Materiały do zajęć z grafiki komputerowej
Wprowadzenie do *pakietu R*

Agnieszka Suchwałko

1 Pobranie i instalacja *pakietu R*

Pakiet ***R*** jest środowiskiem, którego będziemy używali podczas zajęć z grafiki komputerowej. Jest to oprogramowanie dostępne na licencji GPL-2, co oznacza, że jest dla Państwa **darmowe**. Główna strona projektu to http://www.r-project.org/. Dostępne są dwa serwery ze źródłami w Polsce:

- http://piotrkosoft.net/pub/mirrors/CRAN/
- http://r.meteo.uni.wroc.pl/.

Tam należy odnaleźć swój system operacyjny oraz jego wersję (dla Windows są dostępne kompilacje dla 32- lub 64-bitowego systemu). Jeśli nie wiedzą Państwo jaką wersję systemu Windows posiadają, należy zainstalować wersję dla systemu 32-bitowego.

Program jest instalowany z podstawowym zestawem paczek - dodatków pozwalających na łatwiejsze programowanie wybranych zagadnień. Obecnie ilość paczek jest już liczona w tysiącach.

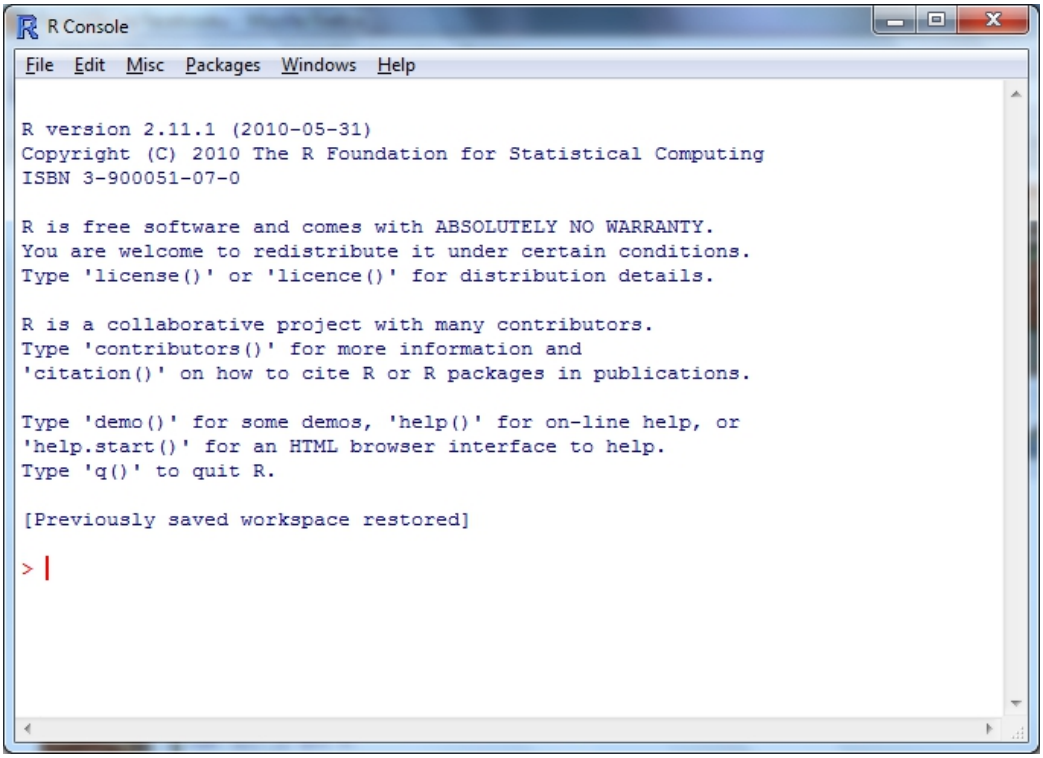
2 Pierwsze uruchomienie

Po zainstalowaniu program jest gotowy do użycia w podstawowej konfiguracji. Po uruchomieniu na ekranie pojawi się główne okno programu, które powinno wyglądać jak Rysunek 1.

