RUDN university

5100

Программирование в среде R

Шевцов Василий Викторович,
директор ДИТ РУДН, shevtsov-vv@rudn.ru

Классы данные (переменных)

- **numeric** — название класса, а также типа объектов. К нему относятся действительные числа. Объекты данного класса делятся на
  - целочисленные (integer)
  - действительные (double или real).
- **complex** — объекты комплексного типа.
- **logical** — логические объекты, принимают только два значения:
  - FALSE (F)
  - TRUE (T)
- **character** — символьные объекты. символьные переменные задаются либо в двойных кавычках, либо в одинарных.
- **raw** — объекты потокового типа

Иерархия типов: raw < logical < integer < real < complex < character.

Numeric

Объект класса **numeric** создаётся при помощи команды **numeric(n)**, где n — количество элементов данного типа.
Создаётся нулевой вектор длины n.
>x=numeric(5)
> x
> [1] 0 0 0 0 0
В результате создан нулевой вектор типа numeric длины 5.

Тип	Создается	Проверяется
numeric	numeric(n)	is.numeric(имя_объекта) → TRUE/FALSE
integer	integer(n)	is.integer(имя_объекта) → TRUE/FALSE
double	double(n)	is.double(имя_объекта) → TRUE/FALSE

Десятичным разделителем для чисел является точка

Numeric

```
> x<-double(1)
> x<-5
> y<-integer(1)
> y<-7
> is.integer(x)
[1] FALSE
> is.double(x)
[1] TRUE
> is.integer(y)
[1] FALSE
> is.double(y)
[1] TRUE
> is.numeric(x)
[1] TRUE
> is.double(y)
[1] TRUE
> y<-integer(1)
> is.integer(y)
[1] TRUE
>
```

1. Создали переменную
2. Присвоили значение
3. Проверили принадлежность

По умолчанию, все числа в R являются вещественными. Чтобы сделать их целочисленными, надо воспользоваться командой **as.integer(имя_объекта)**

Numeric

```
>
> x<-double(1)
> x<-5
> is.double(x)
[1] TRUE
> is.integer(x)
[1] FALSE
> x<-as.integer(x)
> is.double(x)
[1] FALSE
> is.integer(x)
[1] TRUE
>
```

1. Создали переменную
2. Присвоили значение
3. Проверили принадлежность
4. Назначили тип данных
5. Проверили принадлежность

Logical

- Объекты этого класса принимают два возможных значения: TRUE (истина) и FALSE (ложь). Сокращенные наименования: T, F
- создаются при помощи команды **logical(n)**, где n — это длина создаваемого вектора.

```
> x<-logical(4)
> x
[1] FALSE FALSE FALSE FALSE
> x<-1;y<-F
> is.logical(x)
[1] FALSE
> is.logical(y)
[1] TRUE
>
```

Проверка типа данных – **is.logical(x)**

Logical

- преобразование данных

```
> x<-1
> is.double(x)
[1] TRUE
> z<-as.logical(x)
> is.logical(z)
[1] TRUE
> z
[1] TRUE
> x<-0
> z<-as.logical(x)
> z
[1] FALSE
>
```

- присвоение числового значения (double)
- преобразование в logical и присвоение значения
- проверка типа данных

Character

- **character** - символьные объекты
- Создаются при помощи команды **character(n)**, результат — пустой символьный вектор размерности n
- Символьные объекты обязательно задаются в кавычках (одинарных или двойных)
- Символьным объектом может быть как просто символ, так и строка.

```
> x<-'q'
> x
[1] "q"
> x<-"w"
> x
[1] "w"
> x<-"'r'"
> x
[1] "'r'"
> x<-"язык программирования R"
> x
[1] "язык программирования R"
> 
```

```
> x<-"qqq";x
[1] "qqq"
> x<-"'qqq'";x
[1] "'qqq'"
> x<-""qqq"";x
Error: unexpected symbol in "x<-""qqq"
> x<-'"qqq"';x
[1] "\"qqq\""
> x<-"\"qqq\"";x
[1] "\"qqq\""
> 
```

Преобразование в character

- Объекты любого типа можно перевести в символьные. Для этого нужно воспользоваться командой **as.character(имя_объекта)**

```
> x<-F;y<-1.23
> x;y
[1] FALSE
[1] 1.23
> z<-as.character(x);x
[1] FALSE
> z
[1] "FALSE"
> z<-as.character(y);y
[1] 1.23
> z
[1] "1.23"
>
```

```
> x<-0123;x
[1] 123
> y<-as.character(x);y
[1] "123"
> y<-as.character("0123");y
[1] "0123"
>
```

Преобразование character

- Символьный объект можно перевести в числовой, если он представляет из себя число, окружённое кавычками.
- Если же в кавычках стоял непосредственно символ (или набор символов), то такой перевод приведёт к появлению NA (Not Available)

```
> x<-"123"
> y<-as.numeric(x)
> y
[1] 123
> x<-"0123"
> y<-as.numeric(x)
> y
[1] 123
> x<-"q123"
> y<-as.numeric(x)
Warning message:
NAs introduced by coercion
> 
```

Преобразования

- Тип любого объекта можно проверить (и изменить) при помощи функции **mode(имя_объекта)**

```
> x<-as.character("T")
> y<-as.logical(x)
> y
[1] TRUE
> z<-as.integer(y)
> z
[1] 1
> mode(x)
[1] "character"
> mode(y)
[1] "logical"
> mode(z)
[1] "numeric"
>
```

```
> x<-as.character("T")
> x
[1] "T"
> mode(x)<-'logical'
> x
[1] TRUE
> x<-as.character("Q")
> x
[1] "Q"
> mode(x)<-'logical'
> x
[1] NA
>
```

Специальные переменные в R

- **Inf**
  - бесконечность: положительная (+∞ — Inf) и отрицательная (−∞ -Inf);
- **NA**
  - отсутствующее значение (Not Available);
- **NaN**
  - не число (Not a Number);
- **NULL**
  - ничто

Специальные переменные в R

- **Inf** появляется при переполнении и в результате операций вида a/0, где a <>0
- Проверить объект на конечность / бесконечность можно при помощи команд **is.finite()** / **is.infinite()**

```
> x<-0;y<-1
> z<-y/x
> z
[1] Inf
> z<-log(x)
> z
[1] -Inf
>
```

```
> x<-0;y<-1
> z<-y/x
> is.finite(z)
[1] FALSE
> is.infinite(z)
[1] TRUE
>
```

Специальные переменные в R

- Объект **NaN** — «не число», появляется при операциях над числами, результат которых не определён (не является числом)
- При помощи **is.nan(имя_объекта)** можно проверить, является ли объект NaN:

```
> x<-0;y<-0
> z<-x/y
> z
[1] NaN
> z<-Inf-Inf
> z
[1] NaN
> x<-Inf;y<-Inf
> z<-x-y
> z
[1] NaN
> x<--2
> z<-log(x);z
Warning message:
In log(x) : NaNs produced
[1] NaN
>
```

```
> x<-Inf;y<-Inf
> z<-x-y
> is.nan(z)
[1] TRUE
>
```

Специальные переменные в R

- Отсутствующее значение **NA** возникает, если значение некоторого объекта не доступно (не задано). Включает в себя и **NaN**.
- Проверка,относится ли объект к **NA**, делается при помощи **is.na(имя_объекта)**

```
> x<-NaN
> y<-NA
> is.na(x)
[1] TRUE
> is.na(y)
[1] TRUE
> 
```

Специальные переменные в R

- Ничто **NULL** нулевой (пустой) объект. Возникает как результат выражений (функций), чьи значения не определены. Обнулить объект можно при помощи команды **as.null(имя_объекта)**
- Проверить объект на принадлежность к **NULL** можно при помощи функции **is.null(имя_объекта)**

```
> x<-1
> y<-as.null(x)
> is.null(x)
[1] FALSE
> is.null(y)
[1] TRUE
>
```

Представление даты и времени, временные ряды

Особенности даты и времени

- разные годы начинаются в разные дни недели
- високосные годы имеют дополнительный день в феврале
- американцы и европейцы по разному представляют даты (например, 8/9/2011 будет 9-м августа 2011 г. для первых и 8-м сентября 2011 г. для вторых)
- в некоторые годы добавляется так называемая "секунда координации"
  - Секунда координации, или високосная секунда - дополнительная секунда, добавляемая ко всемирному координированному времени для согласования его со средним солнечным временем UT1
- страны различаются по временным поясам и в ряде случаев применяют переход на "зимнее" и "летнее" время

Форматы представления даты и времени

```
> Sys.time()
[1] "2018-02-28 21:44:34 MSK"
> sys.time()
Error in sys.time() : could not find function "sys.time"
```

```
> substr(as.character(Sys.time()),1,10)
[1] "2018-02-28"
> substr(as.character(Sys.time()),12,19)
[1] "21:47:21"
> date()
[1] "Wed Feb 28 21:47:30 2018"
> unclass(Sys.time())
[1] 1519843681
```

Последней операцией получаем время в формате **POSIXct**, т.е. выраженное в секундах, прошедших с 1 января 1970 г. (его еще трактуют как Unix-время, по названию операционной системы). Такой "машинный" формат удобен для включения в таблицы данных.

