

# Wprowadzenie do środowiska programistycznego R

Agnieszka Goroncy



**UNIWERSYTET  
MIKOŁAJA KOPERNIKA  
W TORUNIU**

Wydział Matematyki  
i Informatyki

# Czym jest R?

R jest **darmowym** środowiskiem programistycznym, które ma niezliczone możliwości zarówno obliczeniowe jak i graficzne.

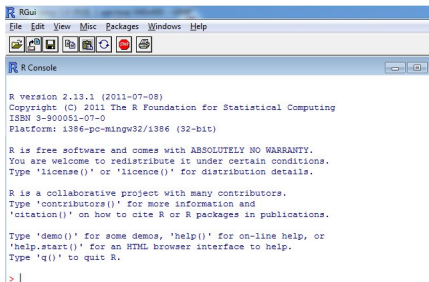
R wykorzystywany jest przede wszystkim do analizy danych różnego rodzaju: finansowych, biologicznych, medycznych, przemysłowych. Korzystają z niego zarówno wiodące ośrodki akademickie, jak i instytucje oraz firmy komercyjne, oferujące kompleksową analizę statystyczną wspierającą krytyczne decyzje biznesowe.

R **w ostatnich latach stał się standardem wśród statystyków i analityków danych (DATA SCIENTISTS!).**

**Główna strona projektu R:**

<http://www.r-project.org/>  
<http://www.rstudio.com/>

# Graficzny interfejs użytkownika i konsola R



```
RGui
File Edit View Misc Packages Windows Help

R Console

R version 2.13.1 (2011-07-08)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

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 or R packages in publications.

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

> |
```

Pierwszą rzeczą, którą warto zrobić po uruchomieniu R, jest ustawienie katalogu roboczego poprzez wybranie z menu **File -> Change dir....**

**Polecenia** R wpisuje się w konsoli po znaku '>', np.  
> 7 - 4

Jeżeli polecenie jest długie i wykracza poza pierwszą linię, wówczas R, czekając na dalszą część, w następnej linii wyświetli znak "+".

```
> x<-c(1,5,7,22,34,55,6,2,5,12,56,37,711,832,47,  
+ 4,5,5,5)
```

**Komentarze** wpisuje się w konsoli po znaku '#', np.  
> # to jest komentarz

Przydatną cechą R jest **zapamiętywanie wpisywanych poleceń**, które można ponownie odtwarzać przy użyciu klawiszy strzałek ↑, ↓.

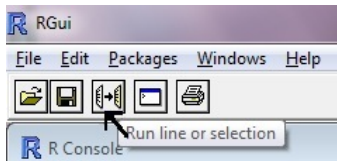
Proszę o testowanie poleceń, które pojawią się w tej i kolejnych prezentacjach, poprzez wpisanie ich do konsoli R i obejrzenie efektu ich działania!

# Skrypty w R

Zestaw często uruchamianych poleceń bądź funkcji warto zapisać w pliku z rozszerzeniem .R. Każde polecenie powinno zaczynać się od nowej linii.

Tak zapisany skrypt można wczytać do R za pomocą menu **File -> Open Script...** **Ctrl+O** bądź poprzez kliknięcie na pierwszą od lewej ikonę znajdującą się pod menu.

Aby **uruchomić polecenie znajdujące się w skrypcie**, należy ustawić kursor w linii, w której się ono znajduje, bądź zaznaczyć myszką grupę poleceń do wykonania i nacisnąć trzecią od lewej strony ikonę znajdującą się pod menu:



# Pakiety R

**Pakiet** jest **zbiorem funkcji, plików danych i plików pomocy powiązanych ze sobą**. Podczas instalacji R instalowane są również standardowe pakiety. Nie wszystkie jednak są domyślnie **ładowane**, czyli dostępne. Domyślnie w R załadowanych jest kilka pakietów podstawowych, których listę można obejrzeć, wpisując w okno konsoli polecenie

```
> getOption("defaultPackages")
```

Aby obejrzeć listę **aktualnie załadowanych pakietów**, należy wpisać polecenie

```
> (.packages())
```

Aby obejrzeć listę **wszystkich dostępnych pakietów, które są zainstalowane**, (nie wszystkie mogą być załadowane), należy wpisać polecenie

```
> (.packages(all.available=TRUE))  
bądź  
> library()
```

W tym drugim przypadku otworzy się nam nowe okno z listą pakietów możliwych do załadowania oraz ich krótkim opisem.

