DISTRIBUTIONS

Functions related to distributions belong to the (automatically loaded) `stats` package and have the form:

`d<distribution>(x, <parameters>)`	returns the value of the PDF ($f(x)$ or $P(X = x)$
`p<distribution>(x, <parameters>)`	returns the value of the CDF ($F(x)$) or $P(X \leq x)$)
`p<distribution>(x, <parameters>, lower=FALSE)`	returns $1 - F(x)$ or $P(X > x)$
`q<distribution>(p, <parameters>)`	returns the min value of x for which $P(X \leq x) \geq p$ (p-th quantile)
`q<distribution>(p, <parameters>, lower=FALSE)`	returns the min value of x for which $P(X > x) \geq p$
`r<distribution>(n, <parameters>)`	returns an iid random sample of size n from the distribution

Continuous Distributions:

The parameters for Continuous Distributions are:

(a) Uniform Distribution ($U(a, b)$)	`unif(<>,a,b)`, `dunif(<>, a, b)`, `punif(<>, a, b)`, …
(b) Exponential Distribution (Exp(λ))	`exp(<>, lambda)`
(c) Gamma Distribution (Gamma(α, λ))	`gamma(<>, alpha, lambda)`
(d) Chi-square Distribution (χ^2_{df})	`chisq(<>,df)`
(e) Beta Distribution (Beta(α, β))	`beta(<>,alpha,beta)`
(f) Normal Distribution ($\mathcal{N}(\mu, \sigma^2)$)	`norm(<>,mu,sigma)`
(g) Log-normal Distribution (Lognormal(μ, σ^2))	`lnorm(<>,mu,sigma)`
(h) Standard Normal Distribution (N(0,1))	`norm(<>)`
(i) Student's t-distribution ($t_{(\nu)}$)	`t(<>, nu)`
(j) F-Distribution ($F(d_1, d_2)$)	`f(<>, d1, d2)`

Discrete Distributions:

The parameters for Discrete Distributions are:

(a) Binomial Distribution (Bin(n, p))	`binom(<>, n, p)`
(b) Poisson Distribution ($\mathcal{P}(\lambda)$)	`pois(<>, lambda)`
(c) Geometric Distribution (Geom(p))	`geom(<>, p)`
(d) Negative Binomial Distribution (NB(k, p))	`nbinom(<>, k, p)`
(e) Hypergeometric Distribution (HG(N, K, n))	`hyper(<>, N, K, n)`

(f) Uniform Distribution:

R does not provide built-in functions to deal with the discrete uniform distribution. Here are some alternatives:

```
x <- seq(a, b, i)                          # generates a sequence a, a+i, a+2i, ... a+ki <= b
length(x[x<=t]) / length(x)                # yields P[X<=t]
length(x[x>=p | x>=q]) / length(x)         # yields P[X>=p or X>=q]
length(x[x>=p & x>=q]) / length(x[x >= q]) # yields P[X=>p | X>=q] (conditional probability)
```

TABLES AND PLOTS

1. Frequency Table:

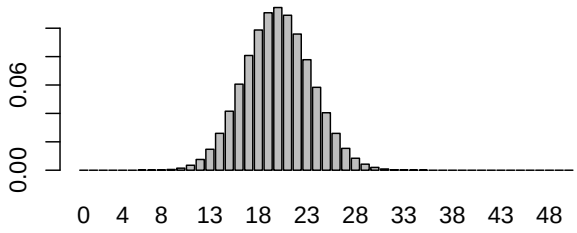
Example: Frequency table for an iid sample X with $X_i \sim \text{Bin}(100, 0.2)$. The bottom row is the number of occurrences of the corresponding value in the top row in the sample.

```
table(rbinom(100, 100, 0.1))


##
##  4  5  6  7  8  9 10 11 12 13 14 15 16 18
##  2  6  3 13 11 13 15 13 12  8  1  1  1  1
```

2. Bar chart:

Example: Bar chart for the PDF of $\text{Bin}(50, 0.4)$:

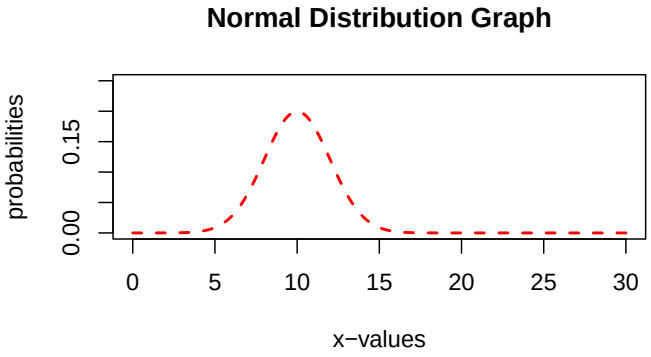
```
x <- 0:50
barplot(dbinom(x, 50, 0.4), names = x)
```



3. Plots:

Example: Plot of the PDF of $\mathcal{N}(10, 2^2)$:

```
x <- seq(0, 30, 0.01)
plot(x, dnorm(x, 10, 2),                          # x and y
     type = "l",                                  # type (p = points, l = lines, o = both overlaid)
     main = "Normal Distribution Graph",          # plot title
     xlab = "x-values", ylab = "probabilities",   # axis labels
     xlim = c(0,30), ylim = c(0,0.25),            # axis limits
     col = "red",  lwd = 2,                       # color and line width
     lty = 2)                                     # line type (1 = solid, 2 = dashed, 3 = dotted)
```



Plots over existing plots

These function calls are ran after a call to `plot()`. They all support the argument `col=<>` to assign colors.

(a) `lines(x, y)`: Adds a line plot.

(b) `points(x, y)`: Adds a scatter plot.

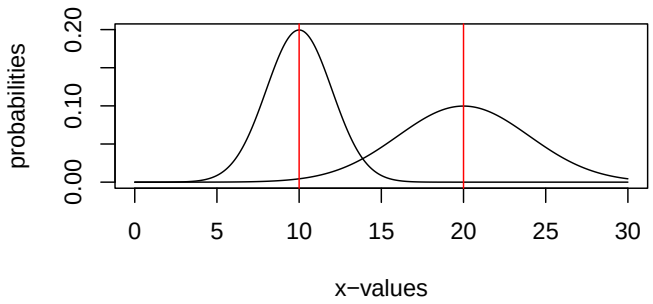
(c) `abline(h = y0)`: Adds a horizontal line at y_0

(d) `abline(v = x0)`: Adds a vertical line at x_0.

(e) `abline(a, b)`: Adds the straight line $y = a + b \cdot x$.

Example: Plot of the PDFs of $\mathcal{N}(10, 2^2)$ and $\mathcal{N}(20, 4^2)$, highlighting the μs:

```
x <- seq(0, 30, 0.01)
plot(x, dnorm(x, 10, 2), type = "l", xlab = "x-values", ylab="probabilities")
lines(x, dnorm(x, 20, 4))
abline(v = 10, col = 'red')
abline(v = 20, col = 'red')
```



Q-Q Plots

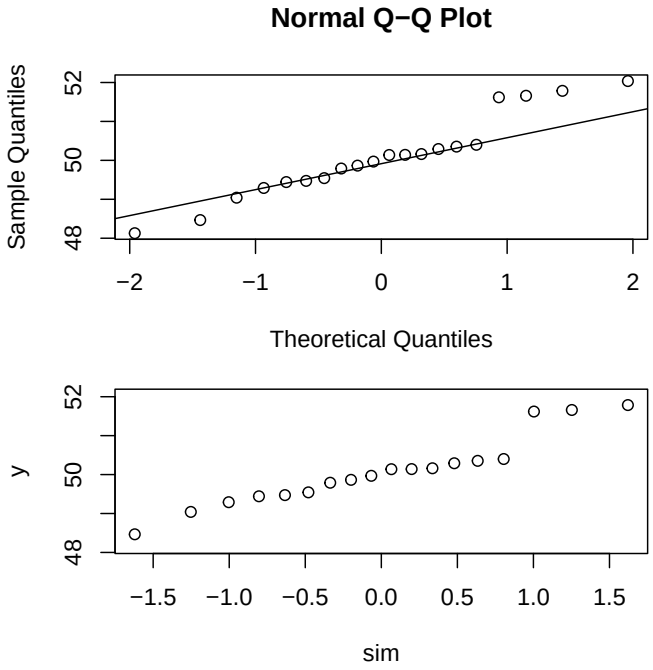
(a) `qqnorm(x)`: Plots quantiles of sample data `x` (vertical axis) vs quantiles of a normal distribution (horizontal axis).

(b) `qqplot(sim,x)`: The same as above, but with generic distribution quantiles `sim` (e.g. generated by `qexp(seq(0, 1, 0.05), lambda)`)

(c) `qqline(x)`: Adds a reference line on a Q-Q plot created with `qqnorm()`.

Example: Q-Q plot of normal data in two different ways:

```
y <- rnorm(20, 50, 1)                  # generate a random sample
qqnorm(y)                              # approach 1: qqnorm
qqline(y)
sim <- qnorm(seq(0, 1, 0.01), 0, 1)    # generate quantiles
qqplot(sim, y)                         # approach 2: qqplot
```

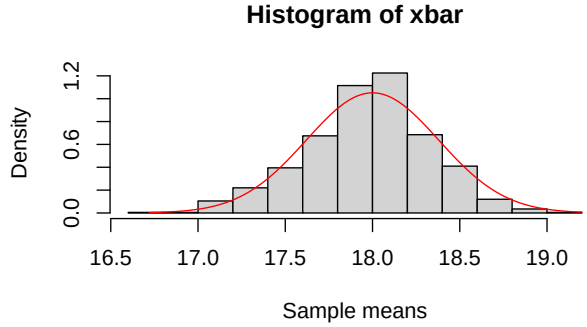


CENTRAL LIMIT THEOREM (CLT)

Example: Assume we have an iid sample $\bar{X}$ of size 50 from Bin(30, 0.6). By the CLT

$$\bar{X} \sim \mathcal{N}(18, 0.144)$$

```
N <- 30
p <- 0.6
set.seed(30)                    # setting a seed makes the code results repeatable!
xbar <- rep(0,1000)             # initialize a sequence [1, 2, 3, ..., 1000]
for (i in 1:1000) {
  x <- rbinom(50, N, p)         # generate a sample of size 50 from Bin(30, 0.6)
  xbar[i] <- mean(x)            # replace each value of the sequence by the sample mean
}
hist(xbar, prob = TRUE, xlab = "Sample means")
rng <- range(xbar)              # returns a vector with [min(xbar), max(xbar)]
xvals <- seq(rng[0], rng[1], 0.001)
lines(xvals, dnorm(xvals, N * p, sqrt((N * p * (1-p)) / 50)), col = "red")
```



DATA ANALYSIS

Scatter Plots:

```
plot(<x>, <y>)       # same function as before
plot(<dataframe>)    # for multivariate data and bivariate data
```

Correlation:

```
cor(<x>, <y>, method = "<>")
cor(<dataframe>, method = "<>")
# method can be "pearson" (by default), "spearman" or "kendall"
```

Correlation Test:

```
cor.test(<x>, <y>, method = "pearson", alt = "two.sided", conf = 0.95)
# method can be "pearson" (by default), "spearman" or "kendall"
```