Форматы представления даты и времени

Для человека более удобным является представление времени в формате класса **POSIXlt**. Объекты этого класса представляют собой списки, включающие такие элементы, как секунды, минуты, часы, дни, месяцы, и годы.
Например, мы можем конвертировать системное время в объект **POSIXlt** класса следующим образом:

```
> dt<-as.POSIXlt(Sys.time())
> dt
[1] "2018-02-28 21:57:51 MSK"
> dt$sec
[1] 51.01744
> dt$min
[1] 57
> dt$hour
[1] 21
> dt$mday
[1] 28
> dt$mon
[1] 1
> dt$year
[1] 118
> dt$wday
[1] 3
> dt$yday
[1] 58
> dt$isdst
[1] 0
```

sec (секунды), min (минуты), hour (часы), mday (день месяца), mon (месяц), year (год), wday (день недели, начиная с воскресенья = 0), yday (день года, начиная с 1 января = 0), isdst ("*is daylight savings time in operation?*" – логическая переменная, обозначающая, используется ли режим перехода на "зимнее" и "летнее" время: 1 если TRUE и 0 если FALSE)

Форматы представления даты и времени

- Для просмотра всего содержимого списка date можно использовать функцию **unclass()** в сочетании с **unlist()**

```
> unlist(unclass(dt))
              sec               min              hour              mday               mon              year
"51.0174360275269"              "57"              "21"              "28"               "1"             "118"
             wday              yday             isdst              zone            gmtoff
              "3"              "58"               "0"             "MSK"           "10800"
> dt1<-unlist(unclass(dt))
> dt1
              sec               min              hour              mday               mon              year
"51.0174360275269"              "57"              "21"              "28"               "1"             "118"
             wday              yday             isdst              zone            gmtoff
              "3"              "58"               "0"             "MSK"           "10800"
> dt1[2]
 min
"57"
> dt1["min"]
 min
"57"
```

Вычисления с датами и временем

- В R можно выполнять следующие типы вычислительных операций с датами и временем:
  - число + время;
  - время – число;
  - время1 – время2
  - время1 "логический оператор" время2 (в качестве логического оператора могут использоваться ==, !=, <=, <, > или >=).
- Важной особенностью является то, что перед выполнением любых вычислений с датами или временем необходимо конвертировать их в объекты класса POSIXlt

Вычисления с датами и временем

```
> d1<-as.POSIXlt("2018-02-28")
> d2<-as.POSIXlt("2010-01-30")
> d1-d2
Time difference of 2951 days
> difftime(d1,d2)
Time difference of 2951 days
```

```
> t1<-as.POSIXlt("2018-02-28 22:21:30")
> t2<-as.POSIXlt("2010-01-30 08:10:45")
> t1-t2
Time difference of 2951.591 days
> t3<-as.POSIXlt("2018-02-28 08:10:45")
> t1-t3
Time difference of 14.17917 hours
> t3-t1
Time difference of -14.17917 hours
```

```
> as.numeric(difftime(t3,t1))
[1] -14.17917
> as.numeric(difftime(t1,t3))
[1] 14.17917
```

Вычисления с датами и временем

R отсутствует возможность для сложения двух дат.

```
> t1+t2
Error in `+.POSIXt`(t1, t2) :
  бинарный '+' не определен для объектов "POSIXt"
```

Извлечение даты/времени из текстовых переменных

- Функция strptime() (от *strip* – раздевать, оголять, и *time* – время) позволяет извлекать даты и время из различных текстовых выражений.
- При этом важно верно указать формат (при помощи аргумента format), в котором приведены временные величины.
- Приняты следующие условные обозначения для наиболее часто используемых форматов дат и времени

Извлечение даты/времени из текстовых переменных

%a – сокращенное название для недели
%A – полное название для недели
%b – сокращенное название месяца
%B – полное название месяца
%d – день месяца (01–31)
%H – часы от 00 до 23
%I – часы от 01 до 12
%j – порядковый номер дня года (001–366)
%m – порядковый номер месяца (01–12)
%M – минуты (00–59)
%S – секунды (00–61, с возможностью добавить "високосную секунду")
%U – неделя года (00–53), первое воскресенье считается первым днем первой недели
%w – порядковый номер дня недели (0–6, воскресенье – 0)
%W – неделя года (00–53), первый понедельник считается первым днем первой недели
%Y – год с указанием века
%y – год без указания века

Извлечение даты/времени из текстовых переменных

```
> strptime("28-02-2018",format = "%d/%m/%Y")
[1] NA
> strptime("28/02/2018",format = "%d/%m/%Y")
[1] "2018-02-28 MSK"
> strptime("28-02-2018",format = "%d-%m-%Y")
[1] "2018-02-28 MSK"
> strptime("02.28.2018",format = "%m.%d.%Y")
[1] "2018-02-28 MSK"
```

```
> strptime(c("02.28.2018","01.30.2018","05.20.1970"),format = "%m.%d.%Y")
[1] "2018-02-28 MSK" "2018-01-30 MSK" "1970-05-20 MSK"
> dates<-c("02.28.2018","01.30.2018","05.20.1970")
> strptime(dates,format = "%m.%d.%Y")
[1] "2018-02-28 MSK" "2018-01-30 MSK" "1970-05-20 MSK"
```

```
> dates1<-c("1янв79", "02фев99", "31мар04", "30авг05")
> strptime(dates1,format = "%d%b%y")
[1] "1979-01-01 MSK" "1999-02-02 MSK" "2004-03-31 MSD" "2005-08-30 MSD"
```

Решение нелинейных уравнений и систем нелинейных уравнений. Интегрирование и дифференцирование.

Функция uniroot

- uniroot(f, interval, ..., lower = min(interval), upper = max(interval), f.lower = f(lower, ...), f.upper = f(upper, ...), tol = .Machine$double.eps^0.25, maxiter = 1000)

Решение считается найденным, либо если значение функции в найденной точке x^* равняется нулю ($f(x^*) == 0$), либо если изменение значения x^* на следующей итерации меньше заданной точности tol. Если достигнут максимум итераций, а решение не найдено, то выдаётся предупреждение.

Результатом функции uniroot является список из четырёх компонент: искомое решение x^* — $root$, значение функции в найденной точке $f(x^*)$ — $f.root$, число итераций $iter$ и точность решения $estim.prec$. Если x^* совпадает с одним из концов заданного интервала поиска, то тогда значение $estim.prec$ — NA.

Существенный недостаток функции uniroot — это то, что ищется только одно решение на заданном интервале. Если существует несколько нулей функции, то будет выводиться только первый найденный.

Функция uniroot

Аргументы:

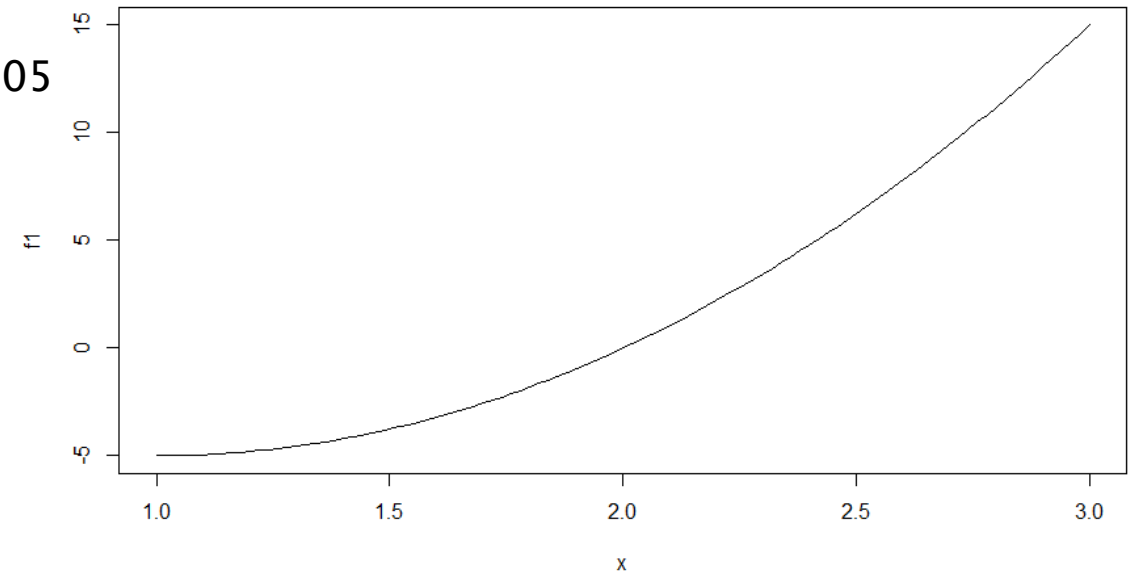
- f — функция, нуль которой (т.е. корень) вычисляется. Отметим, что нуль функции f ищется только по её первому аргументу.
- interval — числовой вектор — интервал, на котором ищется корень (необходимо, чтобы значения функции на концах этого интервала имели разные знаки).
- lower и upper — **альтернативное** задание интервала поиска interval через его начало и конец.
- f.lower и f.upper — граничные значения функции **(по умолчанию значения функции f на границах интервала поиска)**.
- tol — желаемая точность.
- maxiter — максимальное число итераций.
- ... — дополнительные аргументы.

