.daddy\_code v1.0

Rasmus Hammar

Table of contents

# 1. Allmänt

Lite allmänna funktioner som är bra att kunna.

## 1.1 Paket & inläsning av data

För att installera ett paket.

# För att installera paketet som heter palmerpenguins  
install.packages("palmerpenguins")  
  
# Stats  
install.packages("MKinfer")  
install.packages("lmboot")  
install.packages("boot.pval")  
  
# Matte  
install.packages("pracma")  
install.packages("expm")  
install.packages("deSolve")  
  
# KvantBio  
install.packages("plotrix")  
install.packages("scales")

För att använda funktioner i ett paket måste man ladda in paketet i R genom library()

# Exempel data  
library(palmerpenguins)  
  
# Stats  
library(MKinfer)  
library(lmboot)  
library(boot.pval)  
  
# Matte  
library(pracma)  
library(expm)  
library(deSolve)  
  
# KvantBio  
library(plotrix)  
library(scales)

För att läsa in data från t.ex. en .csv eller .txt fil. Öppna filen och identifiera vilka symboler som **sep**arerar kolumner samt används som **dec**imaltecken.

file.choose() # Returnerar vägen till filen man väljer, hitta fönstret med alt + tab.  
  
my\_data <- read.table(  
 file = file.choose(), # Välj fil interaktivt, alt. filens sökväg som en text string  
 header = TRUE, # Om det finns namn på kolumnerna  
 sep = ",", # Här används ett kommatecken för att skilja på kolumner  
 dec = "." # Här används en punkt som decimaltecken  
)

För att titta på data kan man använda

# View(penguins) # Öppnar som separat flik  
head(penguins, # Printar de första n raderna  
 n = 3)

# A tibble: 3 × 8  
 species island bill\_length\_mm bill\_depth\_mm flipper\_length\_mm body\_mass\_g  
 <fct> <fct> <dbl> <dbl> <int> <int>  
1 Adelie Torgersen 39.1 18.7 181 3750  
2 Adelie Torgersen 39.5 17.4 186 3800  
3 Adelie Torgersen 40.3 18 195 3250  
# ℹ 2 more variables: sex <fct>, year <int>

tail(penguins, # Printar de sista n raderna  
 n = 3)

# A tibble: 3 × 8  
 species island bill\_length\_mm bill\_depth\_mm flipper\_length\_mm body\_mass\_g  
 <fct> <fct> <dbl> <dbl> <int> <int>  
1 Chinstrap Dream 49.6 18.2 193 3775  
2 Chinstrap Dream 50.8 19 210 4100  
3 Chinstrap Dream 50.2 18.7 198 3775  
# ℹ 2 more variables: sex <fct>, year <int>

För att skapa olika datastrukturer

## 1.2 Variabel

var\_1 <- "Text/character string" # Någonting inuti "" är text  
var\_2 <- 25 # Nummer, för decimaltal 25.99  
var\_3 <- TRUE # Boolean, kan bara vara TRUE (T) eller FALSE (F)  
var\_4 <- NA # Not Available (NA), inte samma som "NA" (text string)!  
var\_5 <- NULL # Ingenting, tenderar att radera saker!

## 1.3 Vektor

vec\_1 <- c() # Skapar en tom vektor  
vec\_1 <- c(36, 56, 48) # Skapa en vektor med tre värden  
vec\_1[4] <- 25 # Sätta in ett värde i index plats 4  
vec\_1 # Printa innehållet av variablen (vektorn)

[1] 36 56 48 25

vec\_1[2] # Printa innehållet i index 2

vec\_1[c(1,3)] # Printa index 1 och 3

Vektorer kan innehålla namn och indexeras med namn

vec\_2 <- c(element\_1 = 99, element\_2 = 100, element\_3 = 101)  
vec\_2["element\_4"] <- 102 # Sätt in ett värde med namn  
vec\_2

element\_1 element\_2 element\_3 element\_4   
 99 100 101 102

vec\_2["element\_2"]

vec\_2[c("element\_1", "element\_3")]

## 1.4 Data frame

df <- data.frame( # Skapa en dataframe med kolumn\_namn = vektor  
 vec\_1, # Kolumn 1, vec\_1 från tidigare  
 letters = LETTERS[1:4] # Kolumn 2, lite bokstäver  
)  
df

vec\_1 letters  
1 36 A  
2 56 B  
3 48 C  
4 25 D

Man kan lägga till kolumner med $

df$col\_3 <- vec\_2 # Kolumn 3  
df

vec\_1 letters col\_3  
1 36 A 99  
2 56 B 100  
3 48 C 101  
4 25 D 102

Andra sätt att lägga till kolumner är med namn

df["col4"] <- letters[1:4]  
df

vec\_1 letters col\_3 col4  
1 36 A 99 a  
2 56 B 100 b  
3 48 C 101 c  
4 25 D 102 d

En kolumn kan användas med $ vilket ger en vektor (viktig distinktion vid vissa andra tillfällen).

df$letters # Ger en vektor, samma som med df[["letters]]

Eller med indexering/namn, N.B. enkla [ ] ger ett subset, alltså en mindre del av samma objekt typ

df[3] # Ger en data frame med enbart kolumn 3  
df["col4"] # Ger en data frame med kolumn "col4"

medan dubbla [[ ]] ger objektet inuti

df[[1]] # Ger det "mindre objektet inuti", alltså en vektor i detta fall

Enkla [ ] kan indexeras med rader och kolumner samtidigt

df[2] # Kolumn 2 (en data frame med en kolumn)  
df[3,2] # Rad 3, kolumn 2 (en vektor av längd ett)  
df[ ,2] # Alla rader, kolumn 2 (en vektor)  
df[3, ] # Rad 3, alla kolumner (en data frame med en rad)

