Krótkie wprowadzenie do R
dla programistów, z elementami statystyki opisowej

Mikołaj Rybiński
WMIM UW

9 stycznia 2009

Spis treści

Wstęp

R jest językiem programowania, a przede wszystkim środowiskiem do obliczeń statystycznych oraz wizualizacji wyników. W tym zakresie posiada liczne gotowe implementacje procedur statystycznych oraz dostosowane do nich, bardzo duże możliwości graficzne. W połączeniu z elastycznością programistyczną daje to daje to potężne środowisko do obliczeń naukowo–statystycznych.

Kod źródłowy R objęty jest licencją GNU GPL, tzn. jest to darmowe narzędzie. Prekompilowane binarne wersje środowiska R dostępne są m.in. dla systemów UNIX'owych, Windows i Mac OS. Można je pobrać ze strony projektu: http://cran.r-project.org/bin/.

Uwaga 0.0.1

Ten skrypt jest napisany z użyciem wersji R pod Linuxa, jednak praktycznie wszystkie zawarte w nim informacje są niezależna od wyboru systemu operacyjnego.

Uwaga [Linux] 0.0.2

We wszystkich popularnych dystrybucjach Linuxa (Ubuntu, Debian, Gentoo, SUSE, Red Hat etc.) środowisko R dostępne jest w wybranym repozytorium pakietów.

Rozdział 1

Podstawy R

Środowisko R używa wiersza poleceń. Dostępne są również graficzne środowiska programistyczne (zob. część 3.1).

Uwaga [Linux] 1.0.3

Konsolę uruchamia się z linii poleceń komendą R.

1.1 R jako kalkulator

Środowiska R możemy używać jak zwykłego kalkulatora wpisując wyrażenie do obliczenia i potwierdzając komendę klawiszem Enter.

Przykład 1.1.1

```
> 2 + 2
[1] 4
> 3%%2
[1] 1
> sqrt(2)
[1] 1.414214
> log(4)
[1] 1.386294
> log(4, 2)
[1] 2
> pi
[1] 3.141593
> sin(pi/2)
[1] 1
> pi^2/6
```

```
[1] 1.644934

> exp(1)

[1] 2.718282
```

Na wyjście wypisywania jest domyślnie wartość ostatniego wyrażenia. Jedynka w nawiasach kwadratowych przy każdym wynik bierze się z faktu, że w R pojedyncze liczby są również wektorami. Oznacza ona indeks w wektorze. W przypadku wielolinijkowych poleceń, znak zachęty > zmienia się na znak kontynuacji polecenia +.

Uwaga 1.1.2

Żeby dowiedzieć się więcej o działaniu i parametrach używanych funkcji lub operatorów wystarczy użyć unarnego operatora ? (np. ?"%%", ?log) lub użyć funkcji help. [1]

```
> help("%%")
> help("log")
```

Cudzysłowia są potrzebne w przypadku operatorów, słów kluczowych języka (zob. część 1.5) etc.

Wyniki obliczeń możemy przypisywać na zmienne przy pomocy operatora =. Porównanie wykonujemy przy pomocy operatora ==.

Przykład 1.1.3

```
> a = "kaka"
> ls()

[1] "a"

> a

[1] "kaka"

> a == print(a)

[1] "kaka"
[1] TRUE

> .a = a
> ls()

[1] "a"

> rm(a)
> ls()

character(0)

> .a

[1] "kaka"

> help(ls)
> help(print)
> help(rm)
> help("==")
```

Według konwencji nazwy zmiennych zaczynają sie od liter lub kropki. W drugim przypadku przyjmuje się, że oznacza to zmienne prywatne/pomocnicze, dlatego też np. `ls` nie wypisuje takich zmiennych. Kropki używa się również standardowo jako spearatora w nazwach zmiennych. Dodatkowo, znaczenie ma wielkość liter.

Uwaga 1.1.4

`R` ma charakter języka funkcyjnego, czego konsekwencją jest m.in. fakt, że zawsze zwracana jest jakaś wartość obliczanego wyrażenia. W przykładzie powyżej funkcja `print` działa jak funkcja identycznościowa, której efektem ubocznym jest wypisanie wartości parametru. Dodatkowo wynik funkcji `print` jest zwracany przy pomocy funkcji `invisible` co powoduje, że nie jest on ponownie wypisywany.

```
> "teraz mnie widac"

[1] "teraz mnie widac"

> invisible("a teraz nie")
> "widac"

[1] "widac"

> invisible("nie widac")
```

Uwaga 1.1.5

Ze względu na starsze wersje środowiska `R` dozwolony jest również dwuznakowy operator przypisania `<-`.

Uwaga [Linux] 1.1.6

Podobnie jak w konsoli działają strzałki (góra/dół) oraz wyszukiwanie wstecz `Ctrl+R`.

Kod `R` możemy zapisywać w ulubionym notatniku. Zwyczajowo takie pliki mają rozszerzenie `.r` lub `.R`. Aby wykonać zawarte w pliku polecenia wywołujemy polecenie `source`, którego parametrem jest ścieżka do pliku.

Przykład 1.1.7 Wykonywanie poleceń z pliku

```
> getwd()

[1] "/home/trybik/work/rps/rguide"

> source("./r-skrypt.R")
```

Komentarze w kodzie umieszcze się za znakiem hash `#`.

1.2 Atomowe typy danych

Liczbowe

Wszystkie liczby rzeczywiste w `R` są przechowywane jako typy `double` (podwójnej precyzji) dlatego liczby będą dla nas zazwyczaj po prostu liczbami (`numeric`). Czasami jednak jest potrzeba i można rozróżnić liczby całkowite (`integer`), rzeczywiste (`double`) czy nawet zespolone (`complex`).

Przykład 1.2.1

```
> x = 44
```

```
> y = as.integer(x)
> typeof(x)

[1] "double"

> typeof(y)

[1] "integer"

> is.double(x)

[1] TRUE

> is.integer(x)

[1] FALSE

> is.double(y)

[1] FALSE

> is.integer(y)

[1] TRUE
```

Przykład 1.2.2 Liczby zespolone

```
> sqrt(-1)

[1] NaN

> sqrt(-1 + (0+0i))

[1] 0+1i
```

Wartość `NaN` jest specjalną wartością oznaczającą "nie liczbę". Specjalnymi wartościami liczbowymi są nieskończoności (`Inf`, `-Inf`).

Przykład 1.2.3 Overflow

```
> 10^308

[1] 1e+308

> 10^308 + 10^308

[1] Inf

> help(Inf)

> Inf + Inf

[1] Inf

> Inf - Inf

[1] NaN
```

Wartości brakujące i puste

Pojawiająca się w poprzednich przykładach wartość "nie liczba" `NaN` jest szczególnym przypadkiem wartości brakującej `NA`. Dodatkowo, rozróżnia się wartość niezdefiniowaną (pustą) `NULL`.

Przykład 1.2.4

```
> x = c(0, NULL, NA, NaN)
> x

[1]   0  NA NaN

> is.na(x)

[1] FALSE  TRUE  TRUE

> is.nan(x)

[1] FALSE FALSE  TRUE

> is.null(x)

[1] FALSE

> as.double(NULL)

numeric(0)
```

Uwaga 1.2.5

Wiele funkcji posiada parametr–flagę `na.rm`, mówiący czy wpierw usunąć brakujące wartości.

```
> sum(1, NA, 3)

[1] NA

> sum(1, NA, 3, na.rm = TRUE)

[1] 4
```

Logiczne

Wartości logiczne to odpowiednio wyróżnione wartości `TRUE`, `FALSE`, ale też `NA`. Można je również otrzymać z użyciem operatorów relacyjnych `<`, `<=`, `>`, `>=`, `==`, `!=` oraz logicznych `&`, `|` i `!` (negacja). Spośród wartości liczbowych `0` reprezentuje wartość `FALSE` w wyrażeniach logicznych, pozostałe liczby reprezentują wartość `TRUE`.

Przykład 1.2.6

```
> -3%%3 == 0

[1] TRUE

> NA == 0

[1] NA

> x = pi^2/6
> x > 1 & x < 2
```

```
[1] TRUE

> as.logical(0)

[1] FALSE

> pi & exp(1)

[1] TRUE
```

Znakowe

Wartości znakowe (`character`) to napisy wprowadzane w cudzysłowiach lub apostrofach. O technicznych detalach (m.in. tzw. escapowaniu) można się dowiedzieć z pomocy:

```
> cat("\t+\n\t-\n")

        +
        -

> help(Quotes)
```

Podstawową funkcją sklejającą napisy jest funkcja `paste`, przyjmująca dowolną liczbę argumentów (dodatkowo jest wektorowa – zob. część 1.3)).

Przykład 1.2.7

```
> paste("2 + 2", "=", 2 + 2)

[1] "2 + 2 = 4"

> e = expression(2^20)
> paste(as.character(e), eval(e), sep = "=")

[1] "2^20=1048576"
```

Uwaga 1.2.8

Użyte w poprzednim przykładzie funkcje `expression` oraz `eval` pozwalają na zdefiniowanie obiektu wyrażenia oraz jego późniejsze wyliczenie (ewaluację) [2].

Ważniejsze funkcje napisowe (nazwy samowyjaśnialne):

```
> help(nchar)
> help(paste)
> help(strsplit)
> help(substr)
```

1.3 Struktury danych

Poza typami atomowymi `R` posiada "atomowe struktur danych", przy pomocy których reprezentowane są wszystkie obiekty w środowisku.

```
> help(typeof)
```

Najważniejszą strukturą atomową jest wektor — `R` jest językiem wektorowym; wszystkie możliwe operacje odbywają się wektorowo po elementach. Wszystkie elementy wektora muszą być tego samego atomowego typu (liczbowe, znakowe, logiczne). Atomową strukturą danych reprezentującą niejednorodne ze względu na typ elementów wektory są listy.

Dodatkowo, `R` posiada prosty mechanizm tworzenia własnych obiektów (tzw. model klas S3).

```
> help(class)
```

Ze względu na statystyczny charakter języka, najważniejszymi strukturami danych "obiektowymi" są tablice (atomowo wektory), tablice jednorodnych kolumn zwane ramkami (atomowo listy) oraz wektory różnych wartościach elementów innych wektorów (klas) zwane faktorami (atomowo wektory liczb naturalnych).

Przykład 1.3.1

```
> class(matrix(1))

[1] "matrix"

> typeof(matrix(1))

[1] "double"

> class(data.frame())

[1] "data.frame"

> typeof(data.frame())

[1] "list"

> class(factor())

[1] "factor"

> typeof(factor())

[1] "integer"
```

Wektory

Standardowymi funkcjami do tworzenia własnych wektorów jest konkatenacja `c` oraz sekwencja `seq` (z pomocniczym operatorem `:`).

Przykład 1.3.2

```
> c(2, 3, c(4, 5))

[1] 2 3 4 5

> seq(2, 5)

[1] 2 3 4 5

> 2:5

[1] 2 3 4 5
```

```
> seq(2, 5, by = 0.5)

[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> seq(2, 5, length = 3)

[1] 2.0 3.5 5.0
```

Uwaga 1.3.3

W środowisku R funkcje mają zazwyczaj duuużo parametrów, z których większość ma domyślne wartości. W przypadku braku nazwy, parametry są czytane w kolejności podanej w nagłówku funkcji. Standardową składnią przypisania możemy przekazywać wartośći wybranym parametrom w dowolnej kolejności (zob. część 1.5).

```
> help(seq)
```

R jest językiem wektorowym i operacje arytmetyczne oraz większość funkcji operuje na wektorach po kolejnych elementach.

Przykład 1.3.4

```
> v = 2^(1:5)
> v

[1]  2  4  8 16 32

> v + v

[1]  4  8 16 32 64

> v * v

[1]    4   16   64  256 1024

> sqrt(v)

[1] 1.414214 2.000000 2.828427 4.000000 5.656854

> 1/v

[1] 0.50000 0.25000 0.12500 0.06250 0.03125

> (1:5)^(1:5)

[1]    1    4   27  256 3125

> v * c(1, 2)

[1]  2  8  8 32 32
```

Uwaga 1.3.5

Dla wektorów o różnej długości obowiązuje reguła cyklicznego przetwarzania krótszego z wektorów.

Krótki opis ważniejszych funkcji wektorowych zawiera tabela 1.1. Przykłady użycia poniżej.