CONFIDENCE INTERVALS AND HYPOTHESIS TESTING

Functions to compute confidence intervals (CI) or conduct hypothesis testing (HT), have the form:

```
<name>.test(<data>,
            alternative = <>,     # can be "two.sided", "less" or "greater"
            conf.level = 0.95,
            mu = <>)              # depends on the parameter in study;
                                  # when set, this it becomes the Null Hypothesis
```

Test function attributes:

`$method:`	type of test
`$statistic:`	test statistic
`$p.value:`	p-value
`$parameter:`	DoF
`$observed:`	observed counts
`$expected:`	expected counts under the Null Hypothesis

Example:

```
test <- t.test(x, alternative = "two.sided", conf.level = 0.95, mu = 0)
test$p.value
```

```
## [1] 1.113842e-43
```

One Sample CI/HT:

1. Mean (σ unknown):

```
t.test(x, alternative = "two.sided", conf.level = 0.95)                  # CI
t.test(x, alternative = "two.sided", conf.level = 0.95, mu = 0)          # HT
# x is a vector of observations
```

2. Probability p of a Bernoulli trial

```
binom.test(x, n, alternative = "two.sided", conf.level = 0.95)          # CI
binom.test(x, n, alternative = "two.sided", conf.level = 0.95, p=0.5)   # HT
# x is number of successes, n is number of trials
```

3. The rate parameter λ (i.e. mean) of a Poisson distribution:

```
poisson.test(x, alternative = "two.sided", conf.level = 0.95)           # CI
poisson.test(x, alternative = "two.sided", conf.level = 0.95, r = 1)    # HT
# x is number of events in one time base, r is the event rate.
```

Example: Output from calling the `t.test` function on a normal sample with a probable $\mu = 2$

```
x <- rnorm(50, 20, 3)
t.test(x, alternative = "two.sided", conf.level = 0.95, mu = 2)
```

```
##
##  One Sample t-test
##
## data:  x
## t = 44.174, df = 49, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 2
## 95 percent confidence interval:
##  19.06296 20.68939
## sample estimates:
## mean of x
##  19.87618
```

Two Sample CI/HT:

1. Difference of means (σ unknown):

```
t.test(x1, x2, alternative = "two.sided", var.equal = F, conf = 0.95)         # CI
t.test(x1, x2, alternative = "two.sided", var.equal = F, conf = 0.95, mu = 0) # HT
# mu is the difference of the means; x1 and x2 are vectors with data observations
```

2. Difference of means (paired data; e.g. before-and-after):

```
t.test(x1, x0, alternative = "two.sided", conf = 0.95, paired = T)            # CI
t.test(x1, x0, alternative = "two.sided", conf = 0.95, paired = T, mu = 0)    # HT
```

3. Variance Ratio:

```
var.test(x1, x2, alternative = "two.sided", conf = 0.95)                      # CI
var.test(x1, x2, alternative = "two.sided", conf = 0.95, ratio = 1)           # HT
# ratio is the ratio of the population variances; x1 and x2 are vectors with data observations
```

4. Difference of proportions (Bernoulli trials):

```
prop.test(x, n, alternative = "two.sided", conf.level = 0.95, correct = T)    # CI
prop.test(x, n, alternative = "two.sided", conf.level = 0.95, correct = T, p) # HT
# x is a vector c(x1, x2) of the number of successes
# n is a vector c(n1, n2) of the number of trials
# p is a vector c(p1, p2) of the proportions to be HT
```

Chi-squared tests:

1. Goodness-of-fit (only HT makes sense; H_0 is the hypothesis that the data is generated from a multinomial distribution with the ps as probabilities):

```
chisq.test(x, p) # HT
# x is a vector of integers (the counts of each class)
# p is the probabilities of each class
```

Example: A man sitting in his balcony has spotted a number of birds belonging to one of six species.
The exact classification is given below:

Species	1	2	3	4	5	6
Frequency	8	10	13	9	11	14

Test at 5% level of significance whether or not the data are compatible with the assumption that the particular area is visited by the same proportion of birds belonging to these six species in the population. (Goodness-of-fit test)

```
# Using the chi-squared test:
obs <- c(8, 10, 13, 9, 11, 14)
n <- length(obs)
ep <- rep(1/n, n)   # a vector c(1/n, 1/n, ..., 1/n)
pval1 <- chisq.test(obs, p = ep)$p.value
pval1

## [1] 0.7799664

# Using First Principles:
ef <- ep * sum(obs)
ts <- sum((obs-ef)^2 / ef)
pval2 <- pchisq(ts, n-1, lower.tail = FALSE)
pval2

## [1] 0.7799664
```

Since the p-value is greater than 0.05, we have no reason to reject the hypothesis that the six species occur with the same frequency.

LINEAR REGRESSION

Linear Regression Models:

```
X <- rnorm(50, 20, 2)                    # generate pseudorandom normal data for X and Y
Y <- 20 * X + 42 + rnorm(50, 20, 2)

model1 <- lm(Y ~ X)       # Y = a + Bx
model2 <- lm(Y ~ X - 1)   # Y = Bx
model3 <- lm(Y ~ 1)       # Y = a

summary(model1)

##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5506 -0.8937 -0.1579  0.9268  3.1274
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  59.5070     2.1692   27.43   <2e-16 ***
## X            20.1292     0.1045  192.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.639 on 48 degrees of freedom
## Multiple R-squared:  0.9987,Adjusted R-squared:  0.9987
## F-statistic: 3.707e+04 on 1 and 48 DF,  p-value: < 2.2e-16
```

Since the p-value is less than 5%, we should reject H_0 (that all coefficients except the intercept are zero).

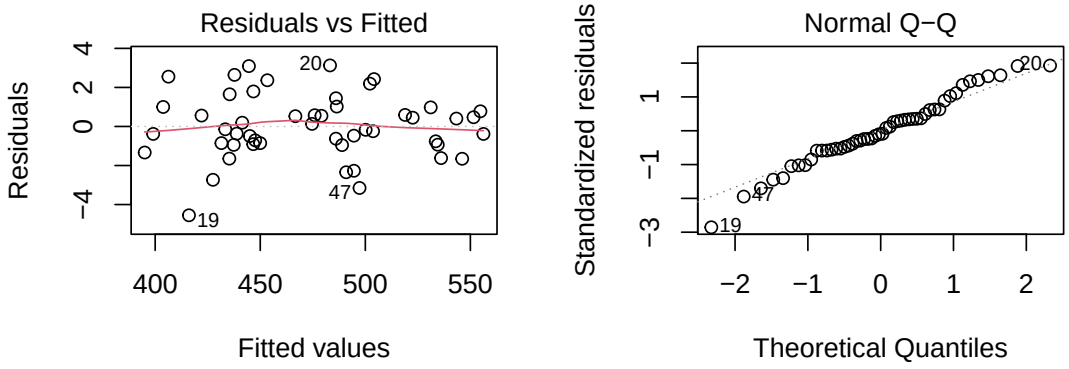
The coefficients could have also been obtained via `model1$coef` or `coef(model1)`.

Using models for prediction and Confidence Interval for Mean and Individual Response:

```
newdata <- data.frame(X=30)
predict(model1, newdata)                                    # point estimate of the response

##        1
## 663.3824

predict(model1, newdata, interval = "confidence", level = 0.95) # mean response

##        fit      lwr      upr
## 1 663.3824 661.3582 665.4067

predict(model1, newdata, interval = "predict", level = 0.95)    # individual response

##        fit      lwr      upr
## 1 663.3824 659.5145 667.2504
```

Checking the residuals:

```
layout(matrix(c(1, 2), 1, 2))
plot(model1, 1)
plot(model1, 2)
```



ANOVA Tables and **Confidence intervals for parameters**:

```
size <- c(3, 4, 5, 6, 4, 5, 6, 7, 7, 8, 9, 10)
pop <- c("A", "A", "A", "A", "B", "B", "B", "B", "C", "C", "C", "C")

model <- lm(size ~ pop)
anv <- anova(model)
anv

## Analysis of Variance Table
##
## Response: size
##           Df Sum Sq Mean Sq F value   Pr(>F)
## pop        2 34.667 17.3333    10.4 0.004572 **
## Residuals  9 15.000  1.6667
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anv[1, 2] # individual table entries are available using this syntax:

## [1] 34.66667

confint(model, level = 0.95) # confidence intervals for the parameters

##                2.5 %   97.5 %
## (Intercept)  3.039784 5.960216
## popB        -1.065058 3.065058
## popC         1.934942 6.065058
```

MULTIPLE LINEAR REGRESSION

Regression Models:

Form	Model
`model <- lm(Y ~ X1 + X2 + X3)`	$Y = a + b_1 \cdot X_1 + b_2 \cdot X_2 + b_3 \cdot X_3$
`model <- lm(Y ~ X1 * X2)`	$Y = a + b_1 \cdot X_1 + b_2 \cdot X_2 + b_{12} \cdot X_1 \cdot X_2$
`model <- lm(Y ~ X1 + X2 + X1:X2)`	$Y = a + b_1 \cdot X_1 + b_2 \cdot X_2 + b_{12} \cdot X_1 \cdot X_2$
`model <- lm(Y ~ I(X1^2))`	$Y = a + b_1 \cdot X_1^2$; the `I(<arg>)` makes `<arg>` "literal"

```r
# generating a fake dataset
set.seed(10)
X1 <- rnorm(100, 20, 2)
X2 <- rnorm(100, 10, 4)
Y <- 42 + 20 * X1 + rnorm(100, 0, 4)

model1 <- lm(Y ~ X1 + X2)
summary(model1)

##
## Call:
## lm(formula = Y ~ X1 + X2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6062  -2.2655   0.2998   2.6385   9.4370
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  43.4326     4.2955   10.11   <2e-16 ***
## X1           19.9993     0.2083   96.00   <2e-16 ***
## X2           -0.1355     0.1011   -1.34    0.183
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.895 on 97 degrees of freedom
## Multiple R-squared:  0.9896,Adjusted R-squared:  0.9894
## F-statistic:  4632 on 2 and 97 DF,  p-value: < 2.2e-16
```

ANOVA Table:

```
anova(model1)

## Analysis of Variance Table
##
## Response: Y
##           Df Sum Sq Mean Sq   F value Pr(>F)
## X1         1 140548  140548 9262.0579 <2e-16 ***
## X2         1     27      27    1.7951 0.1834
## Residuals 97   1472      15
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Confidence Interval of parameters:

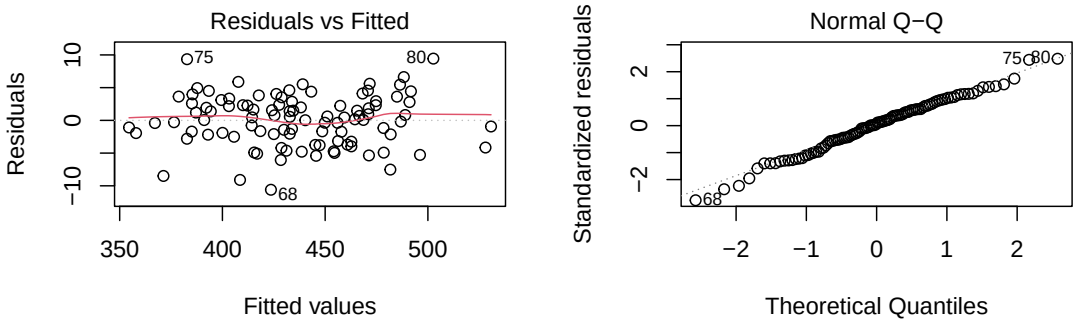
```
confint(model1, c(1, 2, 3), level = 0.95)

