Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 2

3. Структуры языка R. Пользовательские функции.

Решение задач линейной алгебры с помощью R.

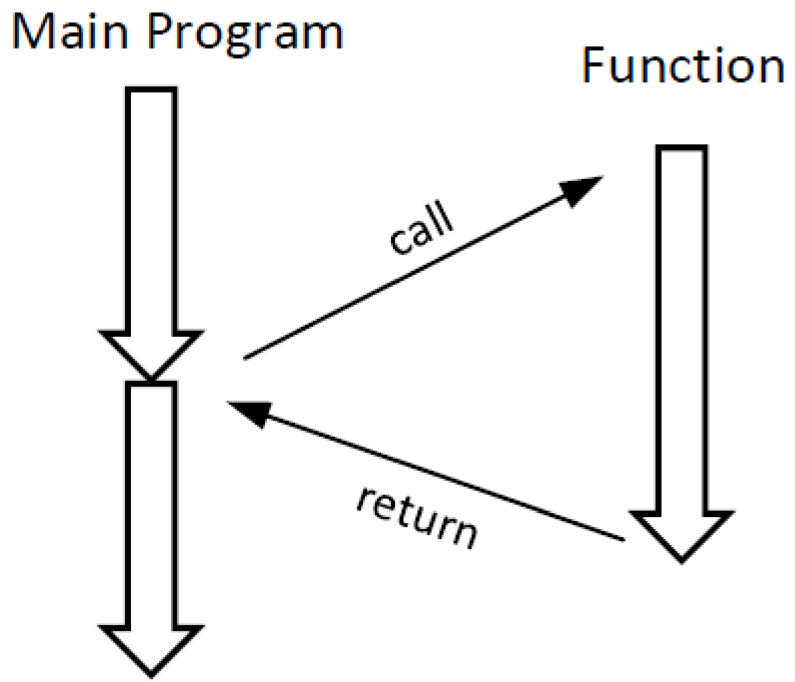
Составитель - Сысоев А.С., к.т.н., доц.

Липецк – 2018

Outline

3.1. Функции. Пользовательские функции

Функции создаются для того, чтобы выполнять определенные стандартные действия более одного раза.



Примеры: 1) стандартные функции для работы с векторами и матрицами

```
sum(x)  mean(x) max(x)  min(x)
```

2) стандартные тригонометрические функции

```
cos(x)  sin(x) tan(x)   ...
```

Тригонометрические функции определены для углов в радианах!!!

```
sin(pi/2)   sin(90)
```

3.1. Функции. Пользовательские функции

3.1.1. Синтаксис

R предоставляет большое количество встроенных функций с возможностью гибкой пользовательской настройки.

ФУНКЦИЯ в R **может** содержать

- ✓ **Имя**, через которое происходит обращение (не должно совпадать с именами существующих функций)
- ✓ **Тело функции**, содержащие основные операции
- ✓ **Аргументы**, определяющие основные параметры ее выполнения
- ✓ **Значение**, которое необходимо вернуть в результате выполнения функции

Простой пример:

```
f <- function(x,y) {
  return(2*x + y^2)
}
f(-3, 5)

## [1] 19
```

3.1. Функции. Пользовательские функции

3.1.1. Синтаксис

СИНТАКСИС

```
functionname <- function(argument1, argument2, ...) {
  function_body
  return(value)
}
```

- ✓ Возвращаемое значение (по умолчанию) - результат, последнего выполненного действия
- ✓ В случае, если функция содержит только одно выражение, фигурные скобки могут быть опущены (однако этого делать не рекомендуется)
- ✓ Порядок аргументов имеет значение
- ✓ Промежуточные значения вычислений внутри функций не выводятся в консоль, однако, `print(...)` может решить эту проблему

```
function_0 <- function(x,y,z){
     x <- y + 3;
     z <- x^3;
     k <- x + y + z;
     return(k)}
> function_0(10,3,4)
[1] 225
```

```
function_1 <- function(x,y,z){
     x <- y + 3;
     z <- x^3; prom <- c(x,z); print(prom);
     k <- x + y + z;
     return(k)}
> function_1(10,3,4)
[1]   6 216
[1] 225
```

