Анализ данных с R (III).

© С. В. Петров*, Е. М. Балдин†

*p2004r@gmail.com
†E.M.Baldin@inp.nsk.su

Эмблема **R** взята с официального сайта проекта `http://developer.r-project.org/Logo/`

Оглавление

Глава 9

Интерфейс для пользователя с мышкой (GUI на примере)

Про наличие графического пользовательского интерфейса в **R** уже упоминалось в более ранних статьях Linux Format. Теперь покажем как этот интерфейс можно использовать с пользой для дела.

Отдавать команды компьютеру можно исключительно из командной строки **R**, аналогично **R Commander** почти всё позволяет сделать мышкой. Очевидно, что часто требуется нечто среднее между этими двумя «мирами». Назовём этот средний путь «point-and-click» интерфейс. Особенно этот путь эффективен когда мы визуализируем обрабатываемые данные.

Анализ данных часто сводится к написанию своей специализированной программы. Если данных действительно много, то программирование для исследователя становится неизбежной необходимостью. Невозможно предусмотреть всё многообразие данных, пока их не исследуешь. Поэтому процесс анализа идёт параллельно с написанием всё более и более интеллектуальной системы этого самого анализа.

Построение динамического графического point-and-click интерфейса программы анализа должен быть прост, незатейлив и не отвлекать от самого главного, а именно от собственно анализа данных. Он должен быть тесно интегрирован с возможностями командной строки. В полной мере отвечает таким качествам пакет **rpanel**.

9.1. rpanel

Пакет **rpanel** — это крайне простая и одновременно высокоуровневая надстройка над Tcl/Tk расширением среды **R**. Tcl является наравне с Perl и Python «клас-

сическим» скриптовым языком высокого уровня, который обычно используют совместно с кроссплатформенной библиотекой графических элементов Tk. Многие языки программирования как и **R** используют Tk для построения простых графических интерфейсов.

Если пакет **rpanel** отсутствует в системе, то скачиваем и устанавливаем его с помощью команды

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

После этого загружаем саму библиотеку:

```
> library(rpanel)
```

Логика использования пакета довольно проста:

1) С помощью функции rp.control нужно создать объект `panel` и объявить в нём нужные нам переменных;

2) Далее необходимо «населить» объект `rpanel` различными элементами графического интерфейса (ползунками, кнопками, картинками и т. д.) и связанными с ними переменными;

3) В самом конце требуется определить функцию, которая будет выполняться в ответ на взаимодействие пользователя с графическим интерфейсом.

Разберём простейший пример, в котором функция выводит на экран одно из двух: или гистограмму полученных данных, или боксплот («ящик-с-усами») в зависимости от выбранного значения в элементе rp.radiogroup.

Объявим объект `panel`, и данные которые будем отображать:

```
> panel <- rp.control(x=rnorm(50))
```

Здесь мы «исследуем» вектор данных `x`, имеющих гауссовское распределение.

Теперь поместим в `panel` переключатель rp.radiogroup и свяжем его с переменной `plot.type` и с функцией hist.or.boxp:

```
> rp.radiogroup(panel,          # сам объект
+   plot.type,                  # переменная
+   c("histogram", "boxplot"),  # значения переключателя
+   title="Plot␣type",          # название переключателя
+   action = hist.or.boxp)      # пользовательская функция
```

В заключении объявим функцию которая будет выполнятся в ответ на взаимодействие с переключателем:

```
> hist.or.boxp <- function(panel) {
+   if (panel$plot.type == "histogram")
+     hist(panel$x)
```

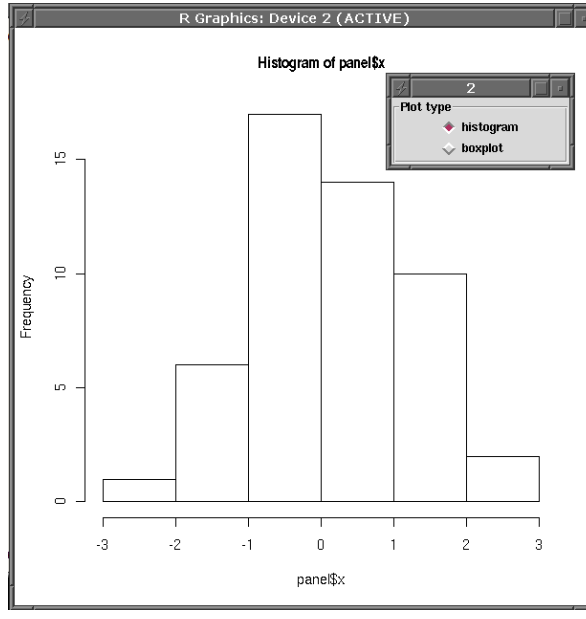

Рис. 9.1. Пример простого переключателя.

```
+   else
+     boxplot(panel$x)
+   panel
+ }
```

Доступ к переменным внутри объекта `panel` производится путём обращения вида `panel$имя_переменной`.