##                  2.5 %      97.5 %
## (Intercept) 34.9071989 51.95802047
## X1          19.5858137 20.41275538
## X2          -0.3362104  0.06521903
```

Individual Response and Mean Response:

```
newdata <- data.frame(X1 = 0.03, X2 = 0.02)
predict(model1, newdata, interval = "confidence")     # For mean response

##        fit      lwr      upr
## 1 44.02988 35.51763 52.54213

predict(model1, newdata, interval = "predict")        # For individual response

##        fit      lwr      upr
## 1 44.02988 32.53062 55.52914
```

Checking the fit of the model (Residuals):

```
# resid(model1)                 # this function yields the residuals
layout(matrix(c(1, 2), 1, 2))
plot(model1, 1)                 # residuals vs fitted values
plot(model1, 2)                 # Q-Q plot of residuals
```



GENERALISED LINEAR MODELS (GLMS)

GLM: `glm(<formula>, family = <family>(link = <link_function>))`

Family	Default link function	Other link functions
gaussian	identity	log, inverse
Gamma	inverse	log, identity
binomial	logit	log, probit
poisson	log	identity, sqrt

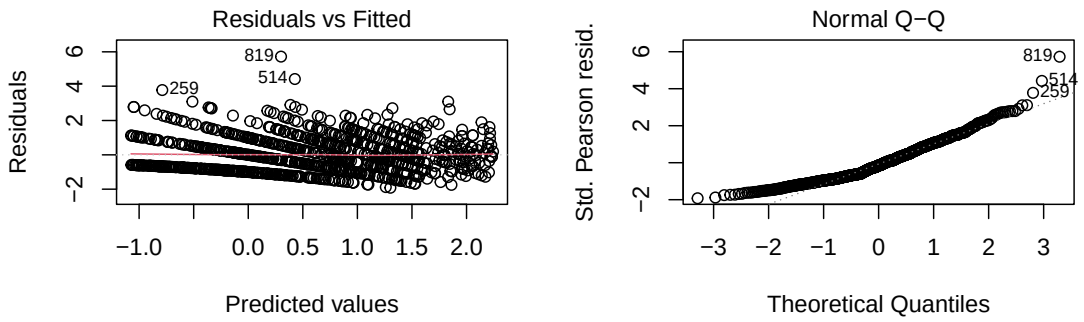
```
# generate fake date
set.seed(10)
n <- 1000
age <- runif(n, min = 18, max = 70)              # Age between 18 and 70
vehicle_type <- rbinom(n, size = 1, prob = 0.4)  # 40% SUVs
b <- list(X0 = -2, X1 = 0.05, X2 = 0.7)
lambda <- exp(b[['X0']] + b[['X1']] * age + b[['X2']] * vehicle_type)
Y <- rpois(n, lambda)

# create a GLM model
DF <- data.frame(Y, age, vehicle_type)
g <- glm(formula = Y ~ age + vehicle_type, data = DF, family = poisson())
g

##
## Call:  glm(formula = Y ~ age + vehicle_type, family = poisson(), data = DF)
##
## Coefficients:
##  (Intercept)           age  vehicle_type
##     -2.00875       0.05106       0.68335
##
## Degrees of Freedom: 999 Total (i.e. Null);  997 Residual
## Null Deviance:     2583
## Residual Deviance: 1115  AIC: 3228
```

ANOVA Table and **Confidence Intervals of Parameters** and **Model Predictions**:

```
# g is the GLM above!
anova(g)

## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: Y
##
## Terms added sequentially (first to last)
##
##
##              Df Deviance Resid. Df Resid. Dev
## NULL                            999     2582.6
## age           1  1193.09       998     1389.5
## vehicle_type  1   274.39       997     1115.1

confint(g, par = c(1, 2, 3), level = 0.90) # 90% confidence interval for all 3 parameters

## Waiting for profiling to be done...

##                      5 %        95 %
## (Intercept)  -2.16265206 -1.85726127
## age           0.04844901  0.05368993
## vehicle_type  0.61529167  0.75160250

new.data <- data.frame(age = c(30, 30), vehicle_type = c(1, 0))
predict(g, new.data)

##          1          2
##  0.2062803 -0.4770718
```

Checking the Model Fit:

```
layout(matrix(c(1, 2), 1, 2))
plot(g, 1)      #residuals vs fitted values
plot(g, 2)      #Q-Q plot of residuals
```



Model Comparison:

```
g1 <- glm(formula = Y ~ age, data = DF, family = poisson())
g1$aic

## [1] 3500.063

g$aic

## [1] 3227.675

# model g is better because AIC(g) < AIC(g1)!
```

BAYESIAN STATISTICS AND CREDIBILITY THEORY

Poisson-Gamma Model: If $X \sim \text{Poisson}(\lambda)$ is the distribution of the number of annual claims, then under the Gamma-Poisson model – having a $\lambda \sim \text{Gamma}(\alpha, \beta)$ prior – yields a posterior $\lambda|X \sim \text{Gamma}(\sum x_i + \alpha, n + \beta)$ on λ. In this case, the posterior mean is:

$$\hat{\lambda}_B = \frac{\sum x_i + \alpha}{n + \beta}$$

```
set.seed(100)
pm <- rep(0, 1000)
for (i in 1:1000) {
  lambda <- rgamma(1, shape = 9, rate = 4)
  x <- rpois(n = 20, lambda)
  pm[i] <- (sum(x) + 9) / (20 + 4)
}
mean(pm)

## [1] 2.208667
```

Credibility Theory: In this case,

$$\hat{\lambda}_{\text{Credibility}} = Z \cdot \hat{\lambda}_{\text{Individual}} + (1 - Z) \cdot \hat{\lambda}_{\text{Prior}}$$

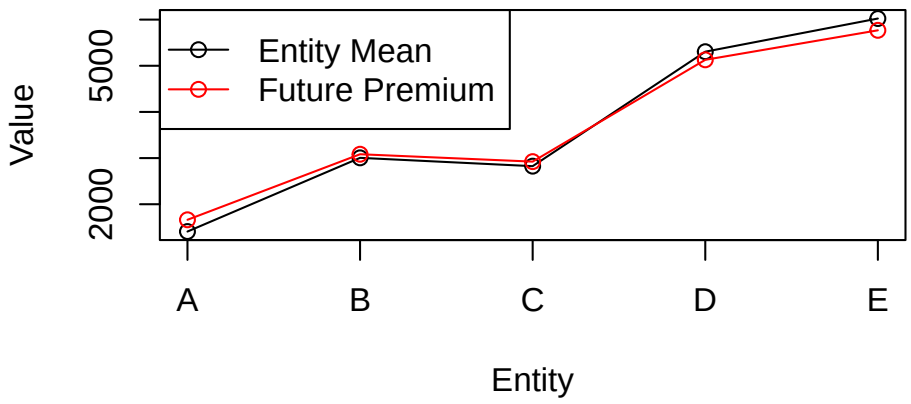
where: $\hat{\lambda}_{\text{Prior}} = \frac{\alpha}{\beta}$, the expected value of the prior, as in the example above; $\hat{\lambda}_{\text{Individual}} = \frac{\sum x_i}{n}$ and $Z = \frac{n}{n+\beta}$ is the Credibility Factor.

```
set.seed(100)
n <- 20
Z <- n / (n + 4)

cp <- rep(0, 1000)
for (i in 1:1000) {
  lambda <- rgamma(1,shape = 9, rate = 4)
  x <- rpois(n = n, lambda)
  cp[i] <- Z * mean(x) + (1-Z) * 9 / 4
}
mean(cp)

## [1] 2.208667
```

EMPIRICAL BAYES CREDIBILITY THEORY (EBCT)

Model 1: For the claims dataset below (5 entities, 6 years), determine the future premium.

```
data <- data.frame(
  X2015 = c(1350, 1000, 2050, 4670, 7150),
  X2016 = c(1080, 3080, 4550, 4270, 3480),
  X2017 = c(1800, 1700, 1900, 4800, 4900),
  X2018 = c(2040, 2820, 1600, 9070, 4810),
  X2019 = c(1000, 5760, 4200, 3770, 8800),
  X2020 = c(1180, 3670, 2650, 5250, 7000)
)
rownames(data) <- c("A", "B", "C", "D", "E")

n <- ncol(data)                                          # Number of time periods (e.g., years)
entity_mean_claims <- rowMeans(data)                     # Mean claim for each entity
entity_claim_variances <- apply(data, 1, var)            # Variance of claims for each entity
overall_mean_claim <- mean(entity_mean_claims)           # Overall mean claim amount
variance_of_entity_means <- var(entity_mean_claims)      # Variance of entity mean claims
average_entity_variance <- mean(entity_claim_variances)  # Average variance across entities
# Adjusting for within-entity variance
var.theta <- variance_of_entity_means - (average_entity_variance / n)
Z <- n / (n + average_entity_variance / var.theta)
# Credibility-adjusted estimate: Z * m_i + (1 - Z) * m
premiums <- Z * entity_mean_claims + (1 - Z) * overall_mean_claim
premiums

##        A        B        C        D        E
## 1662.853 3083.215 2923.090 5129.247 5768.262
```



Model 2: For this model, include a second dataframe consisting of policy volume.

```
volume <- data.frame(
  X2015 = c(7, 9, 4, 6, 4),
  X2016 = c(7, 11, 5, 6, 8),
  X2017 = c(8, 3, 6, 7, 8),
  X2018 = c(7, 11, 3, 8, 8),
  X2019 = c(13, 8, 3, 6, 9),
  X2020 = c(7, 12, 2, 13, 10)
)
rownames(volume) <- c("A", "B", "C", "D", "E")

n_i <- rowSums(volume)                                  # Total exposure for each group (n_i)
m_i <- rowSums(data) / n_i                              # Observed claim frequency per group (m_i)
total_claims <- sum(data)                               # Total claims across all groups
total_exposure <- sum(volume)                           # Total exposures across all groups
m <- total_claims / total_exposure                      # Overall average claim frequency (m)
P_ij <- data / volume                                   # Claim frequencies per period and group
s2i <- numeric(nrow(data))                              # Initialize vector for s_i^2
for (i in 1:nrow(data)) {
  exposures_i <- as.numeric(volume[i, ])
  P_ij_i <- as.numeric(P_ij[i, ])
  squared_deviations <- exposures_i * (P_ij_i - m_i[i])^2
  s2i[i] <- sum(squared_deviations) / n_i[i]
}

Es <- mean(s2i)                                         # Average within-group variance (E[s(theta)])
V_m_theta <- sum(n_i * (m_i - m)^2) / sum(n_i)          # Variance of m_i's (V[m(theta)])
K <- Es / V_m_theta                                     # K = E[s(theta)] / V[m(theta)]
Z_i <- n_i / (n_i + K)                                  # Credibility factor per group (Z_i)
hat_P_i <- Z_i * m_i + (1 - Z_i) * m                    # Credibility-adjusted claim frequency
hat_P_i

##        A        B        C        D        E
## 180.9165 337.8925 725.0477 687.0478 762.1102
```

PRINCIPAL COMPONENT ANALYSIS (PCA)

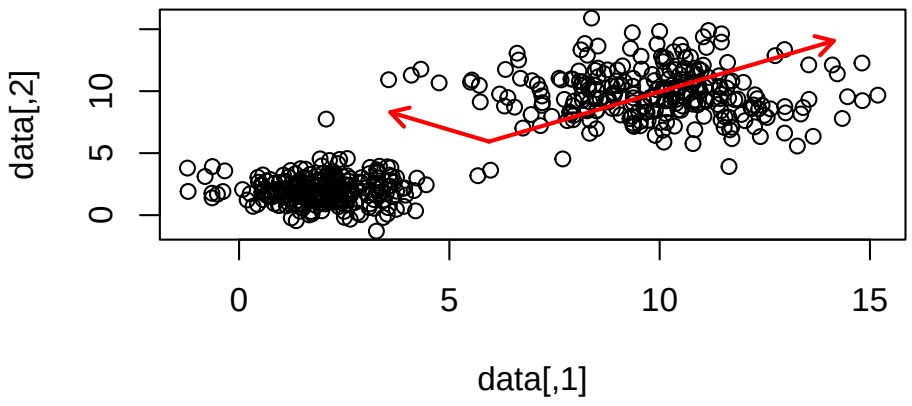
Computing PCA:

Example: From First Principles.

