|  |
| --- |
| Экономический Факультет МГУ им. Ломоносова |
| Основы статистического и регрессионного анализа в программной среде «R» |
|  |
|  |
| **Финансовая эконометрика** |
| **2013** |

|  |
| --- |
|  |

# Оглавление

[Раздел 1. Программная среда «R» 4](#_Toc346817263)

[1. Операции с числами и векторами 4](#_Toc346817264)

[2. Массивы и матрицы 10](#_Toc346817265)

[3. Циклы и условия 12](#_Toc346817266)

[4. Пользовательские функции 13](#_Toc346817267)

[4. Рисование графиков 15](#_Toc346817268)

[5. Установка пакетов 19](#_Toc346817269)

[6. Загрузка и сохранение данных 21](#_Toc346817270)

[Раздел 2. Основы статистического анализа 23](#_Toc346817271)

[1. Эмпирические оценки плотности и распределения 24](#_Toc346817272)

[2. Сравнение эмпирического распределения с затабулированным 28](#_Toc346817273)

[3. Тесты на нормальность 30](#_Toc346817274)

[4. Сравнение параметров двух нормальных выборок 32](#_Toc346817275)

[5. Сравнение двух произвольных выборок 34](#_Toc346817276)

[6. Домашнее задание 35](#_Toc346817277)

[Раздел 3. Основы регрессионного анализа 36](#_Toc346817278)

[1. Оценка параметров регрессионной модели 36](#_Toc346817279)

[2. Проверка нормальности остатков модели 39](#_Toc346817280)

[3. Тесты на гетероскедастичность 43](#_Toc346817281)

[4. Учёт автокорреляции 45](#_Toc346817282)

[5. Построение прогноза и доверительных интервалов 48](#_Toc346817283)

[6. Домашнее задание 49](#_Toc346817284)

[Раздел 4. Моделирование волатильности 51](#_Toc346817285)

[1. Моделирование средней доходности 51](#_Toc346817286)

[2. Тест на ARCH-эффекты 54](#_Toc346817287)

[3. Модели условной гетероскедастичности 55](#_Toc346817288)

[4. Анализ качества GARCH-модели 57](#_Toc346817289)

[5. Стационарность и единичные корни 59](#_Toc346817290)

[6. Тесты на единичные корни 60](#_Toc346817291)

[7. Построение оценок риска 62](#_Toc346817292)

[8. Проверка качества оценок риска 62](#_Toc346817293)

[9. Домашнее задание 65](#_Toc346817294)

[Раздел 5. Оценка рыночных рисков с помощью моделей семейства обобщённого гиперболического распределения. 67](#_Toc346817295)

[1. Оценка параметров модели 67](#_Toc346817296)

[2. Анализ качества модели 68](#_Toc346817297)

[3. Расчёт оценок риска 71](#_Toc346817298)

[4. Многомерный случай 72](#_Toc346817299)

[5. Оптимизация портфеля 73](#_Toc346817300)

[6. Домашнее задание 74](#_Toc346817301)

[Раздел 6. Выделение цикла, тренда и сезонности временного ряда. 75](#_Toc346817302)

[1. Выделение сезонной и остаточной компонент 75](#_Toc346817303)

[2. Разделение цикла и тренда 78](#_Toc346817304)

[3. Определение поворотных точек цикла 80](#_Toc346817305)

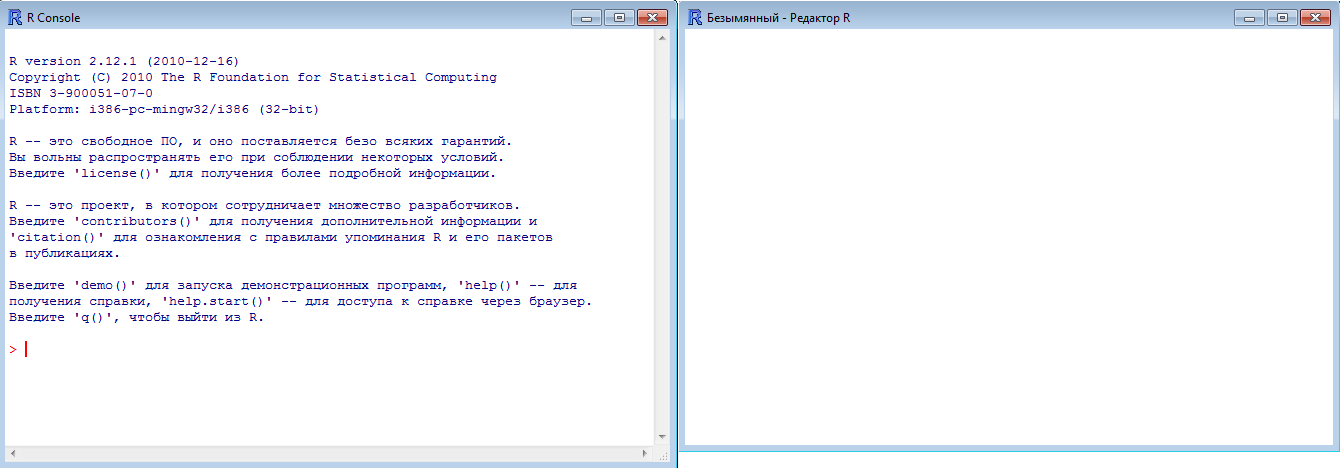
[4. Домашнее задание 81](#_Toc346817306)

# Раздел 1. Программная среда «R»

**R** — это программная среда, позволяющая проводить расчёты, статистический и эконометрический анализ данных, а также содержащая в себе набор методов для графического представления результатов.

Среда **R** доступна для свободной загрузки на сайте <http://cran.r-project.org>.

Команды в **R** вводятся двумя способами: через консоль, где каждая команда немедленно исполняется, или через окно скрипта, вызываемое выбором меню «Файл → Новый скрипт», в котором можно записывать серии команд и затем нажатием клавиш «Ctrl+R» исполнять выделенные участки программного кода.



*Рисунок 1. Консоль (слева) и окно скрипта (справа).*

## 1. Операции с числами и векторами

Базовым объектом R является вектор определённого типа, например, численный вектор, логический вектор, текстовый вектор и др. Чтобы присвоить какое-либо значение некоторой переменной **x**, используется оператор присвоения **<-**, состоящий из двух последовательных знаков: «меньше» и «минус». В результате выполнения команды

x <- 5

в переменной **x** будет содержаться число «5». Убедиться в этом можно, распечатав значение переменной на экране с помощью команды print:[[1]](#footnote-1)

print(x)

[1] 5

Можно также воспользоваться процедурой автоматической печати, просто набрав имя переменной в консоли. Результат будет идентичным.

x

[1] 5

Надпись «[1]», отображаемая перед выводимым на экран значением указывает на номер первого элемента вектора в текущей строке, что бывает полезным при печати векторов большой длины, полный список элементов которых не умещается в одну строку.

Хотя, по сути, объявленная нами переменная **x** имеет целочисленное значение, в памяти **R** оно хранится как число с плавающей запятой двойной точности. Если есть необходимость экономно расходовать системные ресурсы (например, при работе с большими массивами данных), то можно явно задать целочисленность присваемого значения, дописав после него заглавную букву «L».

x <- 5L

Проверить тип значения, содержащегося в переменной, можно с помощью функции typeof.

typeof(x)

[1] "integer"

Второе замечание состоит в том, что, хотя переменная **x** в нашем случае имеет скалярное значение, тем не менее, она интерпретируется как вектор единичной длины: к ней применимы все функции и операции, которые используются для векторов большей длины.

Создавать вектора большой длины можно различными способами.

Чтобы создать вектор из произвольных элементов, необходимо воспользоваться функцией c (возможно, от слова «consolidate»[[2]](#footnote-2)). Значения, которые нужно консолидировать в вектор, перечисляются через запятую в качестве аргументов функции c. Аргументы функции записываются в круглых скобках после её наименования.

x <- c(4,5,6)

x

[1] 4 5 6

Полезным инструментом для создания последовательностей чисел с единичным шагом является оператор двоеточия :. Для его применения перед двоеточием следует указать начальное значение последовательности, после него — конечное[[3]](#footnote-3).

x <- 6:10

[1] 6 7 8 9 10

Если конечное значение меньше, чем увеличенное на единицу последнее сгенерированное значение последовательности, то создание последовательности останавливается на этом значении, то есть диапазон чисел, покрываемый последовательностью, может быть меньше диапазона из указываемых начального и конечного значений.

x <- 1.3:4.2

[1] 1.3 2.3 3.3

Оператор двоеточия имеет более высокий приоритет, чем многие арифметические операции (сложение, вычитание, умножение, деление), поэтому в случае, если начальное или конечное значение задаётся в виде арифметического выражения, во избежание недоразумений рекомендуется использовать скобки. Сравните, например:

1:3\*2

[1] 2 4 6

1:(3\*2)

[1] 1 2 3 4 5 6

Ещё одна часто используемая для создания последовательностей функция — seq(from,to,by,length). У неё 4 основных параметра: начальное и конечное значение, **from** и **to**, параметр **by**, определяющий шаг между значениями последовательности, и параметр **length**, указывающий её длину. На практике для достижения нужного результата используют различные комбинации этих параметров. Например, создадим последовательность чисел от 0 с шагом 0.2 и длиной 5:

seq(0,by=0.2,length=5)

[1] 0 0.2 0.4 0.6 0.8

Последовательность от 6 до 11 с шагом 2:

seq(6,11,by=2)

[1] 6 8 10

Как и в случае с оператором двоеточия, диапазон значений последовательности может быть уже, чем диапазон из задаваемых начального и конечного значений. Наконец, последовательность от 0 до 8 длиной 5:

seq(0,8,length=5)

[1] 0 2 4 6 8

Для повторения одного и того же вектора значений несколько раз служит функция rep(x,each,times). Она имеет три основных параметра: вектор значений **x**, параметр **each**, указывающий количество повторений каждого из элементов **x**, и параметр **times**, определяющий количество повторений вектора **x** или вектора, полученного после применения параметра **each**, если он задан.

rep(1:2,times=2)

[1] 1 2 1 2

rep(1:2,each=2)

[1] 1 1 2 2

rep(1:2,each=2,times=2)

[1] 1 1 2 2 1 1 2 2

Сортировка вектора значений осуществляется с помощью функции sort(x,decreasing), в которой, кроме вектора **x**, следует указать логическое значение (TRUE или FALSE) параметра decreasing. Если decreasing равен TRUE, то сортировка производится по убыванию, иначе — по возрастанию.

x <- sort(c(10,-3,6,0,2),decreasing=TRUE)

[1] 10 6 2 0 -3

x <- sort(c(10,-3,6,0,2),decreasing=FALSE)

[1] -3 0 2 6 10

Следует обратить особое внимание на то, что язык **R** является чувствительным к регистру. Таким образом, нельзя вместо логического значения TRUE написать «True» или «true». Столь же тщательно следует различать строчные и заглавные буквы в названиях переменных.

Важным свойством векторов в R является то, что они могут содержать значения только одного типа. Допустимыми являются численные, логические вектора и другие, однако не может существовать вектор, состоящий, например, из числа, текстового значения и логического значения. Если пользователь попытается объединить разнородные значения в вектор, то все эти значения будут преобразованы в один из типов данных, по определённому порядку. Для наиболее часто используемых типов порядок следующий: «character», «numeric», «logical». Это означает, что если в разнородном наборе есть хотя бы один текстовый элемент, то все элементы этого набора будут преобразованы в текст.

c("a",5,TRUE)

[1] "a" "5" "TRUE"

Если же переменной присваивается набор из численных и логических значений, то на выходе получится численный вектор, в котором логическое значение TRUE будет заменено на 1, а FALSE — на 0.

c(5,TRUE)

[1] 5 1

Пользователь может по своему желанию изменять тип данных с помощью семейства функций as.<data.type>(x). Вместо конструкции <data.type> следует подставить название типа. Например, следующая запись преобразует текстовый вектор в числовой.

as.numeric(c("1","2"))

[1] 1 2

Если какой-либо элемент исходного вектора не удаётся преобразовать в желаемый тип, на его месте появляется пропущенное значение NA (от словосочетания «Not Available»). Проверить, принадлежит ли вектор к заданному типу данных, аналогичным образом можно, используя семейство функций is.<data.type>(x).

Все операции с векторами в **R** выполняются покомпонентно. Например, если два вектора складываются, то первый элемент результирующего вектора будет равен сумме первых элементов исходных векторов, второй — сумме вторых и так далее. Если вектора имеют разную длину, то при выполнении парных операций с ними меньший вектор будет повторен несколько раз, чтобы его длина сравнялась с длиной большего. При этом если длины векторов не кратны, то в последний раз меньший вектор будет повторен лишь частично, т.е. будет повторено лишь некоторое количество его первых элементов.

Проиллюстрируем это на примерах. Возьмём переменную **x**, в последний раз определённую функцией sort выше.

x

[1] -3 0 2 6 10

Прибавим к этой переменной скалярную величину (вектор единичной длины).

x+1

[1] -2 1 3 7 11

Мы видим, что к каждому элементу вектора **x** была прибавлена эта скалярная величина. Говоря в терминах векторов, вектор меньшей длины, единичный вектор, был повторен 5 раз, чтобы его длина сравнялась с длиной **x**. Затем операция сложения была выполнена покомпонентно.

Рассмотрим другой пример.

x+c(1,2,3,4,5)

[1] -2 2 5 10 15

В этом примере длины двух векторов одинаковы, и элементы результирующего вектора равны сумме соответствующих элементов **x** и второго вектора.

Наконец, пример с векторами некратной длины.

x+c(1,2,3)

[1] -2 2 5 7 12

Меньший вектор в этом примере был повторен лишь частично, точнее, были повторены только первые два его элемента, и четвёртый элемент результирующего вектора равен сумме четвёртого элемента **x** и первого элемента второго вектора. Вдобавок, **R** в таких случаях выдаёт пользователю предупреждение о длине векторов.

Аналогичным образом осуществляются и другие арифметические операции: вычитание, умножение, деление и возведение в степень.

Кроме парных арифметических операций, в R имеется набор алгебраических функций, таких как log(x), exp(x), sin(x), cos(x), tan(x), min(x), max(x), abs(x) и другие.

Для анализа данных могут быть полезны статистические функции: range(x) — возвращает вектор длины 2 из минимального и максимального элементов **x**, length(x) — длина вектора, sum(x) и prod(x) — сумма и произведение элементов, mean(x) и var(x) — среднее и выборочная дисперсия.

Важную роль при составлении алгоритмов играют логические вектора, служащие для проверки условий и в качестве индикаторных функций. Хотя создать логический вектор можно теми же способами, что и числовой, чаще всего они возникают при выполнении логических операций.

Рассмотрим, например, операции сравнения. Сравнение в **R** осуществляется в основном с помощью стандартных операторов: «больше», «меньше», «больше или равно», «меньше или равно». Внимание следует обратить на знаки логического равенства ==, который состоит из двух последовательных знаков «равно», и неравенства !=, состоящего из восклицательного знака и следующего за ним «равно».

В примере ниже показано создание логического вектора при выполнении сравнения.

y <- x>3

[1] FALSE FALSE FALSE TRUE TRUE

Операция сравнения, как и другие операции с векторами, выполняется покомпонентно. Вектор **x** состоит из элементов {-3; 0; 2; 6; 10}. Первые три элемента **x** меньше 3, поэтому первые три элемента **y** равны FALSE. Последние два элемента **x** действительно больше 3, поэтому последние два элемента **y** равны TRUE.

Для проверки условий часто используются логические операторы «и» (&), «или» (|) и «не» (!), как это показано в следующем примере.

y <- x>=3 & x<=6

[1] FALSE FALSE FALSE TRUE FALSE

y <- x<3 | x>6

[1] TRUE TRUE TRUE FALSE TRUE

y <- !(x>3)

[1] TRUE TRUE TRUE FALSE FALSE

При выполнении арифметических операций с логическими векторами, согласно порядку типов, значение TRUE заменяется на 1, а FALSE — на 0. Это делает очень удобным использование логических векторов в качестве индикаторных функций. Например, выражение можно изящно записать в R как

sum(x>3)

[1] 2

Если пользователь захочет рассмотреть не весь вектор, а какую-либо его часть, он может воспользоваться несколькими правилами обращения к элементам вектора.

Во-первых, это можно сделать с помощью логического вектора, указав его в квадратных скобках после названия исходного вектора. Квадратные скобки служат для обозначения элементов вектора. При этом результирующий вектор будет состоять из тех элементов исходного вектора **x**, которым соответствуют значения TRUE логического вектора **y**.

x[y]

[1] -3 0 2

Если вектора **x** и **y** имеют разную длину, то работают те же правила удлинения меньшего вектора, которые были рассмотрены ранее, за исключением того, что исходный вектор **x** в этом случае не может выступать в роли меньшего.

Второй способ обратиться к элементам **x** — с помощью набора натуральных чисел. В качестве такого набора может использоваться последовательность или произвольный вектор натуральных чисел. Каждое число интерпретируется как номер элемента **x**.

x[1:3]

[1] -3 0 2

x[c(3,2,5)]

[1] 2 0 10

Элементы результирующего вектора будут расположены в том порядке, в котором следуют числа в квадратных скобках.

Иногда бывает удобно выбирать элементы вектора методом исключения. Если пользователь хочет получить результирующий вектор, состоящий из всех элементов **x**, кроме перечисленных, он может отбросить ненужные элементы, вместо натуральных чисел указав соответствующие им отрицательные значения.

x[-(1:3)]

[1] 6 10

Ограничение этого метода заключается в том, что в списке элементов исходного вектора не могут одновременно присутствовать и положительные, и отрицательные числа.

Наконец, если элементы **x** имеют наименования, то вместо номера элемента можно указать его имя. В примере ниже сначала с помощью функции names(x) элементам вектора **x** присваиваются имена, а затем из этого вектора выбираются элементы с указанными названиями.

names(x) <- c("mon","tue","wed","thu","fri")

x[c("mon","thu")]

[1] -3 6

## 2. Массивы и матрицы

Матрицы и массивы в **R** в основном создаются путём присвоения размерности какому-либо вектору. Это можно проделать несколькими способами.

Во-первых, можно задать размерность с помощью функции dim(x).

z <- 1:1500

dim(z) <- c(3,5,100)

В результате выполнения этих команд вектор **z** длиной 1500 был преобразован в трёхмерный массив . Размерность конечного массива задаётся с вектором после оператора присвоения. При этом необходимо следить, чтобы произведение элементов этого вектора (произведение размерностей конечного массива) было в точности равно длине исходного вектора. Заполнение массива начинается с первой размерности, т.е. **z**[1] → **z**[1;1;1], **z**[2] → **z**[2;1;1]. После того, как исчерпывается вместимость первой размерности, значение следующей увеличивается на единицу, и снова начинается заполнение первой размерности: **z**[3] → **z**[3;1;1], **z**[4] → **z**[1;2;1], **z**[5] → **z**[2;2;1]. Те же рассуждения применимы и для всех последующих размерностей.

Получить массив или матрицу из вектора, не изменяя сам вектор, позволяют функции array(x,dim) и matrix(x,nrow,ncol). В функции array, кроме исходного вектора **x**, следует задать вектор размерностей массива **dim**, а в функции matrix — параметры **nrow** и **ncol**, отвечающие за количество строк и столбцов матрицы. В следующем примере из вектора длиной 20 создаётся матрица .

z <- matrix(1:20,nrow=5,ncol=4)

z <- array(1:20,dim=c(5,4))

[,1] [,2] [,3] [,4]

[1,] 1 6 11 16

[2,] 2 7 12 17

[3,] 3 8 13 18

[4,] 4 9 14 19

[5,] 5 10 15 20

Заполнение массива элементами вектора в этом случае осуществляется аналогично функции dim.

Обращение к элементам массива происходит так же, как и к элементам вектора, однако особенность массива состоит в том, что пользователь может не указывать список элементов в некоторых размерностях. Это интерпретируется как желание выбрать все элементы в текущей размерности. Например, чтобы выбрать все элементы матрицы, принадлежащие к её первому столбцу, следует, не указывая номера строк, во второй размерности записать первый столбец.

z[,1]

[1] 1 2 3 4 5

Аналогично, не указывая столбцов, можно выбрать все элементы из первой строки **z**.

z[1,]

[1] 1 6 11 16

Если в квадратных скобках указываются списки элементов для нескольких размерностей, то конечный массив будет состоять из элементов, находящихся на пересечении заданных размерностей.

z[1:2,1:2]

[,1] [,2]

[1,] 1 6

[2,] 2 7

Рассмотрим ряд действий, которые можно осуществлять с матрицами.

Транспонирование с помощью функции t(x).

tz <- t(z)

Нахождение обратной матрицы.

inv.z <- solve(z)

Умножение в том смысле, как это понимается в линейной алгебре.

z.tz <- z %\*% tz

Создание диагональной матрицы, на главной диагонали которой находится заданный вектор.

y <- diag(1:5)

Нахождение собственных значений и векторов.

e <- eigen(y)

Переменная **e**, которой был присвоен результат выполнения функции eigen, имеет тип «list» (список). В отличие от векторов, каждый элемент списка может иметь произвольный тип. Функция eigen создаёт список из двух элементов. Первый элемент, называемый «values», содержит вектор из собственных значений матрицы **y**, второй, называемый «vectors», — матрицу из собственных векторов **y**, записанных по столбцам. Обратиться к элементу списка можно, указав его номер в двойных квадратных скобках или его имя после знака «$». Таким образом, для получения, например, вектора из собственных значений **y** есть выбор из двух эквивалентных команд.

e[[1]]

e$values

[1] 5 4 3 2 1

Наконец, с помощью собственных значений и векторов можно найти квадратный корень из матрицы.

y.sqrt <- e$vectors %\*% diag(sqrt(e$values)) %\*% t(e$vectors)

## 3. Циклы и условия

В R выделяют два типа циклов: цикл с предусловием и цикл «для».

Цикл с предусловием начинается с проверки некоторого условия. Если условие выполнено, то исполняются команды, составляющие тело цикла, и затем цикл переходит на следующую итерацию.

y <- numeric(); i <- 1

while (i <= 5) {

y[i] <- i^2

i <- i + 1

}

[1] 1 4 9 16 25

Разберём пример построчно.