# Ładowanie pakietów w R

Aby **obejrzeć podstawowe informacje na temat wybranego pakietu**, należy użyć polecenia `library`, wywołanego z argumentem `help="nazwa_pakietu"`, np.

```
> library(help="stats")
```

Jeżeli będziemy próbować użyć funkcji, która dostarczana jest z niezaladowanym aktualnie pakietem, R zgłosi błąd.

Aby **załadować pakiet do R**, należy wywołać funkcję `library` z argumentem będącym nazwą pakietu, np.:

```
> library(grid)
```

# Instalacja pakietów w R

Podczas pracy z R większości użytkowników wystarczą domyślnie zainstalowane pakiety. Jeżeli jednak chcesz skorzystać z pakietu niestandardowego, musisz go pobrać i zainstalować. Podstawowym źródłem pakietów jest tzw. CRAN (Comprehensive R Archive Network), który obecnie zawiera ponad 3 tys. pakietów, m.in. tych dodanych przez użytkowników R na całym świecie.

Listę dostępnych pakietów można znaleźć np. na stronie

<http://cran.at.r-project.org/>

Aby **zainstalować wybrany pakiet**, należy wpisać w konsoli R następujące polecenie

```
> install.packages("nazwa_pakietu")
```

bądź wybrać element menu **Packages -> Install package(s)...** Jeżeli instalujesz pakiet po raz pierwszy w danej sesji, musisz wybrać źródło (np. Germany (Regensburg)). Następnie należy wybrać pakiet z listy i kliknąć OK.

Pamiętaj, aby po zainstalowaniu pakietu załadować go do biblioteki R!



# Przykładowe zbiory danych w R

Każdy pakiet posiada przykładowe zbiory z danymi do analizy. Aby **wyświetlić listę aktualnie dostępnych baz danych** wraz z ich krótkim opisem, należy wywołać polecenie

```
> data()
```

Aby **wyświetlić dane znajdujące się w wybranej bazie**, należy wywołać ją poprzez nazwę, np.

```
> Orange
```

Aby **sprawdzić rozmiar bazy danych**, należy wywołać funkcję `dim()`:

```
> dim(Orange)
```

```
> dim(Orange)[1] # liczba obserwacji
```

```
> dim(Orange)[2] # liczba zmiennych
```

Aby **uzyskać nazwy zmiennych** znajdujących się w bazie, wystarczy wywołać funkcję `names()`:

```
> names(Orange)
```

# Podłączanie i odłączanie do bazy danych R

W bazie Orange znajdują się 3 zmienne: Tree, age, circumference. Aby dokonać ich analizy, należy najpierw **podłączyć bazę do R**:

```
> attach(Orange)  
> Tree #teraz już mamy dostęp do zmiennej Tree
```

Po skończonej pracy **bazę należy odłączyć**:

```
> detach(Orange)  
> Tree #R nie rozpoznaje już obiektu 'Tree'
```

Jeżeli nie chcemy podłączać bazy, ale chcemy analizować zmienne się w niej znajdujące, wystarczy odwoływać się do nich poprzez nazwę bazy, znak \$ oraz nazwę zmiennej:

```
> Orange$Tree #nadal możemy pracować na wektorze 'Tree'
```

R oferuje rozbudowaną pomoc dotyczącą zainstalowanych pakietów. Aby uzyskać informacje dotyczące danej funkcji, sposobu jej wywołania, parametrów, należy wywołać jedną z komend:

`help`, `?` z argumentem będącym nazwą funkcji, np.

```
> help(mean)
```

```
> ?sd
```

