

2.1. Использование базовых математических функций и операций

Задание

Запустить все команды из разделов 2.1.1 и 2.1.2. Разобраться в особенностях применения базовых математических операций и функций.

2.1.1. Логические операции

Логические операции, предусмотренные в R, приведены ниже:

Запись в R	Описание
$! x$	логическое отрицание
$x \& y$	поэлементное логическое И
$x \&\& y$	логическое И только для первых элементов векторов x и y
$x y$	поэлементное логическое ИЛИ
$x y$	логическое ИЛИ только для первых элементов x и y
$xor(x, y)$	если x и y различны, то получим TRUE, иначе FALSE

В следующей таблице приводятся операторы сравнения, результатом работы которых являются логические объекты.

Оператор	Описание
$>$	больше
$>=$	больше или равно
$<$	меньше
$<= y$	меньше или равно
$==$	тождество
$!=$	не тождественны

Эти операторы сравнения применимы не только к числовым переменным, но и к символьным. Тогда сравнение происходит в лексикографическом порядке.

```
> 'a' < 'b'
```

```
[1] TRUE
```

```
> 'иэ' >= 'ия'
```

```
[1] FALSE
```

```
> 'иэ' != 'ия'
```

```
[1] TRUE
```

```
> 'и' > 'э'
```

```
[1] FALSE
```

```
> 'и' 'вг'>='вв'
```

```
[1] TRUE
```

Если сравниваются символьный объект и числовой, то символьный будет больше.

```
> x=10000; y='а'; x>y
```

```
[1] FALSE
```

```
> x=10000;y='а';y>x
```

```
[1] TRUE
```

Также можно сравнивать и логические переменные:

```
> x=F;y=T
```

```
> x<y
```

```
[1] TRUE
```

Логические переменные можно сравнивать как с числовыми, так и с символьными. При этом логическая переменная переводится либо в числовой тип, либо в символьный.

При сравнении любого объекта с **NA** или **NaN** результатом будет **NA**.

```
> x=10;y=NA
```

```
> x>y
```

```
[1] NA
```

```
> x=NA;y=NaN > x>=y
```

```
[1] NA
```

2.1.2. Математические функции

Числа в R являются одним из основных типов данных. Дробная часть числа от целой отделяется точкой (только точкой). Показатель отделяется от мантиссы либо символом **e**, либо **E**. В примере

```
> x=0.235;x
```

```
[1] 0.235
```

```
> x=.235;x
```

```
[1] 0.235
```

```
> x=23.5e-2;x
```

```
[1] 0.235
```

```
> x=235E-3; x
```

```
[1] 0.235
```

```
> x=0.00235e2; x
```

```
[1] 0.235
```

представлены различные способы записи одного и того же числа.

В R над числами можно выполнять обычные арифметические операции:

+ (сложение),

– (вычитание),

* (умножение),

/ (деление),

^ (возведение в степень),

%% (целочисленное деление),

%% (остаток от деления).

Операции имеют обычный приоритет, т.е. сначала выполняется возведение в степень, затем умножение или деление, потом уже сложение или вычитание. В выражениях также используются круглые скобки и операции в них имеют приоритет. Набранное в командной строке арифметическое выражение после нажатия на Enter вычисляется и результат сразу же отображается:

```
> 2*(7-9)^8+6/3
```

```
[1] 514
```

Ответ — 514.

Стоит заметить, что все арифметические операции, приведённые выше, являются на самом деле функциями:

```
> '+'(2,3) [1] 5 > '/'(9,2) [1] 4.5
```

Логарифмические и экспоненциальные функции

Экспоненты:

- `exp(x)` — вычисление e^x ;
- `expm(x)` — для $|x| \leq 1$ нахождение $e^x - 1$.