Funkcja	Opis
`c`	Konkatenacja wektorów
`paste`	Sklejanie napisów
`seq`	Generowanie sekwencji liczb
`rep/rep.int`	Generowanie sekwencji potwarzającego się wektora
`rev`	Odwrócenie wektora
`length`	Długość wektora
`sort`	Zwraca wektor posortowany (rosnąco lub malejąco)
`order`	Zwraca wektor kolejności (rang) elementów wektora (według rosnącego lub malejącego porządku; remisy rozstrzygane wg kolejnych parametrów–wektorów);
`rank`	Zwraca wektor rang elementów wektora (rosnąco; remisy rozstrzygane wg wybranej startegii)
`max/min`	Wartość maksymalna/minimalna w wektorze
`range`	Zakres wartości: min i max.
`pmax/pmin`	Maksimum/minimum po pozycjach (wektorowe)
`sum/prod`	Suma/iloczyn elementów wektora
`diff`	Różnica sąsiednich elementów wektora
`cummax/cummin`	Maximum/minimum do dotychczasowej pozycji (kumulacyjne)
`cumsum/cumprod`	Sumy/iloczyny częściowe (kumulacyjne)
Podstawowe miary i funkcje statystyczne	
`mean`	Średnia arytmetyczna
`var/sd`	Wariancja i odchylenie standardowe
`cov/cor`	Kowariancja i korelacja
`median`	Mediana
`quantile`	Kwantyl
`summary`	Podsumowanie: min, max, średnia, kwantyle .25, .5, .75.
`ecdf`	Funkcja dystrybuanty empirycznej
`sample`	Próbkowanie z wartości wektora

Tabela 1.1: Ważniejsze funkcje wektorowe

Przykład 1.3.6 Ważniejsze funkcje wektorowe

```
> s = c("o", "e")
> paste(s, 1:8, sep = "")

[1] "o1" "e2" "o3" "e4" "o5" "e6" "o7" "e8"

> .x = c(rep.int(1, 5), rep.int(2, 3), 3)
> .x

[1] 1 1 1 1 1 2 2 2 3

> x = c(.x, rev(.x))
> x

 [1] 1 1 1 1 1 2 2 2 3 3 2 2 2 1 1 1 1 1

> length(x)

[1] 18

> sort(x)

 [1] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3

> order(x)

 [1]  1  2  3  4  5 14 15 16 17 18  6  7  8 11 12 13  9 10

> x[order(x)] == sort(x)

 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE

> rank(x, ties.method = "min")

 [1]  1  1  1  1  1 11 11 11 17 17 11 11 11  1  1  1  1  1

> max(x)

[1] 3

> min(x)

[1] 1

> pmax(.x, rev(.x))

[1] 3 2 2 2 1 2 2 2 3

> sum(x)

[1] 28

> prod(x)

[1] 576

> diff(x)
```

```
 [1]  0  0  0  0  1  0  0  1  0 -1  0  0 -1  0  0  0  0

> cummax(x)

 [1] 1 1 1 1 1 2 2 2 3 3 3 3 3 3 3 3 3

> cumsum(x)

 [1]  1  2  3  4  5  7  9 11 14 17 19 21 23 24 25 26 27 28

> mean(x)

[1] 1.555556

> var(x)

[1] 0.496732

> sd(x) == sqrt(var(x))

[1] TRUE

> median(x)

[1] 1

> quantile(x, 0.5)

50%
  1

> summary(x)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   1.556   2.000   3.000

> ecdf(x)(c(0.99, 1, 2, 44))

[1] 0.0000000 0.5555556 0.8888889 1.0000000

> sample(x, 10, replace = TRUE)

 [1] 1 1 2 1 1 1 1 2 3 1

> cor(sort(sample(x, 10^2, replace = TRUE)), sort(sample(x, 10^2,
+     replace = TRUE)))

[1] 0.9425308
```

Indeksowanie wektorów

Indeksowanie wektorów jest realizowane poprzez nawiasy kwadratowe `[...]`. Naturalnym sposobem indeksowanie jest zastosowanie wektorów logicznych. W takim przpadku gdy wektor indeksowany i logiczny wybranych elementów są różnej długości to obowiązuje reguła cyklicznego przetwarzania.

Przykład 1.3.7

```
> x = -5:5
> x > 0

 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> x[x > 0]

[1] 1 2 3 4 5

> x[c(TRUE, FALSE)]

[1] -5 -3 -1  1  3  5
```

Uwaga 1.3.8

Dla indeksów spoza zakresu długości wektora zwracana jest wartość brakująca `NA`.

```
> (1:2)[rep(TRUE, 3)]

[1]  1  2 NA
```

Indeksowanie może odbywać się również standardowo poprzez podanie interesujących nas numerów pozycji (jako wektor) lub poprzez dopełnienie — podanie nieinteresujących nas numerów pozycji.

Przykład 1.3.9

```
> x = -5:5
> x[6]

[1] 0

> n = length(x)
> x[c(1, n)]

[1] -5  5

> x[-c(1, n)]

[1] -4 -3 -2 -1  0  1  2  3  4

> x[(n%/%2):n]

[1] -1  0  1  2  3  4  5
```

Przy pomocy funkcji `match` i `which` możemy pozyskać numery pozycji, odpowiednio zadanych elementów i elementów spełniających dany warunek logiczny.

Przykład 1.3.10

```
> x = -5:5
> match(c(10, 0), x)

[1] NA  6

> which(x > 0)
```

```
[1]  7  8  9 10 11
```

Istnieje również możliwość indeksowania wektorów po nazwach jeżeli takie zostały wcześniej elementom wektora nadane (`names`).

Przykład 1.3.11

```
> x = -12:0
> names(x) = letters[1:length(x)]
> x

  a   b   c   d   e   f   g   h   i   j   k   l   m
-12 -11 -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0

> x[c("k", "a", "k", "a")]

  k   a   k   a
 -2 -12  -2 -12
```

Tablice

Tablice to wektory z dodatkową informacją o wymiarach (funkcja `dim`) i ewentualnie o ich nazwach (ogólna funkcja `dimnames` lub dla macierzy funkcje `rownames` i `colnames`). Obie wymienione informacje to dodatkowe atrybuty wektora, które możemy nadać przy pomocy funkcji typu `nazwa.atrybutu(...)=`.

Przykład 1.3.12

```
> x = 1:8
> dim(x) = c(2, 4)
> x

     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8

> dimnames(x) = list(letters[1:2], LETTERS[1:4])
> x

  A B C D
a 1 3 5 7
b 2 4 6 8

> attributes(x)

$dim
[1] 2 4

$dimnames
$dimnames[[1]]
[1] "a" "b"

$dimnames[[2]]
[1] "A" "B" "C" "D"
```

Innym sposobem tworzenia tablic jest użycie funkcji–konstruktora `matrix` (dla dwuwymiarowych tablic zwanych macierzami) lub `array` (ogólny).

Przykład 1.3.13

```
> dn = list(letters[1:2], LETTERS[1:4])
> matrix(1:8, 2, 4, dimnames = dn)

  A B C D
a 1 3 5 7
b 2 4 6 8

> array(1:8, c(2, 4), dn)

  A B C D
a 1 3 5 7
b 2 4 6 8
```

Macierze możemy również tworzyć poprzez kolumnowe lub wierszowe łączenie wektorów lub macierzy, odpowiednio przy pomocy funkcji `cbind` i `rbind`.

Przykład 1.3.14

```
> rbind(1:3, 4:6)

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

> cbind(1:3, 4:6)

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

> cbind(rbind(1:3, 4:6), c(7, 8))

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    7
[2,]    4    5    6    8
```

Tablicę możemy indeksować po wymiarach `x[i1, i2, ...]` lub tak jak wektor (w kolejności `x[1,1,...]`, `x[2,1,...]`, `...`). Nie podanie wartości przy którymś z wymiarów odpowiada wybraniu całego.

Przykład 1.3.15

```
> x = array(1:8, c(2, 4))
> x[2, 1]

[1] 2

> x[5]

[1] 5

> x[2, ]
```

```
[1] 2 4 6 8

> x[, 3:4]

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Uwaga 1.3.16

Środowisko R przy indeksowania stara się upraszczać tabele do wektorów, kiedy to tylko jest możliwe. Problem pojawia się przy wybieraniu zmiennej liczby kolumn, dla zmiennej o wartości 1. Aby uniknąć uproszczenia do wektora należy użyć atrybutu `drop` funkcji indeksującej.

```
> x = array(1:8, c(2, 4))
> k = 1:3
> x[, k]

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

> k = 1
> x[, k]

[1] 1 2

> x[, k, drop = FALSE]

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

Tablice są wektorami i standardowe operatory arytmetyczne działają tak jak dla wektorów, tzn. po elementach. Ciekawsze funkcje operujące na tablicach zawiera tabela 1.2.

Funkcja	Opis
`%*%`	Standardowy operator mnożenia tablic (zgodnych wymiarów)
`outer`	Produkt zewnętrzny tablic (“każdy z każdym”)
`t`	Transpozycja macierzy
`aperm`	Dowolna permutacja wymiarów tablicy
`diag`	Diagonala (przekątna) macierzy
`solve`	Rozwiązanie układu równań liniowych
`qr/svd/chol`	Rozkłady macierzy (QR, wartości osobliwych, Choleskiego)

Tabela 1.2: Ciekawsze funkcje tablicowe

Przykład 1.3.17 Ciekawsze funkcje tablicowe

```
> a = array(1:12, c(3, 4))
> a

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
```

```
[3,]    3    6    9   12
```

```
> a * a
```

```
     [,1] [,2] [,3] [,4]
[1,]    1   16   49  100
[2,]    4   25   64  121
[3,]    9   36   81  144
```

```
> a %*% t(a)
```

```
     [,1] [,2] [,3]
[1,]  166  188  210
[2,]  188  214  240
[3,]  210  240  270
```

```
> diag(a)
```

```
[1] 1 5 9
```

```
> dim(a) = c(2, 3, 2)
> aperm(a, c(2, 1, 3))
```

```
, , 1

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

, , 2

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

```
> solve(diag(1:3), rep.int(1, 3))
```

```
[1] 1.0000000 0.5000000 0.3333333
```

```
> solve(diag(1:3))
```

```
     [,1] [,2]      [,3]
[1,]    1  0.0 0.0000000
[2,]    0  0.5 0.0000000
[3,]    0  0.0 0.3333333
```

```
> b = outer(1:3, 1:3)
> b
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
```

```
> solve(diag(1:3), b)
```

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

Faktory

Faktory (`factor`) to wektory z dodatkową informacją o różnych wartościach i liczbie ich powtórzeń (dostępne odpowiednio przez funkcje `levels` i `table`). Faktory służą do reprezentacji tzw. dane kategoryczne, natomiast realizacja odpowiada typom "enum" (Java, C++) i wartości wylistowania są zawsze przechowywane jako kolejne liczby naturalne.

Przykład 1.3.18

```
> pyt1 = c(rep("Mezczyzna", 4), rep("Kobieta", 5), "Hermafrodyta")
> odp1 = sample(pyt1, 10, replace = TRUE)
> x = factor(odp1)
> x

 [1] Mezczyzna    Mezczyzna    Kobieta      Mezczyzna    Kobieta
 [6] Mezczyzna    Mezczyzna    Hermafrodyta Kobieta      Kobieta
Levels: Hermafrodyta Kobieta Mezczyzna

> levels(x)

[1] "Hermafrodyta" "Kobieta"      "Mezczyzna"

> table(x)

x
Hermafrodyta      Kobieta    Mezczyzna
           1            4            5

> typeof(x)

[1] "integer"

> pyt2 = c("Deska", rep("Narty", 2))
> odp2 = sample(pyt2, 10, replace = TRUE)
> table(x, factor(odp2))

x              Deska Narty
  Hermafrodyta     0     1
  Kobieta          1     3
  Mezczyzna        1     4
```

Uwaga 1.3.19

Obiekty typu faktor są przechowywane jako liczby całkowite i wyglądają jak wektory liczbowe, ale nimi nie są i standardowe operacje numeryczne wykonane na faktorach nie zwracają nic sensownego poza ostrzeżeniem.

```
> x = factor(sample(4:8, 10, replace = TRUE))
> x

 [1] 5 6 7 8 5 4 7 4 8 6
Levels: 4 5 6 7 8

> cat("Typ:", typeof(x), "; Klasa:", class(x), ";\n", sep = "")

Typ:integer; Klasa:factor;
```

```
> as.numeric(x)

 [1] 2 3 4 5 2 1 4 1 5 3

> x - 4

 [1] NA NA NA NA NA NA NA NA NA NA
```

Jeżeli porządek poziomów faktoru ma znaczenie można go narzucić tworząc faktor przy pomocy funkcji `ordered`.

Przykład 1.3.20

```
> lev = c("Malo", "Aby aby", "W sam raz")
> ordered(sample(lev, 5, replace = TRUE), levels = lev)

[1] W sam raz Aby aby   Malo      Malo      Malo
Levels: Malo < Aby aby < W sam raz
```

Funkcja `cut` pozwala dzielić dane numeryczne na przedziały (tworzyć z wektorów liczbowych faktory).

Przykład 1.3.21

```
> wiek = sample(1:100, 16, replace = TRUE)
> kat.wiekowe = cut(wiek, c(0, 18, 26, 100))
> kat.wiekowe

 [1] (0,18]   (26,100] (0,18]   (26,100] (26,100] (26,100] (26,100] (26,100]
 [9] (0,18]   (26,100] (26,100] (26,100] (26,100] (26,100] (0,18]   (26,100]
Levels: (0,18] (18,26] (26,100]

> table(kat.wiekowe)

kat.wiekowe
  (0,18]  (18,26] (26,100]
       4        0       12
```

Uwaga 1.3.22

Niezależnie od faktorów dostępne są funkcje pozwalające traktować wektory jako zbiory (wektory o niepowtarzających się wartościach): `unique`, `union`, `intersect`, `setdiff`, `setequal`, `is.element`.

```
> help(unique)
> help(union)
```

Listy

Lista to wektor niejednorodnych elementów. Do tworzenia list służy konstruktor `list`, w którym można od razu podać nazwy elementów listy.

Przykład 1.3.23

```
> x = list(a = 1, 2, c = 3)
```

```
> x

$a
[1] 1

[[2]]
[1] 2

$c
[1] 3

> names(x)

[1] "a" ""  "c"
```

Indeksowanie listy różni się kilkoma rzeczami od indeksowania wektora:

- nazwane elementy można indeksować przy pomocy znaku `$` i początku nazwy jednoznacznie identyfikującym element,
- pojedyncze nawiasy kwadratowe służa do cięcia list i w przypadku pojedynczej wartości dadzą listę jednoelementową,
- w związku z tym aby wybrać element na liście używa się podwójnych nawiasów kwadratowych `[[...]]`.

Przykład 1.3.24

```
> x = list(aaa = 1, 2, aba = 3)
> x$ab

[1] 3

> x[2:3]

[[1]]
[1] 2

$aba
[1] 3

> x[3]

$aba
[1] 3

> x[[3]]

[1] 3
```

Elementy do listy możemy dodawać poprzez standardową konkatenacje albo poprzez przypisywanie wartości na wybrane pozycje.

Przykład 1.3.25

```
> x = list(a = FALSE)
> x$b = letters[1:3]
```

```
> x

$a
[1] FALSE

$b
[1] "a" "b" "c"

> x[2:4] = 2:4
> x[[3]] = 1:8
> x

$a
[1] FALSE

$b
[1] 2

[[3]]
[1] 1 2 3 4 5 6 7 8

[[4]]
[1] 4
```

Przydatną funkcją do oglądania wewnętrznej reprezentacji obiektów w R jest funkcja `str`.

Przykład 1.3.26

```
> x = c(1:3, list(4:10), as.list(11:12))
> x[[3]] = list(TRUE, c = letters[1:3])
> str(x)

List of 6
 $ : int 1
 $ : int 2
 $ :List of 2
  ..$  : logi TRUE
  ..$ c: chr [1:3] "a" "b" "c"
 $ : int [1:7] 4 5 6 7 8 9 10
 $ : int 11
 $ : int 12
```

Dodatkowo, listy z nazwanymi elementami można dołączać do i odłączać od środowiska, tak że elementy są dostępne są *do odczytu* bezpośrednio jako zmienne (o odpowiadających im nazwach).

Przykład 1.3.27

```
> x = list(e1 = 1, e2 = 2)
> attach(x)
> e1

[1] 1

> e1 = NA
> detach(x)
> cat(" x$e1 =", x$e1, "\n e1 =", e1, "\n")
```

```
x$e1 = 1
e1 = NA
```

Uwaga 1.3.28

Dostępne w środowisku zmienne o nazwach takich jak przyłączane elementy przysłaniają te drugie.