3.1. Функции. Пользовательские функции

3.1.2. Примеры функций

```
square <- function(x) x*x
> square(5)
[1] 25
```

```
cubic <- function(x){
    print(c("Значение: ", x, "Куб: ", x*x*x))}
> cubic(5)
[1] "Значение: " "5     "Куб: "     "125"
```

```
hello <- function() print("Привет!")
> hello()
[1] "Привет!"
```

```
my_mean <- function(x){
     return(sum(x)/length(x))}
> my_mean(seq(10,20))
```

Переменные, созданные в функции, существуют только в ней, они **ЛОКАЛЬНЫ**.

```
> x <- "A"
g <- function(x){
    x <- "B"; return(x)}
> x <- "C"
> g(x)             [1] ???
> x                [1] ???
```

ПРИМЕР ПОЛЬЗОВАТЕЛЬСКОЙ ФУНКЦИИ

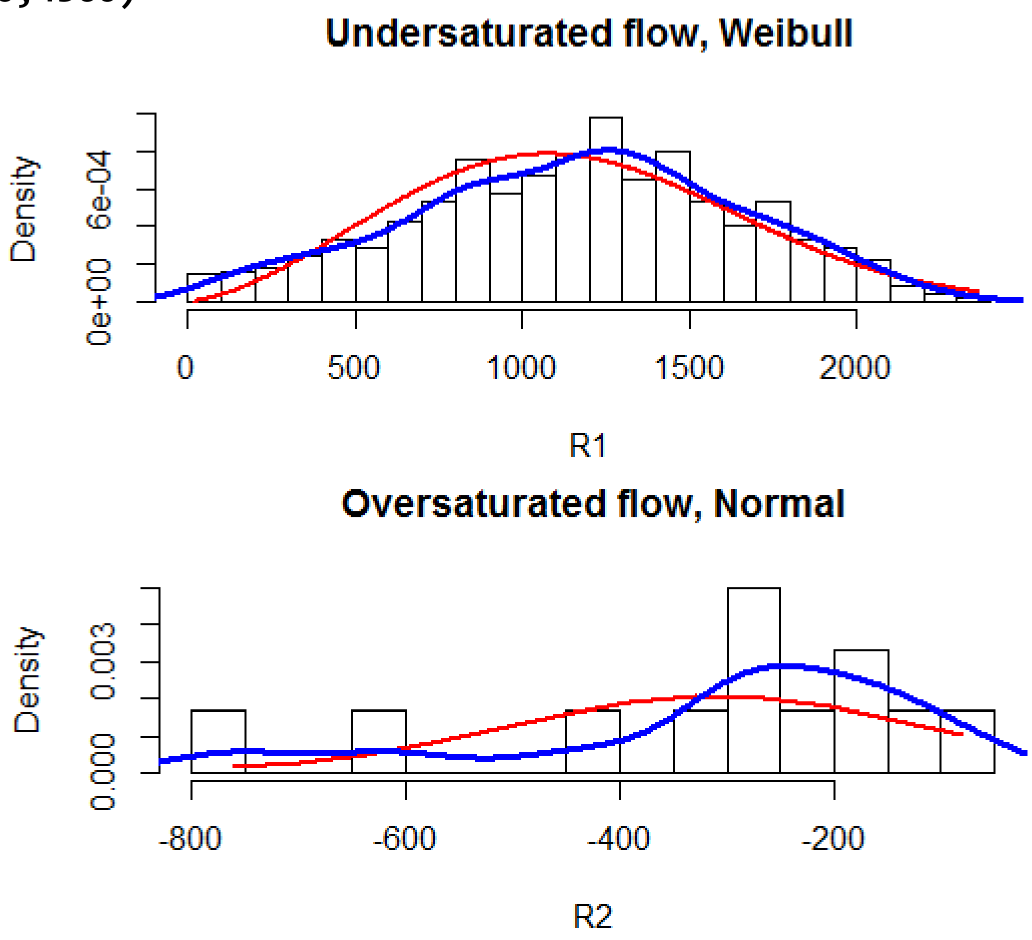
Задача: На автомагистрали могут возникать транспортные заторы. Существует множество программных продуктов, способных моделировать это. У автомагистрали есть определенная пропускная способность, экспериментально доказано. что это случайное значение, которое можно описать распределением Вейбулла. Определить, какое распределение имеет резерв транспортного потока (разность пропускной способности и интенсивности поступления).