Środowisko w takiej postaci, pozwala na wprowadzanie komend (podobnie jak Matlab) i jest bardzo rozbudowanym kalkulatorem.

3 Podstawy - wprowadzenie do środowiska

R to zarówno środowisko jak i język programowania, który został stworzony na bazie języka **S** (dystrybucja komercyjna). **R** nie jest dostępny z GUI



Rysunek 1: Główne okno *pakietu R*.

oferującym wszystkie możliwości systemu, chociaż do pewnych konkretnych zastosowań zostały stworzone interfejsy użytkownika.

Widoczny znak zachęty, informuje użytkownika, że R czeka na polecenie.

```
>
```

Każde polecenie potwierdzamy przez wciśnięcie klawisza ENTER. Jeśli znaku zachęty brak, to albo R wykonuje jakieś obliczenia i należy poczekać, albo wyświetla znak:

```
+
```

który oznacza, że komenda nie została zakończona (zwykle liczba nawiasów otwierających i zamykających się nie zgadza).

Jedną z podstawowych własności, która różni **R** od innych języków programowania jest możliwość nadawania zmiennym nazw posiadających w nazwie

symbol . (kropkę). Zwykle kropka pozwala na dostęp do różnych właściwości bądź elementów obiektu, bądź wywoływania metod na pewnym obiekcie. W **R** kropka jest częścią nazwy, np. jakas.zmienna. Dodatkowo nazwy mogą zawierać polskie znaki, czego nie będziemy wykorzystywać podczas zajęć. Nazewnictwo zmiennych, które jest zgodne z ich wykorzystaniem będzie jednym z wymagań.

3.1 Pomoc w R

Szukanie pomocy jest w **R** prawie intuicyjne - wystarczy znak zapytania! Żeby zapytać **R** o szczegóły odnośnie jakiejś komendy wystarczy wpisać po znaku zachęty ?szukana_komenda, np.:

```
> ?setwd
```

Po zatwierdzeniu komendy klawiszem ENTER pojawi się nowe okno z dokumentacją polecenia w języku angielskim. Dokumentacja zawiera zarówno opis każdego z poleceń, jego parametry, przykłady użycia jak i odniesienia do podobnych poleceń.

Jeśli nie znamy polecenia, ale wiemy co powinno robić, czyli znamy jakieś słowo klucz (po angielsku), możemy skorzystać z drugiej opcji pomocy - dwóch znaków zapytania. Możemy szukać funkcji, które pozwolą nam na odczytania i ustawienie katalogu roboczego, czyli „working directory" po angielsku, np:

```
> ??"working directory"
```

Znów otworzy się okno. Tym razem będzie w nim lista znalezionych przez pakiet funkcji i paczek, w których się znajdują, które odpowiadają wprowadzonemu zapytaniu. Elementy listy są wyświetlane w formacie paczka::funkcja. Powyższy przykład wygeneruje listę jednoelementową. Poszukiwanie hasła directory pozwoli uzyskać znacznie dłuższą listę. Warto zwrócić uwagę na to, że jeśli poszukiwany element jest opisany przez więcej niż jedno słowo trzeba go umieścić w cudzysłowie.

Oba wymienione sposoby szukają wyłącznie wśród paczek zainstalowanych w danym momencie.

3.2 Podstawowe typy danych

W **R** nie trzeba deklarować typu danych, ale za to mamy ich więcej. Żeby przypisać do jakiejś zmiennej wartość wystarczy napisać:

```
> a <- 3
```

Można też przypisać literę:

```
> a <- 'a'
```

Nie jest zabronione używanie znaku równości (=) do przypisywania wartości, ale nie jest ono polecane. Znak = jest stosowany do podawania wartości parametrów funkcji i do operacji logicznych. Z tego powodu strzałka jest używana jako symbol przypisania wartości do zmiennej.