В первой строке с помощью функции numeric(x) объявляется числовой вектор **y** нулевой длины. Функция numeric имеет один аргумент, отвечающий за длину создаваемого вектора. Далее следует команда присвоения: переменной **i** присваивается начальное значение 1. Команды в **R** разделяются либо новой строкой, либо символом «;».

Во второй строке с оператора while начинается цикл. После оператора в круглых скобках записывается логический вектор единичной длины, который обычно образуется в результате проверки условия. В нашем случае проверяется, не превосходит ли **i** 5. Фигурные скобки служат для объединения команд в единое целое, например, в тело цикла. Для удобства чтения и редактирования программного кода рекомендуется открывать фигурную скобку в первой строке цикла и закрывать в конце отдельной строкой. Тело же рекомендуется отделять от начала строки цикла (или позиции, занимаемой началом первой строки) несколькими пробелами.

В третьей строке i-му элементу вектора **y** присваивается значение i2. Несмотря на то, что вначале вектор **y** имел нулевую длину, R позволяет присваивать значения ранее не обозначенным элементам вектора. При этом элементы, находящиеся между ранее не обозначенным определяемым элементом и последним из определённых элементов, получают значение по умолчанию. Для числовых векторов оно равно нулю. Таким образом, на первой итерации цикла первый элемент вектора **y** получит значение , на второй итерации второй элемент получит значение и так далее.

Четвёртая строка служит для того, чтобы изменить значение служебной переменной **i**, называемой счётчиком цикла, при переходе к следующей его итерации.

В пятой строке закрывающаяся фигурная скобка обозначает конец цикла.

Рассмотрим тот же самый алгоритм, реализованный с помощью цикла «для».

y <- numeric()

for (i in 1:5) {

y[i] <- i^2

}

[1] 1 4 9 16 25

После оператора for, с которого начинается цикл, в круглых скобках указывается набор значений, пробегаемых счётчиком. Далее следует тело цикла, в котором нет нужды явно задавать изменение служебной переменной, так как она автоматически примет следующее значение из указанного набора при переходе к новой итерации цикла.

Условный оператор if позволяет совершать набор действий при выполнении определённого условия.

if (y[1] == 1) {

y <- y + 5

}

[1] 5 9 14 21 30

Для этого после оператора в круглых скобках указывается логический вектор единичной длины, а затем в фигурных скобках следует набор команд, которые выполняются, если логический вектор примет значение TRUE.

## 4. Пользовательские функции

Часто возникает необходимость рассчитывать значения каких-либо величин по единому алгоритму, но с разными входными параметрами. Вместо того, чтобы каждый раз при решении таких задач заново вводить один и тот же программный код, целесообразно применять пользовательские функции.

Пользователям дана возможность самим создавать необходимые им функции. В качестве примеров мы будем рассматривать самые простые из них, выполнение которых сводится к расчёту алгебраического выражения, зависящего от набора параметров, однако в общем случае в теле функции может содержаться сколь угодно сложный алгоритм расчётов, включающий в себя условия, циклы и различные манипуляции с массивами данных.

Рассмотрим функцию . Хотя в алгебраическом смысле аргументами этой функции являются *x1*и *x2*, а *c* и *α* — параметры, при более широком взгляде на этот вопрос становится понятно, что все четыре величины имеют право называться аргументами или параметрами, и их разделение на две категории носит чисто субъективный характер.

Таким образом, определим функцию *f*, зависящую от четырёх параметров: *x1*, *x2*, *c* и *α*.

f <- function(x1,x2,c,alpha) {

c - x1^alpha - x2^alpha

}

После оператора function в круглых скобках перечисляются параметры функции, а далее, по аналогии с циклами и условиями, в фигурных скобках следует собственно тело функции. Значением, возвращаемым функцией, является последнее выражение, вычисляемое в её теле. В нашем случае тело функции состоит из одного выражения, которое и является последним.

Как уже отмечалось выше, в теле функции могут присутствовать различные операторы. Внутри функции также можно использовать переменные, ранее определённые в рабочем пространстве, но допускается и определение новых переменных. Переменная, определённая в теле функции, является доступной только внутри неё, и её значение не сохраняется после завершения работы функции. Имена таких переменных могут совпадать с именами ранее определённых в рабочем пространстве. В этом случае внутри функции эта переменная примет новое значение, а при выходе из неё возвратит себе старое. Если возникает необходимость сохранить изменённое значение, можно использовать оператор абсолютного присваивания <<-. При проведении расчётов внутри функции необязательно использовать все её параметры, но если в этих расчётах будут фигурировать переменные, не определённые ни внутри функции, ни вне её, возникнет ошибка.

В нашем примере объявленная нами функция сохраняется в переменной **f**. Для того чтобы вычислить значение этой функции при определённой комбинации параметров, после её имени в круглых скобках следует указать список их значений, расположенных в том же порядке, что и при объявлении функции. Например, значению соответствует следующее выражение.

f(0.5,0.5,1,2)

[1] 0.5

В этом примере параметрам **x1** и **x2** соответствуют значения 0.5, параметру **c** — 1, а параметру **alpha** — 2, потому что именно в таком порядки параметры были перечислены после оператора function.

Иногда имеет смысл для некоторых параметров определять значения по умолчанию, чтобы не указывать их каждый раз при использовании функции. Для этого при объявлении функции им нужно присвоить эти значения.

f <- function(x1,x2,c=1,alpha=2) {

c - x1^alpha - x2^alpha

}

Теперь становится возможным вызов функции без указания параметров имеющих значение по умолчанию.

f(0.5,0.5)

[1] 0.5

Конечно, ничто не мешает при вызове функции определить для этих параметров какое-либо другое значение.

Обратиться к функции также можно, в явном виде задав её параметры. В этом случае соблюдать порядок их следования необязательно.

f(alpha=2,x2=0.5,x1=0.5,c=1)

[1] 0.5

Если название параметра состоит из большого количества символов, при определении можно указывать лишь первые несколько букв его имени, следя за тем, чтобы введённый набор символов однозначно определял этот параметр. В рассмотренном примере для задания параметра **alpha** достаточно использовать символ «a».

f(a=2,x2=0.5,x1=0.5,c=1)

[1] 0.5

## 4. Рисование графиков

Для построения графика в **R** необходимо определить три основные составляющие: значения аргументов, соответствующие им значения функции и набор параметров, отвечающих за внешний вид графика.

Двумерные графики на плоскости строятся с помощью функции plot. Прежде чем её применять, зададим значения аргумента **x** и определим функциональную зависимость, описав некоторую функцию **g**.

x <- seq(-5,5,length=101)

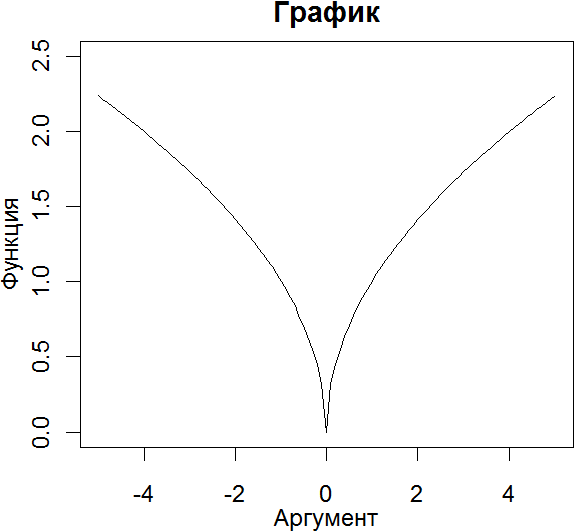
g <- function(x,c=1) c\*abs(x)^0.5

Здесь мы использовали краткую запись функции, состоящей из одного выражения. Подобное представление также допустимо для условий и циклов. После этого, определив сетку значений аргумента и вычислив соответствующие им значения функции, мы можем построить график.

plot(x,g(x),type="l",lty="solid",xlim=c(-5,5),ylim=c(0,2.5),

main="График",xlab="Аргумент",ylab="Функция")

Заметим, что R позволяет переносить длинную запись команды на следующую строку. При этом, однако, не допускается разделы внутри имён переменных, параметров или присваиваемых им значений.



*Рисунок 2. Результат выполнения функции plot.*

Функция plot имеет ряд параметров.

Во-первых, это значения по осям абсцисс и ординат. Хотя в примере задана чёткая функциональная связь между ними, вообще говоря, это не обязательно. Формально для построения графика требуются лишь ряд значений *x* и ряд соответствующих им значений *y*. Если значения аргумента не указываются, вместо них используются значения по умолчанию: ряд натуральных чисел длиной, равной длине ряда значений ординат.

Текстовый параметр **type** определяет вид графика: «p» соответствует точечному графику, «l» — линейному. По умолчанию этот параметр имеет значение «p».

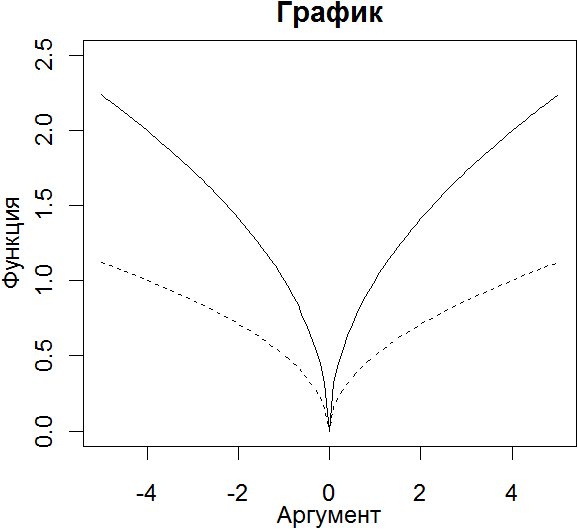
Текстовый параметр **lty** служит для изменения вида линии: «solid» обозначает сплошную линию, «dashed» — пунктирную, «dotted» — состоящую из точек. Значением по умолчанию является «solid».

Параметры **xlim** и **ylim** отвечают за отображаемый диапазон значений по осям абсцисс и ординат. Они задаются в виде числовых векторов из двух элементов, соответствующих минимальным и максимальным значениям. По умолчанию значение этих параметров подбирается так, чтобы на экране отобразился весь график.

Наконец, с помощью текстовых параметров **main**, **xlab** и **ylab** можно наносить подписи к осям и ко всему графику. Значениями по умолчанию являются названия переменных для осей и название ряда значений ординат для подписи графика.

Добавить линии на уже существующий график позволяет функция lines.

lines(x,g(x,c=0.5),lty="dashed")



*Рисунок 3. Добавление линий на существующий график.*

Она имеет те же параметры, что и функция plot, за рядом исключений: нет необходимости задавать вид графика, диапазон отображаемых значений и подписи.

Аналогом функции lines для добавления точек на уже существующий график является функция points.

Идея построения трёхмерных пространственных графиков та же, что и в двумерном варианте: задаются значения по осям *x*, *y* и *z* и определяется набор дополнительных параметров.

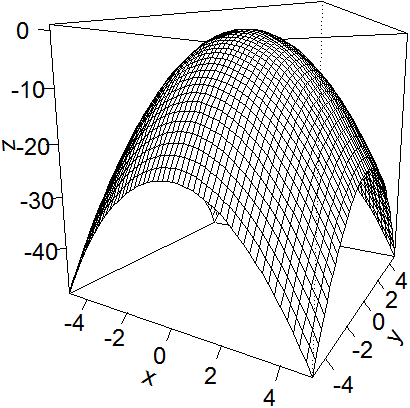
Для примера рассмотрим определённую ранее функцию .

x <- y <- seq(-5,5,length=51)

z <- outer(x,y,f)

Расчёт значений **z** в трёхмерном случае производится с помощью функции outer, в которой, кроме двух векторов значений аргументов, необходимо задать некоторую функциональную зависимость, в нашем случае представленную функцией *f*. Итоговая матрица **z** будет иметь следующую структуру: . Определив все необходимые значения, мы можем построить график, используя функцию persp.

persp(x,y,z,theta=30,phi=10,col="white",ticktype="detailed")



*Рисунок 4. Трёхмерный пространственный график.*

Эта функция имеет несколько основных параметров.

**x**, **y** и **z** — наборы значений по соответствующим осям, которые должны иметь ту структуру, которая показана в примере. Как и в случае с функцией plot, чёткая функциональная связь между этими параметрами необязательна.

Параметры **theta** и **phi** определяют углы обзора по горизонтали и вертикали, измеряемые в градусах.

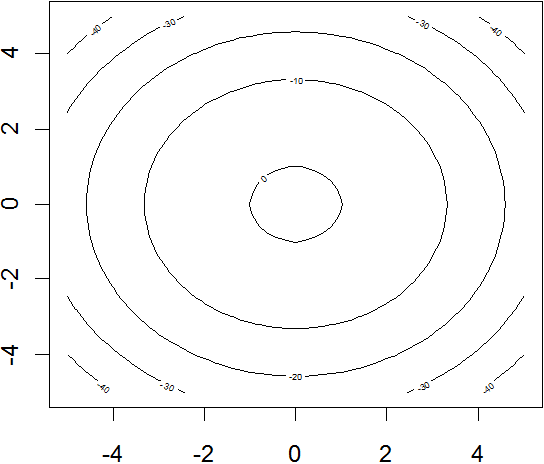
**col** — параметр, отвечающий за цвет графика. Чтобы просмотреть список всех возможных его значений, следует воспользоваться функцией colors(), не имеющей аргументов.

Способ маркировки осей определяется параметром **ticktype**. Если указано значение «detailed», то на оси наносится разметка. При значении «simple» отображаются только стрелки, указывающие в направлении возрастания значений по осям.

Наконец, параметры **xlim**, **ylim**, **zlim**, **xlab**, **ylab**, **zlab** и **main** служат для тех же целей, что и в функции plot.

Функция contour рисует линии уровня трёхмерного пространственного графика.

contour(x,y,z,nlevels=5)



*Рисунок 5. Линии уровня трёхмерного пространственного графика.*

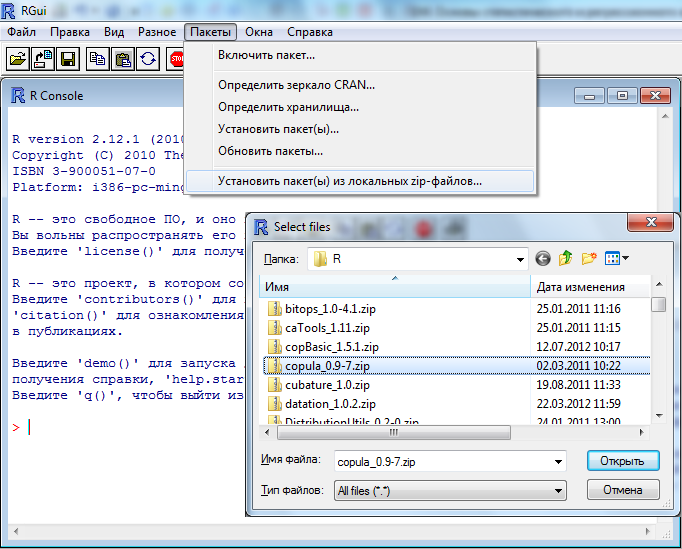
Наборы значений по осям задаются в ней точно так же, как и в функции persp, а изменяя параметр **nlevels**, можно регулировать количество отображаемых линий.

## 5. Установка пакетов

Большинство процедур и функций не содержатся изначально в **R**, а загружаются в виде пакетов (библиотек). Пакет — это совокупность объектов, методов и функций, служащих для решения определённого класса задач.

Рассмотрим несколько способов установки пакетов.

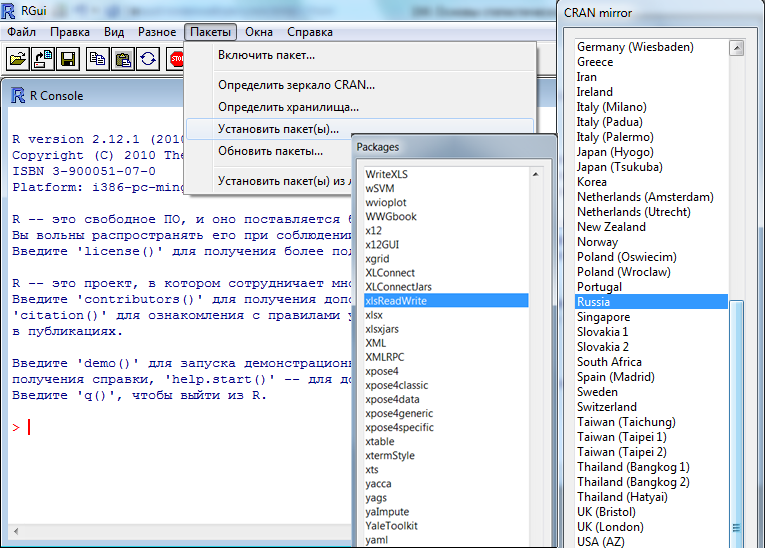
Пакеты **R** обычно хранятся в сжатых папках формата «zip». Если у пользователя на локальном диске его компьютера есть zip-файл, содержащий пакет, то он с помощью меню «Пакеты → Установить пакет(ы) из локальных zip-файлов…» может загрузить пакет в **R**.



*Рисунок 6. Установка пакетов из локальных zip-файлов.*

Содержащиеся в этом пакете объекты и функции становятся доступны после объявления пакета командой library(<package.name>), где вместо конструкции в круглых скобках следует подставить название пакета.

Второй способ установки пакетов — загрузка с сайта **R**. Она осуществляется путём выбора меню «Пакеты → Установить пакет(ы)…». В ходе установки пользователю будет предложен выбор зеркала загрузки и список доступных пакетов, в котором он с помощью клавиш «Shift» и «Ctrl» может выбрать необходимые библиотеки.



*Рисунок 7. Установка пакетов с сайта R.*

Преимущество этого способа состоит в том, что он позволяет установить не только те пакеты, которые, собственно, необходимы пользователю, но и те, на которые они ссылаются. Например, для корректной работы пакета «fGarch», кроме него самого, необходимо также установить пакеты «timeDate», «timeSeries», «fBasics» и «MASS». В описываемом способе все эти пакеты устанавливаются автоматически.

Третий и наиболее эффективный способ состоит в использовании функции install.packages. В качестве аргументов на вход этой функции текстовый вектор наименований пакетов, которые затем по той же схеме загружаются с сайта **R**.

После загрузки пакетов с сайта **R**, как и в случае установки с локального диска, их необходимо объявить командой library.

## 6. Загрузка и сохранение данных

Для взаимодействия с внешним миром в **R** предусмотрены функции, работающие с файлами различных типов. Мы рассмотрим три типа файлов: текстовый документ (.txt), значения, разделённые запятой, (.csv) и excel-файлы (.xls).

Загрузка данных из текстового файла осуществляется с помощью функции read.table.

dat <- read.table("C:/Folder/input.txt",header=TRUE,sep=",",dec=".")

У неё четыре основных аргумента.

Во-первых, это полное имя файла. Полное имя состоит из пути, начинающегося с названия диска и включающего в себя все подкаталоги, собственно названия файла и расширения (у текстовых файлов — «.txt»). Полное имя файла можно не указывать, ограничившись названием и расширением, если файл находится в рабочей директории R. Функция getwd() без аргументов отображает текущую рабочую директорию, а функция setwd(path) позволяет определить новую, указав путь к ней в параметре **path**.

Логический параметр **header** устанавливается равным TRUE, если в загружаемом файле есть заголовки данных, и FALSE — в противном случае.

Наконец, параметры **sep** и **dec** определяют разделитель столбцов и разделитель целой и дробной частей числа.

Аналогичным образом работает функция read.csv.

dat <- read.csv("C:/Folder/input.csv")

Поскольку формат таких файлов предполагает наличие заголовков, запятую и точку в качестве разделителей столбцов и частей чисел, нет необходимости указывать соответствующие параметры.

Функции для работы с файлами «Excel» содержатся в пакете **xlsx**. Для корректной работы этого пакета, возможно, потребуется [установить ПО «Ява»](http://java.com/en/download/)[[4]](#footnote-4).

library(xlsx)

Для чтения данных используется функция read.xlsx.

dat <- read.xlsx("C:/Folder/input.xlsx",sheetIndex=1,header=TRUE)

В этой функции, кроме полного имени файла, необходимо также задать параметр **sheetIndex**, определяющий номер листа, с которого будет производиться загрузка данных, и параметр **header**, позволяющий указать, есть ли на листе заголовки столбцов.

Переменная, в которую загружаются данные из файла, в каждом случае обозначенная нами «dat», будет иметь тип «data.frame» (таблица данных). По сути, она является матрицей, каждый столбец которой имеет некоторый тип. Прямое использование этой переменной в расчётах может повлечь проблемы, связанные с несоответствием форматов данных, поэтому каждый её столбец, необходимый для работы, рекомендуется сохранять в отдельной переменной.

Сохранить данные в формате «Excel» позволяет функция write.xlsx.

write.xlsx(output.data,"C:/Folder/output.xlsx",sheetName="Таблица1",

col.names=TRUE,row.names=FALSE,append=FALSE)

Кроме массива данных, подлежащего записи в файл, в ней необходимо указать ряд параметров. **sheetName** присваивает имя листу выгружаемых данных, **col.names** и **row.names** отвечают за наличие заголовков столбцов и строк, а **append** позволяет дописывать данные в уже существующий файл вместо создания нового. Заметим, что выходной файл, в отличие от случая загрузки данных, может и не существовать. В таком случае он будет создан автоматически.

# Раздел 2. Основы статистического анализа

В набор стандартных процедур **R** входят функции для работы со многими статистическими распределениями. Список наиболее часто используемых распределений представлен в табл. 1.

*Таблица 1. Затабулированные статистические распределения в* ***R****.*

|  |  |  |
| --- | --- | --- |
| **Название распределения** | **Обозначение в R** | **Параметры** |
| Нормальное | norm | mean, sd |
| t-распределение Стьюдента | t | df |
| Равномерное | unif | min, max |
| Хи-квадрат | chisq | df |
| F-распределение Фишера | f | df1, df2 |
| Гамма | gamma | shape, scale |
| Логнормальное | lnorm | meanlog, sdlog |
| Пуассона | pois | lambda |
| Экспоненциальное | exp | rate |
| Биномиальное | binom | size, prob |

Каждому распределению из табл. 1 соответствуют четыре функции: генератор случайных чисел, квантильная функция, функции распределения и плотности. Первый вид функций записывается с помощью прибавления строчной буквы «r» (от слова «random») к названию распределения. Например, rnorm — генератор случайных нормальных величин. Остальные функции записываются аналогично, путём прибавления букв «q» (от слова «quantile»), «p» (от слова «probability») и «d» (от слова «density») соответственно.