På tal om subsetting, för att ta ett subset av en data frame kan man filtrera rader baserat på ett kriterium

penguins\_chin <- penguins[penguins$species == "Chinstrap", ]   
# Utan komma läses det som att du vill filtrera kolumner! '==' är det matematiska 'lika med' (eftersom '=' är samma som '<-' )  
head(penguins\_chin, 3)

# A tibble: 3 × 8  
 species island bill\_length\_mm bill\_depth\_mm flipper\_length\_mm body\_mass\_g  
 <fct> <fct> <dbl> <dbl> <int> <int>  
1 Chinstrap Dream 46.5 17.9 192 3500  
2 Chinstrap Dream 50 19.5 196 3900  
3 Chinstrap Dream 51.3 19.2 193 3650  
# ℹ 2 more variables: sex <fct>, year <int>

penguins\_heavy <- penguins[penguins$body\_mass\_g > 3500, ]  
head(penguins\_heavy, 3)

# A tibble: 3 × 8  
 species island bill\_length\_mm bill\_depth\_mm flipper\_length\_mm body\_mass\_g  
 <fct> <fct> <dbl> <dbl> <int> <int>  
1 Adelie Torgersen 39.1 18.7 181 3750  
2 Adelie Torgersen 39.5 17.4 186 3800  
3 <NA> <NA> NA NA NA NA  
# ℹ 2 more variables: sex <fct>, year <int>

Man kan även göra detta för att få en kolumn filtrerad/subset baserad på en annan kolumn.

penguins\_chin\_weight <- penguins$body\_mass\_g[penguins$species == "Chinstrap", ]

Error in penguins$body\_mass\_g[penguins$species == "Chinstrap", ] : incorrect number of dimensions

Det blir en error därför att nu försöker man ta både rader och kolumner (2 dimensioner) på en kolumn (kolumn/vektor har bara 1 dimension) därför att $ ger en vektor. Om vi istället provar utan extra kommatecken för att ange rader & kolumner samtidigt fungerar det (ger en vektor). Alternativt ange kolumnen med namn istället för med $ (ger en mindre data frame).

# Utan extra kommatecken  
penguins\_chin\_weight <- penguins$body\_mass\_g[penguins$species == "Chinstrap"]  
head(penguins\_chin\_weight)

[1] 3500 3900 3650 3525 3725 3950

# Ange kolumnen med namn  
penguins\_chin\_weight <- penguins[penguins$species == "Chinstrap", "body\_mass\_g"]  
head(penguins\_chin\_weight)

# A tibble: 6 × 1  
 body\_mass\_g  
 <int>  
1 3500  
2 3900  
3 3650  
4 3525  
5 3725  
6 3950

Vi kan även bli av med NA på ett kontrollerat sätt med den här metoden. Om man vill se till att vikt och näbblängd inte har NA kan man använda funtionen in.na() och operatorn !.

is.na(NA) # Kolla om NA är NA

[1] TRUE

!is.na(NA) # ! negerar ett TRUE/FALSE uttryck

[1] FALSE

vec\_NA <- c(NA, 50)  
is.na(vec\_NA) # Kolla vilka element som är NA

[1] TRUE FALSE

!is.na(vec\_NA)

[1] FALSE TRUE

# Alla rader där kolumnen body\_mass\_g inte är NA och alla kolumner  
penguins\_no\_NA <- penguins[!is.na(penguins$body\_mass\_g), ]  
# Alla rader där kolumnen bill\_length\_mm inte är NA och alla kolumner  
penguins\_no\_NA <- penguins\_no\_NA[!is.na(penguins\_no\_NA$bill\_length\_mm), ]  
  
# Tabell med antal rader & kolumner  
data.frame(  
 rows = c(penguins = nrow(penguins), no\_NA = nrow(penguins\_no\_NA)),  
 columns = c(penguins = ncol(penguins), no\_NA = ncol(penguins\_no\_NA))  
)

rows columns  
penguins 344 8  
no\_NA 342 8

## 1.5 Matris

Vi kan skapa matriser med rbind() (går även att använda cbind() men då placeras vektorerna vertikalt från det övre vänstra hörnet, istället för horisontellt). Med rbind() blir matrisen som den ser ut när man skriver den.

rbind( # en "character matrix"  
 c("a", "b"),  
 c("c", "d")  
)

[,1] [,2]  
[1,] "a" "b"   
[2,] "c" "d"

cbind( # skillnaden med cbind()  
 c("a", "b"),  
 c("c", "d")  
)

[,1] [,2]  
[1,] "a" "c"   
[2,] "b" "d"

M <- rbind( # Skapar en matris radvis med namngivna rader och kolumner  
 r1 = c(col1 = 0.90, col2 = 0, col3 = 0, col4 = 0, col5 = 0.45),  
 r2 = c( 0, 0.45, 0, 0, 0),  
 r3 = c( 0, 0.23, 0.45, 0, 0),  
 r4 = c( 0, 0, 0.23, 0.45, 0),  
 r5 = c( 0, 0, 0, 0.23, 0.20)  
)  
M

col1 col2 col3 col4 col5  
r1 0.9 0.00 0.00 0.00 0.45  
r2 0.0 0.45 0.00 0.00 0.00  
r3 0.0 0.23 0.45 0.00 0.00  
r4 0.0 0.00 0.23 0.45 0.00  
r5 0.0 0.00 0.00 0.23 0.20

För matris matte se [Matte](#matte) sektionen.

Matriser kan indexeras ungefär på samma sätt som data frames.

