Глава 1

Введение в R

Вы ещё анализируете данные по старинке методом внимательного вглядывания в набор точек на графике, изредка подгоняя их к прямой с помощью многочисленных «мышекликов»? Пусть это делает за Вас компьютер, и **R** — это именно тот язык, который позволит разъяснить ему Вашу проблему.

1.1. Что такое R

Прежде всего **R** — язык программирования для статистической обработки данных и работы с графикой, но в тоже время это свободная программная среда с открытым исходным кодом, развиваемая в рамках проекта GNU. В любом дистрибутиве GNU/Linux, если его целью не является размещения всего дистрибутива на дискетке, можно найти эту среду[1].

R применяется везде, где нужна работа с данными. Это не только статистика в узком смысле слова, но и «первичный» анализ (графики, таблицы сопряжённости), и продвинутое математическое моделирование. **R** без особых проблем может использоваться и там, где сейчас принято использовать коммерческие программы анализа уровня MatLab/Octave. С другой стороны вполне естественно, что основная вычислительная мощь **R** лучше всего его проявляется при статистическом анализе: от вычисления средних величин до вейвлет-преобразований временных рядов.

География использования **R** очень разнообразна. Трудно найти американский или западноевропейский университет, где бы не работали бы с **R**. Очень многие серьёзные компании (например, Boeing) устанавливают **R** для работы. **R** для статистиков — это действительно глобально.

[1] В Debian GNU/Linux базовый пакет носит имя **r-base**, а подключаемые модули проще искать по акрониму «cran».

1.2. Немного истории

R возник как свободный аналог среды S-PLUS, которая в свою очередь является коммерческой реализацией языка расчётов S.

Язык S — довольно старая разработка (почти как TEX). Он возник ещё в 1976 году в компании Bell Labs, и был назван, естественно, «по мотивам» языка C. Первая реализация S была написана на FORTRAN и работала под управлением операционной системы GCOS. В 1980 г. реализация была переписана под UNIX, и с этого момента S стал распространяться, пока ещё в основном в научной среде. Начиная с третьей версии (1988 г.), коммерческая реализация S называется S-PLUS. Последняя в настоящее время распространяется компанией Insightful, и доступна под Windows и различные версии UNIX, естественно, за плату, причём весьма и весьма немаленькую[2]. Собственно говоря, именно высокая цена и сдерживала широкое распространение этого во многих отношениях замечательного продукта. Тут-то и начинается история **R**.

В августе 1993 г. двое молодых новозеландских учёных анонсировали свою новую разработку, которую они назвали **R**. По замыслу создателей (Robert Gentleman и Ross Ihaka), это должна была быть новая реализация языка S, отличающаяся от S-PLUS некоторыми деталями, например, обращением с глобальными и локальными переменными, а также работой с памятью. Фактически, они создали не полный аналог S-PLUS, а новую «ветку» на «дереве S». Многие вещи, которые отличают **R** от S-PLUS, связаны с влиянием языка Scheme[3].

Сначала проект развивался довольно медленно, но когда в нём появилось достаточно возможностей, в том числе уникальная по лёгкости система написания дополнений или пакетов, всё большее количество людей стало переходить с S-PLUS на **R**. Когда же, наконец, были устранены свойственные первым версиям проблемы с памятью, то среди пользователей **R** стали появляться и «любители» других статистических пакетов (прежде всего тех, которые имеют интерфейс командной строки: SAS, Stata, SYSTAT). Количество книг, написанных про **R**, за последние годы выросло в несколько раз, а количество пакетов уже приближается к полутора тысячам.

Идея центральной системы хранения и распространения пакетов — CRAN известного как Comprehensive R Archive Network (`http://cran.r-project.org/`) была заимствована из TEX-сообщества (CTAN, или Comprehensive TeX Archive Network; аналогичной схемой пользуется и Perl-сообщество: CPAN или Comprehensive Perl Archive Network). Все три упомянутых проекта объединяет одно: стабильная база и множество дополнений. В отличие от добавления новой функциональности в монолитную программу, качественный пакет может сравнительно легко написать один человек за вполне обозримый промежуток времени.