Теперь всё готово. У нас есть готовая панель с переключателем, который в зависимости от выбора показывает либо гистограмму, либо боксплот.

Если хочется, то можно «вмонтировать» график прямо в панель как один из элементов графического интерфейса. Для это понадобится воспользоваться командами rp.tkrplot и rp.tkrreplot из пакета **tkrplot**.

Для установки нового пакета необходимо убедиться, что установлены системные пакеты **tk-dev** и **tcl-dev**:

```
=> sudo aptitude install tk-dev tcl-dev
```

Затем, как обычно, устанавливаем и загружаем сам пакет:

```
> install.packages("tkrplot")
> library(tkrplot)
```

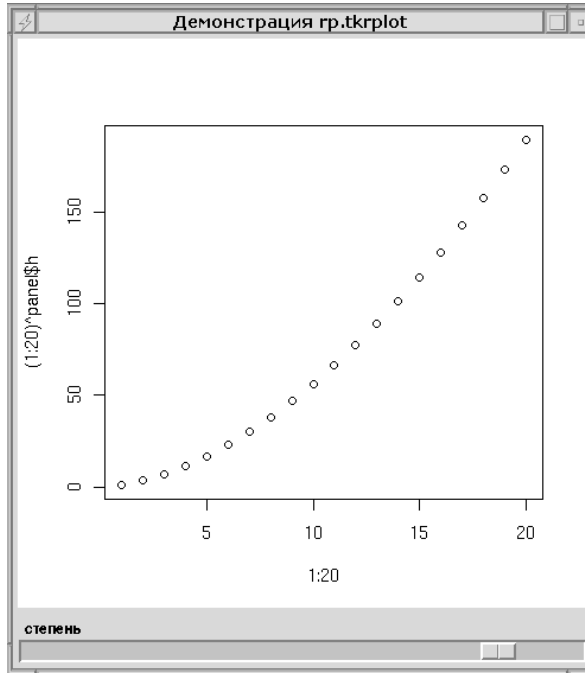

Рис. 9.2. Пример ползунка и графика как графического элемента.

Открываем панель `rpplot` и объявляем переменную `h`:

```
> rpplot <- rp.control(title = "Демонстрация rp.tkrplot",
+                      h = 1)
```

Помещаем на панели `rpplot` график `tkrp`:

```
> rp.tkrplot(rpplot,
+    tkrp,
+    function(panel) plot(1:20,(1:20)^panel$h))
```

В зависимости от параметра `h` на графике отображается степенная функция.

Определяем пользовательскую функцию для перерисовки графика:

```
> redraw <- function(panel) {
+  rp.tkrreplot(panel, tkrp)
+ }
```

Для динамически меняющегося параметра `h` в панель `rpplot` добавим ползунок и свяжем его с пользовательской функцией `redraw`:

```
# Помещаем на панель ползунок
> rp.slider(rpplot,           # панель
+    h,                       # переменная ползунка
+    action = redraw,         # пользовательская функция
+    from = 0.05, to = 2.00,  # параметры ползунка
+    resolution = 0.05,
+    title="степень")       # название ползунка
```

9.2. Tcl/Tk

Если возможностей **rpanel** не хватает, то можно спуститься уровнем ниже и обратиться напрямую к Tcl/Tk. Нужный пакет так и называется **tcltk**.

Тулкит Tk предоставляет богатый набор элементов интерфейса. К ним относятся окна редактирования текста, полосы прокрутки, поля ввода текста, кнопки, метки, перечни и рисунки. Всё это можно комбинировать и получать сложный графический интерфейс.

Простейший пример программы «Здравствуй, мир!»:

```
> library(tcltk)       # Загружаем пакет Tcl/Tk
> tt <- tktoplevel()   # Объявляем контейнер
# Объявляем метку
> lbl <- tklabel(tt,text="Здравствуй, мир!")
> tkpack(lbl)          # Размещаем метку в контейнере
```

В результате выполнения этого кода на экран будет выведено окно с популярной надписью «Здравствуй, мир!». Картинку этого шедевра программистского искусства специально не предоставляем, так как она достаточно банальна. Выполните пример самостоятельно — он не сложен.