M[3, 2]  
M[3:5, 2:3]  
M[c(3,5), c(2,3)]  
M[c("r3","r5"), c("col2","col3")]

## 1.6 Lista (den riktiga listan)

I en vektor **måste** alla element vara av samma typ (numerisk/text osv.) och i en data frame **måste** alla kolumner (som är vektorer) vara lika långa.

Listor kan ha olika typer av objekt i sig och brukar användas för att bunta ihop olika objekt man vill hålla tillsammans. Till exempel är output från t.test() en lista.

my\_list <- list() # En tom lista  
my\_list$df\_1 <- df # Lägg till en data frame med $  
my\_list[[2]] <- var\_2 # Lägg till en variabel med index, N.B. [[]]  
my\_list[["plats\_3"]] <- vec\_2 # Lägg till en vektor med namn, N.B. [[]]  
my\_list

$df\_1  
 vec\_1 letters col\_3 col4  
1 36 A 99 a  
2 56 B 100 b  
3 48 C 101 c  
4 25 D 102 d  
  
[[2]]  
[1] 25  
  
$plats\_3  
element\_1 element\_2 element\_3 element\_4   
 99 100 101 102

Och man få ut objekten i en lista på samma sätt.

## 1.7 Lite andra funktioner

Ett objekts typ heter “klass” och kan kollas med funktionen class()

class(var\_1) # var\_1 är text  
class(var\_2) # var\_2 är numerisk  
class(var\_3) # var\_3 är logical/boolean (TRUE/FALSE)  
class(vec\_1) # vec\_1 är numerisk vektor  
class(df) # df är en data frame  
class(M) # M är en matris  
class(my\_list) # my\_list är en lista

För att se till att någonting skrivs i konsollfönstret används print()

print(var\_1) # Tar enbart ett objekt! Se paste()

[1] "Text/character string"

För att skriva ihop text m.m. används paste(), den tar dock enbart enskilda element (inte vektor/data frame/list

message <- paste(  
 "Hello", "world", var\_1, 55, vec\_2[3], # Alla objekt man vill skriva ihop  
 sep = " " # Separator att infoga mellan varje objekt, här ett mellanslag  
)  
print(message)

[1] "Hello world Text/character string 55 101"

Notera att print() enbart accepterar ett objekt, därav den vanliga kombinationen print(paste(arg1, arg2)).

print(paste("Nu","klarar","vi","tentan!", sep = "! "))

[1] "Nu! klarar! vi! tentan!"

Kolla längden på saker med

length(vec\_1) # Längden på en vektor  
length(penguins) # Längden på en data frame är antalet kolumner  
nrow(penguins) # För antalet rader i en data frame  
ncol(penguins) # Antalet columner på en data frame

Få en vektor med namnen på kolumner

colnames(penguins)

[1] "species" "island" "bill\_length\_mm"   
[4] "bill\_depth\_mm" "flipper\_length\_mm" "body\_mass\_g"   
[7] "sex" "year"

Få alla unika element i en vektor

unique(penguins$species)

[1] Adelie Gentoo Chinstrap  
Levels: Adelie Chinstrap Gentoo

Konvertera/tvinga (eng. coerce) objekt till en viss typ.

as.numeric() # Bra ifall en kolumn är text men ska vara siffror  
as.character()  
as.factor() # Kan vara viktig för ANOVA om grupperna är 1,2,3,4...  
  
as.data.frame() # Kan tvinga en matris till data frame  
as.matrix() # Kan tvinga en data frame till matris

## 1.8 Kontrollstrukturer

Kontrollstrukturer är några speciella funktioner som möjliggör mer kontroll över vilka rader med kod som ska köras.

### 1.8.1 Loop

För att köra några rader kod flera gånger används loopar. Detta är exempel på for-loop.

vector <- c("Nu", "klarar", "vi", "tentan!")  
for (variable in vector) {  
 print(variable)  
}

[1] "Nu"  
[1] "klarar"  
[1] "vi"  
[1] "tentan!"

for (i in 1:4) {  
 print(paste(i, ":", vector[i]))  
}

[1] "1 : Nu"  
[1] "2 : klarar"  
[1] "3 : vi"  
[1] "4 : tentan!"

n <- c()  
n[1] <- 55  
for (i in 1:5) {  
 n[i+1] <- i  
}  
n

[1] 55 1 2 3 4 5

### 1.8.2 If-else

För att köra vissa rader enbart om ett visst kriterium uppfylls används if-else kombination. Resultatet av att “köra koden” i parantesen ska bli TRUE för att köra den följande koden.

if ("Tenta!" == "Tenta!") { # Blir TRUE  
 print(TRUE)  
} else {  
 print(FALSE)  
}

[1] TRUE

if (5^2 < 100) { # Blir TRUE  
 print("Smaller")  
} else {  
 print("Bigger")  
}

[1] "Smaller"

if (50^2 < 100) { # Blir FALSE  
 print("Smaller")  
} else {  
 print("Bigger")  
}

[1] "Bigger"

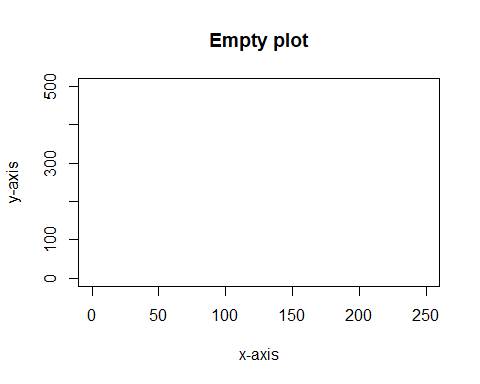
## 1.9 Plotting

Här följer plotfunktioner med många möjliga argument. Vissa argument visas flera gånger som alternativ.

