

0.1 O projekcie R

R to nazwa języka programowania oraz nazwa platformy programistycznej wyposażonej w interpretator języka R. W platformie tej zaimplementowano wiele metod statystycznych, przez co często nazywana jest platformą do wykonywania analiz statystycznych. Jej możliwości są jednak znacznie większe, wystarczy wymienić automatyczne generowanie raportów, wysyłanie maili czy renderowanie trójwymiarowych animacji.

Język R jest językiem interpretowanym a nie kompilowanym. Kolejne komendy mogą być podawane linia po linii lub też wykonywane jako skrypt (plik tekstowy z listą komend do wykonania). Skrypty można udostępniać w postaci kodów źródłowych, można je wykonywać niezależnie od platformy sprzętowej. Wiele osób uważa (często słusznie), że języki interpretowane są wolne i wymagają dużo pamięci, jednak obecne komputery są szybkie i mają dużo pamięci więc w standardowych zastosowaniach nie należy się tym przejmować.

Osoby wolące okienkowe interfejsy użytkownika, w których można „wyklikać” wyniki mogą skorzystać z kilku darmowych GUI do platformy R (np. z pakietu Rcmdr). Zdecydowanie jednak polecam przełamanie tej niechęci, przygotowywanie skryptów jest proste i umożliwia prostą automatyzację pracy.

R jest projektem GNU opartym o licencje GPL GNU. W uproszczeniu oznacza to, iż jest w zupełności darmowy do wszystkich zastosowań.

R jest podobny do języka S opracowanego w laboratoriach Bell’a, był bowiem po części na tym języku wzorowany. Znacząca część programów w języku S będzie działała też na platformie R. Osoby znające inne popularne platformy np. Matlab, Octave, S+, SPSS, SAS’a itp. nie będą miały większych problemów by zacząć używać języka R. Istnieje wiele dokumentów prezentujących różnice pomiędzy danym językiem a R’em ich lista znajduje się np. w sekcji `getting-started:translations` w Rwiki.

Platforma R wyposażona jest w świetną dokumentację, dostępną w postaci dokumentów PDF lub stron html. Aktualnie dokumentacja ta jest angielskojęzyczna, jednak trwają prace nad różnymi lokalizacjami.

Cztery główne (ale nie jedyne) zalety platformy R, dzięki którym deklasuje ona konkurencję to:

- pozwala na tworzenie i upowszechnianie pakietów implementujących nowe funkcjonalności. Obecnie dostępnych jest blisko 1000 pakietów do różnorodnych zastosowań,
- pozwala na wykonywanie funkcji z bibliotek przygotowywanych w innych językach (C, C++, Fortran) oraz wykonywanie funkcji w R z poziomu innych języków (Java, C++, C i wiele innych),
- jest w zupełności darmowa do wszelkich zastosowań (również większość pakietów jest darmowych i dostępnych w ramach licencji GNU GPL lub GNU

GPL 2.0),

- można wygenerować wykresy o wysokiej jakości co jest bardzo istotne przy prezentacji wyników.

Jedną z niewielu rzeczy których nie można zrobić na platformie R jest cappuccino.

W internecie i księgarniach dostępnych jest wiele angielskojęzycznych źródeł poświęconych R'owi i analizie danych w systemie R, listę najpopularniejszych można znaleźć na stronach:

- popularne manuale do R'a <http://cran.r-project.org/manuals.html>,
- R wiki <http://wiki.r-project.org/rwiki/>,
- książki o analizie danych w R <http://www.r-project.org/doc/bib/R-books.html>.

0.2 Instalacja

0.2.1 Instalacja środowiska

R jest dostępny w postaci źródłowej oraz skompilowanej dla większości systemów operacyjnych, w tym wszystkich dystrybucji Linuxa, Unixa dla wersji Windowsa począwszy od Windowsa 95 a nawet na MacOsa. Instalacja jest prosta, wystarczy wybrać jeden z mirrorów z plikiem instalacyjnym, ściągnąć, uruchomić ten plik a następnie postępować zgodnie z instrukcjami. Adresy mirrorów znaleźć można pod adresem <http://cran.r-project.org/mirrors.html>).