[2] Версия S-PLUS для UNIX стоит, например, $6500

[3] Scheme — это функциональный язык программирования. Один из наиболее популярных диалектов языка Lisp

1.3. Как скачать и установить R

Поскольку **R** — свободная система, то его можно скачать и установить совершенно свободно. Есть несколько способов, которые зависят от того, какая у Вас операционная система.

GNU/Linux Если у Вас какой-нибудь из распространённых дистрибутивов, то **R** наверняка входит в репозитарий «прилагающихся» к Вашей системе пакетов. Единственное, что нужно учесть — обновление пакетов часто отстаёт от выхода версий [4]. На момент написания статьи современной версией была 2.6.0.

Версия **R** нумеруется тремя числами, первые два — это главная версия, которая обновляется два раза в год. С каждой главной версией в **R** привносятся изменения, причём часто довольно значительные. Как правило, это множество новых команд, исправленные алгоритмы выполнения старых, и, разумеется, исправления ошибок. К недостаткам смены версии можно отнести возможные проблемы с обратной совместимостью. Естественно, разработчики стараются минимизировать такие изменения. С другой стороны, написанные на **R** программы пяти-семилетней давности, как правило, работают без проблем. В общем и целом, мораль такова: обновляйте **R** смело, но при этом всегда читайте список изменений.

На каждую главную версию выходит, как правило, две минорных версии (нулевая и первая). Первая минорная версия обычно ничего нового, кроме исправления ошибок, не содержит. Таким образом, если Вы хотите всегда иметь самую свежую версию, то репозитарий пакетов особенно в случае стабильных дистрибутивов не годится. В этом случае надо будет скачивать **R** из CRAN (`http://cran.r-project.org/`). У этого сайта довольно много зеркал, так что можно выбрать подходящее.

Компиляция **R** из исходного кода очень проста и не требует каких-то экзотических библиотек. Поэтому, если процесс компилирования не пугает, то собрать **R** из исходников не составит труда.

Mac OS X Для того чтобы начать работать с **R** в Mac OS X, надо сначала убедиться, что у Вас последняя версия этой операционной системы. Последние версии **R** выходят только под Mac OS 10.4.x («Tiger», версия под 10.5.x «Leopard» сейчас в разработке). Установочный пакет скачивается с CRAN. Имейте в виду, что графическая оболочка **R** для Mac («**R** Mac GUI») обновляется быстрее, чем сам **R**, поэтому разработчики предусмотрели возможность скачивания и установки оболочки отдельно. Установка происходит практически также, как установка любой программы под Mac.

[4] Цикл до выхода новой версии **R** длится примерно 3 месяца.

Ещё надо отметить, что **R** завоевал себе популярность не в последнюю очередь тем, что он запускался под Macintosh, в то время как S-PLUS под эту платформу был недоступен.

Windows Для того чтобы запускать **R**, можно иметь любые версии этой операционной системы, начиная с Windows 95. Как и в предыдущем случае, установочный пакет скачивается с CRAN. Опять-таки, установка напоминает обычную для среды Windows. Есть инсталлятор, который задаёт несколько вопросов, от ответов на которые ничего серьёзного не зависит. После инсталляции появляется ярлык, щёлкая на котором, можно запускать **R**.

Здесь интересно заметить, что Windows-инсталляция **R** является, как это принято сейчас говорить, «portable», и может запускаться, например, с USB-флешки или CD. **R** делает пару записей в реестр, но для работы они совершенно не критичны. Единственное, надо иметь в виду то, что рабочая папка должна быть открыта для записи, иначе некуда будет записывать результаты работы.