Рассмотрим пример для нормального распределения.

N <- 100; x <- seq(-5,5,by=0.1); alpha <- 0.95

rnorm(n=N,mean=0,sd=1) # генератор случайных чисел

qnorm(alpha,mean=0,sd=1) # квантиль

pnorm(x,mean=0,sd=1) # функция распределения

dnorm(x,mean=0,sd=1) # функция плотности

Текст после решётки — комментарии, которые пользователь может оставлять в программном коде.

Каждая из четырёх функций зависит от параметров конкретного распределения, представленных в третьем столбце табл. 1. Помимо этого, в генераторе случайных чисел необходимо задать количество реализаций (**n**), в квантильной функции — значение вероятности попадания в левый хвост, в функциях распределения и плотности — точки, в которых рассчитываются их значения.

## 1. Эмпирические оценки плотности и распределения

Наиболее простой и широко известной эмпирической оценкой плотности распределения является гистограмма. Идея её построения состоит в разбиении отрезка, с близкой к единице вероятностью накрывающего все возможные значения рассматриваемой величины, на несколько интервалов и подсчёте количества выборочный значений, попадающих в каждый интервал. При этом если дважды пронормировать количество попаданий в какой-либо интервал (на общее количество наблюдений и на его длину), то получится оценка плотности распределения на этом интервале. Варьируя длины интервалов, можно добиться желаемого уровня детализации.

Рассмотрим пример. В **R**, помимо процедур и функций, имеются учебные наборы данных, которые мы будем использовать для иллюстрации работы тех или иных статистических методов. В частности, массив **faithful** представляет собой «таблицу данных», первый столбец («eruptions») которой соответствует длительностям извержений гейзера «Старый служака» («Old Faithful») в секундах, а второй («waiting») — промежуткам между извержениями.

С помощью функции head мы можем вывести на экран первые несколько строк этого массива.

head(faithful)

eruptions waiting

1 3.600 79

2 1.800 54

3 3.333 74

4 2.283 62

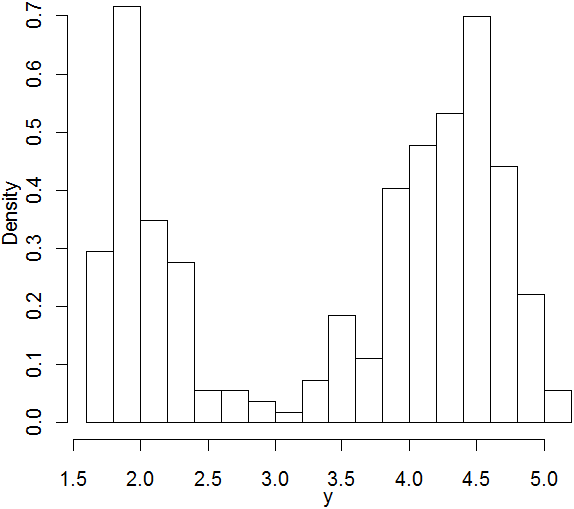
5 4.533 85

6 2.883 55

Построим гистограмму длительностей извержений.

y <- faithful$eruptions

hist(y,breaks=seq(1.6,5.2,by=0.2),prob=TRUE)



*Рисунок 8. Гистограмма распределения длительностей извержений.*

Параметр **breaks** функции hist задаёт совокупность интервалов, на которых рассчитываются оценки. Необходимо следить, чтобы эти интервалы накрывали все выборочные значения. Параметр **breaks** можно и не указывать, задав с помощью **nclass** количество интервалов. Следующая команда приведёт к идентичному результату.

hist(y,nclass=18,prob=TRUE)

Параметр **probability** определяет, будут ли рассчитываться оценки плотности распределения на интервалах или же количество попаданий в них.

Гистограмма является довольно грубой оценкой плотности распределения. Во-первых, она предполагает одинаковые значения плотности на всём интервале, а во-вторых, сама конфигурация гистограммы зависит от определения отрезка, накрывающего все выборочные значения. В этом можно убедиться, построив гистограммы при различных значениях параметра **breaks**.

Более точную эмпирическую оценку плотности распределения предоставляют ядерные оценки. Теоретические основы построения этих оценок находятся за рамками нашего курса, однако мы рассмотрим их практическую реализацию в **R**.

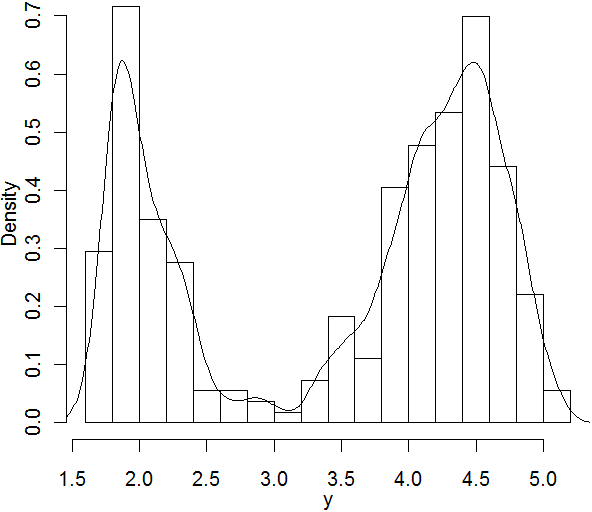
За построение ядерных оценок отвечает функция density.

y.pdf <- density(y,n=1024,bw="ucv")

Переменная, обозначенная **y.pdf**, в которой сохраняется результат выполнения функции density, имеет тип «список». Компоненты «списка», как обсуждалось ранее, могут содержать в себе данные произвольных типов. Нас будут интересовать первые две компоненты, обозначаемые «x» и «y», представляющие собой числовые векторы координат, в которых были произведены оценки плотности, и самих этих оценок.

В функции density, кроме выборочного вектора, следует также в параметре **n** указать с точностью до степени двойки число точек, в которых будет оцениваться плотность, а в параметре **bw** (от слова «bandwidth») — метод оценки. С помощью оператора lines мы можем построить ядерную оценку плотности поверх гистограммы.

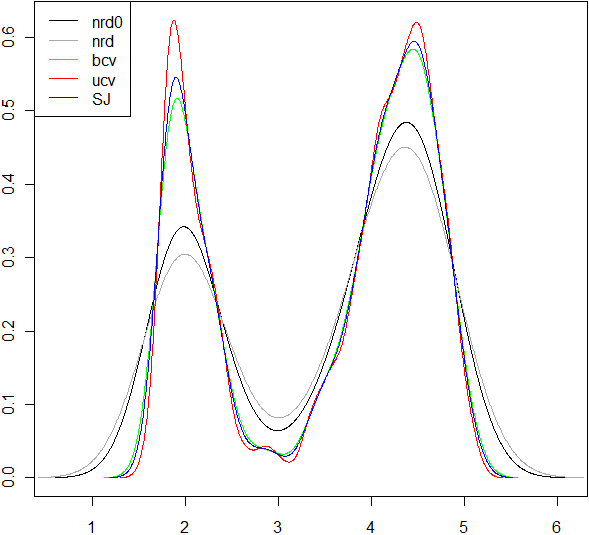
lines(y.pdf$x,y.pdf$y)



*Рисунок 9. Гистограмма и ядерная оценка.*

Решающее значение для построения ядерных оценок имеет выбор сглаживающего параметра. Этот выбор может осуществляться различными методами. В функции density представлены несколько из них: правило подстановки с нормальным распределением (значение параметра **bw** — «nrd0»), модифицированное правило подстановки («nrd»), смещённый и несмещённый методы перекрёстной проверки («bcv» и «ucv») и метод Шизера–Джонса («SJ»).

Различие между оценками, построенными этими методами, представлено на рис. 10.

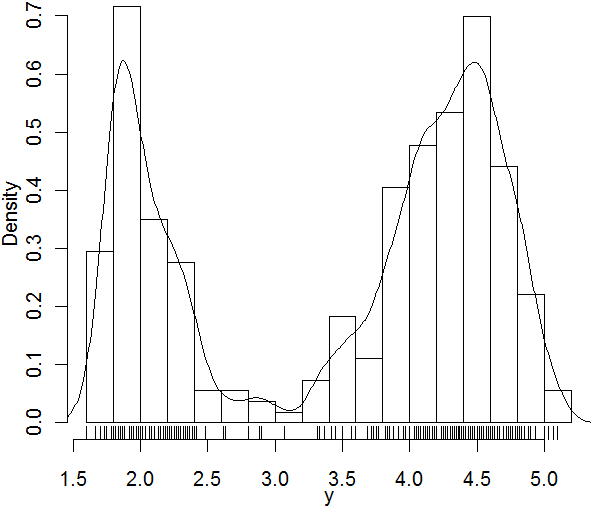


*Рисунок 10. Различные ядерные оценки.*

Параметр **bw** также позволяет вручную задать числовое значение сглаживающего параметра. Чем оно больше, тем более гладкой получается итоговая оценка.

Ещё одна полезная функция для эмпирического анализа, которая наносит на график выборочные данные, — функция rug. Для её использования достаточно указать в качестве аргумента ряд исходных данных.

rug(y)



*Рисунок 11. Гистограмма, ядерная оценка и эмпирические значения.*

Эмпирическая оценка функции распределения строится по формуле , где — выборочные значения. В **R** такая оценка реализуется функцией ecdf.

y.cdf <- ecdf(y)

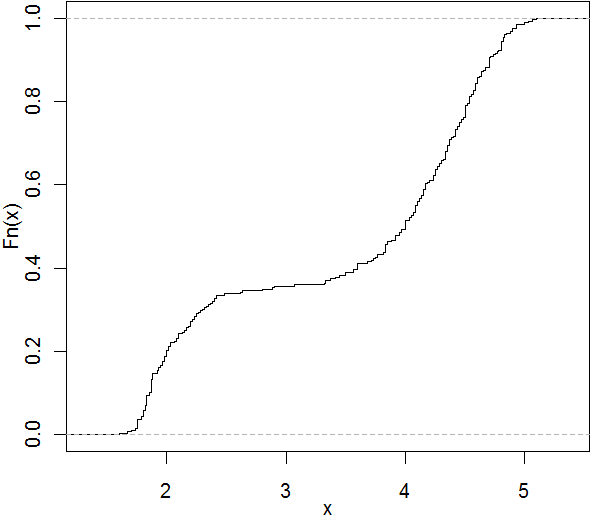
Функция ecdf является конструктором, т.е. возвращаемый ею объект также является функцией. Подставляя в **y.cdf** квантили, мы будем получать значения эмпирической оценки функции распределения в заданных точках.

y.cdf(3)

[1] 0.3566176

На рис. 12 представлен график оценки, построенный функцией plot.

plot(y.cdf,do.points=FALSE,verticals=TRUE)



*Рисунок 12. Эмпирическая оценка функции распределения.*

Функция plot — универсальная («generic»). С каждым объектом она выполняет свой набор действий, который может определяться различными параметрами. Для эмпирических оценок, полученных с помощью функции ecdf, существуют два дополнительных параметра: **do.points** — логический параметр, определяющий, нужно ли рисовать точки, в которых была произведена оценка, и **verticals** — логический параметр, определяющий, нужно ли рисовать вертикальные линии на ступенчатом графике эмпирической оценки.

## 2. Сравнение эмпирического распределения с затабулированным

В примере с длительностями извержений в ходе эмпирического анализа мы получили двухмодульную оценку плотности распределения этих величин (см. рис. 11). В качестве следующего примера мы возьмём правый колокол этого распределения, начинающийся со значения 3 по оси абсцисс, и сравним его с нормально распределённой случайной величиной.

Пусть *μ* и *σ* — параметры нормального распределения, тогда их несмещённые оценки, определённые через выборочные значения, выглядят следующим образом: . Нормальное распределение с параметрами и наилучшим образом, среди семейства нормальных, описывает исходные данные.

Первое, что мы можем сделать для сравнения эмпирического распределения с наиболее подходящим нормальным, — построить графики их плотностей и функций распределения.

y.right <- y[y>3]

x <- seq(2.5,5.5,by=0.01)

mu <- mean(y.right); sigma <- var(y.right)^0.5

# графики плотностей

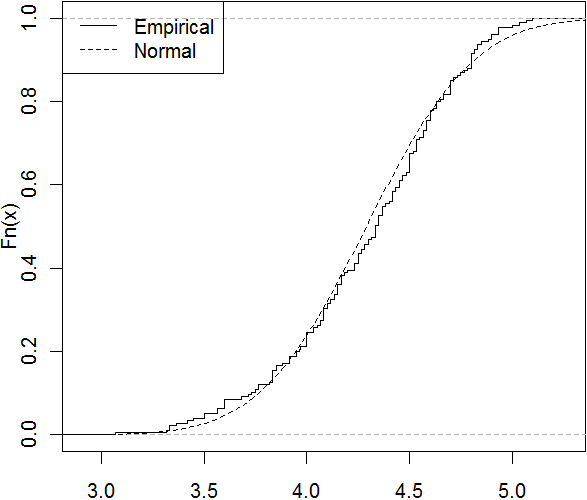
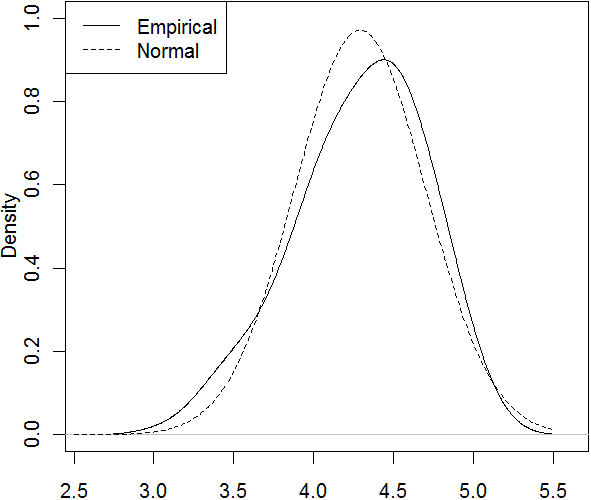
plot(density(y.right,n=1024,bw="ucv"),type="l",ylim=c(0,1))

lines(x,dnorm(x,mean=mu,sd=sigma),lty="dashed")

# графики функций распределения

plot(ecdf(y.right),do.points=FALSE,verticals=TRUE)

lines(x,pnorm(x,mean=mu,sd=sigma),lty="dashed")



*Рисунок 13. Сравнение эмпирических и нормальных функций плотности и распределения.*

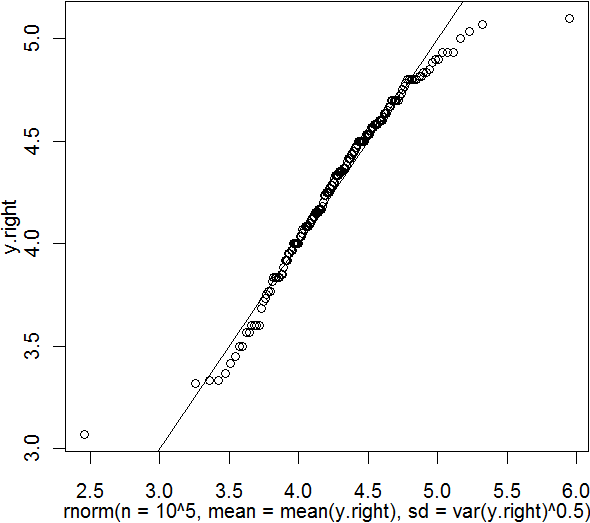
По виду графиков можно сказать, что распределение исследуемых величин скошено вправо. Его правый хвост заметно легче, чем у нормального распределения. Графически это выражается в более быстром убывании эмпирической функции плотности и более быстром возрастании эмпирической функции распределения в правых частях графиков. Левый хвост эмпирического распределения на отрезке [2.8; 3.7] тяжелее нормального, но затем он быстро истончается, поэтому нормальная функция распределения на далёких левых участках графика проходит выше эмпирической, равно как и соответствующая функция плотности.

Полезным инструментом для анализа хвостов распределения является график квантиль–квантиль. По осям этого графика откладываются значения квантилей эмпирического и теоретического распределения. Обычно эмпирическому распределению соответствует ось ординат. На этот график с некоторым шагом наносятся точки, одна из координат которых отвечает эмпирическому квантилю, а другая — теоретическому. Если подогнанное распределение в точности соответствует эмпирическому, то точки будут лежать на прямой линии , соответствующей равенству эмпирических и теоретических квантилей. По отклонениям точек от прямой можно судить об особенностях исследуемого распределения.

График квантиль–квантиль строится с помощью функции qqplot.

qqplot(rnorm(n=10^5,mean=mean(y.right),sd=var(y.right)^0.5),y.right)

abline(0,1)



*Рисунок 14. График квантиль–квантиль.*

В эту функцию в качестве аргументов необходимо подставить два числовых вектора, один из которых имеет известное распределение, а другой является исследуемым. В нашем примере в качестве вектора с известным распределением используется вектор из нормально распределённых случайных величин, полученный с помощью генератора случайных чисел.

Команда **ablilne(0,1)**  служит для проведения прямой линии. Её видимый наклон может отличаться от 45° из-за разного масштаба осей.

Приведённая на рисунке конфигурация графика отражает относительно тонкие (по сравнению с теоретическим) хвосты эмпирического распределения. Это можно понять по тому, что и в левой, и в правой его частях значения эмпирических квантилей ближе расположены к центру распределения, чем квантили теоретических. Если бы на каком-либо хвосте эмпирические квантили дальше отстояли бы от центра, чем соответствующие уровни теоретического распределения, то можно было бы говорить о сравнительно тяжёлых хвостах.

Наконец, по графику квантиль–квантиль на рис. 14 явно видно, что правый хвост распределения гораздо легче левого: значения правых эмпирических квантилей расположены ближе к центру, и поэтому сильнее отклоняются от прямой, чем эмпирические квантили на левом хвосте. Всё это свидетельствует о асимметричности (скошенности) распределения.

## 3. Тесты на нормальность

Методы графического анализа, являясь, безусловно, наглядными, допускают некоторую субъективность в суждениях. Разними исследователями одни и те же отклонения на графиках могут интерпретироваться и как значимые, и как незначимые. Далее мы рассмотрим более формальные процедуры, позволяющие проверить с определённой долей вероятности нормальность эмпирического распределения. Предпосылка о нормальности распределения встречается во многих статистических и эконометрических моделях, так что подобного рода проверки важны как с теоретической, так и с практической точек зрения.

Первый из рассматриваемых нами тестов был разработан в рамках параметрического подхода и предложен Шапиро и Уилком. Нулевая гипотеза этого теста состоит в том, что исследуемые величины имеют нормальное распределение:

.

Для проверки гипотезы формулируется тестовая статистика:

Критические значения статистики теста Шапиро–Уилка находятся из таблиц, рассчитанных Пирсоном и Хартли (1974).

За реализацию этого теста **в** R отвечает функция shapiro.test. Она имеет единственный аргумент — исходный ряд данных длиной от 3 до 5 000 значений.

shapiro.test(y.right)

Shapiro-Wilk normality test

data: y.right

W = 0.9793, p-value = 0.01052

В ходе выполнения этой функции будут выданы значения тестовой статистики *W* и p-значение, которое интерпретируется как вероятность отклонить верную нулевую гипотезу. Если p-значение мало, то гипотеза *H0* не принимается, иначе — не отклоняется. Определение порогового уровня значимости остаётся на усмотрение исследователя. Обычно он составляет от одного до десяти процентов.

В рассматриваемом нами случае мы можем принять нормальность ряда лишь на однопроцентном уровне значимости. Таким образом, тест показывает, что отклонения эмпирического распределения от нормального весьма велики.

Второй из обсуждаемых нами тестов предложен в рамках непараметрического подхода Колмогоровым и Смирновым. Его идея состоит в рассмотрении наибольшего отклонения между теоретической и эмпирической функциями распределения.

Нулевой гипотезой, как и в предыдущем тесте, является предположение о нормальности исходного распределения:

.

Тестовая статистика имеет следующий вид:

.

При условии истинности нулевой гипотезы статистика имеет распределение Колмогорова–Смирнова, из которого находятся его критические значения.

За выполнение этого теста в **R** отвечает функция ks.test.

ks.test(y.right,"pnorm",mean=mu,sd=sigma)

One-sample Kolmogorov-Smirnov test

data: y.right

D = 0.0661, p-value = 0.4284

alternative hypothesis: two-sided

Кроме вектора исходных данных, в ней следует указать вид распределения, по отношению к которому проводится тест («pnorm» для нормального, «pt» — для Стьюдента и т.д.), а также параметры этого распределения, в качестве которых мы используем несмещённые оценки. В результате выполнения функции будут найдены значение тестовой статистики и p-значение, которое интерпретируется аналогично тесту Шапиро–Уилка.

Можно заметить, что в нашем примере наблюдается разногласие в показаниях двух тестов: тест Шапиро–Уилка говорит о низкой вероятности принадлежности эмпирического распределения к классу нормальных, тест Колмогорова–Смирнова — о достаточно высокой. При принятии окончательного решения следует ориентироваться на мощность тестов. Мощность — это величина, обратная вероятности принять неверную нулевую гипотезу. Тест Шапиро–Уилка обладает более высокой мощностью, тогда как тест Колмогорова–Смирнова склонен принимать нормальность ряда, даже если это не так, поэтому в нашем случае гипотезу о нормальности, согласно тесту Шапиро–Уилка, не следует принимать на всех более-менее разумных уровнях значимости.

Преимуществом теста Колмогорова–Смирнова является то, что его можно использовать не только по отношению к нормальному распределению, как тест Шапиро–Уилка, но и ко всем распределениям из табл. 1.

## 4. Сравнение параметров двух нормальных выборок

Пусть и — две выборки из случайных реализаций нормальных величин. Обозначим — выборочные средние; — выборочные дисперсии.

Мы рассмотрим два теста на равенство параметров нормальных выборок: тест на равенство дисперсий и тест на равенство средних.

Статистические гипотезы F-теста на равенство средних выглядят следующим образом:

.