Структура построения графического интерфейса всегда похожа: объявляется контейнер, и в нём с помощью менеджера геометрии (tkpack) размещаются дочерние элементы графического интерфейса.

В контейнер можно добавить сколько угодно графических элементов. Процесс построения сложного приложения как раз и заключается в комбинировании элементарных блоков. Например добавим в наше окно кнопку «Выполнить!»:

```
> but <- tkbutton(tt, text="Выполнить!")
> tkpack(but)
```

На кнопку можно нажимать, хотя никаких действий с ней не связано.

Окно контейнера в котором выведен наш пример называется «1» (значение по умолчанию). Изменим это:

```
> tktitle(tt) <- "Моё окно"
```

Рис. 9.3. 4 кнопки.

В Tcl/Tk доступно три менеджера геометрии. Простейший из них placer почти никогда не используется. Более популярны оставшиеся два: packer и grid. У packer есть два параметра: «порядок» и «направление».

В предыдущем примере, если попробовать изменять размеры окна мышью, то видно, что элементы графического интерфейса располагаются по центру окна у верхней его границы. Если начать уменьшать размер окна примера по вертикали, то увидим, что сначала сожмётся и исчезнет кнопка «Выполнить!», а потом начнёт изменяться надпись «Здравствуй, мир!». Попытки ручной коррекции размеров окна приводят к ещё одному эффекту: теперь окно не будет автоматически корректировать свои размеры при размещении в нем новых элементов. Восстановить автоматическую подгонку размеров окна можно выполнив команду:

```
> tkwm.geometry(tt,"")
```

Элементы графического интерфейса можно привязать к любой из сторон основного окна. Например, вот так (рис. 9.3):

```
> tkdestroy(tt)          # Убираем предыдущий пример
> tt <- tktoplevel()
# Английские названия используются и как имена кнопок,
#и как параметры
> edge <- c("top","right","bottom","left")
> for ( i in 1:4 )         # Четыре кнопки
+   tkpack(buttons[[i]], side=edge[i],
+          fill="both")
```

Элемент интерфейса при этом занимает достаточную для своего расположения полосу, параллельно стороне привязке. Указанный параметр `fill` заставляет элемент занять полосу целиком. Аналогично, если указать параметр `expand=TRUE`, то элементы интерфейса займут всё свободное место.

Когда размещаемый графический элемент интерфейса не занимает полосу целиком, то его можно «заякорить», подав «морские» команды вида «n» (Nord или Север или Верх основного окна) или «sw» (Зюйд-Вест или Внизу-слева).

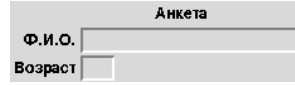

Рис. 9.4. Пример использования tkgrid.

Если нам к примеру понадобится разместить в основном окне область ввода текста с полосой прокрутки, то мы:

1) Добавляем полосе прокрутки свойство `fill="y"`;

2) Добавляем области ввода текста `fill="both"`и `expand=TRUE`, чтобы она заняла все свободное пространство;

3) Первой в основное окно размещаем полосу прокрутки, а потом область ввода текста, чтобы полоса прокрутки не сокращалась при изменении размеров окна пользователем.

Если же вы задумались как объединить ряд кнопок и область ввода текста, то нет ничего проще: надо всего лишь объединить кнопки во фрейм. Каждый фрейм — это контейнер со своим собственным менеджером геометрии. Гибкость в размещении графических элементов интерфейса достигается этими средствами весьма изрядная. Однако, иногда такой способ требует от пользователя изощрённых подходов. Например, достаточно трудно выровнять ряды кнопок одновременно и по вертикали и по горизонтали. В случае если вас замучила именно эта проблема, то вас спасёт менеджер геометрии grid. Он размещает графические элементы интерфейса стройными рядами и колоннами.

Следующий пример рекомендуется выполнить построчно, наблюдая за изменением вывода в основное окно (рис. 9.4):

```
> t2 <- tktoplevel()
> heading <- tklabel(t2, text="Анкета")
> l.name <- tklabel(t2, text="Ф.И.О.")
> l.age <- tklabel(t2, text="Возраст")
> e.name <- tkentry(t2, width=30)
> e.age <- tkentry(t2, width=3)
> tkgrid(heading, columnspan=2)
> tkgrid(l.name, e.name)
> tkgrid(l.age, e.age)
> tkgrid.configure(e.name, e.age, sticky="w")
> tkgrid.configure(l.name, l.age, sticky="e")
```

С таким достаточно приличным и доставшимся «малой кровью» графическим интерфейсом уже хочется общаться. А именно: получать данные от построенного интерфейса и отправлять в него результаты обработки. Для этого есть два основных способа «управляющие переменные» и «обратные вызовы» (callbacks).