3.1. Функции. Пользовательские функции

3.1.2. Примеры функций

```
reserve<-function(num, fl_rate, c_hbs){
    library(nortest); library(MASS);
    flow_rate<-fl_rate*rnorm(n=num,mean=1,sd=4/sqrt(fl_rate));
    capacity<-rweibull(n=num,shape=15,scale=1.275*c_hbs);
    raz<-capacity-flow_rate;
    raz.1 <- split(raz[raz>0],cumsum(raz==0)[raz>0]); R1<-sapply(raz.1, as.numeric);
    raz.2 <- split(raz[raz<0],cumsum(raz==0)[raz<0]); R2<-sapply(raz.2, as.numeric);
    opar <- par(no.readonly=TRUE); par(mfrow=c(2,1));
    hist(R1, freq=FALSE, breaks=20, main="Undersaturated flow, Weibull");
    xfit<-seq(min(R1), max(R1), length=length(R1));
    fit.weibull <- fitdistr(R1, "weibull");
    yfit<-dweibull(xfit, shape= fit.weibull$estimate["shape"],
    + scale=fit.weibull$estimate["scale"]);
    lines(xfit, yfit, lwd=2, col="red"); lines(density(R1), col="blue", lwd=3);
    hist(R2, freq=FALSE, breaks=20, main="Oversaturated flow, Normal");
    xfit<-seq(min(R2), max(R2), length=length(R2));
    fit.normal <- fitdistr(R2, "normal");
    yfit<-dnorm(xfit, mean=fit.normal$estimate["mean"], sd=fit.normal$estimate["sd"])
    lines(xfit, yfit, lwd=2, col="red"); lines(density(R2), col="blue", lwd=3);
    par(opar); print("Undersaturated flow, Weibull distribution"); print(fit.weibull);
    print(ks.test(R1, "pweibull", scale=fit.weibull$estimate["scale"],
    + shape=fit.weibull$estimate["shape"]));
    print("Oversaturated flow, Normal distribution");
    print(fit.normal); print(lillie.test(raz)) }
```

```
> reserve(500,4500,4560)
```



3.1. Функции. Пользовательские функции

3.1.2. Примеры функций

```
> reserve(500,4500,4560)
[1] "Undersaturated flow, Weibull distribution"
      shape          scale
  2.550446e+00   1.305352e+03
 (9.459161e-02) (2.424581e+01)

        One-sample Kolmogorov-Smirnov test

data:  R1
D = 0.055762, p-value = 0.09616
alternative hypothesis: two-sided

[1] "Oversaturated flow, Normal distribution"
      mean          sd
  -310.62464    192.93086
 (  55.69434) (  39.38185)

        Lilliefors (Kolmogorov-Smirnov) normality test

data:  raz
D = 0.045931, p-value = 0.01357
```

3.2. Математические выражения

В R выражения могут хранить в себе символьные математические структуры, которые в последствии могут быть модифицированы (например, вычислены их частные производные).