Jeżeli nie znamy dokładnie nazwy funkcji, którą chcielibyśmy użyć, ale wiemy, jakiej tematyki dotyczy, możemy szukać pomocy przy użyciu polecenia `help.search` lub `??`, np.

```
> help.search("empirical")
```

```
> ??density
```

# R jako kalkulator

Z uwagi na to, że R jest szybki i prosty w obsłudze, może być wykorzystywany jako kalkulator. Zaimplementowane są w nim podstawowe działania i funkcje matematyczne, takie jak:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$  (potęgowanie), `exp` (funkcja eksponencjalna), `log` (funkcja logarytmiczna), `sqrt` (pierwiastek kwadratowy), `abs` (wartość bezwzględna), itp. Oto przykłady:

```
> log(16, 4)
```

```
> log(9, base=3)+sqrt(64)
```

```
> exp(1)+0.25
```

```
> abs(cos(pi))
```

# Struktura danych w R

Podstawowym typem danych w R jest **wektor**, czyli uporządkowany ciąg wielkości (niekoniecznie liczbowych). Jeżeli wpisujemy do konsoli R polecenie zwracające wartość liczbową, R ją wyświetli:

```
> atan(pi)
[1] 1.262627
```

[1] widniejąca przed powyższym wynikiem oznacza, że R interpretuje wynik powyższej operacji jako wektor - jest to indeks pierwszego elementu wektora znajdującego się w pierwszym wierszu.

```
> 1:40
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40
```

Funkcja `c()` umożliwia tworzenie wektora z pojedynczych elementów oraz innych wektorów:

```
> c(1,5,7,9,12)
> c(5:24, 23, 1:4)
```

## Uwaga

Składowymi wektora mogą być zarówno liczby, jak i ciągi znaków.

```
> c(1, 4, 6, "zielony", "niebieski", "biały")
```

# Nazwy obiektów w R

Każdemu obiektowi w R można przypisać nazwę. Do pobierania i nadawania nazwy elementom danego obiektu służy funkcja `names()`:

```
> kolory=c(4,5,24,1)
> names(kolory)=c("zielony", "niebieski", "biały",
+ "czerwony")
> kolory
```

Sekwencje (ciągi), będące elementami wektora można tworzyć za pomocą operatora `:` bądź funkcji `seq`

```
> 30:10  
> seq(1,45,by=3)  
> c(seq(from=1, by=3, length=15),1:10)
```



Środowisko R umożliwia replikację, a więc **powielanie danych**. Służy do tego funkcja **rep()**. Jej pierwszym argumentem jest obiekt (najczęściej wektor), który ma zostać powielony. Kolejne (opcjonalne) argumenty to następujące nieujemne liczby całkowite:

- **times** - ilość powtórzeń obiektu,
- **length.out** - długość wektora wynikowego,
- **each** - ilość powtórzeń każdego elementu wektora źródłowego.

## Przykład:

```
> rep(5:8,4)
> rep(5:8,each=4) # inny wynik niż poprzednio!
> rep(5:8,length.out=10)
```

# Zmienne

W środowisku R mamy możliwość definiowania zmiennych i odwoływania się do nich poprzez nazwę. Operatorem przypisującym zmiennej określoną wartość jest `<-` oraz `=`, analogicznie działa także funkcja **assign()**:

```
> x <- c(4:10)
> y = c("jeden", "dwa")
> assign("z", c(x,y))
```

Po przypisaniu zmiennej określonej wartości, R nie wyświetla jej na ekranie. Jeżeli chcemy zobaczyć wynik przypisania, należy wywołać zmienną poprzez jej nazwę:

```
> x
> y[2]
> z[5]
```

# Macierze

Tworzenie macierzy umożliwia funkcja `matrix()`. Jej parametrami są:

- wektor, który inicjuje zawartość macierzy,
- liczba wierszy,
- liczba kolumn.

