Chapter 7 Mathematics | Chapter 8 Classical Tests

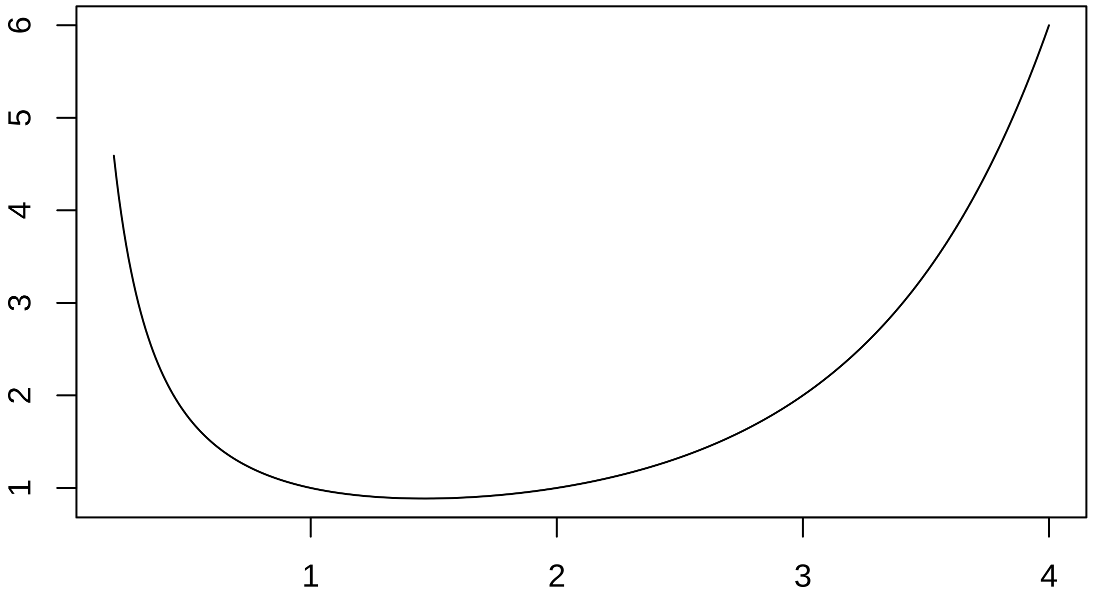
Qianqian Shan

May 26, 2017

Mathematical functions

1. **Logarithmic functions** : $y = a \cdot ln(bx)$ and $y = a \cdot e^{bx}$.
2. **Trigonometric functions** : $sin, cos, tan \cdots$.
3. **Power laws** : $y = a \cdot x^b$, different b values can result in different shapes.
4. **Polynomial functions** : useful for describing curves with humps, inflections, local maxima …
5. **Gamma function** : $\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$ with plot

```
plot(seq(0.2, 4, 0.01), gamma(seq(0.2, 4, 0.01)), type = "l",
     xlab = "", ylab = "")
```



6. **Asymptotic functions** :

- $y = \frac{ax}{1+bx}$, extreme values at $x = 0$ or $x = \infty$.
- Asymptotic exponential, $y = a(1 - e^{-bx})$, with asymptotic value is a when x goes to ∞.

7. **Sigmoid(S-shaped) functions** :

- two parameter logistic: $y = \frac{e^{a+bx}}{1+e^{a+bx}}$, central to the generalized linear models.
- three parameter logistic: $y = \frac{a}{1+be^{-cx}}$, allows y to vary on any scale.
- four parameter logistic: $y = a + \frac{b-a}{1+e^{c(d-x)}}$, it has a as left asympototes, b as right asympototes, c as scale, d as midpoint.
- **Gompertz growth model** : $y = ae^{be^{cx}}$, much used in demography and life insurance work, the shape depends on the signs of the parameters b and c.

8. **Bi-exponential model** : $y = ae^{bx} + ce^{dx}$

Summary for usefull transformations

- $log(y)$ against x for **exponential** relationships.
- $log(y)$ against $log(x)$ for power functions.
- $exp(y)$ against x for logarithmic relationships.
- $1/y$ agains $1/x$ for asympototic relationships.
- $log(\frac{p}{1-p})$ against x for proportional data.
- $\sqrt{y}$ to stabilize the variance for count data.
- $arcsin(y)$ to stablize hte variance of percentage data.

7.3 Probability functions , ignored

7.4 Discrete probability distributions , ignored

7.5 Matrix algebra

- **Determinant** : More details on linear algebra books. 1. If any row or column of a determinant is multiplied by a scaler λ, then the value of the determinant is multiplied by λ. 2. If all the elements of a row or a column are zero then the determinant $|A| = 0$. If $detA \neq 0$ then the rows and columns of A must be linearly independent, more details on contrasts on Chapter 9.
- **Inverse of a matrix** : $A^{-1} = \frac{adjA}{|A|}$ where $adjA$ is the adjoint matrix of A with $A_{ij} = (-1)^{i+j} M_{ij}$. More details here.

$(AB)^{-1} = B^{-1}A^{-1}$,
$(A^{-1})' = (A')^{-1}$,
$(A^{-1})^{-1} = A$,
$|A| = \frac{1}{|A^{-1}|}$, `ginv` from `MASS` can be used to find the inverse.

- **Eigenvalues and eigenvectors** : check wiki for more details. `eigen` is useful.

Solving systems of linear equations using matrices

`solve` is used.

```r
A <- matrix(c(3, 4, 1, 2), nrow = 2)
A
```

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

```r
kv <- matrix(c(12, 8), nrow = 2)
kv
```

```
##      [,1]
## [1,]   12
## [2,]    8
```

```r
# solve the equations
solve(A, kv) # x and y values
```

```
##      [,1]
## [1,]    8
## [2,]  -12
```

Calculus

- `D` and `integrate` for derivative and integration
- differential equations by `deSolve`

```
# use D
D(expression(x^2), "x")
```

```
## 2 * x
```

```
# use deriv
## formula argument :
dx2x <- deriv(~ x^2, "x") ; dx2x
```

```
## expression({
##     .value <- x^2
##     .grad <- array(0, c(length(.value), 1L), list(NULL, c("x")))
##     .grad[, "x"] <- 2 * x
##     attr(.value, "gradient") <- .grad
##     .value
## })
```

```
mode(dx2x)
```

```
## [1] "expression"
```

```
# evaluate the drivative at specific values
x <- -1:2
eval(dx2x)
```

```
## [1] 1 0 1 4
## attr(,"gradient")
##       x
## [1,] -2
## [2,]  0
## [3,]  2
## [4,]  4
```

```
## Something 'tougher':
trig.exp <- expression(sin(cos(x + y^2)))
( D.sc <- D(trig.exp, "x") )
```

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

```
all.equal(D(trig.exp[[1]], "x"), D.sc)
```

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

```
( dxy <- deriv(trig.exp, c("x", "y")) )
```

```
## expression({
##     .expr2 <- x + y^2
##     .expr3 <- cos(.expr2)
##     .expr5 <- cos(.expr3)
##     .expr6 <- sin(.expr2)
##     .value <- sin(.expr3)
##     .grad <- array(0, c(length(.value), 2L), list(NULL, c("x",
##         "y")))
##     .grad[, "x"] <- -(.expr5 * .expr6)
##     .grad[, "y"] <- -(.expr5 * (.expr6 * (2 * y)))
##     attr(.value, "gradient") <- .grad
##     .value
## })
```

```
y <- 1
eval(dxy)
```

```
## [1]  0.8414710  0.5143953 -0.4042392 -0.8360219
## attr(,"gradient")
##               x         y
## [1,]  0.0000000  0.000000
## [2,] -0.7216061 -1.443212
## [3,] -0.8316919 -1.663384
## [4,] -0.0774320 -0.154864
```

```
eval(D.sc)
```

```
## [1]  0.0000000 -0.7216061 -0.8316919 -0.0774320
```

```
## Higher derivatives
deriv3(y ~ b0 + b1 * 2^(-x/th), c("b0", "b1", "th"),
     c("b0", "b1", "th", "x") )
```

```
## function (b0, b1, th, x)
## {
##     .expr3 <- 2^(-x/th)
##     .expr6 <- log(2)
##     .expr7 <- th^2
##     .expr9 <- .expr6 * (x/.expr7)
##     .expr10 <- .expr3 * .expr9
##     .value <- b0 + b1 * .expr3
##     .grad <- array(0, c(length(.value), 3L), list(NULL, c("b0",
##         "b1", "th")))
##     .hessian <- array(0, c(length(.value), 3L, 3L), list(NULL,
##         c("b0", "b1", "th"), c("b0", "b1", "th")))
##     .grad[, "b0"] <- 1
##     .grad[, "b1"] <- .expr3
##     .hessian[, "b1", "b1"] <- 0
##     .hessian[, "b1", "th"] <- .hessian[, "th", "b1"] <- .expr10
##     .grad[, "th"] <- b1 * .expr10
##     .hessian[, "th", "th"] <- b1 * (.expr10 * .expr9 - .expr3 *
##         (.expr6 * (x * (2 * th)/.expr7^2)))
##     attr(.value, "gradient") <- .grad
##     attr(.value, "hessian") <- .hessian
##     .value
## }
```

```
## Higher derivatives:
DD <- function(expr, name, order = 1) {
   if(order < 1) stop("'order' must be >= 1")
   if(order == 1) D(expr, name)
```

```r
   else DD(D(expr, name), name, order - 1)
}
DD(expression(sin(x^2)), "x", 3)
```