För att kontrollera hur många plots att visa på samma gång, använd.

### 1.9.1 Tomt plotfönster

plot(  
 x = NULL,  
 y = NULL,  
   
 xlim = c(0, 250),  
 ylim = c(0, 500),  
   
 xlab = "x-axis",  
 ylab = "y-axis",  
 main = "Empty plot"  
)

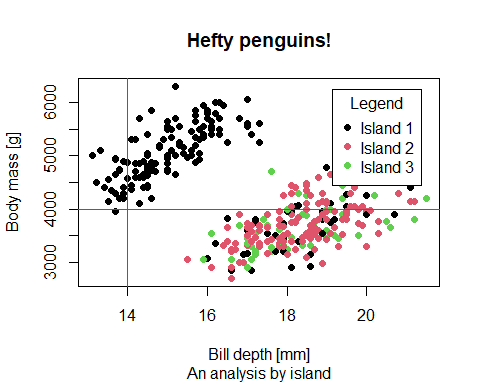


### 1.9.2 Scatterplot

Om x = kontinuerlig & y = kontinuerlig ger detta ett punktdiagram eller linjediagram.

De flesta argument fungerar likadant/liknande för andra typer av plots.

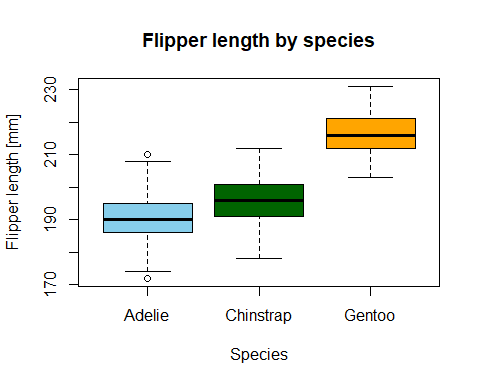
plot(  
 # Värden för axlarna  
 x = penguins$bill\_depth\_mm,  
 y = penguins$body\_mass\_g,  
   
 # Annotering  
 xlab = "Bill depth [mm]",  
 ylab = "Body mass [g]",  
 main = "Hefty penguins!",  
 sub = "An analysis by island",  
# ann = FALSE, # alt. ta bort alla vanliga annoteringar.  
   
# xaxt = "n", # x-axis text = 'n' (none)  
# yaxt = "n", # Tar bort axel markörerna  
   
 # Linjer & punkter  
 type = "p", # 'l' = line, 's' = stairs, 'p' = point,   
 # 'b' = both (line & pont), 'o' = both (overplotted),  
 # 'h' = histogram liknande, 'n' = none.  
 pch = 16, # 15 = ifylld kvadrat, 16 = ifylld prick,  
 # 17 = ifylld triangel, 18 = ifylld diamant.  
 cex = 1, # Storlek på punkt.  
  
# lwd = 3, # Storlek på linje.  
# lty = 1, # "blank", "solid", "dashed", "dotted",   
 # "dotdash", "longdash", twodash".  
  
# col = "blue", # Färg  
 col = penguins$island, # Färg baserat på faktor  
# col = alpha("blue", 0.30) # Färg (blå) & transparens (30%)  
)  
  
# Lägg till en linje från 'a' till 'b'.  
abline(  
# a = intercept,  
# b = slope,  
 h = 4000, # y-värde för horizontella linjer  
 v = 14, # x-värde för vertikala linjer  
# reg = lm\_objekt, # Regressionsmodell, se sektion 2.4.2.4 Predict  
 col = "purple"  
)  
  
# För att lägga till en legend  
legend(  
# x = 20, y = 6000, # x & y koordinater för legenden.  
   
 x = "topright", # Kan ges "top", "topleft", "topright",  
 # "bottom", "bottomleft", "bottomright".  
 inset = 0.05, # Avstånd till ytterkanterna som en % av plot storleken.   
   
 legend = c("Island 1", "Island 2", "Island 3"), # Namn i legenden.  
 pch = 16, # Sätt olika punkter med c(15, 16, 17). (Krävs för färger!)  
 col = 1:3, # Eftersom faktorn har tre grupper.   
 # Kan även vara c("blue","green","orange").  
   
# lty = c(1, 2, 3), # Sätt olika linje typer,  
# fill = 1:3, # Färg, men utan att ange form/linje.  
  
 title = "Legend", # Legend title  
 bty = "o", # 'o' = box, 'n' = no box, eller '7', 'L', 'U', 'C'.  
)



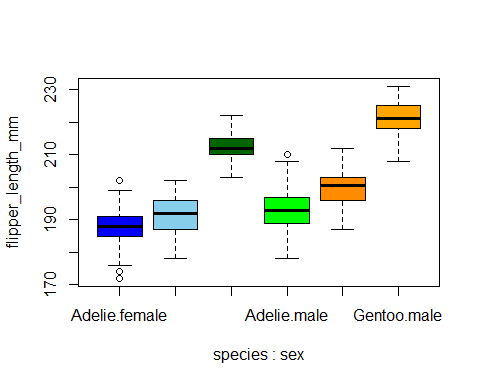
### 1.9.3 Boxplot

Om x = faktor & y = kontinuerlig ger detta en boxplot/låddiagram.

plot(  
 x = penguins$species,  
 y = penguins$flipper\_length\_mm,  
   
 xlab = "Species",  
 ylab = "Flipper length [mm]",  
 main = "Flipper length by species",  
   
 col = c("skyblue", "darkgreen", "orange")  
)