Есть одна, важная для всех операционных систем особенность: **R** (в отличие от того же S-PLUS) держит все свои вычисления в оперативной памяти, поэтому если в процессе работы, скажем, выключится питание, то результаты сессии, не записанные явным образом в файл, пропадут. Эта особенность, к сожалению, также не позволяет **R** работать с действительно большими объёмами (порядка сотен тысяч и более записей) данных, отдавая их на откуп гораздо менее удобной системе анализа ROOT[5] (`http://root.cern.ch`)

1.4. Как начать работать в R

Предполагается, что программная среда **R** уже установлена, поэтому приступим...

1.4.1. Запуск

Опять-таки, каждая операционная система имеет свои особенности работы. Но в целом можно сказать, что под все три упомянутые выше операционные системы существует так называемый «терминальный» способ запуска, а под Mac и Windows имеется и штатная GUI с некоторыми дополнительными возможностями (разными в разных ОС).

Терминальный способ прост: достаточно в командной строке набрать:

```
=> R
```

[5] В сентябрьском номер LInux Format за 2006 год была статья посвящённая ROOT. Сама статья доступна в открытом доступе по адресу `http://www.inp.nsk.su/~baldin/DataAnalysis/`

```
R version 2.4.0 Patched (2006-11-25 r39997)
Copyright (C) 2006 The R Foundation for Statistical
Computing ISBN 3-900051-07-0

R -- это свободное ПО, и оно поставляется безо всяких
гарантий. Вы вольны распространять его при соблюдении
некоторых условий. Введите 'license()' для получения
более подробной информации.

R -- это проект, в котором сотрудничает множество
разработчиков. Введите 'contributors()' для получения
дополнительной информации и 'citation()' для ознакомления
с правилами упоминания R и его пакетов в публикациях.

Введите 'demo()' для запуска демонстрационных программ,
'help()' -- для получения справки, 'help.start()' --
для доступа к справке через браузер. Введите 'q()', чтобы
выйти из R.

>
```

и появится приглашение в виде символа >. Теперь можно приступать к работе.
► Под Windows это немного сложнее — необходимо вызывать программу **R.exe**, причём для этого надо либо «находиться» в той же папке, где находится эта программа, либо путь к этой папке должен быть прописан в переменной PATH. Кроме того, для того чтобы в русскоязычной Windows вводный экран был читаем, надо сменить в окне терминала кодировку (`chcp 1251`) и поставить соответствующий шрифт (скажем, Lucida Console).

Если терминал запущен без графической среды, то все изображения будут «скидываться» в один многостраничный PostScript-файл `Rplots.ps`. Под Mac это будет происходить даже если X11 запущен, так что полноценно использовать **R** под Mac можно только в GUI-варианте. Терминальный запуск под Windows таких ограничений не имеет.

В дальнейшем договоримся, что под «сессией **R** » мы будем иметь в виду терминальный запуск под X11 в GNU/Linux и GUI-запуск в Windows и Mac. GUI под эти операционные системы построены так, что они всё равно запускают терминал-подобное окно. Общение с **R** возможно только в режиме диалога, «команда-ответ». Полноценного GUI[6] с **R** не поставляется, хотя существуют многочисленные попытки создать такую систему. Пока все эти попытки далеки от завершения. Возможно, это и к лучшему, так как система из меню-окошек-опций

[6]Скорее всего ближе всего подошла к рабочему состоянию программа **Rcommander**, о котором мы ещё поговорим.

не способна заменить полноценный интерфейс командной строки, особенно в случае таких сложных систем как **R**. Интересно, что S-PLUS имеет очень приличный GUI, но если открыть любой учебник по этой системе, то можно заметить, что автор настоятельно рекомендует пользоваться командной строкой.

► Для **R** GUI под Windows существует возможность запустить его в много- (MDI) и однооконном (SDI) режимах. Для использования настоятельно рекомендуется однооконный режим (SDI), тем более что все остальные реализации **R** только его и «умеют».