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

```r
# integrals
integrate(dnorm, 0, Inf)
```

```
## 0.5 with absolute error < 4.7e-05
```

```r
integrate(dnorm, -Inf, Inf)
```

```
## 1 with absolute error < 9.4e-05
```

```r
# integrate a self-defined function
integrate(function(x) rep(2, length(x)), 0, 1)
```

```
## 2 with absolute error < 2.2e-14
```

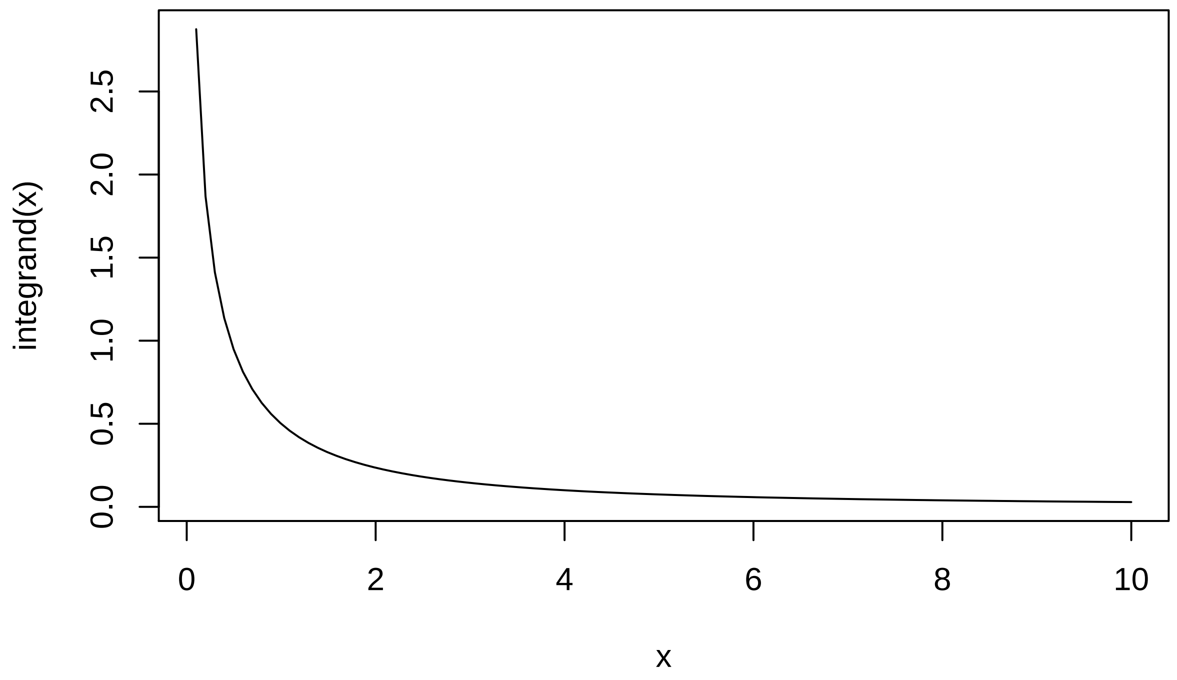
```r
integrand <- function(x) {1/((x + 1) *sqrt(x))}
integrate(integrand, 0, Inf)
```

```
## 3.141593 with absolute error < 2.7e-05
```

```r
integrand(seq(0.1, 10, 0.1))
```

```
##   [1] 2.87479787 1.86338998 1.40441681 1.12938488 0.94280904 0.80687153
##   [7] 0.70307565 0.62112999 0.55478555 0.50000000 0.45402980 0.41494133
##  [13] 0.38132957 0.35214761 0.32659863 0.30406516 0.28406111 0.26619857
##  [19] 0.25016422 0.23570226 0.22260179 0.21068746 0.19981226 0.18985212
##  [25] 0.18070158 0.17227046 0.16448125 0.15726692 0.15056929 0.14433757
##  [31] 0.13852728 0.13309928 0.12801904 0.12325594 0.11878277 0.11457528
##  [37] 0.11061175 0.10687275 0.10334075 0.10000000 0.09683623 0.09383655
##  [43] 0.09098921 0.08828357 0.08570991 0.08325936 0.08092379 0.07869577
##  [49] 0.07656847 0.07453560 0.07259138 0.07073049 0.06894798 0.06723929
##  [55] 0.06560022 0.06402684 0.06251551 0.06106285 0.05966572 0.05832118
##  [61] 0.05702650 0.05577912 0.05457665 0.05341685 0.05229764 0.05121704
##  [67] 0.05017321 0.04916442 0.04818905 0.04724556 0.04633251 0.04544854
##  [73] 0.04459236 0.04376278 0.04295863 0.04217885 0.04142241 0.04068835
##  [79] 0.03997573 0.03928371 0.03861145 0.03795817 0.03732313 0.03670561
##  [85] 0.03610497 0.03552054 0.03495173 0.03439795 0.03385866 0.03333333
##  [91] 0.03282146 0.03232257 0.03183621 0.03136193 0.03089932 0.03044798
##  [97] 0.03000754 0.02957762 0.02915788 0.02874798
```

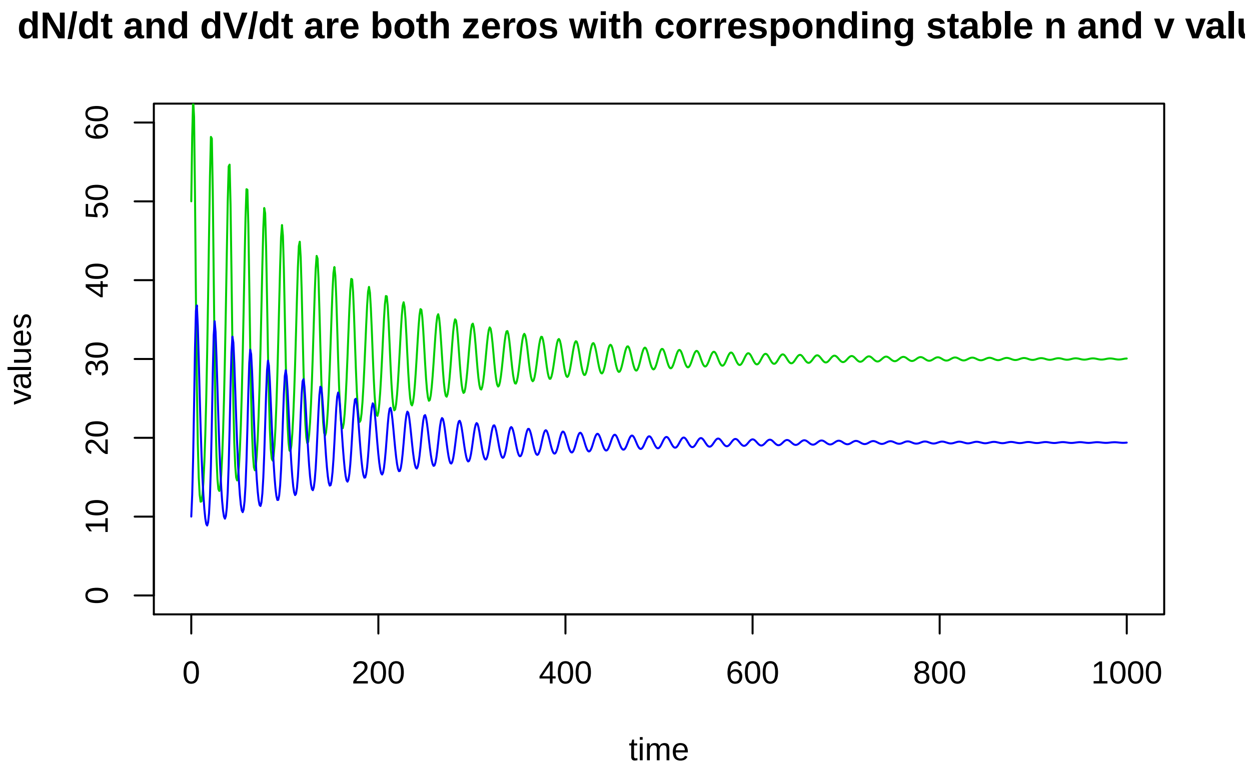
```r
curve(integrand, 0.1, 10) # area under curve is pi
```