```
> e1 = "global"
> x = list(e1 = "local")
> attach(x)

        The following object(s) are masked _by_ .GlobalEnv :

         e1

> e1

[1] "global"

> detach(x)
```

Do wykonywania wyrażeń na środowisku stworzonym tylko i wyłącznie z elementów listy służy funkcja `with`.

Przykład 1.3.29

```
> x = list(a = 1, b = 2)
> with(x, {
+     print(ls())
+     a + b
+ })

[1] "a" "b"
[1] 3
```

Ramki

Przypadkiem szczególnym listy jest ramka (`data.frame`): lista kolumn, być może różnych typów atomowych. Ramki reprezentują macierze o różnych typach kolumn i poza funkcjami działającymi na listach również standardowe, "nie numeryczne" funkcje na macierzach mają sens użyte na ramkach (indeksowanie, nazywanie kolumn/wierszy, bindowanie etc). Innymi słowy: ramka to (kolumnowo niejednorodna) macierz zaimplementowana przy pomocy listy.

Przykład 1.3.30

```
> x = cbind(data.frame(1:3, c(TRUE, TRUE, NA)), letters[1:3])
> dimnames(x) = list(LETTERS[1:3], c("int", "logi", "char"))
> x[["int"]]

[1] 1 2 3

> x$int

[1] 1 2 3
```

```
> x[1]

  int
A   1
B   2
C   3

> x[, 1]

[1] 1 2 3

> x[, 1, drop = FALSE]

  int
A   1
B   2
C   3

> x[2:3]

  logi char
A TRUE    a
B TRUE    b
C   NA    c

> x[, 2:3]

  logi char
A TRUE    a
B TRUE    b
C   NA    c

> x[1:2, 2:3]

  logi char
A TRUE    a
B TRUE    b
```

Uwaga 1.3.31

Przy tworzeniu nowych ramek wektory znakowe konwertowane są na faktory. Aby uniknąć konwersji można użyć funkcji chroniącej tożsamość obiektów `I` [3]. Taki manewr pozwala też np. traktować całe macierze jako pojedyncze kolumny ramki.

```
> x = data.frame(letters[1:3], array(1:6, c(3, 2)))
> str(x)

'data.frame':        3 obs. of  3 variables:
 $ letters.1.3.: Factor w/ 3 levels "a","b","c": 1 2 3
 $ X1          : int  1 2 3
 $ X2          : int  4 5 6

> x[, 2]

[1] 1 2 3

> x = data.frame(I(letters[1:3]), I(array(1:6, c(3, 2))))
> str(x)
```

```
'data.frame':        3 obs. of  2 variables:
 $ letters.1.3.       :Class 'AsIs'  chr [1:3] "a" "b" "c"
 $ array.1.6..c.3..2..: 'AsIs' int [1:3, 1:2] 1 2 3 4 5 6

> x[, 2]

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

Ramki mogą służyć do wczytywania danych z zewnętrznych plików, jako format wejściowy obiektów. Używa się do tego funkcji z rodziny `read.table`. Wczytywane pliki mogą posiadać nagłówki kolumn i wartości rozdzielane wybranymi znakami np. przecinkami (pliki `.csv`) lub tabulatorami. Do wczytywania danych danych do wektora lub listy służy funkcja `scan` (przy czym w tym przypadku można również wczytywać z konsoli).

```
> help(read.table)
> help(scan)
```

Środowisko `R` zawiera liczne zbiory gotowych danych[4]. Do ich wylistowania lub wczytania służy funkcja `data`.

Przykład 1.3.32

```
> help(data)
> data()$results[46:48, ]

     Package    LibPath               Item
[1,] "datasets" "/usr/lib/R/library" "beaver2 (beavers)"
[2,] "datasets" "/usr/lib/R/library" "cars"
[3,] "datasets" "/usr/lib/R/library" "chickwts"
     Title
[1,] "Body Temperature Series of Two Beavers"
[2,] "Speed and Stopping Distances of Cars"
[3,] "Chicken Weights by Feed Type"

> data(beavers)
> help(beavers)
```

Podobnie jak na listach, również na ramkach działają funkcje `attach`, `detach` i `with`.

1.4 Tablice rozkładów prawdopodobieństwa

Środowisko `R` zawiera zestaw dokładnych tablic statystycznych najczęściej stosowanych rozkładów prawdopodobieństwa oraz algorytmy losowania z nich. Są to zestawy czterech funkcji typu `{d|p|q|r}nazwa.rozkladu`, gdzie prefiksy oznaczają odpowiednio:

- `d` - gęstość/f-cja prawdopodobieństwa,
- `p` - dystrybuanta,
- `q` - kwantyle,

[4] Dokładniej pakiet `datasets` zawiera zbiory danych (`?datasets`).

- `r` - próbka z rozkładu.

Dostępne rozkłady prawdopodobieństwa to: `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geom`, `hyper`, `lnorm`, `logis`, `nbinom`, `norm`, `pois`, `t`, `unif`, `weibull`, `wilcox`.

Każda z w/w funkcji przyjmuje jako parametry parametry rozkładu, przy czym mają one zawsze swoje wartości domyślne. Dodatkowo jako pierwszy parametr:

- gęstość oraz dystrybuanta przyjmują wartość z nośnika rozkładu,
- funkcja zwracająca kwantyle przyjmuje wartość prawdopodobieństwa,
- funkcja próbkująca przyjmuje liczbę próbkowanych wartości.

Przykład 1.4.1

```
> dnorm(0, mean = 0, sd = 1) == dnorm(0)

[1] TRUE

> dnorm(0, sd = 2)

[1] 0.1994711

> pnorm(0, sd = 2)

[1] 0.5

> qnorm(0.5, sd = 2)

[1] 0

> x = rnorm(10^2, sd = 2)
> y = rnorm(10^4, sd = 2)
> mean(x)

[1] -0.1912603

> sd(x)

[1] 2.153085

> mean(y)

[1] 0.02439462

> sd(y)

[1] 1.985479
```

1.5 Programowanie w R

Instrukcja warunkowa

Przykład 1.5.1

```
> if (sample(c(TRUE, FALSE), 1)) {
+     print("Galaz TRUE")
```

```
+     TRUE
+ } else {
+     print("Galaz FALSE")
+     FALSE
+ }

[1] "Galaz FALSE"
[1] FALSE
```

Uwaga 1.5.2

Instrukcje grupuje się w bloki nawiasami klamrowymi `{`, `}`.

Instrukcja w warunku musi się wyliczyć do pojedynczej wartości logicznej. Ponieważ standardowe operatory logiczne "i" `&` oraz "lub" `|` działają wektorowo to do tego celu można użyć odpowiednio operatorów `&&` oraz `||` zawsze zwracających jedną wartość. Jeżeli ich argumenty nie są jednoelementowe to brane są pod uwagę tylko pierwsze elementy.

Przykład 1.5.3 Operatory logiczne

```
> c(F, T) & c(T, T)

[1] FALSE  TRUE

> c(F, T) | c(T, T)

[1] TRUE TRUE

> c(F, T) && c(T, T)

[1] FALSE

> c(F, T) || c(T, T)

[1] TRUE

> any(c(F, T))

[1] TRUE

> all(c(F, T))

[1] FALSE
```

Wynik instrukcji warunkowej to wynik ostatniej instrukcji w bloku, który się wykonał.

Wektorową wersją instrukcji warunkowej jest funkcja `ifelse`. Zgodnie z wektorem logicznym podanym jako pierwszy argument wybiera elementy z odpowiednich pozycji wektorów podanych jako dwa kolejne argumenty, odpowiadającym wartościom `TRUE` i `FALSE`

Przykład 1.5.4

```
> pos = rep.int(c(11, 1), 2)
> n = length(pos)
> ifelse(rep(TRUE, n), letters[pos], LETTERS[pos])

[1] "k" "a" "k" "a"

> ifelse(1:n > n%/%2, letters[pos], LETTERS[pos])
```

```
[1] "K" "A" "k" "a"

> ifelse(1:n%%2, letters[pos], LETTERS[pos])

[1] "k" "A" "k" "A"
```

Iterowanie

R jest również częściowo językiem funkcyjnym, tzn. możemy się całkowicie obejść bez konstrukcji imperatywnych typu `for`. Czasami dla wygody dobrze jest je również znać.

Przykład 1.5.5 Pętle

```
> for (i in rep.int(c(11, 1), 2)) {
+     print(letters[i])
+ }

[1] "k"
[1] "a"
[1] "k"
[1] "a"
```

Szczegółowe informacje o pętlach (i innych instrukcjach sterowania) można znaleźć wpisując

```
> help("for")
```

czy też

```
> help(Control)
```

Do iterowania funkcyjnego służa funkcje z rodziny `apply`, aplikujące podaną funkcje do wszystkich elementów tablicy (lub jej wierszy/kolumn). Dla wygody dostępne są funkcje `sapply` (simplify; upraszcza wynik), `lapply` (zwraca listę), `tapply` (dla faktorów).

Przykład 1.5.6 `apply`

```
> apply(outer(1:10, 1:10), 1, sum)

 [1]  55 110 165 220 275 330 385 440 495 550

> invisible(sapply(letters[rep.int(c(11, 1), 2)], print))

[1] "k"
[1] "a"
[1] "k"
[1] "a"
```

Wygodnym skrótem dla bardzo szczególnego użycia funkcji typu `apply` jest funkcja `replicate`.

Przykład 1.5.7 `replicate`

```
> sapply(rep.int(2, 10), rgeom, 0.25)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    2    0    1    4    1    0    3    0     2
[2,]    0   13    1    2    0    9    1    3    0     0
```

```
> replicate(10, rgeom(2, 0.25))

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    1    0    2    2    2    0    2    9     5
[2,]    0    8    1    4    1   15    1    0    7     4
```

Własne funkcje

Wszystkie zadania wykonywane przez nas w środowisku R to wywołania funkcji. R pozwala na definiowanie własnych funkcji przy pomocy składni

```
> nazwa.funkcji = function(parametr1, parametr2, ...) {
+     ...
+ }
```

Nazwa funkcji to po prostu nazwa zmiennej pod którą zapisany zostanie nagłówek oraz zestaw instrukcji do wykonania, , tzn. funkcje są po prostu kolejnymi obiektami środowiska R. W konsekwencji możemy je przekazywać jako parametry funkcji (por. funkcja `apply`). Wynikiem działania funkcji jest jej ostatnie wyrażenie lub argument funkcji `return`, którą można wywołać w dowolnym miejscu ciała definiowanej funkcji.

Funkcje mogą być anonimowe, tzn. nie ma obowiązku przypisania ich na zmienne.

Przykład 1.5.9 Anonimowa funkcja

```
> lapply(10^(2:4), function(n) {
+     x = rnorm(n, sd = 3)
+     m = mean(x)
+     s = sd(x)
+     c(m - s, m, m + s)
+ })

[[1]]
[1] -2.64874162 -0.03111512  2.58651138

[[2]]
[1] -3.1045001 -0.1133004  2.8778993

[[3]]
[1] -3.02496455 -0.03179387  2.96137681
```

Kod funkcji ze środowiska

Uwaga 1.5.10

Z pomocą odnośnie pisania funkcji idzie kod środowiska R. Wpisanie samej nazwy funkcji (zmiennej przechowującej funkcję!) wypisze jej kod.

```
> mean
```

```
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

Często możemy zobaczyć odwołanie przez `UseMethod` jak wyżej. Oznacza to wiele wariantów

funkcji, których wywołanie jest zależne od typu obiektu przekazanego jako pierwszy parametr (mechanizm klas S3). Wtedy zazwyczaj dostępna jest wersja domyślna `nazwa.default`.