Функция uniroot

$$5x^2 - 10x = 0$$

```
f1 <- function(x){y <- 5*x^2-10*x;return(y)}
x1 <- 1
x2 <- 3
plot(f1,x1,x2)
uniroot(f1,c(x1,x2))
```

```
$root [1] 2
$f.root [1] -2.678252e-06
$iter [1] 6
$init.it [1] NA
$estim.prec [1] 6.535148e-05
```

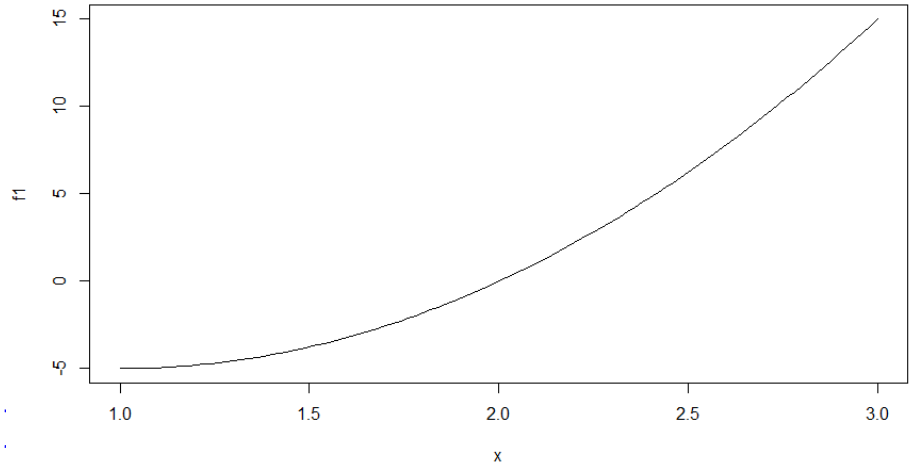


Функция uniroot

```
решение функции
$root [1] 2

значение функции в найденной точке
$f.root [1] -2.678252e-06

число итераций
$iter [1] 6

точность решения
$estim.prec [1] 6.535148e-05
```

Функция uniroot

```
> uniroot(f1,interval=c(x1,x2))

$root [1] 2
```



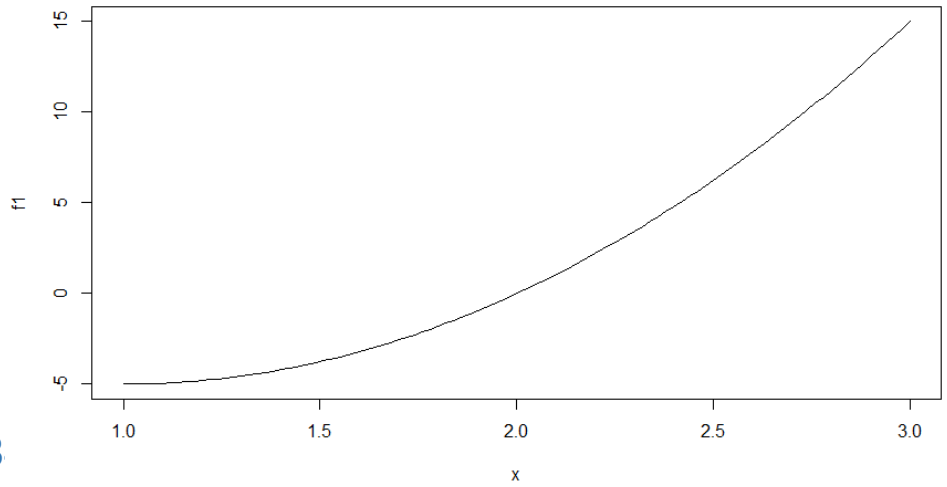

```
> uniroot(f1,lower = x1, upper = x2

$root [1] 2
```

```
> uniroot(f1,interval=c(x1,x2),f.lower = -2, f.upper = 5)
$root [1] 2
```

Функция uniroot

f.lower и f.upper — граничные значения функции (по умолчанию значения функции f на границах интервала поиска).

```
> f1 <- function(x){y <- 5*x^2-10*x;return(y)}
> x1 <- 1
> x2 <- 3
> plot(f1,x1,x2)
> uniroot(f1,interval=c(x1,x2))
$root [1] 2
```

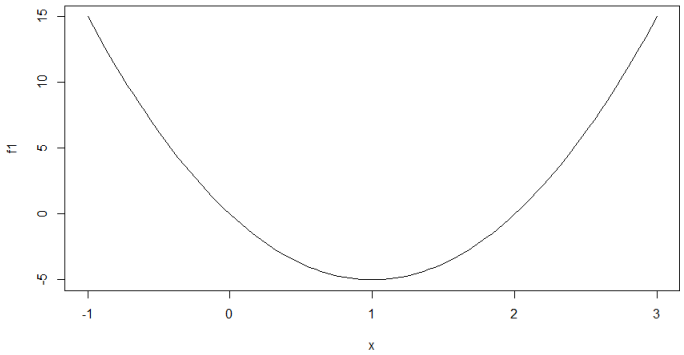


Функция uniroot

f.lower и f.upper — граничные значения функции (по умолчанию значения функции f на границах интервала поиска).

```
> f1 <- function(x){y <- 5*x^2-10*x;return(y)}
> x1 <- -1 > x2 <- 3
> plot(f1,x1,x2)
> uniroot(f1,interval=c(x1,x2),f.lower = 15, f.upper = 15)
Error in uniroot(f1, interval = c(x1, x2), f.lower = 15, f.upper = 15) :
f() значений на концевых точках не противоположного знака
```

```
> uniroot(f1,interval=c(x1,x2),f.lower = -5, f.upper = 15)
$root [1] 0
```



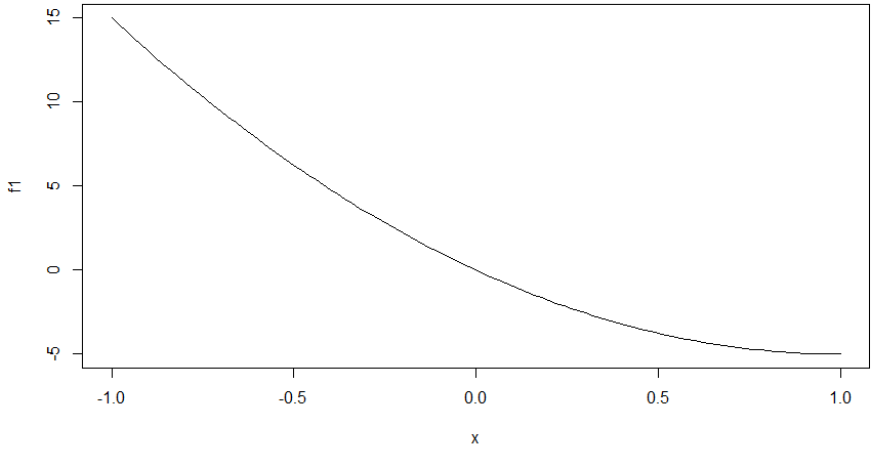

```
> f1 <- function(x){y <- 5*x^2-10*x;return(y)}
> x1 <- -1
> x2 <- 3
> plot(f1,x1,x2)
> uniroot(f1,interval=c(x1,x2))
Error in uniroot(f1, interval = c(x1, x2)) :
f() значений на концевых точках не противоположного знака
```

Функция uniroot

```
> f1 <- function(x){y <- 5*x^2-10*x;return(y)}
> x1 <- -1
> x2 <- 1
> plot(f1,x1,x2)
> uniroot(f1,interval=c(x1,x2),f.lower = -5, f.upper = 15)
$root [1] -1
```

```
> uniroot(f1,interval=c(x1,x2))
$root [1] 2.678252e-07
```

```
> uniroot(f1,interval=c(x1,x2),f.lower = 15, f.upper = -5)
$root [1] 2.678252e-07
```



