y0[i-1:i])
}
```

Test normalności

```
x=rnorm(500)
qqnorm(x)
```
#wykres kwantylowo-normalny - zależność miedzy wartościami zmiennej
#a kwantylami rozkładu normalnego. W idealnym przypadku, jeśli rozkład jest
#czysto normalny, wykres ten przedstawia linię prostą

```
shapiro.test(x)
```

```
qqnorm(runif(500,0,1))
```
#wykres dla 500 liczb, równomiernie rozmieszczonych
#w przedziale (0, 1).

Wykresy kombinowane

#funkcja curve - rysuje zadane funkcje matematyczne

```
curve(pnorm(x),xlim=c(-4,4));curve(pt(x,1),lty=2,add=T)
```
#rysuje wykres dystrybuanty rozkładu normalnego linią ciągłą, a następnie
#dodaje dystrybuantę rozkładu t-Studenta o 1 stopniu swobody.

```
hist(rnorm(500),prob=TRUE);curve(dnorm(x),add=T)
```
#rysuje histogram 500 losowych liczb o rozkładzie normalnym wraz z „idealną”
#krzywą rozkładu.

```
curve(sin(x),xlim=c(-2*pi,2*pi));
curve(cos(x),col="red",add=T,xlim=c(-pi,pi))
```
#rysuje wykres funkcji y = sin x w przedziale (-2pi, 2pi), a następnie dodaje
#wykres y = cos x czerwona linia, w przedziale (-pi, pi).

#Wszystkie funkcje rysujące mają bogaty zestaw wspólnych parametrów, których
#dokumentacja jest dostępna przez ?par.

Najczęściej stosowane **testy statystyczne**:

- test t-Studenta dla jednej średniej (sprawdzający, czy wyniki pochodzą z populacji o danej średniej),
- test Shapiro-Wilka na rozkład normalny (sprawdzający, czy próba pochodzi z populacji o rozkładzie normalnym)
- przy wnioskowaniu dla jednej średniej, w razie stwierdzenia rozkładu innego niż normalny, zamiast testu t stosuje się test rang WILCOXONA

#Przykład:
```
dane = c(96.19,98.07,103.53,99.81,101.60,103.44)
shapiro.test(dane)  # czy rozkład jest normalny? - duże p więc tak
mean(100)
t.test(dane,mu=100) # tak, a zatem sprawdzamy testem Studenta
wilcox.test(dane, mu=100)  # tak byśmy sprawdzali, gdyby nie był normalny
```

#Przykład: **metoda AKAIKE** – nowsza metoda sprawdzająca istotność zmiennych
#należy dołączyć pakiet MASS

#AKAIKE – znajduje model prawdziwy o najmniejszym wymiarze
```
n=10
x1=rnorm(n);
x2=rnorm(n);
x3=rnorm(n);
x4=rnorm(n);
y=3+0.1*x1+2*x2+0.3*x3+0.5*rnorm(n);

wynik=lm(y~x1+x2+x3+x4);
summary(wynik);

stepAIC(wynik);
```