В условиях истинности нулевой гипотезы, как следует из теоремы Фишера, статистика

имеет распределение Фишера со степенями свободы *n – 1* и *m – 1*. В такой формулировке F-тест является двусторонним, и нулевая гипотеза не отклоняется, если , где — квантиль уровня *α* распределения Фишера.

В **R** реализована более общая версия этого теста.

n1 <- rnorm(n=100,mean=0,sd=1)

n2 <- rnorm(n=100,mean=0.1,sd=1.1)

var.test(n1,n2,ratio=1,alternative="two.sided",conf.level=0.95)

F test to compare two variances

data: n1 and n2

F = 0.7713, num df = 99, denom df = 99, p-value = 0.1981

alternative hypothesis: true ratio of variances is not equal to 1

95 percent confidence interval:

0.5189394 1.1462805

sample estimates:

ratio of variances

0.7712653

В функции var.test, кроме двух выборочных векторов, в параметре **ratio** следует указать гипотетическое отношение выборочных дисперсий, если предполагается, что оно отлично от единицы.

Параметр **alternative** служит для определения альтернативной гипотезы. При значении «two.sided» тест проводится в том виде, как он описан выше. Если задано значение «greater», то проверяется альтернатива ; если «less» — . Уточнять альтернативную гипотезу имеет смысл в случае наличия содержательных априорных соображений о характере выборочных дисперсий. Например, если известно, что дисперсия первой выборки не может превышать дисперсию второй, то сформулированная соответствующим образом альтернативная гипотеза позволит получить более точные доверительный интервал и p-значение.

Наконец, параметр **conf.level** задаёт уровень значимости при проведении теста.

В результате выполнения функции var.test будут рассчитаны, помимо прочего, доверительный интервал для отношения дисперсий и p-значение.

F-тест на равенство дисперсий крайне чувствителен к отклонению истинного закона распределения случайных величин от нормального, поэтому применять его следует, только убедившись в нормальности распределения исходных векторов.

Статистические гипотезы непарного t-теста на равенство средних выглядят следующим образом:

.

Если нулевая гипотеза верна, то тестовая статистика

имеет распределение Стьюдента с *n – m + 2* степенями свободы.

В **R** этот тест проводится с помощью функции t.test.

t.test(n1,n2,mu=0,alternative="two.sided",var.equal=TRUE,

conf.level=0.95)

t = -7.557, df = 198, p-value = 1.496e-12

alternative hypothesis: true difference in means is not equal to 1

95 percent confidence interval:

-0.4567099 0.1462155

Параметры этой функции те же, что и у var.test, только вместо гипотетического отношения дисперсий **ratio**, пользователь может задать гипотетическую разность средних **mu**.

Превышение тестовой статистикой критического уровня может быть вызвано как значимой разницей в средних значениях двух выборок, так и отличием их дисперсий. Ввиду этого тест на равенство средних рекомендуется проводить, указав в логическом параметре **var.equal** результат теста на равенство дисперсий.

## 5. Сравнение двух произвольных выборок

Непарный t-тест на равенство средних требовал нормальность распределения двух анализируемых выборок. Отказаться от этой предпосылки можно, обратившись к непараметрическому аналогу t-теста — критерию суммы рангов Уилкоксона[[5]](#footnote-5).

Алгоритм проведения теста таков. Выборочные ряды объединяются в один общий ряд: . Затем по этому ряду рассчитываются две суммы рангов: для элементов, принадлежащих первой выборке, и для элементов из второй.

.

Определяется максимальная сумма рангов и длина соответствующего ряда: , — и рассчитывается тестовая статистика

,

которая при истинности нулевой гипотезы о равенстве выборочных средних имеет затабулированное распределение. Критические значения статистики находятся из таблиц.

Функция wilcox.test, реализующая критерий Уилкоксона, имеет параметры, аналогичные тем, которые использовались в функции t.test.

t <- rt(n=100,df=5); n <- rnorm(n=100,mean=0,sd=1)

wilcox.test(t,n,mu=0,alternative="two.sided",conf.level=0.95)

Wilcoxon rank sum test with continuity correction

data: t and n

W = 5408, p-value = 0.3194

alternative hypothesis: true location shift is not equal to 0

Параметр **var.equal**, отвечавший за равенство выборочных дисперсий, указывать нет необходимости, поскольку в рамках теста мы не делали предположений относительно распределения двух выборок.

Тест Колмогорова–Смирнова может быть использован для проверки того, что две выборки имеют одинаковое неспецифицируемое распределение: . При условии истинности нулевой гипотезы тестовая статистика имеет распределение Колмогорова–Смирнова.

ks.test(t,n,alternative="two.sided")

Two-sample Kolmogorov-Smirnov test

data: t and n

D = 0.14, p-value = 0.2810

alternative hypothesis: two-sided

Функция ks.test, кроме выборочных рядов, имеет параметр **alternative**, позволяющий определить альтернативную гипотезу. Приведённому выше варианту соответствует значение «two.sided», для неравенств используются значения «less» и «greater».

## 6. Домашнее задание

1. Рассмотреть массив данных **nottem** о среднемесячных температурах в Ноттингеме в 1920–1939 годах.
   * определить ряды январских и декабрьских данных

jan <- nottem@.Data[seq(1,by=12,length=20)]

dec <- nottem@.Data[seq(12,by=12,length=20)];

* + проверить гипотезу о равенстве среднемесячных температур в январе и декабре;
  + проверить, одинаково ли распределение температур в эти месяцы.

1. Рассмотреть данные о доходности одного из биржевых индексов.
   * проверить нормальность распределения доходностей;
   * сделать выводы о скошенности распределения и тяжести его хвостов.

# Раздел 3. Основы регрессионного анализа

В модели множественной линейной параметрической регрессии рассматривается зависимость эндогенной переменной от набора экзогенных регрессоров.

В матричном виде:

, где .

Величины называются коэффициентами регрессии. Они отражают степень влияния каждого из факторов на объясняемую переменную. Разность между объясняемой переменной и совокупным влиянием регрессоров называется ошибкой регрессии, обычно обозначаемой , которая интерпретируется как случайная величина. В классической параметрической регрессионной модели на неё налагается ряд предпосылок. Во-первых, предполагается нормальность её распределения; во-вторых, — постоянство дисперсии для любых конкретных величин регрессоров; в-третьих, — независимость от своих предыдущих значений.

При соблюдении указанных условий оценки коэффициентов регрессии находятся по формуле

.

## 1. Оценка параметров регрессионной модели

Для практической иллюстрации построения регрессионной модели воспользуемся учебным примером из библиотеки **datasets**.

library(datasets)

ozone <- airquality$Ozone

rad <- airquality$Solar.R

В переменную **ozone** были записаны данные о концентрации озоне в атмосфере, а в переменную **rad** — уровень солнечной радиации. Данные собирались в г. Нью-Йорк за период с мая по сентябрь 1973 г.

Мы рассмотрим зависимость концентрации озона от солнечной радиации, исходя из следующих соображений. Как известно из курса неорганической химии, под действием солнечного света (ультрафиолетовой его части) молекулы кислорода, содержащиеся в атмосфере, расщепляются на атомы, которые затем объединяются в молекулу озона. Эту химическую реакцию можно записать в виде . Чем сильнее облучение, тем более интенсивно идёт реакция, поэтому мы можем ожидать прямую зависимость между регрессором и объясняемой переменной.

Для метеорологических данных, как и для многих других, характерно наличие пропущенных значений, т.е. отсутствие данных об озоне или радиации за некоторые дни. До построения модели необходимо решить, как поступать с пропусками. Самый простой и часто применяемый способ — исключить наблюдения, имеющие пропуски.

miss <- is.na(ozone) | is.na(rad)

ozone <- ozone[!miss]; rad <- rad[!miss]

Переменная **miss** представляет собой логический вектор, значения TRUE которого соответствуют пропущенным значениям озона или радиации. Пропуски отслеживаются с помощью функции is.na, имеющей один аргумент — вектор значений произвольного типа. Функция is.na возвращает логический вектор со значениями TRUE на месте значений NA (пропущенных значений) в векторе-аргументе и значениями FALSE на остальных местах. Мы применяем эту функцию и к регрессору, и к объясняющей переменной, а затем логическим оператором «или» объединяем два полученных логических вектора в переменную **miss**. Наконец, мы модифицируем переменные **ozone** и **rad**, исключая из них пропущенные значения. Подробнее об операциях с векторами см. в Разделе 1.

Разделим всю выборку наблюдений на обучающую и экзаменующую части. На пространстве обучающей выборки мы будем осуществлять оценку коэффициентов регрессии и проверку предпосылок насчёт её ошибки, а на экзаменующей части — строить прогноз.

Поскольку сбор данных осуществлялся в течение последовательного набора дат, то анализируемые ряды имеют временную структуру. Определить экзаменующую выборку, не нарушая этой структуры, можно, взяв в качестве неё несколько последних (или первых) наблюдений.

N <- length(ozone); E <- 20; T <- N-E

train.obs <- 1:T; eval.obs <- (T+1):N

t.rad <- rad[train.obs]; t.ozone <- ozone[train.obs]

e.rad <- rad[eval.obs]; e.ozone <- ozone[eval.obs]

Если бы данные имели пространственный характер, т.е. собирались бы с некоторых объектов, разделённых в пространстве, а не во времени, то в качестве экзаменующей выборки целесообразно было бы взять несколько наблюдений со случайно определёнными номерами, как в примере ниже.

eval.obs <- sample(1:N,size=E,replace=FALSE)

train.obs <- (1:N)[-eval.obs]

В этом примере использована функция sample, которая позволяет случайным образом выбрать элемент вектора-аргумента. Параметр **size** этой функции определяет объём выборки, логический параметр **replace** — возможность выбрать один и тот же элемент несколько раз. У пользователя также есть возможность задать дополнительный параметр **prob**, определяющий вероятности выбора каждого из элементов вектора. Параметр задаётся с помощью числового вектора длиной, равной длине вектора, из которого производится выборка. Значения **prob** прямо пропорциональны вероятностям выбора соответствующих элементов целевого вектора, однако они или их сумма могут превышать единицу (в этом случае вектор **prob** автоматически нормируется). Если параметр **prob** не задан, каждый элемент целевого вектора имеет равную вероятность быть выбранным.

Расчёт параметрической линейной регрессионной модели в R проводится функцией lm.

fit.par <- lm(ozone ~ radiation, data=data.frame(radiation=t.rad,ozone=t.ozone),weights=NULL)

Первый аргумент этой функции — спецификация модели. Слева от тильды стоит обозначение эндогенной (зависимой) переменной, справа — набор регрессоров (в рассматриваемом примере он один), разделённых знаком «+».

Параметр **data** служит для расшифровки обозначений эндогенной и экзогенных переменных. Подставляемое в него значение должно иметь формат «таблица данных» («data frame»), а названия столбцов этой таблицы должны соответствовать обозначениям, введённым при спецификации модели. Для оценки регрессии мы используем данные из обучающей выборки.

Наконец, параметр **weight** позволяет задать веса каждому наблюдению при проведении процедуры взвешенного МНК[[6]](#footnote-6). Если не задавать этот параметр (или задать значение NULL), каждое наблюдение будет иметь одинаковый вес.

Расшифровку спецификации модели в параметре **date** можно не приводить, если всем обозначениям регрессоров и объясняющей переменной соответствуют уже определённые переменные в рабочем пространстве. Например, следующая запись полностью эквивалентна приведённой выше.

fit.par <- lm(t.ozone ~ t.rad)

При поиске значений переменных регрессионной модели **R** сначала обращается к параметру **data**. Если описание какой-либо переменной отсутствует в этом параметре, или он не задан, **R** продолжает поиск среди объявленных переменных в рабочем пространстве, что позволяет в некоторых случаях не указывать параметр **data**.

Функция summary позволяет получить таблицу регрессионных коэффициентов. Эта функция, как и функция plot, является универсальной и при подстановке в неё переменной с результатами подгонки линейной регрессионной модели отображает коэффициенты регрессии и соответствующие статистики.

summary(fit.par)

Estimate Std. Error t value Pr(>|t|)

(Intercept) 22.18688 8.02140 2.766 0.00690 \*\*

radiation 0.12879 0.03816 3.375 0.00109 \*\*

Multiple R-squared: 0.1135, Adjusted R-squared: 0.1035

F-statistic: 11.39 on 1 and 89 DF, p-value: 0.001094

Коэффициенты при регрессоре и константе значимы на низком уровне, p-значение для F-теста на равенство нулю всех коэффициентов также мало (здесь оно совпадает с p-значением при коэффициенте единственного регрессора), поэтому, несмотря на низкий R2, мы можем сделать вывод о наличии тесной связи между рассматриваемыми показателями. Коэффициенты регрессии в этой модели стандартного вида интерпретируются как изменение эндогенной переменной при изменении экзогенной на 1 единицу.

Переменная **fit.par**, в которую был записан результат выполнения функции lm, имеет тип «список» и состоит, по крайней мере, из трёх компонент: **fit.par$coefficients** — вектор коэффициентов регрессии, **fit.par$fitted.values** — вектор модельных значений объясняемой переменной и **fit.par$residuals** — вектор остатков регрессии, равных разности между фактическими и модельными значениями объясняемой переменной.

На рис. 15 представлен график регрессионной линии на фоне диаграммы рассеяния. Диаграмма рассеяния состоит из точек, одна из координат которых представляет собой значение экзогенной переменной, а другая — значение эндогенной. Она строится с помощью функции plot.

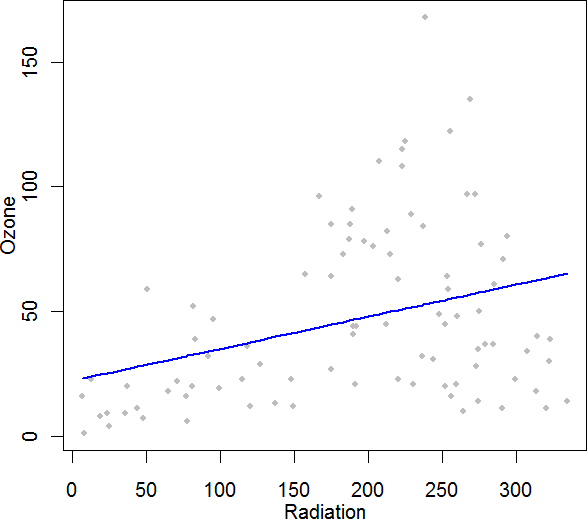
plot(t.rad,t.ozone,xlab="Radiation",ylab="Ozone",col="grey",pch=20)

Для проведения линии регрессии необходимо упорядочить наблюдения относительно объясняющей переменной. Функция order возвращает такую перестановку элементов её аргумента, при которых их значения следуют в порядке возрастания.

z <- order(t.rad)

Теперь мы можем построить линию регрессии оператором lines, нужным образом переставляя элементы векторов.

lines(t.rad[z],fit.par$fitted.values[z],col="blue",lwd=2)



*Рисунок 15. Линия регрессии Ozone ~ Rad.*

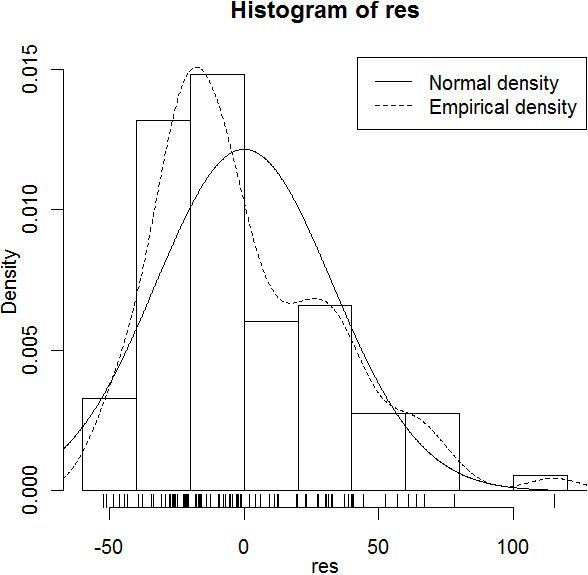
В функциях lines и plot использовались два параметра, не обсуждённые ранее. Параметр **pch** (от слов «point character») устанавливает символ для обозначения точки. По умолчанию это окружность, при значении 20 — круг. Параметр **lwd** (от слов «line width») изменяет толщину рисуемой линии. Единичная толщина является стандартной.

## 2. Проверка нормальности остатков модели

Помимо значимости коэффициентов регрессии, важной характеристикой построенной модели является выполнение предпосылок о свойствах ошибки регрессии. Сама ошибка — ненаблюдаемая величина, однако мы можем использовать остатки регрессии в качестве её оценок.

Первое, что можно и нужно сделать, — проверить остатки на нормальность. Графический анализ представлен на рис. 16.

res <- fit.par$residuals



*Рисунок 16. Гистограмма и эмпирическая плотность остатков регрессии Ozone ~ Rad.*

На графике хорошо заметна левосторонняя скошенность и даже двумодальность эмпирической плотности распределения остатков. Кроме того, хвосты эмпирического распределения не ведут себя как нормальные: левый хвост значительно легче, а правый — тяжелее. Таким образом, можно заключить, что имеющаяся выборка остатков, скорее всего, распределена по отличному от нормального закону.

Выводы, сделанные нами на основе графического анализа, подтверждаются формальными тестами. В табл. 2 представлены p-значения тестов на нормальность Шапиро–Уилка и Жарка–Бера.

shapiro.test(res)

library(fBasics)

jarqueberaTest(res)

*Таблица 2. p-значения тестов на нормальность остатков регрессии Ozone ~ Rad.*

|  |  |
| --- | --- |
| Тест | p-значение |
| Шапиро–Уилка | 0.000 |
| Жарка–Бера | 0.001 |

Тест Жарка–Бера основан на сравнении скошенности (skewness) и островершинности (kurtosis) эмпирического и нормального распределений: , где

— выборочная скошенность,

— выборочная островершинность.

В случае истинности нулевой гипотезы тестовая статистика асимптотически имеет хи-квадрат распределение с двумя степенями свободы.

Неудовлетворительные результаты тестов на нормальность остатков требуют переформулировки модели, поскольку в текущем её виде она не удовлетворяет начальным предпосылкам, а корректность выводов, основанных на ней, вызывает сомнения. Выбор подходящей спецификации модели — это творческий процесс, который иногда облегчается некоторыми содержательными соображениями. Рассмотрим следующую модель.

fit.par <- lm(log(ozone) ~ rad + rad2,

data=data.frame(rad=t.rad,rad2=t.rad^2,ozone=t.ozone))

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.585e+00 2.514e-01 6.304 1.12e-08 \*\*\*

rad 2.337e-02 3.313e-03 7.055 3.77e-10 \*\*\*

rad2 -5.660e-05 9.564e-06 -5.919 6.11e-08 \*\*\*

Multiple R-squared: 0.4234, Adjusted R-squared: 0.4103

F-statistic: 32.3 on 2 and 88 DF, p-value: 3.019e-11

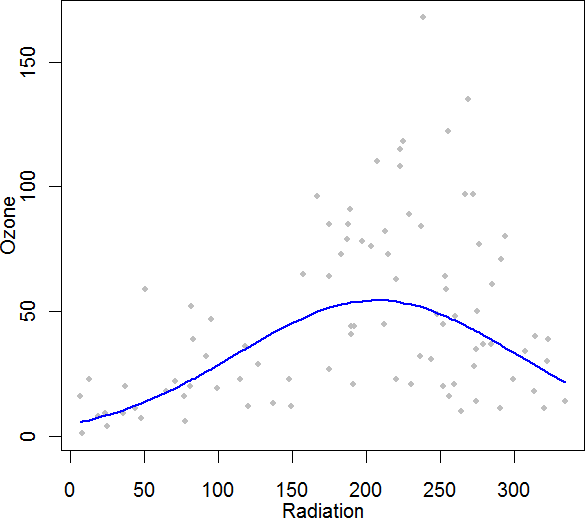
В эту модель нами был введён второй регрессор — квадрат уровня радиации. Это было сделано для учёта возможной нелинейной связи между показателями. Кроме того, был осуществлён переход к полулогарифмической спецификации модели. Коэффициенты регрессии в этом случае интерпретируются как процентное изменение эндогенной переменной при изменении экзогенной на одну единицу.

Все коэффициенты регрессии по-прежнему значимы, R2 значительно возрос. Качество подгонки модели, тем самым, улучшилось.

График линии регрессии второй модели представлен на рис. 17. Заметим, что при построении регрессионной линии в функции lines мы указываем не сами модельные значения, а их экспоненты, поскольку объясняемая переменная модели — это логарифм исследуемой величины.

plot(t.rad,t.ozone,xlab="Radiation",ylab="Ozone",col="grey",pch=20)

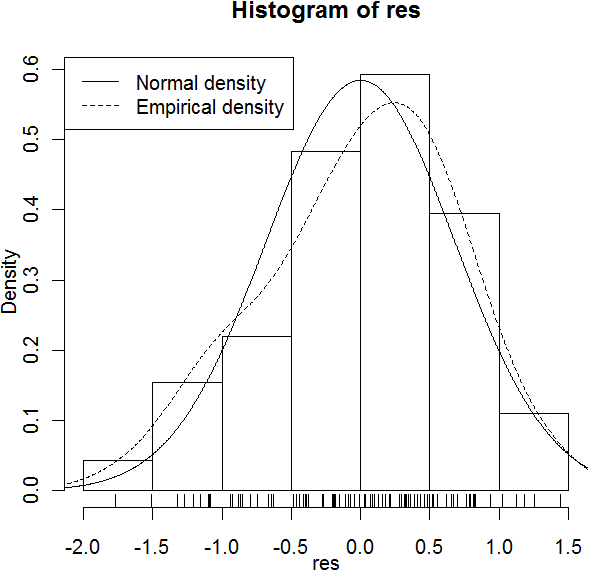
lines(t.rad[z],exp(fit.par$fitted.values[z]),col="blue",lwd=2)



*Рисунок 17. Регрессионная линия расширенной модели.*

Линия регрессии показывает нам интересную особенность представленной модели: начиная с некоторой величины экзогенного показателя, его влияние на объясняемую переменную сменяется с прямого на обратное. Действительно, если сформировать выборку наблюдений только для высоких значений радиации и построить на ней первую из рассмотренных нами моделей, то коэффициент при регрессоре будет отрицательным. Таким образом, ещё одним плюсом расширенной модели является учёт непостоянства влияния регрессора на эндогенную переменную.