Для задания такого рода объекта существует функция `expression()`

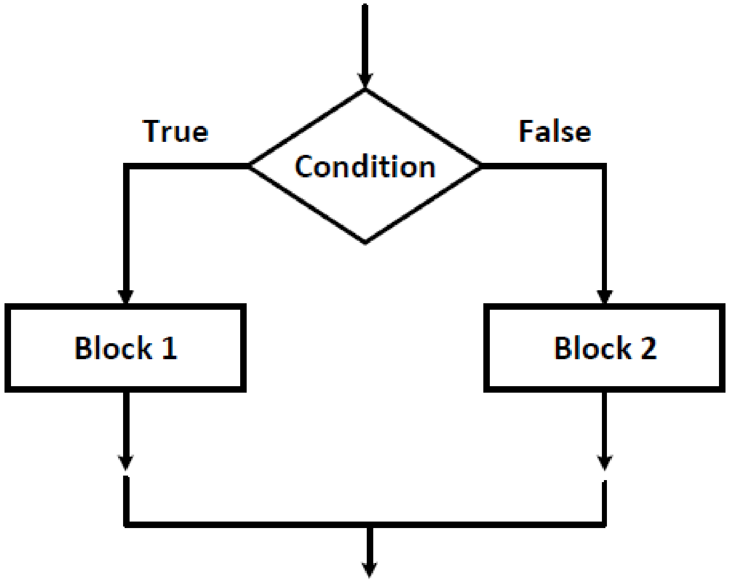
```
f <- expression(x^3 + 3*y - y^3 - 3*x)
> f
expression(x^3 + 3 * y - y^3 - 3 * x)
```

Если необходимо вычислить значение созданного выражения при определенных значениях переменных, входящих в него, используют функцию `eval()`

```
> x <- 2
> y <- 3
> eval(f)
[1] -16
```

3.3. Циклы

3.3.1. Условие If-Else. Использование ifelse



Условие If

```
if (condition) {
  statement1
}
```

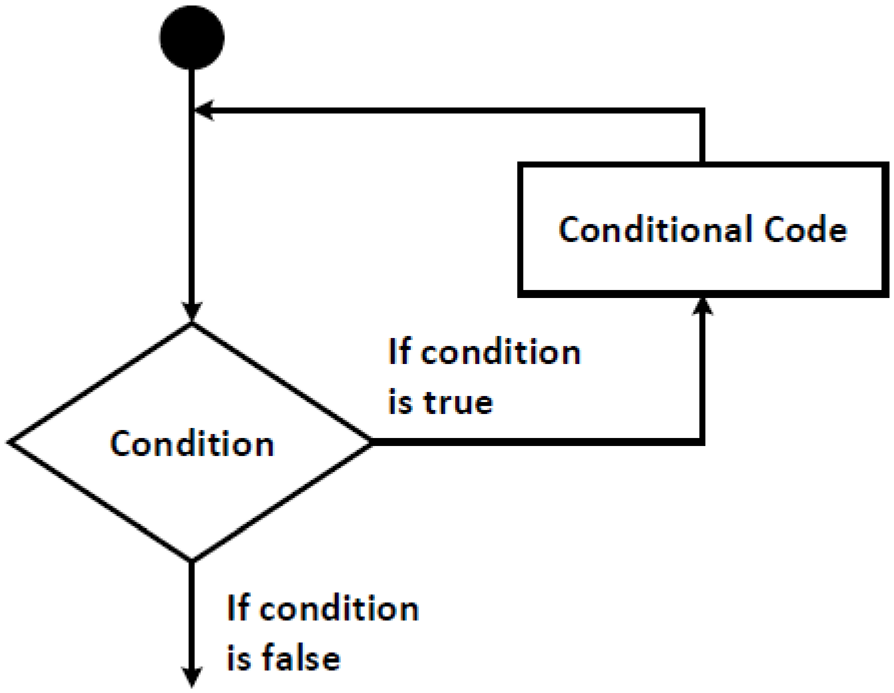
Условие If-Else

```
if (condition) {
  statement1
} else {
  statement2
}
```

```
grades <- c(1, 2, 3, 4, 5)
ifelse(grades <= 4, "Passed", "Failed")

## [1] "Passed" "Passed" "Passed" "Passed" "Failed"
```

3.3. Циклы

3.3.2. Цикл For




```
for (i in 4:7) {
  print(i)
}

## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