Żeby wypisać zawartość dowolnej zmiennej wystarczy w linii komend wpisać jej nazwę:

```
> a
```

W bardzo prosty sposób tworzy się w **R** wektory. Wektory te, są odpowiednikami tablic znanych z innych języków programowania, a nie wektorów znanych z algebry - mają tylko jeden wiersz i wiele kolumn, nie odwrotnie. Do tworzenia wektorów służy polecenie jedno-znakowe c(). Wektory mogą się składać z danych różnego typu. Kika przykładów poniżej:

```
> w <- c(3, 8, 13)
> w <- c(3, 'a', 'wektor')
> w <- c('ala', 'ma', 'kota')
> w <- c(2.5, 4.0, 15.72)
```

Żeby wyświetlić zawartość całego wektora, wystarczy pisać w konsoli **R**'a jego nazwę. Żeby wyświetlić konkretny element wektora trzeba podać jeszcze jego pozycję w kwadratowych nawiasach (liczymy od 1):

```
> w[2]
```

Wektory jakie znane są z algebry mają w **R** postać macierzy o odpowiednich wymiarach (wektor to przecież macierz). Macierze tworzy się przy pomocy polecenia matrix() z odpowiednimi parametrami, które determinują wymiary macierzy oraz dane, które będą w niej zawarte. Parametr data zawiera dane, nrow odpowiada za liczbę rzędów, analogicznie ncol mówi o liczbie

kolumn. Nie wszystkie parametry trzeba podawać. Na przykład jeśli liczba elementów wektora to 3 i rzędów ma być 3, to łatwo wywnioskować, że kolumna będzie tylko jedna:

```
> M <- matrix(c(1,2,3), nrow=3)
```

Żeby się teraz dostać do poszczególnych elementów wektora M, należy pamiętać, że w rzeczywistości jest on macierzą. Oznacza to, że musimy posłużyć się dwoma wartościami oznaczającymi wiersz i kolumnę w tej kolejności:

```
> M[2, 1]
```

Może też zdarzyć się sytuacja, w której będziemy znali wymiary macierzy, ale nie będziemy jeszcze znali danych jakimi ta macierz będzie wypełniona. Np. znamy wymiary obrazu, ale nie wiemy jaki będzie wynik pewnego przekształcenia, które stworzy nowy obraz. W takim wypadku parametr opisujący dane pozostanie pusty (jak i cała macierz), ale będzie stworzona zmienna w której poszczególne komórki będziemy mogli wpisywać wartości pikseli, kiedy tylko będą znane. Pustą macierz o wymiarach $m \times n$ tworzymy w następujący sposób:

```
> n <- 10
> m <- 3
> M <- matrix(nrow=n, ncol=m)
```

Próbując wyświetlić tak stworzony obiekt otrzymają Państwo macierz o 10 wierszach i 3 kolumnach wypełnioną elementami NA. Obiekt NA w **R** oznacza wartość brakującą, nieznaną. Zgadza się to z naszym założeniem, że elementy macierzy w chwili jej tworzenia nie są znane.

Jest to tylko wąski fragment typów danych dostępnych w **R**, ale na potrzeby tego kursu taka wiedza powinna Państwu wystarczyć. Jeśli będzie potrzeba wprowadzenia innych typów danych zostaną Państwo z nimi zaznajomieni.

3.3 Paczki

Do pracy z obrazami musimy uzbroić świeży system **R** w odpowiednie paczki do obsługi grafiki. Żeby doinstalować poniżej wymienione paczki należy wybrać z menu polecenie Packages → Install packages... Pojawi się okno, w którym mamy wybrać serwer, z którego chcemy ściągać potrzebne paczki. Możemy wybrać dowolny serwer z listy. Z Polski mamy do wyboru te same serwery, na

których można znaleźć sam pakiet **R**, czyli Oświęcim i Wrocław. Po wyborze serwera pojawi się kolejne okno, w którym zostaniemy poproszeni o wybór paczek, które mają zostać zainstalowane. Z tej listy należy wybrać poniżej wymienione paczki:

rimage – przetwarzanie obrazu w podstawowym zakresie: wybrane filtry, wyrównywanie histogramu, fft oraz obsługa formatu JPEG. Dokumentacja: http://cran.r-project.org/web/packages/rimage/rimage.pdf. Wymagania: fftw-2 (http://www.fftw.org/) oraz libjpeg (http://www.ijg.org).