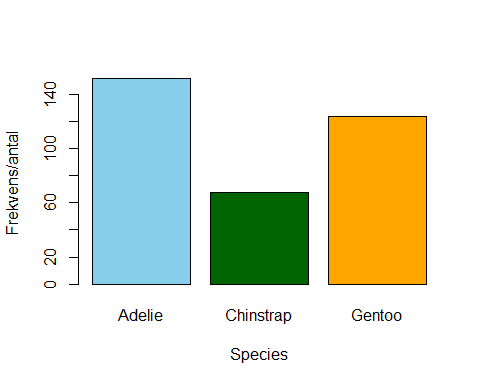
boxplot(  
 flipper\_length\_mm ~ species \* sex,  
 data = penguins,  
 col = c("blue","skyblue",  
 "darkgreen","green",  
 "darkorange","orange")  
)



### 1.9.4 Barplot

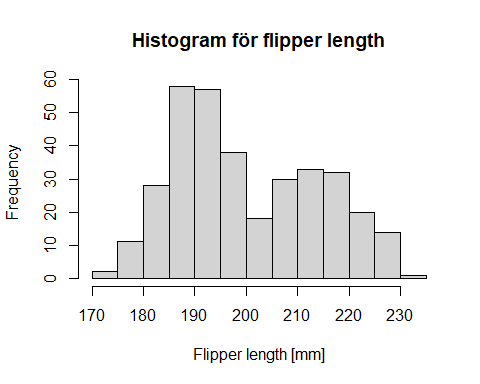
Om x = faktor & inget y ger detta en barplot/stapeldiagram.

plot(  
 x = penguins$species,  
   
 xlab = "Species",  
 ylab = "Frekvens/antal",  
   
 col = c("skyblue", "darkgreen", "orange")  
)



### 1.9.5 Histogram

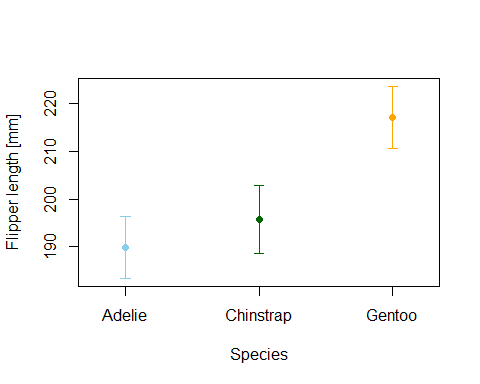
hist(  
 penguins$flipper\_length\_mm,  
 main = "Histogram för flipper length",  
 xlab = "Flipper length [mm]",  
   
# breaks = 99, # Hur många staplar  
# col = "blue", # Färg  
)



### 1.9.6 Error plot med plotCI()

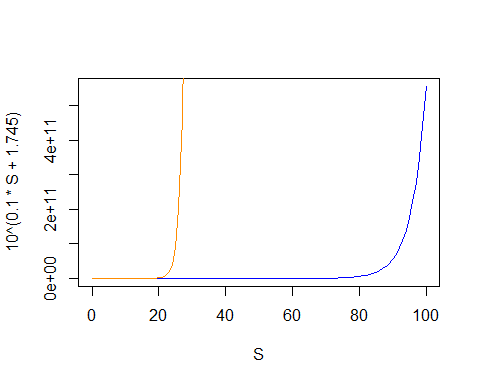
Används för att illustrera medelvärdet (en punkt) standardavvikelse eller standard error of the mean (SEM, se 2.1 Enkel statistik).

# Data för att plotta  
peng\_mean <- aggregate(  
 flipper\_length\_mm ~ species,  
 data = penguins,  
 FUN = mean  
)  
peng\_sd <- aggregate(  
 flipper\_length\_mm ~ species,  
 data = penguins,  
 FUN = sd  
)  
  
# Plot  
plotCI(  
 x = c(2,3,4), # x-koordinater att placera punkter på  
 y = peng\_mean$flipper\_length\_mm, # y-koordinater att placera punkter på  
   
 xlab = "Species",  
 ylab = "Flipper length [mm]",  
   
 # Upper interval, alltså övre gränsen för sträcken, är lika med  
 # medelvärdet + standardavvikelsen  
 ui = peng\_mean$flipper\_length\_mm + peng\_sd$flipper\_length\_mm,  
 # Lower interval, alltså den nedre gränsen, är lika med  
 # medelvärdet - standardavvikelsen  
 li = peng\_mean$flipper\_length\_mm - peng\_sd$flipper\_length\_mm,  
   
 xlim = c(1.75,4.25), # För att ge lite extra plats på sidorna  
   
 xaxt = "n", # Ta bort siffrorna på x-axeln  
   
 pch = 16,  
 col = c("skyblue", "darkgreen", "orange")  
)  
axis(  
 side = 1, # vilken sida/axel att lägga till (1 = botten, 2 = vänster, osv.)  
 at = c(2,3,4), # Vid vilka x-koordinater att placera text  
 labels = peng\_mean$species # Lägg till arterna på x-axeln  
)



### 1.9.7 Plot av funktion

# Definiera en funktion som en funktion med bara ett argument.  
curve\_fun <- function(x) {  
 10^(0.36\*x+1.945)  
}  
  
curve(  
 10^(0.10\*S+1.745), # Matematiska funktionen att rita  
 xname="S", # Vad 'x' heter om man ger den matematiska funktionen direkt  
 xlim=c(0,100), # Intervallet  
 col = "blue"  
)  
curve(  
 curve\_fun, # Namn på funktionen med funktionen  
 xlim = c(0,100),  
 col = "darkorange",  
 add = TRUE # Lägg till linjen i existerande graf  
)



# 2. Stats

### 2.0.1 Tolkning av p-värde

Drivna över bristningsgränsen av kursen har en student blivit galen och vandrar, med slö blick, vilset i stadsskogen. Studenten betraktar ett träd. Hur högt är trädet? Vilsen betraktar studenten ett annat träd och stapplar vidare. Vad är sannolikheten att genomsnittshöjden på de träd studenten slumpmässigt observerar skiljer sig från genomsnittshöjden på alla träd i stadsskogen? Kanske de skiljer sig från träden i fjällen?