Szczegółową instrukcję instalacji można znaleźć pod adresem <http://cran.r-project.org/doc/manuals/R-admin.pdf>. W dalszej części będzie opisywana Windowsowa wersja R'a (osoby korzystające z Linuxa z pewnością świetnie radzą sobie z korzystaniem z manuali). Windowsową wersję 2.4.1 można ściągnąć z wrocławskiego serwera <http://r.meteo.uni.wroc.pl/bin/windows/base/> (należy uruchomić plik `R-2.4.1-win32.exe`).

Osoby używające platformy R do bardzo wymagających obliczeniowo analiz powinny raczej używać Linuxowej lub Unixowej wersji platformy R z uwagi na wydajniejsze zarządzanie pamięcią.

0.2.2 Instalacja pakietów

Funkcjonalności dostępne w platformie R umieszczone są w pakietach. Platforma R instalowana jest wraz z podstawowym ich zbiorem pozwalającym na wykonywanie prostych analiz. Kolejne pakiety można zainstalować po uruchomieniu systemu R poleceniem

```
install.packages(nazwa.pakietu, dependencies = TRUE)
```

lub wybierając z menu opcje `packages\install package(s)...`

0.2.3 Korzystanie z pakietów

Zainstalowanie pakietu oznacza, że implementacja odpowiednich funkcji znajduje się na dysku twardym komputera. Aby móc skorzystać z tej funkcjonalności należy ją załadować (włączyć). Po każdym uruchomieniu platformy R, ładowane są pakiety podstawowe takie jak *base*, *graphics*, *stats*. Aby skorzystać z dodatkowych funkcji lub zbiorów danych należy załadować pakiet w którym się one znajdują (zakładamy, że pakiety te zostały już zainstalowane). Pakiety ładuje się poleceniem

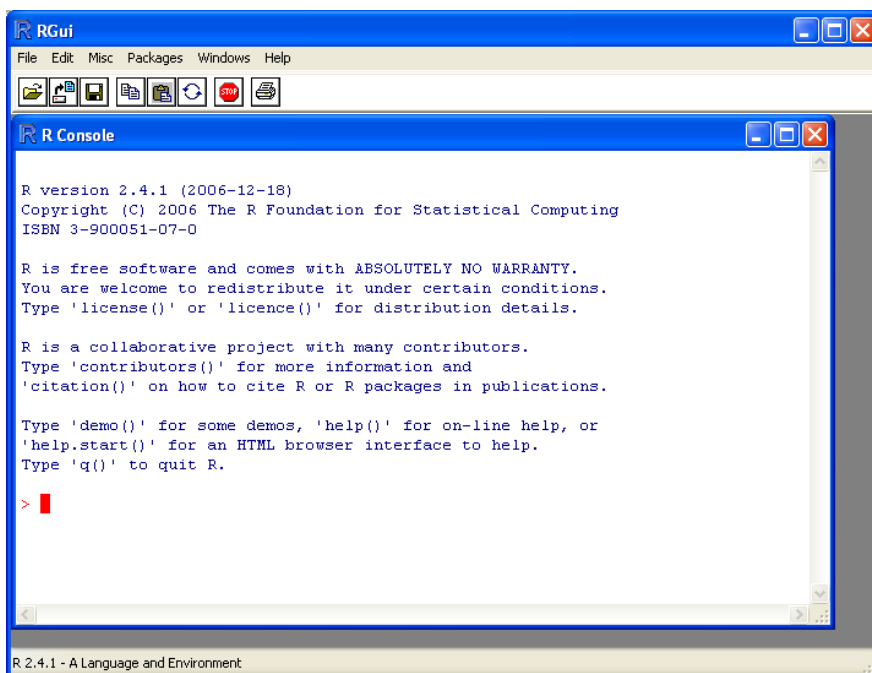
```
library(nawa.pakietu)
```

Na dzień dzisiejszych dostępnych jest blisko 1000 pakietów. W tym zbiorze trudno czasem odnaleźć pakiet z interesującą nas funkcjonalnością. Poniżej zamieszczono listę najpopularniejszych (będzie aktualizowana w trakcie semestru)

- rgl - pakiet z funkcjami do grafiki 3D,
- Rcmdr - pakiet z GUI pozwalającym na przeprowadzenie większości podstawowych statystyk bez znajomości nazw funkcji,
- stats - pakiet z większością podstawowych funkcji statystycznych,
- graphics - pakiet z funkcjami do rysowania wykresów.