pixmap – import, eksport, rysowanie i inne manipulacje na bitmapach (dokumentacja: http://cran.r-project.org/web/packages/pixmap/pixmap.pdf)

rtiff – obsługa obrazów w formacie tiff (wymaga paczki pixmap, dokumentacja: http://cran.r-project.org/web/packages/pixmap/pixmap.pdf)

R podczas uruchomienia nie ładuje wszystkich zainstalowanych paczek. Żeby skorzystać z możliwości jakie daje paczka w swoim programie należy po każdym uruchomieniu **R**'a podać komendę, która załaduje potrzebną paczkę. Będzie to jedna z pierwszych linii każdego skryptu (programu). Do tego celu użyjemy polecenia library() z parametrem będącym nazwą paczki:

```
> library(rimage)
```

Z pozostałymi paczkami należy postąpić w analogiczny sposób.

3.4 Niezbędnik poleceń

3.4.1 Ścieżki

Do operacji na obrazach będą nam też potrzebne polecenia do ustawiania ścieżek oraz wczytywania i zapisywania plików zawierających grafikę. Ustawianie ścieżek jest niezbędne w systemie Windows. Pod Linux'em uruchamiamy **R**'a w konkretnym katalogu i nie ma potrzeby manipulowania ścieżkami.

Dlaczego musimy ustawiać ścieżki? Dlatego, że zwykle posługujemy się ścieżkami względnymi i wówczas poziomem odniesienia w hierarchii katalogów jest aktualnie ustawiony katalog roboczy - od niego zależy gdzie **R**

będzie szukał i zapisywał wskazane pliki. Korzystając wyłącznie z pełnych (bezwzględnych) ścieżek, nie musimy ustawiać katalogu roboczego, jednak przy przenoszeniu katalogów z programami i danymi musimy zmieniać każdy skrypt aby móc go uruchomić. Więcej o ścieżkach względnych i bezwzględnych na Wikipedii: http://en.wikipedia.org/wiki/Path_%28computing%29 (wersja polska tego hasła nie zawiera przykładów ścieżek względnych!).

Do pobierania informacji o aktualnie ustawionym katalogu roboczym służy polecenie getwd(). Skrót pochodzi od angielskiego zwrotu „get working directory". Polecenie wywołuje się następująco:

```
> getwd()
```

W odpowiedzi dostajemy napis, który zawiera bezwzględną ścieżkę do aktualnego katalogu roboczego. Aby zmienić katalog roboczy należy stworzyć napis, który będzie zawierał bezwzględną ścieżkę do katalogu, mającego służyć jako roboczy (należy zwrócić uwagę na znaki \ oraz /). Do ustawienia katalogu E://uczelnia/grafika jako katalogu roboczego użyjemy polecenia:

```
> setwd('E://uczelnia/grafika')
```

Teraz możemy odwoływać się do podkatalogów i plików przy pomocy ścieżek względnych. Ten katalog będzie podany we wszystkich przykładowych programach. Jeśli nie jest to dla Państwa domyślny katalog, poleceniem setwd() można będzie go zmienić na pożądany.

3.4.2 Rysowanie

Podstawowym poleceniem do rysowania w **R** jest komanda plot(). Jest przeładowana do rysowania niemal wszystkich obiektów. Do rysowania obrazów będących obiektami typu imagematrix również istnieje przeładowana wersja polecenia plot(). Wywołujemy je z parametrem będącym instancją obrazu:

```
> plot(jakis.istniejacy.obraz)
```

3.4.3 Odczyt i zapis plików graficznych

Odczyt i zapis plików graficznych jest zależny od formatu pliku. W zależności od tego jaki jest format pliku graficznego, taka będzie metoda odczytu i zapisu. Pliki w formacie JPEG będę wczytywane przez **R**'a inaczej niż pliki w formacie TIFF, GIF, czy innym. Większość komend wczytujących pliki

graficzne w **R** zaczyna się od słowa read po którym dołożona jest nazwa formatu w jakim jest plik. Dla grafiki w formacie JPEG mamy polecenie z paczki rimage:

```
> read.jpeg(plik.graficzny)
```

Dla plików w formacie TIFF należy skorzystać z polecenia readTiff z paczki rtiff:

```
> readTiff(plik.graficzny)
```

Zapis grafiki nie wymaga żadnych dodatkowych paczek, ponieważ jednym z podstawowych zadań pakietu **R** jest generowanie wykresów do przeprowadzanych analiz i możliwość ich zapisu w wielu formatach. Mamy do dyspozycji 4 formaty plików do wyboru: BMP, JPEG, PNG i TIFF. Parametry dostępne dla każdej z tych funkcji i ich zakresy można znaleźć w pomocy pakietu **R** odnośnie tych funkcji.