```
a <- c()
for (i in 1:3){
  a[i] <- sqrt(i)
}
a

## [1] 1.000000 1.414214 1.732051
```

```
for (counter in looping_vector){
  # code to be executed for each element in the sequence
}
```

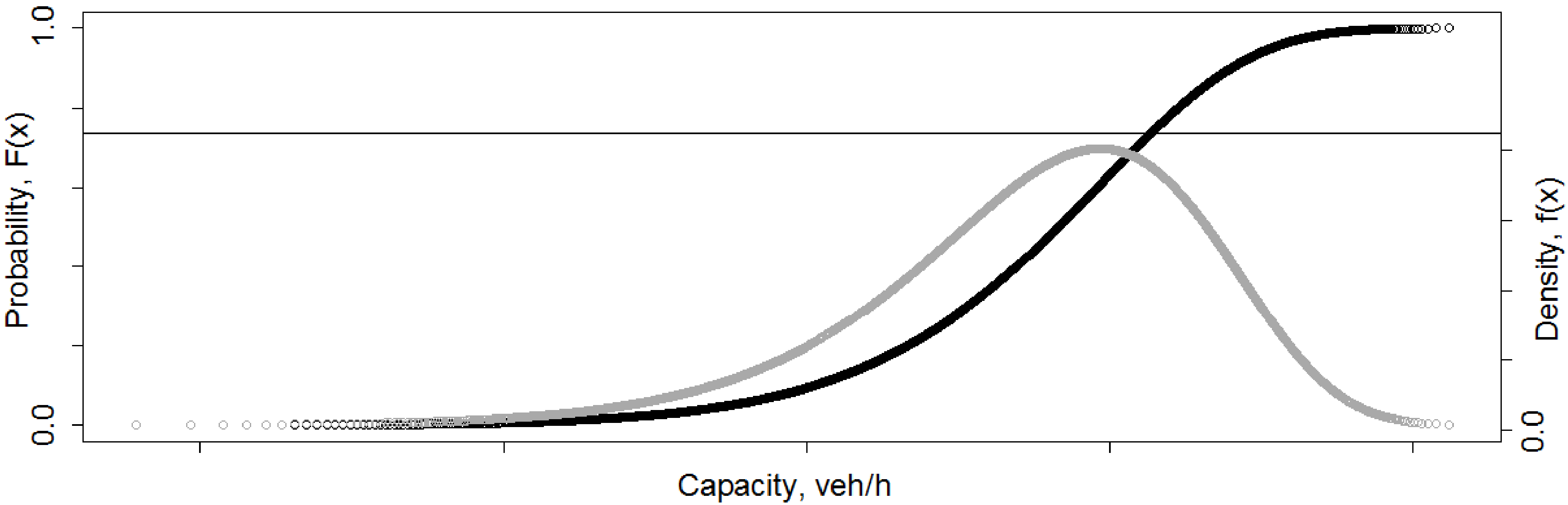
```
> z <- 1
> while(z <= 4){
   print(z);
   z <- z + 1}
[1] 1
[1] 2
[1] 3
[1] 4
```

- ✓ Количество итераций определяется выполнением заданного условия
- ✓ Условия проверяется в начале каждой итерации

```
while (condition) {
  # code to be executed
}
```

3.3.4. Пример сложной функции

<u>Задача:</u> При моделировании потока транспортных средств на автомагистрали используется распределение Вейбулла пропускной способности полосы. Но при компьютерном моделировании необходимо найти такие наименьшее и наибольшее значения случайного равномерно распределенного числа, используемого для моделирования, чтобы получить адекватные значения смоделированной пропускной способности.



3.3.4. Пример сложной функции