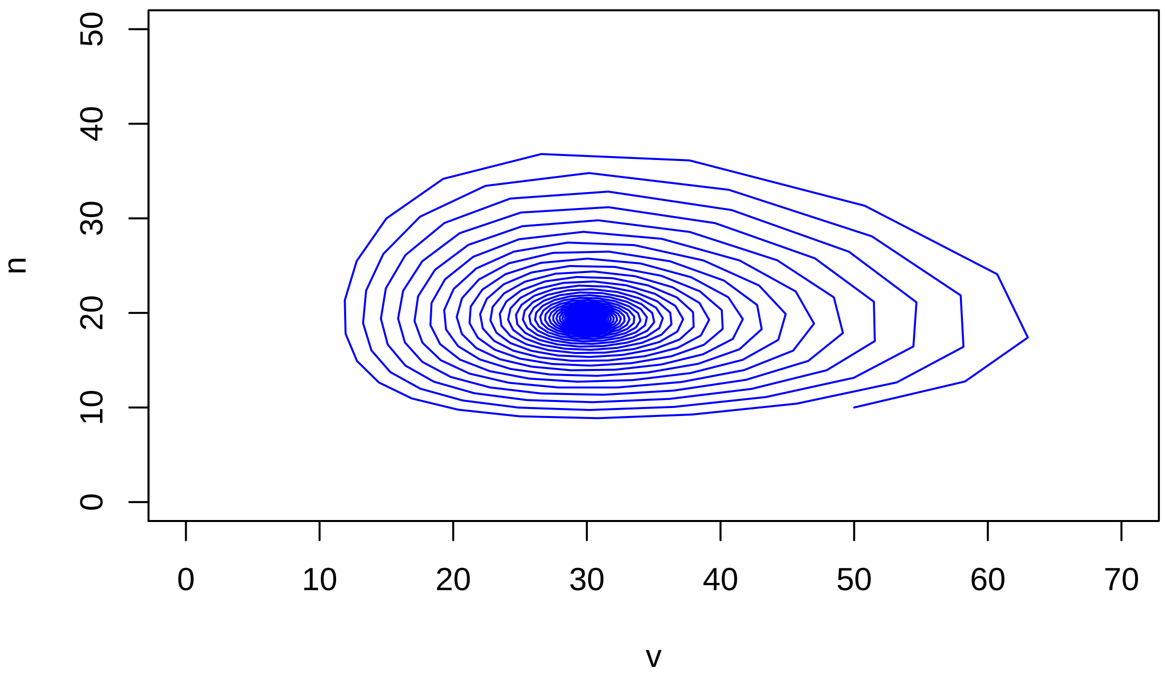
```
# install.packages("deSolve")
library(deSolve)

# 1. define a function to contain the equations

phmodel <- function(t, state, parameters){
  with(as.list(c(state, parameters)), {
    dv <- r*v*(k-v)/k - b*v*n # equation one
    dn <- c*v*n - d*n  # equation two
    result <- c(dv, dn)
    list(result)
  })
}


# 2. generate a times series over which to solve the equations and set parameters
times <- seq(0, 1000, length = 1001)
parameters <- c(r = 0.4, k = 1000, b = 0.02, c = 0.01, d = 0.3)

# set initial values for v and n
initial <- c(v = 50, n = 10)

# use ode to create a matrix with the time series of v and n
# ode : Solves a system of ordinary differential equations;
#       a wrapper around the implemented ODE solvers

output <- ode(y = initial, time = times, func = phmodel, parms = parameters)
head(output)
```

```
##      time        v        n
## [1,]    0 50.00000 10.00000
## [2,]    1 58.29220 12.75106
## [3,]    2 62.99695 17.40172
## [4,]    3 60.70065 24.09264
## [5,]    4 50.79407 31.32860
## [6,]    5 37.68312 36.12636
```

```
plot(output[, 1], output[, 2],
     ylim = c(0, 60), type = "n", ylab = "values",
     xlab = "time", main = "dN/dt and dV/dt are both zeros with corresponding stable n and v values")
lines(output[, 1], output[, 2], col = 3)
lines(output[, 1], output[, 3], col = 4)
```



```
# alternative way is to show the phase plane
plot(output[, 2], output[, 3],
     ylim = c(0, 50), xlim = c(0, 70), type = "n",
     ylab = "n", xlab = "v")
lines(output[, 2], output[, 3], col = 4)
```



Chapter 8 Classical Tests

Single samples

Summary

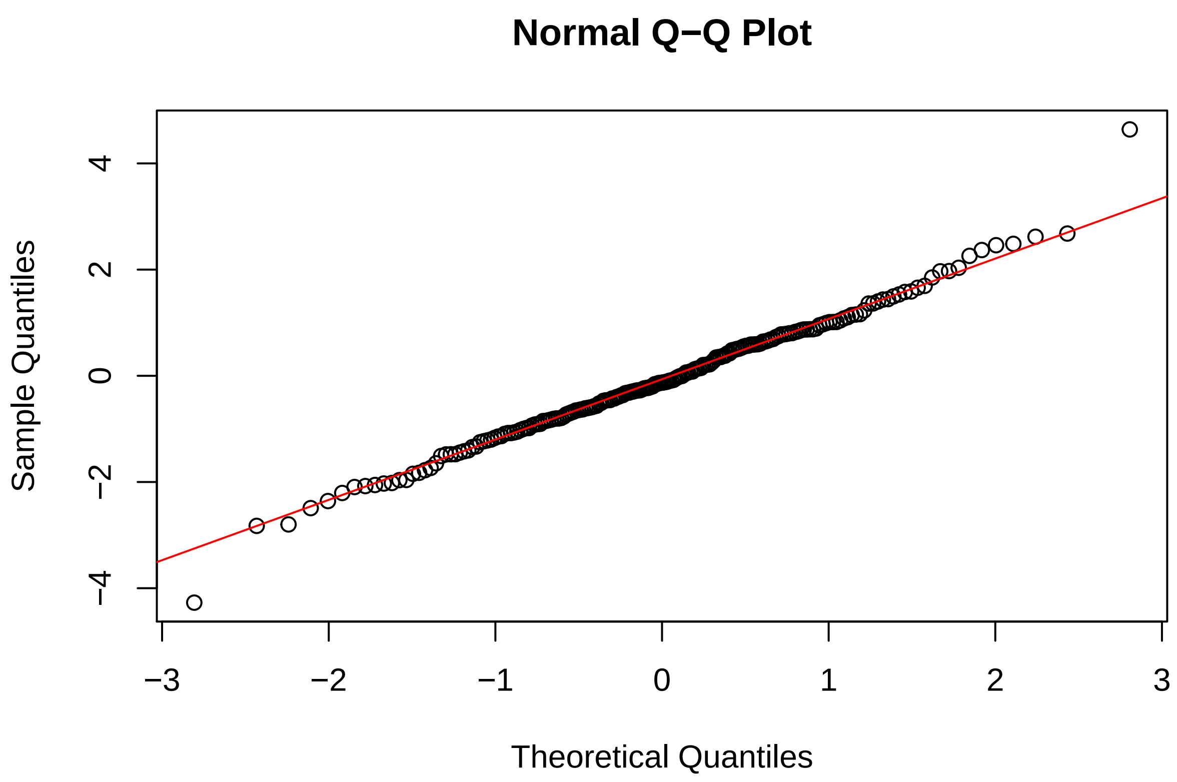
- `plot(y)` for index plot
- `hist` for histogram
- `boxplot`
- `summary`
- `fivenum(y)` for Tukey's five number
- `outliers` rule of thumb: an **outlier** is a value that is more than 1.5 times the interquartile range above the third quartile or below the first quartile.