```
> mean.default

function (x, trim = 0, na.rm = FALSE, ...)
{
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)
    }
    if (na.rm)
        x <- x[!is.na(x)]
    if (!is.numeric(trim) || length(trim) != 1)
        stop("'trim' must be numeric of length one")
    n <- length(x)
    if (trim > 0 && n > 0) {
        if (is.complex(x))
            stop("trimmed means are not defined for complex data")
        if (trim >= 0.5)
            return(stats::median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort.int(x, partial = unique(c(lo, hi)))[lo:hi]
    }
    .Internal(mean(x))
}
<environment: namespace:base>
```

Niestety, kod funkcji `R` jest mało przyjazny dla oka. Nie ma co się zrażać, z czasem staje się coraz bardziej czytelny. Zdarzają się też funkcje pre–kompilowane, których kodu po prostu nie ma w `R` (można je rozpoznać przez wywołania typu `.C`, `.Internal` czy `.Call`).

Parametry opcjonalne

Opcjonalność parametrów określa się przez podanie ich wartości domyślnych (składnią standardowego przypisania `=`). Parametry podawane przy wywołaniu funkcji dopasowywane są według jawnie podanych nazw, natomiast jeżeli takich nie mają to według kolejności z nagłówka funkcji.

Przykład 1.5.11 Parametr opcjonalny i sposoby wywołania

```
> params.test = function(x, y = 2, z) {
+     print(paste(x, y, z, collapse = "; "))
+ }
> params.test(1, 2, 3)

[1] "1 2 3"

> params.test(y = 1, 2, 3)

[1] "2 1 3"
```

Natomiast takie przypisanie

```
> params.test(1, 2)
```

zwróci błąd, ze względu na brak wartości dla argumentu `z` — kolejność w nagłówku ma znaczenie. Aby wykonać zamyślone wywołanie należy jawnie nazwać parametr bez wartości domyślnej.

```
> params.test(1, z = 2)

[1] "1 2 2"
```

Argument trzy kropki

Specjalnym argumentem są trzy kropki `...`. Oznacza on nieokreśloną liczbę argumentów. Najczęściej jest używany do przekazywania parametrów funkcjom wywoływanym wewnątrz definiowanej funkcji.

Przykład 1.5.12

```
> paste.collapse = function(...) {
+     paste(..., collapse = "")
+ }
> paste(1:10, letters[1:10])

 [1] "1 a"  "2 b"  "3 c"  "4 d"  "5 e"  "6 f"  "7 g"  "8 h"  "9 i"  "10 j"

> paste.collapse(1:10, letters[1:10])

[1] "1 a2 b3 c4 d5 e6 f7 g8 h9 i10 j"
```

Sama funkcja `paste` używa argumentu `...` w inny sposób:

```
> paste

function (..., sep = " ", collapse = NULL)
{
    args <- list(...)
    if (length(args) == 0)
        if (length(collapse) == 0)
            character(0)
        else ""
    else {
        .Internal(paste(lapply(args, as.character), sep, collapse))
    }
}
<environment: namespace:base>
```

Widoczność zmiennych

Standardowo przypisania wewnątrz ciała funkcji są lokalne. To oznacza, że zwykłe przypisanie wewnątrz funkcji nie nadpisze obiektów o tej samej nazwie spoza ciała funkcji. Obiekt stowrzony podczas wywołania funkcji zginie wraz z jej zakończeniem. Można to ominąć używając specjalnego przypisania `<<-` rozpoczynającego szukania obiektów na które przypisujemy poziom (środowisko) wyżej.

Przykład 1.5.13

```
> x = 0
```

```
> functionx1 <- function() x = 1
> functionx2 <- function() x <<- 2
> functionx1()
> x

[1] 0

> functionx2()
> x

[1] 2
```

1.6 Zadania

Zadania są przede wszystkim z tzw. statystyki opisowej i estymacji punktowej. Część zadań jest zaadoptowana ze skryptu "simple R" Johna Verzaniego (http://www.math.csi.cuny.edu/Statistics/R/simpleR/).

Zadanie 1.1

Zgadnij jakie będą wyniki następujących poleceń:

```
> y = c(2, 3, 5, 7, 11, 13)
> length(y)
> x = 2 * 1:5 - 1
> length(x)
> x + y
> sum(x > 5 | x < 3)
> y[-(3:5)]
> y[x]
```

Zadanie 1.2

Twoje czasy dojazdu na uczelnię przez ostatnie dwa tygodnie (10 dni; w minutach) to

17 16 20 24 22 15 21 15 17 22

Jakie były nadjłuższy, średni i minimalny czasy dojazdu? Jakie było odchylenie stanadardowe czasu dojazdu? Ile razy dojazd zajął Ci mniej/więcej niż średnia -/+ odchylenie standardowe? Jakie były średnie czasy dojazdu dla wartości poniżej/ponad pierwszym/trzecim kwartylem?

Zadanie 1.3

Wczytaj wbudowany zbiór danych `mtcars`. Zobacz czego dotyczy i sprawdź:

1. ile wynosi maksymalny przebieg (w milach/galon) i który samochód go osiągnął?
2. jakie wygląda pierwsza trójka samochodów o największej liczbie konii mechanicznych?
3. jakie są średnie przyspieszenia i odchylenie standardowe liczby konii dla:
   - wszystkich samochodów,
   - samochodów z/bez automatycznej skrzyni biegów,
   - mercedesów,
   - czołowych 20% samochodów pod względem liczby konii mechanicznych?

Zadanie 1.4

Dla próby losowej $X_1, \ldots, X_n$ zdefiniuj własne funkcje estymatorów średniej

$$\widehat{\mathbb{E}_n(X)} = \frac{\sum_{i=1}^n X_i}{n}$$

i (nieobciążonej) wariancji próby

$$\widehat{\mathrm{Var}_n(X)} = \frac{\sum_{i=1}^n \left(X_i - \widehat{\mathbb{E}_n(X)}\right)^2}{n-1}.$$

Porównaj z wartościami standardowych implmenetacji tych funkcji w `R` dla losowych prób rozmiaru $n = 100$ z rozkładów $Bin(10, .5)$, $Geom(.25)$, $Unif([-1, 1])$, $Exp(1.4)$.

Zadanie 1.5

Spośród średniej i mediany — estymatorów środka populacji o zadanym rozkładzie, chcemy wybrać ten o mniejszej wariancji. W tym celu oblicz stosunek wariancji średniej do wariancji mediany na podstawie $n = 1000$ prób dla prób populacji liczności $m = 100$. Za rozkład populacji przyjmij:

1. $Norm(0, 1)$,
2. $t(2)$ (rozkład t z dwoma stopniami swobody),
3. $Exp(1)$ (niesymetryczny rozkład — teoretyczna wariancja i mediana mają różne wartości).

Zadanie 1.6

Wiemy, że rzucając k razy d–ścienną kostką liczby poszczególnych wyników rzutów w stosunku do łącznej liczby rzutów, przy $k \to \infty$, dążą do d–punktowego rozkładu jednostajnego ($\mathbb{P}(X = m) = \frac{1}{d}$, $m = 1, \ldots, d$). Chcemy sprawdzić eksperymentalnie jak szybko? Zrobimy to sprawdzając jak maleje wariancja.

Dla d–wymiarowego estymatora $\widehat{X}_k = \left(\widehat{X}_k^1, \ldots, \widehat{X}_k^d\right)$ przedstawiającego proporcje wyników rzutów symetryczną monetą ($d = 2$) oszacuj $d \times d$–wymiarową macierz wariancji $\mathrm{Var}(X_k) = \left(\mathrm{Cor}\left(X_k^l, X_k^j\right)\right)_{l,j=1,\ldots,d}$, dla liczby rzutów $k = 4, 10, 20, 40$. Aby oszacować wariancję, dla każdego k, weź próby liczności $n = 50$.

Oglądanie macierzy wariancji dla dużych d może być uciążliwe. Z otrzymanych wyników chcielibyśmy wycisnąć pojedyncze liczby. W tym celu, oszacuj błąd średniokwadratowy estymatora $\widehat{X_k}$, tzn.

$$\mathbb{E}\left(\|\widehat{X}_k - \frac{1}{d}(1, \ldots, 1)\|\right)^2 \approx \sum_{i=1}^{n} \frac{\|X_{k,i} - \frac{1}{d}(1, \ldots, 1)\|^2}{n-1} = \sum_{i=1}^{n} \frac{\sum_{j=1}^{d}\left(X_{k,i}^j - \frac{1}{d}\right)^2}{n-1}$$
$$\approx \sum_{j=1}^{d}\sum_{i=1}^{n} \frac{\left(X_{k,i}^j - \widehat{\mathbb{E}_n\left(X_k^j\right)}\right)^2}{n-1} = \sum_{j=1}^{d} \widehat{\mathrm{Var}_n\left(X_k^j\right)} = \mathrm{tr}\left(\widehat{\mathrm{Var}(X_k)}\right),$$

gdzie $\mathrm{tr}(A)$ oznacza ślad macierzy A.

Zadanie 1.7

Zaimplementuj próbkowanie z odwrotnej dystrybuanty rozkładu wykładniczego $Exp(1)$ przy pomocy zmiennej losowej o rozkładzie $Unif([0, 1])$. Zrób to wektorowo. Jak sprawdzić czy dobrze próbkujesz?

Zadanie 1.8

Znajdź x, takie że:

1. $\mathbb{P}(Z \leqslant x) = .05$
2. $\mathbb{P}(\|Z\| \leqslant x) = .05$

dla $Z \sim Norm(0, \sigma^2)$, σ^2 dowolnie zadane.

Zadanie 1.9

Sprawdź eksperymentalnie zbieżność z centralnego twierdzenia granicznego dla zmiennej z ulubionego rozkładu (jak nie masz ulubionego to $Unif(0, 1)$ jest w sam raz... politycznie poprawny

etc).

Wskazówka: najprościej skorzystać z twierdzenia Gliwienki-Cantellego

Zadanie 1.10

W czasie II wojny światowej alianci przechwytywali sprzęt Niemców i na tej podstawie oceniali ile Niemcy go produkują. Oszacowania statystyków okazały się nieprawdopodobnie dokładne (i różniły się o rząd wielkości od oszacowań ekspertów). Ssprowadzało się to do estymacji parametru d rodziny d–punktowych rozkładów jednostajnych (na zbiorze $\{0, \ldots, d-1\}$ (tj. dostajemy pięć numerów seryjnych samolotów z fabryki X, roku Y, miesiąca Z; numery są od 0 do nieznanego $d-1$; zadanie: zgadnąć d na podstawie tych numerów, które widzimy). Najprostszy sposób to policzyć średnią i pomnożyć przez 2. Inny sposób to pomnożyć maximum z próby przez $\frac{n+1}{n}$, gdzie n jest rozmiarem proby. Zbadaj eksperymentalnie, który estymator jest efektywniejszy, tzn. ma mniejszą wariancję? Które z tych estymatorów są nieobciążone?

Zadanie 1.11 Teoretyczne: estymator nieobciążony $\neq$ dobry

Liczba telefonów na minutę w helpdesku, z założenia ma rozkład $X \sim Poiss(\lambda)$, gdzie λ nieznane. Odbieramy telefony przez jedną minutę i jest ich X_1. Z pewnych powodów musimy odejść od stanowiska na dwie minuty i interesuje nas prawdopodobieństwo tego, że w ciągu tych dwóch minut nie będzie żadnego telefonu, tj. $p = \mathbb{P}(X=0)^2 = e^{-2\lambda}$. Pokazać, że jedyny estymator nieobciążony dla p to $\widehat{p}(X_1) = (-1)^{X_1}$ (estymator p na podstawie X_1), czyli albo absurdalna wartość 1, albo niemożliwa wartość -1.

Rozdział 2

Wizualizacja danych

Ogromną zaletą środowiska R jest jego system graficzny i możliwość łatwej wizualizacji danych na wiele różnych sposobów. Możliwe jest kreślenie standardowych (statystycznych) wykresów, modyfikowanie ich czy też tworzenie zupełnie nowych. Funkcje graficzne R można podzielić na dwa rodzaje:

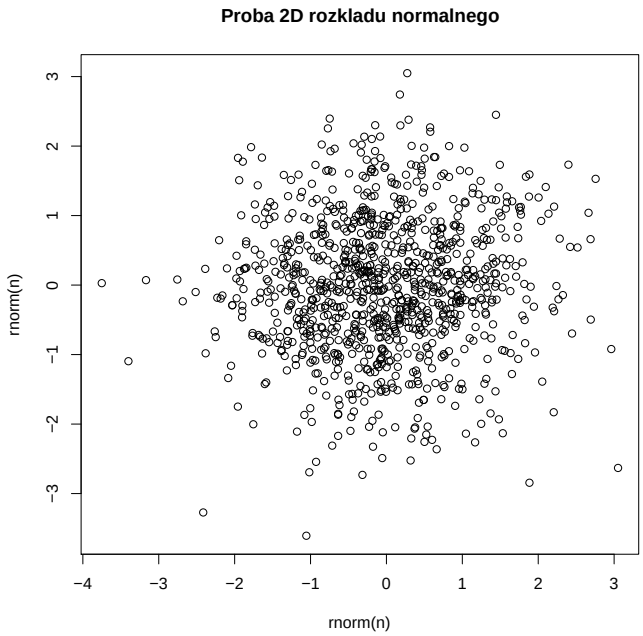
- wyskopoziomowe funkcje rysujące kompletne wykresy (przy tym usuwające poprzedni wykres),
- niskopoziomowe funkcje dodające do wykresów nowe elementy typu legenda, punkty, linie, tekst.

2.1 Rysowanie ogólne

Podstawową funkcją rysującą jest funkcja `plot` kreśląca zestaw dwuwymiarowych punktów o podanych współrzędnych.

Przykład 2.1.1 Funkcja `plot`