Grafikę można zapisać na co najmniej dwa sposoby: *a*) najpierw wyświetlić (poleceniem plot()), a potem zapisać, albo *b*) zapisać bez wyświetlania. Niezależnie od tego który sposób wybierzemy, wyświetlanie oraz zapis odbywa się w **R** z użyciem urządzeń graficznych. Polecenia odnoszące się do urządzeń graficznych zwykle zaczynają się od dev.

Jeśli wybraliśmy opcję a to kod zapisu wygląda tak:

```
> plot(obraz)
> dev.print(jpeg, file="zapisany_obraz.jpg")
> dev.off()
```

Powyższy fragment kodu wyświetli obraz, a następnie zapisze go do pliku w formacie JPEG z domyślnymi parametrami. Szczegóły odnośnie parametrów jakie można ustawić przy zapisie obrazu do pliku można znaleźć poleceniem ?jpeg.

Jeśli chcemy zapisać plik w innym formacie, np. TIFF, wystarczy w linii 2 zmienić pierwszy parametr z jpeg na tiff, oraz rozszerzenie nazwy zapisywanego pliku w drugim parametrze. Ostatnia linia wyłącza urządzenie graficzne, które zostało uruchomione do zapisu pliku.

Jeśli nie chcemy wyświetlić pliku, a tylko go zapisać, czyli wybraliśmy punkt b, to kod wygląda tak:

```
> jpeg(file="zapisany_obraz.jpg")
> plot(obraz)
> dev.off()
```

Powyższy kod działa w następujący sposób: Pierwsza linia uruchamia urządzenie graficzne i powoduje, że wszystkie instrukcje rysujące wywołane w następnych liniach będą rysowały do wskazanego pliku, a nie na ekran. Ostatnia linia to znów wyłączenie urządzenia graficznego.

Zmiana formatu zapisywanego pliku jest równie banalna jak w pierwszym przypadku. Zamiast w linii 1 wywołać funkcję jpeg() wystarczy wywołać np. funkcję tiff().

3.4.4 Polecenia statystyczne

Do wykonania części zadań jest przydatna znajomość podstawowych poleceń ze statystyki. Do takich użytecznych poleceń zaliczają się te liczące średnią i medianę. Do ich wywołania nie trzeba instalować żadnych dodatkowych paczek. Do policzenia średniej zbioru wartości $1, 4, 5, 19, 27$ wystarczy wywołać:

```
> mean(c(1, 4, 5, 19, 27))
```

gdzie z wartości został najpierw stworzony wektor przy pomocy funkcji c(), a następnie funkcja mean(). Średnią można też bez przeszkód policzyć dla macierzy, podając macierz jako parametr funkcji mean(). Analogicznie liczy się medianę dla tego samego zbioru wartości:

```
> median(c(1, 4, 5, 19, 27))
```

Również medianę można policzyć dla macierzy.

3.4.5 Różne polecenia

Polecenia, które mogą się przydać.

rep(a, b) tworzy obiekt typu wektor składający się z wartości a powtórzonych b razy;

abs() liczy wartość bezwzględną wyrażenia podanego jako argument funkcji.

3.4.6 Polecenia paczki rimage

Kilka podstawowych poleceń, które mogą się często przydawać:

imagematrix() tworzy obiekt obrazu z obiektu macierzy. Takie obiekty można modyfikować, wyświetlać itd.

rgb2grey() transformuje obraz kolorowy (rgb) na obraz w skali szarości. Parametrem funkcji jest obiekt obrazu.