Test for normality

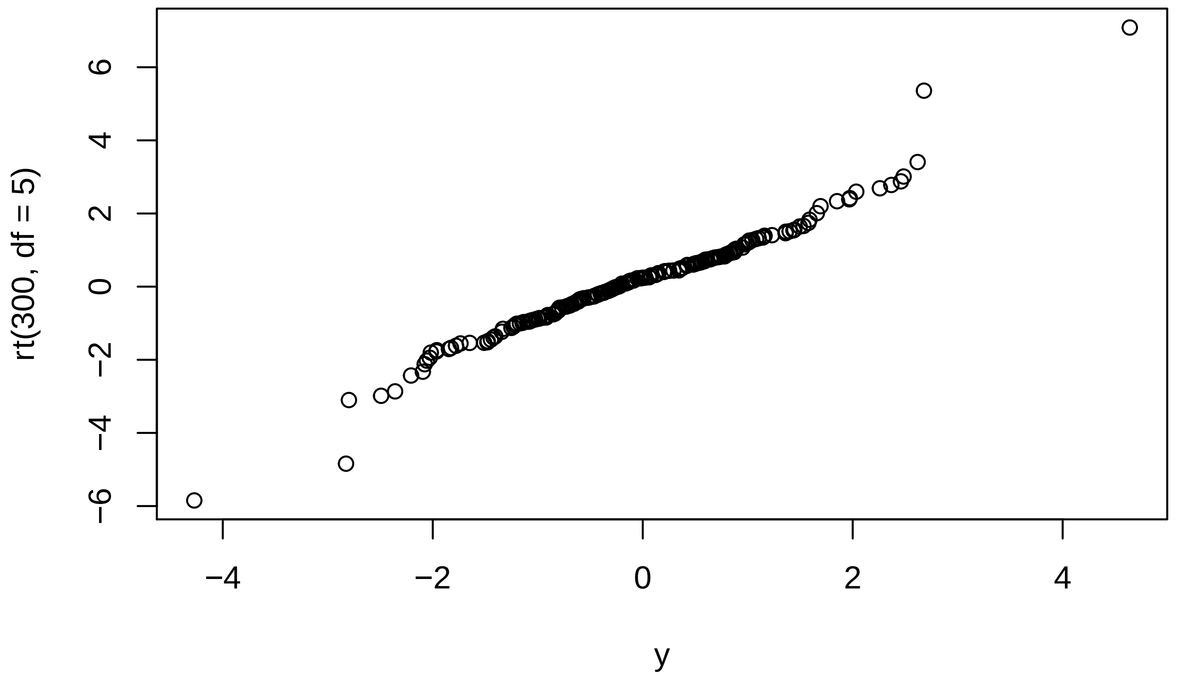
- `qqnorm` and `qqline` , `qqplot`(produce QQ plot of **two** datasets).
- `shapiro.test` for testing whether the data in a vector com from a normal distribution. Note again that the **p value** is an estimate of the probability that a particular result or a more extreme result than the observed result could have been observed.

However, p values only reflect the effect sizes, while sample sizes are equally important.

```r
y <- rt(200, df = 5)
qqnorm(y); qqline(y, col = 2)
```



```r
qqplot(y, rt(300, df = 5))
```



```r
shapiro.test(y) # reject the null that the sample data are normally distributed
```

```
##
##  Shapiro-Wilk normality test
##
## data:  y
## W = 0.98997, p-value = 0.1769
```

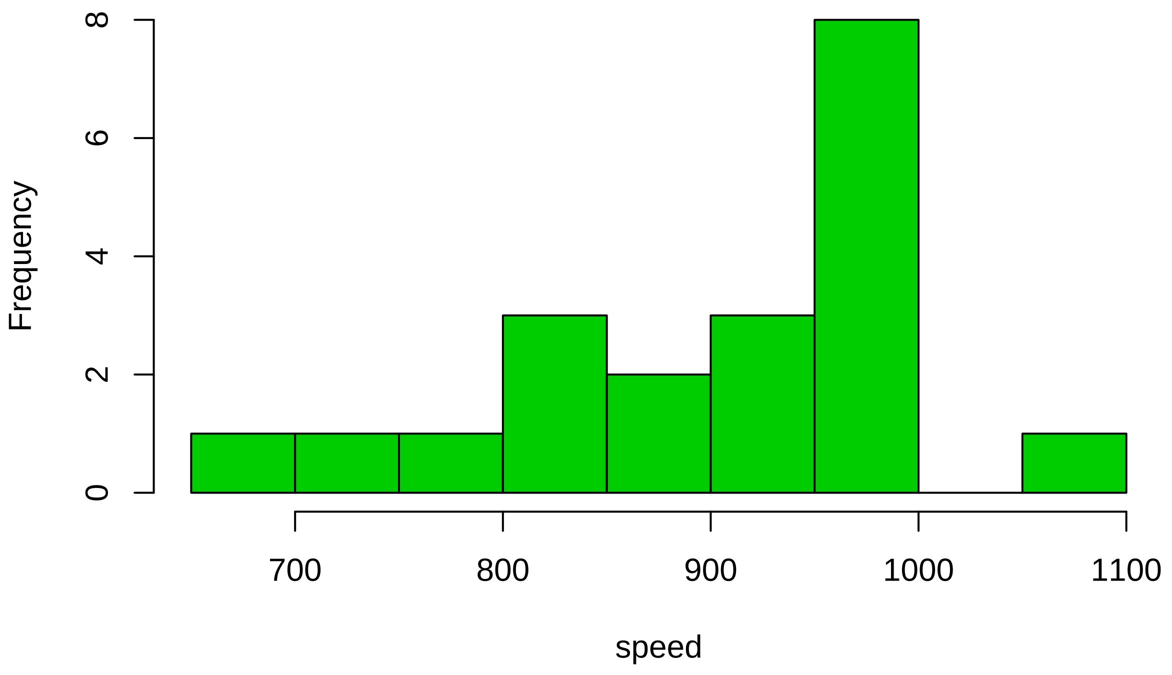
```r
# example

light <- read.table("light.txt", header = TRUE)
attach(light)
names(light)
```

```
## [1] "speed"
```

```r
length(speed)  # only 20 samples
```

```
## [1] 20
```

```r
hist(speed, main = "", col = 3)  # not normal
```



```r
# as it's not normal, use wilcoxon singed-rank test to test if the speed is significantly different fro
wilcox.test(speed, mu = 990) # reject the null
```

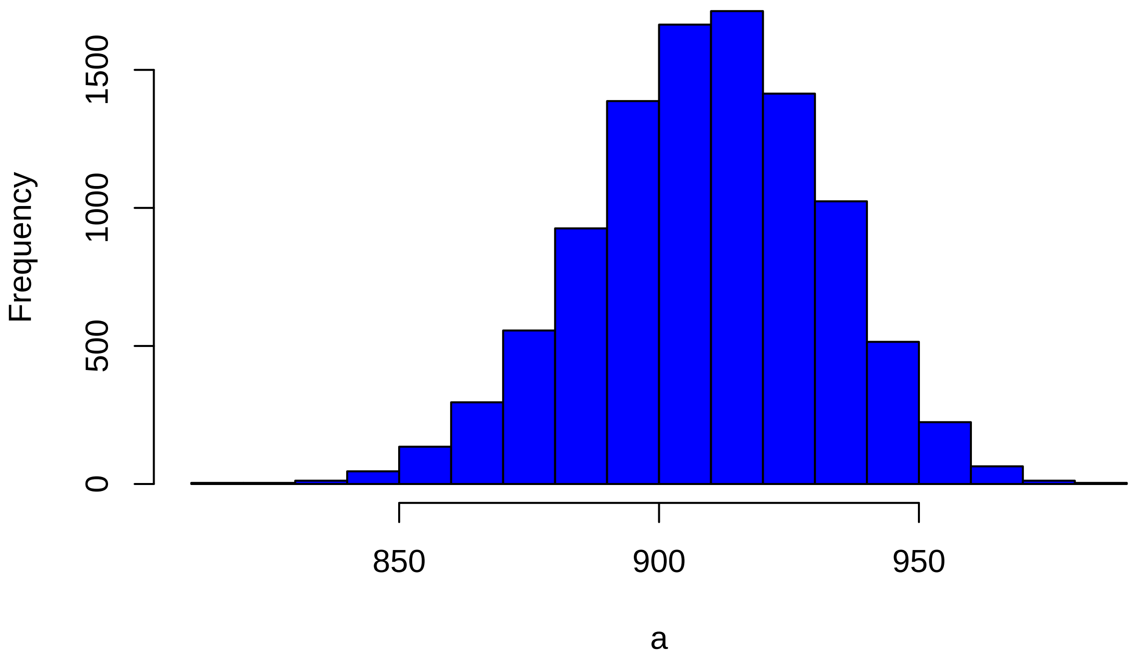
```
## Warning in wilcox.test.default(speed, mu = 990): cannot compute exact p-
## value with ties

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  speed
## V = 22.5, p-value = 0.00213
## alternative hypothesis: true location is not equal to 990
```

Bootstrap in hypothesis testing

```r
# test if the mean is as big as 990
mean(speed)
```

```
## [1] 909
```

```r
a <- numeric(10000)
for(i in 1:10000) a[i] <- mean(sample(speed, replace = TRUE))
hist(a, main = "", col = "blue")
```



```r
detach(light)
```

Skew and Kurtosis

Skew(ness) is the dimensionless version of the third moment about the mean, it measures the extent to which a distribution has long, drawn-out tails on one side or the other:

$skew = \gamma_1 = \frac{m_3}{s_3}$, where $m_3 = \frac{\sum (y-\overline{y})^3}{n}$ and $s_3 = (sd(y))^3 = (\sqrt{s^2})^3$. Negative values means skew to the left.