```
> n = 10^3
> plot(rnorm(n), rnorm(n), main = "Proba 2D rozkladu normalnego")
```



Elementy wykresu takie jak tytuł, legenda, podpisy osi można modyfikować poprzez wspólne, ogólne parametry graficzne lub bezpośrednio poprzez niskopoziomowe funkcje, za pomocą których można również dorysowywać linie, punkty lub tekst. Ich krótkie podsumowanie zawierają odpowiednio tabele 2.1 oraz 2.2.

Użycie parametrów i funkcji niskopoziomowych będzie ilustrowane wraz z kolejnymi przykładami. O ogólnych parametrach graficznych można też przeczytać w pomocy środowiska dotyczącej funkcji `par`, która to służy do modyfikacji ustawień globalnych.

Przykład 2.1.2 Tymczasowa modyfikacja ustawień graficznych funkcją `par`

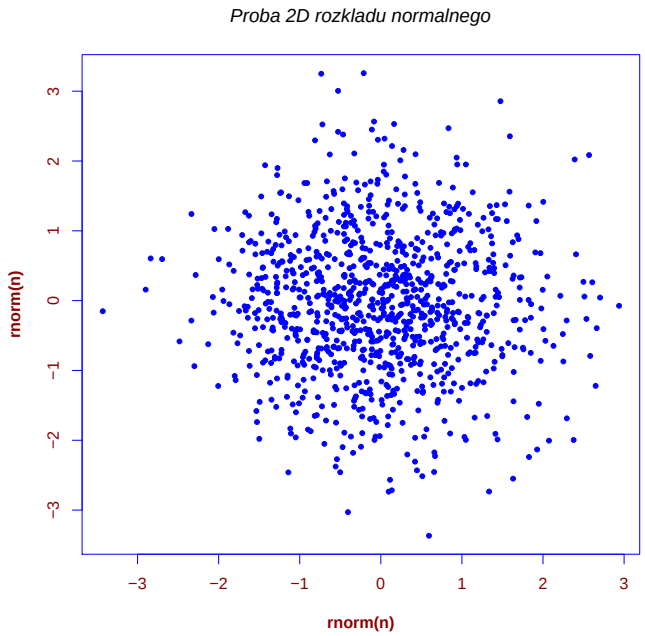
```
> old.par = par(no.readonly = TRUE)
> par(pch = 20, col.lab = "darkred", col.axis = "darkred", fg = "blue",
+     col = "blue", font.main = 3, font.lab = 2)
> n = 10^3
> plot(rnorm(n), rnorm(n), main = "Proba 2D rozkladu normalnego")
> par(old.par)
```

Parametr	Opis
`type`	Typ wykresu: `"p"` — punktowy (domyślnie), `l` — linia, `b` — punktowo–liniowt, `s`/`S` — schodkowy, `h` — kreskowy.
`main`/`sub`	Tytuł/podtytuł wykresu.
`xlab`/`ylab`	Etykiety osi.
`xlim`/`ylim`	Zakresy osi wykresu (format `c(min, max)`).
`col{.main|.sub|.axis|.lab}`	Kolor kreślonych punktów (i odpowiednio tytułu, podtytułu, wartości oraz etykiet osi).
`fg`/`bg`	Kolor elementów pierwszego planu/tła wykresu.
`lty`	Typ linii: 0 — `"blank"`, 1 — `"solid"`, 2 — `"dashed"`, 3 — `"dotted"`, 4 — `"dotdash"`, 5 — `"longdash"`, 6 — `"twodash"`.
`lwd`	Grubość linii (domyślnie 1).
`pch`	Symbol punktu: liczba lub wpisany bezpośrednio (`plot(1:25, pch=1:25)`).
`font{.main|.sub|.axis|.lab}`	Typ czcionki tekstu na wykresie (i odpowiednio tytułu, podtytułu, wartości oraz etykiet osi): 1 — normalny, 2 — pogrubiony, 3 — kursywa, 4 — pogrubiona kursywa.
`cex{.main|.sub|.axis|.lab}`	Powiększenie/pomniejszenie czcionki tekstu na wykresie (i odpowiednio tytułu, podtytułu, wartości oraz etykiet osi) względem domyślnego rozmiaru.
`family`	Rodzina czcionek tekstu na wykresie: `"serif"`, `"sans"`, `"mono"`, `"symbol"` (zależy od urządzenia graficznego).
Ustawiane tylko przez wywołanie funkcji `par`	
`xlog`/`ylog`	Flagi logarytmicznej skali osi wykresu.
`mar`/`oma`	Czteroelementowe wektory marginesu regionu kreślenia-/zewnętrznego odpowiednio z góry, z lewej, z dołu i z prawej strony.
`ps`	Rozmiar czcionki w punktach (rozmiar tekstu = `ps`·`cex`).
`mfcol`/`mfrow`	Liczba kolumn/wierszy na które jest dzielone pole wykresu (do rysowania wielu wykresów na jednym urządzeniu — zob. dalej).
`new`	Flaga pozwalająca nakreślić wykres na poprzednim (jeśli `TRUE`).

Tabela 2.1: Wybrane parametry graficzne

Funkcja	Opis
`title`	Dodaje etykiety wykresu (tytuł, podtytuł, etykiety osi). Pozwala wyspecyfikować ich położenie.
`legend`	Dodaje legendę do wykresu.
`axis`	Dodaje oś do wykresu (np. dodatkową oś z prawej strony).
`text`/`mtext`	Dodaje tekst w rejonie kreślenia/na marginesie wykresu.
`points`	Dodaje dodatkowe punkty do wykresu.
`lines`/`abline`	Dodaje do wykresu linie łamane/proste.

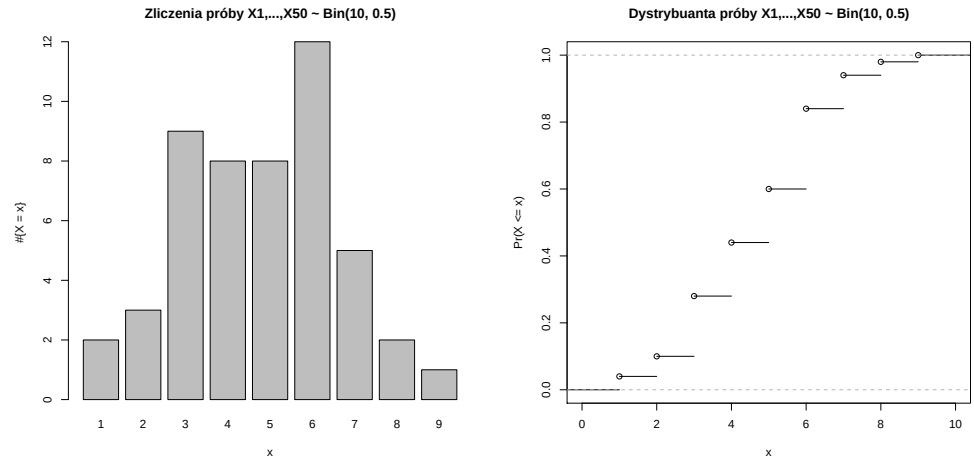
Tabela 2.2: Wybrane niskopoziomowe funkcje graficzne



W zależności od klasy obiektu wywoływana jest odpowiednia funkcja `plot`. Dzieje się to automatycznie przy pomocy prostego mechanizmu klas (zwanego S3).

Przykład 2.1.3 Automatyczne wywołanie odpowiedniej funkcji `plot`

```
> x = rbinom(50, 10, 0.5)
> plot(factor(x), main = "Zliczenia próby X1,...,X50 ~ Bin(10, 0.5)",
+      xlab = "x", ylab = "#{X = x}")

> plot(ecdf(x), main = "Dystrybuanta próby X1,...,X50 ~ Bin(10, 0.5)",
+      xlab = "x", ylab = "Pr(X <= x)")
```



W pozostałych częściach tego rozdziału zostaną omówione wysokopoziomowe funkcje rysujące standardowe, kompletne wykresy statystyczne służące do wizualizacji różnego rodzaju danych. W tej części przedstawione zostaną podstawowe funkcje kreślące oraz aspekty systemu graficznego środowiska `R`, takie jak palety kolorów, urządzenia graficzne i wiele wykresów w jednym oknie.

Kreślenie krzywych

W przypadku kreślenia funkcji lub wyrażeń (`?expression`) jednej zmiennej wygodną nakładką na funkcję `plot` jest funkcja `curve`, wykonująca za nas część pracy. W przypadku wyrażeń muszą one być wyrażeniami zmiennej `x`.

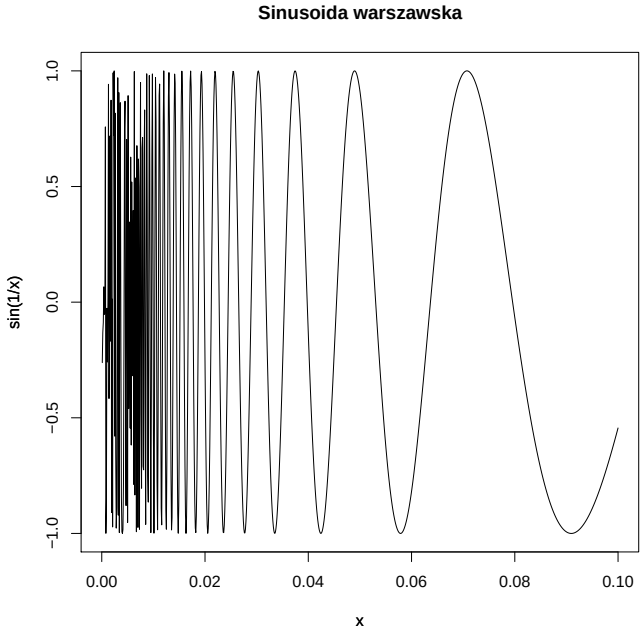
Przykład 2.1.4

Poniższy kod

```
> x = seq(0, 0.1, length = 10^3)
> plot(x, sin(1/x), type = "l")
> title(main = "Sinusoida warszawska", xlab = "x", ylab = "sin(1/x)")
```

spowoduje wykreślenie tego samego wykresu co następujący kod

```
> curve(sin(1/x), 0, 0.1, n = 10^3)
> title(main = "Sinusoida warszawska", xlab = "x", ylab = "sin(1/x)")
```



Kreślenie wielokątów

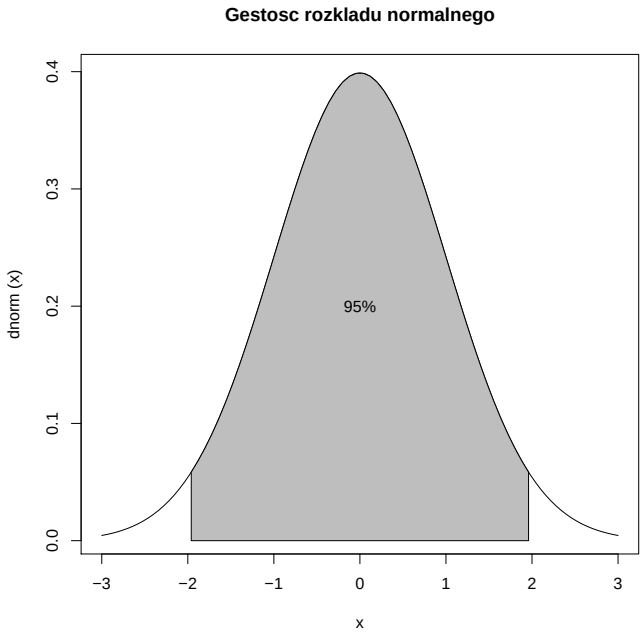
UNDER CONSTRUCTION

`rect/polygon`

Przykład 2.1.5 Wyszarzenie pola pod krzywą

```
> curve(dnorm, xlim = c(-3, 3), main = "Gestosc rozkladu normalnego")
```

```
> n = 50
> x = seq(qnorm(0.025), qnorm(0.975), length = n)
> polygon(c(x, rev(x)), c(rep(0, n), rev(dnorm(x))), col = "gray")
> text(0, dnorm(0)/2, "95%")
```

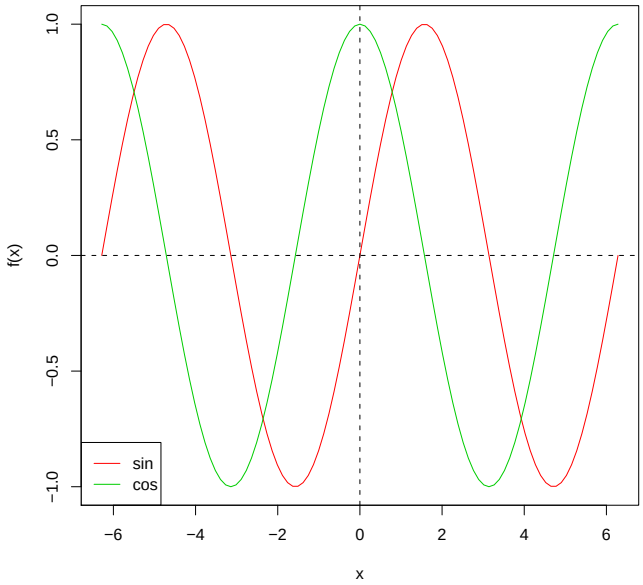


Kreślenie wielu zestawów współrzędnych

Do kreślenia wielu zestawów punktów na jednym wykresie służy funkcja `matplot` (kolejna nakładka na funkcje `plot`). Zamiast wektorów współrzędnych przyjmuje jako parametry macierze i kreśli punkty kolumnami. W przypadku nierównej liczby kolumn pierwszych i drugich współrzędnych, są one brane cyklicznie.

Przykład 2.1.6