0.3 Pierwsze uruchomienie

Po zainstalowaniu w wybranym miejscu na dysku utworzony zostanie katalog R-2.4.1 (lub inny w zależności od instalowanej wersji), a w nim podkatalog bin z plikami wykonywalnymi. Platformę R można uruchomić w trybie wsadowym wykonując plik R.exe lub też w trybie z prostym GUI okienkowym wykonując plik Rgui.exe. Wybór trybu uruchomienia proponujemy oprzeć na prostej zasadzie, jeżeli nie wiesz czym te tryby się różnią to uruchom wersje z GUI.



Rysunek 1: Okno powitalne po uruchomieniu pliku Rgui.exe.

Po uruchomieniu tej wersji pokaże się okienko takie jak na Rys 1. Znak > to znak zachęty. Oznacza on, że platforma R jest gotowa do realizacji kolejnego polecenia. Jeżeli linia rozpoczęta jest znakiem + to znaczy, że polecenie wpisane w poprzedniej linii nie zostało jeszcze zakończone i platforma czeka na jego część dalszą zanim zostanie ono wykonane (np. rozpoczęta jest pętla, otwarty jest nawias lub wpisywany jest łańcuch znaków).

Pierwsze polecenie, które warto przeciwiczyć to

```
q()
```

czyli zamknięcie platformy R. Po wykonaniu tego polecenia zostaniemy zapytani, czy zachować aktualny stan środowiska a następnie platforma zostanie zamknięta.

Jak to zostało zaznaczone w poprzedniej sekcji platforma R wyposażona jest w bogaty system pomocy. Aby z niej skorzystać można użyć następujących funkcji

- `help()` - wyświetla stronę powitalną systemu pomocy,

- `help(nazwa.funkcji)` lub `?nazwa.funkcji` - wyświetla informacje o danej funkcji. Kolejne sekcje zawierają zwięzły opis, listę argumentów, objaśnienie poszczególnych argumentów, szczegółowy opis funkcji, literaturę, odnośniki do innych funkcji oraz przykłady użycia,
- `args(nazwa.funkcji)` - wyświetla listę argumentów dla danej funkcji,
- `example(nazwa.funkcji)` - uruchamia skrypt z przykładowymi wywołaniami poszczególnych funkcji, na dobry początek proszę sprawdzić `example(plot)`,
- `help.search(slowo.kluczowe)` - przegląda opisy funkcji znajdujących się w zainstalowanych pakietach i wyświetla te w których znaleziono zadane *slowo.kluczowe*. W liście wyników znajduje się również informacja w którym pakiecie znajdują się znalezione funkcje.

0.4 Podstawy składni języka R

0.4.1 Obiekty

Wszystko czym można operować w języku R jest obiektem. Obiekty można podzielić (nie wdając się w formalne szczegóły) na kilka typów.

- **Liczba.** Ten typ nie wymaga komentarza. Dozwolona jest notacja naukowa (np. `2.5e3`), kropką dziesiętną jest kropka.
- **Łańcuch znaków** są rozpoczynane znakiem `'` lub `"` oraz kończone tym samym znakiem.
- **Typ logiczny.** Obiekty tego typu przechowują jedną z dwóch wartości. Logiczna prawda to `T` lub `TRUE`, logiczny fałsz to `F` lub `FALSE`.
- **Wektor** to uporządkowany zbiór obiektów tego samego typu. Konstrukctorem wektora jest funkcja `c()`. Przykładowa konstrukcja wektora trzech liczb

```
c(1,3,4)
```

- **Lista** to uporządkowany zbiór elementów. Elementy listy mogą mieć różne typy. Elementy listy mają nazwy. Konstrukctorem listy jest funkcja `list()`. Przykładowa konstrukcja listy czterech obiektów to

```
list(name=c("Jan","Tomasz"), surname="Kowalski", age=25, married=T)
```

- **Macierz.** Konstrukctorem macierzy dwuwymiarowej jest funkcja `matrix()`. Przykładowa macierz o wymiarach 4×2 wypełniona zerami może być skonstruowana poleceniem

```
matrix(0,4,2)
```

Można też konstruować macierze o większej liczbie wymiarów, o tym w kolejnych sekcjach.