Для создания Tcl управляющей переменной служит функция tclVar(), которая создаёт объект класса `tclVar`. На стороне Tcl созданные переменные будут поименованы автоматически, и в обычной ситуации беспокоится о задании какого-то специального имени переменной нет необходимости. Получить значение созданной Tcl переменной можно с помощью функции tclvalue().

Чтобы поместить содержимое в созданную переменную, не столкнувшись при этом с проблемами кавычек и других символов имеющих служебное значение в строках Tcl, следует задействовав функцию tclObj():

```
# Создаем переменную, в R имя foo
> foo <- tclVar()
# Заполняем foo символами
> tclObj(foo) <- c(pi,exp(1))
# Выводим значение foo
> tclvalue(foo)
[1] "3.141592653589793␣2.718281828459045"
# Разбираем строку на символьные значения
> as.character(tclObj(foo))
[1] "3.141592653589793" "2.718281828459045"
# Разбираем строку на вещественные значения
> as.double(tclObj(foo))
[1] 3.141593 2.718282
# Чего в строке нет, того нет
> as.integer(tclObj(foo))
[1] NA NA
# Округляем до целых самостоятельно.
> round(as.double(tclObj(foo)))
[1] 3 3
```

Доступ через tclObj() экономит значительные усилия. Например, так выглядит подготовка списка названий месяцев:

```
# Имена месяцев в представлении R
> month.name
 [1] "January"   "February"  "March"     "April"
 [5] "May"       "June"      "July"      "August"
 [9] "September" "October"   "November"  "December"
# Теперь строка пригодная для экспорта в Tcl
> paste(month.name, collapse="␣")
[1] "January␣February␣March␣April␣May␣June␣July
+␣August␣September␣October␣November␣December"
```

Заменяется достаточно простым выражением:

```
> mylist <- tclVar()
```

Рис. 9.5. Список месяцев.

Рис. 9.6. Список месяцев (множественный выбор).

```
> tclObj(mylist) <- month.name
> tclObj(mylist)
<Tcl> January February March April May June July
+ August September October November December
```

Теперь очень просто имея переменную Tcl общаться с созданным интерфейсом. Построим форму со списоком названий месяцев:

```
> tkpack(lb <- tklistbox(tt <- tktoplevel(),
+                listvariable = mylist))
```

В любой момент мы можем узнать какой месяц выбран в данный момент (рис. 9.5). При этом помним, что в Tcl индексы начинаются с 0:

```
> as.character(tclObj(mylist))[as.integer(tkcurselection(lb))+1]
[1] "July"
```

Возвращаемый из Tcl вызовов значения так же являются объектами `tclObj`. Например, мы можем включить в окне с нашим построенном списком месяцев режим множественного выбора:

```
> tkconfigure(lb, selectmode="multiple")
```

Выбрав нужные позиции списка (рис. 9.6) мы легко получаем их в виде номеров строк:

```
> as.integer(tkcurselection(lb))
[1] 1 3 5
```

Теперь попробуем привязать к кнопке какое либо действие, например пусть кнопка сообщает о своем нажатии в **R**, и меняя надпись:

```
# Создаём основное окно
> t <- tktoplevel()
# Создаем кнопку
> b <- tkbutton(t, text = "Не жми на меня!")
# Размещаем её в основном окне
> tkpack(b)
# Объявляем функцию кнопки
> change.text <- function() {
+   cat("ОГО!\n")
+   tkconfigure(b, text = "Говорю тебе: не жми на меня!")
+ }
# Прикрепляем функцию к кнопке
> tkconfigure(b, command = change.text)
```

Сейчас кнопка реагирует на нажатия, а именно, при каждом нажатии на кнопку в консоли появляются восклицания «ОГО!» и меняется текст на самой кнопке. Это простой пример — попробуйте его в динамике.

Кнопку для краткости можно сразу объявить с нужной нам функцией:

```
> b <- tkbutton(t, text="Не жми на меня!",
+   command=expression(cat("ОГО!\n"),
+   tkconfigure(b, text = "Говорю тебе: не жми на меня!")))
```

Каждый из элементов графического интерфейса Tk имеет массу полезных применений. Например, окно для ввода текста представляет из себя практически готовый текстовый редактор. В нём можно вводить и редактировать введенный текст, перемещаться по тексту клавишами курсора, выделять куски текста для операций копирования и вставки.