В R реализовано несколько логарифмических функций:

- $\log(x)$ и $\log_b(x)$ — натуральный логарифм числа x ;
- $\log_{10}(x)$ — логарифм по десятичному основанию;
- $\log_2(x)$ — логарифм по основанию 2;
- $\log_{1p}(x)$ — вычисление логарифма $\ln(1+x)$ для $|x| \leq 1$ (при $x \approx -1$ точность вычислений снижается);
- $\log(x, \text{base=})$ и $\log_b(x, \text{base=})$ — вычисление логарифмов по произвольному основанию.

```
> x=exp(2); x
[1] 7.389056
> log(x)
[1] 2
> logb(x)
[1] 2
> log10(100)
[1] 2
> log(8)
[1] 2.079442
> log1p(0.01)
[1] 0.00995033
> log1p(-0.99999)
[1] -11.51293
> log(125,base=5)
[1] 3
> logb(256,base=16)
[1] 2
```

Функции округления

Доступны следующие функции округления.

- $\text{ceiling}(x)$ — находит минимальное целое, не меньшее x (округление в сторону +1);
- $\text{floor}(x)$ — возвращает максимальное целое, не превосходящее x

(округление в сторону -1);

- `trunc(x)` — отбрасывает дробную часть (округление в сторону 0);
- `round(x)` — округляет к ближайшему целому;
- `round(x, dig)` — общий вид функции `round(x)`, дополнительный параметр `dig` указывает на число знаков после запятой, до которого нужно произвести округление;
- `signif(x, digits = 6)` — округляет до заданного (`digits`) числа значащих знаков (т.е. чисел, отличных от нуля), по умолчанию параметр `digits` равен 6.

```
> x=1.5555
```

```
> ceiling(x)
```

```
[1] 2
```

```
> floor(x)
```

```
[1] 1
```

```
> trunc(x)
```

```
[1] 1
```

```
> y=-1.5
```

```
> ceiling(y)
```

```
[1] -1
```

```
> floor(y)
```

```
[1] -2
```

```
> trunc(y)
```

```
[1] -1
```

```
> x=1/33;x
```

```
[1] 0.03030303
```

```
> round(x)
```

```
[1] 0
```

```
> round(x, 4)
```

```
[1] 0.0303
```

```
> signif(x, 5)
```

```
[1] 0.030303
```

Модуль и квадратный корень

- `abs(x)` — модуль аргумента x , который может быть как числовым (действительный или комплексный), так и логическим аргументом.

```
> x=-5; y=F; > abs(x)
```

```
[1] 5
```

```
> abs(y)
```

```
[1] 0
```

```
> z=T; abs(z)
```

```
[1] 1
```

```
> z=5-1i*2 > abs(z)
```

```
[1] 5.385165
```

- `sqrt(x)` — корень квадратный переменной x .

```
> x=4; sqrt(x)
```

```
[1] 2
```

```
> y=-4; sqrt(y)
```

```
[1] NaN
```

Предупреждение

In `sqrt(y)` : созданы NaN

Чтобы найти квадратный корень отрицательного числа x , нужно x определить как комплексное число:

```
> y=as.complex(y); sqrt(y)
```

```
[1] 0+2i
```

Специальные функции

К специальным функциям относятся: бета-функция и её логарифм, гамма-функция, логарифм модуля гамма-функции, производные гамма-функции, факториал и число сочетаний.

- `gamma(x)` — гамма-функция $\Gamma(x)$, где x — действительное число, не равное нулю и не являющееся целым отрицательным.
- `lgamma(x)` — натуральный логарифм абсолютного значения гамма-функции.
- `digamma(x)` — первая производная натурального логарифма

гаммафункции.

- `trigamma(x)` — вторая производная натурального логарифма гаммафункции.

- `psigamma(x, deriv)` — вычисляет *deriv*-производную от ψ -функции (первой производной гамма-функции) (по умолчанию *deriv* = 0, т.е. $psigamma(x) = digamma(x)$).

```
> x=-8.9
```

```
> gamma(x)
```

```
[1] -3.507258e-05
```

```
> x=2.15
```

```
> gamma(x)
```

```
[1] 1.072997
```

```
> digamma(x)
```

```
[1] 0.5152385
```

```
> psigamma(x)
```

```
[1] 0.5152385
```

```
> trigamma(x)
```

```
[1] 0.5894157
```

- `beta(a, b)` — вычисляется значение бета-функции $B(a, b)$ с параметрами a и b , где параметры a и b больше нуля.

