Wprowadzenie do środowiska R

Łukasz Komsta

21 sierpnia 2004

Spis treści

Warunki dystrybucji

Aktualna wersja dokumentu: **0.1**
Strona domowa autora: http://www.komsta.net.
Opracowanie złożono w systemie TEX/LATEX.

1. Wstęp

R-project jest pakietem matematycznym przeznaczonym do zaawansowanych obliczeń statystycznych. Objęty jest licencją GPL, a zatem jest programem całkowicie bezpłatnym i wolnym. Środowisko i składnia R jest bardzo zbliżona do języka S, stosowanego np. w pakiecie S-plus. Można powiedzieć, że R jest jedną z implementacji tego języka. Pomiędzy nimi istnieją wprawdzie pewne (niekiedy istotne) różnice, jednak większość procedur i kodu napisanego dla S będzie działać także w R.

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.

Jednym z największych „problemów” dla początkującego użytkownika R jest fakt, iż program ten działa na odmiennej zasadzie od arkuszy kalkulacyjnych. Praca odbywa się w trybie komend, w sposób analogiczny do DOS-a, czy też rozbudowanego kalkulatora. Narzuca to pewną dyscyplinę w organizacji danych lub nazywaniu zmiennych. Jednak taki system pracy jest nieoceniony w przypadku bardziej zaawansowanych obliczeń. Przy pracy z arkuszem sporo czasu zajmuje zaplanowanie rozmieszczenia danych i wyników. Tutaj nie musimy tego robić, gdyż takiego rozmieszczenia po prostu nie ma. Ponadto pakiet R pozwala na wiele obliczeń, których nie da się prosto wykonać nawet na bardziej zaawansowanych arkuszach kalkulacyjnych. Jego funkcjonalność uzupełniają dodatkowe biblioteki do konkretnych zastosowań, dostarczane wraz z obszerną dokumentacją.

Celem niniejszego opracowania jest zapoznanie czytelnika z podstawowymi operacjami dokonywanymi przy użyciu pakietu R. Założono także, że czytelnik zna podstawowe pojęcia i testy statystyczne, a tekt ten pokaże tylko sposób ich przeprowadzenia przy użyciu pakietu. Bardziej zaawansowane korzystanie z programu pociąga konieczność studiowania dokumentacji anglojęzycznej. Mam nadzieję, że uda mi się w tym opracowaniu przedstawić program R od najlepszej strony, zachęcając do dalszego odkrywania jego możliwości. Zdaję sobie również sprawę z niedoskonałości niniejszego tekstu, prosząc o wszelkie uwagi i komentarze (dane kontaktowe na mojej stronie domowej).

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).

2. Pierwsze kroki

Po uruchomieniu programu zobaczymy okno zawierające powitanie, zbliżone np. do tego:

```
R : Copyright 2004, The R Foundation for Statistical Computing
Version 1.9.0  (2004-04-12), ISBN 3-900051-00-3

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

Znak `>` jest zachętą do wprowadzenia polecenia (*prompt*). 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 kątem wystąpienia określonego słowa, np. `help.search("mean")`.

2.1. Najprostsze obliczenia

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

```
> 1+1
[1] 2
> sqrt(30)
[1] 5.477226
> log(100)
[1] 4.60517
> log(100,10)
[1] 2
> pi
[1] 3.141593
> sin(30*pi/180)
[1] 0.5
> 1/(2*sqrt(45*89))
[1] 0.007900758
> 16^(0.5)
[1] 4
```

Zwracam uwagę na to, że argumenty funkcji trygonometrycznych podaje się w radianach. Natomiast funkcja `log` dotyczy logarytmu naturalnego, ale podając 10 jako drugi jej argument, można otrzymać logarytm dziesiętny.

2.2. Przykłady operacji na wektorach

Uważnego czytelnika zastanawia zapewne jedynka w nawiasie kwadratowym, umieszczona przed każdym wynikiem. Otóż R umożliwia organizację danych w różne struktury, a najprostszą z nich jest wektor (*vector*). Jest to seria liczb o uporządkowanej kolejności. 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. Będzie to doskonale widoczne w dalszych przykładach.

Na wektorach można wykonywać przeróżne operacje. Ich rezultatem jest również wektor, którego wszystkie dane zostały w odpowiedni sposób przeliczone.

Aby przedstawić podstawowe operacje na wektorach, 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)
> dane
 [1] 10.34 10.87 10.32  9.64 10.39  9.48 10.55  9.36  9.67 10.58
> summary(dane)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  9.360   9.648  10.330  10.120  10.510  10.870
> mean(dane)
[1] 10.12
> shapiro.test(dane)
        Shapiro-Wilk normality test
data:  dane
W = 0.9004, p-value = 0.2216
> var(dane)
[1] 0.2822667
> sd(dane)
[1] 0.5312877
> 1/dane
 [1] 0.09671180 0.09199632 0.09689922 0.10373444 0.09624639 0.10548523
 [7] 0.09478673 0.10683761 0.10341262 0.09451796
> sin(dane)
 [1] -0.79269796 -0.99212590 -0.78034795 -0.21356435 -0.82217533 -0.05519398
 [7] -0.90236331  0.06473267 -0.24277173 -0.91488462
> dane^2
 [1] 106.9156 118.1569 106.5024  92.9296 107.9521  89.8704 111.3025  87.6096
 [9]  93.5089 111.9364
> t.test(dane,mu=9)
        One Sample t-test
data:  dane
t = 6.6664, df = 9, p-value = 9.2e-05
alternative hypothesis: true mean is not equal to 9
95 percent confidence interval:
  9.73994 10.50006
sample estimates:
mean of x
    10.12
> length(dane)
[1] 10
```

Jak widać, nie jest to takie straszne. Funkcja `c()` łączy podane argumenty w wektor. Zapisujemy go w zmiennej `dane` [1] . Listę wszystkich zmiennych przechowywanych w środowisku uzyskuje się poleceniem `ls()`. Na utworzonej zmiennej możemy wykonywać dalsze operacje. Po obliczeniu

[1] Do niedawna w R nie istniał operator równości, zaś przypisania dokonywało się poprzez strzałkę składającą się

średniej wykonujemy na tych danych test SHAPIRO-WILKA. Wysoka wartość p świadczy o tym, że nie ma podstaw do odrzucenia hipotezy o normalności rozkładu tych danych. Potem obliczamy wariancję i odchylenie standardowe. Kolejne przykłady są ilustracją operacji arytmetycznych na wektorach — możemy w prosty sposób obliczyć odwrotności wszystkich liczb w wektorze, ich sinusy, lub też podnieść je do kwadratu. Na zakończenie wywołujemy funkcję obliczającą test t-STUDENTA dla jednej średniej, równej 9. Jak widać, wartość p przemawia za odrzuceniem hipotezy, że wyniki pochodzą z populacji o takiej średniej. Przy okazji otrzymujemy przedział ufności 95% dla tych danych.

Jeśli długość wektora jest potrzebna w obliczeniach, można skorzystać z funkcji `length` (jak pokazano na samym końcu).

2.3. Najprostsza regresja

Innym typowym zadaniem może być dopasowanie przykładowych danych (np. krzywej kalibracyjnej) 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)
> summary(lin)
Call:
lm(formula = y ~ x)
Residuals:
      1       2       3       4       5       6
 0.9048  2.6762 -5.5524  3.2190 -5.0095  3.7619
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1.133      4.377  -0.259    0.808
x            101.229      1.124  90.070 9.11e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 4.702 on 4 degrees of freedom
Multiple R-Squared: 0.9995,     Adjusted R-squared: 0.9994
F-statistic:  8113 on 1 and 4 DF,  p-value: 9.11e-08

> sq = lm(y~x+I(x^2))
> summary(sq)
Call:
lm(formula = y ~ x + I(x^2))
Residuals:
     1      2      3      4      5      6
-1.179  3.093 -3.886  4.886 -4.593  1.679
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.7000     8.8746    0.53 0.633077
x            96.8536     5.8060   16.68 0.000469 ***
```

z minusa i odpowiedniego znaku równości. Dlatego też `x = 2` zapisywało się przy użyciu `x <- 2` bądź też `2 -> x`. Składnia ta działa cały czas i można jej używać zamiennie.

```
I(x^2)        0.6250     0.8119    0.77 0.497499
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 4.961 on 3 degrees of freedom
Multiple R-Squared: 0.9996,     Adjusted R-squared: 0.9993
F-statistic:  3643 on 2 and 3 DF,  p-value: 8.349e-06

> anova(lin,sq)
Analysis of Variance Table
Model 1: y ~ x
Model 2: y ~ x + I(x^2)
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1      4 88.419
2      3 73.836  1    14.583 0.5925 0.4975
>
```

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 R^2, reszty regresji (lub ich kwantyle, gdy danych jest więcej), estymatory poszczególnych współczynników, ich błędy standardowe, jak również wartość t oraz odpowiadającą im wartość p dla istotności tych współczynników. Widać wyraźnie, że przecięcie (*intercept*) jest nieistotne w obu przypadkach. Przy dopasowywaniu do modelu kwadratowego współczynnik przy x^2 jest również nieistotny. Na zakończenie porównujemy oba dopasowania testem Mandella (czyli test ANOVA pomiędzy modelami). Wysoka wartość p świadczy o tym, że nie ma istotnych różnic pomiędzy tymi modelami, a zatem zależność jest liniowa.

Wszystko, co powyżej zaprezentowałem, stanowi pewną demonstrację możliwości programu. Można to „przeżyć” własnoręcznie, wpisując sekwencje komend, podstawiając nawet inne liczby. Jednak ciężko wymagać od Czytelnika, aby takie z życia wzięte przykłady doprowadziły go do faktycznej umiejętności w posługiwaniu się środowiskiem R (nawet, jeśli będą coraz bardziej zaawansowane i odpowiednio dobrane merytorycznie). Dlatego w dalszej części opracowania przejdę do początku. Właściwe i świadome posługiwanie się językiem R wymaga czegoś, co można nazwać „myśleniem w R”. Takie zjawisko wymaga pewnej wiedzy, zbudowanej na solidnych podstawach.

3. Struktury danych

3.1. Wektory

Poprzedni rozdział wprowadził już pojęcie wektora (*vector*), czyli uporządkowanego ciągu danych. Wektor jest najprostszą, a jednocześnie najważniejszą, strukturą w środowisku R. Może zawierać liczby, ciągi znaków (np. podpisy do danych), jak również dane logiczne (prawda lub fałsz). Zdecydowana większość argumentów lub wyników różnych funkcji zawarta jest w wektorach, stąd też opanowanie operacji na nich jest bardzo istotne.

3.1.1. Generowanie ciągów

W operacjach na wektorach potrzebne są często ciągi liczb. Środowisko R posiada kilka funkcji pozwalających na generowanie różnych ciągów. Należą do nich:

1. Operator : (dwukropek). Powoduje on wygenerowanie wektora z liczbami całkowitymi zawierającymi się w określonym przedziale. Na przykład zapis `1:10` jest równoważny `c(1,2,3,4,5, 6,7,8,9,10`.
2. Funkcja `seq`, pozwalająca na generowanie wartości od pewnej granicy do innej granicy, jednak z krokiem innym niż 1. Np. `seq(-5,5,by=0.2)` jest równoważne `c(-5,-4.8,-4.6, ...,4.6,4.8,5)`. To samo można osiągnąć podając ilość elementów ciągu (równomierny krok dobiera się sam), w tym przypadku `seq(-5,5,length=51)`.
3. Funkcja `rep`, pozwalająca na powtórzenie wektora określoną ilość razy, np. `rep(c(1,2),2)` jest równoważne `c(1,2,1,2)`. Jeśli drugi argument jest wektorem o długości takiej samej, jak pierwszy, każdy z elementów pierwszego wektora jest powtarzany tyle razy, ile wynosi wartość odpowiadająca mu w wektorze drugim.

Popatrzmy na przykłady[2]:

```
> 1:30
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30
> 30:1 # sekwencja malejąca
 [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6
[26]  5  4  3  2  1
> n = 10
> 1:n-1 # dwukropek ma priorytet, zatem otrzymamy 1:10, pomniejszone o 1
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(n-1) # a teraz sekwencja 1:9
[1] 1 2 3 4 5 6 7 8 9
> seq(along=dane) # sekwencja od 1 do długości zmiennej dane
 [1]  1  2  3  4  5  6  7  8  9 10
> rep(1:5,5) # powtarzamy 1:5 pięć razy
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:5,rep(5,5)) # powtarzamy każdy z elementów 5 razy
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5
> rep(1:5,each=5) # inna droga, aby to uzyskać
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5
> rep(1:5,length.out=43) # wektor o długości 43, składający się z powtórzeń 1:5
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3
[39] 4 5 1 2 3
```

3.1.2. Indeksowanie wektorów

Indeksowanie wektorów jest czynnością, w której odnosimy się do konkretnych elementów, stanowiących podzbiór całości. W wyniku indeksowania otrzymuje się również wektor, jednak jest on krótszy i zawiera elementy wybrane z większego wektora.

Indeksowanie wektora polega na dodaniu tuż za nim (bez spacji) wyrażenia w nawiasach kwadratowych. Wyrażenie to jest nazwane indeksem i pokazuje, które elementy z wektora nas interesują. W charakterze indeksu można użyć:

[2] Przykłady zaopatrzone są w komentarze znajdujące się za znakiem *hash*. Oczywiście nie trzeba ich wpisywać do programu.

1. Wektora zawierającego liczby naturalne z przedziału od jedynki (pierwszy element) do długości wektora (ostatni element). Rezultatem jest wektor zawierający tylko wskazane elementy.

2. Wektora zawierającego ujemne liczby całkowite, analogicznie do p. 1. Rezultatem jest wektor zawierający wszystkie elementy *oprócz* wskazanych.

3. Wektora logicznego o takiej samej długości (złożonego z elementów TRUE lub FALSE, czyli prawda lub fałsz). Rezultatem jest wektor zawierający tylko te elementy, dla których indeks zawierał wartość TRUE.

4. Wektora zawierającego ciągi znakowe identyfikujące elementy. Ma to zastosowanie w wektorach wyposażonych wcześniej w atrybut `names`, zawierający nazwy poszczególnych elementów.

Poniższe przykłady pokazują typowe zastosowania indeksowania wektorów:

```
> a = seq(-1,1,length=10) # generujemy sekwencję
> a
 [1] -1.0000000 -0.7777778 -0.5555556 -0.3333333 -0.1111111  0.1111111
 [7]  0.3333333  0.5555556  0.7777778  1.0000000
> a[2] # drugi element
[1] -0.7777778
> a[-2] # wszystkie oprócz drugiego
[1] -1.0000000 -0.5555556 -0.3333333 -0.1111111  0.1111111  0.3333333  0.5555556
[8]  0.7777778  1.0000000
> a[c(1,5)] # pierwszy i piąty
[1] -1.0000000 -0.1111111
> a[-c(1,5)] # wszystkie oprócz nich
[1] -0.7777778 -0.5555556 -0.3333333  0.1111111  0.3333333  0.5555556  0.7777778
[8]  1.0000000
> a > 0 # wektor logiczny - które większe od zera
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> a[a>0] # ten sam wektor jako indeks
[1] 0.1111111 0.3333333 0.5555556 0.7777778 1.0000000
> a == 1 # inny wektor logiczny - uwaga na podwójną równość
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
>
```

Podwójny znak równości oznacza operator logiczny. Gdybyśmy zastosowali w ostatnim przypadku znak pojedynczy, przypisalibyśmy jedynkę do zmiennej a, zamiast sprawdzić warunek.

3.1.3. Operacje arytmetyczne

Wektory są strukturami, na których można wykonywać praktycznie wszystkie działania arytmetyczne. Np. pierwiastek z wektora jest wektorem, którego elementy są pierwiastkami elementów oryginalnego wektora. Suma dwóch wektorów o tej samej długości jest wektorem zawierającym sumy poszczególnych elementów (dwa pierwsze, dwa drugie etc.). Jeśli natomiast wektory mają różne długości, to po dojściu do końca pierwszego wektora obliczenia zaczynają się od początku:

```
> a = seq(-1,2,length=15)
```

```
> a
 [1] -1.00000000 -0.78571429 -0.57142857 -0.35714286 -0.14285714  0.07142857
 [7]  0.28571429  0.50000000  0.71428571  0.92857143  1.14285714  1.35714286
[13]  1.57142857  1.78571429  2.00000000
> mean(a) # średnia z danych
[1] 0.5
> a/mean(a) # każdą liczbę podziel przez ich średnią
 [1] -2.0000000 -1.5714286 -1.1428571 -0.7142857 -0.2857143  0.1428571
 [7]  0.5714286  1.0000000  1.4285714  1.8571429  2.2857143  2.7142857
[13]  3.1428571  3.5714286  4.0000000
> a*c(1,2) # co drugą liczbę pomnóż przez 2
 [1] -1.0000000 -1.5714286 -0.5714286 -0.7142857 -0.1428571  0.1428571
 [7]  0.2857143  1.0000000  0.7142857  1.8571429  1.1428571  2.7142857
[13]  1.5714286  3.5714286  2.0000000
Warning message:
longer object length
        is not a multiple of shorter object length in: a * c(1, 2)
>
```

Ostrzeżenie otrzymane przy ostatniej operacji mówi, iż dłuższy wektor nie jest bezpośrednią wielokrotnością krótszego.

Korzystanie z indeksów pozwala na łatwą zmianę określonych elementów wektora. Można również użyć tej samej zmiennej po lewej i po prawej stronie równania.

```
> b = a * c(1,2) # co drugi element mnożymy przez 2
> b
 [1] -1.0000000 -1.5714286 -0.5714286 -0.7142857 -0.1428571  0.1428571
 [7]  0.2857143  1.0000000  0.7142857  1.8571429  1.1428571  2.7142857
[13]  1.5714286  3.5714286  2.0000000
> a-b # różnice między elementami
 [1]  0.00000000  0.78571429  0.00000000  0.35714286  0.00000000 -0.07142857
 [7]  0.00000000 -0.50000000  0.00000000 -0.92857143  0.00000000 -1.35714286
[13]  0.00000000 -1.78571429  0.00000000
> b[c(1,3,5)] = c(10,11,12) # wstawiamy 10,11,12 na 1,3 oraz 5 pozycję
> b
 [1] 10.0000000 -1.5714286 11.0000000 -0.7142857 12.0000000  0.1428571
 [7]  0.2857143  1.0000000  0.7142857  1.8571429  1.1428571  2.7142857
[13]  1.5714286  3.5714286  2.0000000
> a[1:5] = 0 # zerujemy 5 pierwszych elementów
> a
 [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.07142857
 [7] 0.28571429 0.50000000 0.71428571 0.92857143 1.14285714 1.35714286
[13] 1.57142857 1.78571429 2.00000000
> a[8] = b[8] # niech 8 element wektora a jest równy 8 elementowi b
> a
 [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.07142857
 [7] 0.28571429 1.00000000 0.71428571 0.92857143 1.14285714 1.35714286
[13] 1.57142857 1.78571429 2.00000000
```

```
> a = 1/a # niech wektor a zawiera odwrotności dotychczasowych wartości
> a
 [1]        Inf        Inf        Inf        Inf        Inf 14.0000000
 [7]  3.5000000  1.0000000  1.4000000  1.0769231  0.8750000  0.7368421
[13]  0.6363636  0.5600000  0.5000000
```

Paru wyjaśnień wymaga `Inf` zawarte w ostatnim wyniku. Jest to zapis oznaczający nieskończoność (∞). Podobnie `NaN` oznacza wartość nieokreśloną (np. rezultat dzielenia zera przez zero lub pierwiastka z liczby ujemnej[3]).

3.1.4. Funkcje operujące na wektorach

Celem łatwego operowania na wektorach, język R zawiera cały zestaw różnych funkcji. Poniżej przytaczam listę najważniejszych:

- `max` — maksymalna wartość z wektora
- `min` — wartość minimalna
- `mean` — średnia arytmetyczna. Jeśli podamy dodatkowy parametr `trim`, to funkcja policzy średnią po odrzuceniu określonego odsetka wartości skrajnych, np. `mean(x,trim=0.1)` to średnia z x po odrzuceniu 10% wartości skrajnych
- `median` — mediana
- `mad` — medianowe odchylenie bezwzględne (*median absolute deviation*)
- `quantile` — dowolny kwantyl, np. `quantile(x,.5)` to mediana z x
- `sd` — odchylenie standardowe
- `var` — wariancja
- `length` — długość wektora (liczba elementów)
- `sum` — suma elementów wektora[4].
- `prod` — iloczyn elementów
- `sort` — daje wektor z wartościami uporządkowanymi rosnąco
- `pmin, pmax` — funkcje te operują na kilku wektorach tej samej długości, dając wektor zawierający najmniejsze lub największe elementy wybrane z poszczególnych wektorów. Tzn. pierwszy element będzie miał wartość największą spośród pierwszych, drugi spośród drugich etc.
- `cummin, cummax` — funkcja zwraca wektor zawierający dla każdego elementu wartości minimalne lub maksymalne znalezione „dotychczas” czyli od pierwszego elementu do aktualnego.
- `which` — daje wektor zawierający indeksy, przy których argument ma wartość TRUE.
- `diff` — zwraca wektor krótszy o 1, zawierający różnice między sąsiadującymi elementami.

[3] Pierwiastek z liczby ujemnej może zostać obliczony jako liczba zespolona, jeśli argument podamy również w tej formie, np. `sqrt(120+0i)`.

[4] W przypadku wektorów logicznych sumę stanowi liczba elementów o wartości TRUE.

- `rank` — daje wektor kolejności, w jakiej należałoby poukładać dane, aby były ułożone rosnąco (wektor rang).

Popatrzmy na kilka przykładów:

```
> 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
[1] 59
> sum(a>3) # ile elementów jest większych od 3?
[1] 8
> sum(a[a>3]) # zsumujmy te elementy!
[1] 52
> pmin(a,b) # wartości minimalne
 [1] 2 2 0 4 2 6 1 5 3 5 4
> pmax(a,b) # wartości maksymalne
 [1] 6 8 4 8 3 6 8 9 4 8 6
> length(a)
[1] 11
> c=c(a,b) # a teraz łączymy wektory!
> c
 [1] 6 2 4 8 2 6 8 9 3 5 6 2 8 0 4 3 6 1 5 4 8 4
> sort(c) # zobaczmy, jak wygląda po posortowaniu
 [1] 0 1 2 2 2 3 3 4 4 4 4 5 5 6 6 6 6 8 8 8 8 9
> which(c>3) # które elementy są większe niż 3?
 [1]  1  3  4  6  7  8 10 11 13 15 17 19 20 21 22
> range(c) # jaki jest zakres (min i max?)
[1] 0 9
> cummin(c) # najmniejsza wartość dotychczasowa
 [1] 6 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0
> cummax(c) # największa
 [1] 6 6 6 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
> diff(c) # różnice między kolejnymi wartościami
 [1] -4  2  4 -6  4  2  1 -6  2  1 -4  6 -8  4 -1  3 -5  4 -1  4 -4
```

W powyższym przykładzie dokonaliśmy po raz pierwszy połączenia dwóch wektorów w jeden większy wektor. Takie operacje są jak oczywiście możliwe, argumentem funkcji `c()` mogą być dowolne wektory, także z indeksem.

3.2. Faktory i dane kategoryczne

Faktor (*factor*) jest specjalną strukturą, przechowującą (oprócz szeregu danych) informacje o powtórzeniach takich samych wartości oraz o zbiorze unikalnych wartości tego ciągu. Zrozumienie działania faktora jest trudne nawet dla doświadczonych użytkowników R, jednak nie jest w pełni konieczne do korzystania z zalet, jakie one oferują.

Wyobraźmy sobie, że przeprowadziliśmy wśród 20 osób ankietę, zadając jakieś pytanie. Możliwymi odpowiedziami są: „tak” (T), „nie” (N) lub „nie wiem” (X). Tworząc wektor uzyskanych odpowiedzi, można z niego utworzyć prosty faktor. Taki wektor z odpowiedziami zawiera tzw. dane kategoryczne (*categorical data*). Zestawienie ilości poszczególnych odpowiedzi możemy uzyskać przy użyciu funkcji `table`:

```
> ankieta=c("T","N","T","T","X","N","T","X","N","T","N","T","T","N","X",
"N","T","N","T","T")
> factor(ankieta) # jak wyświetli się faktor z takiej ankiety?
 [1] T N T T X N T X N T N T T N X N T N T T
Levels: N T X
> table(ankieta) # ile jakich odpowiedzi?
ankieta
 N  T  X
 7 10  3
> 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)
 [1]  1  2  2  2  3  3  4  4  4  5  5  5  5  6  6  6  7  8  8  9 32 76
> factor(inne)
 [1] 1  2  3  2  3  4  5  2  4  32 4  8  9  76 5  6  5  6  5  8  7  6
Levels: 1 2 3 4 5 6 7 8 9 32 76
> levels(factor(inne)) # jakie unikalne wartości?
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "32" "76"
```

Funkcja `table` potrafi wykonać zestawienie dwuwymiarowe. Wyobraźmy sobie, że każdy respondent odpowiadał na 2 pytania, i chcemy znać liczby respondentów odpowiadające wszystkim kombinacjom odpowiedzi. Tworzymy zatem drugą tabelę odpowiedzi `ankieta2`. W poniższym przykładzie użyję do tego celu funkcji `rev`, w realnym świecie byłyby to całkiem odrębne dane:

```
> ankieta2=rev(ankieta)
> table(ankieta, ankieta2)
       ankieta2
ankieta N T X
      N 0 5 2
      T 5 4 1
      X 2 1 0
```

Stosując inną funkcję `prop.table` można otrzymać analogiczne zestawienie, zawierające ułamki respondentów w stosunku do całości.

Funkcję `table` można też stosować do zestawień powtarzających się wartości liczbowych:

```
> inne
 [1]  1  2  3  2  3  4  5  2  4 32  4  8  9 76  5  6  5  6  5  8  7  6
> table(inne)
inne
 1  2  3  4  5  6  7  8  9 32 76
 1  3  2  3  4  3  1  2  1  1  1
```

Bardziej zaawansowane operacje na tych danych można przeprowadzić funkcją `tapply`. Grupuje ona dane liczbowe w grupy odpowiadające poszczególnym danym kategorycznym, a następnie przeprowadza na tych grupach określone operacje. Utwórzmy wektor `wiek`, zawierający dane wiekowe poszczególnych respondentów, a następnie policzmy średnią i odchylenie standardowe wieku dla poszczególnych odpowiedzi:

```
> wiek=c(20,23,45,21,67,34,52,31,59,38,19,44,64,18,40,50,32,31,18,20)
> length(wiek)
[1] 20
> tapply(wiek,ankieta,mean)
       N        T        X
33.42857 35.40000 46.00000
> tapply(wiek,ankieta,sd)
       N        T        X
15.75708 15.85490 18.73499
```

Dane kategoryczne można w prosty sposób utworzyć z danych liczbowych. Wtedy kategorie będą oznaczać przedziały wartości. Pogrupujmy funkcją `cut` wiek ankietowanych w przedziale 0-20, 20-40, 40-50, powyżej 50:

```
> wiek2 = cut(wiek,c(0,20,40,50,100))
> wiek2
 [1] (0,20]   (20,40]  (40,50]  (20,40]  (50,100] (20,40]  (50,100] (20,40]
 [9] (50,100] (20,40]  (0,20]   (40,50]  (50,100] (0,20]   (20,40]  (40,50]
[17] (20,40]  (20,40]  (0,20]   (0,20]
Levels: (0,20] (20,40] (40,50] (50,100]
> table(wiek2) # ile osób w każdym przedziale?
wiek2
  (0,20]  (20,40]  (40,50] (50,100]
       5        8        3        4
```

3.3. Tablice

3.3.1. Tworzenie i indeksowanie tablic

Tablica (*array*) jest wektorem, zawierającym dodatkowe dane określające uporządkowanie elementów. Najprostszą i najczęściej stosowaną formą jest tablica dwuwymiarowa (macierz), jednak istnieje możliwość stosowania dowolnej liczby wymiarów.

Tablice są bardzo wygodnym narzędziem do przechowywania informacji, gdyż niezależnie od uporządkowania elementów w wiersze i kolumny, cała tabela jest dostępna pod postacią jednolitego wektora.

Indeksowanie tablic odbywa się podobnie do wektorów, w nawiasie kwadratowym podajemy „współrzędne” indeksowanego elementu. W razie pominięcia współrzędnej wynikiem indeksowania jest cały rząd lub kolumna. Jeśli pomijamy współrzędną w indeksie, należy pamiętać o pozostawieniu przecinka.

Tablicę można stworzyć z istniejącego już wektora, poprzez przypisanie wymiarów do funkcji `dim` wywołanej na nim. Innymi, bardziej naturalnymi funkcjami tworzącymi tablice są `matrix` i `array`. Popatrzmy na przykłady:

```
> tbl=1:20
> dim(tbl)=c(4,5) # wektor staje się tablicą o wymiarach 4,5
```

```
> tbl
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> tbl[1]
[1] 1
> tbl[12] # cały czas można się odwoływać do poszczególnych elementów!
[1] 12
> tbl[,1] # pierwsza kolumna
[1] 1 2 3 4
> tbl[1,] # pierwszy rząd
[1]  1  5  9 13 17
> tbl[2,2]# drugi rząd i druga kolumna
[1] 6
```

Istnieje możliwość zmiany wymiarów istniejącej już tablicy bez żadnych ograniczeń[5]. Oto przykład:

```
> dim(tbl) # wyświetlamy wymiary
[1] 4 5
> dim(tbl) = c(2,10) # zmieniamy wymiary!
> tbl
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17    19
[2,]    2    4    6    8   10   12   14   16   18    20
> tbl[,6] # a teraz szósta kolumna
[1] 11 12
```

Jeśli parametrem w nawiasie kwadratowym (indeksem) jest tablica o dwóch kolumnach, rezultatem jest wektor zawierający dane z tabeli o współrzędnych zamieszczonych w poszczególnych wierszach indeksu (kolumna w pierwszym, wiersz w drugim). W ten sposób można się jednorazowo odnosić do całej serii elementów tabeli, niezależnie od ich układu:

```
> dim(tbl)=c(4,5) # znów narzucamy wymiary 4,5
> i = array(c(1:4,4:1),dim=c(4,2)) # tworzymy tablicę współrzędnych
> i
     [,1] [,2]
[1,]    1    4
[2,]    2    3
[3,]    3    2
[4,]    4    1
```

Narzuciliśmy tabeli `tbl` poprzednie wymiary, a następnie utworzyliśmy tablicę współrzędnych 1,4; 2,3; 3,2; 4,1. Możemy się teraz jednorazowo odwołać do wszystkich elementów o tych współrzędnych:

[5] Jedynym ograniczeniem jest to, iż iloczyn wymiarów musi być równy ilości elementów.

```
> tbl[i] # wartości z tabeli tbl o współrzędnych zawartych w i
[1] 13 10  7  4
> tbl[i]=NA # ustawmy je na NA
> tbl # i zobaczmy, co wyszło
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   NA   17
[2,]    2    6   NA   14   18
[3,]    3   NA   11   15   19
[4,]   NA    8   12   16   20
```

3.3.2. Operacje na tablicach

Jedną z ciekawszych funkcji operujących na tablicach jest `outer`. Tworzy ona z dwóch tablic znacznie większą tablicę wielowymiarową. Wymiary tej tablicy są połączeniem wektorów wymiarów dwóch tablic, zaś jej zawartość stanowią wszystkie możliwe kombinacje iloczynów (lub innych operacji) pomiędzy elementami. Najprostszą ilustracją użycia tej funkcji jest stworzenie tabliczki mnożenia, oraz „tabliczki dzielenia”:

```
> outer(1:10,1:10)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    2    3    4    5    6    7    8    9    10
 [2,]    2    4    6    8   10   12   14   16   18    20
 [3,]    3    6    9   12   15   18   21   24   27    30
 [4,]    4    8   12   16   20   24   28   32   36    40
 [5,]    5   10   15   20   25   30   35   40   45    50
 [6,]    6   12   18   24   30   36   42   48   54    60
 [7,]    7   14   21   28   35   42   49   56   63    70
 [8,]    8   16   24   32   40   48   56   64   72    80
 [9,]    9   18   27   36   45   54   63   72   81    90
[10,]   10   20   30   40   50   60   70   80   90   100
> outer(1:10,1:10,"/") # wskazujemy operator jako trzeci argument
      [,1] [,2]      [,3] [,4] [,5]      [,6]      [,7]  [,8]      [,9] [,10]
 [1,]    1  0.5 0.3333333 0.25  0.2 0.1666667 0.1428571 0.125 0.1111111   0.1
 [2,]    2  1.0 0.6666667 0.50  0.4 0.3333333 0.2857143 0.250 0.2222222   0.2
 [3,]    3  1.5 1.0000000 0.75  0.6 0.5000000 0.4285714 0.375 0.3333333   0.3
 [4,]    4  2.0 1.3333333 1.00  0.8 0.6666667 0.5714286 0.500 0.4444444   0.4
 [5,]    5  2.5 1.6666667 1.25  1.0 0.8333333 0.7142857 0.625 0.5555556   0.5
 [6,]    6  3.0 2.0000000 1.50  1.2 1.0000000 0.8571429 0.750 0.6666667   0.6
 [7,]    7  3.5 2.3333333 1.75  1.4 1.1666667 1.0000000 0.875 0.7777778   0.7
 [8,]    8  4.0 2.6666667 2.00  1.6 1.3333333 1.1428571 1.000 0.8888889   0.8
 [9,]    9  4.5 3.0000000 2.25  1.8 1.5000000 1.2857143 1.125 1.0000000   0.9
[10,]   10  5.0 3.3333333 2.50  2.0 1.6666667 1.4285714 1.250 1.1111111   1.0
```

Ostatnim argumentem `outer` może być również nazwa funkcji operującej na dwóch zmiennych. Dlatego też istnieje możliwość przeprowadzenia dowolnych operacji pomiędzy tablicami przez utworzenie własnej funkcji (o tym dalej).

Kolejna funkcja `aperm` dokonuje przegrupowania tabeli. Drugi argument musi być permutacją liczb z zakresu $1 \ldots k$, gdzie k oznacza liczbę wymiarów tabeli. Jeśli argumentem jest np. `c(3,1,2)`, to trzeci wymiar stanie się pierwszym, pierwszy drugim, a drugi trzecim. Popatrzmy na przykład:

```
> tbl
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   NA   17
[2,]    2    6   NA   14   18
[3,]    3   NA   11   15   19
[4,]   NA    8   12   16   20
> aperm(tbl,c(2,1))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3   NA
[2,]    5    6   NA    8
[3,]    9   NA   11   12
[4,]   NA   14   15   16
[5,]   17   18   19   20
```

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 tabelę. Odwrotnie, każdą tabelę można prosto przekształcić na wektor, stosując funkcję `as.vector`. Oto przykłady:

```
> cbind(1:5,6:10,11:15)
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
> rbind(1:5,6:10,11:15)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
> as.vector(tbl)
 [1]  1  2  3 NA  5  6 NA  8  9 NA 11 12 NA 14 15 16 17 18 19 20
```

3.4. Listy

Lista (*list*) jest uporządkowanym zbiorem elementów różnego typu. Każdy z nich może posiadać również nazwę, przez którą można się do niego odwołać w dalszych operacjach.

Do tworzenia list służy funkcja `list`. Jeśli wywołamy ją z listą liczb, otrzymamy listę jednoelementowych wektorów:

```
> s = list(1,2)
> s
[[1]]
[1] 1

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

Jeśli chcemy odwołać się do konkretnego elementu listy i znamy jego numer, to można listę indeksować, jednak indeks należy podać w podwójnych nawiasach kwadratowych. W powyższym przypadku `s[[1]]` to pierwszy element listy, czyli jednoelementowy wektor z jedynką. Jeśli elementem listy jest wektor wieloelementowy lub tablica, można oczywiście odwoływać się dalej, podając jeszcze jeden indeks, tym razem w pojedynczym nawiasie, np. `s[[2]][1]`.