3.5 Trudniejsze kawałki kodu

3.5.1 Jak wybrać kawałek obrazu

Chcemy wyciągnąć fragment obrazu (zmienna obraz), który będzie miał taki sam wymiar jak maska, a jego środek znajduje się w i-tym wierszu i j-tej kolumnie. Maska ma rozmiar $n \times n$. Zmienna pol.maski ma wartość $(n-1)/2$. Taki fragment obrazu możemy uzyskać następującym poleceniem:

```
> fragment <- obraz[(i-pol.maski):(i+pol.maski), (j-pol.maski):(j+pol.maski)]
```

Tak skonstruowane polecenie wybiera z obrazu wiersze o indeksach od i-pol.maski do i+pol.maski oraz kolumny o indeksach od j-pol.maski do j+pol.maski. Wybrane punkty są zwracane w postaci macierzy.

3.5.2 Jak wybrać z macierzy wartości spełniające zadany warunek

Bardziej precyzyjnie naszym zadaniem jest wybrać z macierzy (np. fragmentu obrazu) wartości, które różnią się od wartości s (np. ziarno, środek wybranego fragmentu obrazu) o nie więcej niż pewna wartość graniczna t. Najpierw ustalmy konkretnie wartości zmiennych:

```
# macierz 3x3 o siedmiu wartościach $2$ jednej $10$ i jednej $5$
> a <- matrix(c(rep(2,7), 10, 5),3,3)
>
# wartość do której będziemy porównywać wartości macierzy
> s <- a[2,2]
>
# wartość graniczna
> t <- 2
```

Wyznaczenie wartości bezwzględnej różnicy elementów macierzy a i zmiennej s to pierwszy krok, który należy wykonać. Żeby to zrobić, w konsoli trzeba wpisać następujący fragment kodu:

```
> abs(a-s)
```

Już widać, w których miejscach macierzy a są wartości spełniające podany na początku warunek. Musimy je teraz wydobyć.

Policzoną wartość bezwzględną trzeba teraz porównać z podaną wartością graniczną t. Takie porównanie pozwoli nam uzyskać wartość TRUE (prawda) dla wartości macierzy a, które spełniają warunek i wartość FALSE (fałsz) dla wartości, które go nie spełniają. Zrobienie tego to dodanie prostego porównania do poprzedniego fragmentu:

```
> abs(a-s) <= t
```

Zadaniem jest znalezienie takich wartości macierzy a, dla których wartość bezwzględna różnicy ze zmienną s jest mniejsza niż wartość graniczna t. Aby to osiągnąć wystarczy jeszcze tylko wyciągnąć wartości macierzy a, dla których ostatnie polecenie dało wartości TRUE, czyli:

```
> a[abs(a-s) <= t]
```

Efektem tego polecenia jest obiekt typu wektor zawierający wartości macierzy a, które spełniają zadany na początku warunek.

3.6 Uruchamiamy skrypt

Wszystkie programy napisane w języku **R** nazywamy skryptami. Skrypty **R**'owe mają rozszerzenie .R. Oczywiście dla systemu Windows wielkość litery będącej rozszerzeniem nie ma znaczenia, ale pod innymi systemami proszę uważać. Skrypty to programy uruchamiane wewnątrz jakiejś aplikacji. Skrypty nie dają się uruchamiać poza aplikacją dla której zostały napisane. Przykładem języka skryptowego jest Matlab, ale C++ już nie.