- Approximate standard error of skew $se_{\gamma_1} = \sqrt{\frac{6}{n}}$.

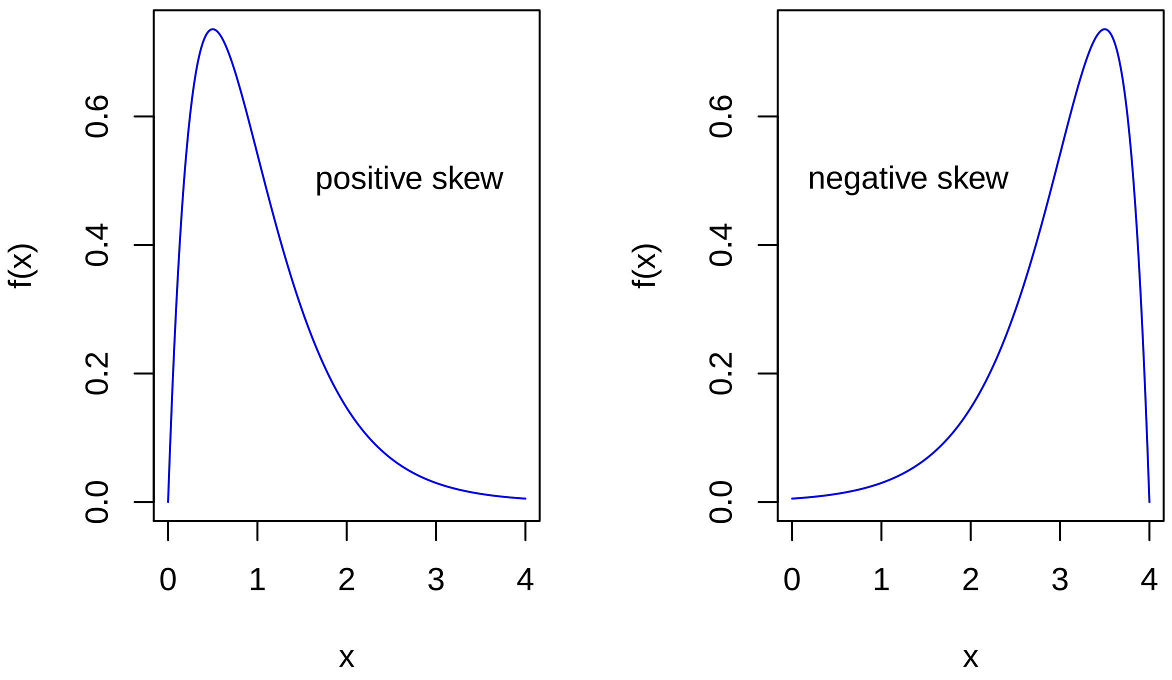
Kurtosis is the dimensionless version of the fourth moment about the mean, it meansures the non-normality that has to do with the peakyness, or flat-toppedness of a distribution.

- A more flat-topped distribution is **platykurtic**, a more pointy distribution is **leptokurtic**. Plots shown below.

$kurtosis = \gamma_2 = \frac{m_4}{s_4} - 3$, where $m_4 = \frac{\sum (y-\overline{y})^4}{n}$, $s_4 = (var(y))^2 = s^4$ and -3 is included as the normal distribution has kurtosis 3.

- Approximate standard error of kurtosis $se_{\gamma_2} = \sqrt{\frac{24}{n}}$.

```r
# positive and negative skew
par(mfrow = c(1, 2))
x <- seq(0, 4, 0.01)
plot(x, dgamma(x, 2, 2), type = "l", ylab = "f(x)",
     xlab = "x", col = "4")
text(2.7,0.5,"positive skew")
plot(4-x, dgamma(x,2,2), type = "l",ylab = "f(x)",xlab = "x",col = 4)
text(1.3, 0.5, "negative skew")
```



```r
par(mfrow = c(1, 1))

# calculate the skew
skew <- function(x){
m3 <- sum((x-mean(x))^3)/length(x); s3 <- sqrt(var(x))^3
m3/s3
}

# find the skew value and perform a t test to test if the skew is significantly different from zero
data <- read.table("skewdata.txt",header=T)
attach(data)
names(data)
```

```
## [1] "values"
```

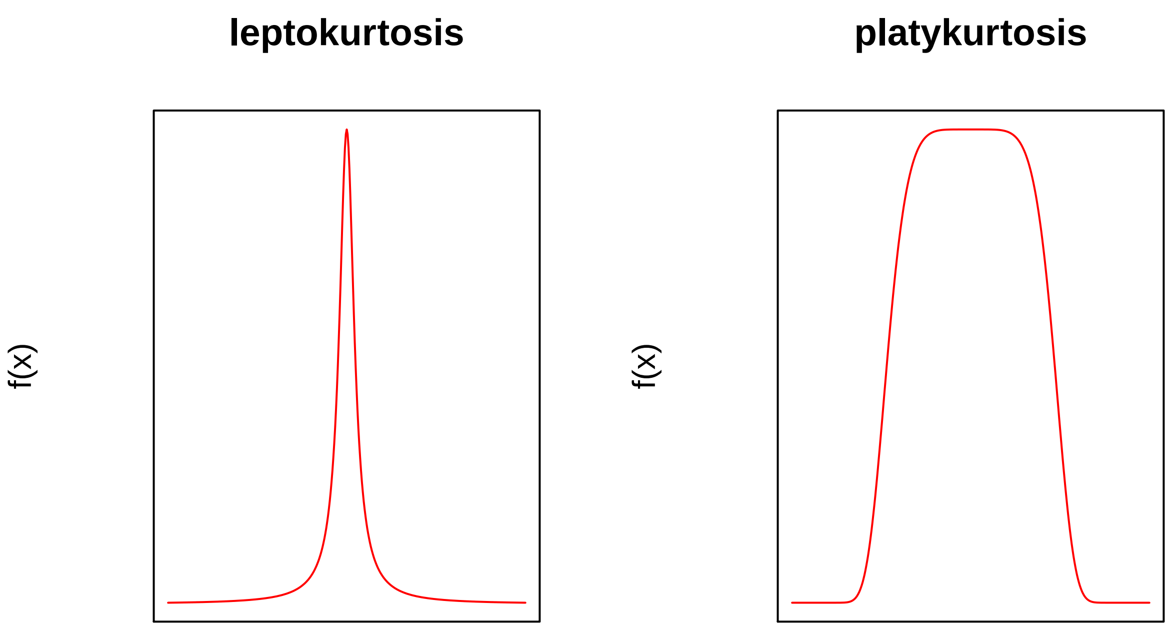
```r
skew(values) # skew value
```

```
## [1] 1.318905
```

```r
# t test
t.statistic <- skew(values)/sqrt(6/length(values))
1 - pt(t.statistic, length(values) - 1) # reject the null (normality)
```

```
## [1] 0.003120195
```

```r
# kurtosis
# difference between leptokurtsis and platykurtosis
par(mfrow = c(1, 2))
plot(-200:200, dcauchy(-200:200,0,10), type="l", ylab="f(x)", xlab="", yaxt="n",
            xaxt="n", main="leptokurtosis", col="red")
xv <- seq(-2, 2, 0.01)
plot(xv, exp(-abs(xv)^6), type="l", ylab="f(x)", xlab="", yaxt="n",
            xaxt="n", main="platykurtosis", col="red")
```



```
par(mfrow = c(1 , 1))


# function to calculate the kurtosis
kurtosis <- function(x) {
m4 <- sum((x - mean(x))^4)/length(x)
s4 <- var(x)^2
m4/s4 - 3 }

kurtosis(values)
```

```
## [1] 1.297751
```

```
detach(data)
```

Two samples

- Fisher's F tset by `var.test` for comparing two variances; Fligner-Killeen test and Bartlett test afor multiple samples by `fligner.test` and `bartlett.test`
- Student's t test by `t.test` for comparing two sample means with **normal** errors
- Wilcoxon's rank test by `wilcox.test` for comparing two sample means with **non-normal** errors : **non-parametric**
- Binomial test by `prop.test` for comparing two proportions
- Pearson's or Spearman's rank correlations by `cor.test` for correlations of two variables
- Chisq square test by `chisq.test` or Fisher's exact test by `fisher.test` for testing the independence of two variables in a contingency table.

```
# compare two variances
data <- read.table("f.test.data.txt", header = TRUE)
attach(data)
names(data)
```

```
## [1] "gardenB" "gardenC"
```

```r
var.test(gardenB, gardenC) # significant variance
```

```
##
##  F test to compare two variances
##
## data:  gardenB and gardenC
## F = 0.09375, num df = 9, denom df = 9, p-value = 0.001624
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.02328617 0.37743695
## sample estimates:
## ratio of variances
##            0.09375
```

```r
detach(data)


# more than two samples

ozone <- read.table("gardens.txt",header=T)
attach(ozone)
names(ozone) # 10 by 3 dataframe
```

```
## [1] "gardenA" "gardenB" "gardenC"
```

```r
y <- c(gardenA,gardenB,gardenC)
garden <- factor(rep(c("A","B","C"),c(10,10,10)))
var.test(gardenB, gardenC)
```

```
##
##  F test to compare two variances
##
## data:  gardenB and gardenC
## F = 0.09375, num df = 9, denom df = 9, p-value = 0.001624
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.02328617 0.37743695
## sample estimates:
## ratio of variances
##            0.09375
```

```r
bartlett.test(y ~ garden)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  y by garden
## Bartlett's K-squared = 16.758, df = 2, p-value = 0.0002296
```