Графический анализ нормальности остатков представлен на рис. 18.



*Рисунок 18. Гистограмма и эмпирическая плотность остатков расширенной регрессии.*

Такое распределение остатков уже больше похоже на нормальное. Хотя в нём также присутствует скошенность и неравномерность хвостов, эти отклонения имеют гораздо менее явный характер, чем в первой модели, и, видимо, могут быть объяснены случайными флуктуациями в не очень большой по размеру выборке.

Результаты тестов на нормальность остатков расширенной модели представлены в табл. 3.

*Таблица 3. p-значения тестов на нормальность остатков расширенной регрессии.*

|  |  |
| --- | --- |
| Тест | p-значение |
| Шапиро–Уилка | 0.288 |
| Жарка–Бера | 0.279 |

Поскольку p-значения больше 0.05, формально на 5% уровне значимости мы можем принять нормальность остатков. Тем самым, первая предпосылка регрессионной модели выполнена.

## 3. Тесты на гетероскедастичность

Проверка второй предпосылки, о постоянстве дисперсии, осуществляется путём проведения тестов на гетероскедастичность. Их идея состоит в том, что дисперсия ошибки может зависеть от некоторого набора показателей, в частности, от регрессоров модели.

Проведение первого из рассматриваемых нами тестов, Бреуша–Пагана, основано на оценки вспомогательной регрессии квадратов остатков на некоторый набор переменных. Пусть основная модель формулируется в виде

Определим остатки как разность между фактическими и модельными значениями:

.

Поскольку матожидание ошибки регрессии *ε­t­* равно нулю, то оценкой её дисперсии в момент *t* является квадрат соответствующей ошибки: . Для проверки зависимости дисперсии от некоторого набора показателей рассматривается регрессия нормированных квадратов остатков:

.

В случае гомоскедастичности дисперсия ошибки является постоянной, поэтому все коэффициенты при регрессорах должны быть равны нулю. Таким образом, мы можем сформулировать статистические гипотезы:

.

Проверка гипотез требует расчёта статистики Бреуша–Пагана , которая в случае истинности нулевой гипотезы имеет хи-квадрат распределение с *n* степенями свободы.

Второй тест на гетероскедастичность, Голдфелда–Квандта, рассматривает зависимость дисперсии от одного показателя, который обычно выбирается среди регрессоров модели. Все наблюдения, по которым строится модель, сортируются по возрастанию показателя, предположительно влияющего на дисперсию. Средние *f* наблюдений, соответствующие промежуточным значениям показателя, опускаются, и независимо рассматриваются две исходные регрессии на оставшихся подвыборках.

Если дисперсия ошибки постоянна, то квадраты остатков, соответствующие меньшим значениям объясняющей переменной, должны примерно совпадать с квадратами остатков на второй части выборки. Для проверки этого предположения, которое формулируется в виде

находится значение тестовой статистики , в случае истинности *H0* имеющей F-распределение с степенями свободы.

Рассмотрим две функции, реализующие описанные тесты в **R**. Эти функции доступны после установки и объявления библиотеки lmtest.

Первая, bptest, осуществляет процедуру Бреуша–Пагана.

library(lmtest)

bptest(fit.par,varformula=NULL,data=NULL,studentize=FALSE)

Breusch-Pagan test

data: fit.par

BP = 3.2609, df = 2, p-value = 0.1958

В ней, во-первых, необходимо указать регрессионную модель, проверяемую на гетероскедастичность, или переменную, в которой сохранены результаты выполнения функции lm. В нашем примере — это переменная **fit.par**.

Следующие два параметра, **varformula** и **data**, применяются, если набор объясняющих переменных в уравнении для дисперсии отличается от такового в основной модели, т.е. для задания набора , отличного от . В параметре **varformula** специфицируется регрессионное уравнение для дисперсии таким же образом, как это делается в функции lm, но без указания эндогенной переменной. Например, будет корректной запись varformula=~z1+z2, если переменные **z1** и **z2** объявлены ранее в рабочем пространстве. В случае если это не так, вектора значений переменных можно указать в параметре **data** в формате «таблица данных». Не указывая значения **varformula**  и **data** или присваивая им NULL, пользователь оставляет тот же набор регрессоров для объяснения дисперсии, что и в основной модели.

Наконец, параметр **studentize** позволяет вычислить модифицированную статистику Бреуша–Пагана, расчёт и интерпретация которой находятся за рамками курса. Ввиду этого параметру рекомендуется присваивать значение FALSE.

Функция bptest возвращает, среди прочего, значение тестовой статистики и p-значение, по которому пользователь может судить об истинности нулевой гипотезы.

Вторая функция, gqtets, проводит тест Голдфелда–Квандта.

gqtest(fit.par,fraction=25,order.by=t.rad,alternative="two.sided")

Goldfeld-Quandt test

data: fit.par

GQ = 0.8085, df1 = 30, df2 = 30, p-value = 0.5641

Как и в функции bptest, прежде всего, необходимо задать тестируемую регрессионную модель или соответствующую ей переменную.

Следующие два параметра, **fraction** и **order.by**, разделяют выборку наблюдений на две подвыборки. Производится сортировка данных по убыванию переменной, указанной в **order.by**, и затем удаляется определённое параметром **fraction** количество центральных наблюдений.

Наконец, **alternative**, по аналогии с ранее рассмотренными тестами, позволяет задать вид альтернативной гипотезы. Допустимы значения: «two.sided» — , «less» — , «greater» — . По умолчанию проверяется последний вариант.

Основными выходными данными функции являются тестовая статистика и p-значение.

В рассмотренном примере мы получили удовлетворительные результаты обоих тестов. В обратной ситуации, во-первых, можно снова переформулировывать модель до тех пор, пока не возникнет гомоскедастичная ошибка. Вторая возможность — сохранить текущую модель, придав каждому наблюдению вес, обратно пропорциональный оценки среднеквадратичного отклонения ошибки. Эти оценки можно рассчитать, например, из модели для дисперсии в тесте Бреуша–Пагана, а затем, используя параметр **weights** функции lm, заново оценить основную модель взвешенным МНК.

## 4. Учёт автокорреляции

Среди тестов на автокорреляцию наиболее широко используется тест Дарбина–Ватсона. Он основан на анализе коррелированности остатков регрессии и имеет статистику, при большом количестве наблюдений линейно зависящую от коэффициента корреляции первого порядка.

.

Статистические гипотезы формулируются следующим образом:

.

Как и ранее, спецификация альтернативной гипотезы может варьироваться.

В функции dwtest, отвечающей за этот тест и также содержащейся в библиотеке lmtest, два основных параметра: регрессионная модель и спецификация альтернативной гипотезы.

dwtest(fit.par,alternative="two.sided")

Durbin-Watson test

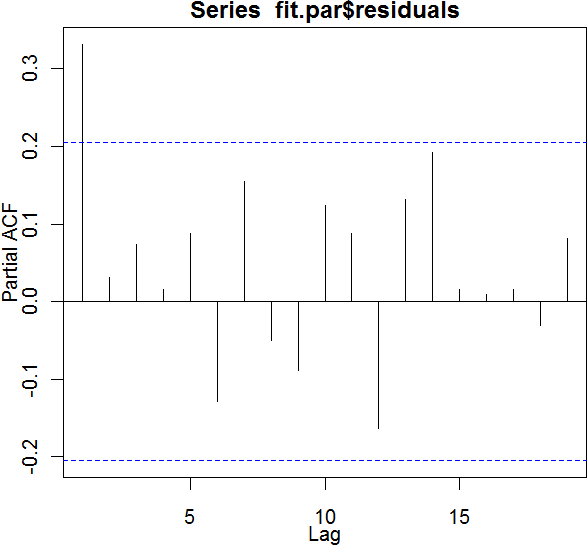
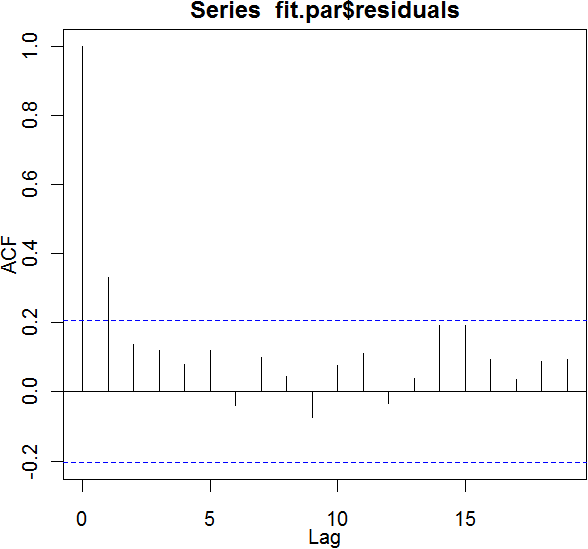
data: fit.par

DW = 1.3138, p-value = 0.0006297

alternative hypothesis: true autocorrelation is not 0

Основные выходные данные — тестовая статистика и p-значение, которое в нашем случае вынуждает не принимать нулевую гипотезу.

Убедиться в наличии автокорреляции остатков в рассматриваемой модели можно также, построив графики автокорреляционной и частной автокорреляционной функций.



*Рисунок 19. Значения автокорреляционной (слева) и частной автокорреляционной (справа) функций для различных лагов в расширенной модели.*

На этих графиках корреляция первого порядка выходит за пределы доверительных интервалов, т.е. значимо отличается от нуля.

С учётом того, что для последующих лагов автокорреляция является незначимой, усовершенствовать модель мы попробуем, добавив в качестве одного из регрессоров лагированные на один период значения эндогенной переменной. Таким образом, модель примет вид:

.

Поскольку модель специфицирована в полулогарифмическом виде, для расчёта численных значений авторегрессионной компоненты также следует использовать логарифм.

ar1 <- log(t.ozone[1:(T-1)])

Оценим модель с авторегрессией. В ходе подгонки модели нами было отброшено первое наблюдение, так как для него не определено значение авторегрессионной компоненты.

fit.par <- lm(log(ozone) ~ rad + rad2 + ar1,

data=data.frame(ozone=t.ozone[2:T],rad=t.rad[2:T],rad2=t.rad[2:T]^2))

Estimate Std. Error t value Pr(>|t|)

(Intercept) 7.296e-01 3.201e-01 2.279 0.025123 \*

rad 1.901e-02 3.313e-03 5.739 1.40e-07 \*\*\*

rad2 -4.365e-05 9.601e-06 -4.546 1.78e-05 \*\*\*

ar1 3.156e-01 8.086e-02 3.904 0.000188 \*\*\*

Multiple R-squared: 0.5108, Adjusted R-squared: 0.4937

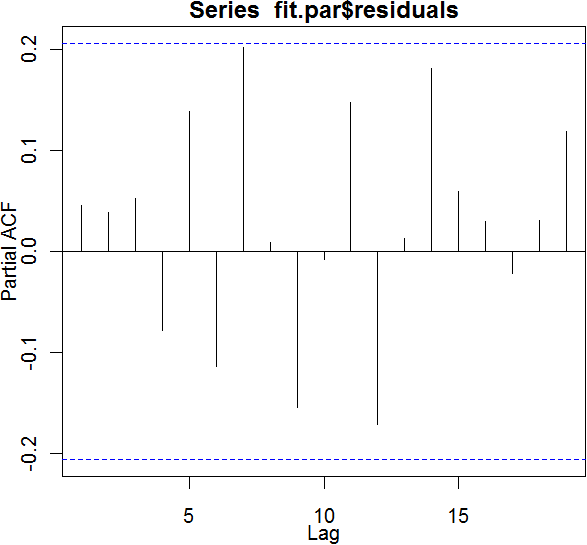
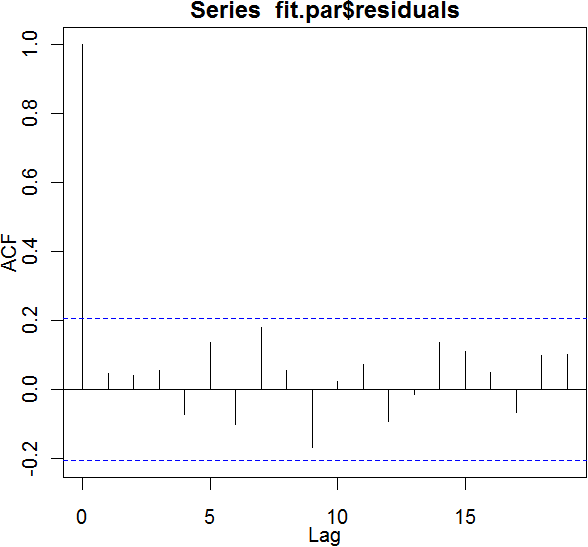
Все коэффициенты модели по-прежнему значимы, а R2 относительно высок.

Результаты тестов на нормальность и гетероскедастичность представлены в табл. 4.

*Таблица 4. Результаты тестов на нормальность и гетероскедастичность авторегрессионной модели.*

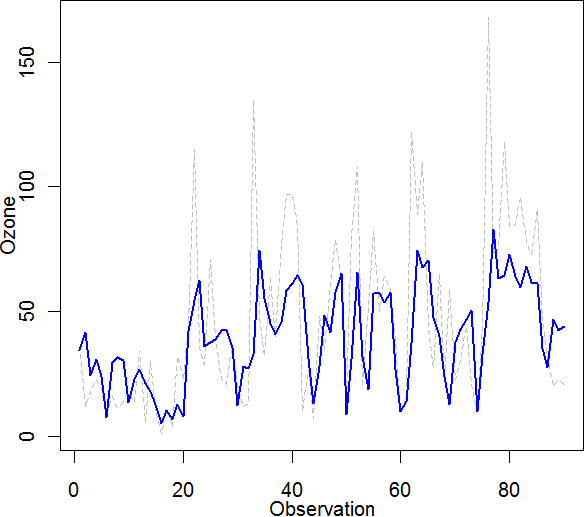
|  |  |
| --- | --- |
| Тест | p-значение |
| Шапиро–Уилка | 0.613 |
| Жарка–Бера | 0.582 |
| Бреуша–Пагана | 0.213 |
| Голдфелда–Квандта | 0.657 |

Тест Дарбина–Ватсона на автокорреляцию можно применять, только когда регрессоры независимы от ошибки, что не так в авторегрессионных моделях, однако отсутствие корреляции ошибок регрессии мы модем подтвердить графиками автокорреляционных функций.



*Рисунок 20. Значения автокорреляционной (слева) и частной автокорреляционной (справа) функций для различных лагов в автокорреляционной модели.*

Линия регрессии авторегрессионной модели представлена на рис. 21, где по горизонтальной оси вместо значений экзогенной переменной теперь отложены номера наблюдений (периоды времени), поскольку авторегрессионная спецификация модели предполагает, что исследуемый процесс, прежде всего, развивается во времени, и его последующие значения влияют на предыдущие.



*Рисунок 21. Регрессионная линия авторегрессионной модели.*

Финальную версию построенной нами модели можно считать хорошей, поскольку она не только имеет значимые коэффициенты, но и удовлетворяет всем начальным предпосылкам. Далее мы используем эту модель для построения прогноза на экзаменующей выборке.

## 5. Построение прогноза и доверительных интервалов

Рассмотрим регрессионную модель , в которой случайная ошибка удовлетворяет обсуждённым выше предпосылкам. В этом случае прогноз , где , является несмещённым () и обладает наименьшей среднеквадратичной ошибкой среди всех линейных несмещённых прогнозов. Среднеквадратичная ошибка прогноза при этом составит . Заменим дисперсию ошибки на её оценку (дисперсию остатков) и рассчитаем оценку среднеквадратичного отклонения прогноза: , — тогда доверительным интервалом для прогнозного значения будет отрезок

,

где — квантиль распределения Стьюдента с *T – d* степенями свободы, 1 – *α* — уровень значимости.

Рассмотрим построение прогноза и доверительных интервалов с помощью функции predict.

frc.par <- predict(fit.par,

newdata=data.frame(rad=e.rad,rad2=e.rad^2,

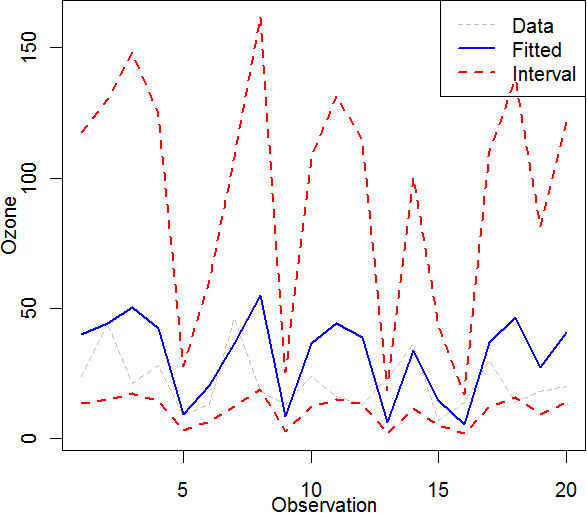
ar1=log(ozone[eval.obs-1])),se.fit=TRUE,

interval="prediction",level=0.90)

В эту функцию, во-первых подставляется модель или соответствующая ей переменная. Во-вторых, в параметре **newdata** указывается набор значений экзогенных переменных, который будет использован для построения прогноза. Если параметр не задан, модельные значения будут рассчитаны по тем значениям регрессоров, которые использовались для подгонки модели, т.е. фактически будет построена линия регрессии. Логический параметр **se.fit** определяет, нужно ли отдельно вычислять среднеквадратические ошибки прогноза, параметр **level** задаёт уровень значимости, а текстовому параметру **interval** при построении прогноза следует присвоить значение «prediction».

Переменная **frc.par**, в которую был записан результат выполнения функции predict, будет иметь тип «список». В компоненте, обозначаемой «fit» будет содержаться матрица из трёх столбцов, первый из которых соответствует самим прогнозным значениям, а второй и третий — нижней и верхней доверительным границам. В компоненту «se.fit» записывается вектор среднеквадратических ошибок, если их расчёт был задан в функции predict.

На рис. 22 изображен прогноз с доверительными интервалами.



*Рисунок 22. Прогноз и доверительные интервалы по авторегрессионной модели.*

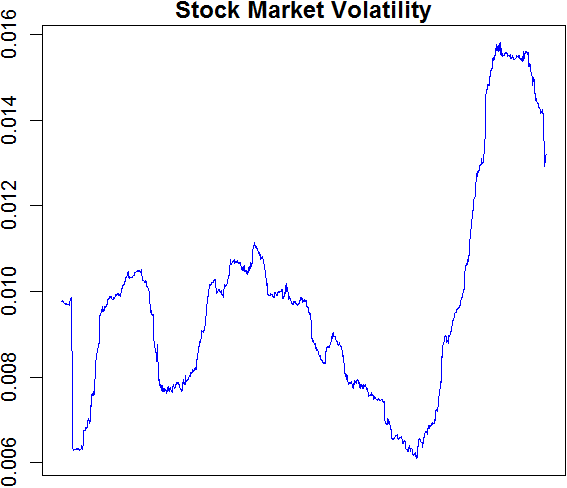
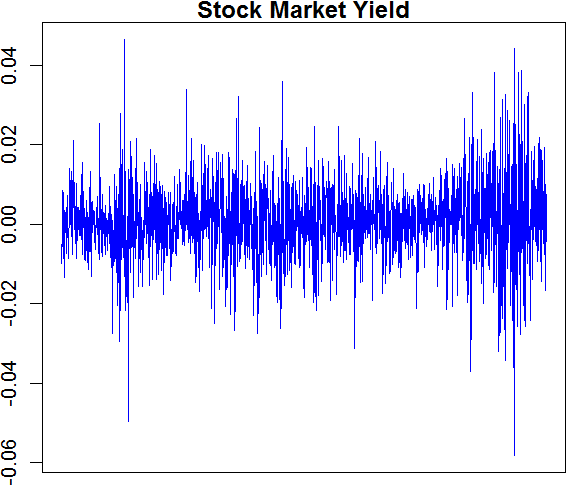
## 6. Домашнее задание

Рассмотреть данные о величине спроса на деньги в пакете lmtest: moneydemand

* разделить выборку на обучающую и экзаменующую части, обосновать способ разделения;
* на пространстве обучающей выборки построить регрессионную модель с эндогенной переменной logM;
* проверить качество модели с помощью тестов на нормальность, гетероскедастичность и автокорреляцию; при необходимости переформулировать модель или внести иные корректировки;
* построить прогноз и доверительные интервалы для эндогенной переменной на экзаменующей выборке;
* написать содержательные комментарии.

# Раздел 4. Моделирование волатильности

Финансовые временные ряды имеют набор характерных особенностей, которые следует учитывать при моделировании. Одна из таких особенностей — эффект кластеризации волатильности, выражающийся в чередовании периодов с резкими изменениями цен финансовых активов и относительно спокойных периодов. Это иллюстрируется на рис. 23.



*Рисунок 23. Доходность (слева) и волатильность (справа) фондового индекса.*

Левый график доходностей обладает волнистой структурой: более широкие его участки соответствуют более высокой волатильности на правом графике.

Волатильность доходности финансовых инструментов, очевидно, не является белым шумом, и в её динамике можно проследить некоторые закономерности, моделированию которых посвящён этот раздел.

## 1. Моделирование средней доходности

Временной ряд доходностей финансового актива, как правило, подвержен автокорреляционным эффектам, и прежде, чем обратить внимание на волатильность, следует рассмотреть модель для самой доходности, или, как говорят, модель для среднего. В качестве таких моделей используют процессы семейства ARMA, где первый компонент, AR (от слова «autoregression»), отвечает за авторегрессионную составляющую, а второй, MA (от слов «moving average»), — за скользящую среднюю ошибок регрессии. Оценка ARMA-моделей предшествует моделированию волатильности, т.е. при объяснении волатильности рассматривают доходности актива, очищенные от ARMA-составляющей.

Пусть *yt* — доходность финансового актива в момент времени *t*, тогда регрессионное уравнение ARMA(n,m)-модели выглядит следующим образом:

.

Часто для записи модели используют лаговый оператор, применённый к некоторому значению временного ряда, возвращающий его предыдущее значение: .

.

Многочлены и называются лаговыми полиномами, и, хотя их содержательная интерпретация наталкивается на определённые трудности, они играют важную роль при изучении свойств как ARMA-моделей, так и моделей волатильности.