- **Ramka danych** to lista wektorów o tej samej długości. Równoważnie może być wyświetlana jako macierz, w której elementy w kolumnie są tego samego typu. Konstrukctorem jest funkcja `data.frame()`. Przykładowa ramka to

```
data.frame(id=c(100,101,102), age=c(25,21,22), male=c(T,T,F))
```

- **Typ wyliczeniowy.** Przydatny do przechowywania wartości występujących na kilku poziomach (np. płeć występuje na dwóch poziomach, tzn. może przyjmować dwie wartości). Konstrukctorem jest funkcja `factor()`. Przykładowy wektor trzech elementów typu wyliczeniowego z dwoma poziomami to

```
factor(c("sierżant", "kapitan", "sierżant"))
```

- **Funkcja.** Jej konstruktorem jest funkcja `function`. Więcej o funkcjach w kolejnych sekcjach.

0.4.2 Konwersja

Najczęstsze konwersje to zamiana na typ znakowy (funkcja `as.character`) lub na typ liczbowy (funkcja `as.numeric`). Konwertować można złożone struktury takie jak lista lub macierz, w tym przypadku konwertowany jest każdy element listy lub macierzy.

Konwertować można też strukturę obiektu, np. funkcjami `as.matrix`, `as.list`, `as.logical`.

Uwaga!!! Konwertując etykiety typu wyliczeniowego na typ liczbowy należy być ostrożnym, zaskakującym może być wynik takiej konwersji, np.

```
> as.numeric(factor(c(-2,2)))  
[1] 1 2
```

Bardziej oczekiwany wynik uzyskamy konwertując „po drodze” dany obiekt na typ znakowy

```
> as.numeric(as.character(factor(c(-2,2))))  
[1] -2 2
```

0.4.3 Zmienne

Nazwa zmiennej powinna rozpoczynać się literą, może składać się z liter, cyfr lub symbolu kropki. Istotna jest wielkość liter, a więc zmienne `x` i `X` to dwie różne zmienne.

Do zmiennej przypisać można wartość jednym z trzech operatorów przypisania

- `->` przypisuje wartość znajdującą się z lewej strony do zmiennej po prawej stronie
- `<-` przypisuje wartość znajdującą się z prawej strony do zmiennej po lewej stronie
- `=` jak wyżej, raczej nie jest polecany.

Sprawdź co się stanie w wyniku wykonania poniższych instrukcji

```
c(13, 13) -> zmienna.z.kropka  
imie <- "Ola"  
i = 4
```

0.4.4 Indeksy

Do elementów wektorów, list, macierzy i ramek danych możemy się odwoływać w następujący sposób

- **zmienna[zakres]**, *zmienna* jest listą (wektorem), *zakres* jest wektorem liczb całkowitych lub jedną liczbą całkowitą. W tym przypadku zwracana jest lista (wektor) zawierająca wybrane elementy. Jeżeli *zakres* to wektor liczb ujemnych to zwrócone będą wszystkie elementy z listy (wektora) poza pozycjami określonymi przez wartości absolutne z *zakres*

```
> wektor ← 1:10
> wektor
[1] 1 2 3 4 5 6 7 8 9 10
> wektor[1:3]
[1] 1 2 3
> wektor[c(-1,-3,-5)]
[1] 2 4 6 7 8 9 10
```

- **zmienna[zakres1,zakres2]**. *zmienna* jest macierzą (ramką danych). Wybierana jest macierz (ramka danych) o wskazanych indeksach. Jeżeli któryś z zakresów nie będzie podany, zostaną wybrane wszystkie pozycje w danym wierszu/kolumnie

```
> macierz ← matrix(1:4,2,2)
> macierz
  [,1] [,2]
[1,] 1 3
[2,] 2 4
> macierz[1,]
[1] 1 3
> macierz[-1,-1]
[1] 4
```