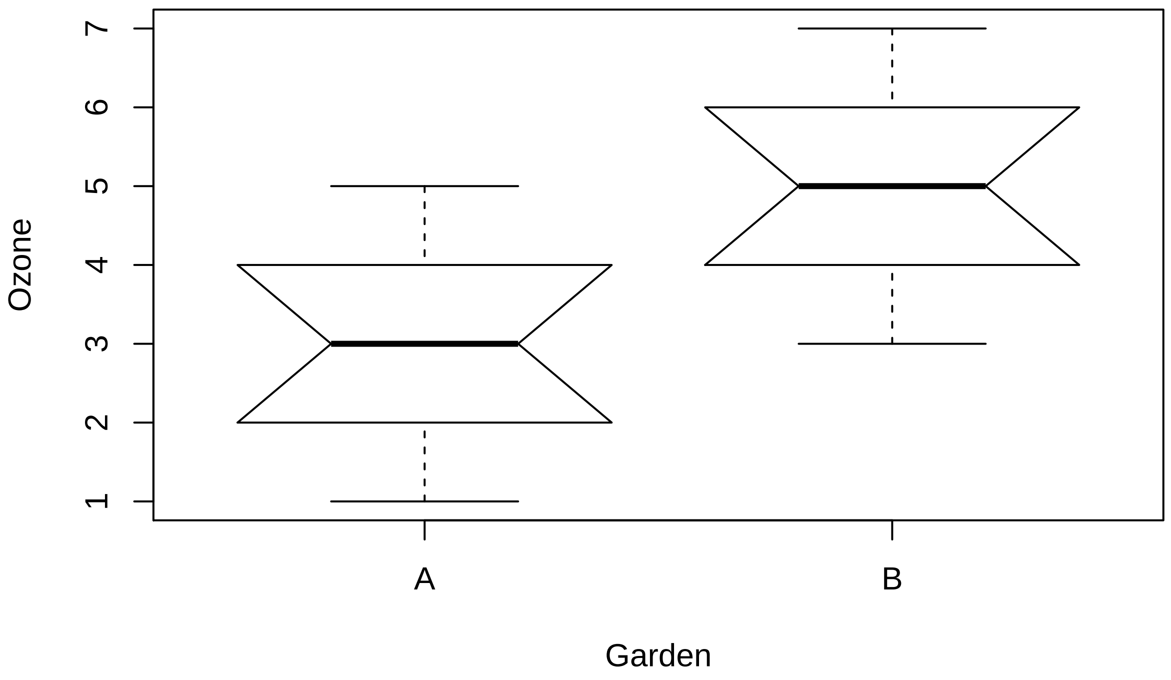
```r
fligner.test(y ~ garden)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  y by garden
## Fligner-Killeen:med chi-squared = 1.8061, df = 2, p-value = 0.4053
```

```r
detach(ozone)
# fligner test is different with the other two tests
# because Fisher and Bartlett are sensitive to outliers while Fligner is not.




# student's t test
t.test.data <- read.table("t.test.data.txt",header=T)
attach(t.test.data)
par(mfrow=c(1,1))
names(t.test.data)
```

```
## [1] "gardenA" "gardenB"
```

```r
ozone <- c(gardenA, gardenB)
label <- factor(c(rep("A",10), rep("B",10)))
boxplot(ozone ~ label, notch=T, xlab="Garden", ylab="Ozone")
```



```r
# carry out the t test in the long hand
s2A <- var(gardenA)
s2B <- var(gardenB)

# t statistic
t.statistic <- (mean(gardenA) - mean(gardenB))/sqrt(s2A/10+s2B/10)

# p value for two tailed test
2*pt(t.statistic, length(gardenA) + length(gardenB) - 2)
```

```
## [1] 0.001114539
```

```r
# an easier way to do t test
t.test(gardenA, gardenB)
```

```
##
##  Welch Two Sample t-test
##
## data:  gardenA and gardenB
## t = -3.873, df = 18, p-value = 0.001115
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.0849115 -0.9150885
## sample estimates:
## mean of x mean of y
##         3         5
```

```
# wilcoxon rank-sum test
ozone <- c(gardenA, gardenB)
ozone
```

```
##  [1] 3 4 4 3 2 3 1 3 5 2 5 5 6 7 4 4 3 5 6 5
```

```
label <- c(rep("A", 10), rep("B", 10))
label
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B"
## [18] "B" "B" "B"
```

```
combined.ranks <- rank(ozone)
combined.ranks
```

```
##  [1]  6.0 10.5 10.5  6.0  2.5  6.0  1.0  6.0 15.0  2.5 15.0 15.0 18.5 20.0
## [15] 10.5 10.5  6.0 15.0 18.5 15.0
```

```
tapply(combined.ranks, label, sum)
```

```
##   A   B
##  66 144
```

```
wilcox.test(gardenA,gardenB) # reject the null (the mean ozone between the two are the same)
```

```
## Warning in wilcox.test.default(gardenA, gardenB): cannot compute exact p-
## value with ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  gardenA and gardenB
## W = 11, p-value = 0.002988
## alternative hypothesis: true location shift is not equal to 0
```

```
#  test on paired samples
streams <- read.table("streams.txt",header=T)
attach(streams)
names(streams)
```

```
## [1] "down" "up"
```

```
# a t test treating as unpaired data
t.test(down, up)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  down and up
## t = -0.40876, df = 29.755, p-value = 0.6856
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -5.248256  3.498256
## sample estimates:
## mean of x mean of y
##    12.500    13.375
```

```
# paired t test for "true mean equals to 0 "
t.test(down, up, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  down and up
## t = -3.0502, df = 15, p-value = 0.0081
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.4864388 -0.2635612
## sample estimates:
## mean of the differences
##                  -0.875
```

```
# another way for paired test
difference <- up - down
t.test(difference)
```

```
##
##  One Sample t-test
##
## data:  difference
## t = 3.0502, df = 15, p-value = 0.0081
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.2635612 1.4864388
## sample estimates:
## mean of x
##     0.875
```

```
detach(streams)

# the sign test
#

sign.test <- function(x, y){
  if(length(x) != length(y)) stop("The two variables must be the same length")
  d <- x - y
  binom.test(sum(d > 0), length(d))
}
sign.test(gardenA,gardenB)
```

```
##
##  Exact binomial test
##
## data:  sum(d > 0) and length(d)
## number of successes = 0, number of trials = 10, p-value = 0.001953
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.0000000 0.3084971
## sample estimates:
## probability of success
##                      0
```

```
# sign test is non parametric, with other things equal, the parametric test will be more powerful.
detach(t.test.data)


# proportion test is useful when sample sizes are not equal
prop.test(c(4,196),c(40,3270))
```

```
## Warning in prop.test(c(4, 196), c(40, 3270)): Chi-squared approximation may
## be incorrect
```

```
##
##  2-sample test for equality of proportions with continuity
##  correction
##
## data:  c(4, 196) out of c(40, 3270)
## X-squared = 0.52289, df = 1, p-value = 0.4696
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.06591631  0.14603864
## sample estimates:
##     prop 1     prop 2
## 0.10000000 0.05993884
```