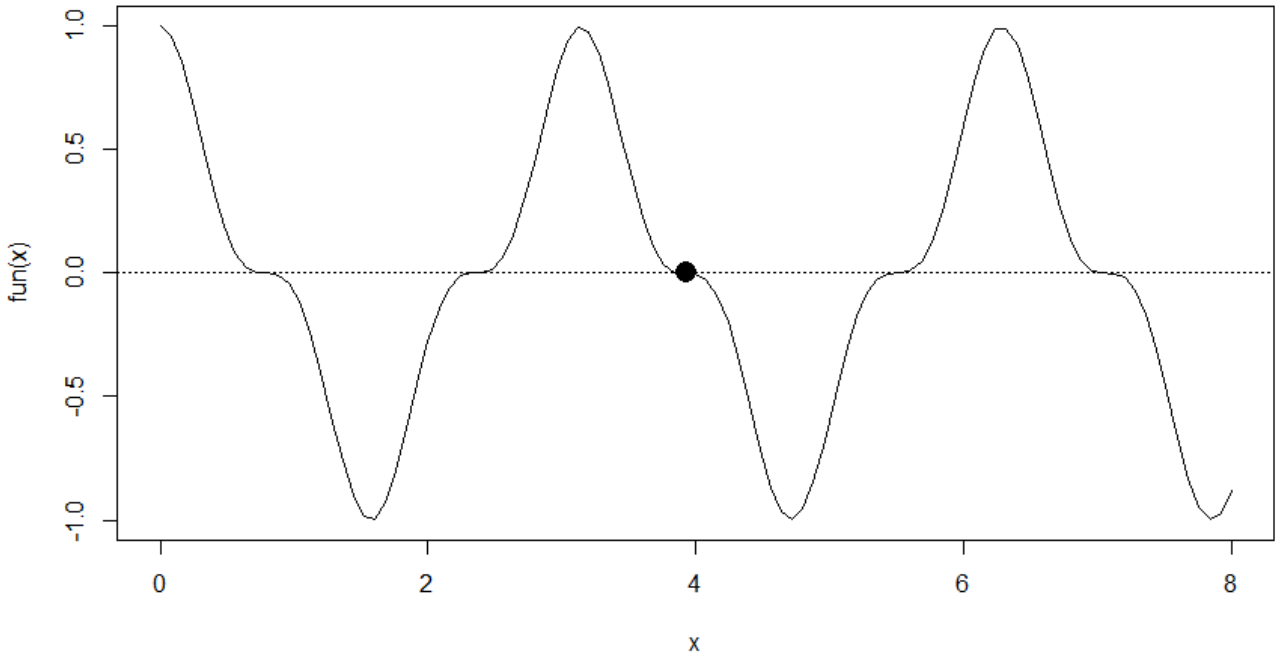
Функция uniroot.all

- Функция uniroot.all устраняет недостатки uniroot, позволяя вычислять несколько корней на заданном интервале.
- uniroot.all(f, interval, lower=min(interval), upper=max(interval), tol=.Machine$double.eps^0.2, maxiter=1000, n=100, ...)
- Отличие заключается в введении аргумента n — число подинтервалов, на которые делится исходный интервал, и на каждом из этих подинтервалов ищется нуль функции. Результат вызова функции – вектор с найденными решениями.

- uniroot.all входит в состав пакета rootSolve

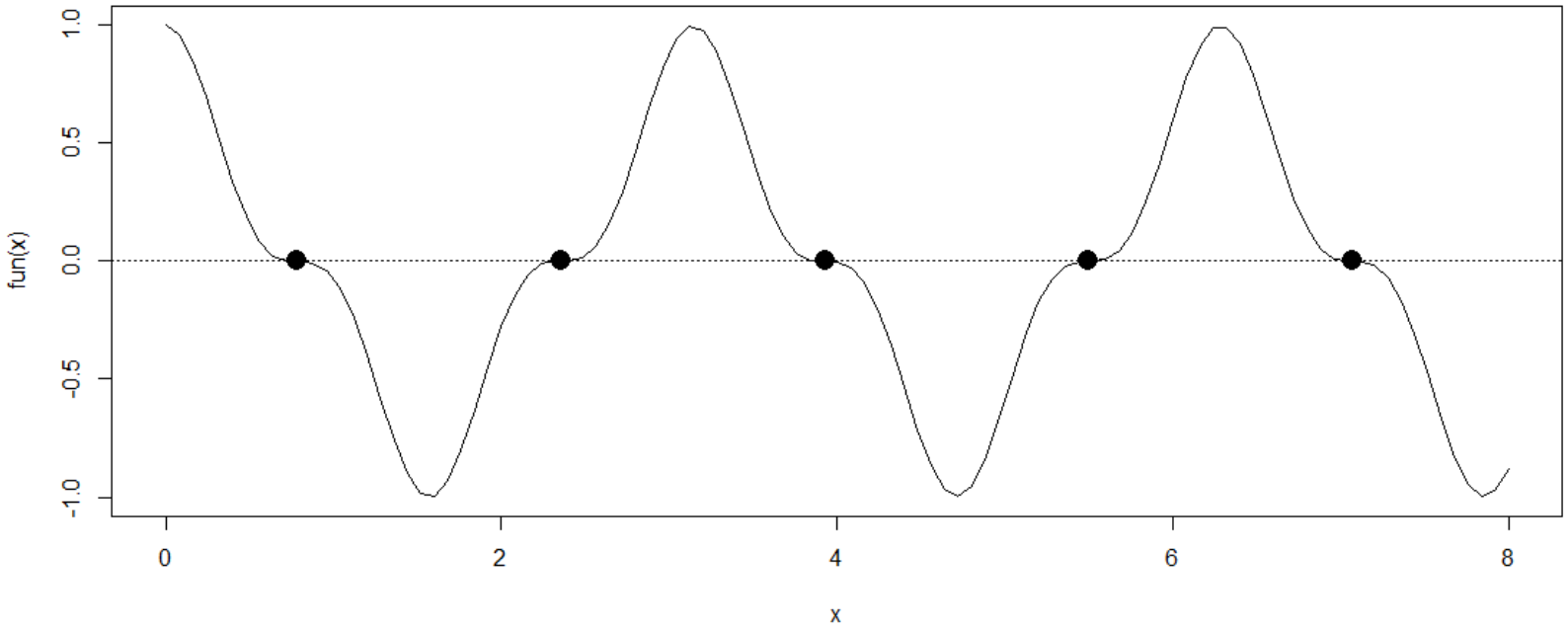
Функция uniroot

```
install.packages("rootSolve")
library(rootSolve)
fun <- function (x) cos(2*x)^3
curve(fun(x), 0, 8)
abline(h = 0, lty = 3)
uni <- uniroot(fun, c(0, 8))$root
points(uni, 0, pch = 16, cex = 2)
```



Функция uniroot.all

```
install.packages("rootSolve")
library(rootSolve)
fun <- function (x) cos(2*x)^3
curve(fun(x), 0, 8)
abline(h = 0, lty = 3)
uni <- uniroot.all(fun, c(0, 8))
points(uni, y = rep(0, length(uni)), pch = 16, cex = 2)
```



Интегрирование

Функция integrate

- integrate(f, lower, upper, ..., subdivisions=100, rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol, stop.on.error = TRUE, keep.xy = FALSE, aux = NULL)
- f — определённая в R функция, интегрируемая по первому аргументу. Результатом интегрирования функции должно быть конечное число, иначе выводится сообщение об ошибке.
- lower и upper — нижний и верхний пределы интегрирования. Могут быть бесконечными. Если известно, что в результате интегрирования на бесконечном (полу-бесконечном) интервале должно получиться конечное число, то в качестве пределов (предела) следует задавать Inf, а не большие числа (см. пример 59).
- ... — дополнительные аргументы, относящиеся к f. Они должны располагаться в том же порядке, что и в задаваемой интегрируемой функции.

Функция integrate

- subdivisions — максимальное число интервалов, на которое разбивается интервал интегрирования.
- rel.tol — требуемая относительная точность, не может быть меньше $0.5 \cdot 10^{-28}$.
- abs.tol — требуемая абсолютная точность.
- stop.on.error — логический аргумент. При возникновении ошибки прекращает вычисление, в противном случае выдаётся результат с предупреждениями.
- последние два аргумента: keep.xy и aux, — не используются, введены для совместимости с языком S.

Функция integrate

Найти интеграл от функции

$$f(x) = \frac{1}{(x+1)\sqrt{x}}$$

```
> int1 = function(x) {1/((x+1)*sqrt(x))}
> integrate(int1, lower = 0, upper = Inf)
3.141593 with absolute error < 2.7e-05

> integrate(int1, lower = 0, upper = 1000000)
Error in integrate(int1, lower = 0, upper = 1e+06) :
the integral is probably divergent
```

Функция integrate

Найти интеграл от функции

$$f(x) = \frac{1}{a(1+(x-b)^2)}$$

```
> int2 = function(x,a,b) {1/(a*(1+(x-b)^2))}
> integrate(int2, lower = 0, upper = Inf, a=2, b=2)

1.338973 with absolute error < 5.5e-08
```

Дифференцирование

Функции

- в базовой установке R реализованы три функции, которые в символьном виде вычисляют производные (в том числе и частные) заданных выражений.
- D (expr, name)
- deriv(expr, namevec, function.arg = NULL, tag = ".expr", hessian = FALSE, ...)
- deriv3(expr, namevec, function.arg = NULL, tag = ".expr", hessian = TRUE, ...)
- Функция D() позволяет вычислять производную функции по одному аргументу.
- Функции deriv() и deriv3() позволяют вычислить частные производные.

Аргументы

- expr — либо выражение (expression) или (за исключением функции D) формула (formula).
- name, namevec — символьный вектор, задающий имена переменных (только одна переменная для D()), по которым берутся производные.
- function.arg — если данный аргумент определён и не равен NULL — символьный вектор аргументов результирующей функции или функция (с пустым телом), или логический аргумент TRUE, определяющий использование функции с аргументами, имена которых определены namevec.
- tag — символьный аргумент — префикс, используемый для обозначения созданных локальных переменных при выводе результата.
- hessian — логический аргумент — нужно ли вычислять вторые производные и должны ли они быть включены в выводимые результаты.
- ... — дополнительные аргументы, определяемые используемыми методами.