* **H0** = Ingen skillnad mellan medelvärden
* **H1** = Skillnad mellan medelvärden

p-värdet är sannolikheten att en stackars vilsen student slumpmässigt skulle betrakta dessa träd och konstatera att deras höjd skiljer sig från resten av skogens (alltså acceptera **H1**), när det i verkligheten inte finns någon skillnad mellan observerade och skogens medelvärden (alltså om **H0** är sann).

* Om p-värde > alfa, alltså p > 0.05, säger vi att det inte finns en skillnad (acceptera **H0**).
* Om p-värde < alfa, alltså p < 0.05, säger vi att det finns en skillnad (acceptera **H1**).

### 2.0.2 Rapportera test

Rapportera alltid testets namn, statistiken (t.ex. t-/F-värdet), frihetsgrader och p-värde, samt en mening som ger slutsatsen/tolkningen av testet. Andra relevanta saker kan också inkluderas, såsom konfidens intervall.

Exempel: Förklarande mening ([testets namn], t([frihetsgrader]) = [värdet], p = [värdet]).

Det fanns en skillnad mellan Grupp 1 och den förväntade vikten (mu = 3000 g) enligt internet (One sample t-test, t(150) = 18.776, p < 2.2e-16).

Tips från coachen, gör detta snyggare än jag…

### 2.0.3 Formel notation

Många av funktionerna för de statistiska testen kan, eller måste, ges en formel. Formler har följande komponenter:

* *Respons*, en kontinuerlig variabel vi vill förutspå, värdet på y-axeln
* *Faktor*, en kategorisk variabel vilken delar in *Respons* i grupper (för regression är detta en kontinuerlig variabel som direkt orsakar *Respons*)
* ‘~’, skiljer *Respons* från *Faktor*, alltså ger Vänster- och Högerled (Respons ~ Faktor)
* ‘+’, adderar en ytterligare *Faktor* (Respons ~ Faktor\_1 + Faktor\_2)
* ‘\*’, adderar en ytterligare *Faktor* **och** interaktionen (Respons ~ Faktor\_1 \* Faktor\_2 )
* ‘:’, adderar enbart interaktionen (Respons ~ Faktor\_1:Faktor\_2)

### 2.0.4 Beslutsschema

|  |
| --- |
| Stats flowchart 1 |

|  |
| --- |
| Stats flowchart 2 |

|  |
| --- |
| Stats flowchart 3 |

|  |
| --- |
| Stats flowchart 4 |

## 2.1 Enkel statistik

Några enkla/grundläggande statistik funktioner.

mean(penguins$bill\_length\_mm, na.rm = TRUE) # Medelvärde

[1] 43.92193

median(penguins$bill\_length\_mm, na.rm = TRUE) # Median

[1] 44.45

sum(penguins$bill\_length\_mm, na.rm = TRUE) # Summa av vektor

[1] 15021.3

sd(penguins$bill\_length\_mm, na.rm = TRUE) # Standardavvikelse

[1] 5.459584

# Funktioner som ger vektorer med svar. Fungerar på både matriser & data frames!  
# N.B. stor bokstav i mitten! Finns andra funktioner med liknande namn.  
rowSums(M) # ger en (kolumn) vektor med summan för varje rad.

r1 r2 r3 r4 r5   
1.35 0.45 0.68 0.68 0.43

colSums(M) # ger en (rad) vektor med summan för varje kolumn.

col1 col2 col3 col4 col5   
0.90 0.68 0.68 0.68 0.65

rowMeans(M) # Samma, men för medelvärde.

r1 r2 r3 r4 r5   
0.270 0.090 0.136 0.136 0.086

colMeans(M) # Samma, men för medelvärde.

col1 col2 col3 col4 col5   
0.180 0.136 0.136 0.136 0.130

# Aggregate är lite speciell. Den grupperar rader efter en faktor och applicerar en annan funktion på varje grupp.  
aggregate(  
 bill\_length\_mm ~ island,  
 data = penguins,  
 FUN = mean, # Använd 'mean' eller 'median'  
)

island bill\_length\_mm  
1 Biscoe 45.25749  
2 Dream 44.16774  
3 Torgersen 38.95098

Standard error of the mean (SEM) finns det ingen egen funktion för, men vi kan skapa en. Formeln är , där Sd = standardavvikelse och n = antal observationer. SEM beskriver hur exakt/pålitligt medelvärdet är om det används som det sanna medelvärdet för populationen.

SEM <- function(x, na.rm = FALSE) {  
 # Om argumentet na.rm = TRUE, ta bort NA från x  
 if (na.rm) {  
 x <- na.omit(x)  
 }  
   
 # Beräkna SEM  
 SEM <- sd(x)/sqrt(length(x))  
 return(SEM)  
}  
  
SEM(penguins$bill\_length\_mm, na.rm = TRUE)

[1] 0.2952205

## 2.2 Transformation

Om data inte ser nice ut, prova att transformera data (för paired/beroende transformera differensen). För denna kurs prova logaritmering.

log() # Basen e  
log10() # Basen 10

## 2.3 Sannolikhet

### 2.3.1 Slumpade tal

Slumpa ett tal från normalfördelning. Ungefär 96 % av alla observationer i en normalfördelad population ligger inom två standardavvikelser.