Każdy z elementów listy może mieć określoną nazwę i takie nazwane listy spotyka się w praktyce najczęściej. Wtedy też można się odnosić do konkretnych elementów poprzez sprzężenie nazwy listy z nazwą elementu znakiem dolara. Nie musimy pamiętać, który z kolei element nas interesuje, wystarczy znać jego nazwę:

```
> s = list(wektor=c(1,2,3),srednia=2,tablica=array(data=c(1,2,3,4),dim=c(2,2)))
> s # zobaczmy, jak wygląda lista
$wektor
[1] 1 2 3
$srednia
[1] 2
$tablica
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Upraszcza to znacznie odwołania do poszczególnych zmiennych listy:

```
> s$wektor # odniesienie do elementu "wektor"
[1] 1 2 3
> s$wektor[1] # odniesienie do pierwszego elementu tej zmiennej
[1] 1
> s[[1]][1] # to samo, lecz z wykorzystaniem numeru kolejnego
[1] 1
> s$tablica[,1] # pierwszy rząd elementu "tablica"
[1] 1 2
```

Nazwy elementów listy można skracać do takiej długości, która wystarcza do jednoznacznej ich identyfikacji. W powyższym przypadku zamiast `s$tablica` wystarczy napisać `s$t`. Istnieje również możliwość łączenia list funkcją `c`, np. `lista3 = c(lista1,lista2)`.

3.5. Ramki

Ramka (*dataframe*) to specyficzna struktura środowiska R. Najprościej można określić ją jako macierz, w której poszczególne kolumny mogą zawierać wartości różnego typu. Do utworzenia takiej struktury służy funkcja `data.frame`. Można również przekształcić inne struktury na ten typ funkcją `as.data.frame`.

Pakiet R pozwala na łatwe ładowanie danych z zewnętrznych plików. Każdą tabelę zawartą w pliku tekstowym[6] można załadować funkcją `read.table`. Popatrzmy na przykładowy plik tekstowy z tabelą:

[6] Plik taki można uzyskać np. w Excelu przez opcję *export*. Oprócz plików z danymi w formie tabel można wczytywać inne pliki - służy do tego funkcja `scan`.

```
Lp Wiek Odpowiedz
1 18 T
2 27 N
3 41 N
4 19 N
5 68 T
```

...który możemy załadować do programu:

```
> dane = read.table("tablica.txt",header=TRUE)
> dane
  Lp Wiek Odpowiedz
1  1   18         T
2  2   27         N
3  3   41         N
4  4   19         N
5  5   68         T
> dane$Lp
[1] 1 2 3 4 5
> dane[[1]]
[1] 1 2 3 4 5
> dane[,2]
[1] 18 27 41 19 68
> dane[2,] # jeden z wierszy
  Lp Wiek Odpowiedz
2  2   27         N
```

Opcja `header` w funkcji `read.table` oznacza, że plik tekstowy zawiera w pierwszym wierszu nazwy kolumn.

Jedną z najważniejszych właściwości obiektów typu *list* i *dataframe* jest możliwość ich przyłączenia (*attach*). Jeśli mamy listę lub ramkę `lista`, zawierającą elementy `a` i `b`, to po jej włączeniu, w środowisku zmienne te istnieją „bezpośrednio" i nie trzeba się odwoływać do nich za pośrednictwem ramki lub listy, aż do czasu jej odłączenia (*detach*):

```
> attach(dane)
> Lp
[1] 1 2 3 4 5
> Wiek
[1] 18 27 41 19 68
> Odpowiedz
[1] T N N N T
Levels: N T
> detach(dane)
> Wiek # po odłączeniu, zmienna nie jest dostępna bezpośrednio,
Error: Object "Wiek" not found
> dane$Wiek # tylko pośrednio
[1] 18 27 41 19 68
```

4. Rozkłady zmiennych

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 literą `d` uzyskujemy funkcję gęstości rozkładu. Analogicznie poprzedzając nazwę literą `p` uzyskujemy wartości dystrybuanty. Funkcja kwantylowa (poprzedzona `q`) podaje taki kwantyl, który po lewej stronie wykresu gęstości zawiera określone prawdopodobieństwo. Generator liczb losowych dostępny jest przy poprzedzeniu nazwy literą `r`.

Funkcje te pozwalają na traktowanie pakietu R jako zestawu bardzo dokładnych tablic statystycznych. Popatrzmy na przykłady:

```
> dnorm(0) # gęstość rozkładu normalnego w zerze
[1] 0.3989423
> pnorm(1)-pnorm(-1) # ile wartości mieści się w N(0,1) w przedziale (0,1) ?
[1] 0.6826895
> qt(0.995,5) # wartość krytyczna t-Studenta dla 99% i 5 stopni swobody
[1] 4.032143
> qchisq(0.01,5) # wartość krytyczna chi-kwadrat dla 5 st. swobody i 99%
[1] 0.5542981
> dpois(0,0.1) # wartość prawdop. Poissona dla lambda 0.1 i n=0
[1] 0.9048374
> qf(0.99,5,6) # wartość krytyczna testu F dla n1=5, n2=6
[1] 8.745895
```

Kilku słów wymaga wartość 0.995 (nie 0.99) w wywołaniu funkcji rozkładu t-Studenta. Rozkład ten jest zwykle stosowany w kontekście dwustronnym, dlatego obszar krytyczny dzielimy równomiernie na obu „końcach” rozkładu. 99% ufności oznacza, że krytyczny 1% jest podzielony na 2 końce i zawiera się w przedziałach $(0, 0.05)$ oraz $(0.995, 1)$. Wartość tablicowa jest kwantylem obliczonym dla takiego właśnie prawdopodobieństwa. Analogicznie np. dla 95% będzie to 0.975, a dla 99.9% — 0.9995.

Korzystając z funkcji generatora losowego można generować dowolne ciągi danych do późniejszej analizy. Wygenerujmy przykładowy zestaw 30 liczb o średniej 50 i odchyleniu standardowym 5, posiadających rozkład normalny:

```
> rnorm(30,50,5)
 [1] 53.43194 58.74333 53.27320 46.42251 53.93869 44.80035 55.57112 43.65090
 [9] 46.78265 55.88207 49.68947 52.65945 55.72740 48.75954 48.16239 50.89369
[17] 51.23270 47.14778 57.83292 45.67989 45.98016 50.45368 44.41436 44.24023
[25] 50.98059 48.69967 53.32837 48.09720 52.57135 49.64967
>
```

Oczywiście funkcja ta wygeneruje za każdym razem całkowicie inne wartości, dlatego też prowadząc analizy należy je zapamiętać w zmiennej, a potem używać tej zmiennej w dalszych operacjach.

Warto przy okazji wspomnieć o funkcji `sample`, generującej wektor danych wylosowanych z innego wektora. Np. funkcja `sample(1:6,10,replace=T)` symuluje 10 rzutów kostką (losowanie ze zbioru 1:6), a dane mogą się powtarzać. Jeśli nie jest podana liczba losowanych danych (np. `sample(1:6)`), funkcja generuje losową permutację wektora podanego jako parametr.

5. Wykresy

5.1. Wykresy słupkowe

Wykresy słupkowe osiągalne są w R poleceniem `barplot`. Popatrzmy na przykładowe wykresy danych, dotyczących spędzanych godzin przed telewizorem i komputerem u 100 osób. Wygenerujemy te dane używając funkcji `rnorm`, zaokrąglając do liczb całkowitych funkcją `round`. Funkcja `abs` zwraca wartości absolutne z danych, gdyż mogą wśród nich trafić się liczby ujemne.

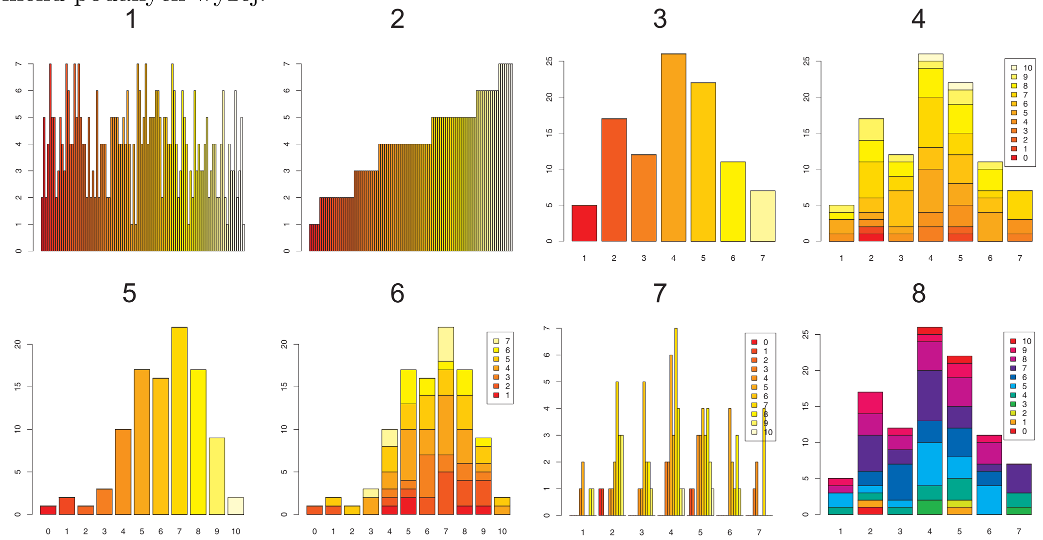
```
> tv = abs(round(rnorm(100,4,1.5)))
> komp = abs(round(rnorm(100,6,2)))
> tv
  [1] 2 5 2 4 7 5 5 2 3 5 4 3 7 6 4 4 7 5 7 5 4 4 2 5 2 3 2 6 2 4 4 4 2 2 5 5 5
 [38] 5 4 4 5 4 6 4 1 4 1 7 6 4 5 7 4 5 5 5 6 6 5 2 5 5 4 4 7 6 4 2 3 6 5 4 1 3
 [75] 2 6 3 4 3 3 4 5 2 3 4 4 2 2 4 6 2 1 4 3 3 6 2 3 5 1
> komp
  [1]  7  9  5 10  4  5  8  6  8  6  6  6  3  6  3  5  7  6  7  4  8  5  7  5  9
 [26]  6  0  9  8  7  8  5  1  6 10  7  8  4  3  4  2  7  7  9  5  7  9  7  5  6
 [51]  6  4  8  1  7  5  8  8  6  9  7  8  7  5  7  5  7  8  6  6  8  5  5  4  7
 [76]  5  6  7  7  7  8  9  4  9  5  4  9  7  7  8  8  8  6  8  6  5  7  5  4  4
```

Następnie będziemy produkować różne przykładowe wykresy komendą barplot:

1. `barplot(tv)` — najprostszy wykres, zawierający 100 słupków odpowiadających poszczególnym danym.
2. `barplot(sort(tv))` — wykres posortowanych danych, pozwalający na wizualną ocenę ilości poszczególnych odpowiedzi.
3. `barplot(table(tv,tv))` — ilość osób oglądających telewizję godzinę, dwie etc.
4. `barplot(table(komp,tv),legend.text=T)` — wykres analogiczny, lecz słupki pokazują dodatkowo poszczególne odpowiedzi dotyczące komputera (są podzielone). Opcjonalny parametr `legend.text` pozwala na zamieszczenie legendy.
5. `barplot(table(komp,komp))` — taki sam wykres, lecz dla czasu spędzonego przed komputerem.
6. `barplot(table(tv,komp),legend.text=T)` — analogiczny, z podzielonymi słupkami proporcjonalnie do czasu oglądania telewizji.
7. `barplot(table(komp,tv),beside=T,legend.text=T)` — te same dane, jednak słupki położone są obok siebie, a nie na sobie.
8. `barplot(table(komp,tv),col=rainbow(10),legend.text=T)` — tutaj listę tęczowych kolorów generujemy funkcją `rainbow`.

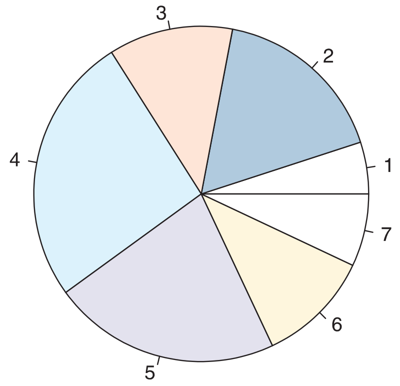
Istnieje kilka funkcji do generowania zestawów kolorów, analogicznie do `rainbow`, np. `heat.colors, terrain.colors, topo.colors, cm.colors`. Wektory szarości uzyskujemy stosując funkcję `grey`, np. `grey(0:10/10)` to 10 kolorów od szarego do białego. Istnieje też możliwość wyspecyfikowania konkretnej listy kolorów, np. `col=c("red","orange","yellow","green","blue",violet")`.

Przy wyspecyfikowaniu za małej ilości kolory będą się powtarzać. Pełna lista nazw kolorów wyświetlana jest komendą `colors()`. Poniższy rysunek przedstawia wykresy wygenerowane przy użyciu komend podanych wyżej.



5.2. Wykresy kołowe

Wykresy kołowe generujemy komendą `pie`, z analogiczną składnią. Przykładowo, poprzez wywołanie `pie(table(tv))` uzyskamy:



Podobnie jak w przypadku wykresów słupkowych, funkcja pozwala na precyzowanie kolorów, podpisów poszczególnych wycinków etc. Pełna lista opcji jest oczywiście dostępna na stronie pomocy (osiągalnej przez `?pie`).

5.3. Histogramy i wykresy rozrzutu zmiennej

W czasie analizy statystycznej niejednokrotnie zachodzi potrzeba przedstawienia graficznego rozkładu danych statystycznych. Pakiet R pozwala na zobrazowanie danych na wykresach w wieloraki sposób.

W tym miejscu należy zaznaczyć, że zarówno sam R (czyli pakiet „base"), jak i dodatkowe pakiety (*packages*) zawierają nie tylko funkcje statystyczne, ale przykładowe zestawy realnych danych. Zestawy te są dołączone w celach edukacyjnych, aby użytkownik mógł prowadzić na nich przykładowe obliczenia.

W pierwszej kolejności załadujemy komendą `data(rivers)` dane dotyczące długości rzek w USA. Najprostszym sposobem zobrazowania rozkładu tych danych jest wykres tekstowy *stem-and-leaf*:

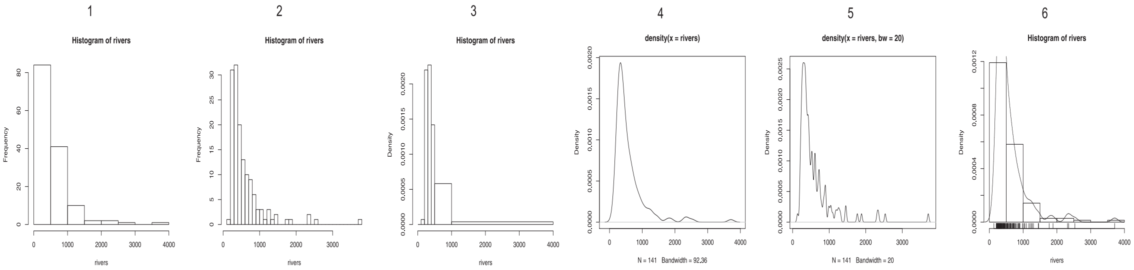
```
> stem(rivers)
   0 | 4
   2 | 011223334555566667778888899900001111223333344455555666688888999
   4 | 111222333445566779001233344567
   6 | 000112233578012234468
   8 | 045790018
  10 | 04507
  12 | 1471
  14 | 56
  16 | 7
  18 | 9
  20 |
  22 | 25
  24 | 3
  26 |
  28 |
  30 |
  32 |
  34 |
  36 | 1
```

Wykres ten nie jest często stosowany, aczkolwiek warto go znać — bardzo łatwo wkleić go wszędzie tam, gdzie operujemy tylko tekstem (np. e-mail). Typowo „graficzne" wykresy to najczęściej histogramy i tzw. wykresy gęstości rozkładu. Oto kilka przykładów:

1. `hist(rivers)` — zwykły histogram.
2. `hist(rivers,40)` — tak samo, lecz narzucamy liczbę „przedziałów" histogramu.
3. `hist(rivers,breaks=c(10,20,50,100,200,300,400,500,1000,4000))` — tutaj narzucamy konkretne miejsca, w których kończą się przedziały.
4. `plot(density(rivers))` — wykres gęstości.
5. `plot(density(rivers,bw=20))` — wykres gęstości z narzuconą szerokością pasma (*bandwidth*).
6. `hist(rivers,prob=T);lines(density(rivers));rug(rivers)` — wykres kombinowany. Na histogram nanosimy wykres gęstości, a następnie komendą `rug` dodajemy wykres rozproszenia 1-D na osi x [7]. Opcja `prob=T` powoduje, że na osi x histogramu oznaczone są wartości prawdopodobieństwa (celem dopasowania do wykresu gęstości).

Rezultaty tych poleceń powinny wyglądać następująco:

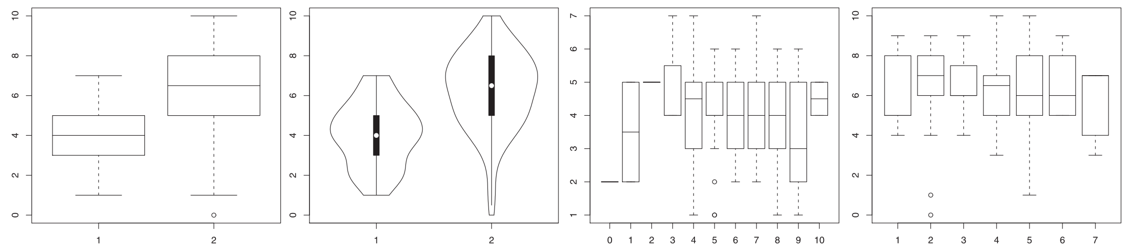
[7] Samodzielny wykres 1-D, analogiczny do `rug`, uzyskuje się poleceniem `stripchart`. Polecam również obejrzenie wykresu uzyskanego przez `stripchart(tv ~ komp)`



Czasem zachodzi potrzeba porównania na jednym wykresie rozkładu kilku zmiennych. Służą do tego wykresy „pudełko z wąsami” (*boxplot*) oraz wykresy „skrzypcowe” (*violin plot*). Wykonanie wykresu skrzypcowego wymaga załadowania pakietu poleceniem `library(vioplot)`. Pakiet ten (oraz pakiet `sm`) powinny być wcześniej zainstalowane. W środowisku *Windows* czynimy to wybierając opcję „packages/install packages from CRAN”. Popatrzmy na przykładowe wykresy:

1. `boxplot(tv,komp)` — porównanie rozrzutu czasu spędzanego przy telewzji i komputerze.
2. `vioplot(tv,komp)` — analogicznie, lecz wykresem „skrzypcowym”.
3. `boxplot(tv ~ komp)` — wykres rozrzutu oglądalności telewizji dla poszczególnych liczb godzin spędzonych przed komputerem.
4. `boxplot(komp ~ tv)` — tak samo, lecz odwrotnie.

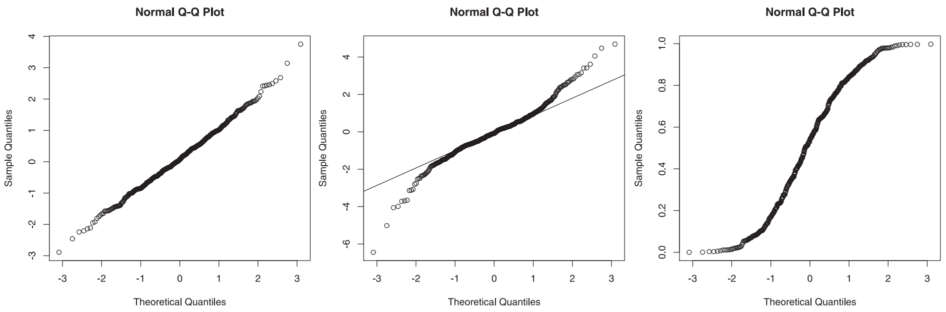
Rezultaty powyższych poleceń będą wyglądać tak:



Niekiedy rozkład danych doświadczalnych prezentuje się na tzw. wykresie kwantylowo-normalnym (*quantile-normal plot*). Jest to zależność pomiędzy wartościami zmiennej, a kwantylami rozkładu normalnego. W idealnym przypadku, jeśli rozkład zmiennej jest czysto normalny, wykres ten przedstawia linię prostą. Popatrzmy na trzy przykłady:

1. `qqnorm(rnorm(500))` — wykres dla 500 liczb o rozkładzie normalnym.
2. `qq=rt(500,1);qqnorm(qq);qqline(qq)` — wykres dla 500 liczb o rozkładzie t-Studenta o 5 stopniach swobody. Funkcja `qqline` dodaje do wykresu linię prostą „uśredniającą” rozkład [8].
3. `qqnorm(runif(500,0,1))` — wykres dla 500 liczb, równomiernie rozmieszczonych w przedziale $(0, 1)$.

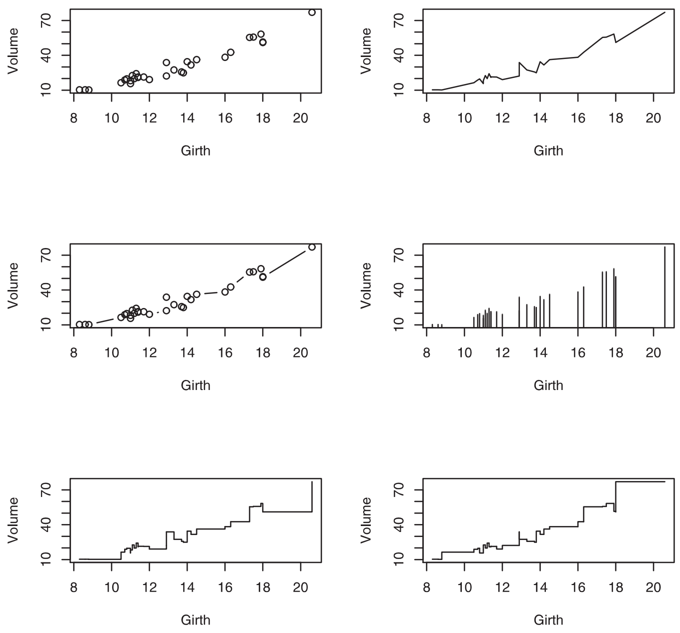
[8] Dlaczego najpierw zapisujemy wynik funkcji `rt` w zmiennej? Uważny czytelnik powinien już wiedzieć. . .



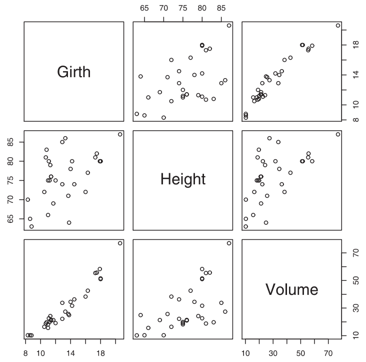
5.4. Wykresy dwuwymiarowe

Funkcja `plot` jest uniwersalnym narzędziem pakietu R, a wykresy generowane przy jej użyciu mogą być skrajnie odmienne, w zależności od argumentów tej funkcji. W najprostszym przypadku funkcja ta generuje dwuwymiarowy wykres punktowy, przedstawiający zależność jednej zmiennej od drugiej. Załadujmy komendą `data(trees);attach(trees)` dane dotyczące wysokości, średnicy i objętości pnia 31 drzew, po czym wykonajmy wykresy:

1. `plot(Girth,Volume,type="p")` — wykres punktowy.
2. `plot(Girth,Volume,type="l")` — wykres liniowy.
3. `plot(Girth,Volume,type="b")` — wykres punktowo-liniowy.
4. `plot(Girth,Volume,type="h")` — wykres „kreskowy”.
5. `plot(Girth,Volume,type="s")` — wykres schodkowy.
6. `plot(Girth,Volume,type="S")` — wykres schodkowy (o odwrotnych schodkach).



Wykres przedstawiający zależności pomiędzy wszystkimi zmiennymi w parach otrzymamy poleceniem `pairs(trees)`:



5.5. Wykresy funkcji

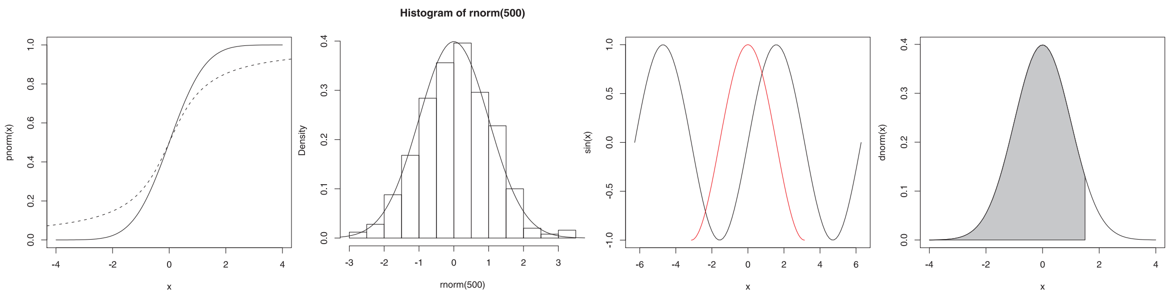
R umożliwia rysowanie wykresów dowolnych funkcji matematycznych, przy czym mogą one stanowić samodzielne wykresy, lub też być dodawane do istniejących. Służy do tego funkcja `curve`. Oto kilka przykładów:

1. `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.

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

3. `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 $(-2\pi, 2\pi)$, a następnie dodaje wykres $y = \cos x$ czerwoną linią, w przedziale $(-\pi, \pi)$.

4. `xv=seq(-4,1.5,length=50); dv=dnorm(xv); curve(dnorm(x),xlim=c(-4,4)); polygon(c(xv,rev(xv)),c(rep(0,50),rev(dv)),col="gray")` — to ciekawy przykład do samodzielnej analizy. Funkcja `polygon` powoduje narysowanie wielokąta o określonych współrzędnych x i y. W tym przypadku stosujemy ją do bardzo nietypowego zadania -zaciemnienia powierzchni pod krzywą gęstości w przedziale $(-4, 1.5)$.



5.6. Ogólne parametry funkcji rysujących

Wszystkie funkcje rysujące mają bogaty zestaw wspólnych parametrów, których dokumentacja jest dostępna przez `?par`. Najważniejsze parametry, które można ustawić w każdym wykresie to:

1. `ask` — ustawienie na TRUE powoduje, że przed rozpoczęciem rysowania użytkownik będzie musiał nacisnąć dowolny klawisz. Jest to przydatne przy rysowaniu wykresów etapami i zapisywaniu każdego z etapów.

2. `bg` — kolor tła wykresu.
3. `col` — kolor wykresu (linii, słupków etc. w zależności od kontekstu).
4. `col.axis` — kolor osi.
5. `col.lab` — kolor legendy osi.
6. `col.main` — kolor głównego tytułu.
7. `col.sub` — kolor podtytułów.
8. `fg` — kolor powierzchni wykresu (*foreground*.
9. `font` — rodzaj czcionki. 1 to krój normalny, 2 — pogrubiony, 3 — kursywa, 4 — pogrubiona kursywa.
10. `lty` — rodzaj linii. 1 to ciągła, 2 — kreskowana, 3 — kropkowana, 4 — kropka-kreska, 5 — długa kreska, 6 — podwójna kreska.
11. `lwd` — grubość linii, standardowo 1.
12. `mar` — marginesy wykresu, standardowo `c(5, 4, 4, 2) + 0.1`.
13. `mfcol, mfrow` — parametry umożliwiające rysowanie kilku wykresów na jednym „ekranie" w formie tabeli. Określają liczbę wierczy i kolumn tej tabeli.
14. `new` — ustawiony na TRUE powoduje narysowanie wykresu na poprzednim, bez czyszczenia ekranu.
15. `pch` — rodzaj symbolu rysowanego na wykresie do oznaczenia punktu. Wartość może być liczbą całkowitą lub jednym znakiem. Np. 2 oznacza drugi symbol ze standardowego zestawu, zaś `"X"` — rysowanie liter X.
16. `xlog, ylog` — ustawiając na TRUE uzyskujemy logarytmiczną skalę osi x lub y.
17. `xlim, ylim` — przedziały osi x i y (przykłady podano już wcześniej).

6. Najczęściej stosowane testy statystyczne

6.1. Testy dla jednej próby

Do najczęściej stosowanych testów statystycznych operujących na jednym zbiorze danych należą: 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) oraz test z (sprawdzający, czy próba pochodzi z populacji o rozkładzie normalnym, gdy znane jest σ tej populacji). Przy wnioskowaniu dla jednej średniej, w razie stwierdzenia rozkładu innego niż normalny, zamiast testu t stosuje się test rang Wilcoxona. Rozważmy np. wyniki np. 6 analiz chemicznych — test posłuży do sprawdzenia, czy mogą pochodzić z populacji o średniej równej 100:

```
> dane = c(96.19,98.07,103.53,99.81,101.60,103.44)
> shapiro.test(dane) # czy rozkład jest normalny?
        Shapiro-Wilk normality test
```

```
data:  dane
W = 0.9271, p-value = 0.5581
> t.test(dane,mu=100) # tak, a zatem sprawdzamy testem Studenta
        One Sample t-test
data:  dane
t = 0.3634, df = 5, p-value = 0.7311
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
  97.32793 103.55207
sample estimates:
mean of x
   100.44
> wilcox.test(dane,mu=100) # tak byśmy sprawdzali, gdyby nie był
        Wilcoxon signed rank test
data:  dane
V = 11, p-value = 1
alternative hypothesis: true mu is not equal to 100
```

W przypadku testu z nie mamy gotowej funkcji w R. Jednak przedział ufności można bardzo prosto policzyć ręcznie dla dowolnego α. Załóżmy, że $\sigma = 3$, zaś $\alpha = 0.05$:

```
> ci = qnorm(1-0.05/2)
> ci
[1] 1.959964
> s = 3/sqrt(length(dane))
> s
[1] 1.224745
> mean(dane) + c(-s*ci,s*ci)
[1]  98.03954 102.84046
```

6.2. Testy dla dwóch prób

Podstawowymi testami dla dwóch zbiorów danych są: test F-Snedecöra na jednorodność wariancji (sprawdzający, czy wariancje obu prób różnią się istotnie między sobą), test t-Studenta dla dwóch średnich (porównujący, czy próby pochodzą z tej samej populacji; wykonywany w razie jednorodnej wariancji) oraz test U-Manna-Whitneya, będący odmianą testu Wilcoxona, służący do stwierdzenia równości średnich w razie uprzedniego wykrycia niejednorodności wariancji. Do poprzedniego zbioru danych dodamy drugi zbiór `dane2` i przeprowadzimy te testy:

```
> dane2 = c(99.70,99.79,101.14,99.32,99.27,101.29)
> var.test(dane,dane2) # czy jest jednorodność wariancji?
        F test to compare two variances
data:  dane and dane2
F = 10.8575, num df = 5, denom df = 5, p-value = 0.02045
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  1.519295 77.591557
sample estimates:
```

```
ratio of variances
         10.85746
> wilcox.test(dane,dane2) # jednak różnice wariancji, więc test Wilcoxona
        Wilcoxon rank sum test
data:  dane and dane2
W = 22, p-value = 0.5887 # dane się istotnie nie różnią
alternative hypothesis: true mu is not equal to 0
> t.test(dane,dane2) # a tak byśmy zrobili, gdyby różnic nie było
        Welch Two Sample t-test
data:  dane and dane2
t = 0.2806, df = 5.913, p-value = 0.7886
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.751801  3.461801
sample estimates:
mean of x mean of y
  100.440   100.085
```

Jeśli zachodzi potrzeba wykonania testu z innym α niż 0.05, korzystamy z opcji `conf.level` każdego testu (standardowo jest to 0.95). W funkcji `t.test` można również ustawić `paired=TRUE` i w ten sposób policzyć test dla danych powiązanych parami. Warto też zwrócić uwagę na parametr `conf.int` w funkcji `wilcox.test`, który pozwala na obliczenie przedziału ufności dla mediany.

6.3. Testy dla większej ilości prób

W przypadku analizy statystycznej większej grupy danych, np. wyników analizy chemicznej wykonanej trzema metodami, konieczne jest umieszczenie wszytkich danych w jednej ramce (*dataframe*). Dołóżmy jeszcze kolejne 6 wyników do wektora `dane3` i umieśćmy wszystko w ramce:

```
> dane3 = c(91.50,96.74,108.17,94.22,99.18,105.48)
> danex = data.frame(wyniki=c(dane,dane2,dane3),metoda=rep(1:3,each=6))
> danex
   wyniki metoda
1   96.19      1
2   98.07      1
3  103.53      1
4   99.81      1
5  101.60      1
6  103.44      1
7   99.70      2
8   99.79      2
9  101.14      2
10  99.32      2
11  99.27      2
12 101.29      2
13  91.50      3
14  96.74      3
```

```
15 108.17      3
16  94.22      3
17  99.18      3
18 105.48      3
```

Na tak wykonanej ramce możemy wykonać już poszczególne testy. Do porównania jednorodności wariancji służy tutaj test Bartletta, który jest rozwinięciem testu F-Snedecöra na większą liczbę prób. W razie stwierdzenia istotnych różnic w wariancji badamy istotne różnice pomiędzy grupami wyników testem Kruskala-Wallisa. Jeśli nie ma podstaw do odrzucenia hipotezy o jednorodności wariancji, rekomendowanym testem jest najprostszy wariant ANOVA:

```
> bartlett.test(wyniki ~ metoda,danex) # czy jest niejednorodność?
        Bartlett test for homogeneity of variances
data:  wyniki by metoda
Bartlett's K-squared = 13.0145, df = 2, p-value = 0.001493
> kruskal.test(wyniki~metoda,danex) # jest, więc Kruskar-Wallis
        Kruskal-Wallis rank sum test
data:  wyniki by metoda
Kruskal-Wallis chi-squared = 0.7836, df = 2, p-value = 0.6758
> anova(aov(wyniki~metoda,danex)) # a tak, gdyby nie było
Analysis of Variance Table
Response: wyniki
          Df  Sum Sq Mean Sq F value Pr(>F)
metoda     1   4.502   4.502  0.2788 0.6047
Residuals 16 258.325  16.145
```

6.4. Dwuczynnikowa ANOVA

Załóżmy, że oznaczyliśmy w 4 próbkach zawartość jakiejś substancji, każdą próbkę badaliśmy 4 metodami (w sumie 16 wyników). Stosując dwuczynnikowy test ANOVA można sprawdzić, czy wyniki różnią się istotnie między próbkami i między metodami. W tym celu znów tworzymy ramkę zawierającą wyniki oraz odpowiadające im próbki i metody:

```
> dane
      wynik probka metoda
1  46.37278      1      1
2  48.49733      1      2
3  46.30928      1      3
4  43.56900      1      4
5  46.57548      2      1
6  42.92014      2      2
7  47.22845      2      3
8  42.46036      2      4
9  43.71306      3      1
10 47.35283      3      2
11 44.87579      3      3
12 46.06378      3      4
13 47.29096      4      1
```

```
14 44.50382      4      2
15 44.26496      4      3
16 45.35748      4      4
> anova(aov(wynik ~ probka + metoda,dane)) # test bez interakcji
Analysis of Variance Table
Response: wynik
          Df Sum Sq Mean Sq F value Pr(>F)
probka     1  0.643   0.643  0.2013 0.6611
metoda     1  5.050   5.050  1.5810 0.2307
Residuals 13 41.528   3.194
> anova(aov(wynik ~ probka * metoda,dane)) # test z~interakcjami
Analysis of Variance Table
Response: wynik
              Df Sum Sq Mean Sq F value Pr(>F)
probka         1  0.643   0.643  0.1939 0.6675
metoda         1  5.050   5.050  1.5227 0.2408
probka:metoda  1  1.728   1.728  0.5211 0.4842
Residuals     12 39.800   3.317
```

Podany powyżej test z interakcjami nie ma sensu merytorycznego w tym przypadku, pokazano go tylko przykładowo.

6.5. Testy chi-kwadrat dla proporcji

Załóżmy np. że wśród 300 ankietowanych osób odpowiedź „tak" padła w 30 przypadkach. Daje to 10% ankietowanych. Czy można powiedzieć, że wśród ogółu populacji tylko 7% tak odpowiada? Jaki jest przedział ufności dla tego prawdopodobieństwa?