- `lbeta(a, b)` — натуральный логарифм бета-функции $B(a, b)$.

```
> a=2;b=3
```

```
> beta(a,b)
```

```
[1] 0.08333333
```

```
> lbeta(a,b)
```

```
[1] -2.484907
```

- `choose(n, k)` — число сочетаний C_n^k из n по k , где n — действительное число, k — целое положительное.

- `lchoose(n, k)` — натуральный логарифм числа сочетаний C_n^k .

- `factorial(x)` — факториал x , где $x \geq 0$.

- `lfactorial(x)` — натуральный логарифм факториала x .

```
> n=5.4;k=3;x=4.6;x1=4
```

```
> choose(n,k)
```

```
[1] 13.464
```

```
> choose(round(n),k)
```

```
[1] 10
```

```
> factorial(x)
```

```
[1] 61.55392
```

```
> factorial(x1)
```

```
[1] 24
```

Тригонометрические функции

- $\sin(x)$ — $\sin(x)$, где x задан в радианах (например $\pi/6$);
- $\cos(x)$ — $\cos(x)$;
- $\tan(x)$ — $\tan(x)$.

Обратные тригонометрические функции:

- $\sin(x)$ — $\arcsin(x)$;
- $\cos(x)$ — $\arccos(x)$;
- $\tan(x)$ — $\arctg(x)$.

```
> sin(pi/2)
```

```
[1] 1
```

где π — это константа π .

Если у функции несколько аргументов, то они отделяются знаком , (запятая).
Например, $\text{atan2}(y, x)$ возвращает угол между осью Ox и вектором $(x; y)$.

Гиперболические функции:

- $\cosh(x)$ — гиперболический косинус аргумента x ;
- $\text{acosh}(x)$ — обратный гиперболический косинус x ;
- $\sinh(x)$ — гиперболический синус x ;
- $\text{asinh}(x)$ — обратный гиперболический синус x ;
- $\tanh(x)$ — гиперболический тангенс x ;
- $\text{atanh}(x)$ — обратный гиперболический тангенс x .

Операции над комплексными переменными

Над комплексными числами в R можно производить операции сложения, вычитания, умножения, деления, возведения в степень:

```
> x=5+1i*5
> y=4-1i*5
> x+y
[1] 9+0i
> x-y
[1] 1+10i
> x*y
[1] 45-5i
> x/y
[1] -0.121951+1.097561i
> x^y
[1] 118961-44137.3i
```

Специальные функции для комплексных переменных:

- $\text{Re}(z)$ — нахождение действительной части комплексного числа z ;
- $\text{Im}(z)$ — мнимая часть комплексного числа z ;
- $\text{Mod}(z)$ — модуль $\rho = \sqrt{x^2 + y^2}$ комплексного числа $z = x + iy$;
- $\text{Arg}(z)$ — аргумент ϕ комплексного числа $z = x + iy$, где $x = \rho \cos \phi$, $y = \rho \sin \phi$;
- $\text{Conj}(z)$ — комплексно сопряжённое к z число. К комплексным переменным можно применять и тригонометрические, и кумулятивные функции.

```
> x=2+1i*2; y=2-1i*3
> Re(x)
[1] 2
> Im(y)
[1] -3
> Mod(x)
[1] 2.828427
> Arg(x)
```

```
[1] 0.7853982
```

```
> Conj(x)
```

```
[1] 2-2i
```

```
> cos(x)
```

```
[1] -1.565626-3.297895i
```

2.2. Написание программ с циклами и условными операторами

Задание

Запустить все команды из разделов 2.2.1-2.1.6. Разобраться в особенностях применения условных операторов и циклов.

2.2.1. Оператор If

Краткий вариант оператора `if`

`If (условие) выражение`

условие — любой оператор условия (`<`, `>`, `>=`, `<=`, `==`, `!=`), результатом выполнения которого является логический вектор единичный длины.