Przykład:

```
> macierz<-matrix(seq(1:4),3,4) ~~~ macierz=
```

$$\begin{bmatrix} 1 & 4 & 3 & 2 \\ 2 & 1 & 4 & 3 \\ 3 & 2 & 1 & 4 \end{bmatrix}$$

Odwoływać się do poszczególnych elementów macierzy można następująco:

```
> # element macierzy z 1. wiersza i 3. kolumny:  
> macierz[1,3]  
> macierz[1,] # cały pierwszy wiersz macierzy  
> macierz[,3] # cała trzecia kolumna macierzy
```

# Tablice

Dwuwymiarowe macierze można definiować również przy użyciu funkcji `array()`. Ma ona jednak więcej możliwości, bo za jej pomocą można tworzyć tablice wielowymiarowe.

Parametry funkcji `array()` są dwa:

- wektor, który inicjuje zawartość tablicy,
- wektor, który określa wymiar tablicy.

```
> tablica1<-array(seq(1:4), c(3,4))
```

```
> tablica2<-array(c(1:5),c(2,2,2))
```

Odwołanie do poszczególnych elementów tablicy jest analogiczne jak w przypadku macierzy:

```
> tablica1[2,2]
```

```
> tablica1[1:2,] # wiersze w tablicy od 1 do 2
```

```
> tablica1[,1]
```

```
> tablica2[2,2,2]
```

# Listy

Lista to typ danych, który **pozwalą łączyć obiekty różnych typów**. Konstrukтором listy jest funkcja `list()`.

```
> przedmiot<-c("piłka", "skrzynia")
> kolor<-c("czerwony", "zielony")
> waga<-c(2,8)
> lista<-list(przedmiot, kolor, waga)
> names(lista)<-c("przedmiot", "kolor", "waga")
```

Do elementów listy można odwoływać się poprzez ich nazwę lub pozycję na liście:

```
> lista[2]
> lista$waga
> lista$waga[which(lista$przedmiot=="piłka")]
> lista$kolor[lista$przedmiot=="skrzynia"]
```

Bardzo przydatny jest tu operator porównania `==`, który pozwala uzyskać elementy listy spełniające określony warunek.

# Ramki danych

Ramka danych jest listą, która **pozwala łączyć kolumnowo wektory równej długości**. Jest ona szczególnie przydatna do prezentowania danych empirycznych, ponieważ jej struktura odpowiada tabeli w bazie danych.

Konstruktorem ramki danych jest funkcja `data.frame()`.

```
> ramka<-data.frame(przedmiot, kolor, waga)
```

Do elementów ramki danych można odwoływać się analogicznie jak w przypadku listy:

```
> ramka[3]
> ramka$przedmiot
> ramka$waga[ramka$kolor=="czerwony"]
```

# Sortowanie danych

Wektor można posortować w kolejności rosnącej bądź malejącej. Służy do tego funkcja `sort()`:

```
> sort(kolory) # domyślnie sortujemy w kolejności  
+ # rosnącej  
> sort(kolory, decreasing = TRUE) # zmieniamy  
+ # kolejność na malejącą
```

# Import danych z pliku

Import danych z plików zewnętrznych możliwy jest w R dzięki funkcji **read.table()**, która czyta plik w formacie tabeli i tworzy na jego podstawie ramkę danych. Każda obserwacja odpowiada linii w pliku, a zmienna - polu, czyli kolumnie.

Ważniejsze parametry:

- **file** - pierwszy parametr, określa nazwę pliku. Jeżeli nie jest podana pełna ścieżka, plik jest pobierany z katalogu roboczego R,
- **header**: TRUE/ FALSE - określa, czy nagłówek (nazwy kolumn) znajdują się w pierwszym wierszu, domyślnie ustawiony na FALSE,
- **sep** - definiuje separator pól, czyli znak pomiędzy kolejnymi wartościami w obrębie obserwacji, np. ";", ",", " " (domyślnie brak: "" ),
- **dec** - definiuje znak używany do oddzielania części dziesiętnych, domyślnie ".".

Proszę dla przykładu pobrać do katalogu roboczego plik `warzywniak.txt` a następnie wczytać jego zawartość:

```
> read.table("warzywniak.txt", header=TRUE)
```

Analogiczna funkcja: **read.csv** służy do czytania plików z rozszerzeniem `.csv`.