```
> prop.test(30,300,p=0.07)
        1-sample proportions test with continuity correction
data:  30 out of 300, null probability 0.07
X-squared = 3.6994, df = 1, p-value = 0.05443
alternative hypothesis: true p is not equal to 0.07
95 percent confidence interval:
 0.0695477 0.1410547
sample estimates:
  p
0.1
```

Jak widać, przedział ufności zawiera się w granicy 6.9 — 14.1%, zatem 7% zawiera się w nim (p nieznacznie większe niż 0.05).

Innym przykładem jest porównanie kilku takich wyników. Załóżmy, że w innej 200-osobowej grupie odpowiedź padła u 25 osób, a w trzeciej, 500-osobowej, u 40 osób. Czy próby te pochodzą z tej samej populacji?

```
> prop.test(c(30,25,40),c(300,200,500))
        3-sample test for equality of proportions without continuity
        correction
data:  c(30, 25, 40) out of c(300, 200, 500)
```

```
X-squared = 3.4894, df = 2, p-value = 0.1747
alternative hypothesis: two.sided
sample estimates:
prop 1 prop 2 prop 3
 0.100  0.125  0.080
```

Jak widać, test nie wykazał istotnych różnic. Jeśli chcielibyśmy wprowadzić do testu poprawkę Yates'a, dodajemy parametr `cont=TRUE`.

6.5.1. Test chi-kwadrat na zgodność rozkładu

Test χ^2 na zgodność rozkładu jest wbudowany w R pod funkcją `chisq.test`. W przypadku wnioskowania zgodności rozkładu pierwszy wektor zawiera dane, a drugi prawdopodobieństwa ich wystąpienia w testowanym rozkładzie. Załóżmy, że w wyniku 60 rzutów kostką otrzymaliśmy następujące ilości poszczególnych wyników 1 — 6: 6,12,9,11,15,7. Czy różnice pomiędzy wynikami świadczą o tym, że kostka jest nieprawidłowo skonstruowana, czy są przypadkowe?

```
> kostka=c(6,12,9,11,15,7)
> pr=rep(1/6,6) # prawdopodobieństwo każdego rzutu wynosi 1/6
> pr
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
> chisq.test(kostka,p=pr)
        Chi-squared test for given probabilities
data:  kostka
X-squared = 5.6, df = 5, p-value = 0.3471
```

6.6. Test chi-kwadrat na niezależność

Jeśli parametrem funkcji `chisq.test` jest ramka, funkcja przeprowadza test χ^2 na niezależność. Załóżmy, że w grupie pacjentów badanych nowym lekiem 19 pozostało bez poprawy, 41 odnotowało wyraźną poprawę, 60 osób całkowicie wyzdrowiało. W grupie kontrolnej (leczonej dotychczasowymi lekami) wartości te wynosiły odpowiednio 46,19,15. Czy nowy lek faktycznie jest lepszy? Jeśli tak, dane powinny być „zależne”, i tak też jest:

```
> lek=c(19,41,60)
> ctl=c(46,19,15)
> chisq.test(cbind(lek,ctl)) # cbind tworzy ramkę !
        Pearson's Chi-squared test
data:  cbind(lek, ctl)
X-squared = 39.8771, df = 2, p-value = 2.192e-09
```

6.7. Pozostałe testy

Wszystkie testy opisane w większości typowych podręczników statystyki są zaimplementowane w R. Można je odnaleźć wspomnianą już funkcją `help.search`, a dokumentacja dotycząca każdego testu podaje kompletną składnię wraz z przykładami. Ogromna ilość bardziej „specjalistycznych” testów znajduje się w dodatkowych pakietach. Wtedy test można odnaleźć wpisując jego nazwę do wyszukiwarki na stronie programu, a następnie instalując stosowny pakiet.

7. Analiza regresji i korelacji

7.1. Najprostsza regresja liniowa

Załóżmy, że sporządziliśmy krzywą kalibracyjną oznaczania pewnego związku dla stężeń $1, 2, \ldots, 9, 10$ mg/ml. Każde oznaczenie powtarzano trzykrotnie. Wyniki zapisaliśmy w dwóch wextorach `x` i `y`:

```
> x
 [1]  1  1  1  2  2  2  3  3  3  4  4  4  5  5  5  6  6  6  7  7  7  8  8  8  9
[26]  9  9 10 10 10
> y
 [1] 2.00 2.01 2.00 2.42 2.42 2.43 2.72 2.74 2.74 3.00 3.00 2.98 3.23 3.24 3.23
[16] 3.44 3.44 3.45 3.66 3.65 3.62 3.82 3.85 3.83 4.00 4.00 4.00 4.18 4.16 4.18
```

Wykonajmy teraz regresję liniową tych danych:

```
> fit = lm(y~x)
> summary(fit)
Call:
lm(formula = y ~ x)
Residuals:
     Min       1Q   Median       3Q      Max
-0.20545 -0.05887  0.02365  0.07616  0.10784
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.973778   0.038883   50.76   <2e-16 ***
x           0.231677   0.006266   36.97   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.09859 on 28 degrees of freedom
Multiple R-Squared: 0.9799,     Adjusted R-squared: 0.9792
F-statistic:  1367 on 1 and 28 DF,  p-value: < 2.2e-16
```

Otrzymaliśmy w ten sposób obiekt `fit`, zawierający dane przeprowadzonej regresji. 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. Otrzymujemy również współczynnik R^2 oraz R^2_{adj}. Na samym dole podano również wartości testu F na istotność samej korelacji pomiędzy zmiennymi.

Funkcja `lm` posiada kilka dodatkowych opcji. Na przykład dodając parametr `weights=1/x` spowodujemy wykonanie regresji ważonej (*weighted*) według $\frac{1}{x}$. Poza tym możemy podać parametr `subset`, który oznacza wektor „wybierający” pewną część danych z ich szerszego zestawu.

Po przeprowadzeniu regresji mamy dostęp do reszt poprzez `fit$residuals` oraz do wartości „dopasowanych” według wyliczonego równania dla każdego z x przez `fit$fitted`.

Wykonajmy zatem wykresy tej regresji:

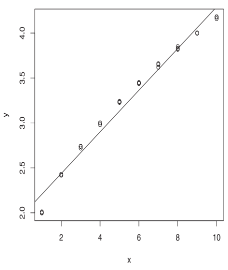
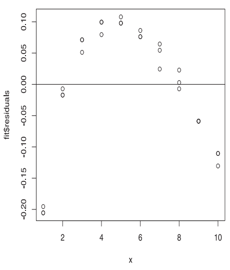
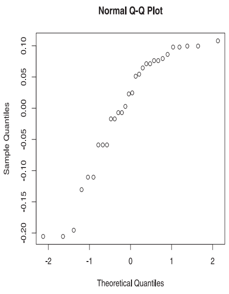
1. `plot(x,y);abline(fit)` — wykres przedstawiający naszą krzywą oraz prostą regresji. Funkcja `abline` dodaje do wykresu linię prostą o zadanych parametrach, w tym przypadku pobranych z obiektu `fit`.

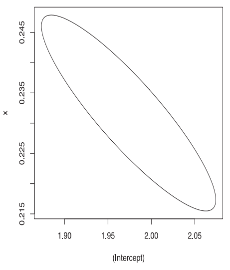
2. `plot(x,fit$residuals);abline(h=0)` — wykres reszt regresji z dodaną linią prostą wzdłuż osi x.

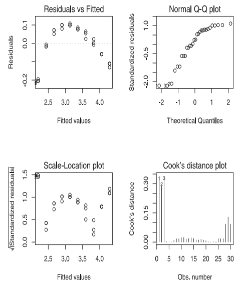
3. `qqnorm(fit$residuals)` — wykres kwantylowy-normalny reszt regresji.

4. `plot(ellipse(fit),type="l")` — wykres „elipsy ufności” estymatorów regresji[9].

5. `par(mfrow=c(2,2));plot(fit);par(mfrow=c(1,1))` — zwykła funkcja `plot` z argumentem będącym rezultatem funkcji `lm` generuje 4 wykresy dotyczące tej regresji. W powyższym przypadku poprzez zmianę parametru `mfrow` uzyskujemy je w jednym oknie.





7.2. Regresja kwadratowa i sześcienna. Testy na liniowość

Na powyższych wykresach widać , że zależność między x,y jest krzywoliniowa. Szczególnie pokazuje to wykres reszt — nie są one losowo rozmieszczone, odbiegają od rozkładu normalnego (można to sprawdzić dodatkowo testem Shapiro-Wilka), wykazują wyraźny trend. Pakiet R zapewnia prosty sposób na wykonanie dwóch ważnych testów statystycznych dotyczących dopasowania w regresji:

1. Test „Lack-of-fit”. Można go stosować wyłącznie w przypadku, gdy dla każdego x mamy kilka wartości y (powtórzone analizy). Polega on na analizie wariancji reszt i stwierdzeniu, czy wariancja międzygrupowa różni się istotnie od wariancji wewnątrzgrupowej.

2. Test dopasowania Mandella — porównanie testem ANOVA regresji prostoliniowej oraz kwadratowej ($y = ax + b + cx^2$) dla tych samych danych.

```
> fitlack = lm(y~factor(x)) # regresja faktora x
> fit2 = lm(y~x+I(x^2)) # model kwadratowy
> anova(fit,fitlack) # test Lack-of-Fit
Analysis of Variance Table
Model 1: y ~ x
Model 2: y ~ factor(x)
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     28 0.27213
2     20 0.00240  8   0.26973 280.97 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> anova(fit,fit2) # Test dopasowania Mandella
Analysis of Variance Table
```

[9] Funkcja `ellipse` znajduje się w pakiecie zewnętrznym, należy go najpierw załadować przez `library(ellipse)`.