```
set.seed(10)
df <- rnorm(1000, c(2, 10), c(1, 2))
data <- matrix(df, 500, 2)

X <- scale(data, center = T, scale = F)  # scale the data
A <- t(X) %*% X                          # compute the unnormalized covariance matrix
W <- eigen(A)$vectors                    # the principal components of X
V <- eigen(A)$values                     # the squares of the singular values of X

explained.var <- V / sum(V)              # the % explained variance of each variable
```

Example: Using the built-in `prcomp(<>)` function.

```
                                         # data and df are the variables in the example above
pca <- prcomp(data)

pca$rotation                             # the principal components of X

vars <- pca$sdev^2
explained.var <- vars / sum(vars)        # the % explained variance of each variable

plot(data)                               # doesn't plot the principal components!
```



BOOTSTRAP INFERENCE

Parametric bootstrap:

Example: X is a sample of size 300 from a normal distribution $\mathcal{N}(10, 5^2)$. Let's compute the bootstrap empirical distribution of $\bar{X}$:

```
x <- rnorm(300, 10, 5)
N <- 300
# the replicate(<n>, <expr>) function generates a vector by evaluating <expr> <n> times.
bm <- replicate(10000, mean(rnorm(N, mean = mean(x), sd = sd(x))))

# The 95% CI of the sample mean is:
quantile(bm, c(0.025, 0.975))

##      2.5%     97.5%
##  9.744772 10.855255
```

Non-parametric bootstrap:

Example: X is now a sample of size 300 from an unknown distribution; let's compute the bootstrap empirical distribution of $\bar{X}$:

```
bm <- replicate(10000, mean(sample(x, replace = T)))

# The 95% CI of the sample mean is:
quantile(bm, c(0.025, 0.975))

##      2.5%     97.5%
##  9.731208 10.866606
```

NON-PARAMETRIC PERMUTATION TESTS

Test whether two sample means are equal using a **Random permutation test**:

Example: X_1 and X_2 are samples: we're interested in finding out whether or not the they come from the same population via Random Permutation Testing the sample means differences. In this case, we don't look at all permutations of indices.

```
X1 <- rnorm(100, 10, 5)                          # generate a normal sample X1
X2 <- rnorm(200, 5, 5)                           # generate a normal sample X2

n1 <- length(X1)                                 # n1 is the size of X1
result <- c(X1, X2)                              # concatenate the two samples together
index <- 1:length(result)                        # generate an index 1, 2, ..., n1 + n2
diff <- rep(0, 10000)                            # initialize 10k permutations

for (i in 1:10000) {
  p <- sample(index, n1, replace = F)            # sample n1 indexes out of the n1 + n2
  diff[i] <- mean(result[p]) - mean(result[-p]) # compute this sample means differences
}
obs <- abs(mean(X1) - mean(X2))                  # sample means differences
length(diff[abs(diff) >= obs]) / length(diff)    # probability that a permutation sample has
                                                 # greater diff than the observed one(i.e. a p-value!)
```

Permutation test: **Example:** X_1 and X_2 are samples: we're interested in finding out whether or not the they come from the same population via Permutation Testing the sample means differences.

```
X1 <- rnorm(10, 10, 5)                           # generate a normal sample X1; needs to be very small
X2 <- rnorm(10, 2, 5)                            # generate a normal sample X2; needs to be very small

n1 <- length(X1)                                 # n1 is the size of X1
result <- c(X1, X2)                              # concatenate the two samples together
index <- 1:length(result)                        # generate an index 1, 2, ..., n1 + n2

p <- combn(index, n1)                            # generate all combinations; explodes with n1 and n2!
n <- choose(n1 + n2, n1)                         # combinations n1 + n2 choose n1
diff <- rep(0, n)                                # initialize all combinations
for (i in 1:n) {
  diff[i] <- mean(result[p[,i]]) - mean(result[-p[,i]]) # compute this sample means differences
}
#P-value
obs <- abs(mean(X1) - mean(X2))                  # sample means differences
length(diff[abs(diff) >= obs]) / length(diff)    # probability that a permutation sample  has
                                                 # greater diff than the observed one(i.e. a p-value!)
```

CS2 - RISK MODELLING AND SURVIVAL ANALYSIS

CREATING FUNCTIONS

Syntax:

```
function.name <- function(arg1, arg2, ...) {
    <expressions>
    <last expression> # this one is returned by the function
}
```

Example: Creating pdf, cdf and random sampling function for Pareto distribution $X \sim \text{Pareto}(a, l)$.

```
dpareto <- function(x, a, l) { # pdf
  (a * l^a) / ((l+x)^(a+1))
}

ppareto <- function(x, a, l) { # cdf
  1 - (l / (l+x))^a
}

rpareto <- function(n, a, l) { # random sampler
  u <- runif(n)
  l * ((1-u)^(-1/a) - 1)
}
```

DIFFERENTIATION AND INTEGRATION

Symbolic Differentiation:

Example: Differentiate the function $f(x) = x^2 + 3x + 5$.

```
expr <- expression(x^2 + 3*x + 5)      # Define the function
D(expr, "x")                           # Differentiate the expression with respect to x

## 2 * x + 3
```

Numerical Integration:

Example: Estimate $\int_0^1 2x + 3dx$

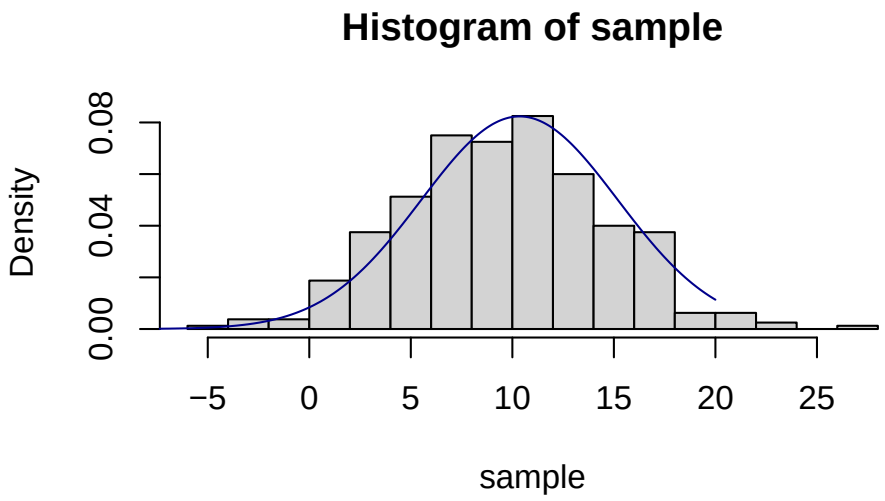
```
f <- function(x) { 2*x + 3 }                 # Define the function
integrate(f, lower = 0, upper = 1)           # Integrate the function from 0 to 1

## 4 with absolute error < 4.4e-14

integrate(f, lower = 0, upper = 1)$value     # Yields the value only

## [1] 4
```

LOSS DISTRIBUTION

Generating Maximum Likelihood Estimates (MLE) and **analysing the fit:**

Example: Use MLE to estimate the parameters of a normally distributed sample:

```
LogL <- function(x, params) {
  if (params[2] <= 0) {                          # sigma can never be negative;
    Inf                                          # make the log-likelihood infinite if it is!
  } else {
    -sum(log(dnorm(x, params[1], params[2])))    # the log-likelihood function for the sample X
  }                                              # and params (mu, sigma)
}

sample <- rnorm(200, 10, 5)
initial.params = c(1, 100)
MLE <- nlm(LogL, initial.params, x = sample)
MLE$estimate                                     # the estimate should be ~ the true parameters!

## [1] 10.353933 4.844735

hist(sample, freq = F, breaks = 10)
x <- seq(-20, 20, 0.05)
y <- dnorm(x, MLE$estimate[1], MLE$estimate[2])
lines(x, y, col = "darkblue")
```



MLE using the MASS package:

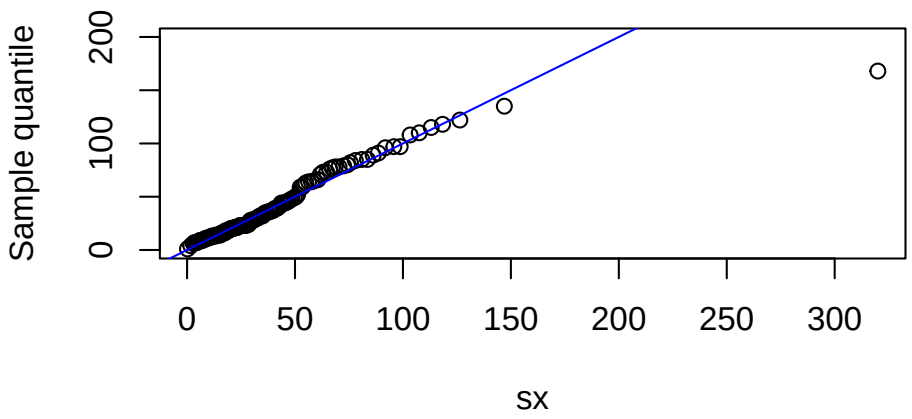
Example: Use the `fitdistr(<data>,<distribution>)` function to compute the MLE of parameters of a Gamma-distributed sample.

```
library(MASS)                                          # install.packages("MASS") if needed!

x <- rgamma(2000, 20, 0.5)                             # generate a sample from a Gamma distribution
fitdistr(x, "gamma")

##      shape         rate
##  19.98918983    0.50248312
## ( 0.62686916) ( 0.01595715)

# fitdistr(x, "gamma", list(shape = 20, rate = 0.5))  # if an initial candidate exists!
```

Using Q-Q plots:

Example: Generate estimates for the parameters using the Method of Moments and then check with a Q-Q plot.

```
ozone <- airquality$Ozone                              # airquality is a built-in dataset included in R
ozone <- ozone[!is.na(ozone)]

a <- mean(ozone)^2 / sd(ozone)^2                       # Estimating a and l using method of moments
l <- a / mean(ozone)

X <- rgamma(2000, a, l)
qqplot(X, ozone, ylab = "Sample quantile", ylim = c(0,200))
abline(0, 1, col = "blue")
```



REINSURANCE

Proportional Reinsurance:

Example: Suppose Insurance claims follow a distribution LogN$(5, 2)$ and the value of α is 75% (that is, the reinsurer pays 25% of the claims). How does this affect the payments?