8.8 Chi-squared contigency tables , ignored

Correlation and covariance

- **Correlation** : $r = \frac{cov(x,y)}{\sqrt{s_x^2 \cdot s_y^2}}$.
- **Partial correlation** for data with more than two variables: the correlation of x and y given the third variable z constant, $r_{xy.z} = \frac{r_{xy} - r_{xz} r_{yz}}{\sqrt{(1-r_{xz}^2)(1-r_{yz}^2)}}$.

```
pollute <- read.table("Pollute.txt", header = TRUE)
attach(pollute)

# correlation matrix
cor(pollute)
```

```
##              Pollution        Temp    Industry  Population        Wind
## Pollution   1.00000000 -0.43360020  0.64516550  0.49377958  0.09509921
## Temp       -0.43360020  1.00000000 -0.18788200 -0.06267813 -0.35112340
## Industry    0.64516550 -0.18788200  1.00000000  0.95545769  0.23650590
## Population  0.49377958 -0.06267813  0.95545769  1.00000000  0.21177156
## Wind        0.09509921 -0.35112340  0.23650590  0.21177156  1.00000000
## Rain        0.05428389  0.38628047 -0.03121727 -0.02606884 -0.01246601
## Wet.days    0.36956363 -0.43024212  0.13073780  0.04208319  0.16694974
##                   Rain    Wet.days
## Pollution   0.05428389  0.36956363
## Temp        0.38628047 -0.43024212
## Industry   -0.03121727  0.13073780
## Population -0.02606884  0.04208319
## Wind       -0.01246601  0.16694974
## Rain        1.00000000  0.49605834
## Wet.days    0.49605834  1.00000000
```

```
# or do the correlation  long hand
varp <- var(Pollution)
varw <- var(Wet.days)
varc <- var(Pollution, Wet.days)

# the correltion is
varc/sqrt(varp*varw)
```

```
## [1] 0.3695636
```

```
# correlation between two vectors
cor(Pollution, Wet.days)
```

```
## [1] 0.3695636
```

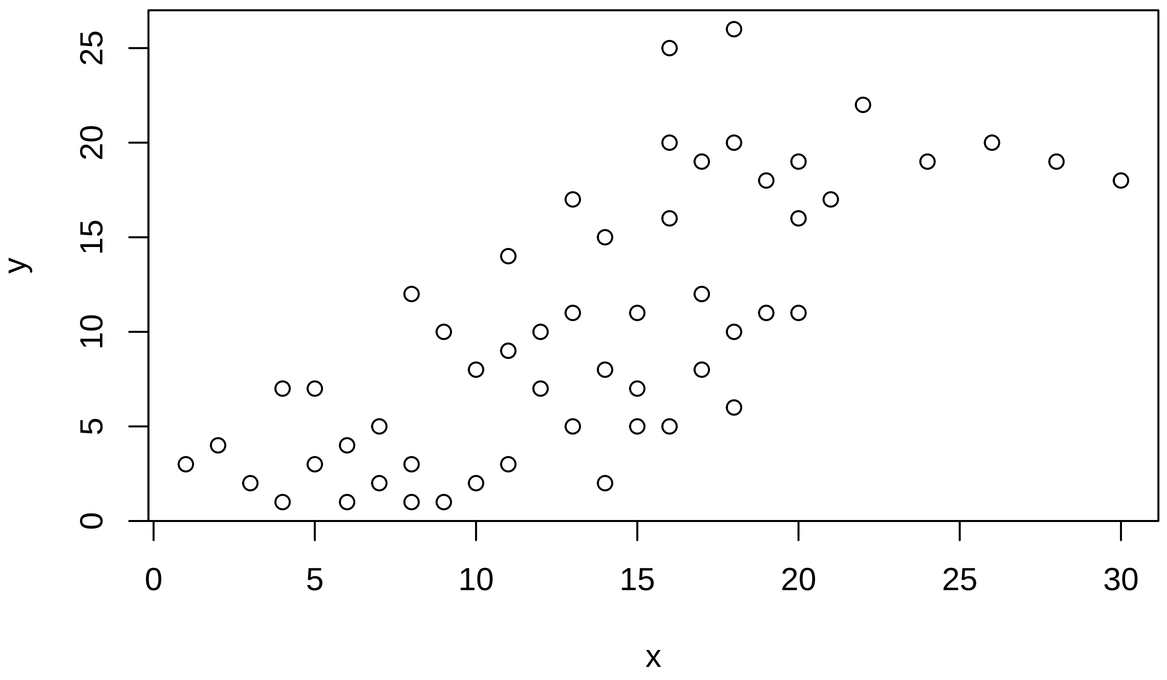
```
# correlation test
cor.test(Pollution, Wet.days)
```

```
##
##  Pearson's product-moment correlation
##
## data:  Pollution and Wet.days
## t = 2.4838, df = 39, p-value = 0.0174
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.0698555 0.6080778
## sample estimates:
##       cor
## 0.3695636
```

```
detach(pollute)



## scale dependent correlations
productivity <- read.table("productivity.txt", header = TRUE)
rm(x)
rm(y)
attach(productivity)
names(productivity)
```

```
## [1] "x" "y" "f"
```

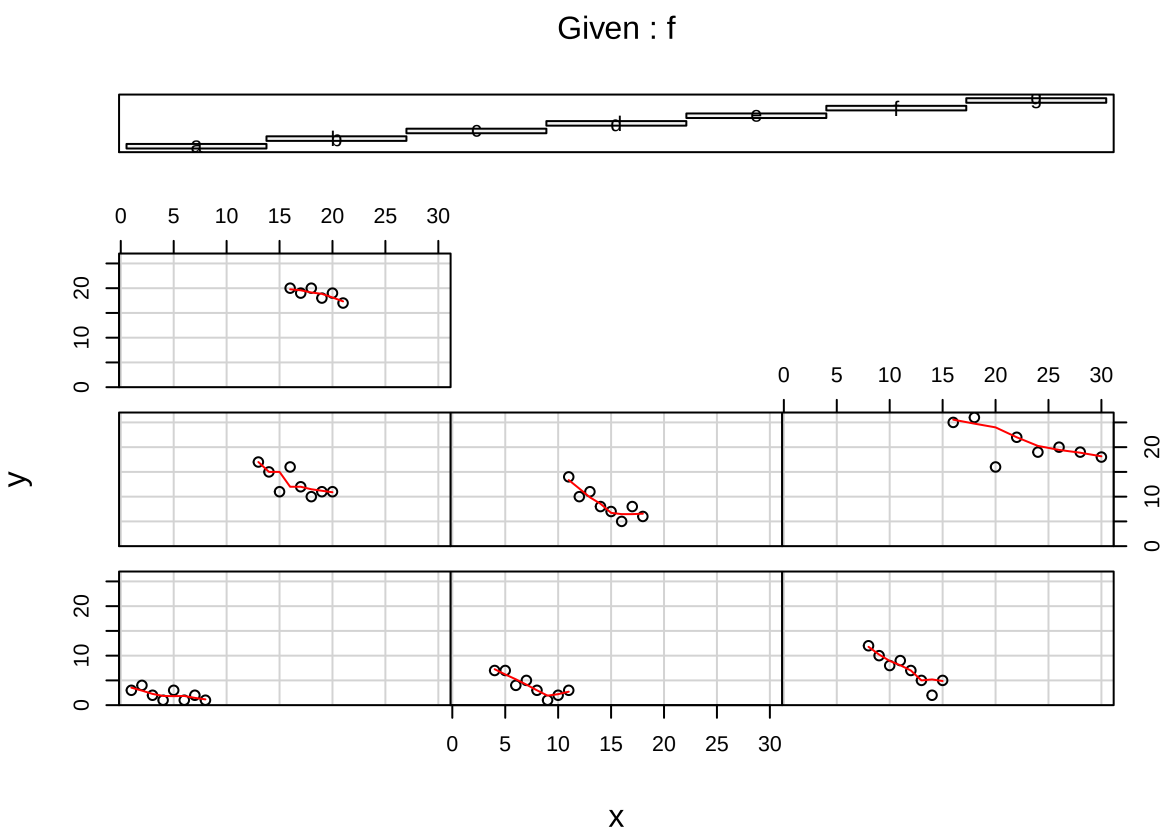
```
# the overall relationship between x and y
plot(x, y, pch = 21)
```



```r
# correlation test
cor.test(x, y) # significant positive correlation
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 7.5229, df = 52, p-value = 7.268e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5629686 0.8293555
## sample estimates:
##       cor
## 0.7219081
```

```r
# correlation for each region(f)
coplot(y~x|f, panel = panel.smooth, main = "") # totally different
```



```r
detach(productivity)
```

Kolmogorov-Smirnov test

It works on **cumulative distribution functions**.