# Eksport danych do pliku

Dane analizowane podczas pracy można zapisać w pliku za pomocą funkcji **write.table()**. Dane niebędące ramką danych ani macierzą są przed zapisem konwertowane do formatu ramki.

Ważniejsze parametry:

- `x` - pierwszy parametr, określa obiekt, który chcemy zapisać,
- `file` - nazwa pliku, w którym mają być zapisane dane. Jeżeli nie jest podana pełna ścieżka, plik jest zapisywany do katalogu roboczego, "" oznacza wydruk w konsoli R
- `append`: TRUE/FALSE - umożliwia dopisanie danych do istniejącego już pliku, jeżeli ustawiony na TRUE. W przeciwnym przypadku plik o tej samej nazwie zostanie zniszczony. Domyślnie ustawiony na FALSE,
- `sep` - definiuje separator pól, czyli znak pomiędzy kolejnymi wartościami w obrębie obserwacji, np. ";", ",", (domyślnie spacja: " "),
- `dec` - definiuje znak używany do oddzielania części dziesiętnych, domyślnie ".",

Przykład:

```
> dane <- data.frame(v1=c(1,2,3), v2=c("a","b","c"))  
> write.table(dane, file = "dane.txt", sep=",")
```

# Własne funkcje w R

Środowisko R umożliwia **definiowania własnych funkcji**. Odbywa się to poprzez przypisanie nazwy funkcji do jej definicji:

```
nazwa <- function(arg1, arg2, ...) wyrażenie  
nazwa <- function(arg1, arg2, ...) {blok wyrażen}
```

Jeżeli w bloku nie znajdzie się wywołanie funkcji `return()`, wówczas zwróci ona ostatnio obliczoną wartość. Funkcję wywołujemy poprzez nazwę, podając w nawiasie jej parametry (argumenty).

**Przykład:** Zdefiniujemy funkcję zwracającą sumę dwóch podanych liczb:

```
> suma<-function(a,b) a+b  
> suma(12,5)  
[1] 17  
> kwadrat_sumy<-function(a,b){  
+ c<-a+b  
+ return(c^2)  
+ }  
> kwadrat_sumy(12,5)  
[1] 289
```

# Pętle: for

Środowisko R umożliwia **konstrukcję pętli**, czyli cyklicznego wykonywania ciągu instrukcji określoną liczbę razy, dla każdego elementu zbioru. Odbywa się to w bloku:

```
for(zmienna in zbiór) instrukcja  
for (zmienna in zbiór) {ciąg instrukcji}
```

**Przykład:** Podsumujemy wiek drzew znajdujących się w bazie Orange.

```
> attach(Orange)  
> suma_wieku=0  
> n=length(age)  
> for(i in 1:n) {  
+ suma_wieku=suma_wieku+age[i]  
+ }  
> suma_wieku  
> sum(age) #to samo
```

# Instrukcje warunkowe

Środowisko R umożliwia **konstrukcję instrukcji warunkowych**, które pozwolą na wykonanie różnych obliczeń w zależności od tego, czy zdefiniowane wyrażenie logiczne jest prawdziwe, czy też nie. Odbywa się to w bloku:

**if** (warunek) instrukcja

**if** (warunek) instrukcja **else** instrukcja

**Przykład:** Przeanalizujemy obwody drzew z bazy Orange, wyświetlając odpowiednie komentarze:

```
> n=dim(Orange)[1]
> srednia=mean(circumference)
> for (i in 1:n) {
+   if (circumference[i]>srednia)
+     print("Obwód większy niż średnia")
+   else if (circumference[i]==srednia)
+     print ("Obwód równy średniej")
+   else print("Obwód mniejszy niż średnia")
+ }
```

**UWAGA:** Przyrównanie dwóch liczb wymaga użycia dwóch znaków równości **==**.

# Instrukcje warunkowe na wektorach

Zastosowanie pętli w poprzednim przykładzie nie jest najbardziej efektywnym rozwiązaniem. Istnieje jednak w R instrukcja warunkowa umożliwiająca pracę na całych wektorach:

```
ifelse(warunek, instrukcja_gdy_TRUE, instrukcja_gdy_FALSE)
```

## Przykład:

```
> ifelse(circumference>srednia, "Obwód większy niż średnia",  
+ ifelse(circumference==srednia,"Obwód równy średniej", "Obwód  
mniejszy niż średnia"))  
> detach(Orange)
```

# Pętle: while, repeat

W R mamy również możliwość konstrukcji pętli innego rodzaju: `while` i `repeat`.