- **zmienna\$watosc1**. *zmienna* to lista lub ramka danych, zwracany jest element listy (lub kolumna z ramki danych) o nazwie *watosc1*

```
> osobnik ← list(name=c("Jan","Tomasz"), surname="Kowalski", age=25, married=T)
> osobnik$name
[1] "Jan" "Tomasz"
> osobnik[[2]]
[1] "Kowalski"
```

- **zmienna[[indeks1]]**. *zmienna* to wektor, lista, macierz lub ramka danych. Wybierany jest jeden element o indeksie *indeks1*.

```
> macierz ← matrix(1:4,2,2)
> macierz[[2]]
[1] 2
> osobnik ← list(name=c("Jan","Tomasz"), surname="Kowalski", age=25, married=T)
> osobnik[[1]]
[1] "Jan" "Tomasz"
```

Przydatną funkcją jest `which()`, która zwraca indeksy obiektu spełniające zadany warunek logiczny.

0.4.5 Sekwencje

Sekwencje liczb całkowitych można generować używając operatora `:` lub funkcji `seq()`.

```
> -2:2
[1] -2 -1 0 1 2
> seq(10,50,by=10)
[1] 10 20 30 40 50
```

Uwaga!!! Należy być ostrożnym korzystając z sekwencji liczb rzeczywistych. Problemy mogą pojawić się nawet w tak prostych sytuacjach

```
> v ← seq(0.7, 0.8, by=0.1)
> v
[1] 0.7 0.8
> v == 0.8
[1] FALSE FALSE
```

0.4.6 Operatory

Poniżej przedstawiono najpopularniejsze operatory języka R. Jeżeli zachodzi taka potrzeba, to można również definiować własne operatory (o czym jeszcze będzie mowa).

- `+`, `-`, `*`, `/`, `^` standardowe operatory arytmetyczne. Oba argumenty powinny mieć taki sam wymiar lub jeden z argumentów powinien być liczbą,
- `%x%` iloczyn kronekera dwóch macierzy,
- `%%` reszta modulo z dzielenia,
- `%/%` dzielenie całkowite,
- `%*%` iloczyn dwóch macierzy,
- `<`, `==`, `>`, `<=`, `>=`, `!=` standardowe operatory logiczne,
- `!` negacja,
- `&`, `&&`, `|`, `||` logiczny iloczyn oraz logiczna suma. Jeżeli chcemy wyznaczyć logiczną sumę(iloczyn) wszystkich elementów wektora można wykorzystać funkcję `any()` lub `all()`. Operatory `&` i `|` służą do wykonywania operacji na listach lub wektorach, podczas gdy `&&` i `||` na pojedynczych wartościach. Warto przeanalizować poniższy przykład.

```
> lubie.statystyke = c(ala=F, ola=T, ewa=T)
> lubie.prowadzacego = c(ala=T, ola=T, ewa=F)
> lubie.statystyke & lubie.prowadzacego
  ala   ola   ewa
FALSE TRUE FALSE
> lubie.statystyke && lubie.prowadzacego
```

```
[1] FALSE
> lubie.statystyke | lubie.prowadzacego
  ala ola ewa
TRUE TRUE TRUE
> lubie.statystyke || lubie.prowadzacego
[1] TRUE
```

0.4.7 Komentarze

Język R, jak każdy przyzwoity (i wiele nieprzyzwoitych) języków programowania umożliwia komentowanie fragmentów kodu. Znakiem komentarza jest #, interpreter ignoruje ten znak i wszystkie po nim występujące do końca linii.

0.5 Odczytywanie i zapisywanie danych

Zanim będzie można przystąpić do analiz należy wczytać dane. Również po wykonaniu analiz dobrze jest móc zapisać uzyskane wyniki. Poniższe sekcje opisują jak wczytywać dane z plików tekstowych lub importować z popularnych formatów.