При запуске **R**, первым делом появляется вводный экран. Если выполнение программы производится в окружении с русской локалью, то стандартное введение будет выведено по русски. Если по какой-то причине требуется нелокализованный **R**, то в этом случае можно установить переменную LANGUAGE, а именно создать файл `Renviron.site`, в который внести строчку

```
LANGUAGE=en
```

Этот файл должен находиться в так называемой «домашней» директории **R**. Более подробно про это можно узнать, если прочитать помощь по команде `Startup`. Под Linux вызвать нелокализованный R другими способами, например, указав язык локали прямо в командной строке:

```
=> LANG=POSIX R
```

1.4.2. Первые шаги

Перед тем как начать работать, надо понять, как выйти. Для этого достаточно ввести одну команду и ответить на один вопрос:

```
> q()
> Save workspace image? [y/n/c]: n
```

Уже такой простой пример демонстрирует, что в **R** любая команда — это функция, которой можно передать аргумент. Даже если аргумент не указан, то скобки всё равно надо указать. Если этого не сделать, то вместо выхода из **R** на экран будет выведено определение функции:

```
> q
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
<environment: namespace:base>
```

Для того чтобы узнать как правильно вызывать функцию следует обучиться пользоваться встроенной справкой. Есть два пути. Первый — вызвать команду справки:

```
> help(q)
```

или

```
> ?q
```

Текст справки будет выведен в основном окне программы. Если внимательно прочитать текст, то становится ясно, что выйти из **R** можно и не отвечая на дополнительный вопрос, если ввести:

```
> q("no")
```

Зачем же нужен этот вопрос? Или, другими словами, что будет, если ответить положительно? В этом случае в рабочую папку **R** (ту, из которой он вызван), запишутся два файла: бинарный `.RData` и текстовый `.Rhistory`. Первый содержит все объекты, созданные за время сессии. Второй — полную историю введённых команд. **R** работает с историей команд стандартным образом: доступ к предыдущей команде осуществляется через клавишу-стрелку «вверх», а поиск в истории команд по комбинации `^r`. Если при выходе сохранить файл `.Rhistory`, то команды из этой сессии команды будут доступны и в следующей при условии, что **R** будет вызван из той же самой папки. И наоборот, если случайно сохранить рабочую среду (эти два файла), то при следующем старте они загрузятся автоматически. Иногда такое поведение **R** становится причиной различных недоумений, так что необходимо быть *внимательным*!

Итак, как войти и как выйти, уже понятно. Осталось ещё немного сказать про помощь. Её можно вызвать несколькими разными способами. Во-первых, с помощью команд `?` или `help()`, как написано выше. Во-вторых, можно вызвать команду `help.start()`. В этом случае откроется окно браузера, в котором будет демонстрироваться так называемая HTML-помощь. Основное её преимущество перед обычной текстовой помощью в том, что её разделы соединены гиперссылками (в Mac OS X такая помощь вызывается обычными командами). В третьих, вместе с **R** устанавливается несколько руководств в формате PDF. Их можно найти в папке, содержащей документацию Наконец, часто бывает нужна «обратная» помощь — Вы знаете, что Вы хотите, но не знаете, как это сделать (какую команду вызвать). В этом случае могут помочь две команды: `help.search()` и `apropos()`. Вот как их надо вызывать:

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

Результатом исполнения команды будет список команд с кратким описанием их действия. Команда `apropos()` выдаст просто список команд, содержащих строку, которая была в кавычках. Кстати, обратите внимание на кавычки. Их надо обязательно использовать в этих командах. Кавычки можно использовать и в обычных командах помощи, например, `help("q")` или `?"q"`, причём иногда эти команды без кавычек просто не работают (например, нельзя получить справку по символу плюса, если ввести `?+`, надо вводить `?"+"`). Если ничего не помогает, и найти нужную функцию не удаётся, то приходится обращаться за справкой в Интернет или в список рассылки R-help.