```
set.seed(50)
claim <- rlnorm(1000, 5, 2)
a <- 0.75

paid.insurance <- a * claim
paid.reinsurer <- claim - paid.insurance
(mean(paid.insurance) - mean(claim)) / mean(claim) * 100  # % decrease in mean

## [1] -25

(sd(paid.insurance) - sd(claim)) / sd(claim) * 100        # % decrease in std. deviation

## [1] -25
```

Excess of Loss Reinsurance:

Example: Assume the claims are distributed as Pareto$(3, 600)$ and the retention limit $M = 1000$. What do the insurer payments look like?

```
set.seed(10)
rpareto <- function(n, a, l) {
  l * ((1 - runif(n))^(-1/a) - 1)
}
claim <- rpareto(10000, 3, 600)

m <- 1000
paid.insurance <- pmin(claim, m)
paid.reinsurer <- pmax(0, claim - m)
(mean(paid.insurance) - mean(claim)) / mean(claim) * 100

## [1] -15.14916

(sd(paid.insurance) - sd(claim)) / sd(claim) * 100

## [1] -48.57061
```

Inflation Case:

Example: Assume an inflation rate of 12% per annum. Assess the impact of inflation on the mean and standard deviation of claims paid by insurance and reinsurer, given that the retention limit $M = 1000$.

```
set.seed(10)
rpareto <- function(n, a, l) {
  l * ((1 - runif(n))^(-1/a) - 1)
}
claim <- rpareto(10000, 3, 600)

m <- 1000
paid.insurance <- pmin(claim, m)
paid.reinsurer <- pmax(claim - m, 0)

i <- 0.12
k <- (1+i)
paid.insurance.i <- pmin(k * claim, m)
paid.reinsurer.i <- pmax(k*claim - m, 0)

# insurer plays under inflation rate!
(mean(paid.insurance.i) - mean(paid.insurance)) / mean(paid.insurance) * 100

## [1] 9.299076

# reinsurer plays above inflation rate!
(mean(paid.reinsurer.i) - mean(paid.reinsurer)) / mean(paid.reinsurer) * 100

## [1] 27.12794
```

RISK MODELS

Collective Risk Models:

Example: Estimate the distribution of

$$S = X_1 + X_2 + X_3 + \cdots + X_N, \qquad N \sim \text{Poisson}(900), \qquad X_i \sim \text{Gamma}(680, 0.23)$$

```
set.seed(10)


s <- rep(0, 10000)
for (i in 1:10000) {
  n <- rpois(1, lambda = 900)
  s[i] <- sum(rgamma(n, shape = 680, rate = 0.23))
}
cat("Mean:", mean(s), "| Variance:", var(s))

## Mean: 2661070 | Variance: 7768288482

length(s[s > 2500000]) / length(s)

## [1] 0.967
```

Example: Estimate the distribution of S, but with a retention limit of 2 100.

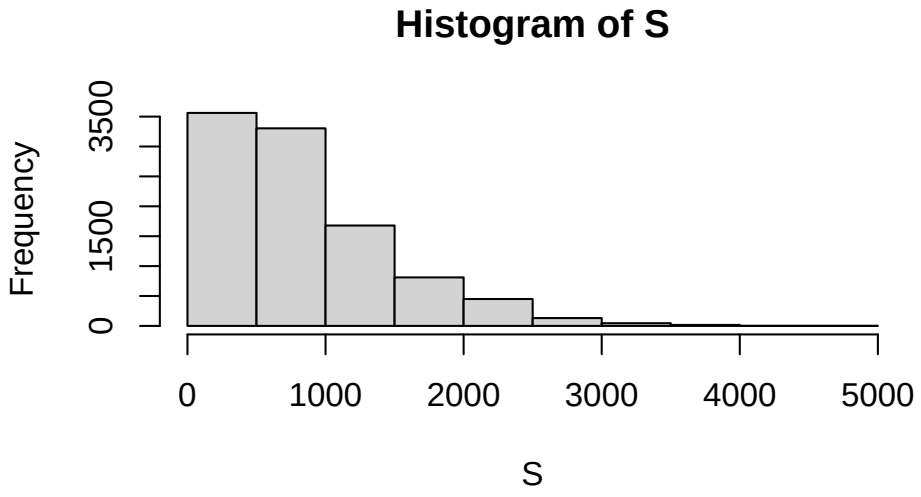
```
si <- rep(0, 10000)
sr <- rep(0, 10000)
m <- 2100
for (i in 1:10000) {
  n <- rpois(1, lambda = 900)
  x <- (rgamma(n, shape = 600, rate = 0.23))
  si[i] <- sum(pmin(x, m))
  sr[i] <- sum(pmax(0, x - m))
}
cat("Mean:", mean(si), "| Variance:", var(si))

## Mean: 1891784 | Variance: 4046519806

cat("Mean:", mean(sr), "| Variance:", var(sr))

## Mean: 458234.3 | Variance: 248071912
```

Individual Risk Model:

Example: Company XYZ has sold 1200 policies; 300 to smokers ($q_S = 0.002$) and 900 to non-smokers ($q_{NS} = 0.001$).

$$S = Y_1 + Y_2 + \cdots + Y_{300} + Z_1 + \cdots + Z_{900}, \quad Y_i \sim \text{Gamma}(220, 0.75), \quad Z_i \sim \text{Gamma}(310, 0.45)$$

```r
set.seed(10)
S <- rep(0, 10000)
for (j in 1:10000) {
  Y <- rgamma(300, shape = 220, rate = 0.75)
  Z <- rgamma(900, shape = 310, rate = 0.45)
  T <- c(Y, Z)
  DY <- rbinom(300, 1, 0.002)
  DZ <- rbinom(900, 1, 0.001)
  D <- c(DY, DZ)
  S[j] <- sum(T * D)
}

p.700 <- length(S[S > 700]) / length(S)
cat("Mean:", mean(S), "| Variance:", var(S), "| P(S > 700)", p.700)

## Mean: 795.0162 | Variance: 475206.3

hist(S)
```



SURVIVAL MODELS

Creating a function for Makeham's Law:

Example: Assuming Makeham's Law, $\mu(x) = A + B * C^x$, we will compute the survival

$$ {}_tp_x = S_x(t) = P[T_x > t] = \exp\left(-\int_0^t \mu(x+s)ds \right) $$

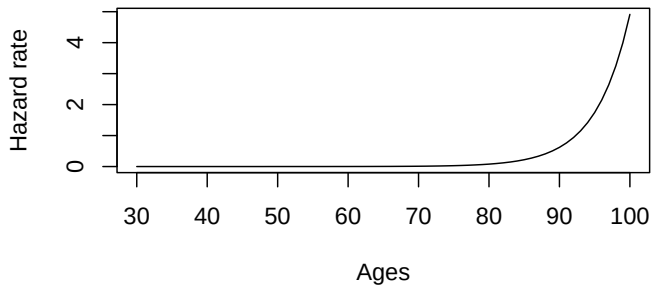
first using first principles, then numerically. After that we compute the expectation $\mathring{e}_x$ and the curtate expectation e_x.

```
p <- list(A = 0.00004, B = 0.0000025, C = 1.1)    # store the model parameters in a list
mu <- function(x) {
  p$A + p$B * p$C^x
}

# Let's compute 10p35
tpx1 <- function(x, t) {
  int <- p$A * t + p$B * (p$C^t - 1) * (p$C^x) / log(p$C)
  exp(-int)
}

tpx2 <- function(x, t) {
  i <- integrate(mu, x, x + t)$value
  exp(-i)
}

tpx1(35, 10)

## [1] 0.9984264

tpx2(35, 10)

## [1] 0.9984264

integrate(tpx1, 0, Inf, x = 50)$value           # E[mu]

## [1] 54.75777

sum(tpx1(50, 1:1000))                           # the curtate expectation at 50 years old

## [1] 54.2578
```

The flexsurv library: The (automatically loaded) `stats` package does not include functions to deal with the Gompertz distribution. That's where the `flexsurv` function comes in.

```r
library(flexsurv)

rate <- 0.0000025                                          # beta (rate parameter)
C <- 1.23                                                  # C = exp(alpha)
shape <- log(C)                                            # alpha (shape parameter)

p <- list(A = 0, B = rate, C = C)

x <- 30
t <- 25
Sxt <- pgompertz(x+t, shape, rate, lower.tail = FALSE)     # S(x + t)
Sx <- pgompertz(x, shape, rate, lower.tail = FALSE)        # S(x)
tpx <- Sxt / Sx
tpx

## [1] 0.3473415

# Simulate 10 lifetimes of lives currently aged 50
set.seed(42)
rgompertz(10, shape, rate * C^50)

##  [1]  9.745937 10.235528  3.080972  8.405690  6.342800  5.206165  7.299388
##  [8]  1.566020  6.490136  6.969379

# Hazard rate at age 60
hgompertz(30, shape, rate)                                 # the same as rate * exp(shape * 30)

## [1] 0.001244782
```

```r
# Plot the hazard rate over the age range 30 to 100
x <- seq(30, 100)
y <- hgompertz(x - 30, shape, rate)
plot(x, y, type = "l", xlab = "Ages", ylab = "Hazard rate")
```



```
1 - exp(-Hgompertz(50, shape, rate))
```

```
## [1] 0.3145845
```

```
pgompertz(50, shape, rate)
```

```
## [1] 0.3145845
```

```
mu <- function(x) {
  p$B * p$C^x
}
```

```
mu(55)
```

```
## [1] 0.2201512
```

```
hgompertz(55, shape, rate)
```

```
## [1] 0.2201512
```

```
integrate(mu, 0, 45)$value
```

```
## [1] 0.1341624
```

```
Hgompertz(45, shape, rate)
```

```
## [1] 0.1341624
```

Survival Distributions:

1. Exponential distribution: $T_x \sim \text{Exp}(0.075)$

```
# P(Tx <= 30)
lambda <- 0.075
pexp(30, lambda)
```

```
## [1] 0.8946008
```

2. Weibull Distribution: $T_x \sim \text{Weibull}(0.0002, 2.5)$

```
shape <- 2.5
scale <- 0.0002^(-1 / shape)

# P(Tx <= 30)
pweibull(30, shape, scale)
```

```
## [1] 0.6268969
```

PROPORTIONAL HAZARDS MODELS

Example: Estimating β from First Principles – according to the Cox Proportional Hazards model.

Consider a study where patients either die within 5 years after surgery or survive beyond 5 years (censored). Surgeries took place at Hospital A or Hospital B; we want to determine if the hospital affects survival – in particular the relative hazard of surgery at Hospital A compared to Hospital B (in this case it's $exp(\beta)$).

```
data <- data.frame(X = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1),  # Covariate: 1 if Hospital A, 0 if B
                   T = c(60, 32, 60, 60, 60, 4, 18, 60, 9, 31, 53, 17),  # Time of death in months
                   C = c(0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1))            # 0 if censored, 1 if death

neg_log_partial_likelihood <- function(beta, df) {
  events <- which(df$C == 1)
  event_times <- T[events]
  sorted_event_indices <- events[order(event_times)]

  log_PL <- 0
  for (i in sorted_event_indices) {
    risk_set <- which(df$T >= df$T[i])
    new.term <- beta * df$X[i] - log(sum(exp(beta * df$X[risk_set])))
    log_PL <- log_PL + new.term
  }

  -log_PL
}

mle <- nlm(neg_log_partial_likelihood, 0, df = data)

beta_hat <- mle$estimate
cat("Estimated beta (regression coefficient):", beta_hat, "\n")

## Estimated beta (regression coefficient): 2.118368

cat("Hazard ratio (exp(beta)):", exp(beta_hat), "\n")

## Hazard ratio (exp(beta)): 8.317556
```

Using the survival Library:

Example: Estimating β from the Cox PH model using the `survival` library.