0.5.1 Odczytywanie i zapisywanie plików tekstowych

Funkcje odczytujące i zapisujące do plików mają wiele parametrów pozwalających na określenie kodowania, znaku separatora, kropki dziesiętnej, typu odczytywanych danych itp. Aby nie utonąć w szczegółach przedstawione są najpopularniejsze przykłady użycia. Zainteresowani oraz potrzebujący powinni potrafić już uruchomić pomoc dla danej funkcji i sprawdzić jakie są inne parametry.

Odczytać wektor liczb rozdzielonych białymi znakami można poleceniem

```
wektor.liczb = scan("nazwa.pliku")
```

Jeżeli chcemy odczytać wektor łańcuchów liczb, rozdzielonych przecinkami użyjemy polecenia

```
wektor.lancuchow = scan("nazwa.pliku", what="character", sep=",")
```

Jeżeli w pliku znajduje się macierz liczb to wygodnie jest ją odczytać następującym poleceniem

```
macierz = read.table("nazwa.pliku")
```

Jeżeli w pliku umieszczone są też informacje o nazwach kolumn i wierszy, a elementy macierzy rozdzielane są znakami tabulacji, to właściwe będzie użycie następującego polecenia

```
macierz = read.table("nazwa.pliku", header=T, sep="\t")
```

Wektor możemy zapisać do pliku korzystając z funkcji `cat()`, np. następująco

```
cat(wektor, file="nazwa.pliku", append=F)
```

Jeżeli zapisać chcemy macierz lub ramkę danych oddzielając kolejne elementy tabulacją to lepiej jest skorzystać z funkcji `write.table()`

```
write.table(macierz, file="nazwa.pliku", sep="\t")
```

Uwaga!!! Nazwa pliku może zawierać ścieżkę dostępu, może być też adresem URL.

0.5.2 Import i eksport danych z Excel'a

Najwygodniej importować dane z Excela zapisując je w formacie csv a następnie odczytywać lub zapisywać funkcjami

```
macierz = read.csv(file="nazwa.pliku.csv", dec=",")
write.csv(macierz, file="nazwa.pliku.csv", dec=",")
```

Należy pamiętać że Excel z polską lokalizacją stosuje , jako kropkę dziesiętną.

0.5.3 Import i eksport danych ze schowka

Dostęp do schowka uzyskuje się podając "clipboard" jako nawę pliku. Tak więc instrukcja

```
wektor = scan("clipboard")
```

wpisze do zmiennej wektor zawartość schowka, a polecenie

```
write.csv(macierz,"clipboard")
```

zachowa w schowku wartości zmiennej macierz w formacie pliku csv.

0.5.4 Import danych z SPSS'a

Aby zaimportować dane z SPSS'a potrzebny jest pakiet `Hmisc`. Do importu danych służy funkcja `spss.get()`. Przykładowa sesja

```
library(Hmisc)
ramka.danych = spss.get("nazwa.pliiku.sav")
```

0.5.5 Import danych z pakietów

Niektóre pakiety poza funkcjami udostępniają również zbiory danych. Dostęp do tych zbiorów uzyskuje się stosując funkcję `data()`. Przykładowe odczytanie danych z pakietu `BSDA`

```
library(BSDA)
data(Coffee)
head(Coffee)
```

0.6 Zapisywanie wykresów

Platforma R umożliwia zapisywanie wykresów do różnych formatów, w tym PNG, BMP, JPG, PDF i PS. Jeżeli chcemy zapisać do pliku już wyświetlony na ekranie wykres, należy uaktywnić okno z wykresem (nazwane `R graphics`), a następnie z menu wybrać `File\Save as`.

Wykresy można też zapisywać do pliku automatycznie. Funkcje `png()`, `bmp()`, `jpg()`, `pdf()` przekierowują domyślne wyjście dla grafiki do pliku. Funkcja `dev.off()` powoduje zamknięcie przekierowania i zapisanie wykresu pod wskazaną nazwą. Przykład poniżej

```
png("obrazek.png", width = 640, height = 480)
example(plot)
dev.off()
```