1.4.3. Калькулятор-перекресток

Так назвал один из вводных разделов своей книги «Introduction to **R** » один из создателей системы, Peter Dalgaard. Это значит просто, что **R** можно использовать в том числе и как обычный калькулятор. Например:

```
> 3+2
[1] 5
> 16+3/5-11*8^2
[1] -687.4
> ((((16+3)/5)-11)*8)^2
[1] 3317.76
```

Для знакомых с интерактивными языками программирования типа Python здесь нет ничего особенно нового. Единственная любопытная деталь — это единичка в квадратных скобках. Она означает номер элемента вектора. Отсюда сразу два логических вывода:

1) **R** результат любой операции с числами трактует как вектор единичной длины. Скаляров в **R**, вообще говоря, просто нет;

2) Элементы векторов нумеруются с единицы, а не с нуля, как во многих языках программирования.

Для читателей, менее знакомых с программированием, отметим, что порядок арифметических действий в **R** стандартный, знакомый со школьной математики. Скобки (раскрывающиеся изнутри наружу) позволяют этот порядок действий менять:

```
> # Первый пример
> 3/7
[1] 0.4285714
> 3/7-0.4285714
[1] 2.857143e-08
> # Второй пример
> sqrt(2)*sqrt(2)
[1] 2
> (sqrt(2)*sqrt(2))-2
[1] 4.440892e-16
```

Эти примеры посложнее, кроме того, в них есть «подводные камни». Почему разность `3/7-0.4285714` не равна нулю, должно быть понятно всем знающим арифметику. **R** при выводе на консоль «невидимо» использует функцию `print()`, которая округляет бесконечную периодическую дробь 0,(428571). Второй пример интереснее. Произведение двух квадратных корней из двойки должно давать 2, что и происходит. Однако если вычесть из результата 2, получится какое-то

очень маленькое число ($4{,}440892 \times 10^{-16}$). Это происходит оттого, что вычисления выполняются на компьютере, который только притворяется, что работает с дробями, в то время как на самом деле оперирует только с целыми числами. Любая компьютерная система расчётов работает подобным образом, и с этим можно только смириться. Ещё один момент: приведённые примеры показывают, как можно пользоваться символом комментария (#).

И ещё немного о работе с аргументами на примере команды `round()` («округлить»). Она имеет два аргумента: число, которое нужно округлить, и значение `digits`, сообщающее, до какого знака округлять. Система аргументов работает разумно, так что все равно, что́ написать:

```
> round(1.5, digits=0)
[1] 2
> round(1.5, d=0)
[1] 2
> round(d=0, 1.5)
[1] 2
> round(1.5, 0)
[1] 2
> round(1.5,)
[1] 2
> round(1.5)
[1] 2
```

Это происходит благодаря тому, что есть значения аргументов по умолчанию. Например, в данном случае значение аргумента по умолчанию `digits` — «0». Об этом говорит и результат вывода команды `args()`:

```
> args(round)
> function (x, digits = 0)
```

Можно заметить также, что некоторые аргументы имеют имена. В результате аргументы можно вызывать не по порядку, а по именам. Имена же можно сокращать вплоть до одной буквы, но только если нет разных аргументов, которые от такого сокращения станут неразличимы. Можно вызывать аргументы по порядку, через запятую (не забудьте, что для десятичных дробей используется точка!), и тогда разрешается не использовать имён.

1.5. Скрипты

Просто открыть сессию **R** и вводить в окно программы команды, одну за другой — это лишь один из возможных способов работы. Гораздо более продуктивный метод, который является заодно и серьёзнейшим преимуществом **R** — это создание скриптов. Иначе говоря, программ, которые потом загружаются в **R** и

интерпретируются им. С самого начала работы следует создавать скрипты, даже для таких задач, которые кажутся пустяковыми — это в будущем значительно сэкономит Ваше бесценное время. Создание скриптов по любому поводу и даже без особого повода — одна из основ культуры работы в **R**.