Рассмотрим учебный пример о доходностях британского фондового индекса FTSE за период с 11.05.1991 по 18.06.1998.

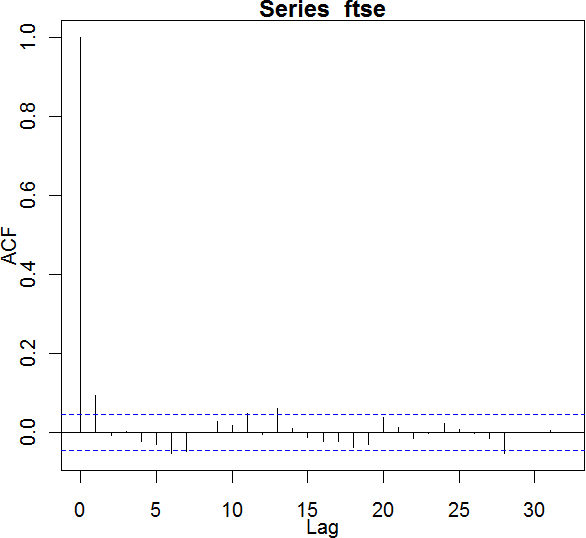
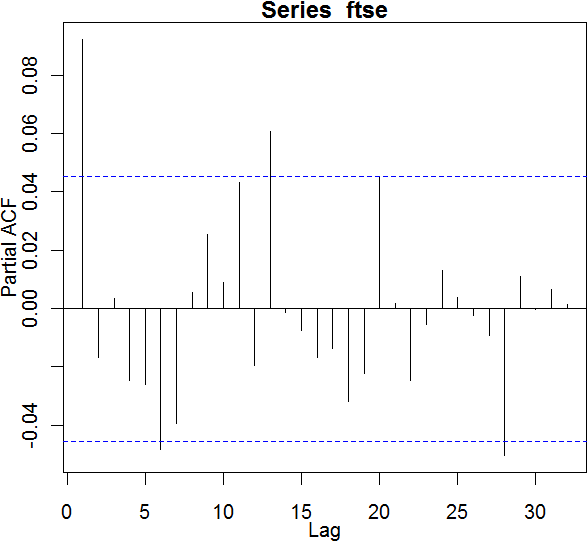
ftse <- EuStockMarkets[,"FTSE"]

Поскольку в массиве **EuStockMarkets** содержатся данные о ценах, к доходностям перейдём с помощью следующего преобразования.

T <- length(ftse) - 1

ftse <- ftse[2:(T+1)]/ftse[1:T] - 1

При использовании моделей класса ARMA, прежде всего, необходимо определить порядок модели, т.е. параметры *n* и *m*. Это можно сделать, рассмотрев графики автокорреляционных функций, которые обсуждались также в предыдущем разделе.

****

*Рисунок 24. Автокорреляционные функции доходностей индекса FTSE.*

По левому графику частной автокорреляционной функции определяется порядок AR-компоненты, правый отвечает за MA. На обоих графиках автокорреляции первого порядка выходят за доверительный интервал, а остальные (за исключением нескольких высоких порядков) являются незначимыми, что позволяет предположить порядок модели, не превышающий (1, 1). С учётом значимости коэффициентов наилучшей выглядит модель ARMA(1,0), характеристики которой, полученные с помощью функции arma из библиотеки tseries, представлены ниже.

library(tseries)

ftse.arma <- arma(ftse,order=c(1,0),lag=NULL)

summary(ftse.arma)

Estimate Std. Error t value Pr(>|t|)

ar1 0.0924415 0.0230993 4.002 6.28e-05 \*\*\*

intercept 0.0004173 0.0001842 2.265 0.0235 \*

AIC = -12703.77

Более формальным критерием выбора порядка модели является значение информационного критерия Акаике («Akaike Informational Criterion»), о котором более подробно будет говориться в следующем разделе. Меньшее значение критерия Акаике говорит о более высоком качестве подгонки модели. Таким образом, для определения точного количества лагов можно построить несколько моделей с различными значениями параметров *n* и *m* и выбрать ту, которой соответствует минимальное значение информационного критерия. Для формализации алгоритма подбора удобно использовать следующую запись значения критерия Акаике:

summary(ftse.arma)$aic

[1] -12703.77

Функция arma имеет три основных параметра.

Во-первых, это временной ряд, представленный числовым вектором, для которого оценивается модель.

Параметр **order**, задаваемый числовым вектором из двух элементов, определяет максимальное число лагов модели. Модель рассчитывается для всех лагов от первого до максимального.

Если есть необходимость рассчитать модель только для некоторых конкретных лагов, её параметр можно задать в параметре **lag**. Ему надлежит присвоить значение типа «список». В первой компоненте списка, которая должна быть названа «ar», указывается набор лагов для авторегрессионной составляющей, во второй, обозначаемой «ma», — лаги для скользящей средней. Например, чтобы учесть все автокорреляции, выходящие за пределы доверительных интервалов на рис. 24, можно специфицировать модель следующим образом.

ftse.arma <- arma(ftse,lag=list(ar=c(1,6,13,28),ma=c(1,6,7,13)))

Estimate Std. Error t value Pr(>|t|)

ar1 -0.0579858 0.1475824 -0.393 0.6944

ar6 -0.3558748 0.2235087 -1.592 0.1113

ar13 0.0287055 0.1694179 0.169 0.8655

ar28 -0.0515146 0.0227591 -2.263 0.0236 \*

ma1 0.1512882 0.1462454 1.034 0.3009

ma6 0.3242732 0.2310701 1.403 0.1605

ma7 -0.0126379 0.0313614 -0.403 0.6870

ma13 0.0183205 0.1659181 0.110 0.9121

intercept 0.0006430 0.0003036 2.118 0.0342 \*

AIC = -12699.91

Хотя некоторые из далёких лагов оказались значимыми, интерпретировать их коэффициенты довольно тяжело, так что мы остановимся на модели ARMA(1,0). Этот выбор также обоснован более низким значением критерия Акаике для первой модели.

## 2. Тест на ARCH-эффекты

Авторегрессионные эффекты могут наблюдаться не только в доходностях финансовых активов, но и в их волатильности. Такого рода зависимость получила название «авторегрессионной условной гетероскедастичности» («Autoregressive Conditional Heteroskedastity», ARCH). Моделирование волатильности и её кластеризации основано на учёте ARCH-эффектов, и прежде чем приступить к моделированию, необходимо убедиться в их наличии.

Тест множителей Лагранжа (LM-тест) позволяет проверить, характерна ли авторегрессионная зависимость волатильности в рассматриваемых данных.

Пусть — остатки регрессионной модели для среднего, где — подогнанные значения по этой модели. Поскольку матожидание остатков равно нулю, оценкой волатильности (дисперсии) служат их квадраты. Рассмотрим регрессию

.

В случае отсутствия ARCH-эффектов все коэффициенты при лагированных остатках должны быть равны нулю. Таким образом, мы можем сформулировать статистические гипотезы LM-теста:

.

Обозначим и , тогда статистика

в случае истинности нулевой гипотезы будет иметь хи-квадрат распределение с *q* степенями свободы.

Вообще говоря, выбор количества лагов *q* теста множителей Лагранжа может повлиять на его результат, однако обычно оно фиксируется на уровне 12.

За проведение этого теста в **R** отвечает функция ArchTest из библиотеки FinTS.

library(FinTS)

ArchTest(ftse.arma$residuals,lags=12)

ARCH LM-test; Null hypothesis: no ARCH effects

data: ftse.arma$residuals

Chi-squared = 93.0058, df = 12, p-value = 1.288e-14

Ряд остатков модели для среднего мы получаем аналогично функции lm, рассмотренной в предыдущем разделе.

Маленькое p-значение говорит против принятия нулевой гипотезы об отсутствии ARCH-эффектов, и, тем самым, у нас есть все основания приступить к моделированию волатильности.

## 3. Модели условной гетероскедастичности

Регрессионное уравнение для дисперсии, GARCH- модель («Generalized ARCH»), выглядит следующим образом:

,

где — случайная ошибка регрессии в модели для среднего.

Величины мультипликативно раскладываются на независимые одинаково распределённые величины с нулевым матожидаением и единичной дисперсией и величины , зависящие от предыдущих значений ошибки и своих лагированных значений. Можно сказать, что отвечает за форму распределения ошибок, а — за размах их колебаний, т.е. за волатильность.

Размах колебаний зависит от квадрата ошибки. Выбор двойки как показателя степени сложился исторически; по тем же содержательным соображениям об учёте как положительных, так и отрицательных изменений, можно было бы использовать, например, модуль.

Существуют разные подходы к дальнейшему обобщению этой модели, среди которых мы рассмотрим степенное, содержащее все основные идеи этих обобщений, — модель APARCH («Asymmetric Power ARCH»).

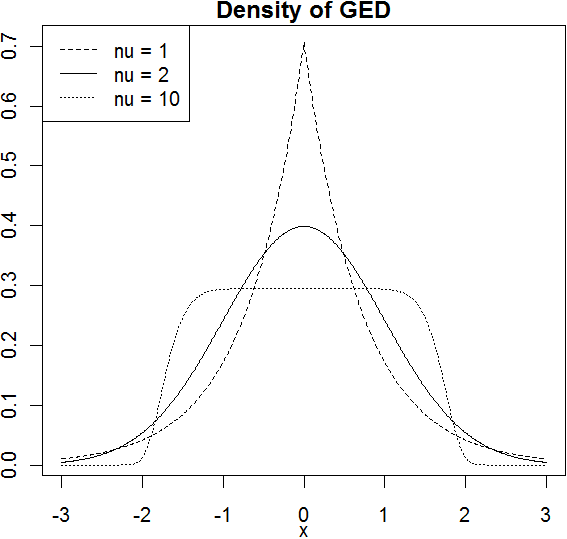
.

В модели APARCH показатель степени *δ* выступает в качестве одного из параметров, а также добавляется набор параметров , моделирующих неодинаковое влияние положительных и отрицательных шоков на дисперсию, что является ещё одной характерной особенностью финансовых данных. В самом деле, беспокойство инвесторов нарастает, когда рынок падает, т.е. когда наблюдаются отрицательные ошибки, но оно не столь велико при растущем рынке.

Важным аспектом формализации модели является выбор вида распределения величины . В стандартном GARCH-подходе имеет нормальное распределение, однако распределение рядов финансовых данных имеет более толстые хвосты, чем у нормального, поэтому для моделирования стандартизированной компоненты часто используется обобщённое распределение ошибок (Generalized Error Distribution, GED). Гибкость этого распределения проиллюстрирована на рис. 25. Функция плотности GED записывается следующим образом:

, где

, Γ(.) — гамма-функция, — параметр формы.



*Рисунок 25. Графики функций плотности стандартного обобщённого распределения ошибок* GED(0,1,ν) *в зависимости от параметра формы ν.*

Отметим, что частными случаями GED являются нормальное распределение (*ν* = 2), распределение Лапласа (*ν* = 1) и равномерное распределение (*ν* → +∞). Альтернативой обобщённому распределению ошибок может служить распределение Стьюдента, не являющееся частным случаем GED. Хвосты t-распределения столь же тяжелы, однако оно, в отличие от толтохвостого варианта GED (с параметром *ν* < 2) имеет округлую, а не острую вершину.

Рассмотрим практическую реализацию APARCH, за которую отвечает функция garchFit из библиотеки fGarch.

library(fGarch)

ftse.gfit <- garchFit(formula=~arma(1,0)+aparch(1,1),data=ftse,

cond.dist="ged",include.delta=TRUE,leverage=TRUE,trace=FALSE)

В параметре **formula** этой функции следует указать спецификацию модели. При этом если исследователь работает с очищенными от ARMA-эффектов доходностями, он может подбирать для них чистую APARCH-модель, а при работе и исходными данными удобно использовать комбинацию ARMA-APARCH, как показано в примере. Порядок авторегрессии и скользящего среднего был определён нами ранее. Что касается количества лагов в модели для волатильности, то оно обычно устанавливается равным единице, чтобы не переусложнять модель.

В параметре **data** указывается моделируемый временной ряд: очищенный или нет, в зависимости от спецификации модели.

В параметре **cond.dist** указывается текстовая строка, соответствующая предполагаемому распределению стандартизированных компонент : «norm» и «snorm» для нормального и скошенного нормального распределений, «std» и «sstd» — для распределений Стьюдента, «ged» и «sged» — для обобщённых распределений ошибок.

Логические параметры **include.delta** и **leverage** (рычаг) определяют, следует ли включать параметры и в модель.

Наконец, логический параметр **trace** носит технический характер и ответственен за вывод на экран информации об этапах подгонки модели. Обычно, чтобы не загромождать экран, его значение устанавливается равным FALSE.

Используя комбинации степенного параметра и рычага, можно получать различные частные случаи модели, наиболее известные из которых показаны ниже. Например, простая GARCH-модель получается путём фиксирования параметра **delta** на уровне 2 и присвоения рычагу значения FALSE.

GARCH(1,1): .

garchFit(formula=~aparch(1,1),data=ftse,delta=2,include.delta=FALSE,

leverage=FALSE,trace=FALSE)

Не упоминавшийся ранее параметр **delta** имеет разный смысл, в зависимости от того, подбирается ли степенной параметр наряду с другими параметрами модели, или же он фиксирован. В первом случае **delta** задаёт начальное значение для итеративного процесса оптимизации, во втором определяет само фиксированное значение. В обоих вариантах значение по умолчанию равно двум.

Модель TS-GARCH(1,1) отличается от классической показателем степени:

.

garchFit(formula=~aparch(1,1),data=ftse,delta=1,include.delta=FALSE,

leverage=FALSE,trace=FALSE)

Очевидно, что набор параметров для её расчёта почти полностью соответствует предыдущему примеру, за исключением значения параметра **delta**.

T-GARCH(1,1) или пороговый GARCH позволяет моделировать неодинаковое влияние положительных и отрицательных ошибок, не прибегая к степенному обобщению.

.

garchFit(formula=~aparch(1,1),data=ftse,delta=2,include.delta=FALSE,

leverage=TRUE,trace=FALSE)

В этом примере значение параметра **leverage** устанавливается равным TRUE. Отметим, однако, что коэффициенты и , являясь аналогами и из модели APARCH, имеют иное численное значение из-за разницы в формулировке моделей.

## 4. Анализ качества GARCH-модели

Рассмотрим спецификацию модели, в которой в качестве условного распределения используется распределение Стьюдента.

ftse.gfit <- garchFit(formula=~arma(1,0)+aparch(1,1),data=ftse,

cond.dist="std",include.delta=TRUE,leverage=TRUE,trace=FALSE)

Как и для остальных регрессионных моделей, с помощью функции summary мы можем увидеть таблицу коэффициентов и их статистик.

summary(ftse.gfit)

Estimate Std. Error t value Pr(>|t|)

mu 3.535e-04 1.656e-04 2.135 0.03278 \*

ar1 6.567e-02 2.348e-02 2.797 0.00515 \*\*

omega 2.861e-05 1.068e-05 2.680 0.00736 \*\*

alpha1 4.064e-02 1.020e-02 3.983 6.79e-05 \*\*\*

gamma1 6.948e-01 2.173e-01 3.197 0.00139 \*\*

beta1 9.503e-01 9.914e-03 95.857 < 2e-16 \*\*\*

delta 1.287e+00 3.001e-01 4.289 1.79e-05 \*\*\*

shape 1.000e+01 1.939e+00 5.157 2.50e-07 \*\*\*

AIC BIC SIC HQIC

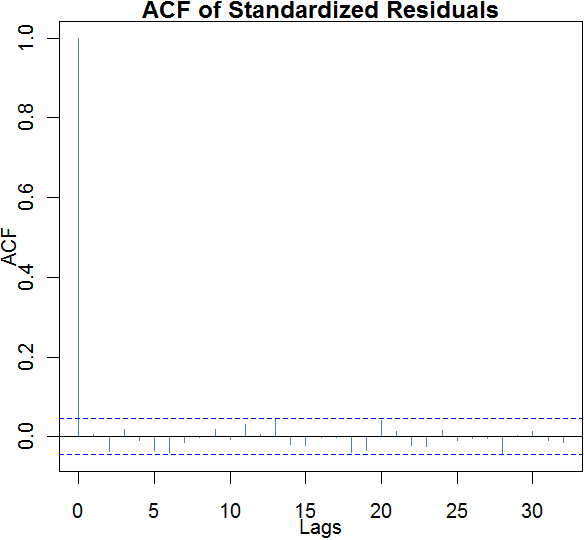
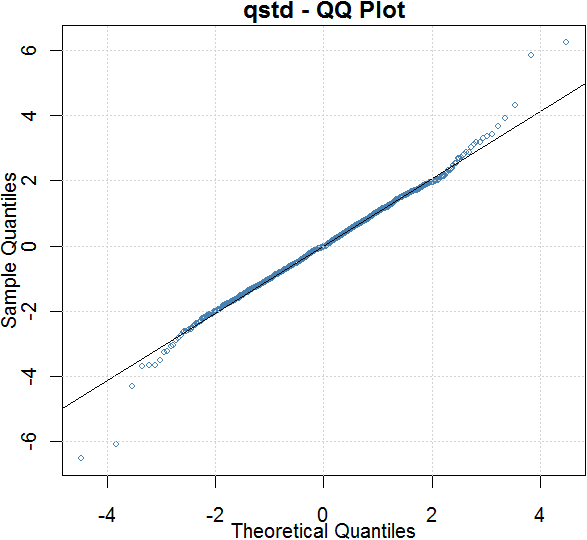
-6.919726 -6.895937 -6.919762 -6.910959

Коэффициенты *mu* и *ar1* относятся к модели для среднего, а *shape* представляет собой количество степеней свободы распределения стандартизованных остатков, которое является параметром модели.

Для управления формой распределения могут быть использованы два до сих пор не упомянутых параметра функции garchFit: **include.shape** и **shape**, — которые по аналогии со степенным параметром позволяют фиксировать параметр формы на определённом уровне.

Наряду с анализом значимости коэффициентов, при выборе наилучшей спецификации модели может быть полезен набор информационных критериев, меньшее значение которых соответствует более высокому качеству подгонки.

Функция plot используется при проведении графического анализа. При подстановке в неё GARCH-модели, которая в нашем случае сохранена в переменной **ftse.gfit**, она выдаёт ряд графиков, в том числе график квантиль–квантиль для условного распределения и автокорреляционную функцию остатков.



*Рисунок 26. График квантиль–квантиль (слева) и автокорреляционная функция (справа).*

Глядя на левый график, можно заключить, что, несмотря на использование тяжелохвостого распределения Стьюдента с малым количеством степеней свободы, хвосты эмпирического распределения остатков тяжелее модельных, и модель поэтому будет склонна недооценивать риск.

На этом графике также можно анализировать асимметричность распределения, которой в нашем случае не наблюдается.

Правый график автокорреляции остатков позволяет ещё раз убедиться в правильности спецификации модели для среднего и при необходимости скорректировать её.

## 5. Стационарность и единичные корни

Понятие стационарности является важным при построении прогнозов и расчёте мер риска. Стационарный ряд — это тот, распределение которого не меняется со временем. Если это так, мы можем использовать данные, накопленные за предыдущие периоды времени, для того, чтобы сделать некие выводы о динамике этого показателя в будущем. В обратной ситуации поведение временного ряда становится непредсказуемым.

Пусть — некоторый стохастический процесс, и мы имеем выборку его значений . В практическом смысле каждый его элемент — это фиксированное число, но, с точки зрения теории, на месте этого набора данных мог бы быть другой, поскольку его реализация случайна. Таким образом, мы можем рассматривать выборку как совокупность случайных величин, порождённых процессом . Раз так, то возникает функция их совместного распределения.

Рассмотрим две подвыборки размера *k*, разделённых *τ* периодами. Процесс называется стационарным в узком смысле (строго стационарным), если для любых *k* и *τ* функции совместного распределения двух подвыборок тождественно равны:

.

Поскольку строгую стационарность довольно трудно проверить, то формулируется также понятие стационарности в широком смысле (слабой стационарности), в котором функции распределения заменяются на матожидания, дисперсии и ковариации:

.

Из стационарности в широком смысле следует стационарность в узком.

В качестве примера рассмотрим процесс AR(n):

.

Выразим случайную ошибку через лаговый полином и рассмотрим его корни:

.

Если значение *L* ≤ 1 является корнем этого уравнения, то дисперсия процесса зависит от времени, и он не является стационарным в широком смысле. Маленький корень *L* ведёт к большому значению соответствующего коэффициента , а это содержательно означает, что влияние ошибки на *i-*м лаге затухает медленно, и оно, накапливаясь, делает процесс нестационарным.

Примером нестационарного процесса может служить случайное блуждание (Random Walk, RW).

.

Очевидно, что корень лагового полинома равен единице.

.

Путём рекурсивных подстановок мы можем выяснить, что величина сводится к сумме ошибок за предыдущие периоды, скорректированной на начальное значение.

.

В случае если начальное значение представляет собой фиксированное число, а ошибки независимы, дисперсия будет равна сумме дисперсий ошибок, т.е. расти со временем, что является признаком нестационарности.

.

## 6. Тесты на единичные корни

На практике стационарность проверяется в ходе применения тестов на единичный корень к остаткам модели. Мы рассмотрим три наиболее часто используемых теста: Дики–Фуллера (Argumented Dickey–Fuller, ADF), Филипса–Перрона (Philips–Perron, PP) и Квятковского–Филипса–Шмидта–Шина (Kwiatkowski–Phillips–Schmidt–Shin, KPSS). Для проведения каждого из них оценивается вспомогательная регрессия и проводится анализ значимости её коэффициентов.

Вспомогательная регрессия теста Дики–Фуллера имеет вид:

.

Если ряд остатков нестационарен, тогда его приращение в момент времени *t* полностью объясняется трендовой составляющей и приращениями в предыдущие моменты времени. В этом случае добавление в качестве регрессора величины не несёт в себе дополнительной объясняющей силы, и коэффициент при ней должен быть равен нулю, т.е. незначим. Это отражено в статистических гипотезах.

.