Функции

- **Результатом работы функций будут символьные выражения, значения которых могут быть найдены в конкретных точках.**

найдём первую производную функции $f(x,y) = \sin(\cos(x + y^2))$ *по переменной* x

```
> exp1 = expression(sin(cos(x + y^2)))
> str1 <- D(exp1, "x")
> str1
-(cos(cos(x + y^2)) * sin(x + y^2))
```

$$-(\cos(\cos(x+y^2))\sin(x+y^2))$$

Функции

найдём первую производную функции $f(x,y) = \sin(\cos(x + y^2))$ *по переменной y*

```
> exp1 = expression(sin(cos(x + y^2)))
> str1 <- D(exp1, "y")
> str1
-(cos(cos(x + y^2)) * (sin(x + y^2) * (2 * y)))
```

$$-(\cos(\cos(x+y^2))(\sin(x+y^2)(2y)))$$

Функция eval()

```
> F.a <- expression(sin(a))
> F.a
expression(sin(a))

> a <- pi > eval(F.a)
[1] 1.224606e-16

> a <- pi/2
> eval(F.a)
[1] 1
```

Функция eval()

Вычисление значения производных в заданных точках

```
> x <- pi/4 > y <- pi/6
> exp1 = expression(sin(cos(x + y^2)))
> fn1.x <- D(exp1,"x")
> eval(fn1.x)
[1] -0.7698184

> fn1.y <- D(exp1,"y")
> eval(fn1.y)
[1] -0.8061519
```

Вторые производные

```
> exp1 = expression(sin(cos(x + y^2)))
> fn1.x <- D(exp1,"x")
> fn1.x.x <- D(fn1.x,"x")
> fn1.x.x

-(sin(cos(x + y^2)) * sin(x + y^2)
* sin(x + y^2) + cos(cos(x + y^2)) * cos(x + y^2))
```

$$-(\sin(\cos(x+y^2))\sin(x+y^2)\sin(x+y^2)+\cos(\cos(x+y^2))\cos(x+y^2))$$

Вторые производные. eval()

```
> x <- pi/4
> y <- pi/6
> eval(fn1.x.x)
[1] -0.7893344
```

```
> exp1 = expression(sin(cos(x + y^2)))
> fn1.y <- D(exp1,"y")
> fn1.y.y <- D(fn1.y,"y")
> fn1.y.y
-(sin(cos(x + y^2)) * (sin(x + y^2) * (2 * y))
*(sin(x + y^2) * (2 * y)) + cos(cos(x + y^2))
* (cos(x + y^2) * (2 * y) * (2 * y) + sin(x + y^2) * 2))
```

```
> x <- pi/4
> y <- pi/6
> eval(fn1.y.y)
[1] -2.405239
```

Решение задачи

Задача

Задача 1.

Найдите площадь, ограниченную осью Ox и кривой

$$y = x^3 - 6x^2 + 11x - 6.$$

Решение. Найдём точки пересечения кривой с осью *Ox*. Для этого решим уравнение $x^3 - 6x^2 + 11x - 6 = 0$. Полученные корни: $x_1 = 1$, $x_2 = 2$, $x_3 = 3$. Построив эскиз графика (рис.1), мы видим, что на отрезке $[2,3]$ функция отрицательна. Поэтому на этом отрезке для вычисления площади берём значение интеграла с противоположным знаком.

$$S = S_1 - S_2 = \int_1^2 (x^3 - 6x^2 + 11x - 6)dx - \int_2^3 (x^3 - 6x^2 + 11x - 6)dx = \frac{1}{2}.$$

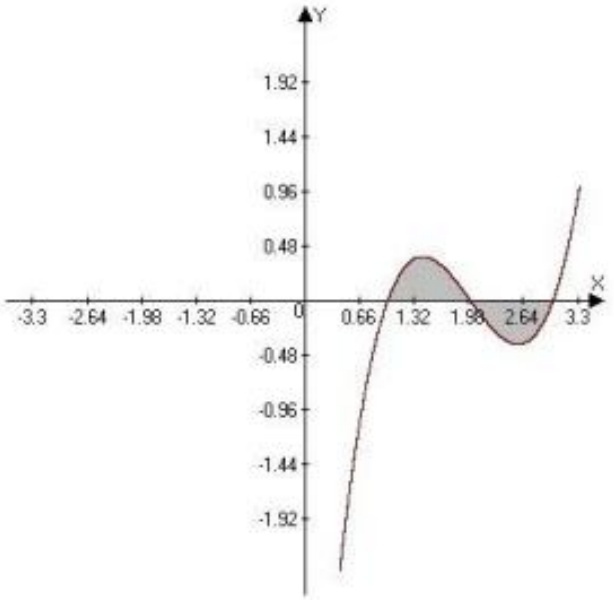
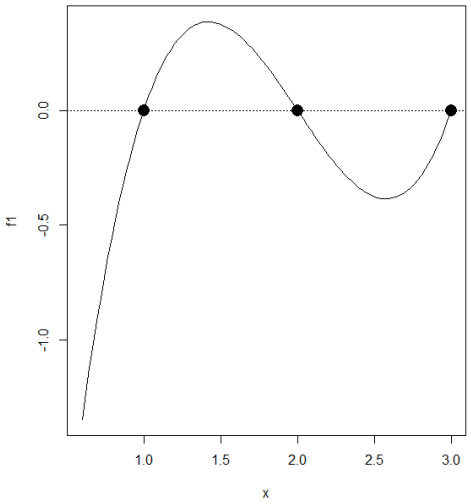


Рис.1

Задача

$$y = x^3 - 6x^2 + 11x - 6.$$



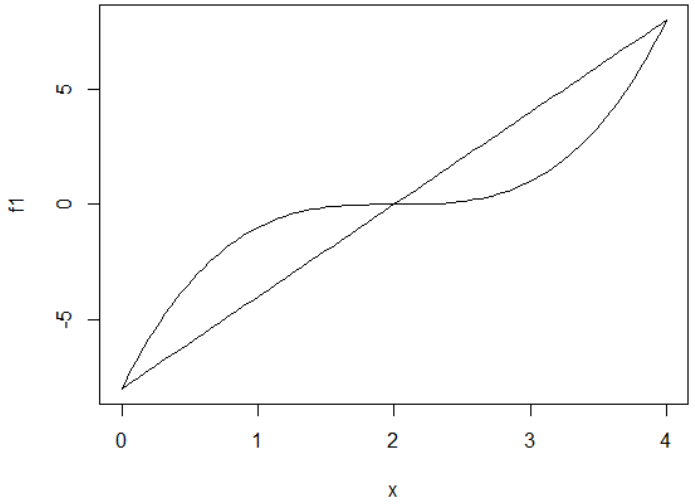

```
> f1 = function(x) {x^3-6*x^2+11*x-6}
> plot(f1,0.6,3)
> abline(h = 0, lty = 3)
> install.packages("rootSolve")
Installing package into ‘C:/Users/v.shevtsov/Documents/R/win-library/3.4’
> library(rootSolve)
> uni <- uniroot.all(f1, c(0.6, 3))
> points(uni, y = rep(0, length(uni)), pch = 16, cex = 2)
> uni
[1] 3.0000000 0.9999999 2.0000010
> uni <- sort(uni)
> uni
[1] 0.9999999 2.0000010 3.0000000
> int1 <- integrate(f1, lower = uni[1], upper = uni[2])
> int1
0.25 with absolute error < 2.8e-15
> int1[1]
$value [1] 0.25
> as.numeric(integrate(f1, lower = uni[1], upper = uni[2])[1])
       -as.numeric(integrate(f1, lower = uni[2], upper = uni[3])[1])
[1] 0.5
```

Задача

Вычислить площадь фигуры, ограниченной графиками функций

$$y = (x-2)^3, y = 4x - 8$$

Ответ: 8




```
install.packages("rootSolve")
library(rootSolve)
f1 <- function(x){4*x-8}
f2 <- function(x){(x-2)^3}
f3 <- function(x){f1(x)-f2(x)}
uni <- uniroot.all(f3,c(-10,10))
plot(f1,uni[1],uni[length(uni)],xlim=c(uni[1],uni[length(uni)]))
curve(f2,uni[1],uni[length(uni)],add=TRUE)
abs(as.numeric(integrate(f1,lower=uni[1],upper=uni[2])$value))
   -abs(as.numeric(integrate(f2,lower=uni[1],upper=uni[2])$value))
   +abs(as.numeric(integrate(f1,lower=uni[2],upper=uni[3])$value))
   -abs(as.numeric(integrate(f2,lower=uni[2],upper=uni[3])$value))
```

Графические функции низкого уровня

Функции низкого уровня

- abline() — построение прямых линий в уже существующем графическом окне;
- arrows() — рисование стрелок;
- axis() — построение оси графика;
- box() — построение рамки вокруг графика;
- grid() — задание прямоугольной сетки на графике;
- legend() — задание различных легенд к графику;
- lines() — построение линий, соединяющих заданные точки;
- mtext() — вывод надписей в соответствующей области;
- points() — добавление точек на график;