```
> x = seq(-2 * pi, 2 * pi, length = 101)
> lty = 1
> col = 2:3
> matplot(x, cbind(sin(x), cos(x)), type = "l", lty = lty, col = col,
+     ylab = "f(x)")
> legend("bottomleft", legend = c("sin", "cos"), lty = lty, col = col)
> abline(h = 0, lty = "dashed")
> abline(v = 0, lty = "dashed")
```



Urządzenia graficzne

UNDER CONSTRUCTION

Urządzeniem graficznym może być np. okienko systemowe lub plik (postscript, pdf, png etc). Zestaw przydatnych funkcji do obsługi urządzeń graficznych:

- `?device` — lista funkcji otwierających urządzenia graficzne; w Linux'ie okienko systemowe do kreślenia otwiera się funkcją `x11`, w Windowsie `windows`;
- `dev.off` — zamknięcie urządzenia graficznego;
- `dev.print` — wygodna funkcja kopiująca zawartość aktualnego urządzenia graficznego do nowego podanego jako pierwszy parametr (+zamknięcie tego urządzenia), np. do pliku pdf.

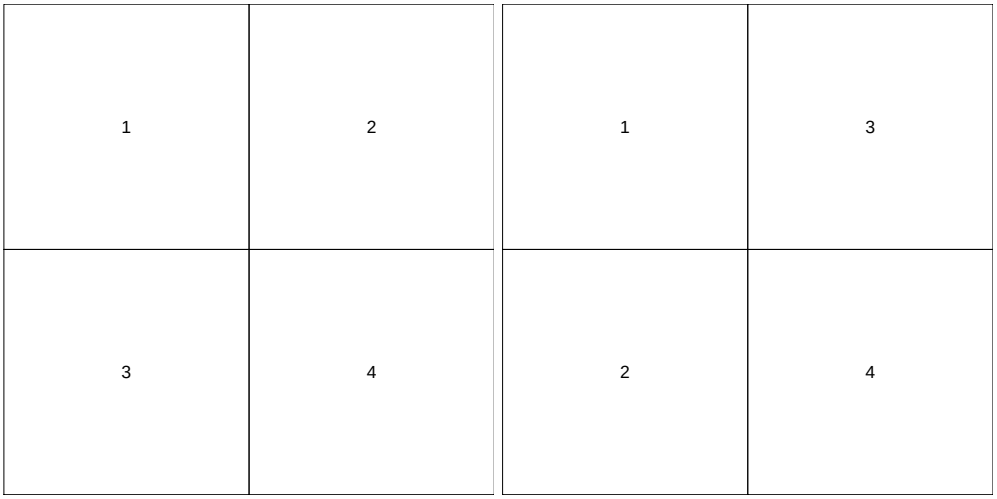
Układ wielu wykresów

Aby móc równocześnie oglądać wiele wykresów jednocześnie najłatwiej jest użyć wielu okien do kreślenia, tzn. wielu urządzeń graficznych (np. otwierać nowe okienka systemowe). Inną możliwością jest rozmieszczenie wykresów w obrębie jednego urządzenia. Można tego dokonać poprzez parametry graficzne `mfcol`/`mfrow` definiujące liczbę kolumn i wierszy macierzy wykresów lub poprzez funkcję `layout`. W przypadku pierwszej opcji wykresy są umieszczane kolejno kolumnami/wierszami w polach równej szerokości i wysokości. Druga opcja daje możliwość kontroli kolejności umieszczania oraz wysokości lub szerokości kreślonych wykresów. Aktualny układ wykresów i kolejność kreślenia można podejrzeć funkcją `layout.show`

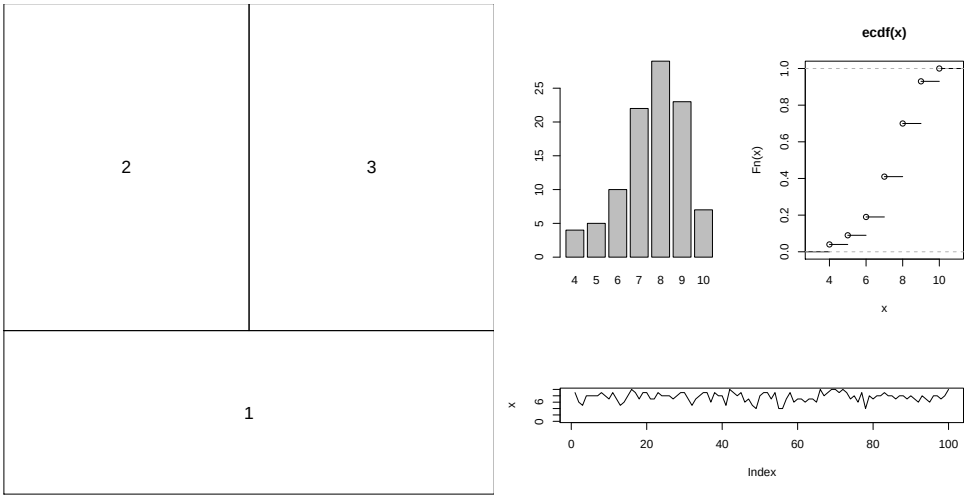
Przykład 2.1.7 Układy wykresów

```
> old.par = par(no.readonly = TRUE)
> par(mfrow = c(2, 2), cex = 1.5)
> layout.show(prod(par()$mfrow))

> par(mfcol = c(2, 2), cex = 1.5)
```

```
> layout.show(prod(par()$mfcol))
> par(old.par)
```




```
> old.par = par(no.readonly = TRUE)
> nf = layout(rbind(c(2, 3), c(1, 1)), heights = c(2, 1))
> par(cex = 1.5)
> layout.show(nf)

> par(cex = 1)
> x = rbinom(100, 10, 0.75)
> plot(x, type = "l", ylim = c(0, 10))
> plot(factor(x))
> plot(ecdf(x))
> par(old.par)
```



Palety kolorów

Żeby trochę ożywić wykres możemy nadać kolorów seriom danych lub etykietom poprzez parametry graficzne (`col{.main|.sub|.axis|.lab}`). Jako wartości można podać napisy albo liczby. Listę dostępnych po nazwach kolorów możemy uzyskać wywołując funkcję `colors`. Liczby są interpretowane jako kolejne kolory z aktualnej palety kolorów

```
> help(palette)
> palette()
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

Dostępne są funkcje pre–definiowanych palet kolorów

```
> help(heat.colors)
```

Funkcje palet kolorów przyjmują jako pierwszy argument liczbę kolorów i na jego podstawie generują paletę z ustalonego zakresu kolorów.

Przykład 2.1.8 Funkcje `palette`, `layout` i `rect`

Bardziej zaawansowany przykład użycia palety kolorów do wykreślenia trzeciego wymiaru. Do oznaczenia punktów kolorami według odpowiadających im wartości użyjemy funkcji `order` wykorzystując aktualną paletę. Legendę kolorów narysujemy ręcznie jako osobny wykres (za pomocą funkcji `layout`).

```
> old.par = par(no.readonly = TRUE)
> data(mtcars)
> palette(heat.colors(nrow(mtcars)))
> palette()
```

```
 [1] "red"     "#FF0B00" "#FF1600" "#FF2100" "#FF2C00" "#FF3700" "#FF4300"
 [8] "#FF4E00" "#FF5900" "#FF6400" "#FF6F00" "#FF7A00" "#FF8500" "#FF9000"
[15] "#FF9B00" "#FFA600" "#FFB100" "#FFBC00" "#FFC800" "#FFD300" "#FFDE00"
[22] "#FFE900" "#FFF400" "yellow"  "#FFFF10" "#FFFF30" "#FFFF50" "#FFFF70"
[29] "#FFFF8F" "#FFFFAF" "#FFFFCF" "#FFFFEF"
```

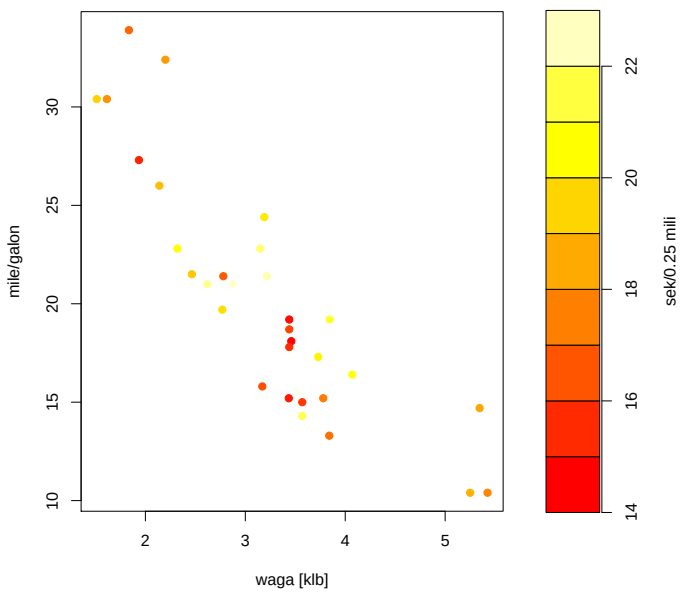
```
> layout(t(c(1, 2)), widths = c(4, 1))
> z = mtcars$qsec
> plot(mtcars$wt, mtcars$mpg, pch = 16, col = order(z), xlab = "waga [klb]",
+     ylab = "mile/galon")
```

Legenda będzie się składać z kolorowych prostokątów nakreślonych funkcją `rect`. Ponieważ `rect` jest funkcją niskopoziomową (dodającą) musimy utworzyć ręcznie nowy, pusty wykres. Do tego celu użyjemy funkcji `plot.new` i `plot.window`. Ponadto, do zdyskretyzowania skali kolorów użyjemy funkcji `pretty`, zwracającej ustalonej długości sekwencje "okrągłych" wartości z podanego zakresu wartości.

```
> lvl = pretty(range(z), 10)
> par(mar = c(4, 0, 3, 4))
> plot.new()
> plot.window(xlim = c(0, 1), ylim = range(lvl))
> rect(0, lvl[-length(lvl)], 1, lvl[-1], col = heat.colors(length(lvl) -
+     1))
```

Pozostaje dorysować oś liczbową dla legendy i jej etykietę.

```
> axis(4)
> mtext("sek/0.25 mili", side = 4, line = 3)
> par(old.par)
```



2.2 Dane etykietowane

UNDER CONSTRUCTION
`dotchart`

Proporcje danych kategorycznych

Standardowymi wykresami do reprezentacji proporcji danych kategorycznych są wykresy słupkowe (`barplot`) i kołowe (`pie`). Przyjmują one jako parametr wektory liczbowe, reprezentujące ilości elementów w poszczególnych kategoriach.

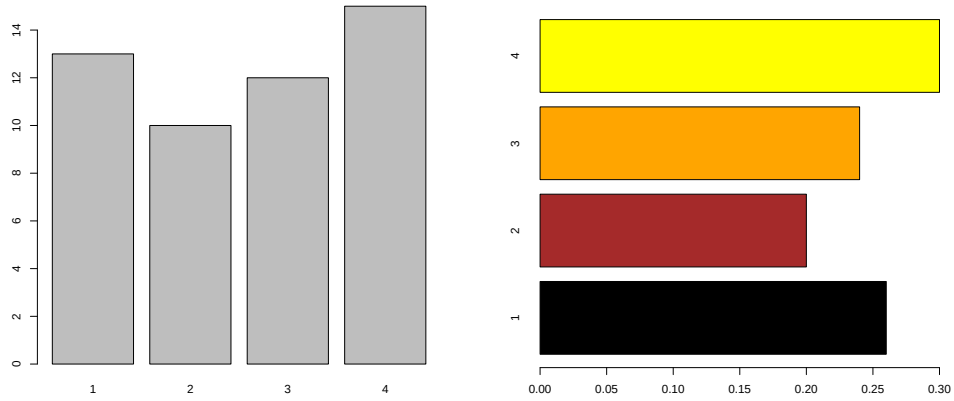
Przykład 2.2.1 Wykresy słupkowe i kołowe

```
> rdice = function(n, d) sample(1:d, n, replace = TRUE)
> d = 4
> n = 50
> dice.res = table(factor(rdice(n, d), levels = 1:d))
> dice.prop = dice.res/n
```

Wykres słupkowy

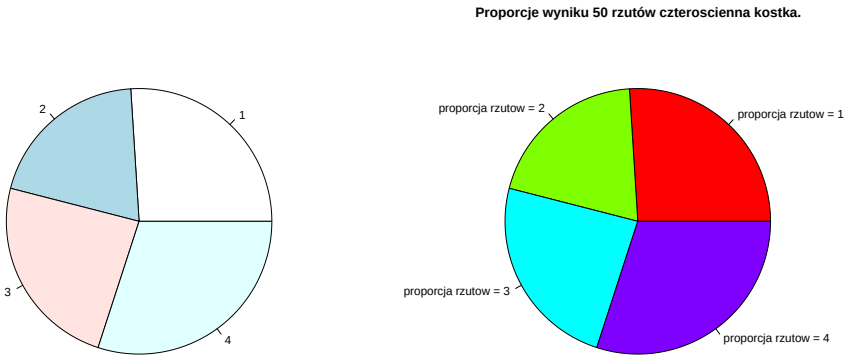
```
> barplot(dice.res)
```

```
> help(colors)
> barplot(dice.prop, horiz = TRUE, col = c("black", "brown", "orange",
+     "yellow"))
```



Wykres kołowy

```
> pie(dice.res)

> names(dice.prop) = paste("proporcja rzutow =", 1:d)
> help(rainbow)
> pie(dice.prop, col = rainbow(d))
> title("Proporcje wyniku 50 rzutów czteroscienna kostka.")
```



Zmieniliśmy etykiety serii danych przez nazwanie elementów wektora danych. Jeżeli nie chcemy na stałe zmieniać nazw elementów można to zrobić przez parametr `labels` funkcji `pie`.

2.3 Rozkład danych numerycznych

Rozkład danych numerycznych możemy kreślić przy pomocy wykresów dla danych kategorycznych, dzieląc je wcześniej na przedziały (`cut`). Środowisko `R` posiada funkcje ułatwiające nam ten proces, które tworzą bezpośrednio wykresy rozkładu danych (z pominięciem kroku ich ręcznego dzielenia). Najprostszy, tekstowy wykres to tzw. wykres "łodygowo–liściowy" (`stem`). Odpowiednikiem wykresu słupkowego jest tzw. histogram (`hist`).