Функции для проведения тестов на единичный корень в **R** содержатся в библиотеке tseries. Определим вектор остатков модели и проведём тест Дики–Фуллера.

res <- ftse.gfit@residuals

adf.test(res)

Augmented Dickey-Fuller Test

data: res

Dickey-Fuller = -11.2673, Lag order = 12, p-value = 0.01

alternative hypothesis: stationary

На выходе функция выдаёт, среди прочего, значение тестовой статистики, p-значение и содержательную формулировку альтернативной гипотезы. Поскольку p-значение мало, нулевая гипотеза о наличии единичного корня не принимается.

Идея теста Филипса–Перрона является, пожалуй, наиболее интуитивно понятной: в нём напрямую проверяется равенство модуля авторегрессионного коэффициента единице.

.

.

Истинность нулевой гипотезы, как и в ADF-тесте, означает наличие единичного корня и нестационарность исследуемого процесса.

pp.test(res)

Phillips-Perron Unit Root Test

data: res

Dickey-Fuller Z(alpha) = -1698.572, Truncation lag parameter = 8,

p-value = 0.01

alternative hypothesis: stationary

Аналогично предыдущему случаю, функция pp.test выводит p-значение и альтернативную гипотезу.

В тесте KPSS предполагается, что константа вспомогательной регрессии представляет собой случайное блуждание.

— стационарный процесс,

.

Однако, если дисперсия процесса равна нулю, то он вырождается в фиксированное число, что соответствует стационарности остатков.

.

kpss.test(res)

KPSS Test for Level Stationarity

data: res

KPSS Level = 0.0795, Truncation lag parameter = 9, p-value = 0.1

Поскольку нулевая гипотеза KPSS-теста противоположна гипотезам двух рассмотренных ранее тестов, высокое p-значение свидетельствует в пользу стационарности остатков.

KPSS-тест является самым мощным среди рассмотренных: он лучше всего разделяет определяет нестационарные процессы, тогда тесты Дики–Фуллера и Филипса–Перрона при большом количестве наблюдений склонны считать ряды стационарными, даже если это далеко не так. Однако, являясь асимптотическим, тест Квятковского даёт правдивые результаты, только если количество наблюдений измеряется тысячами. Таким образом, мы можем рекомендовать применять ADF и PP-тесты при малом количестве наблюдений, а KPSS — при большом.

## 7. Построение оценок риска

«Граница потерь» (Value-at-Risk, VaR) — мера риска, определяемая как квантиль распределения доходностей актива:

.

Пусть — прогноз на один период по модели для среднего, — прогноз волатильности по модели условной гетероскедастичности, тогда «границу потерь» можно рассчитать следующим образом:

,

где — квантиль уровня 1 – *α*.

Вычисление прогноза осуществляется с помощью функции predict.

ftse.frc <- predict(ftse.gfit,n.ahead=1)

meanForecast meanError standardDeviation

1 0.001028503 0.01336069 0.01336069

При подстановке в неё оценённой GARCH-модели или соответствующей переменной и указании горизонта прогноза в параметре **n.ahead** она выдаёт матрицу из **n.ahead** строк и трёх столбцов, первый из которых представляет собой прогноз среднего, а третий — прогноз волатильности.

Теперь, задав некоторый уровень значимости *α*, мы можем рассчитать границу потерь.

alpha <- 0.95

VaR <- ftse.frc[,1]+ftse.frc[,3]\*qstd(1-alpha,mean=0,sd=1,

nu=ftse.gfit@fit$par["shape"])

-0.02063070

Выбор функции для расчёта квантиля стандартизированных остатков зависит от того, какое условное распределение оценивалось в GARCH-модели. В примере использована функция qstd, возвращающая квантиль распределения Стьюдента, поскольку ранее мы задавали параметр **cond.dist** равным «std», т.е. условным распределением служило распределение Стьюдента.

Параметры **mean** и **std** квантильной функции задаются нулём и единицей соответственно, поскольку таковы предпосылки для распределения в GARCH-модели.

Параметр формы **nu** (для распределений, имеющих этот параметр), в случае если он был фиксирован в модели для дисперсии задаётся этим же значением. В случае, если он подбирался наряду с остальными параметрами модели, его значение берётся из вектора подогнанных параметров **ftse.gfit@fit$par**. Элементы этого вектора имеют собственные имена; именем параметра формы является «shape».

Аналогично можно задать параметр скошенности **xi**, если в качестве условного распределения рассматривался асимметричный вариант. Имя параметра скошенности — «skew».

## 8. Проверка качества оценок риска

Основной процедурой проверки качества построенных оценок доверительного уровня ожидаемых потерь (оценки качества прогноза) является тест Купика.

Пусть мы имеем набор последовательных во времени оценок доверительного уровня потерь, оценённых при помощи показателя «граница потерь» с некоторым уровнем значимости *α*. Легко понять, что если модель, лежащая в основе расчёта VaR, адекватно описывает исследуемый объект, то относительное количество пробоев кривой VaR (т.е. количество элементов экзаменующей выборки, превышающих эту границу, по отношению к длине всей экзаменующей выборки) должно быть равно *α*. В противном случае модель либо переоценивает (в случае меньшего количества пробоев), либо недооценивает (в случае большего количества) риск. Пусть *E* — длина экзаменующей выборки, *K* — количество пробоев кривой VaR, тогда  — эмпирический уровень значимости. Тест Купика заключается в проверке следующей статистической гипотезы:

, .

Проверка осуществляется с помощью статистики

,

которая в случае истинности нулевой гипотезы имеет  распределение.

Таким образом, для построения кривой VaR нам необходимо определить обучающую и экзаменующую выборки. Поскольку ряд данных имеет чёткую временную структуру, в качестве экзаменующей выборки возьмём некоторое количество последних наблюдений. Обучающую выборку определим так, что оценки риска будут рассчитываться по фиксированному количеству предыдущих наблюдений. Такой подход разумен при наличии большого количества наблюдений, так как далеко отстоящие во времени значения уже вряд ли оказывают какое-либо влияние на текущую динамику доходности актива. Обозначим количество наблюдений в обучающей выборке как **h**, в экзаменующей — **E**.

h <- 100; E <- 300

Выбор длины обучающей выборки может сильно повлиять на величину и качество оценок риска, однако не существует каких-то чётких рекомендаций относительно этого выбора. Из общих соображений можно сказать, что чем меньше **h**, тем быстрее оценки риска реагируют на изменение рыночной конъюнктуры. С другой стороны, при малом количестве наблюдений в обучающей выборке мы существенно обедняем её наблюдениями, находящимися на хвостах распределения, т.е. не принимаем во внимание возможность возникновения значительных доходов и убытков. Выбор конкретного значения **h** должен определяться компромиссом между двумя указанными свойствами, и он остаётся на усмотрение исследователя.

Рассмотрим модель условной гетероскедастичности

garchFit(formula=~aparch(1,1),data=h.ftse,cond.dist="ged",

include.shape=FALSE,shape=1.15,include.delta=FALSE,delta=0.5,

leverage=TRUE,trace=FALSE)

Построение кривой VaR для проверки качества оценок риска на основе этой модели естественно организовать в виде цикла, счётчик которого пробегает все номера наблюдений экзаменующей выборки.

for (i in (T-E+1):T) {

h.ftse <- ftse[(i-h):(i-1)]

h.ftse.gfit <- garchFit(formula=~aparch(1,1),data=h.ftse,

cond.dist="ged",include.shape=FALSE,shape=1.15,include.delta=FALSE,

delta=0.5,leverage=TRUE,trace=FALSE)

h.ftse.frc <- predict(h.ftse.gfit,n.ahead=1)

VaR[i-T+E] <- h.ftse.frc[1,1]+h.ftse.frc[1,3]\*qged(1-alpha,

mean=0,sd=1,nu=1.15)

}

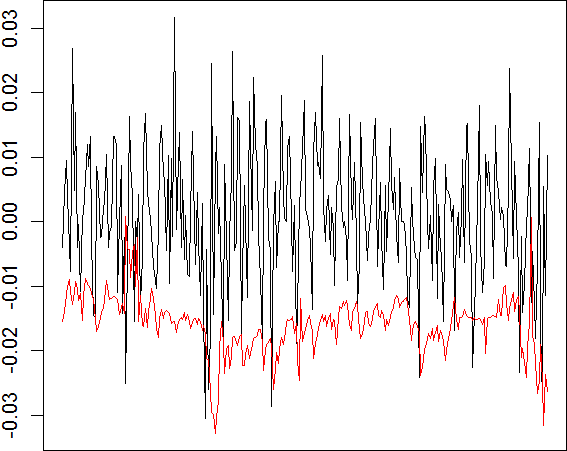
Первая команда в теле цикла определяет набор наблюдений, входящий в обучающую выборку на каждой итерации. Затем по этим данным производится подгонка представленной выше модели и рассчитывается оценка риска. Цикл повторяется до тех пор, пока не исчерпается вся экзаменующая выборка.

На рис. 27 представлено сравнение кривой VaR с соответствующими ей фактическими доходностями.

fact <- ftse[(T-E+1):T]

plot(fact,type="l",ylim=range(c(fact,VaR)))

lines(VaR,col="red")



*Рисунок 27. Кривая «границы потерь».*

Рассчитаем количество превышений фактическими убытками кривой VaR и проведём тест Купика.

K <- sum(fact<VaR)

alpha0 <- 1-K/E

S <- -2\*log((1-alpha)^K\*alpha^(E-K)) +

2\*log((1-alpha0)^K\*alpha0^(E-K))

p.value <- 1-pchisq(S,df=1)

[1] 0.4403898

Высокое p-значение теста Купика говорит в пользу нулевой гипотезы о равенстве эмпирического и теоретического уровней значимости, поэтому мы можем сказать, что представленная модель позволяет получать хорошие оценки риска.

Ещё одним показателем качества, помогающим сделать выбор между моделями, прошедшими тест Купика, является значение функции потерь, измеряющей среднюю величину превышения фактическими убытками уровня VaR. Чем меньше значение функции потерь, тем более адекватно рассматриваемая модель оценивает риски.

Наиболее часто используемыми функциями потерь являются функции Лопеса[[7]](#footnote-7) и Бланко-Ила:

,

.

Функция потерь Лопеса отличается тем, что она придаёт больший вес значительным отклонениям. Это обосновано с содержательной точки зрения, поскольку единичные крупные превышения, как правило, более опасны, чем несколько мелких.

В нашем случае значения этих функций равны соответственно

LL <- 10^4\*sum((fact-VaR)^2\*(fact<VaR))/K

LBI <- sum(abs((fact-VaR)/VaR)\*(fact<VaR))/K

[1] 0.8434786

[1] 3.412754

## 9. Домашнее задание

Рассмотреть данные о доходности одного из биржевых индексов.

* написать пользовательскую функцию, в качестве аргументов в которую подставляются вектор доходностей и вектор из двух элементов, обозначающих максимальное количество лагов в модели для среднего, и которая возвращает оптимальный порядок ARMA-модели по критерию Акаике;
* построить модель условной гетероскедастичности и получить с помощью неё оценку «границы потерь»;
* обосновать качество полученной оценки путём построения кривой VaR и проведения теста Купика.

# Раздел 5. Оценка рыночных рисков с помощью моделей семейства обобщённого гиперболического распределения.

Распределение доходностей финансовых активов, как это обсуждалось в предыдущем разделе, обладает рядом специфических характеристик: унимодальность, среднее значение близко к нулю, тяжёлые хвосты плотности распределения, асимметричность. Поскольку нормальное распределение в силу недостаточной гибкости не может отразить все эти особенности, то для параметрического моделирования плотности распределения такого вида мы будем использовать семейство обобщённого гиперболического распределения (ОГР), которое наиболее полно среди параметрических моделей позволяет учесть указанные статистические особенности распределения доходности финансовых активов.

## 1. Оценка параметров модели

Класс обобщённого гиперболического распределения включает в себя целый ряд распределений (нормальное, t-распределение Стьюдента, нормальное обратное гауссовское, гиперболическое и др.) в качестве частных случаев, поэтому потенциально может обеспечить очень хорошую подгонку модели.

Функция плотности ОГР для одномерного случая выглядит следующим образом:

 где

 — параметр расположения;

 и — параметры, отвечающие за толщину хвостов;

 — параметр асимметричности;

 — параметр масштаба;



 — модифицированная функция Бесселя второго рода.

Из функции плотности  можно получать плотности других гиперболических распределений, соответствующим образом подбирая параметры. Например, распределение Стьюдента:



Параметры многомерных распределений можно оценивать различными способами; наиболее распространённый — метод максимального правдоподобия (ММП).

В качестве практической иллюстрации расчёта ОГР-моделей мы вновь обратимся к учебному примеру о доходностях европейских биржевых индексов. Рассмотрим германский индекс DAX.

dax <- EuStockMarkets[,"DAX"]

T <- length(dax) - 1

dax <- dax[2:(T+1)]/dax[1:T] - 1

Функции для работы с обобщённым гиперболическим распределением содержатся в библиотеке ghyp. Ниже представлена общая схема использования такой функции для подгонки модели.

library(ghyp)

fit.[…]uv(dax,symmetric=FALSE,silent=TRUE)

Вместо конструкции «[…]» в названии функции следует подставить обозначение распределения: ghyp — обобщённое гиперболическое, hyp — гиперболическое, NIG — нормально-обратное гауссовское, VG — дисперсионное гамма, t — t-распределение Стьюдента, gauss — нормальное.

Логический параметр **symmetric** позволяет оценивать как симметричные, так и скошенные варианты распределений, а параметр **silent** носит технический характер, аналогично параметру **trace** в функции garchFit.

## 2. Анализ качества модели

Рассмотрим модель ОГР:

dax.ghyp <- fit.ghypuv(dax,symmetric=FALSE,silent=TRUE)

С помощью функции summary можно увидеть оценки её параметров и значения оптимизационных критериев.

summary(dax.ghyp)

Parameters:

lambda alpha.bar mu sigma gamma

0.6789381670 0.6714003334 0.0008547615 0.0101770578 -0.0001500769

Optimization information:

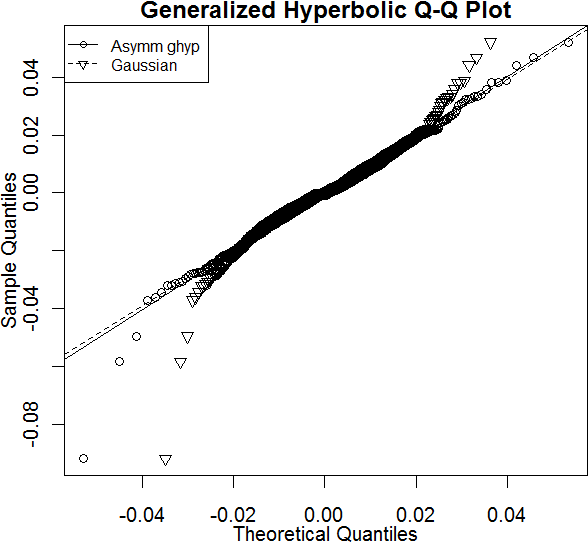
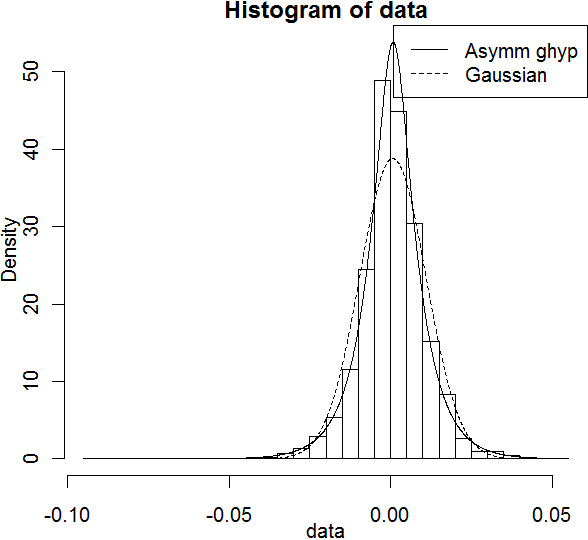
log-Likelihood: 5983.626

AIC: -11957.25

Для проведения графического анализа качества модели используются гистограмма распределения и график квантиль–квантиль.

hist(dax.ghyp)

qqghyp(dax.ghyp)



*Рисунок 28. Гистограмма плотности распределения (слева) и график квантиль–квантиль (справа).*

Пунктирная линия на гистограмме слева представляет собой оценку плотности нормального распределения, сплошная — оценку по модели ОГР. Гистограммы бывают полезны для определения скошенности и островершинности эмпирического распределения.

На графике квантиль–квантиль справа треугольниками изображены квантили нормального распределения по отношению к эмпирическим, а окружностями — квантили ОГР. Видно, что обобщённое гиперболическое распределение довольно хорошо подгоняет исходные данные. В частности, правый хвост ОГР, в отличие от нормального, практически совпадает с эмпирическим, а на левом хвосте расхождение намного меньше. По виду графика квантиль–квантиль можно сделать выводы о тяжелохвостости и асимметричности распределения. Например, можно сказать, что исследуемые доходности обладают тяжёлыми хвостами даже по меркам финансовой статистики, причём левый хвост заметно тяжелее правого.

Несмотря на гибкость ОГР, бывают случаи, когда использование этого распределения не оправдано с теоретической точки зрения. Если истинный закон распределения относится к классу гиперболических — например, является распределением Стьюдента — оценки параметров ОГР будут менее точны, чем оценки по распределению Стьюдента просто потому, что при заданной точности для большего количества параметров, которым обладает ОГР, требуется большее количество наблюдений. Существует ряд тестов, позволяющих соблюсти баланс между точностью подгонки и количеством параметров.

Рассмотрим упоминавшийся ранее тест отношения правдоподобия.

Несмотря на то, что, благодаря большему количеству параметров, более общее распределение даёт лучшую подгонку модельной функции плотности под эмпирическую, разница в качестве подгонок, измеряемых максимизированными значениями функции правдоподобия[[8]](#footnote-8), может оказаться статистически незначимой, т.е. близкой к нулю. Идея LR-теста состоит в приведении этой разницы к случайной величине, имеющей затабулированное распределение. Гипотезы и тестовая статистика формулируются следующим образом:

 разница в качестве подгонки близка к нулю (частная модель предпочтительней);

 гипотеза  неверна (общая модель предпочтительней).

 где

 и  — максимизированные значения функции правдоподобия для частной и более общей модели соответственно;

 — разница в степенях свободы между моделями.

Рассмотрим модель распределения доходностей, которая является частным случаем ОГР, — распределение Стьюдента.

dax.t <- fit.tuv(dax,symmetric=FALSE,silent=TRUE)

Тест отношения правдоподобия проводится с помощью функции lik.ratio.test.

lik.ratio.test(dax.ghyp,dax.t,conf.level=0.95)

$p.value

[1] 0.1671139

$H0

[1] TRUE

В качестве аргументов в эту функцию следует подставить две оценённые ОГР-модели, одна из которых обобщает другую. Также в параметре **conf.level** можно задать уровень значимости для проведения этого теста.

LR-тест позволяет сделать выбор между более общей моделью и её частным случаем, но не позволяет сравнивать частные модели.

Для сравнения частных моделей используется критерий Акаике (AIC), вводящий штраф за использование большого количества параметров.

 где

 — количество параметров модели.

Согласно этому критерию, предпочтительной следует считать ту модель, значение AIC для которой наименьшее.

Сравнить ряд распределений из класса ОГР и выбрать среди них наиболее подходящее в терминах информационного критерия позволяет функция stepAIC.ghyp.

aic.uv <- stepAIC.ghyp(dax,dist=c("gauss","t","ghyp"),

symmetric=NULL,silent=TRUE)

Для использования этой функции, кроме ряда доходностей, в параметре **dist** следует указать текстовый вектор с названиями распределений, среди которых будет проводиться выбор. Обозначения распределений соответствуют тем, которые используются в функции для подгонки модели.

Параметр **symmetric** может принимать три значения. При значении TRUE будут рассматриваться только симметричные варианты распределений; при значении FALSE — только скошенные; если задано значение NULL, — оба варианта.

В ходе работы этой функции будут оценены несколько моделей, в зависимости от конфигурации её параметров, и в целевую переменную **aic.uv**, которая будет иметь тип «список», среди прочего, в компоненту **best.model**  запишутся данные о наилучшей в терминах критерия Акаике модели. Увидеть эту информацию можно, воспользовавшись функцией summary.

summary(aic.uv$best.model)

Symmetric Generalized Hyperbolic Distribution:

Parameters:

lambda alpha.bar mu sigma gamma

1.2709094386 0.0045687496 0.0006422894 0.0101642556 0.0000000000

Optimization information:

log-Likelihood: 5984.574

AIC: -11961.15

В отличие от первого теста критерий Акаике отдаёт предпочтение более общей модели, а точнее, её симметричному варианту. Это кажущееся противоречие можно снять, проведя LR-тест для нескошенных распределений.

## 3. Расчёт оценок риска

По аналогии с распределениями из табл. 1 для распределений семейства ОГР существуют четыре специальных функции: генератор случайных чисел (rghyp), квантильная функция (qghyp), функции плотности и распределения (dghyp и pghyp). Эти функции одни и те же для всех распределений семейства, а конкретный вид распределения указывается в их параметре **object**. Например, чтобы сгенерировать 100 случайных значений по модели распределения Стьюдента, оценённой нами ранее, воспользуемся функцией rghyp.

rghyp(n=100,object=dax.t)

Заметим, что определённая по критерию Акаике наилучшая модель сохраняется в том же формате, что выдают функции подгонки, так что в эти четыре функции можно подставлять результаты расчётов по stepAIC.ghyp. Так, например, выглядит вычисление квантиля по наилучшей модели.

alpha <- 0.9

qghyp(1-alpha,object=aic.uv$best.model)

Указанные в примерах функции используются для расчёта измерителей риска. Кроме обсуждавшейся в предыдущем разделе «границы потерь», мы рассмотрим ещё один показатель — «ожидаемые потери» (Expected Shortfall, ES). Он определяется как матожидание убытков, превышающих VaR:

.

Для расчёта значений этих показателей используем метод Монте-Карло. Он состоит в том, что по известному модельному закону распределения генерируется большое число случайных реализаций исследуемой величины. Поскольку количество реализаций большое, определив интересующие нас характеристики этой выборки, мы без значительных потерь точности можем перенести их на всю генеральную совокупность (например, найдя среднее значение, приравнять его к матожиданию, а рассчитанный эмпирический квантиль — к соответствующему теоретическому). Таким образом, «границе потерь» будет соответствовать эмпирический квантиль модельной выборки, а «ожидаемым потерям» — среднее значение всех элементов модельного вектора, меньших этого квантиля.