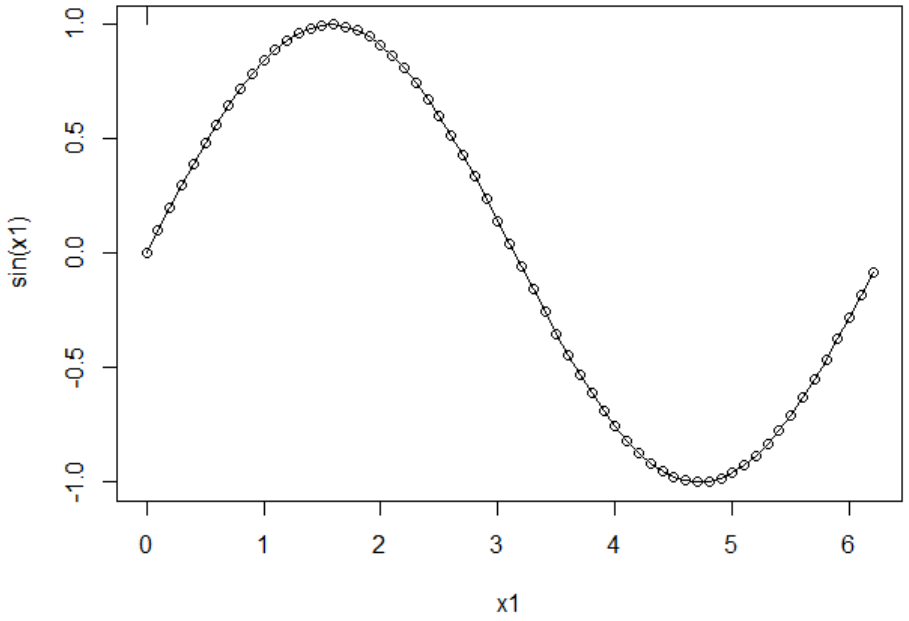
Функции низкого уровня

- polygon() — построение многоугольников;
- rect() — построение прямоугольников;
- segments() — соединение точек прямыми отрезками;
- symbols() — построение одного из шести видов фигур (круг, квадрат, прямоугольник, звезда, термометр, boxplot) на графике;
- text() — добавление текста к графику;
- title() — добавление заголовков;
- xspline() — построение сплайна относительно заданных контрольных точек.

Функция lines()

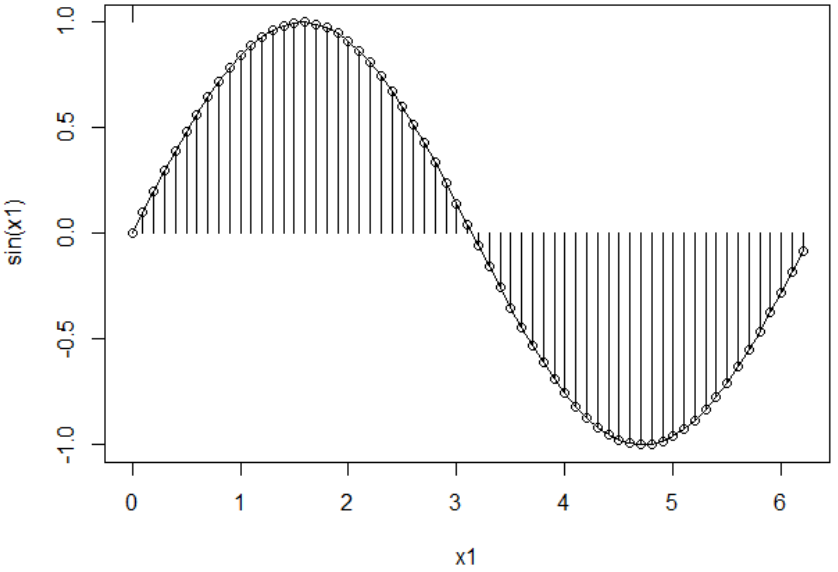
- Функция lines() соединяет заданные точки отрезками. Её вид:
- lines(x, y = NULL, type = "l", ...)
- Единственным аргументом функции является аргумент x — список из двух компонент (первая компонента — координаты по оси 0x, вторая — координаты по оси 0y) или матрица из двух столбцов (первый столбец — координаты по оси 0x, второй — координаты по оси 0y).
- Если аргумент x — числовой вектор — координаты по оси 0x, то должен быть задан аргумент y — координаты по оси 0y.
- Среди значений аргументов x и (или) y могут быть и NA. В таком случае к точке с такой координатой (координатами) (или от неё) прямая просто не строится — создаются разрывы в линиях.
- Если type = 'h', то параметр col, отвечающий за цвет линий, можно задать как вектор.

Функция lines()



type = 'l'

type = 'h'

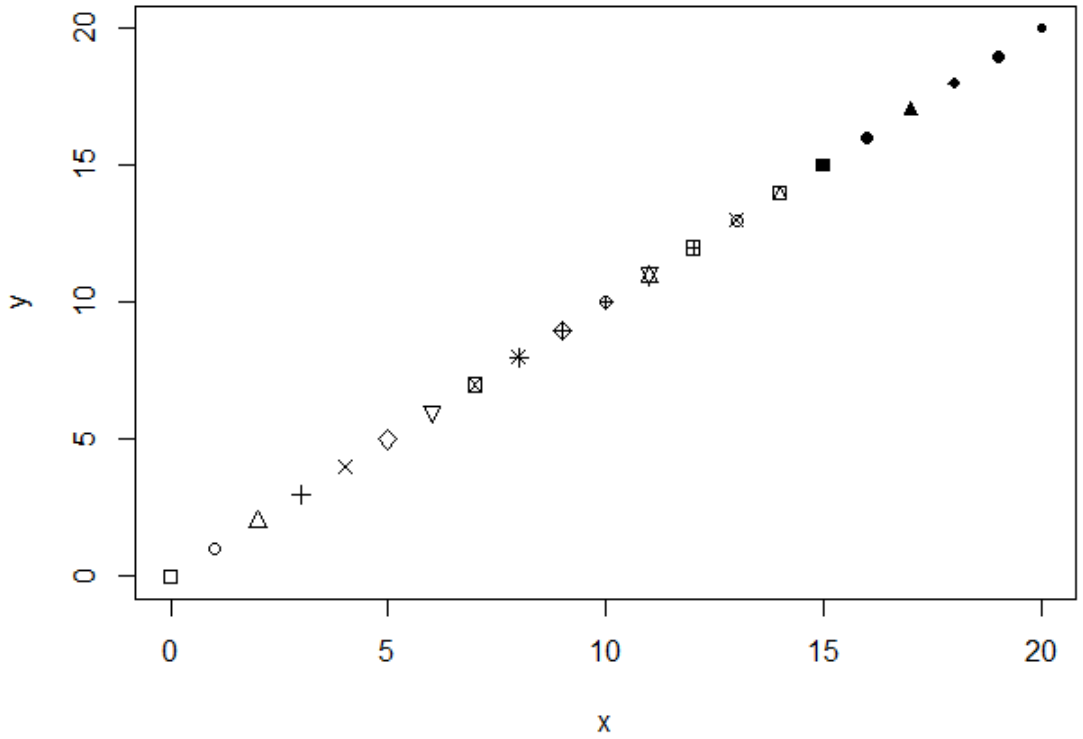


Функция points()

- Функция points() рисует точки заданного типа (по умолчанию - круги) в заданных координатах. Вид функции: points(x, y = NULL, type = "p", ...)
- Снова, только один обязательный аргумент — x, если x — список или матрица с координатами строящихся точек. Если x – числовой вектор — координаты по оси 0x, то дополнительно задаются координаты по оси 0y — аргумент y.
- pch — числовой аргумент — тип рисуемого символа в точке с заданными координатами. Возможные значения — от 0 до 255

Функция points()

```
> x <- 0:20
> y <- 0:20
> plot(x,y,type="n")
> points(x,y,pch=x)
```



Нахождение экстремумов функции. Решение задач оптимизации

Функция optim()

- **optim(par, fn, gr = NULL, ..., method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"), lower = -Inf, upper = Inf, control = list(), hessian = FALSE)**
  - par — первоначальные значения параметров, относительно которых проводится оптимизация функции.
  - fn — минимизируемая (или максимизируемая) функция, первый аргумент которой — вектор параметров, относительно которых проводится оптимизация. Результатом вызова функции должен быть скаляр.
  - gr — функция, возвращающая градиент для методов ’BFGS’, ’CG’ и ”L-BFGS-B’.
  - ... — дополнительные аргументы, используемые fn и gr.

Функция optim()

- **method — используемый метод. Для функции optim() реализованы следующие методы:**
  - “Nelder-Mead” — базовый метод. Устойчивый, используется только сама функция, но медленно сходится. Применим для недифференцируемых функций.
  - “BFGS” — квази-Ньютоновский метод, использующий функцию и её градиент.
  - “CG” — метод сопряжённых градиентов, менее устойчив по сравнению с предыдущими двумя, но также и менее ресурсозатратен.
  - “L-BFGS-B” — модификация метода ’BFGS’, использующая ограничения на переменные.
  - “SANN” — стохастический оптимизационный метод. Использует только саму функцию, но медленно сходится. Применим для недифференцируемых функций. Сильно зависит от контрольных параметров.