```
poisk <- function(lambda,k,h1){
B=0;i=0; ... # определяем локальные переменные
    for(i in 1:((1/h)-h)){
        point <- point + h;  value[i] <- lambda*(-log(1-point))^(1/k);  prob[i] <- i*h}
    for(i in 1:length(value)){
    z[i] <- k*(lambda^(-k))*(value[i]^(k-1))*exp(-(lambda/value[i])^(-k));
    t[i] <- (k*exp(-(value[i]/lambda)^k)*((value[i]/lambda)^k)*
        *(k-1-((value[i]/lambda)^2)*k))/(value[i]^2)}
    low <- which(t==max(t)); upp <- which(t==min(t));
    for(i in 1:length(value)){
    x[i] <- (k*exp(-(value[i]/lambda)^k)*(exp(-(value[i]/lambda)^k)*k^2-
        -3*exp(-(value[i]/lambda)^k)*k-3*exp(-(value[i]/lambda)^2*k)*k^2+2*exp(-
        (value[i]/lambda)^k)+3*exp(-(value[i]/lambda)^2*k)*k+exp(-
        -(value[i]/lambda)^3*k)*k^2))/(value[i]^3)}
    for(i in 2:((length(value)))){
        diff[i] <- x[i]-x[i-1]; diff1[i-1] <- diff[i]; ind[i] = i; ind[1] = 1;
        diff1[length(value)]=diff1[length(value)-1]}
    fit_power <- nls(diff1 ~ a * ind^b, start = list(a=0.1, b=0.1))
    for(i in 2:length(ind)){
        deriv1[i] <- coef(fit_power)[1]*coef(fit_power)[2]*ind[i]^(coef(fit_power)[2]-1)
        deriv2[i] <- coef(fit_power)[1]*coef(fit_power)[2]*(coef(fit_power)[2]-1)*
                *ind[i]^(coef(fit_power)[2]-2);
        curve[i] <- abs(deriv2[i])/((1+deriv1[i]^2)^1.5)}
## продолжение далее
```

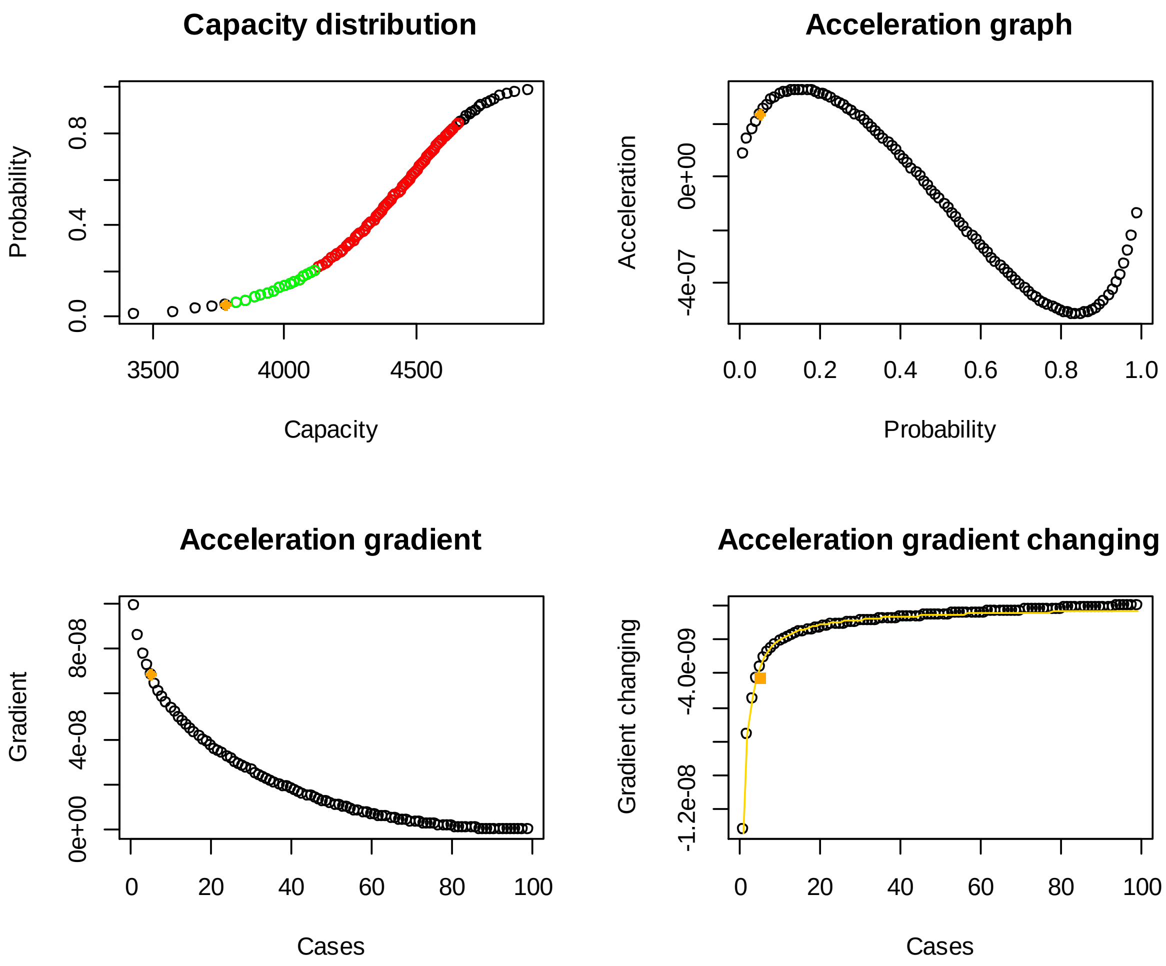
```
## продолжение