Przykład 2.3.1 Wykres "łodygowo–liściowy" i histogram

```
> data(beavers)
> x = beaver1$temp
```

Wykres "łodygowo–liściowy" to wykres tekstowy, tzn. w wyniku jego działania nie powstaje obiekt graficzny. Przydaje się do szybkiego oglądu niedużych zbiorów danych.

```
> x

  [1] 36.33 36.34 36.35 36.42 36.55 36.69 36.71 36.75 36.81 36.88 36.89 36.91
 [13] 36.85 36.89 36.89 36.67 36.50 36.74 36.77 36.76 36.78 36.82 36.89 36.99
 [25] 36.92 36.99 36.89 36.94 36.92 36.97 36.91 36.79 36.77 36.69 36.62 36.54
 [37] 36.55 36.67 36.69 36.62 36.64 36.59 36.65 36.75 36.80 36.81 36.87 36.87
 [49] 36.89 36.94 36.98 36.95 37.00 37.07 37.05 37.00 36.95 37.00 36.94 36.88
 [61] 36.93 36.98 36.97 36.85 36.92 36.99 37.01 37.10 37.09 37.02 36.96 36.84
 [73] 36.87 36.85 36.85 36.87 36.89 36.86 36.91 37.53 37.23 37.20 37.25 37.20
 [85] 37.21 37.24 37.10 37.20 37.18 36.93 36.83 36.93 36.83 36.80 36.75 36.71
 [97] 36.73 36.75 36.72 36.76 36.70 36.82 36.88 36.94 36.79 36.78 36.80 36.82
[109] 36.84 36.86 36.88 36.93 36.97 37.15

> stem(x)

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

  363 | 345
  364 | 2
  365 | 04559
  366 | 224577999
  367 | 011234555566778899
  368 | 000112223344555566777788889999999
  369 | 11122233334444556777889999
  370 | 00012579
  371 | 0058
  372 | 0001345
  373 |
  374 |
  375 | 3
```

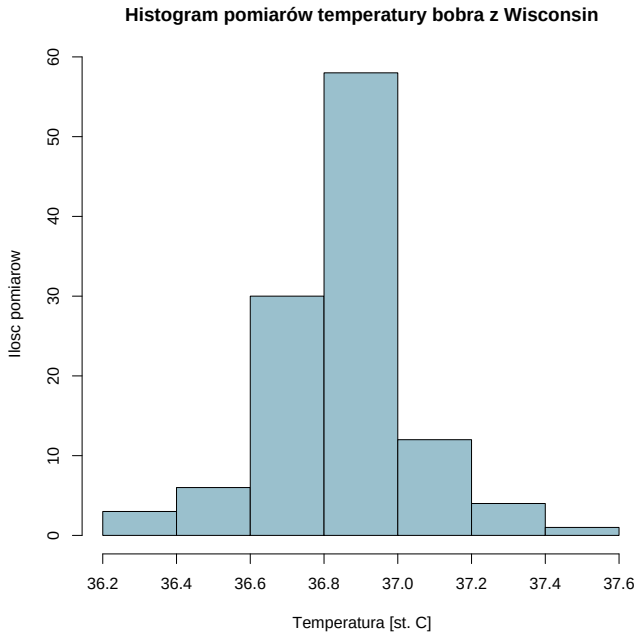
Parametr `scale` funkcji `stem` pozwala nam kontrolować szerokość kubełków.

```
> stem(x, scale = 2)

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

  363 | 34
  363 | 5
  364 | 2
```

```
  364 |
  365 | 04
  365 | 559
  366 | 224
  366 | 577999
  367 | 011234
  367 | 555566778899
  368 | 000112223344
  368 | 555566777788889999999
  369 | 11122233334444
  369 | 55677788999
  370 | 00012
  370 | 579
  371 | 00
  371 | 58
  372 | 000134
  372 | 5
  373 |
  373 |
  374 |
  374 |
  375 | 3
```

Histogram — popularny, graficzny wykres rozkładu danych numerycznych realizowany w `R` przez funkcję `hist`. Etykiety osi można nadawać poprzez ogólne parametry wykresów `x/ylab`

```
> hist(x, main = "Histogram pomiarów temperatury bobra z Wisconsin",
+      xlab = "Temperatura [st. C]", ylab = "Ilosc pomiarow", col = "lightblue3")
```



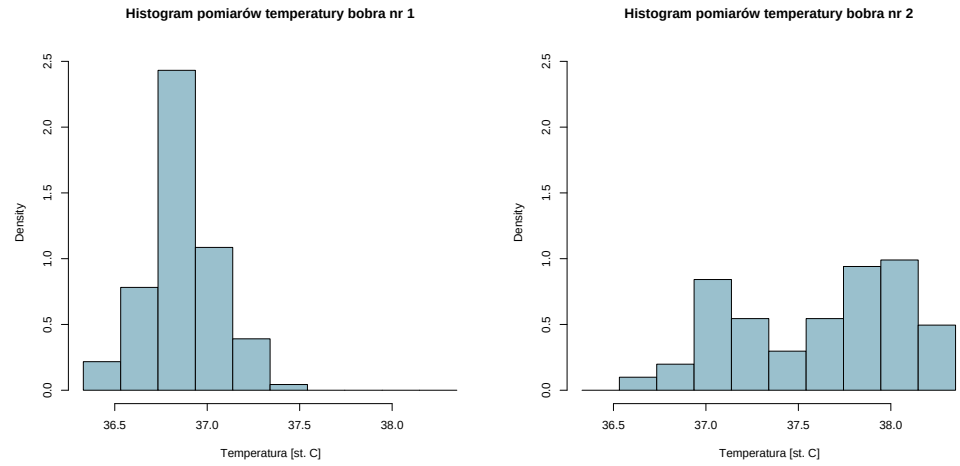
Kilka ważniejszych parametrów funkcji `hist`:

- `breaks` — parametr kontrolujący sposób kubełkowania przez: liczbę kubełków (pojedyncza wartość) lub dokładne wartości cięć (wektor) lub gotowy algorytm wyliczający liczbę cięć (napis z nazwą algorytmu; `?hist` — sekcja `Value`), czy też własną funkcją wyliczającą cięcia;
- `freq` — flaga określająca czy ma być kreślona częstość (ilość elementów w kubełkach) czy proporcja (w sensie gęstości, tzn. pole pod histogramem wynosi 1);
- `plot` — flaga pozwalająca nie kreślić histogramu i skorzystać z wyliczonych własności obiektu klasy `"histogram"`;

Przykład 2.3.2 Wizualne porównanie rozkładu dwóch serii danych

Do dopasowania szerokości i wysokości wykresu, użyjemy odpowiednio parametrów `breaks` i parametru ogólnego rażenia `ylim`, określającego zakres osi, (w formacie `c(min, max)`; analogicznie można podać parametr `xlim`).

```
> data(beavers)
> x1 = beaver1$temp
> h1 = hist(x1, plot = FALSE)
> str(h1)

List of 7
 $ breaks     : num [1:8] 36.2 36.4 36.6 36.8 37.0 ...
 $ counts     : int [1:7] 3 6 30 58 12 4 1
 $ intensities: num [1:7] 0.132 0.263 1.316 2.544 0.526 ...
 $ density    : num [1:7] 0.132 0.263 1.316 2.544 0.526 ...
 $ mids       : num [1:7] 36.3 36.5 36.7 36.9 37.1 ...
 $ xname      : chr "x1"
 $ equidist   : logi TRUE
 - attr(*, "class")= chr "histogram"

> sum(h1$density * diff(h1$breaks))

[1] 1

> x2 = beaver2$temp
> h2 = hist(x2, plot = FALSE)
> br = seq(min(x1, x2), max(x1, x2), length = max(length(h1$br),
+     length(h2$br)))
> yl = c(0, max(h1$density, h2$density))
> beaver.hist = function(temp, nr = 1, freq = FALSE, breaks = br,
+     ylim = yl, col = "lightblue3", xlab = "Temperatura [st. C]",
+     main = paste("Histogram pomiarów temperatury bobra nr",
+         nr), ...) {
+     hist(temp, breaks = breaks, freq = freq, ylim = ylim, xlab = xlab,
+         col = col, main = main, ...)
+ }

> h1 = beaver.hist(x1, nr = 1)

> h2 = beaver.hist(x2, nr = 2)
```



W celu zachowania pełnej informacji o kreślonych danych, można funkcją `rug` dodać do histogramu jednowymiarowy wykres rozrzutu (kreślony nad osią x). Do estymacji (jądrowej) ciągłej funkcji gęstości na podstawie próby służy funkcja `density`. Dzięki niej oraz funkcji `lines` pozwalającej dodawać linie do wykresu możemy uzyskać bardzo dobry obraz rozkładu danych numerycznych.

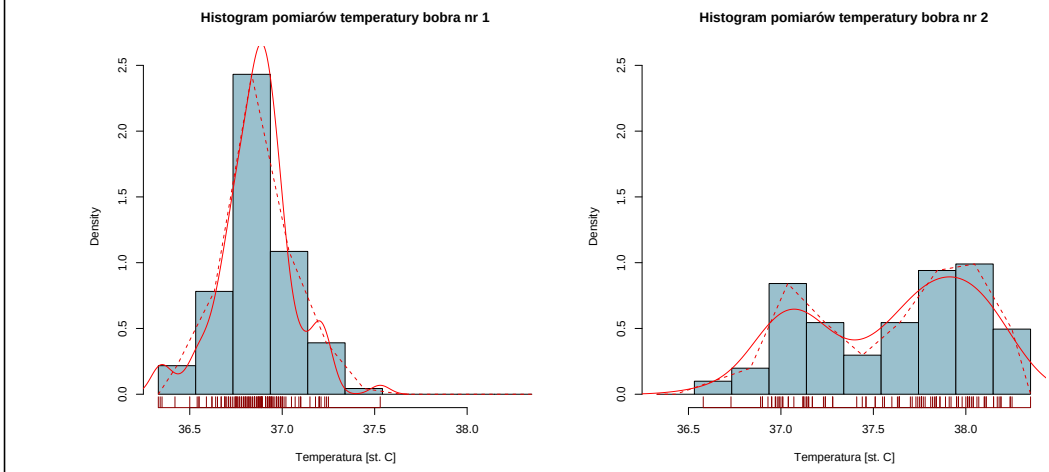
Przykład 2.3.3 Kontynuacja przykładu 2.3.2

Uzupełnienie o rozrzut i estymator gęstości oraz dla porównania przerywaną łamaną łączącą środki kubełków.

```
> hist.extras = function(x, col = "red", ...) {
+     rug(x, col = "red4")
+     lines(density(x), col = col, ...)
+ }
> hist.segline = function(h, col = "red2", lty = "dashed", ...) {
+     lines(c(min(h$breaks), h$mids, max(h$breaks)), c(0, h$density,
+         0), col = col, lty = lty, ...)
+ }
```

```
> h1 = beaver.hist(x1, nr = 1)
> hist.extras(x1)
> hist.segline(h1)
```

```
> h2 = beaver.hist(x2, nr = 2)
> hist.extras(x2)
> hist.segline(h2)
```



Opis statystyczny danych numerycznych

UNDER CONSTRUCTION

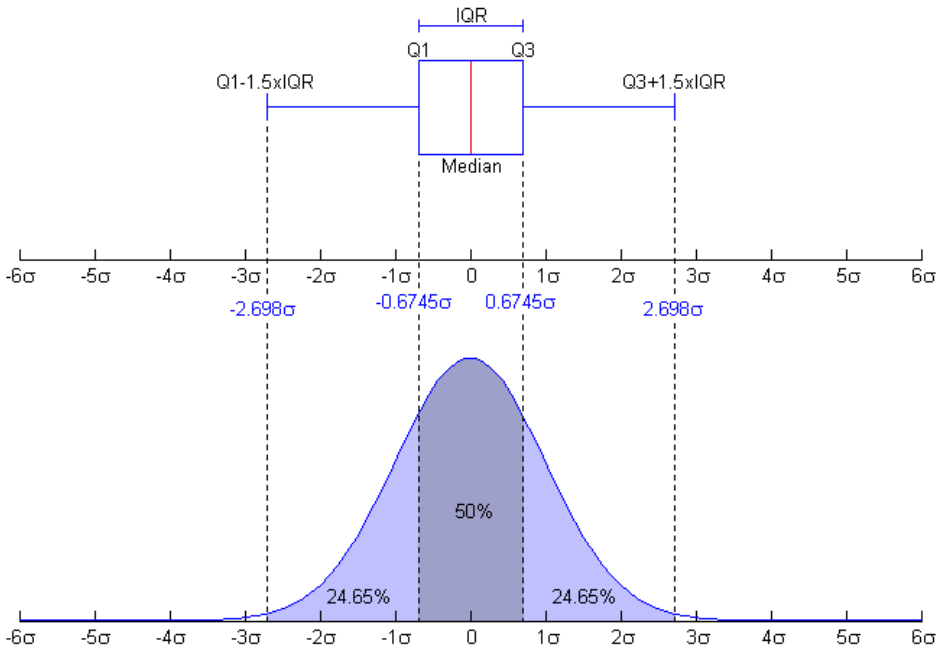
Dla małych próbek: stripchart (vide rug); przy dużych nie widać za dużo szczegółów (vide poprzedni przykład);

Wizualnym odpowiednikiem funkcji `summary` wypisującej podstawowe miary statystyki opisowej jest wykres pudełkowy (`boxplot`). Rysunek 2.1 wyjaśnia jego zawartość.

Wykresy pudełkowe pozwalają na zwięzłe i szybkie podsumowanie danych, uchwytując ich symetryczność albo skośność (położenie mediany wzlgędem pudełka i wąsów oraz ich długości) oraz "grubość ogonów" (długość wąsów i obserwacje odstające).