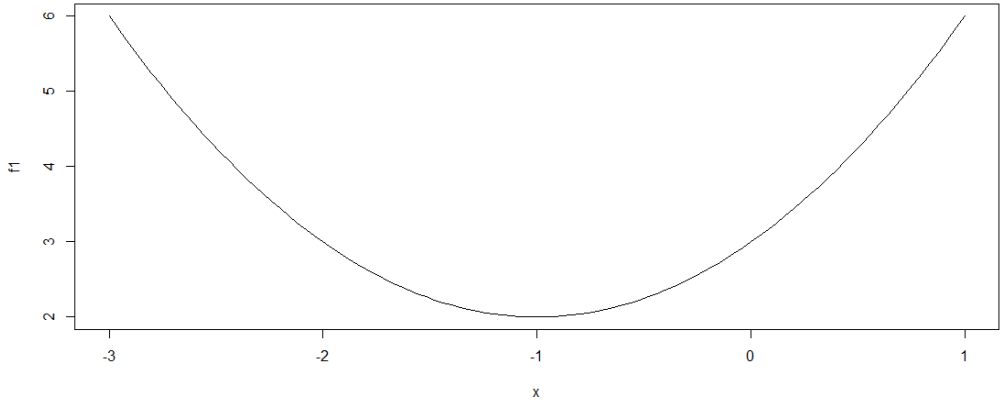
- **lower и upper — границы для переменных, используются только при методе L-BFGS-B.**
- **control — список, в который могут входить следующие управляющие параметры:**
  - trace — неотрицательное целое число. Если положительное, то выводится информация о ходе оптимизации. Чем больше значение принимает контрольный параметр trace, тем более подробная информация выводится.
  - fnscale — масштабирующий параметр для оптимизируемой функции. Если принимает отрицательное значение, то решается задача максимизации. Таким образом, оптимизация происходит для fn(par)/fnscale.
  - parscale — числовой вектор, масштабирующий параметры оптимизации. Таким образом, оптимизация производится по par/parscale параметрам.
  - ndeps — вектор, определяющий размер шага оптимизации, по умолчанию — 10−3.
  - maxit — максимальное число итераций. Значения по умолчанию: 100 — для методов ’BFGS’, ’CG’ и ’L-BFGS-B’, 500— ’NelderMead’, 10000 — для ’SANN’.
  - abstol — абсолютная точность сходимости, применима только для неотрицательных функций.

- reltol — относительная точность сходимости. Значение по умолчанию 10−8.
- alpha , beta, gamma — масштабирующие параметры для метода ’Nelder-Mead’.
- REPORT — частота выводимых сообщений для методов ’BFGS’, ’L-BFGS-B’ и ’SANN’, если контрольный параметр trace положителен. По умолчанию одно сообщение выводится на каждые 10 итераций для методов ’BFGS’ и ’L-BFGS-B’ или на каждые 100 итераций для метода ’SANN’.
- type — контрольный параметр для метода сопряжённых градиентов. Выбор разновидности метода. Значение 1 — вариант Fletcher–Reeves, 2 — Polak–Ribiere, 3 — Beale–Sorenson.
- lmm — целое число — вариант модификации метода bf’L-BFGS-B’. По умолчанию — значение 5.
- factr — числовой параметр, контролирующий сходимость bf’LBFGS-B’ метода. Метод сходится, если изменение оптимизируемой функции меньше либо равно значению машинной точности (10−15), умноженному на factr. По умолчанию — 107.
- pgtol — параметр, управляющий сходимостью метода bf’L-BFGS-B’.
- temp и tmax — управляющие параметры для метода ’SANN’.
- **hessian — логический аргумент — нужно ли выводить численные значения матрицы вторых производных.**

В результате работы функции optim() создаётся список со следующими полями:

- **par — оптимальные найденные значения параметров (точка минимума или максимума).**
- **value — значение оптимизируемой функции в найденной точке.**
- **counts — целочисленный вектор из двух компонент, описывающий, сколько раз использовалась функция и её градиент при оптимизации.**
- **convergence — целое число - сообщение о типе завершения оптимизации:**
  - 0 — удачное завершение (практически всегда для ’SANN’).
  - 1 — достигнуто максимальное число итераций.
  - 10 — расхождение метода ’Nelder-Mead’.
  - 51 — ошибка в методе bf’L-BFGS-B’.
  - 52 — ошибка в методе bf’L-BFGS-B’..
- **message — дополнительно выводимая информация об оптимизации (либо NULL).**
- **hessian — матрица — оценка матрицы вторых производных в найденной точке (выводится только если аргумент hessian функции optim() принимает значение TRUE).**

Пример




```
> f1 <- function(x){(x+1)^2+2}
> plot(f1,-3,1)
> optim(2,f1,method="Nelder-Mead")$par
[1] -0.8
Warning message:
In optim(2, f1, method = "Nelder-Mead") :
  одномерная оптимизация методом Нелдера-Мида ненадежна:
используйте "Brent" или прямо optimize()
> optim(2,f1,method="BFGS")$par
[1] -1
> optim(2,f1,method="CG")$par
[1] -0.9999993
> optim(2,f1,method="L-BFGS-B")$par
[1] -1
> optim(2,f1,method="SANN")$par
[1] -1.000068
```

Пример

$$z = \frac{x^2}{a^2} + \frac{y^2}{b^2}$$

a=1, b=1

```
> f1 <- function(x,a,b){x1 <- x[1];x2 <- x[2];x1^2/a^2 + x2^2/b^2}
> optim(c(-2,-2),f1,a=1,b=1,method="Nelder-Mead")$par
[1] -8.750437e-05 -1.581483e-04
> optim(c(-2,-2),f1,a=1,b=1,method="BFGS")$par
[1] -2.853841e-16 -2.853841e-16
> optim(c(-2,-2),f1,a=1,b=1,method="CG")$par
[1] 4.924372e-07 4.924372e-07
> optim(c(-2,-2),f1,a=1,b=1,method="L-BFGS-B")$par
[1] -1.482312e-20 -1.482312e-20
> optim(c(-2,-2),f1,a=1,b=1,method="SANN")$par
[1] 0.014004992 0.009766761
```

Пример

$$z = \frac{x^2}{a^2} + \frac{y^2}{b^2}$$

a=3, b=-6

```
> f1 <- function(x,a,b){x1 <- x[1];x2 <- x[2];x1^2/a^2 + x2^2/b^2}
> optim(c(-2,-2),f1,a=3,b=-6,method="Nelder-Mead")$par
[1]  0.0001045455 -0.0002875222
> optim(c(-2,-2),f1,a=3,b=-6,method="BFGS")$par
[1] -1.166185e-09 -7.724236e-11
> optim(c(-2,-2),f1,a=3,b=-6,method="CG")$par
[1] -1.335628e-19 -4.460854e-04
> optim(c(-2,-2),f1,a=3,b=-6,method="L-BFGS-B")$par
[1] 4.402558e-07 3.585110e-06
> optim(c(-2,-2),f1,a=3,b=-6,method="SANN")$par
[1] -0.1001479 -0.1888506
```

Функции

- optimize(f = , interval = , ..., lower = min(interval), upper = max(interval), maximum = FALSE, tol = .Machine$double.eps^0.25)
- optimise(f = , interval = , ..., lower = min(interval), upper = max(interval), maximum = FALSE,

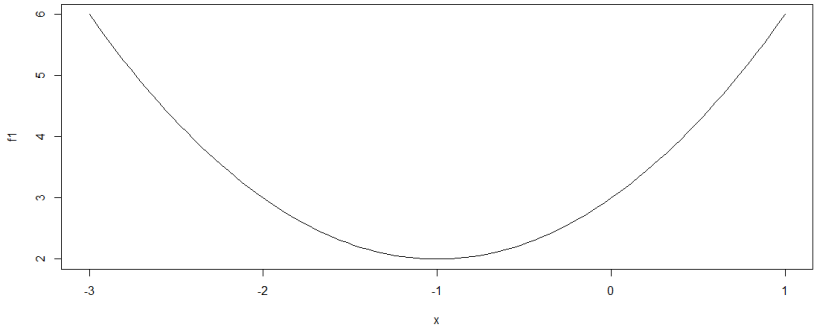
- функции одинаковы и различаются только написанием. Позволяют определить точку минимума (максимума) функции на заданном интервале

Аргументы

- f — оптимизируемая функция. В зависимости от значения аргумента maximum ищется либо минимум, либо максимум функции f.
- interval — числовой вектор, задающий интервал, на котором ищется экстремум функции.
- ... — дополнительные аргументы для f.
- lower — нижняя граница интервала оптимизации.
- upper — верхняя граница интервала оптимизации.
- maximum — логический аргумент. Нужно искать минимум (по умолчанию) или максимум функции?
- tol — желаемая точность.

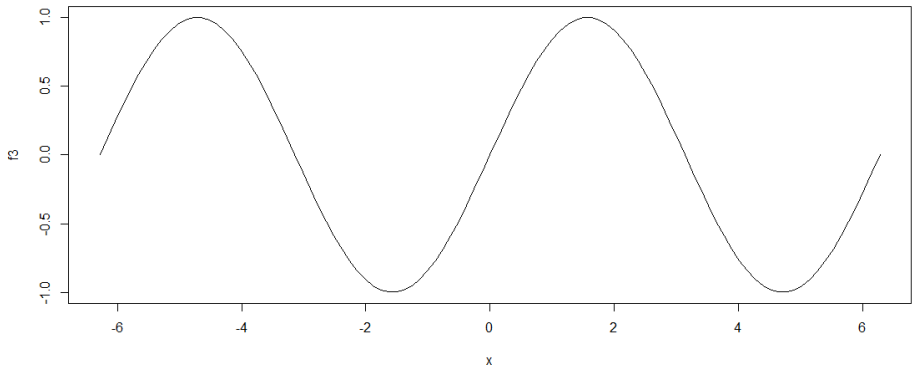
- Функции optimize и optimise применимы (только) для непрерывных функций. Если оптимизируемая функция унимодальна, то буде найден (при правильном задании интервала оптимизации) глобальный минимум (максимум), если же оптимизируемая функция не унимодальна, то, скорее всего, будет найден локальный минимум (максимум).
- Результат вызова optimize или optimise — список из двух компонентов: minimum — найденной точки минимума (максимума) и значения функции в этой точке — objective.