    sred <- mean(curve)
    for(i in 1:length(curve)){
        if(abs(curve[i]-sred) >
        0.1^abs(ceiling(log10(mean(curve)))))
             {number <- ind[i]} }
a1 <- value[low];
a2 <- pweibull(value[low], k, lambda);
a3 <- value[upp];
a4 <- pweibull(value[upp], k, lambda);
a5 <- value[number];
a6 <- pweibull(value[number], k, lambda)
B <- c(a1,a3)
return(B)
}
```

```
poisk(4500,17,100)
[1] "lower theshold"
[1] 4043.851
[1] "lower probability rate"
[1] 0.15
[1] "adjusted lower theshold"
[1] 3778.624
[1] "adjusted lower probability rate"
[1] 0.05
[1] "upper theshold"
[1] 4663.23
[1] "probability rate"
[1] 0.84
```

```
poisk(4500, 17, 100)
[1] 4043.851 4663.230
```

3.3. Циклы

3.3.4. Пример сложной функции



3.4. Задачи линейной алгебры

3.4.1. Произведение векторов и матриц. Единичная матрица

- Произведение вектора или матрицы на скаляр

```
5*c(1, 2, 3)

## [1]  5 10 15
```

```
m <- matrix(c(1,2, 3,4, 5,6), ncol=3)
```

- Произведение векторов или матриц

```
m %*% x

##      [,1]
## [1,]   22
## [2,]   28
```

- Единичная матрица

```
diag(3)

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

- Обратная матрица

```
sq.m <- matrix(c(1,2, 3,4), ncol=2)
sq.m
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
solve(sq.m)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
sq.m %*% solve(sq.m) - diag(2) # post check
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

- Псевдообратная матрица

```r
library(MASS)
```

```r
ginv(m)
```

```
##                [,1]       [,2]
## [1,] -1.3333333  1.0833333
## [2,] -0.3333333  0.3333333
## [3,]  0.6666667 -0.4166667
```

```r
m %*% ginv(m)
```

```
##                  [,1] [,2]
## [1,] 1.000000e+00    0
## [2,] 2.664535e-15    1
```

3.4.3. Определитель матрицы. Собственные числа и собственные векторы

```r
det(sq.m)