Если значение вектора `TRUE`, то выполняется выражение. Выражение — одно или несколько выражений, выполняемых в случае верности условия. Если задано несколько выражений, то они должны быть заключены в фигурные скобки `{}` и разделяться точкой с запятой (если на одной строке). Оператор возвращает значение выражения в случае верности условия или ничего не возвращает (`NULL`).

```
> x=5; y=4
```

```
> if (x>y) { z=x+y; z }
```

```
[1] 9
```

Условие выполнено и в результате получено значение `z`.

```
> x=5; y=4
```

```
> if (x<y) w=x+y
```

```
> w
```

Ошибка: объект `'w'` не найден

В этом примере условие не выполняется, следовательно, заданное выражение не будет посчитано и объект `w` не создаётся.

Пусть `x` и `y` являются векторами одинаковой длины (10). Зададим условие: если `x` не равен `y`, то берётся отношение `x/y`. В результате должны получить вектор, чья размерность совпадает с размерностью исходных векторов. Но из каких элементов он состоит?

```
> x=1:10; y=10:1
```

```

> x
[1] 1 2 3 4 5 6 7 8 9 10
> y
[1] 10 9 8 7 6 5 4 3 2 1
> if (x<y) {x/y}
[1] 0.1000000 0.2222222 0.3750000 0.5714286 0.8333333
1.2000000
[7] 1.7500000 2.6666667 4.5000000 10.0000000

```

Получается, что взято отношение только первых элементов векторов. Кроме того, выводится предупреждение

Предупреждение

```

In if (x != y) { :
  длина условия > 1, будет использован только первый элемент

```

Таким образом, длина условия должна быть равной единице.

Полный вариант оператора `if`

```
if(условие) выражение1 else выражение2
```

Если условие выполняется, то вычисляется выражение1 (или блок выражений), в противном случае выражение2 (или блок выражений).

```

> x=5; y=4
> if(x<y) {x+y} else {x-y}
[1] 1

```

2.2.2. Оператор *Ifelse*

Функция

```
ifelse(условие, yes, no)
```

позволяет переменной в зависимости от выполнения (невыполнения) некоторого условия принимать различные значения. Отличие от оператора `if` состоит в том, что здесь условие является логическим вектором любой заданной размерности (зависит от размерности сравниваемых объектов). Порядковый номер элемента логического вектора и его значение определяет, какой элемент вектора `yes` или `no` берётся новой переменной.

Создадим два вектора x и y одинаковой длины и присвоим переменной z либо со знаком $+$ номер совпадающих элементов, либо со знаком $-$ номер несовпадающих элементов.

```
> x=c(1,3,1,5,1,7,1,9)
> y=c(2,3,4,5,2,7,1,8)
> z=ifelse(x==y,1:10,(-1):(-10))
> z
[1] -1 2 -3 4 -5 6 7 -8
```

Создадим последовательность чисел от 6 до -4 и вычислим корень квадратный.

```
> x <- c(6:-4)
> sqrt(x)
[1] 2.449490 2.236068 2.000000 1.732051 1.414214
1.000000 0.000000
[8] NaN NaN NaN NaN
```

Предупреждение

In sqrt(x) : созданы NaN

При извлечении квадратного корня из отрицательных чисел были созданы NaN и выведено предупреждение. Попробуем вычислить корень так, чтобы не было предупреждения.

```
> sqrt(ifelse(x >= 0, x, NA))
[1] 2.449490 2.236068 2.000000 1.732051 1.414214
1.000000 0.000000
[8] NA NA NA NA
```

Ещё один пример:

```
> x=rep(c(1,2,3),3)
> x=matrix(x,3,3)
> z=cos(ifelse(1<x,x*pi,x*pi/2))
> z
[,1] [,2] [,3]
[1,] 6.123032e-17 6.123032e-17 6.123032e-17
```

[2,]	1.000000e+00	1.000000e+00	1.000000e+00
[3,]	-1.000000e+00	-1.000000e+00	-1.000000e+00