Что бы получить текст из окна надо выполнить команду

```
> tkget(txt, "0.0", "end")
```

Поскольку Tcl имеет дело со строками, то и передаём мы только строки. «Первая строка, первый символ» — это `0.0`, а `end` — это, соответственно, конец текста. Если мы захотим получить текст в **R** также разбитым на строки, как в текстовом окне, то нам надо конвертировать Tcl строку в вектор состоящий из строк:

```
> strsplit(tkget(txt, "0.0", "end"),"\n")
```

Похожим способом мы можем получить и выделение текста в окне:

```
> X <- tkget(txt, "sel.first", "sel.last")
```

Здесь нам уже не обойтись без проверки ошибок, а то вдруг выделение текста не было сделано? Проверить это просто, путём сравнения с пустой строкой:

```
> tktag.ranges(txt, "sel") != ""
```

Можно получить информацию и о нажатии кнопки мыши, например, получить координаты её курсора и не переместить при этом текстовый курсор:

```
> tkbind(txt, "<Control-Button-1>",
+   expression(function(x,y) cat(x,y,"\n"),
+                break))
```

Добавить текст, к примеру, в конец текста расположенного в окне:

```
> tkinsert(txt, "end",    # Добавляем в конец текста
          string)          # Строка которую добавляем
```

Естественно, что если мы хотим передать в Tcl несколько строк сразу, то нам понадобится объединить их, например вот таким способом:

```
> tkinsert(txt, "end",
          paste(deparse(ls), collapse="\n"))
```

Если нам надо переместить текстовый курсор в определенное место окна, то мы можем воспользоваться сочетанием:

```
# Переместить курсор в начало первой строки
> tkmark.set(txt, "insert", "0.0")
# Сделать видимой область с курсором
> tksee(txt, "insert")
```

Мы также можем контролировать куда будет смещаться вставляемый текст, если ему не хватает места. По умолчанию он вставляется перед выделением, но это легко изменить:

```
> tkmark.gravity(txt, "insert", "left")
```

Если наше приложение достаточно сложное, то ему безусловно понадобится меню. Меню у нас тоже есть — это отдельный элемент. Можно использовать всплывающее меню, но чаще меню размещают в основном окне в качестве полоски, или кнопки «Пуск», или как меню вложенное внутри главного меню приложения.

Наиболее практично меню создавать по шагам. Сначала нужно объявить меню командой tkmenu, затем наполняем его элементами меню с помощью tkadd. Это позволит легко воспользоваться многочисленными опциями команд создания элементов меню.

Существует множество разновидностей элементов меню. Это эквивалентные уже знакомым нам кнопкам `command entry`. Вложенное меню получают вставляя `Cascade entries`. `Checkbutton` и `radiobutton` позволяют делать различные

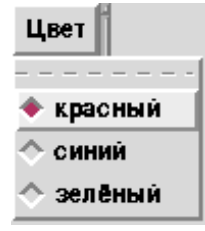

Рис. 9.7. Пример меню.

варианты выбора параметров. Есть и доступ к элементам организующим содержимое меню: это `separators`, отделяющие элементы меню визуально друг от друга, и `tear-offs`, позволяющие открeплять меню от родительского и делать его плавающим.

Чтобы не быть голословными создадим кнопку меню с тремя `radiobutton` (рис. 9.7):

```
# Объявляем переменную изменяемую из меню
> color <- tclVar
#и её значение по умолчанию
> color<-"красный"
> tt <- tktoplevel()
# Размещаем кнопку меню
> tkpack(mb <- tkmenubutton(tt, text="Цвет"))
# Объявляем вложенное меню
> m <- tkmenu(mb)
# Присоединяем вложенное меню к кнопке
> tkconfigure(mb,menu=m)
# Создаём элементы меню
> for ( i in c("красный", "синий", "зелёный"))
   tkadd(m, "radio", label=i, variable=color, value=i)
```

Опросить что же выбрано пользователем из вариантов меню, можно обратившись к функции

```
> tclvalue(color)
[1] "красный"
```

Вот теперь мы во всеоружии, то есть можем не просто наслаждаться сами высокопродуктивной обработкой данных в **R**, но и сделать свои наработки доступными для не столь искушённых в **R** окружающих.

К сожалению нет в мире совершенства. Вдруг нам понадобится сделать доступным наше предложение через корпоративный вебсервер? Или нам не нравится

вид Tk? Если Вы еще не привыкли к тому что в **R** как в Греции есть всё, то вот оно решение: **gWidgets**.

Вместо того, что бы опускаться вглубь конкретной библиотеки графического интерфейса, можно просто сделать свое приложение библиотеко-независимым. Но это тема для отдельного рассказа.