## [1] -2
```

```r
sq.m

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

e <- eigen(sq.m)
e$val # eigenvalues

## [1]  5.3722813 -0.3722813

e$vec # eigenvectors

##            [,1]       [,2]
## [1,] -0.5657675 -0.9093767
## [2,] -0.8245648  0.4159736
```

Матрица Q является *отрицательно определённой*, если $-Q$ есть положительно определённая матрица.

Матрица Q является *отрицательно полуопределённой*, если $-Q$ есть положительно полуопределённая матрица.

Матрица Q является *неопределённой*, если квадратичная форма x^TQx может принимать как положительные, так и отрицательные значения.

$$Q = \begin{pmatrix} 1 & -1 \\ 1 & -2 \end{pmatrix}, \quad \varphi(x) = (x_1, x_2)\begin{pmatrix} 1 & -1 \\ 1 & -2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} =$$
$$= (x_1, x_2)\begin{pmatrix} x_1 - x_2 \\ x_1 - 2x_2 \end{pmatrix} = x_1(x_1 - x_2) + x_2(x_1 - 2x_2) =$$
$$= x_1^2 - x_1x_2 + 2x_2^2 = x_1^2 - 2x_2^2.$$

Для проверки определённости (полуопределённости) матрицы служат *критерии Сильвестра*:

1. Матрица положительно определена, если:
   а) все диагональные элементы положительны;
   б) все угловые миноры матрицы положительны.
2. Матрица отрицательно определена, если:
   а) все диагональные элементы отрицательны;
   б) все угловые миноры матрицы имеют чередующиеся знаки, начиная со знака «–».
3. Матрица положительно полуопределена, если значения диагональных элементов и главных миноров матрицы неотрицательны.

```r
library(matrixcalc)
```

```r
I <- diag(3)
I
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
is.negative.definite(I)
```

```
## [1] FALSE
```

```r
is.positive.definite(I)
```

```
## [1] TRUE
```

```r
C <- matrix(c(-2,1,0, 1,-2,1, 0,1,-2),
            nrow=3, byrow=TRUE)
C
```

```
##      [,1] [,2] [,3]
## [1,]   -2    1    0
## [2,]    1   -2    1
## [3,]    0    1   -2
```

```r
is.positive.semi.definite(C)
```

```
## [1] FALSE
```

```r
is.negative.semi.definite(C)
```

```
## [1] TRUE
```

3.4.5. Гессиан. Аппроксимация функции разложением в ряд Тейлора

$$H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\boldsymbol{x}) \end{bmatrix}$$

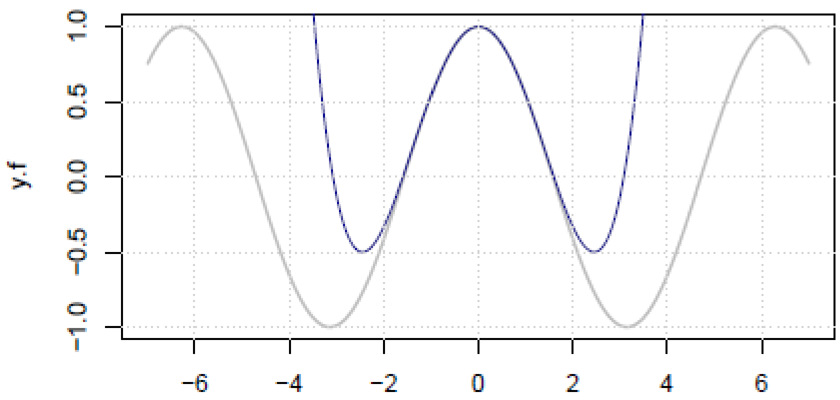
```r
f <- function(x) (x[1]^3*x[2]^2-x[2]^2+x[1])
optimHess(c(3,2), f, control=(ndeps=0.0001))

##       [,1] [,2]
## [1,]    72  108
## [2,]   108   52
```

3.4.5. Гессиан. Аппроксимация функции разложением в ряд Тейлора

```
library(pracma)
```

```
f <- function(x) cos(x)
taylor.poly <- taylor(f, x0=0, n=4)
taylor.poly

## [1]  0.04166733  0.00000000 -0.50000000  0.00000000  1.00000000
```

```
x <- seq(-7.0, 7.0, by=0.01)
y.f <- f(x)
y.taylor <- polyval(taylor.poly, x)
plot(x, y.f, type="l", col="gray", lwd=2, ylim=c(-1, +1))
lines(x, y.taylor, col="darkblue")
grid()
```



Список литературы

Кабаков Р. К. (2014) R в действии. Анализ и визуализация данных на языке R Издательство: ДМК Пресс, 580 с.

Numerical Analysis // Computational Ecomonics Practice by Stefan Feuerrigel (Freiberg Universitaet)