2.2.3. Оператор *for*

`for(переменная in последовательность) выражение`

Оператор цикла. Пока переменная находится в рамках заданной числовой последовательности, выполняется выражение (или блок выражений).

```
> x=1:10; y=10:1
> x
[1] 1 2 3 4 5 6 7 8 9 10
> y
[1] 10 9 8 7 6 5 4 3 2 1
> w=vector(length=10,mode='numeric')
> for(i in 1:10)
+ { if (x[i]<y[i]) {w[i]=x[i]/y[i]}
+ else {w[i]=x[i]*y[i]}
+ }
> w
[1] 0.1000000 0.2222222 0.3750000 0.5714286 0.8333333 30.0000000
[7] 28.0000000 24.0000000 18.0000000 10.0000000
```

В цикле мы задали условный оператор, выполнение которого означает, что элемент вектора w равен отношению x/y , невыполнение условия приводит к появлению элемента $w = xy$. Отметим, что до цикла нужно задать вектор w , иначе будет выдано сообщение об ошибке.

```
> x=1:10; y=10;1
> for(i in 1:10)
+ { if (x[i]<y[i]) {w=x[i]/y[i]}
+ else {w=x[i]*y[i]}
+ }
> w                                     # w – не вектор, а число
[1] 10
```

В результате переменной w будет присвоено значение, созданное на последнем витке цикла.

Если по каким-либо причинам последовательность, задающая число витков в цикле, имеет нулевую длину, то цикл выполняться не будет.

```
c=integer(0)
for (i in c) {m=i}
```

m

Ошибка: Объект 'm' не найден

2.2.4. Оператор while

Ещё один оператор цикла.

while (условие) выражение

Пока выполняется условие вычисляется выражение, как только результатом условия становится FALSE, выходим из цикла.

```
> x=-10
> while (x<0) {z=x; x=x+1}
> z
[1] -1
```

2.2.5. Операторы repeat, break и next

Оператор repeat

repeat выражение

создаёт бесконечный цикл, в котором вычисляется выражение (или блок выражений). Для выхода из этого цикла нужно использовать break, а также условный оператор в качестве одного из выражений.

Построим цикл при помощи оператора repeat, при помощи условного оператора if и break зададим выход. Результатом работы оператора repeat будет вычисление натурального логарифма абсолютного значения переменной t.

```
> t=-10
> repeat{
+ if (t>0) break
+ f=log(abs(t))
+ t=t+1}
> f
[1] -Inf
```

Как только t стало больше 0 выходим из цикла с помощью оператора break и получаем последнее вычисленное значение $f = \ln |t|$, равное $f = \ln 0 = -\infty$.

Оператор `next` как и `break` может применяться только внутри циклов. Отличие от `break` состоит в том, что происходит не прерывание цикла, а переход на следующий виток.

2.2.6. Оператор `switch`

Оператор

`switch`(управляющее выражение, альтернативные действия) позволяет выполнять одну из нескольких операций в зависимости от результатов управляющего выражения.

Управляющее выражение возвращает:

- либо целое число (от 1 до числа альтернатив), которое является номером выполняемого действия;
- либо символьную переменную (строку), соответствующую имени выполняемой операции.

Если возвращаемое контрольным выражением значение не соответствует ни номеру выполняемой операции, ни имени, то результатом оператора `switch` будет `NULL`. Оператор `switch` не является самостоятельным, используется либо внутри функций, либо внутри других управляющих конструкций.

Создадим числовой вектор x длины 5, элементы которого принимают значения в зависимости от значения i — либо $\cos \pi$, либо e , либо $\log_2 4$, либо $\lg_{10}(0.01)$, либо, наконец, число, соответствующее логическому значению `TRUE`.

```
> x=numeric(5)
> for (i in 1:5)
+                                     {
x[i]=switch(i,cos(pi),exp(1),log2(4),log10(0.01),TRUE) }
> x
[1] -1.000000 2.718282 2.000000 -2.000000 1.000000
```