```
library(survival)
data <- data.frame(X = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1),  # Covariate: 1 if Hospital A, 0 if B
                   T = c(60, 32, 60, 60, 60, 4, 18, 60, 9, 31, 53, 17),  # Time of death in months
                   C = c(0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1))            # 0 if censored, 1 if death

surv_object <- Surv(time = data$T, event = data$C)

cox_model <- coxph(surv_object ~ X, data = data)
summary(cox_model)

## Call:
## coxph(formula = surv_object ~ X, data = data)
##
##   n= 12, number of events= 7
##
##    coef exp(coef) se(coef)     z Pr(>|z|)
## X 2.118     8.318    1.092 1.939   0.0525 .
## ---
##   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##   exp(coef) exp(-coef) lower .95 upper .95
## X     8.318     0.1202    0.9776     70.77
##
## Concordance= 0.741  (se = 0.068 )
## Likelihood ratio test= 5.54  on 1 df,   p=0.02
## Wald test            = 3.76  on 1 df,   p=0.05
```

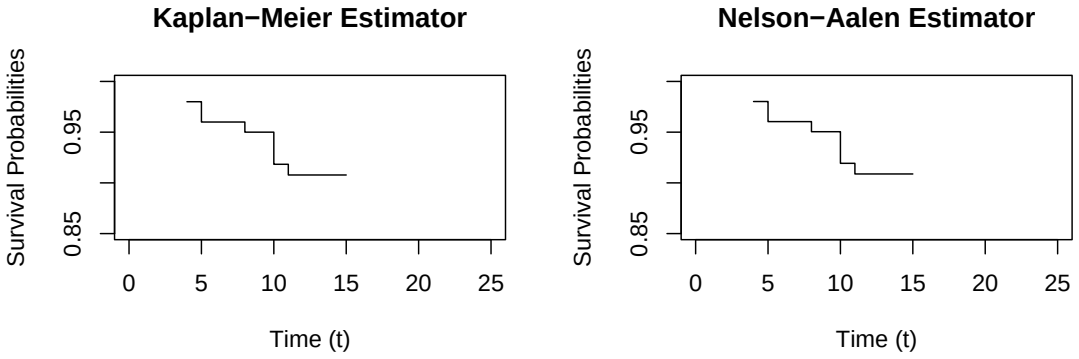
ESTIMATING THE LIFETIME DISTRIBUTION FUNCTION

Kaplan-Meier:

```
data <- data.frame(tj = c(  4,  5,  8, 10, 11, 15),
                   dj = c(  2,  2,  1,  3,  1,  0),
                   nj = c(100, 98, 96, 90, 87, 86))

survival.km <- cumprod(1 - data$dj / data$nj)

plot(data$tj, survival.km, type = "s",
     xlim = c(0, 25), ylim = c(0.85, 1),
     xlab = "Time (t)", ylab = "Survival Probabilities",
     main = "Kaplan-Meier Estimator")
```

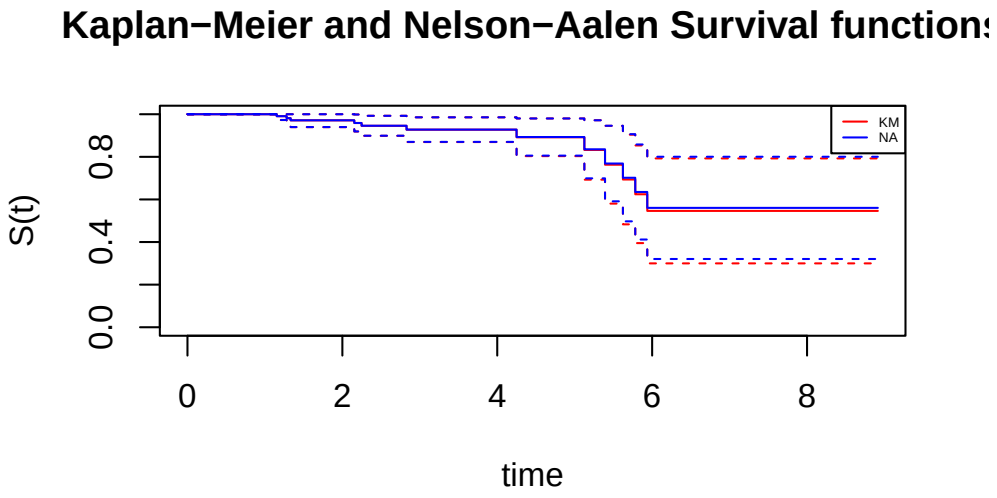
Nelson-Aalen:

```
# data is the same as in the previous example!

lambda <- cumsum(data$dj / data$nj)
survival.na <- exp(-lambda)

plot(data$tj, survival.na, type = "s",
     xlim = c(0, 25), ylim = c(0.85, 1),
     xlab = "Time (t)", ylab = "Survival Probabilities",
     main = "Nelson-Aalen Estimator")

cbind(data$tj, survival.km, survival.na) # Kaplan-Meyers < Nelson-Aalen!
```



Using the Survival library:

```
library(survival)

# generate a synthetic dataset
set.seed(10)
n <- 200
event_time <- rweibull(n, shape = 2, scale = 10)
censor_time <- rexp(n, rate = 0.5)
observed_time <- pmin(event_time, censor_time)
status <- ifelse(event_time <= censor_time, 1, 0)

# Create the survival data frame
surv.data <- data.frame(Patient = 1:n, event.time = observed_time, event.code = status)

#Creating a Survival Object
surv.obj <- Surv(surv.data$event.time, surv.data$event.code)

#Generating estimates
survival.km <- survfit(surv.obj ~ 1, conf.type = "plain")
survival.na <- survfit(surv.obj ~ 1, conf.type = "plain", stype = 2)

plot(survival.km, main = "Kaplan-Meier and Nelson-Aalen Survival functions",
     xlab = "time", ylab = "S(t)", col = "red")
lines(survival.na, col = "blue")
legend("topright", legend = c("KM", "NA"), cex = 0.4, col = c("red", "blue"), lwd = 1)
```



MORTALITY RATES

Example: Compute μ_{70} for the insured population in this example, between 2013 and 2017. In this case, each person is exposed while they are

- alive
- between 70 and 71 years of age
- within the study period (2013 and 2017)
- insured

```
# Creating a synthetic dataset
set.seed(10)
n <- 1000
data <- data.frame(ID = 1:n, BIRTH = runif(n, 1920, 1955),
                   DEATH = NA, ENTRY = runif(n, 2010, 2015))
data$DEATH <- ifelse(runif(n) < 0.8, data$BIRTH + runif(n, 70, 90), NA)

d.70 <- data$BIRTH + 70                    # 70th birthday
d.71 <- data$BIRTH + 71                    # 71st birthday
d.0 <- 2013                                # investigation start date
d.1 <- 2017                                # investigation end date
data$DEATH[is.na(data$DEATH)] <- d.1        # if someone dies after d1, consider d1

d.A <- pmax(d.70, d.0, data$ENTRY)
d.B <- pmin(d.71, d.1, data$DEATH)

exposure <- pmax(0, d.B - d.A)             # ignore negative exposures!
ecx <- sum(exposure)

pop <- data$DEATH > d.0 & data$DEATH < d.1 # consider only deaths in the study period
age <- data$DEATH[pop] - data$BIRTH[pop]   # age at death
d <- age[age >= 70 & age < 71]             # age of death is 70 years old
mu70 <- length(d) / ecx
mu70                                       # deaths per person.year

## [1] 0.06773931
```

MARKOV CHAINS

Markov Chain library: Install (using `install.packages(<>)`) and load (using `library(<>)` the package `markovchain` before running this code.

```
P <- matrix(c(0.25, 0.75, 0.2, 0.8), 2, 2, byrow = T)  # transition probability matrix
states <- c("S1", "S2")
mc <- new("markovchain", transitionMatrix = P, states = states)

rowSums(mc@transitionMatrix)    # notice the @ when accessing properties of mc!
is.irreducible(mc)

## [1] TRUE

period(mc)

## [1] 1
```

Example: Computing steady-state behaviour:

```
exp.dist <- function(start, P, n){
  pos <- start
  for (i in 1:n) {
    pos <- pos %*% P        # the operator %*% is matrix multiplication
  }
  pos
}

X0 <- c(0.1, 0.9)
exp.dist(X0, mc@transitionMatrix, 10) # equivalently, this package allows for the syntax: X0 * mc^10

##             S1        S2
## [1,] 0.2105263 0.7894737

steadyStates(mc)

##             S1        S2
## [1,] 0.2105263 0.7894737
```

Premium Calculations:

Example: Computing premiums using first principles.

```
discount <-  c(0, 0.15)
prem <- 100 * (1 - discount)
P <- matrix(c(0.1, 0.9, 0.05, 0.95), 2, 2, byrow = T)
P

##      [,1] [,2]
## [1,] 0.10 0.90
## [2,] 0.05 0.95

X0 <- c(0.5, 0.5)
prem.inc <- 0
for (i in 1:4) {
  prem.inc <- prem.inc + sum(X0 * prem)
  X0 <- X0 %*% P
}
prem.inc

## [1] 350.2216
```

Example: Computing premiums using the `markovchain` package.

```
discount <-  c(0, 0.15)
prem <- 100 * (1 - discount)
P <- matrix(c(0.1, 0.9, 0.05, 0.95), 2, 2, byrow = T)

states <- c("0%", "25%")
mc <- new("markovchain", transitionMatrix = P, states = states)

X0 <- c(0.5, 0.5)
X0 %*% expectedRewards(mc, 4 - 1, prem)

##          [,1]
## [1,] 350.2216
```

Sampling: **Example:** Sampling from a Markov Chain and fitting a Markov Chain from a sequence of states, using the `markovchain` library.