Pętla `while` wykonuje polecenia dopóty, dopóki warunek jest spełniony. Pętla `repeat` wykonuje instrukcje do momentu, gdy ją wprost zatrzymamy.

**Przykład:** Pętla `while` wypisująca kwadraty liczb od 1 do 10:

```
> i<-1  
> while (i<=10){  
+   print(i^2)  
+   i<-i+1  
+ }
```

**Przykład:** Pętla `repeat` wypisująca kwadraty liczb od 1 do 10:

```
> i<-1  
> repeat {  
+   print(i^2)  
+   i<-i+1  
+   if (i>10) break  
+ }
```

# Historia poleceń w R

Po zakończeniu pracy w R, możemy zapisać w pliku `.Rhistory` historię wykonywanych poleceń. Jest to szczególnie przydatne, gdy przez dłuższy czas pracujemy nad jednym projektem i chcemy wielokrotnie wywoływać to samo polecenie z różnymi argumentami, również po zamknięciu sesji.

Jest to możliwe poprzez wybór w menu

`File -> Save History....`

Aby wczytać historię poleceń z poprzednio zapisanej sesji, należy wybrać odpowiedni plik poprzez menu

`File -> Load History....`

Aktualny wykaz wywoływanych poleceń dostępny jest poprzez wywołanie funkcji

```
> history()
```

# Obszar roboczy w R

**Obszar roboczy** zawiera informacje o wszystkich utworzonych w danej sesji obiektach. Można go zapisać w pliku `.RData`, wybierając z menu **File -> Save Workspace...** Jest to przydatne wówczas, gdy w kolejnej sesji będziemy chcieli pracować na tych samych danych.

Aby wczytać obszar roboczy z poprzednio zapisanej sesji, należy wybrać plik poprzez menu **File -> Load Workspace...** Odczytując obszar roboczy wracamy do stanu zapisanego we wczytanym pliku.

Lista obiektów sesji jest dostępna po wprowadzeniu jednego z poleceń

```
> objects()  
> ls()
```

Pliki `.Rhistory` oraz `.RData` zawierające historię poleceń oraz obszar roboczy zapisywane i wczytywane są z aktualnie ustawionego katalogu roboczego.







# Podsumowanie możliwości R

R często jest nazywany pakietem statystycznym, z uwagi na to, że przede wszystkim wykorzystywany jest do obliczeń statystycznych. Jednak jego możliwości są o wiele większe. Użytkownicy R na całym świecie rozwijają go tworząc nowe pakiety, dzielą się swoimi osiągnięciami np. przy generowaniu trójwymiarowych animacji, tworzeniu gier komputerowych, automatyzacji zadań np. generowania raportów itp. Aby się o tym przekonać, proszę wywołać następujące polecenia:

```
> demo(graphics)
> demo(persp)
> demo(Japanese)

> install.packages("rgl") # instalujemy pakiet 'rgl'
> library(rgl) # ładujemy bibliotekę pakietu 'rgl'
> demo(rgl) # demonstracja możliwości pakietu 'rgl'
```

Darmowy dostęp oraz w zasadzie nieograniczone możliwości R sprawiają, że ma on przewagę nad komercyjnymi pakietami statystycznymi takimi jak np. SAS, SPSS i rekompensują jego ubogi interfejs graficzny użytkownika.

-  <https://www.r-project.org/doc/bib/R-books.html>, Books related to R
-  Przemysław Biecek, **Przewodnik po pakiecie R**, Oficyna Wydawnicza GiS, Wrocław, 2011
-  Joseph Adler, **R in a Nutshell**, O'Reilly Media, 2009
-  Phil Spector, **Data Manipulation with R**, Use R!, Springer, 2008