Создать скрипт очень просто. Допустим, после открытия сессии была введена необходимая для получения искомого результата последовательность команд. Чтобы сделать свою работу воспроизводимой — надо просто сохранить историю команд. Лучше всего это сделать с помощью команды:

```
> savehistory(file="myscript.r")
```

После этого в текущей папке появляется файл `myscript.r`, который можно отредактировать любимом текстовом редакторе, а потом загрузить в **R** командой:

```
> source("myscript.r", echo=TRUE)
```

Опция `echo` добавлена для того чтобы можно было видеть сами команды, а не только результат их выполнения.

Есть ещё более эффективный способ работы: Вы открываете Ваш скрипт в текстовом редакторе, а потом посылаете отдельные его строки прямо в **R**. Есть несколько редакторов, которые умеют так делать. Во-первых, это **Emacs** с установленным пакетом **ESS** («Emacs Speaks Statistics»). Прелесть этой системы в том, что **R** запускается прямо в одном из окон редактора. Во-вторых, штатные **R** GUI под Windows и под Mac также имеют встроенные редакторы скриптов. К сожалению, Windows-редактор не подсвечивает синтаксис, и вообще довольно неудобен. Вместо него тем пользователям, которых пугает перспектива освоения **Emacs**, можно порекомендовать отличный редактор **Tinn-R**, специально «заточенный» под такую работу.

Скрипт **R** можно выполнить и не запуская интерактивную сессию. Для этого используются специальные опции командной строки. Вот так можно, например, выполнить наш скрипт-пример:

```
=> R --no-save < myscript.r > out
```

Опция `-no-save` говорит **R** не сохранять результаты сессии в файле истории `.RData/.Rhistory` (фактически, отвечает «no» на упомянутый выше заключительный вопрос).

1.6. Пакеты

Ещё одно важное преимущество **R** — наличие для него многочисленных расширений или пакетов буквально на все случаи жизни.

При установки **R** на компьютер, несколько пакетов уже в наличии: так называемые базовые пакеты, без которых система просто не работает (скажем, пакет,