```
P <- matrix(c(0.35, 0.65, 0.8, 0.2), 2, 2, byrow = T)
P

##      [,1] [,2]
## [1,] 0.35 0.65
## [2,] 0.80 0.20

states <- c("0%", "20%")
mc <- new("markovchain", transitionMatrix = P, states = states)

#Sampling
#rmarkovchain(<n>,<mc>,t0="start state",include.t0=TRUE)
set.seed(10)
data <- rmarkovchain(5, mc, t0 = "20%", include.t0 = T) # generates a vector/sequence of states
data

## [1] "20%" "0%"  "20%" "0%"  "20%" "0%"

#Fitting the markov chain
data <- rmarkovchain(20, mc, t0 = "20%", include.t0 = T)
fit <- markovchainFit(data)
fit$estimate

## MLE Fit
##  A  2 - dimensional discrete Markov Chain defined by the following states:
##  0%, 20%
##  The transition matrix  (by rows)  is defined as follows:
##            0%       20%
## 0%  0.5384615 0.4615385
## 20% 0.8571429 0.1428571
```

Example: Simulating a time-inhomogeneous Markov chain

$$P(t) = \begin{pmatrix} 1 - 0.015t & 0.015t \\ 1 - 0.02t & 0.02t \end{pmatrix}$$

```
prob_01 <- function(x){
  0.015 * x
}

prob_11 <- function(x){
  0.02 * x
}

px <- function(x) {
  matrix(c((1 - prob_01(x)), prob_01(x), 1 - prob_11(x), prob_11(x)), 2, 2, byrow = T)
}

fx <- function(age, years, pos){
    for (i in age:(age + years - 1)) {
        pos <- pos %*% px(i)
    }
    pos
}

# starting from the first state at 35 years old, these are the state probabilities 10 years later:
fx(35, 10, c(1, 0))

##           [,1]      [,2]
## [1,] 0.1606768 0.8393232
```

MARKOV JUMP PROCESSES

Time-homogeneous case:

Example: In a continuous-time Markov Chain with generator matrix (λ_is in year^{-1})

$$G = \begin{pmatrix} -1 & 0.3 & 0.7 \\ 0.8 & -1.5 & 0.7 \\ 0.6 & 1.2 & -1.8 \end{pmatrix}$$

what's the probability of going from state 2 to state 3 in a month?

```
states <- paste("State", 1:3)
GM <- matrix(c(-1, 0.3, 0.7, 0.8, -1.5, 0.7, 0.6, 1.2, -1.8),
             3, 3, byrow = T, dimnames = list(states, states))

# only works for small time steps!!
h <- 1 / 12                 # a month is 1 / 12 of a year
ph <- diag(3) + h * GM      # diag(3) is the three-dimensional identity matrix
ph[2, 3]

## [1] 0.05833333
```

Example: Now we're interested in the same probability, but over 30 years (we already computed it for one month!)

```
# ph and states as in the example above
library("markovchain")
markov_chain <- new("markovchain", transitionMatrix = ph, states = states)
markov_chain ^ (12 * 30)

## Unnamed Markov chain^360
##  A  3 - dimensional discrete Markov Chain defined by the following states:
##  State 1, State 2, State 3
##  The transition matrix  (by rows)  is defined as follows:
##           State 1   State 2 State 3
## State 1 0.4133333 0.3066667    0.28
## State 2 0.4133333 0.3066667    0.28
## State 3 0.4133333 0.3066667    0.28
```

Simulating a Markov jump process:

Example: Let's use the approximation $P \approx I_n + h \cdot G$, valid for small h.

```
states <- paste("state",1:3)
GM <- matrix(c(-1, 0.3, 0.7, 0.8, -1.5, 0.7, 0.6, 1.2, -1.8),
             3, 3, byrow = T, dimnames = list(states, states))

ph <- function(h, GM){
  diag(nrow(GM)) + h * GM
}

library(markovchain)
set.seed(10)
sims <- rmarkovchain(1000, markov, include.t0 = T)
head(sims)

## [1] "state 3" "state 3" "state 3" "state 3" "state 3" "state 3"
```

Example: Now for an exact method:

```
# GM is as above
library(markovchain)
set.seed(10)

TPM <- generatorToTransitionMatrix(GM)
markov <- new("markovchain", transitionMatrix = TPM)
sims <- rmarkovchain(1000, markov, include.t0 = T)
head(sims)

## [1] "state 3" "state 2" "state 1" "state 3" "state 2" "state 1"
```

Example: Let's simulate the waiting times for `sims`.

```
ljs = -diag(GM)                          # the diagonal values of the generator matrix are the
waiting_times <- rexp(1000, ljs[sims2])  # rate parameters of the waiting times (exp.) distributions.
head(waiting_time)

## [1] 0.2335325 0.8978444 0.4619815 0.3163581 0.2385942 0.9672260
```

Time-Inhomogeneous: **Example:** Simulating a time-inhomogenous Markov jump process. We use the approximation

$$P(t, t+h) \approx I + h \cdot \mathrm{GM}(t)$$

and iterate it:

$$P(t, s) \approx P(t, t+h) \cdots P(t+h, t+2h) \cdots P(s-h, s)$$

```
states <- paste("State", 1:3)
GM <- function(t){
  matrix(c(-t, 0.3*t, 0.7*t,
           0.8*t, -1.5*t, 0.7*t,
           0.6*t, 1.2*t, -1.8*t), 3, 3, byrow = T, dimnames = list(states, states))
}

Ph <- function(h, GM) {
  diag(nrow(GM)) + h*GM
}

Pts <- function(t, s, h){
  M <- diag(length(states))
  dimnames(M) <- list(states, states)
  n <- (s-t) / h - 1
  for (i in 0:n) {
    M <- M %*% Ph(h, GM(t + h*i))
  }
  M
}

Pts(1, 2, 1/10000)
```

```
##           State 1   State 2   State 3
## State 1 0.4562438 0.2703372 0.2734190
## State 2 0.3890577 0.3375233 0.2734190
## State 3 0.3765769 0.3265005 0.2969225
```

TIME SERIES

Simulating a Stationary Time Series

$$y_t = \underbrace{0.6y_{t-1} - 0.2y_{t-2}}_{\text{AR}} + \underbrace{e_t + 0.3e_{t-1}}_{\text{MA}} \qquad (\text{ARMA}(2,\,1) \qquad e_t \sim \mathcal{N}(0, 5^2)$$

```
set.seed(10)
yt <- arima.sim(n = 10, list(ar = c(0.6, -0.2), ma = 0.3), sd = 5)
yt

## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##  [1]  8.3867410  4.6225682  5.6760659  7.6692436  5.0251545 -3.1594544
##  [7] -5.3088712  1.7814489  5.9338181  0.9469157
```

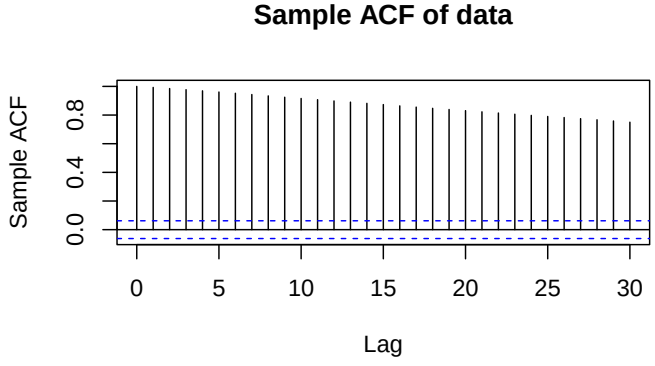
Simulating a non-stationary Time Series:

$$x_t = x_{t-1} + e_t \qquad e_t \sim \mathcal{N}(0, 5^2) \qquad \text{a random walk!}$$

```
set.seed(50)
N <- 10
et <- rnorm(N, mean = 0, sd = 5)
xt <- cumsum(et)
```

Using the ACF to determine if the series needs differencing to become stationary:

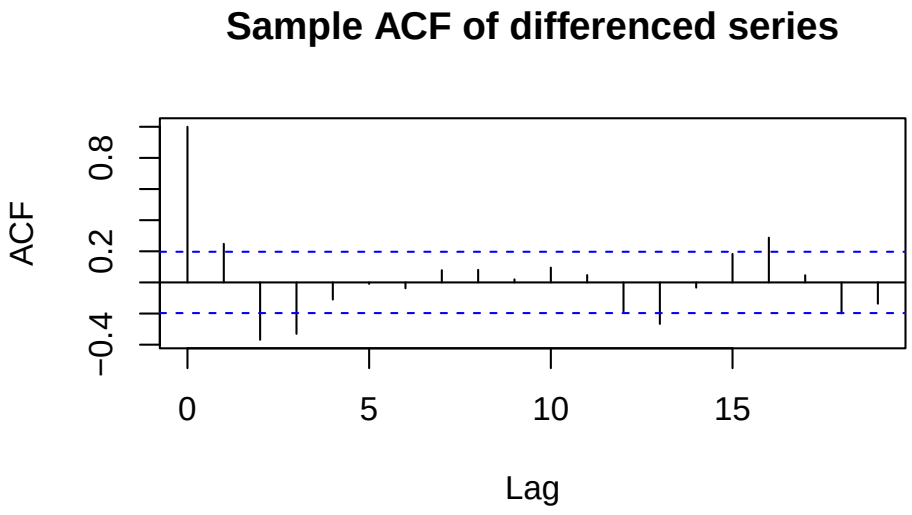
```
N <- 100
et <- rnorm(n, mean = 0, sd = 5)
xt <- cumsum(et)
xt

acf(xt, main = "Sample ACF of data", ylab = "Sample ACF")
# Differencing is needed, since the decay in the ACF is slow.
```



Phillips-Perron test:

```
PP.test(xt)  # H0 is the hypothesis that the series has a unit root (and therefore is not stationary).

##
##  Phillips-Perron Unit Root Test
##
## data:  xt
## Dickey-Fuller = -1.9202, Truncation lag parameter = 7, p-value = 0.6121

# since the p-value is greater than the significance level of 0.05, we don't reject H0.
```

```
# take differences:
zt <- diff(xt, lag = 1, differences = 1)
acf(zt, main = "Sample ACF of differenced series")
```




```
PP.test(zt)

##
##  Phillips-Perron Unit Root Test
##
## data:  zt
## Dickey-Fuller = -7.2067, Truncation lag parameter = 3, p-value = 0.01
```

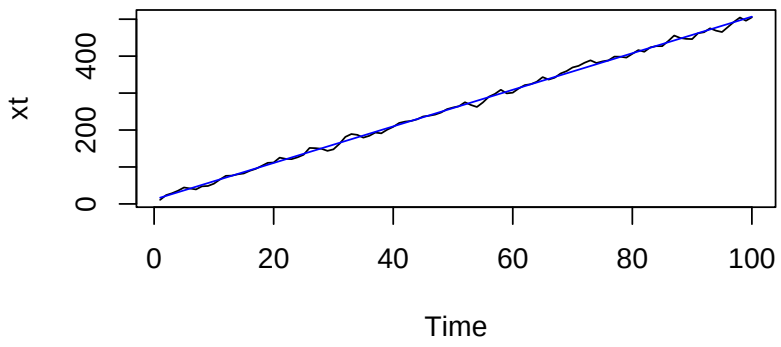
This time, H_0 should be rejected; no further differencing seems to be needed. This is also visible in the ACF plot – the decay of autocorrelations is very swift.

A heuristic to pick the order of differencing is to take the point where the variance of the series goes up!