```
Model 1: y ~ x
Model 2: y ~ x + I(x^2)
  Res.Df      RSS Df Sum of Sq      F    Pr(>F)
1     28 0.272134
2     27 0.028122  1  0.244012 234.28 7.883e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> summary(fit2) # Estymatory równania kwadratowego
Call:
lm(formula = y ~ x + I(x^2))
Residuals:
      Min        1Q    Median        3Q       Max
-0.056515 -0.021508 -0.001735  0.026038  0.046369
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.7007222  0.0219150   77.61  < 2e-16 ***
x            0.3682045  0.0091526   40.23  < 2e-16 ***
I(x^2)      -0.0124116  0.0008109  -15.31 7.88e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.03227 on 27 degrees of freedom
Multiple R-Squared: 0.9979,     Adjusted R-squared: 0.9978
F-statistic:  6494 on 2 and 27 DF,  p-value: < 2.2e-16
```

Oba testy potwierdziły, że model liniowy nie jest dobry do naszych przykładowych danych. Model kwadratowy jest istotnie lepszy i należałoby rozważyć użycie go do kalibracji. Analogicznie możemy policzyć model sześcienny [10] i sprawdzić, czy jest istotnie lepszy od kwadratowego:

```
> fit3=lm(y~x+I(x^2)+I(x^3)) # model sześcienny
> summary(fit3) # estymatory
Call:
lm(formula = y ~ x + I(x^2) + I(x^3))
Residuals:
      Min        1Q    Median        3Q       Max
-0.023486 -0.012087 -0.001174  0.009785  0.030197
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.5691111  0.0174678  89.829  < 2e-16 ***
x            0.4849365  0.0130929  37.038  < 2e-16 ***
I(x^2)      -0.0377214  0.0027006 -13.968 1.35e-13 ***
I(x^3)       0.0015339  0.0001619   9.472 6.48e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.01559 on 26 degrees of freedom
Multiple R-Squared: 0.9995,     Adjusted R-squared: 0.9995
F-statistic: 1.859e+04 on 3 and 26 DF,  p-value: < 2.2e-16
```

[10] Model wielomianowy jest szczególnym przypadkiem regresji wielokrotnej, którą można również obliczać przez np. `lm(y ~ x + y + z)`.

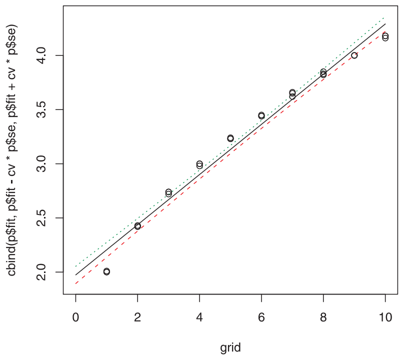
```
> anova(fit2,fit3) # porównanie
Analysis of Variance Table
Model 1: y ~ x + I(x^2)
Model 2: y ~ x + I(x^2) + I(x^3)
  Res.Df       RSS Df Sum of Sq      F   Pr(>F)
1     27 0.0281218
2     26 0.0063185  1 0.0218033 89.718 6.48e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

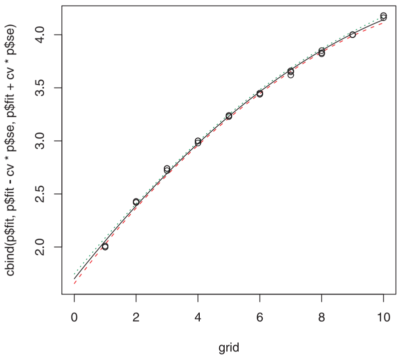
Jak widać, model sześcienny jest jeszcze lepszy! Co więcej, dalsze zwiększanie stopnia wielomianu dalej owocuje lepszym dopasowaniem (wyników nie przytaczam ze względu na objętość opracowania). Jeszcze dla równania piątego stopnia współczynnik przy x^5 oraz ANOVA mają istotność rzędu $p = 0.01$. Dopiero w równaniu stopnia szóstego współczynniki stają się nieistotne, zaś test ANOVA między stopniem piątym a szóstym ma wysokie $p = 0.736$. W praktyce najczęściej najlepszy okazuje się model liniowy lub kwadratowy, rzadko kiedy sześcienny.

Aby narysować wykres zawierający dane oraz krzywą regresji wyższego stopnia, należy się troszkę pomęczyć. Jednak przy okazji istnieje łatwa możliwość dodania „krzywych ufności" dla błędu predykcji. Popatrzmy, jak wygląda dopasowanie liniowe i kwadratowe.

```
> grid=seq(0,10,.01) # tworzymy sekwencję x
> cv=qt(.975,27) # kwantyl dla 95% ufności
> p=predict(fit,data.frame(x=grid),se=T)
> matplot(grid,cbind(p$fit,p$fit-cv*p$se,p$fit+cv*p$se),type="l");points(x,y)
> p=predict(fit2,data.frame(x=grid),se=T)
> matplot(grid,cbind(p$fit,p$fit-cv*p$se,p$fit+cv*p$se),type="l");points(x,y)
```

Funkcja `predict` powoduje obliczenie wartości y dla zadanych x na podstawie równania regresji. W tym przypadku tworzy siatkę współrzędnych krzywej. W zmiennej `p$fit` jest wynik tego dopasowania, a w `p$se` błąd standardowy predykcji. Funkcja `matplot` powoduje narysowanie kilku wykresów na podstawie współrzędnych w zadanej macierzy, w tym przypadku są to wykresy podające y oraz $y \pm S_E t$. Na takim wykresie rysujemy punkty odpowiadające danym poprze `points`.





7.3. Regresja nieliniowa

W praktyce większość danych doświadczalnych daje się wystarczająco dopasować do wielomianu kwadratowego lub sześciennego. Nie interesuje nas wtedy rzeczywista zależność między zmiennymi, gdyż wielomian taki jest wystarczającym przybliżeniem tej zależności (co wynika z twierdzenia

Taylora). Czasami jednak zachodzi potrzeba dopasowania danych do konkretnej nieliniowej funkcji. Umożliwia to polecenie `nls`. W tym przypadku dokonamy dopasowania naszych danych do równania $y = x^m + b$.

```
> fitn=nls(y~x^m+b,start=c(m=0.6,b=1.5)) # musimy zadeklarować
                                           przybliżone wartości startowe
> summary(fitn)
Formula: y ~ x^m + b
Parameters:
   Estimate Std. Error t value Pr(>|t|)
m 0.5001452  0.0008658   577.7   <2e-16 ***
b 1.0006124  0.0038929   257.0   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.01097 on 28 degrees of freedom
Correlation of Parameter Estimates:
       m
b -0.8575
```

Na pewno każdy czytelnik zauważy, że estymatory są bardzo „okrągłe”. W tym jest cały sekret — analizowane przez nas dane specjalnie dobrałem na podstawie funkcji $y = x^{\frac{1}{2}} + 1$. Dlatego też były one tak „paskudne” w dopasowaniu do wielomianów niższego stopnia w poprzednim rozdziale.

Pakiet R dysponuje całym zestawem funkcji dopasowujących do najczęściej spotykanych modeli nieliniowych. Zaczynają się one literami `SS`. Dla przykładu dopasujmy nasze dane do modelu Michaelisa-Menten:

```
> fitmm = nls(y ~ SSmicmen(x,V,K))
> summary(fitmm)
Formula: y ~ SSmicmen(x, V, K)
Parameters:
  Estimate Std. Error t value Pr(>|t|)
V   4.6586     0.1215   38.36  < 2e-16 ***
K   1.8435     0.1764   10.45 3.62e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.1847 on 28 degrees of freedom
Correlation of Parameter Estimates:
      V
K 0.9206
```

Do wyboru optymalnego modelu regresji służy gotowa funkcja, wyliczająca współczynnik AIC (*Akaike's Information Criterion*). Jest to najczęściej wybierane kryterium optymalnego dopasowania. Wartość tego współczynnika jest zależna nie tylko od sumy kwadratów reszt, ale również od ilości zmiennych w równaniu. Dlatego też zwiększając rząd wielomianu, mimo iż suma kwadratów reszt zawsze będzie maleć, od pewnego momentu współczynnik AIC zacznie rosnąć (i to będzie optymalny stopień wielomianu). Użycie tego współczynnika ma również związek z niemożnością stosowania współczynnika determinacji R^2 dla modeli nieliniowych. Zobaczmy zatem który model ma najmniejszy AIC:

```
> AIC(fit,fit2,fit3,fit4,fit5,fit6,fitn,fitmm)
      df        AIC
fit    3  -49.94341
fit2   4 -116.03595
fit3   5 -158.82781
fit4   6 -175.68776
fit5   7 -181.84331
fit6   8 -179.99476
fitn   2 -183.69302
fitmm  2  -14.27206
```

W przypadku regresji wielomianowej potwierdzają się wcześniejsze wyniki — wielomian 5 stopnia ma najmniejszą wartość, a 6 stopnia nieco większą. Zupełnie najmniejszą wartość ma model nieliniowy $y = x^m + b$ (co się zgadza, w związku ze sposobem wygenerowania tych danych). Model Michaelisa-Menten okazał się jeszcze gorszy, niż zwykła regresja liniowa.

8. Co dalej?

Zapewne większość czytelników jest zaskoczona, że to już koniec. Ale to nieprawda. To dopiero początek. Niniejszy tekst zaznajomił z podstawowymi problemami dotyczącymi stosowania R i jest to wręcz wierzchołek góry lodowej. Dlatego na zakończenie doszedłem do wniosku, że trzeba napisać kilka słów dotyczących zaawansowanej nauki środowiska R.

Autorzy R podkreślają wielokrotnie, że powinno się nazywać ich program nie „pakietem", co sugerowałoby zamknięty program do konkretnego celu, ale „językiem", bądź „środowiskiem". W pełni na to miano zasługuje. W środowisku R można tworzyć własne polecenia i funkcje, co zdecydowanie upraszcza wykonywanie skomplikowanych, a jednocześnie rutynowych obliczeń. R posiada wszystkie struktury znane z innych języków, jak pętle, instrukcje warunkowe etc. Jest więc całkowicie otwarty, a liczba dostępnych pakietów i rozszerzeń świadczy o możliwości zaadaptowania do wielu zaawansowanych zadań.

Pisząc to opracowanie długo zastanawiałem się nad tym, czy umieszczać w nim podstawy programowania w R. Po długich namysłach zrezygnowałem z tego. Chciałem pokazać, że podstawowe obliczenia statystyczne można wykonać bez znajomości programowania w R, nawet bez znajomości jakiegokolwiek oprogramowania. Natomiast każdy, kto w życiu programował, nauczy się struktur programistycznych R w ciągu jednego popołudnia z oryginalnej dokumentacji.

Poniżej przedstawiam interesujące dokumenty anglojęzyczne dotyczące R, dostępne bezpłatnie w sieci:

1. *„An introduction to R"* - to oficjalna dokumentacja środowiska, która jest doskonałym kompendium teoretycznym. Omawia wszystkie struktury środowiska R, programowanie, oraz import i eksport danych.

2. *„Using R for Data Analysis and Graphics - An introduction"* (J.H.Maindonald) — jest doskonałym uzupełnieniem oficjalnej dokumentacji. Opisuje zastosowanie R do różnych typowych zadań statystycznych.

3. *„simpleR - Using R for Introductory Statistics"* (J.Verzani) —bardzo obszerne opracowanie dotyczące podstaw statystyki z użyciem R. Zawiera wyczerpujące przykłady oraz zadania do samodzielnego wykonania.

4. *„R for beginners”* (E.Paradis) — krótkie opracowanie omawiające podstawowe funkcje arytmetyczne, graficzne oraz programowanie.

5. *„Practical Regression and ANOVA using R”* (J.Faraway) — bardzo obszerny tekst o różnych metodach regresji i analizy wariancji, z konkretnymi przykładami.

6. Dodatki do dzieła *„An R and S-PLUS Companion to Applied Regression”* Johna Foxa. Sama praca jest w sieci niedostępna i należy ją kupić. Dodatki są natomiast do pobrania. Dotyczą regresji nieparametrycznej, nieliniowej, regresji danych zmiennych w czasie etc.

Mam nadzieję, że mój krótki tekst oraz powyższe dzieła przyczynią się do szybkiego „zaprzyjaźnienia” z R każdego użytkownika. Życzę tego każdemu, aby szybko docenił możliwości pakietu i stosował go w codziennej pracy.

Powodzenia!