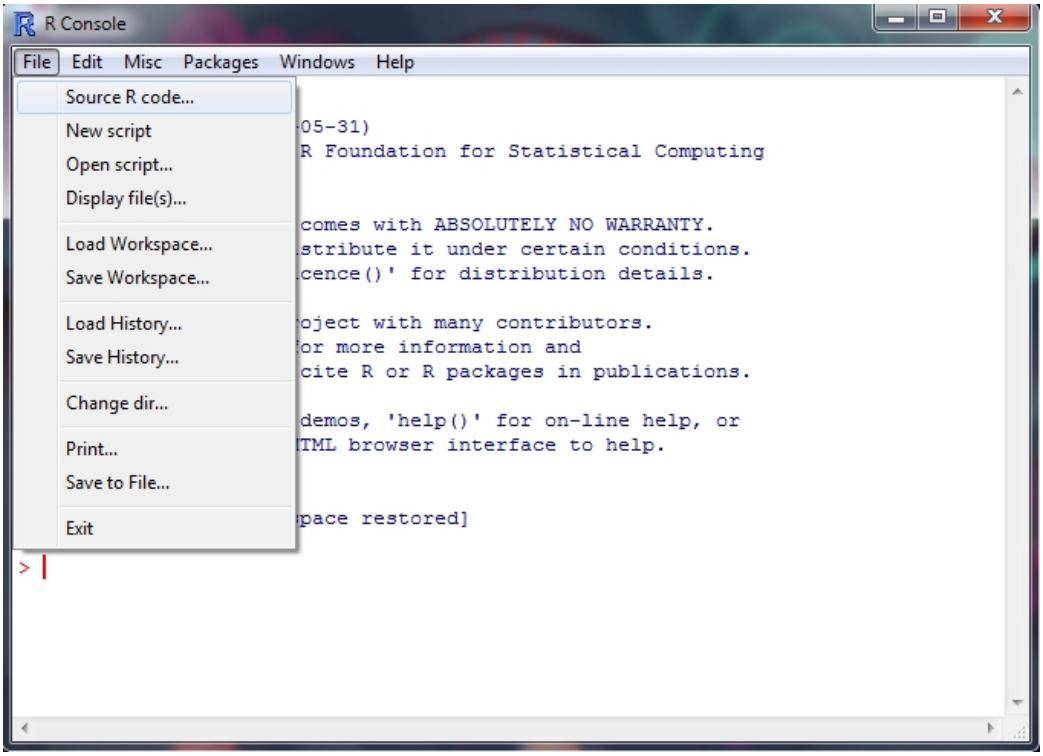
Przed uruchomieniem dowolnego skryptu należy sprawdzić czy mamy ustawioną poprawną dla naszego komputera ścieżkę roboczą (w tym konkretnym skrypcie) i czy mamy w odpowiednich lokalizacjach (katalogach) wszystkie niezbędne do wykonania skryptu pliki, przede wszystkim te z obrazami.

Jednym z przykładowych skryptów jest odczyt_i_zapis.R. Żeby uruchomić dowolny skrypt musimy wybrać z menu File → Source R code... (Rysunek 2) i w oknie, które się otworzy wskazać skrypt, który chcemy uruchomić. Żeby uruchomić skrypt odczyt_i_zapis.R potrzebny jest nam podkatalog o nazwie dane, w który znajduje się plik brain_scan.jpg.

Jeśli wybierzemy skrypt odczyt_i_zapis.R w linii komend pojawi się wpis:

```
> source("E:\\uczelnia\\grafika\\odczyt_i_zapis.R")
```

a program po chwili zacznie rysować w osobnym oknie obraz. Zanim skończy swoją pracę zapisze jeszcze dwa pliki w katalogu dane, bądź innym jeśli inny



Rysunek 2: Uruchamianie skryptu w *pakiecie R*.

został wskazany. Kiedy program zakończy pracę w linii komend pojawi się znak zachęty i **R** będzie gotowy na kolejną komendę.

Żeby uruchomić skrypt nie trzeba niczego klikać. Można w linii komend podać polecenie source() ze ścieżką do odpowiedniego pliku jako parametrem, czyli wpisać dokładnie to co było efektem klikania.

Jeśli chcemy powtórzyć wywołanie skryptu, w którym coś właśnie zmieniliśmy, wystarczy klawisz ↑ i ENTER. W linii komend pojawi się ostatnio wykonane polecenie, a ENTER zatwierdzi jego ponowne wywołanie. W taki sposób można wykonać ponownie dowolną, wcześniej wykonaną komendę.

3.7 Dalsze informacje

Więcej informacji o pakiecie **R** i jego działaniu można znaleźć między innymi w książce „An Introduction to R”, która jest dostępna pod adresem `http:`

//cran.r-project.org/doc/manuals/R-intro.pdf. Istnieją też materiały w języku polskim. Można je znaleźć na stronie http://cran.r-project.org/other-docs.html w sekcji *Polish*.