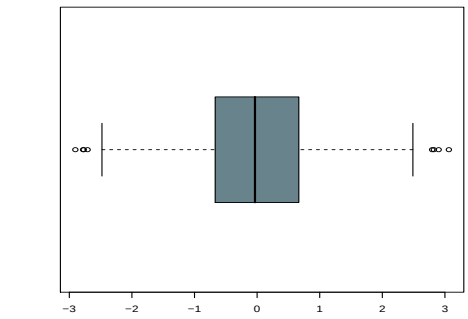
Przykład 2.3.4 Histogramy vs. wykresy pudełkowe

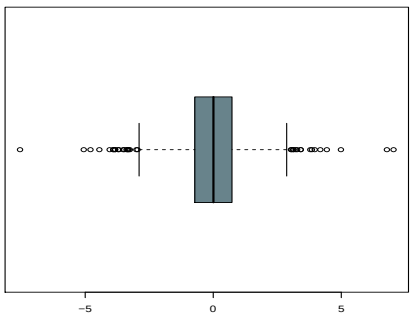
```
> n = 10^3
> xnorm = rnorm(n)
> xt = rt(n, df = 5)
> xlnorm = rlnorm(n, sdlog = 0.5)
> xexp = rexp(n)
> my.hist = function(x, freq = FALSE, col = "lightblue3", dfun = NULL,
+     ...) {
+     hist(x, freq = freq, col = col, ...)
+     rug(x, col = "red4")
+     if (is.function(dfun))
+         curve(dfun, add = TRUE, col = "red")
+ }
> my.boxplot = function(x, horizontal = TRUE, col = "lightblue4",
+     ...) {
+     boxplot(x, horizontal = horizontal, col = col)
+ }

> my.hist(xnorm, dfun = dnorm, main = "Normalny")

> my.boxplot(xnorm)

> my.hist(xt, dfun = function(x) dt(x, df = 5), main = "t (5 stopni swobody)")
```

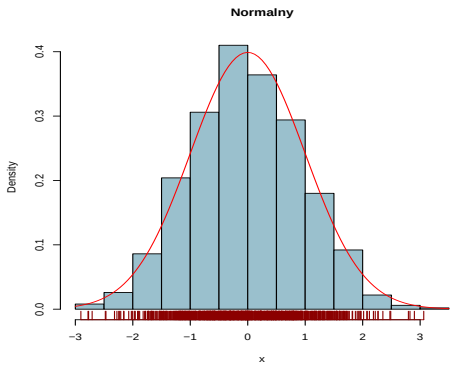


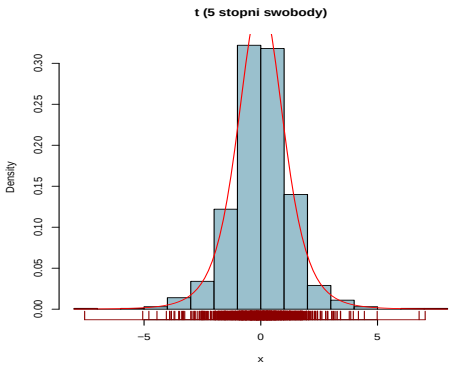
Rysunek 2.1: Elementy wykresu pudełkowego na przykładzie rozkładu normalnego: mediana, pierwszy i trzeci kwartyl ($Q1$, $Q3$), zakres pomiędzykwartylowy (IQR). Wartości poza tzw. wąsami (przedziałem $[Q1 - 1.5 \cdot IQR, Q3 + 1.5 \cdot IQR]$) są nazywane obserwacjami odstającymi (ang. outliers) i w środowisku R są rysowane jako pojedyncze punkty. Źródło rysunku: http://en.wikipedia.org.

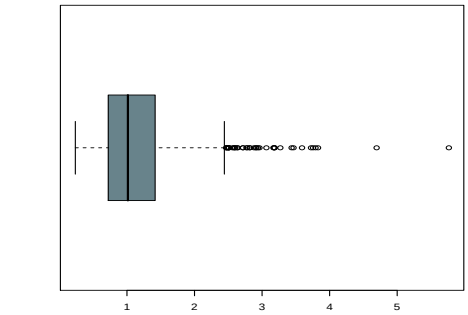
```
> my.boxplot(xt)

> my.hist(xlnorm, dfun = function(x) dlnorm(x, sdlog = 0.5), main = "Lognormalny (odchylenie .

> my.boxplot(xlnorm)

> my.hist(xexp, dfun = dexp, main = "Wykladniczy")

> my.boxplot(xexp)
```

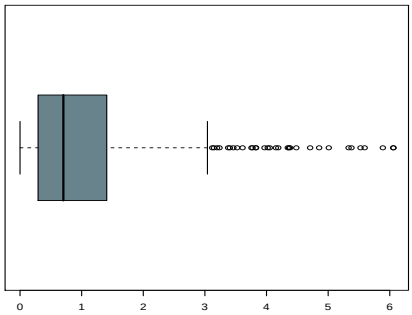

−3
−2
−1
0
1
2
3

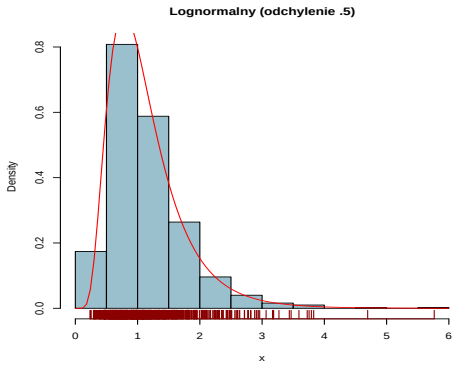

−5
0
5

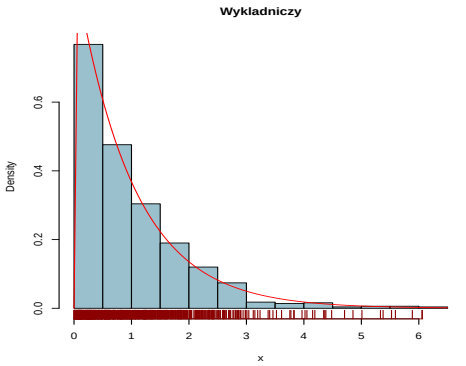

Normalny
Density
0.0
0.1
0.2
0.3
0.4
−3
−2
−1
0
1
2
3
x


t (5 stopni swobody)
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
−5
0
5
x


1
2
3
4
5


0
1
2
3
4
5
6


Lognormalny (odchylenie .5)
Density
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
5
6
x


Wykladniczy
Density
0.0
0.2
0.4
0.6
0
1
2
3
4
5
6
x

Z wykresów pudełkowych możemy wyczytać, że wylosowane próby kolejnych rozkładów są odpowiednio:

1. symetryczna,
2. symetryczna z grubymi ogonami,
3. (dodatnio) skośna z grubym dłuższym ogonem,
4. mocno (dodatnio) skośna z grubym dłuższym ogonem.

UNDER CONSTRUCTION

wykresy kwantyl–kwantyl `qqplot`/`qqnorm`

wykresy skrzypcowe `vioplot` (`library(vioplot)`).

2.4 Dane dwu i więcej wymiarowe

UNDER CONSTRUCTION

Dwuwymiarowe kategoryczne

`barplot(table(,))`.

Etykietowane, numeryczno–kategoryczne

UNDER CONSTRUCTION

`dotchart`

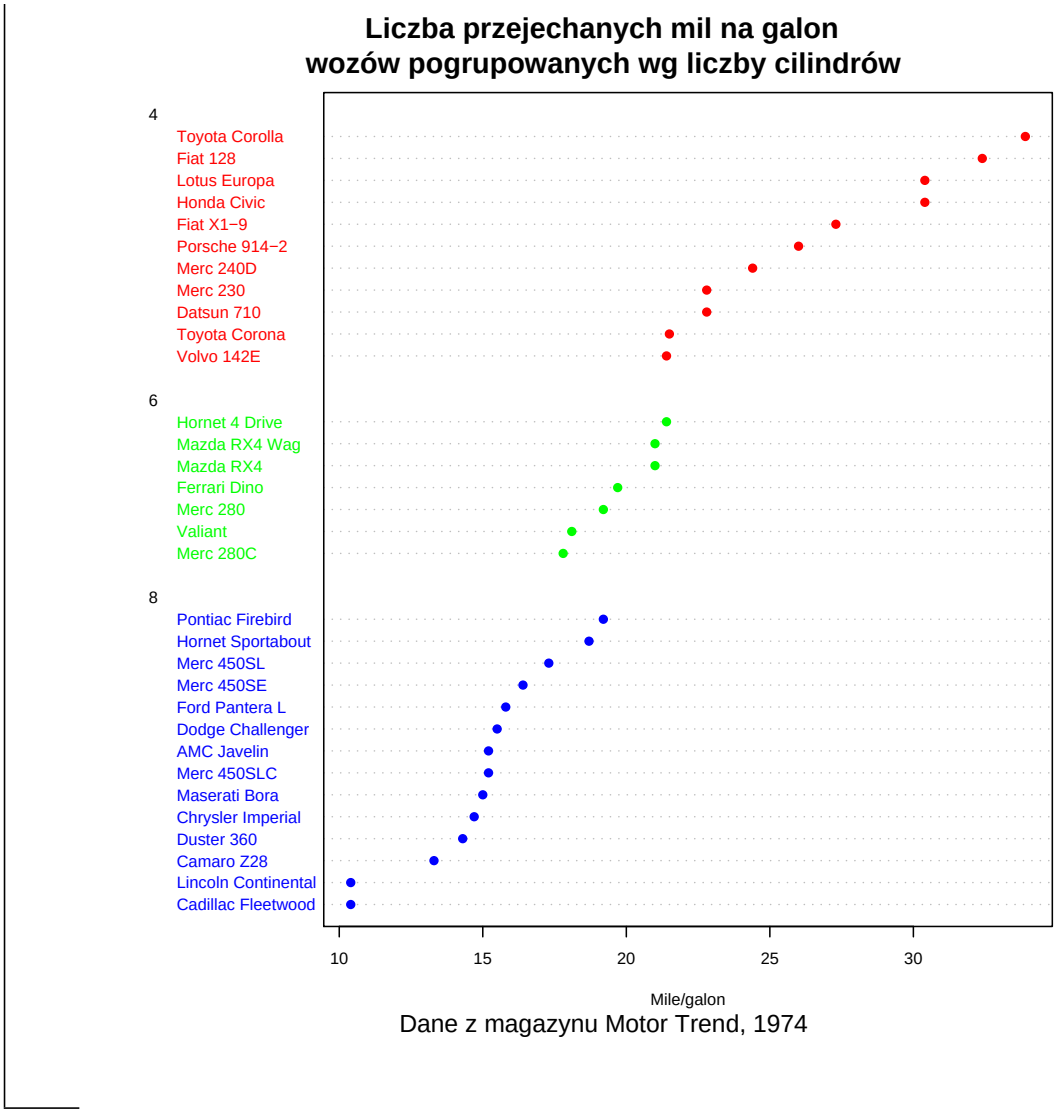
Przykład 2.4.1

```
> x = mtcars[order(mtcars$mpg), ]
> x$cyl = factor(x$cyl)
> levels(x$cyl)
```

```
[1] "4" "6" "8"
```

```
> my.palette = rainbow(length(levels(x$cyl)))
> for (i in seq_along(levels(x$cyl))) {
+     x$color[x$cyl == levels(x$cyl)[i]] = my.palette[i]
+ }
```

```
> dotchart(x$mpg, labels = row.names(x), groups = x$cyl, cex = 0.7,
+     pch = 19, xlab = "Mile/galon", color = x$color)
> title(main = "Liczba przejechanych mil na galon\nwozów pogrupowanych wg liczby cilindrów",
+     sub = "Dane z magazynu Motor Trend, 1974")
```



Numeryczne

UNDER CONSTRUCTION

Dwuwymiarowy pudełkowy.

Rozrzutu `scatterplot` i parami rozrzutu `pairs`.

Trzy wymiary

Trzeci wymiar jako: rozmiar symbolu (`symbols`), kolor (`image`), perspektywa `persp`, perspektywa + kolor `contour`/`filled.contour`.

2.5 Zadania

UNDER CONSTRUCTION

- zadanie 1.7 z wizualnym sprawdzeniem poprawności próbkowania;

- zadanie **??** z wizualnym sprawdzeniem zbieżności do rozkładu normalnego;

Rozdział 3

Informacje dodatkowe

3.1 Organizacja kodu i środowisko pracy

UNDER CONSTRUCTION

- pliki–skrypty i source;
- programowanie obiektowe/prototypowe w `R`;
- instalacja i ładowanie dodatkowych pakietów;
- pisanie własnych pakietów;
- środowisko programistyczne;

3.2 Różne uwagi programistyczne

UNDER CONSTRUCTION

- leniwe wyliczanie (promises);
- optymalizacja kodu: pętle; przykład z `R` Wiki;
- profilowanie kodu;

3.3 Kreślenie zależności

UNDER CONSTRUCTION

- wykresy treliażowe (ang. trellis; wykresy warunkowania zmienny; pakiet `lattice`)
- korelogramy (wykresy korelacji; pakiet `corrgram`)

3.4 Gdzie szukać więcej informacji?

Strona projektu

Dokumentacja na stronie domowej projektu `R`:

`http://cran.r-project.org/`

Przede wszystkim sekcje "Manuals" i "Contributed"; w szczególności wstęp do `R`, definicja języka i administracja (np. do instalacji dodatkowych pakietów).

R Seek

Wygodna wyszukiwarka pakietów, procedur itp:
http://www.rseek.org/

R Wiki

Strona wiki, z przydatnymi praktycznymi informacjami i przykładami:
http://wiki.r-project.org

R Graph Gallery

Zbiór przykładowych grafik wygenerowanych przy pomocy R, wraz z kodem:
http://addictedtor.free.fr/graphiques/

Alternatywne wprowadzenia/kursy

- Oficjalny wstęp do R:
  http://cran.r-project.org/doc/manuals/R-intro.html ,
- Bardzo dobry wstęp do R Longhow Lama:
  http://www.splusbook.com/Rintro/RCourse.pdf
- Kurs "simple R" Johna Verzaniego (z kodem dostępnym jako pakiet):
  http://www.math.csi.cuny.edu/Statistics/R/simpleR/
- Polski skrypt Łukasza Komsty:
  http://cran.r-project.org/doc/contrib/Komsta-Wprowadzenie.pdf