```
ut <- xt
for (i in 0:4) {
    print(paste0("order = ", i, ", var = ", var(ut)))     # paste0 concatenates strings!
    ut <- diff(ut, lag = 1, differences = 1)
}

## [1] "order = 0, var = 21044.8336811315"
## [1] "order = 1, var = 53.5795254513324"
## [1] "order = 2, var = 79.279157622838"
## [1] "order = 3, var = 173.982472108447"
## [1] "order = 4, var = 473.573641010363"
```

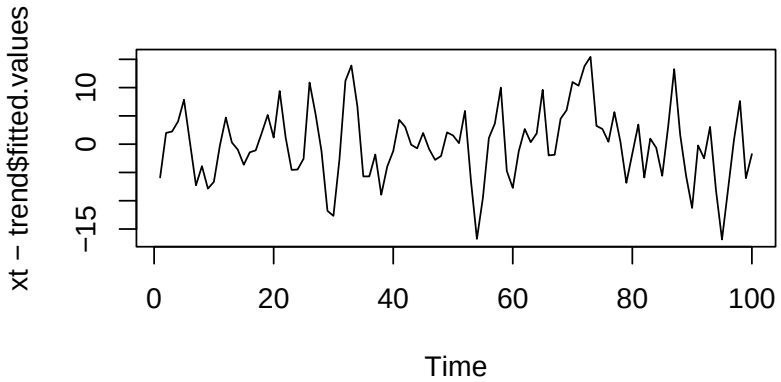
Identifying and removing trends using linear least-squares: **Example:** Identifying the trend

```
N <- 100
yt <- arima.sim(n = N, list(ar = c(0.6, -0.2), ma = 0.3), sd = 5)
xt <- 10 + 5 * (1:N) + yt
ts.plot(xt)
t <- seq(1:N)
trend <- lm(xt ~ t)
lines(t, trend$fitted.values, type = "l", col = "blue")
```



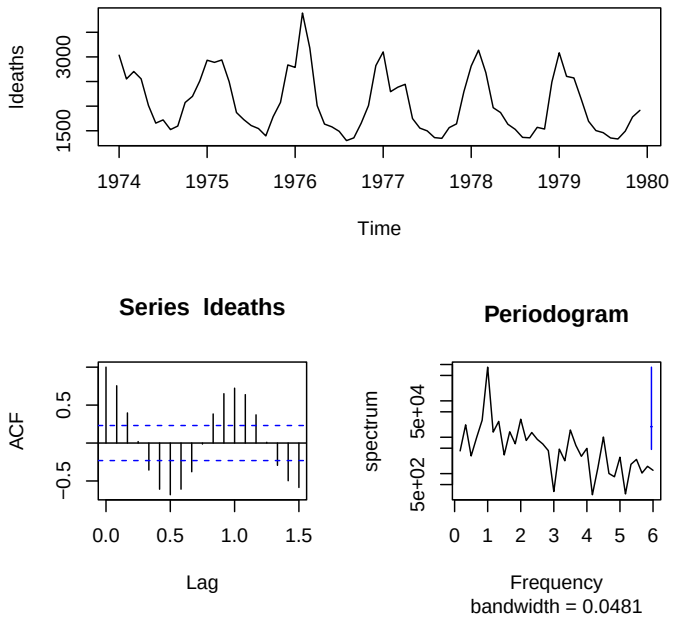
Example: Removing the trend

```
ts.plot(xt - trend$fitted.values)
```

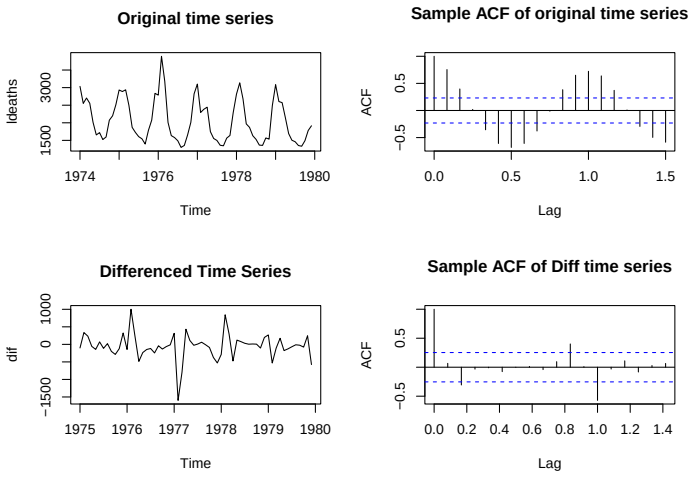


Identifying and Removing Seasonality:

```
# ldeaths is a built-in time series data set
plot(ldeaths)
acf(ldeaths)
spectrum(ldeaths, main = "Periodogram", xlab = "Frequency")
# the 1-year seasonality is clear by visual inspection of all plots!
```




```
# removing the seasonality
dif <- diff(ldeaths, lag = 12, differences = 1)
plot(ldeaths, main = "Original time series")
acf(ldeaths, main = "Sample ACF of original time series")
plot(dif, main = "Differenced Time Series")
acf(dif, main = "Sample ACF of Diff time series")
```



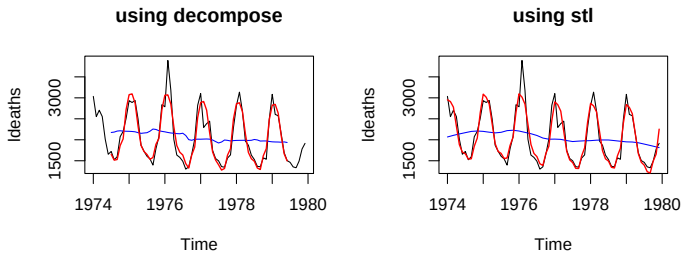
Identifying and removing seasonality using built-in functions:

Example: Using `decompose(<>)`

```
d <- decompose(ldeaths, type = "additive")
plot(ldeaths)
lines(d$trend, col = "blue")
lines(d$seasonal, col = "green", lwd = 1.3)
lines(d$seasonal + d$trend, col = "red", lwd = 1.3)
```

Example: Using `stl(<>)`

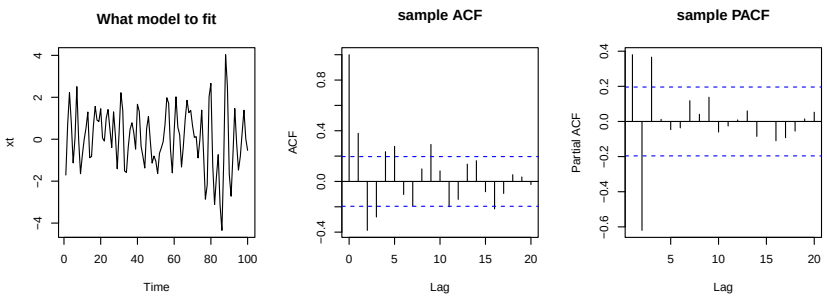
```
d <- stl(ldeaths, s.window = "periodic")
plot(ldeaths)
lines(d$time.series[, "trend"], col = "blue")
lines(d$time.series[, "seasonal"] + d$time.series[, "trend"], col = "red", lwd = 1.3)
```



Fitting a time series model: For MA(q) models, we determine the order by looking at the ACF and counting how many lags it takes to "cut off", and confirming that the ACF "tails off" (decays exponentially).

For AR(p) models, we determine the order by looking at the PACF and counting how many lags it takes to cut off, and confirming that the ACF tails off.

```
xt <- arima.sim(list(order = c(3, 0, 0),        # simulating an AR(3) process
                ar = c(1, -1, 1/2)), n = 100)

plot(xt, main = "Time series")
acf(xt, main = "sample ACF")                    # the ACF tails off
pacf(xt, main = "sample PACF")                  # the PACF cuts off at lag 3
```



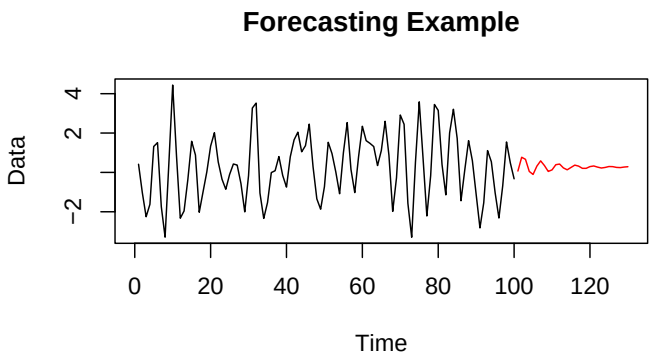
Example: Fitting an ARIMA model (i.e. determining the coefficients): the lower the AIC the better!

```
xt <- arima.sim(list(order = c(0, 0, 3), ma = c(1, -1, 1/2)), n = 100)
arima(xt, order = c(0, 0, 3))

##
## Call:
## arima(x = xt, order = c(0, 0, 3))
##
## Coefficients:
##           ma1      ma2     ma3  intercept
##       -0.1770  -0.2840  0.1213     0.2499
## s.e.   0.1008   0.1186  0.1215     0.1402
##
## sigma^2 estimated as 4.453:  log likelihood = -216.68,  aic = 443.37

arima(xt, order = c(0, 0, 4))$aic

## [1] 440.5388
```

Time Series Forecasting:

Example: Forecasting a time series model

```
xt <- arima.sim(list(order = c(3, 0, 0), ar = c(1, -1, 1/2)), n = 100)
model <- ar(xt)
preds <- predict(model, n.ahead = 30)

plot(xt, xlim = c(0, 130), main = "Forecasting Example", ylab = "Data")
lines(preds$pred, col = "red")
```



Exponential Smoothing:

```
set.seed(50)
N <- 100
etx <- rnorm(N, mean = 0, sd = 5)
xt <- cumsum(etx)
xt <- ts(xt, start=c(2020, 1), frequency = 12)
HW <- HoltWinters(xt, alpha = 0.7, beta = F, gamma = F)
HW

## Holt-Winters exponential smoothing without trend and without seasonal component.
##
## Call:
## HoltWinters(x = xt, alpha = 0.7, beta = F, gamma = F)
##
## Smoothing parameters:
##  alpha: 0.7
##  beta : FALSE
##  gamma: FALSE
##
## Coefficients:
##        [,1]
## a -58.94712

predict(HW, n.ahead = 1, level = 0.95, prediction.interval = T)  # yields a confidence interval

##                fit       upr       lwr
## May 2028 -58.94712 -48.66337 -69.23087

HW2 <- HoltWinters(xt, beta = F, gamma = F)                      # uses the optimal alpha
predict(HW2, level = 0.95, prediction.interval = T)

##                fit       upr       lwr
## May 2028 -56.76395 -47.00935 -66.51856
```

Multivariate Time Series:

Example: Co-integrated time series (two series X_t and Y_t, both $I(1)$, such that there is a stationary linear combination $aX_t + bY_t$).

```
# Generate two random walks (they are I(1)).
set.seed(50)
N <- 10
etx <- rnorm(N, mean = 0, sd = 5)
xt <- cumsum(etx)

ety <- rnorm(N, mean = 0, sd = 10)
yt <- cumsum(ety)

find.cointegrated <- function(x) {      # the lowest the p-value the most likely
  comb <- xt*x[1] + yt*x[2]             # for the linear combination to be stationary
  PP.test(comb)$p.value
}

v <- c(1, 1)                            # initial search parameters are a = 1, b = 1
X <- nlm(find.cointegrated, v)          # nonlinear minimization (Newton-like algorithm)

X$estimate

## [1] 1.924800 0.470916

# Confirm stationarity.
zt <- X$estimate[1]*xt + X$estimate[2]*yt
PP.test(zt)

##
##  Phillips-Perron Unit Root Test
##
## data:  zt
## Dickey-Fuller = -4.0434, Truncation lag parameter = 2, p-value =
## 0.02174
```

Example: Stationarity of multivariate time series

$$\begin{cases} X_t = 0.3X_{t-1} + 0.45Y_{t-1} + e_{t,x} \\ Y_t = 0.2X_{t-1} + 0.4Y_{t-1} + e_{t,y} \end{cases}$$

```
A <- matrix(c(0.3, 0.2, 0.45, 0.4), 2, 2)
eigen(A)$values

## [1] 0.65413813 0.04586187

# Both values have absolute value less than 1, so the process is stationary.
```