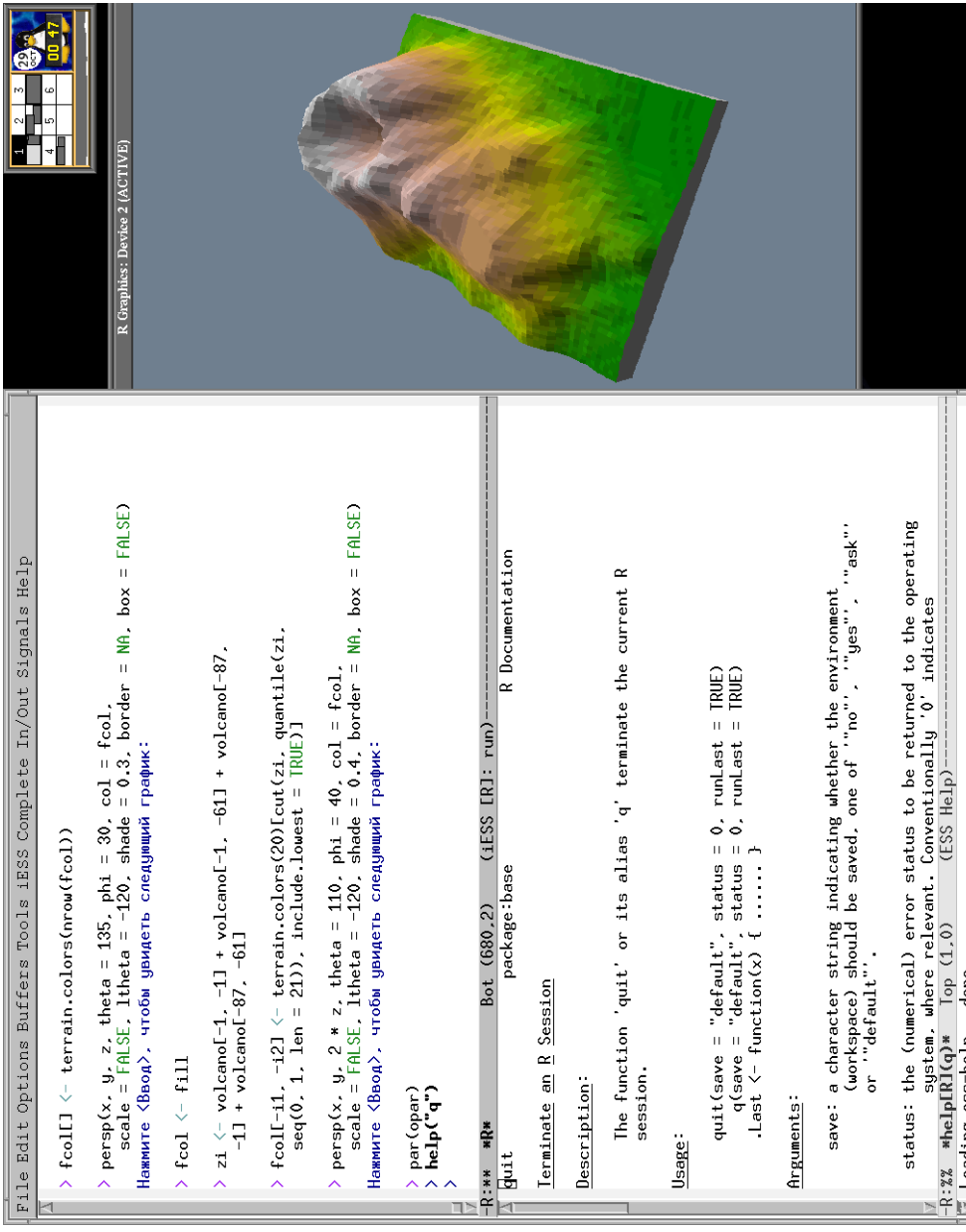

Рис. 1.1. emacs+ESS

который так и называется **base**, или пакет **grDevices**, который управляет выводом графиков), и ещё «рекомендованные» пакеты (пакет для специализированного кластерного анализа **cluster**, пакет для анализа нелинейных моделей **nlme** и пр.).

Кроме того можно поставить любой из почти полутора тысяч (!) доступных на CRAN пакетов. При доступном Интернете, это можно сделать прямо из **R** командой `install.packages()` (а под Mac и Windows есть соответствующие пункты в меню). Если соединение с сетью похуже, то можно скачать исходные тексты пакетов (под GNU/Linux) или скомпилированные пакеты (под Mac или Windows) и установить прямо с диска. После скачивания исходников пакета перед использованием сначала его нужно скомпилировать, потому что многие пакеты содержат код на FORTRAN или C. Для этого есть специальная форма вызова **R**:

```
=> R CMD INSTALL package.tar.gz
```

Естественно, это всё следует делать если **R** устанавливается не и из стандартного репозитария дистрибутива GNU/Linux. В случае стандартной установки можно поискать пакеты по сочетанию cran. В Debian GNU/Linux Etch таких пакетов ровно 85 — это не 1500 пакетов с CRAN, но скорее всего там уже есть многое из того, что необходимо.

Как только пакет установлен, то он сразу готов к работе. Нужно только инициализировать его перед употреблением. Для этого служит команда `library()`.

1.7. Полезные сайты, списки рассылки и документация

В заключение хотелось бы представить список самых полезных на наш взгляд сетевых ресурсов по **R**:

- `http://www.r-project.org/` — сайт проекта
- `http://cran.r-project.org/` — CRAN
- `https://stat.ethz.ch/pipermail/r-help/` — список рассылки R-help
- `http://finzi.psych.upenn.edu/nmz.html` — поиск в материалах по **R**
- `http://www.statmethods.net/index.html` — хороший справочный ресурс
- `http://zoonek2.free.fr/UNIX/48_R/all.html` — ещё один справочный ресурс
- `http://pj.freefaculty.org/R/Rtips.html` — советы по использованию **R**