- Are the two sample distributions the same?
- Does a particular sample distribution arise from a particular hypothesized distribution?

```r
x <- rnorm(50)
y <- runif(30)
# Do x and y come from the same distribution?
# Perform a one- or two-sample Kolmogorov-Smirnov test.

ks.test(x, y) # marginally not significant
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x and y
## D = 0.62667, p-value = 2.057e-07
## alternative hypothesis: two-sided
```

```r
# Does x come from a shifted gamma distribution with shape 3 and rate 2?
ks.test(x+2, "pgamma", 3, 2) # two-sided, exact
```

```
##
##  One-sample Kolmogorov-Smirnov test
```

```
##
## data:  x + 2
## D = 0.26136, p-value = 0.001676
## alternative hypothesis: two-sided
```

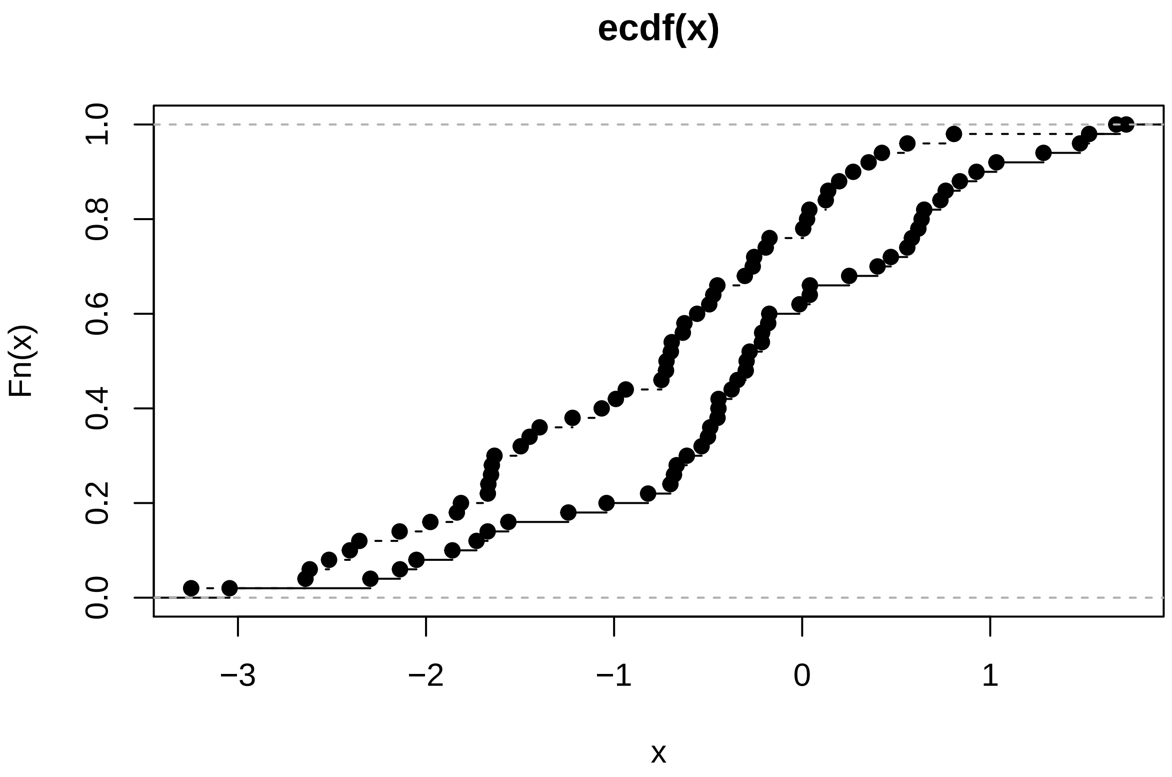
```
ks.test(x+2, "pgamma", 3, 2, exact = FALSE)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x + 2
## D = 0.26136, p-value = 0.00216
## alternative hypothesis: two-sided
```

```
ks.test(x+2, "pgamma", 3, 2, alternative = "gr")
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x + 2
## D^+ = 0.11105, p-value = 0.2711
## alternative hypothesis: the CDF of x lies above the null hypothesis
```

```
# test if x is stochastically larger than x2
x2 <- rnorm(50, -1)
plot(ecdf(x), xlim = range(c(x, x2)))
plot(ecdf(x2), add = TRUE, lty = "dashed")
```



```
t.test(x, x2, alternative = "g")
```

```
##
##  Welch Two Sample t-test
```

```
##
## data:  x and x2
## t = 2.963, df = 97.984, p-value = 0.001912
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.2712899       Inf
## sample estimates:
##  mean of x  mean of y
## -0.2574336 -0.8746006
```

```
wilcox.test(x, x2, alternative = "g")
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x and x2
## W = 1671, p-value = 0.001873
## alternative hypothesis: true location shift is greater than 0
```

```
ks.test(x, x2, alternative = "l")
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x and x2
## D^- = 0.3, p-value = 0.01111
## alternative hypothesis: the CDF of x lies below that of y
```

```
rm(x)
rm(y)
```

Power analysis

The power of a test is the probability of rejecting the null hypothesis when it is false.

- Type I error: reject the null when it's correct.
- Type II error: accept the null when it's false.
- `power.t.test` for power calculations of one- and two-sample t tests;
- `power.prop.test` for power calculation of two sample test for proportions;
- `power.anova.test` for power calculations of balanced one-way ANOVA tests.

For example, for a t test for two sample means, the t statistic is $t = \frac{d}{\sqrt{2\frac{s^2}{n}}}$, or equivalently, $n = \frac{2s^2t^2}{d^2}$.

```
# detect a difference of 10% when mean is 20, i.e., delta = 20, power is 80% and sd 3.5
power.t.test(delta = 2, sd = 3.5, power = 0.8)
```

```
##
##      Two-sample t test power calculation
##
##               n = 49.05349
##           delta = 2
##              sd = 3.5
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

```r
# the size of difference for two samples (n =15 for each)

power.t.test(n = 15, sd = 3.5, power = 0.8)
```

```
##
##      Two-sample t test power calculation
##
##               n = 15
##           delta = 3.709303
##              sd = 3.5
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Bootstrap

- Obtain confidence intervals for the mean vector

```r
data <- read.table("skewdata.txt", header = TRUE)
attach(data)
names(data)
```

```
## [1] "values"
```

```r
ms <- numeric(10000)
for(i in 1:10000){
  ms[i] <- mean(sample(values, replace = T))
}

quantile(ms, c(0.025, 0.975)) # the CI with skewed data is also skewed
```

```
##     2.5%    97.5%
## 24.97591 37.96576
```

```r
# use "boot" package to do the same thing
library(boot)

# boot Generate R bootstrap replicates of a statistic applied to data.

# function which is the statistic of interest
mymean <- function(values, i) mean(values[i])

myboot <- boot(values, mymean, R = 10000)
myboot
```

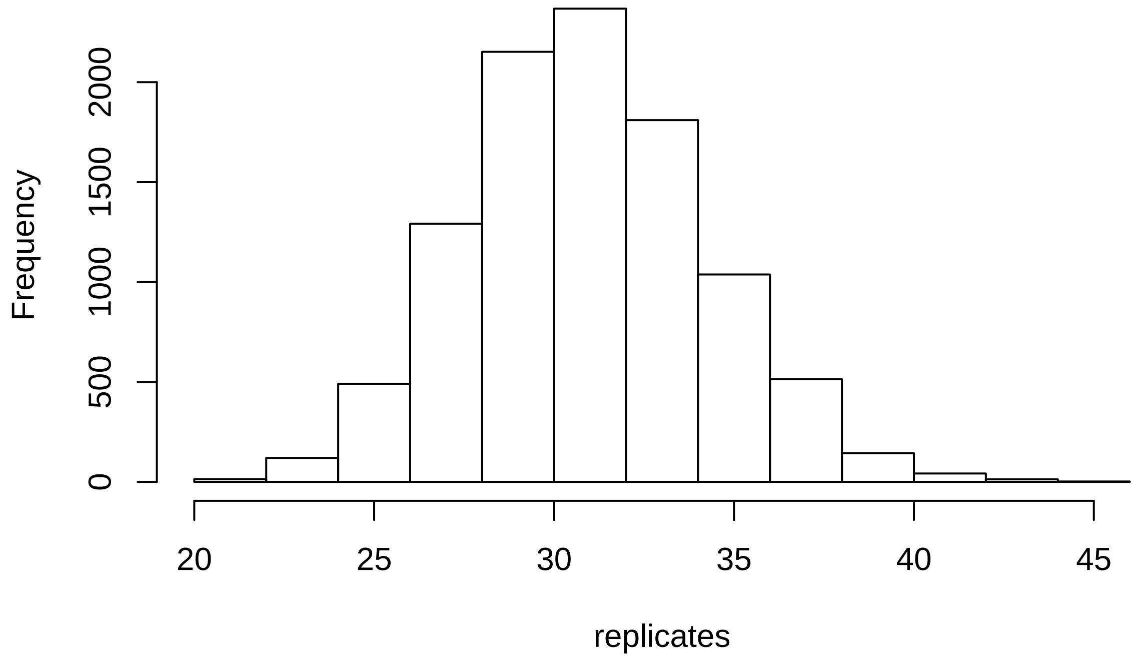
```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
```

```
## boot(data = values, statistic = mymean, R = 10000)
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1* 30.96866 -0.04598958    3.315804
```

```r
names(myboot)
```

```
##  [1] "t0"        "t"         "R"         "data"      "seed"
##  [6] "statistic" "sim"       "call"      "stype"     "strata"
## [11] "weights"
```

```r
hist(myboot$t, main = "", xlab = "replicates")
```



```r
quantile(myboot$t, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 24.82988 37.69340
```

```r
detach(data)
```