rnorm(  
 n = 5, # Antal tal  
 mean = 0, # Väntevärde för fördelningen  
 sd = 1 # Standard avvikelse  
)

[1] 0.437518724 -1.670881811 -2.355736497 0.009593165 0.342310286

Slumpade tal från binomialfördelning. Ger n tal där varje tal är hur många gynnsamma utfall det blev, baserat på totalt antal försök och sannolikheten.

rbinom(  
 n = 5, # Antal tal  
 size = 10, # Totalt antal försök  
 prob = 0.5 # Sannolikhet för gynnsamt utfall  
)

[1] 6 7 4 7 5

Slumpade tal från en uniform fördelning (samma sannolikhet för för varje tal).

runif(  
 n = 5, # Antal tal  
 min = 0, # Nedre intervall  
 max = 10 # Övre intervall  
)

[1] 5.389269 8.484125 7.552771 7.705024 2.542631

### 2.3.2 Binomial fördelning

Min binomiala studie:

* 20 försök
* varje försök är Ja/Nej
* 37 % sannolikhet för Ja för varje oberoende försök

För utfall med binomial fördelning används dbinom() för att beräkna sannolikheten för exakt x antal gynsamma utfall.

f5 <- dbinom(  
 x = 5, # Exakt antal 'Ja'  
 size = 20, # Totalt antal försök  
 prob = 0.37 # Sannolikheten för försöket  
)  
print(paste("Sannolikhet att exakt fem av 20 försök blir 'Ja':", f5))

[1] "Sannolikhet att exakt fem av 20 försök blir 'Ja': 0.10508981363014"

Dessa sannolikheter kan adderas.

f6 <- dbinom(  
 x = 6, # Exakt 6st Ja  
 size = 20,  
 prob = 0.37  
)  
print(paste("p att exakt 5-6st av 20 försök blir 'Ja':", (f5+f6)))

[1] "p att exakt 5-6st av 20 försök blir 'Ja': 0.259388349515663"

eller 1 - pbinom() för att beräkna sannolikheten för alla utfall **över** det man anger.

f15\_20 <- (1 - pbinom(  
 q = 14, # utfall över det angivna  
 size = 20,  
 prob = 0.37  
))  
print(paste("p att få 15 eller fler 'Ja':",f15\_20))

[1] "p att få 15 eller fler 'Ja': 0.000621595787321794"

Enbart pbinom() ger den motsatta sannolikheten (alltså 0-14 Ja) vilket är kompliment till 15-20 Ja. Använd lower.tail = FALSE för att få 15-20 Ja istället.

f15\_20\_igen <- pbinom(  
 q = 14,  
 size = 20,  
 prob = 0.37,  
 lower.tail = FALSE # Istället för 1 - pbinom()  
)  
print(paste("p att få 15 eller fler 'Ja':",f15\_20\_igen))

[1] "p att få 15 eller fler 'Ja': 0.000621595787321803"

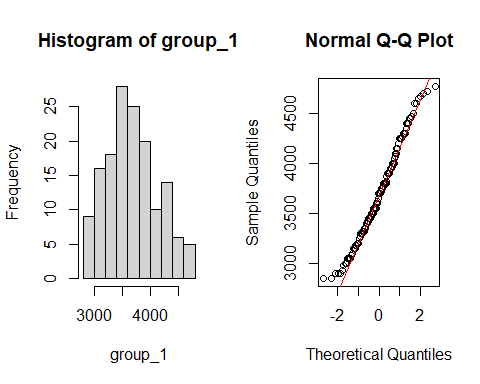
## 2.4 Skillnad mellan grupper

### 2.4.1 T-test

#### 2.4.1.1 Krav/antaganden

* Data på intervall eller kvotskala (kontinuerlig)
* Normalfördelad (parametrisk) inom varje grupp
* Lika varians (homogen varians, ej för Welch t-test)
* Inga outliers

group\_1 <- penguins$body\_mass\_g[penguins$species == "Adelie"]  
  
par(mfrow = c(1,2)) # Ändrar hur många "rutor/celler" för   
# plots det finns enligt rad x column. Bra vana att alltid   
# ändra tillbaka till standard c(1,1) efteråt.   
  
hist(group\_1)  
  
qqnorm(group\_1)  
qqline(group\_1, col = "red")



par(mfrow = c(1,1)) # Ändra tillbaka inställningarna.

#### 2.4.1.2 Ensidigt

För att testa om en grupp med kontinuerlig data skiljer sig från ett känt eller teoretiskt väntevärde.

grupp\_1 <- penguins$body\_mass\_g[penguins$species == "Adelie"]  
  
t.test(  
 x = grupp\_1, # En vektor  
 mu = 3000 # Väntevärdet vi hittade på internet  
)

One Sample t-test  
  
data: grupp\_1  
t = 18.776, df = 150, p-value < 2.2e-16  
alternative hypothesis: true mean is not equal to 3000  
95 percent confidence interval:  
 3626.926 3774.398  
sample estimates:  
mean of x   
 3700.662

##### Resultat:

Det fanns en skillnad mellan Grupp 1 och den förväntade vikten enligt internet (One sample t-test, t(150) = 18.776, p < 2.2e-16).