Функция optimise




```
> f1 <- function(x){(x+1)^2+2}
> plot(f1,-3,1)
> optim(2,f1,method="Nelder-Mead")$par
[1] -0.8
Warning message:
In optim(2, f1, method = "Nelder-Mead") :
  одномерная оптимизация методом Нелдера-Мида ненадежна:
используйте "Brent" или прямо optimize()
> optim(2,f1,method="BFGS")$par
[1] -1
> optim(2,f1,method="CG")$par
[1] -0.9999993
> optim(2,f1,method="L-BFGS-B")$par
[1] -1
> optim(2,f1,method="SANN")$par
[1] -0.99963
> optimise(f1,c(-2,2),maximum = FALSE)
$minimum
[1] -1

$objective
[1] 2
```

Функция optimise

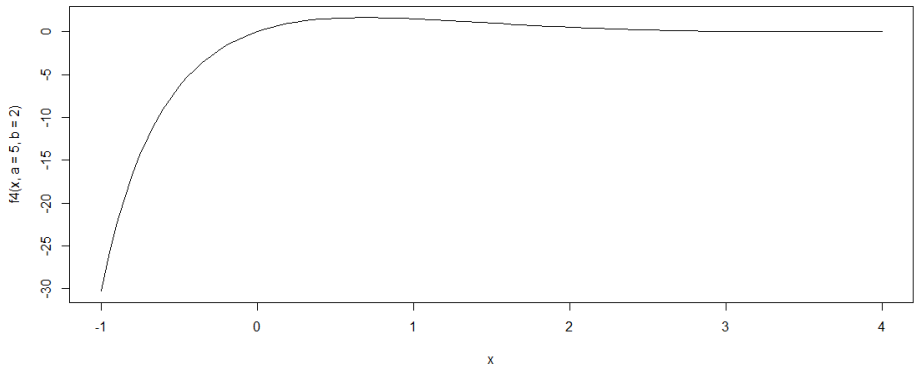



```
> f3 <- function(x){sin(x)}
> x1 <- -2*pi
> x2 <- 2*pi
> plot(f3,x1,x2)
> optimise(f3,c(x1,x2),maximum = FALSE)
$minimum
[1] -1.570789

$objective
[1] -1

> optimise(f3,c(x1,x2),maximum = TRUE)
$maximum
[1] 1.570789

$objective
[1] 1
```

Функция optimise

$$f(x) = \frac{(x-5)^2 \sin x}{(x+2)^2}$$

[−1;4], [0;3] *и* [1,3]




```
> f4 <- function (x,a,b) {(x-a)^2*sin(x)/((x+b)^2)}
> x=seq(from=-1,to=4,by=1000)
> curve(f4(x,a=5, b=2),xlim=c(-1,4))
> optimise(f4, c(-1, 4), a=5, b=2,maximum = FALSE)$minimum
[1] 3.642265
> optimise(f4, c(-1, 4), a=5, b=2,maximum = TRUE)$maximum
[1] 0.6917657
> optimise(f4, c(0, 3), a=5, b=2,maximum = FALSE)$minimum
[1] 2.999924
> optimise(f4, c(0, 3), a=5, b=2,maximum = TRUE)$maximum
[1] 0.6917887
> optimise(f4, c(1, 3), a=5, b=2,maximum = FALSE)$minimum
[1] 2.999959
> optimise(f4, c(1, 3), a=5, b=2,maximum = TRUE)$maximum
[1] 1.000041
```

Функция nlm()

- Функция nlm() находит минимум заданной функции *f* при помощи метода Ньютона. Используются частные производные и вторые производные.
- nlm(f, p, ..., hessian = FALSE, typsize = rep(1, length(p)), fscale = 1, print.level = 0, ndigit = 12, gradtol = 1e-6, stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000), steptol = 1e-6, iterlim = 100, check.analyticals = TRUE)

- **f — минимизируемая функция. Если среди атрибутов функции заданы градиент и гессиан, то они будут использованы при вычислении. Иначе будут использованы результаты численного дифференцирования. К примеру, deriv позволяет получить функцию с градиентом в качестве атрибута.**
- **p — начальные значения для аргументов, по которым ищется минимум.**
- **... — дополнительные для f аргументы.**
- **hessian — логический аргумент. Если значение TRUE, то выводится значение матрицы вторых производных в найденной точке минимума.**
- **typsize — оценка размера каждого параметра в точке минимума.**
- **fscale — оценка размерности f в точке минимума.**
- **print.level — числовой аргумент — определяет объём информации, выводимой в процессе минимизации.**
  - 0 (значение по умолчанию) — информация не выводится,
  - 1 — данные о первом и последнем этапам минимизации выводятся.
  - 2 — полностью вся информация о процессе оптимизации выводится.
- **ndigit — число значащих знаков для функции f.**
- **gradtol — положительное число, достигая которое, градиент считается равным нулю и процесс оптимизации прерывается.**
- **stepmax — положительное число — отвечает за шаг оптимизации.**
- **steptol — положительное число — минимальный допустимый шаг оптимизации.**
- **iterlim — максимальное число итераций.**
- **check.analyticals — логический аргумент — нужно ли сравнивать аналитические градиенты и вторые производные с результатами численного дифференцирования. Помогает выявить некорректно сформулированные аналитические градиенты и вторые производные.**

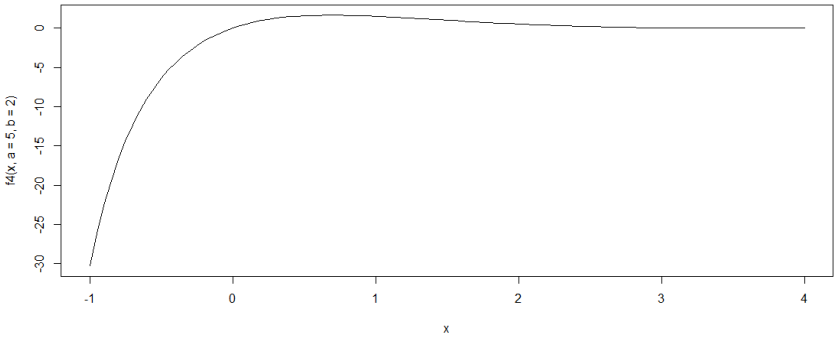
Результатом работы функции nlm() является список со следующими полями:

- minimum — значение f в точке минимума.
- estimate — точка, в которой достигает своего минимума функция f.
- gradient — значение градиента в найденной точке минимума.
- hessian — значение матрицы вторых производных в найденной точке минимума.
- code — код завершения процесса оптимизации. 1 и 2 - найденные значения являются (скорее всего) решением, 3–5 — сообщения об ошибках.
- iterations — число выполненных итераций.

Функция nlm

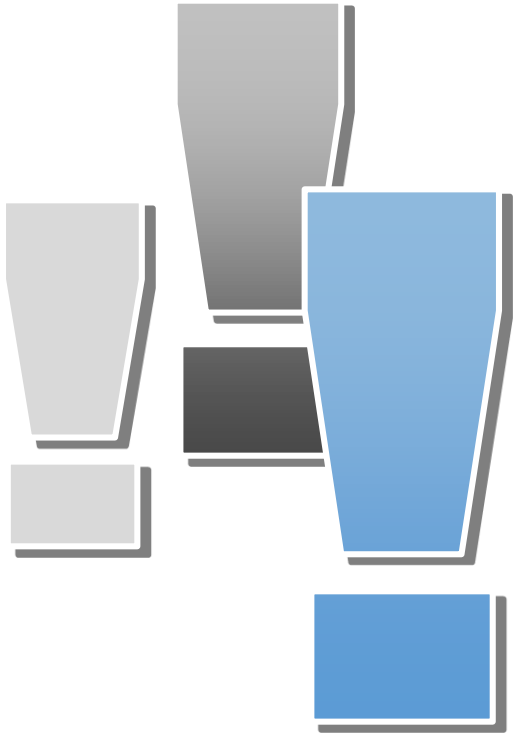
$$f(x) = \frac{(x-5)^2 \sin x}{(x+2)^2}$$

[−1;4], [0;3] *u* [1,3]




```
> f4 <- function (x,a,b) {(x-a)^2*sin(x)/((x+b)^2)}
> x=seq(from=-1,to=4,by=1000)
> curve(f4(x,a=5, b=2),xlim=c(-1,4))
> nlm(f4,-1,a=5,b=2)$minimum
[1] -9.986739e+14
> nlm(f4,0,a=5,b=2)$minimum
[1] -8.076856e+14
> nlm(f4,1,a=5,b=2)$minimum
[1] -0.02779574
```

Спасибо за внимание!



Шевцов Василий Викторович

shevtsov-vv@rudn.ru
+7(903)144-53-57