N <- 10^6

dax.sim <- rghyp(n=N,object=aic.uv$best.model)

dax.sim <- sort(dax.sim)

VaR <- dax.sim[(1-alpha)\*N]

[1] -0.01594284

ES <- mean(dax.sim[1:((1-alpha)\*N-1)])

[1] -0.02262264

Значение границы потерь также можно найти, используя квантильную функцию.

VaR <- qghyp(1-alpha,object=aic.uv$best.model)

[1] -0.01594899

«Граница потерь», полученная методом Монте-Карло будет тем ближе к результату по квантильной функции, чем более велико **N**.

## 4. Многомерный случай

В многомерном случае используется практически тот же набор функций, что и для построения одномерных моделей. Составим матрицу активов, по столбцам которой будут располагаться доходности биржевых индексов.

smi <- EuStockMarkets[,"SMI"]

smi <- smi[2:(T+1)]/smi[1:T] - 1

prt <- cbind(dax,smi)

head(prt,n=2)

dax smi

[1,] -0.009283193 0.006197485

[2,] -0.004412412 -0.005863192

Для формирования матрицы мы использовали функцию cbind, которая составляет из набора векторов матрицу, в которой эти вектора расположены по столбцам. Существует также функция rbind, составляющая матрицу из векторов-строк.

Когда данные организованы таким образом, как это показано в примере, они пригодны для подгонки многомерного ОГР, общая схема которой представлена ниже.

prt.fit <- fit.[…]mv(prt,symmetric=FALSE,silent=TRUE)

Отличие от одномерного случая состоит в том, что названия функций для подгонки оканчиваются на «mv», а не на «uv» (и, разумеется, в формате исходных данных).

Квантиль многомерного распределения представляет собой не точку, а целую линию уровня, поэтому мы не можем использовать его для расчёта мер риска, однако нам по-прежнему доступен метод Монте-Карло.

Набор случайных реализаций многомерной величины представляет собой матрицу, количество столбцов которой равно размерности модели. В нашем случае оно будет равно двум.

prt.fit <- fit.ghypmv(prt,symmetric=FALSE,silent=TRUE)

sim <- rghyp(n=N,object=prt.fit)

Чтобы получить из матрицы **sim** вектор модельных доходностей портфеля, необходимо каждому активу задать вес, с которым он входит в портфель.

w <- c(0.5,0.5)

Теперь мы можем рассчитать модельные доходности как средневзвешенное столбцов матрицы **sim** и далее, по аналогии с одномерным случаем, найти оценки риска.

prt.sim <- w[1]\*sim[,1]+w[2]\*sim[,2]

prt.sim <- sort(prt.sim)

VaR <- prt.sim[(1-alpha)\*N]

[1] -0.01420918

ES <- mean(prt.sim[1:((1-alpha)\*N-1)])

[1] -0.02290383

## 5. Оптимизация портфеля

При расчёте измерителей риска веса активов в портфеле были нами заданы произвольно, однако можно поставить задачу нахождения таких весов, при которых какая-либо целевая характеристика портфеля (например, значение «границы потерь») была бы оптимальна. Решать подобные задачи позволяет функция portfolio.optimize.

opt <- portfolio.optimize(prt.fit,risk.measure="value.at.risk",

type="minimum.risk",target.return=NULL,risk.free=NULL,level=0.95,

method="BFGS",silent=TRUE)

В эту функцию подставляется оцененная многомерная ОГР-модель, а затем с помощью набора дополнительных параметров специфицируется конкретная оптимизационная задача.

Параметр **type** определяет вид этой задачи. Значению «minimum.risk» соответствует минимизация значения одного из измерителей риска, указанного в параметре **risk.measure**. Доступные показатели: «граница потерь» («value.at.risk»), «ожидаемые потери» («expected.shortfall») и стандартное отклонение («sd»). Значению «tangency» параметра **type** соответствует максимизация коэффициента Шарпа (очищенная от безрисковой ставки доходность, делённая на риск). Для решения этой задачи в параметре **risk.free** должна быть указана безрисковая ставка доходности, а также определён измеритель риска **risk.measure**. Наконец, при значении «target.return» производится минимизация заданного показателя риска при определённой в параметре **target.return** фиксированной доходности.

Параметр **level** задаёт уровень значимости при расчёте VaR и ES, параметр **method**  определяет алгоритм оптимизации. Наиболее часто используемые его значения — «BFGS» для квази-Ньютоновского метода, который работает сравнительно быстро, но может принять локальный экстремум за глобальный, и «SANN» для метода глобальной оптимизации, который, однако, работает гораздо медленнее.

Переменная **opt**, в которую был записан результат оптимизации, имеет тип «список». Её компонента, называемая **opt.weights** содержит оптимальные в смысле целевого критерия веса активов.

opt$opt.weights

[1] 0.3914557 0.6085443

Приведённый результат означает, что минимальной «границей потерь», определённой на 95%-м уровне значимости, обладает портфель, на 39.1% состоящий из германского индекса DAX и на 60.9% — из швейцарского SMI.

## 6. Домашнее задание

Рассмотреть данные о доходности одного из биржевых индексов.

* построить модель на основе ОГР и получить с помощью неё оценку «границы потерь»;
* обосновать выбор распределения из класса ОГР;
* обосновать качество полученной оценки путём построения кривой VaR и проведения теста Купика.

# Раздел 6. Выделение цикла, тренда и сезонности временного ряда.

В предыдущем разделе обсуждались некоторые статистические особенности финансовых временных рядов. Для многих микро- и макроэкономических показателей характерно также наличие нескольких компонент, определяющих их динамику. Обычно выделяют трендовую, сезонную и остаточную компоненты. Влияние этих компонент может объединяться как мультипликативно, так и аддитивно. В первом случае имеем:

,

где — исследуемый показатель, — циклическая составляющая, — трендовая составляющая, — сезонная компонента, — остаточная (нерегулярная) компонента. От мультипликативной модели путём логарифмирования мы можем перейти к аддитивной:

,

.

Подобный переход сопряжён с дополнительными корректировками, рассмотрение которых, однако, находится за рамками этого курса. В дальнейшем при рассмотрении декомпозиции временного ряда мы будем иметь в виду аддитивную модель.

## 1. Выделение сезонной и остаточной компонент

Основная идея методов разделения исходного ряда на набор различных компонент состоит в последовательном применении сглаживающих фильтров. В рамках процедуры выделения сезонности (Seasonal-Trend Loess decomposition, STL) важную роль играет метод сглаживания «loess», который мы коротко рассмотрим далее.

По сути, этот метод представляет собой нахождение взвешенного среднего исходного ряда по полиному некоторой независимой переменной, в качестве которой может выступать другой экономический показатель или время:

.

Для расчёта сглаженного значения в каждой конкретной точке *x* подгоняется приведённая выше регрессия по всему массиву данных , где каждому наблюдению присваивается вес в зависимости от его расположения относительно точки *x*:

,

— расстояние от *x* до *q*-го ближайшего *xt*.

Ближайшие к точке *x* наблюдения получают наибольшие веса, а наблюдениям, отстоящим дальше присваивается нулевой вес. Чем больше *q*, тем более гладкая оценка ; при *q* → +∞ метод «loess» эквивалентен обычной полиномиальной МНК-регрессии. Степень полинома выбирается, исходя из вариативности показателя : единица для гладкой динамики и двойка — для сложных кривых. Если наблюдениям придаются разные веса (например, ), тогда веса для процедуры «loess» рассчитываются как .

Для проведения процедуры STL из исходного ряда выделяются *n(p)* сезонных подпоследовательностей (например, январские данные, февральские данные, …, декабрьские данные).

Сама процедура STL состоит из *n(o)* итераций внешнего цикла и *n(i)* итераций внутреннего. Во внутреннем цикле обновляются значения трендовой и сезонной компонент, во внешнем после выполнения внутреннего рассчитываются робастные веса.

Допустим, на *k-*м шаге внутреннего цикла мы имеем и — промежуточные оценки сезонной и трендовой компонент. Рассмотрим процесс получения оценок и .

1. Из исходного ряда удаляется тренд: рассчитывается ряд ;
2. Каждая из сезонных подпоследовательностей сглаживается методом «loess» при *q = n(s)* и *d =* 1. Сглаженные значения рассчитываются для всех выборочных периодов времени, плюс к этому — для значений, предшествующего первому и следующему за последним. Например, если подпоследовательность включает в себя январские данные с 2000-го по 2010-й год, то сглаженные значения рассчитываются также для января 1999-го и января 2011-го. Таким образом, общее количество сглаженных значений для всех подпоследовательностей составляет *T +* 2*n(p)*;
3. Из объединённой совокупности сезонных подпоследовательностей выделяется остаточная трендовая компонента путём последовательного применения серии фильтров: скользящего среднего с окном *n(p)*, такого же скользящего среднего, скользящего среднего с окном 3 и метода «loess» при *q = n­(l)* и *d =* 1. Полученный трендовый ряд будет иметь длину *T* исходного ряда, поскольку при применении скользящих средних часть наблюдений на краях теряется;
4. Рассчитываются значения как разность между сезонными значениями, полученными на шаге 2., и трендовыми сериями, полученными на шаге 3.;
5. Из исходного ряда удаляется сезонность: рассчитывается ряд ;
6. Рассчитываются новые значения для трендовой компоненты путём применения метода «loess» с *q = n(t)* и *d =* 1 к очищенному от сезонности ряду.

Для выполнения пункта 1. при самом первом прохождении внутреннего цикла задаётся начальное значение трендовой компоненты .

При обновлении значений трендовой и сезонной компонент несколько раз использовался метод «loess» с разным набором параметров. Существуют рекомендации по выбору их значений, которые были получены в ходе эмпирических наблюдений:

* — наименьшее нечётное число, большее или равное ;
* зависит от соотношения дисперсий сезонной и остаточной компонент; чем больше , тем меньше дисперсия сезонности;
* — наименьшее нечётное число, удовлетворяющее условию .

Во внешнем цикле рассчитывается остаточная компонента и на основе её определяются робастные веса:

.

Веса определены так, что большим по абсолютной величине остаткам соответствует меньший вес. Это позволяет получать более гладкие оценки значений компонент.

После расчёта робастных весов, имеющих единичное начальное значение, внутренний цикл повторяется очередной раз, однако в методе «loess» при этом используются веса .

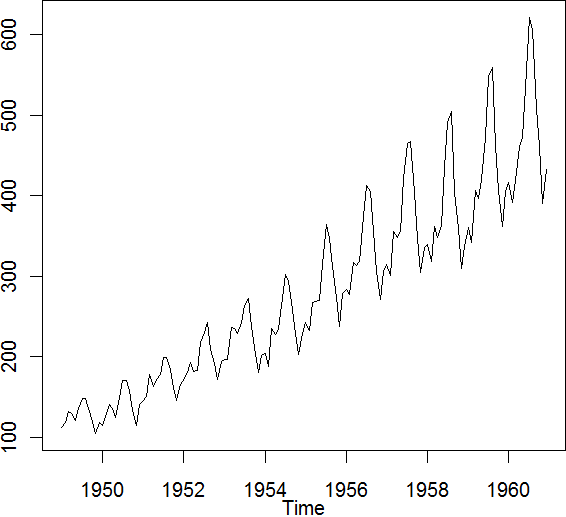
Если ряд исходных данных не содержит значительных выбросов, то от использования внешнего цикла обычно отказываются, а поскольку алгоритм внутреннего цикла обладает очень быстрой сходимостью, рекомендуемые значения параметров процедуры в этом случае *n(o) =* 0 и *n(i)* = 2. В противном случае, когда требуется переопределять робастные веса для сглаживания выбросов, рекомендуется использовать *n(o) =* 15 и *n(i)* = 1.

Для практической иллюстрации применения процедуры STL рассмотрим помесячные данные о количестве пассажиров, перевезённых американскими авиалиниями в 1949–1960 годах в тысячах человек.

library(datasets)

y <- AirPassengers

На рис. 29 представлен график этого показателя.



*Рисунок 29. График количества перевезённых пассажиров.*

На этом графике отчётливо видны структурные компоненты временного ряда. Во-первых, это сезонная компонента: перелёты в середине года происходят более активно, чем вначале и в конце. Хорошо заметен тренд и его нелинейность, свидетельствующая о возможном наличии цикла.

Процедура STL реализуется в R с помощью одноимённой функции из библиотеки stats.

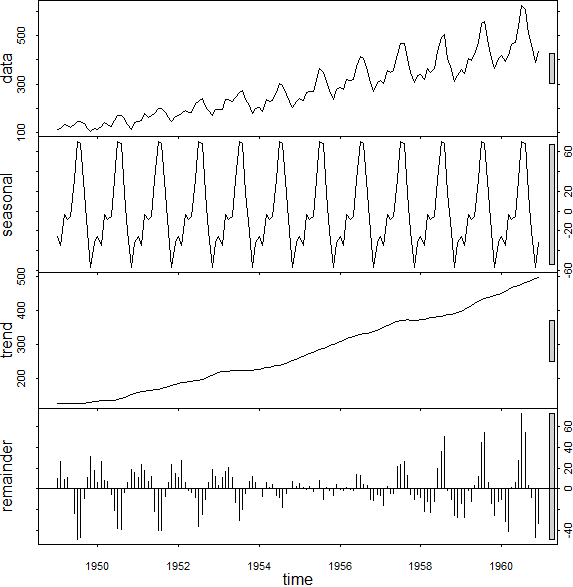
library(stats)

stl.y <- stl(y,s.window="periodic",robust=FALSE)

Двумя её параметрами, для которых не существует чётких формальных рекомендаций, являются **s.window** (*n(s)*) и **robust** (*n(o)*). Ширину окна для сглаживания сезонных подпоследовательностей можно задать как числом, так и текстовым значением «periodic», как в примере выше. В этом случае метод «loess» без значительных потерь точности заменяется расчётом среднего. Логический параметр **robust** определяет, нужно ли включать в процедуру внешний цикл. Поскольку динамика рассматриваемого нами ряда достаточно гладкая, в этом нет необходимости.

В итоге функция stl разделяет исходный ряд на три составляющие (см. рис. 30): сезонную (в наших обозначениях — **stl.y$time.series[,"seasonal"]**), трендовую (**stl.y$time.series[,"trend"]**) и остаточную (**stl.y$time.series[,"remainder"]**).

plot(stl.y)



*Рисунок 30. Сезонная, трендовая и остаточная компоненты временного ряда.*

При этом трендовая компонента явно содержит в себе цикл, т.е., фактически, представляет собой сумму двух составляющих .

## 2. Разделение цикла и тренда

Для решения задачи отделения цикла от тренда используется фильтр Ходрика–Прескотта. Он основан на решении следующей оптимизационной задачи.

Пусть рассматриваемый ряд состоит из двух компонент (трендовой и циклической): , тогда трендовую компоненту мы можем выделить, рассмотрев задачу минимизации.

Первая сумма минимизирует отклонение тренда от фактических значений, вторая — вариацию тренда. Фактически, трендовая компонента должна достаточно точно аппроксимировать исходный ряд, но при этом быть достаточно гладкой. Степень гладкости тренда регулируется параметром *λ*: чем он больше, тем больше штраф за вариацию и тем более гладок тренд. Существуют эмпирические рекомендации по выбору конкретного значения сглаживающего параметра в зависимости от периодичности ряда. Так, для годичных данных *λ* следует устанавливать равной 6.25, для квартальных — 1 600, а для месячных — 129 600.

Функция, отвечающая за этот тест, — hpfilter из библиотеки mFilter.

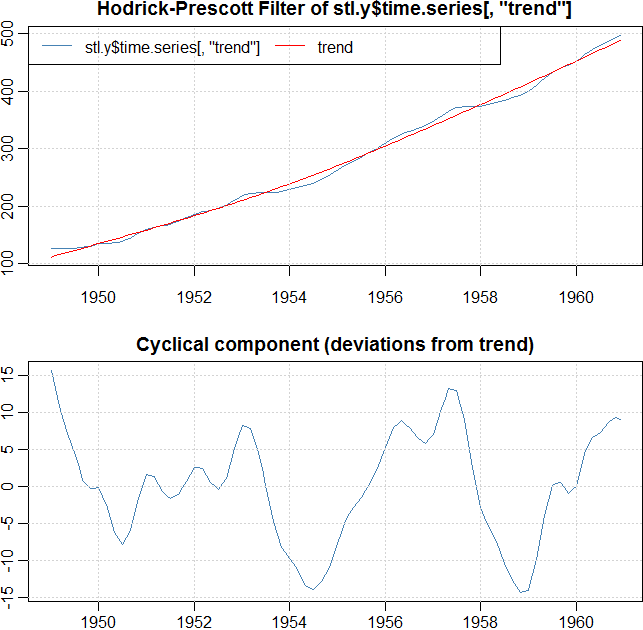
library(mFilter)

hp.y <- hpfilter(stl.y$time.series[,"trend"],freq=129600)

В ней кроме анализируемого ряда, в качестве которого в нашем случае выступает трендовая компонента из процедуры STL, в параметре **freq** следует задать либо численное значение *λ*, либо значение NULL для автоматического подбора сглаживающего параметра, исходя из формата переменной, в которой содержится временной ряд.

Функция возвращает трендовую и циклическую компоненты (см. рис. 31), которые в нашем случае обозначаются как **hp.y$trend** и **hp.y$cycle**.

plot(hp.y)

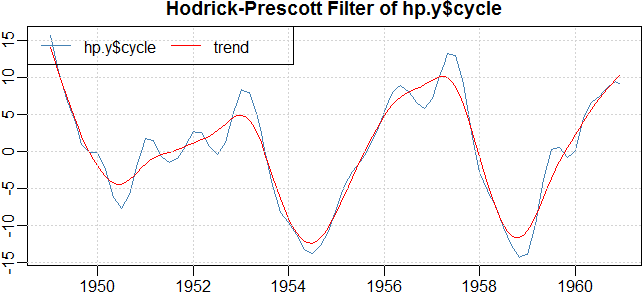


*Рисунок 31. Трендовая и циклическая компоненты временного ряда.*

Бывают случаи, как на рисунке выше, когда циклическая компонента подвержена мелким колебаниям, искажающим общую картину, поэтому у исследователя может возникнуть желание её сгладить. Для этого рекомендуется вновь использовать фильтр Ходрика–Прескотта, на этот раз с малым значением *λ*, и взять в качестве цикла трендовую составляющую результата.

y.cycle <- hpfilter(hp.y$cycle,freq=150)$trend

На рис. 32 новый, сглаженный, цикл показан красным цветом.



*Рисунок 32. Сглаженная циклическая компонента.*

## 3. Определение поворотных точек цикла

После выделения циклической компоненты из экономического показателя бывает интересно рассмотреть поворотные точки цикла, т.е. те точки, в которых падение сменяется ростом и наоборот. С более формальной точки зрения, можно сказать, что если *A* —множество поворотных точек цикла ,тогда оно состоит из тех элементов циклической составляющей, в которых её прирост меняет знак:

.

Определим логический вектор **tp** длиной, равной длине цикла , значения TRUE которого будут соответствовать поворотным точкам.

T <- length(y.cycle)

tp <- logical(T)

Пробежав циклом «for» по всем элементам вектора **y.cycle**, кроме первых двух, для которых не определена вторая разность, и сравнив знаки приростов, найдём номера элементов, являющихся поворотными точками.

for (i in 3:T) tp[i] <- (sign(y.cycle[i]-y.cycle[i-1])!=

sign(y.cycle[i-1]-y.cycle[i-2]))

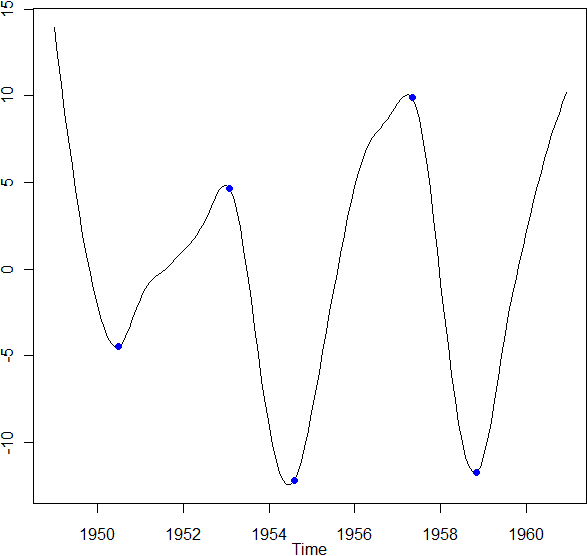
Для удобства графического отображения значения FALSE вектора **tp** заменим пропущенными значениями NA.

tp[!tp] <- NA

На рис. 33 представлено графическое изображение поворотных точек циклической компоненты.

plot(y.cycle)

points(tp\*y.cycle,pch=16,col="blue")



*Рисунок 33. Поворотные точки цикла.*

Поворотные точки используются, среди прочего, при построении опережающих индикаторов, которым будет посвящено домашнее задание.

## 4. Домашнее задание

* рассмотреть динамику цен газового и нефтяного индексов с сайта <http://siteresources.worldbank.org/INTPROSPECTS/334934-1304428586133/PINK_DATA.xls>;
* по поворотным точкам их циклов определить, является ли один показатель опережающим индикатором для другого, и рассчитать величину опережения.

1. Здесь и далее выполняемые команды обозначаются красным цветом, а результат их выполнения — синим. [↑](#footnote-ref-1)
2. Мнения расходятся. [↑](#footnote-ref-2)
3. Здесь и далее функция print для вывода значений на экран опускается. [↑](#footnote-ref-3)
4. http://java.com/en/download/ [↑](#footnote-ref-4)
5. Этот тест также известен как «U-критерий Манна–Уитни» и «критерий Уилкоксона–Манна–Уитни». [↑](#footnote-ref-5)
6. Метод наименьших квадратов. [↑](#footnote-ref-6)
7. Множитель 104 был введен нами для удобства численной записи значений этой функции. [↑](#footnote-ref-7)
8. Пусть  — вектор наблюдаемых значений случайной величины,  — модель плотности её распределения, тогда значение функции правдоподобия для этой модели равно  [↑](#footnote-ref-8)