0.7 Przykładowa sesja - przećwiczyć na zajęciach

```
# zrobmy kilka zmiennych
wektor ← 1:100
macierz ← matrix(1:100,10,10)
fragment← sample(wektor,50,T)
ramka ← read.csv("http://semestr.pl/cogito/download/drzewka.csv",
                 header=T, sep="\t")

# wyświetlmy te zmienne
fragment
ramka
ramka$waga
str(ramka)
head(ramka,3)
ramka[ramka$plec=="M",]
colnames(ramka)

# policzmy cos
BMI ← ramka$waga/(ramka$wzrost/100)^2
ramka$BMI = BMI

# dowiedzmy sie czegos o funkcji hist
?hist
args(hist)

# narysujmy cos
hist(fragment)
hist(ramka$wiek, xlab="Wiek")

plot(ramka$waga, ramka$wzrost/100, xlim=c(50,100), ylim=c(1.5,2),
     xlab="waga [kg]", ylab="wzrost [m]", type="p", lwd=2, col="red")

ls()
attach(ramka)
ls()

etykietki = paste(ramka$imie, ramka$nazwisko)
identify(waga, wzrost/100, etykietki)

# a teraz uruchamiamy R commandera
library(Rcmdr)

# demo mozliwosci pakietu graphics
demo(graphics)

library(rgl)
demo(rgl)

# kilka podsumowan
summary(fragment)
summary(ramka)
```

```

# co to za pokoj?
add.box ← function(x1,x2,y1,y2,z1,z2,col) {
  rgl.quads(c(x1,x1,x2,x2),c(y1,y2,y2,y1),c(z1,z1,z1,z1),col)
  rgl.quads(c(x1,x1,x2,x2),c(y1,y2,y2,y1),c(z2,z2,z2,z2),col)
  rgl.quads(c(x1,x1,x1,x1),c(y1,y2,y2,y1),c(z1,z1,z2,z2),col)
  rgl.quads(c(x2,x2,x2,x2),c(y1,y2,y2,y1),c(z1,z1,z2,z2),col)
  rgl.quads(c(x1,x2,x2,x1),c(y1,y1,y1,y1),c(z1,z1,z2,z2),col)
  rgl.quads(c(x1,x2,x2,x1),c(y2,y2,y2,y2),c(z1,z1,z2,z2),col)
}

rgl.open()
rgl.quads(c(-0.01,-0.01,4.5,4.5),c(0,2.5,2.5,0),c(2.21,2.21,2.21,2.21),
  "blue",alpha=0.5)
rgl.quads(c(-0.01,-0.01,4.5,4.5),c(0,2.5,2.5,0),c(-0.01,-0.01,-0.01,-0.01),
  "blue",alpha=0.5)
rgl.quads(c(4.5,4.5,4.5,4.5),c(0,2.5,2.5,0),c(-0.01,-0.01,2.21,2.21),"blue",
  alpha=0.5)
rgl.quads(c(-0.01,-0.01,-0.01,-0.01),c(0,2.5,2.5,0),c(-0.01,-0.01,2.21,2.21),
  "blue",alpha=0.5)
rgl.quads(c(4.49,4.49,4.49,4.49),c(1,2.3,2.3,1),c(0.5,0.5,2.1,2.1),"black")
rgl.quads(c(0,0,4.5,4.5),c(0,0,0,0),c(0,2.2,2.2,0),"#AB5400")
add.box(0,2.2,0,2.1,2.2,1.8,"#E7F179")
rgl.quads(c(0.55,0.55,1.65,1.65),c(0,2.1,2.1,0),c(1.79,1.79,1.79,1.79),"#B1B1B1")
add.box(0,0.4,0,0.8,0.6,1.2,"#E7F179")
add.box(2.55,2.15,0,0.4,0,0.4,"#E7F179")
add.box(2.6,4.3,0,0.4,0,2.1,"#E7F179")
add.box(1,2.1,0,0.8,0,0.45,"#E7F179")
rgl.quads(c(1.3,1.3,2.1,2.1),c(1,1.8,1.8,1),c(0.01,0.01,0.01,0.01),"#B1B1B1")
rgl.quads(c(0,0,0.8,0.8),c(0,2,2,0),c(0.01,0.01,0.01,0.01),"#7A7E84")
rgl.material(ambient="#FFFFFF")
rgl.light(diffuse="#AAAAAA")

for(i in 1:360) {
  rgl.viewpoint(i);
}

```