#### 2.4.1.3 Tvåsidigt

För att testa om det finns en skillnad mellan två grupper av kontinuerlig data.

grupp\_1 <- penguins$body\_mass\_g[penguins$species == "Adelie"]  
grupp\_2 <- penguins$body\_mass\_g[penguins$species == "Gentoo"]  
  
t.test(  
 x = grupp\_1, # En vektor  
 y = grupp\_2, # En annan vektor  
)

Welch Two Sample t-test  
  
data: grupp\_1 and grupp\_2  
t = -23.386, df = 249.64, p-value < 2.2e-16  
alternative hypothesis: true difference in means is not equal to 0  
95 percent confidence interval:  
 -1491.183 -1259.525  
sample estimates:  
mean of x mean of y   
 3700.662 5076.016

Alternativt sätt att skriva koden är med formel.

peng\_groups <- penguins[ which(penguins$species == "Adelie" | penguins$species == "Gentoo"), ]  
  
t.test( # Respons = mätdata & Faktor = två olika arter (alltså två grupper)  
 body\_mass\_g ~ species,  
 data = peng\_groups  
)

Welch Two Sample t-test  
  
data: body\_mass\_g by species  
t = -23.386, df = 249.64, p-value < 2.2e-16  
alternative hypothesis: true difference in means between group Adelie and group Gentoo is not equal to 0  
95 percent confidence interval:  
 -1491.183 -1259.525  
sample estimates:  
mean in group Adelie mean in group Gentoo   
 3700.662 5076.016

##### Resultat:

Det fanns en skillnad mellan Grupp 1 och Grupp 2 (Welch two sample t-test, t(249.64) = -23.386, p < 2.2e-16).

#### 2.4.1.4 Paired/beroende

First rule of Crayfish Fight Club is you do not talk about Crayfish Fight Club.

Vi studerar kräftor och vill undersöka vilken effekt storleken på klorna har på vilken kräfta som vinner strider om resurser. För att göra detta snorklar vi någonstans tropiskt och letar efter kräftor som slåss. Efter några månader “hårt arbete” har vi samlat ihop följande data. Varje rad är storleken på klorna i cm med vinnare och förlorare i respektive kolumn.

crayfish\_fight\_club <- data.frame(  
 winner = c(86, 84, 75, 93, 102, 87, 88, 91, 87, 74),  
 loser = c( 70, 79, 68, 85, 90, 89, 91, 82, 80, 69)  
)  
head(crayfish\_fight\_club)

winner loser  
1 86 70  
2 84 79  
3 75 68  
4 93 85  
5 102 90  
6 87 89

Alltså består varje rad av mätningar på två individer där grupperingen är **beroende** på vem de jämförs med, och därför gör vi ett parat t-test. (Det känns inte rimligt att jämföra vinnaren på rad tre (75 cm) med förloraren på rad fyra (85 cm) eftersom de inte stred med varandra, vilket är vad vi hade gjort om vi behandlat data som oberoende, alltså paired = FALSE.) Notera att för parad/beroende data måste **differensen** vara normalfördelad.

t.test(  
 crayfish\_fight\_club$winner,  
 crayfish\_fight\_club$loser,  
 paired = TRUE  
)

Paired t-test  
  
data: crayfish\_fight\_club$winner and crayfish\_fight\_club$loser  
t = 3.5266, df = 9, p-value = 0.006448  
alternative hypothesis: true mean difference is not equal to 0  
95 percent confidence interval:  
 2.29474 10.50526  
sample estimates:  
mean difference   
 6.4

##### Resultat:

Paired t-test, t(9) = 3.5266, p = 0.006448, Signifikant.

#### 2.4.1.5 Wilcox-test (icke parametriskt)

För när data inte uppfyller krav för t-test eller vi vill titta på skillnader i median.

wilcox.test( # Motsvarar vanligt t-test för två grupper  
 grupp\_1,  
 grupp\_2  
)  
  
wilcox.test( # Med formel  
 body\_mass\_g ~ species,  
 data = peng\_groups  
)

wilcox.test( # Motsvarar parat t-test för beroende data  
 crayfish\_fight\_club$winner,  
 crayfish\_fight\_club$loser,  
 paired = TRUE  
)

Warning in wilcox.test.default(crayfish\_fight\_club$winner,  
crayfish\_fight\_club$loser, : cannot compute exact p-value with ties

Wilcoxon signed rank test with continuity correction  
  
data: crayfish\_fight\_club$winner and crayfish\_fight\_club$loser  
V = 52, p-value = 0.01431  
alternative hypothesis: true location shift is not equal to 0

##### Resultat:

Ingen skillnad.

För denna kurs, ignorera Warning: cannot compute exact p-value with ties. Det betyder att det finns två av samma mätvärde vilket inte går att rangordna. (Om data är c(25, 26, 26, 27) är det (1,2,3,4) eller (1,3,2,4)?)

#### 2.4.1.6 Bootstrap (icke parametriskt)

För när data inte uppfyller krav för t-test men vi behöver ett konfidensintervall. Resurskrävande, alltså om du har mycket data tar det längre tid att köra funktionen. Ekvivalent med Wilcoxon men ger konfidensintervall.

boot.t.test( # Med två vektorer  
 grupp\_1,  
 grupp\_2  
)

Bootstrap Welch Two Sample t-test  
  
data: grupp\_1 and grupp\_2  
bootstrap p-value < 2.2e-16   
bootstrap difference of means (SE) = -1374.176 (58.50902)   
95 percent bootstrap percentile confidence interval:  
 -1489.464 -1259.411  
  
Results without bootstrap:  
t = -23.386, df = 249.64, p-value < 2.2e-16  
alternative hypothesis: true difference in means is not equal to 0  
95 percent confidence interval:  
 -1491.183 -1259.525  
sample estimates:  
mean of x mean of y   
 3700.662 5076.016

boot.t.test( # Med formel  
 body\_mass\_g ~ species,  
 data = peng\_groups  
)  
boot.t.test( # Paired/beroende data  
 crayfish\_fight\_club$winner,  
 crayfish\_fight\_club$loser,  
 paired = TRUE  
)

### 2.4.2 ANOVA

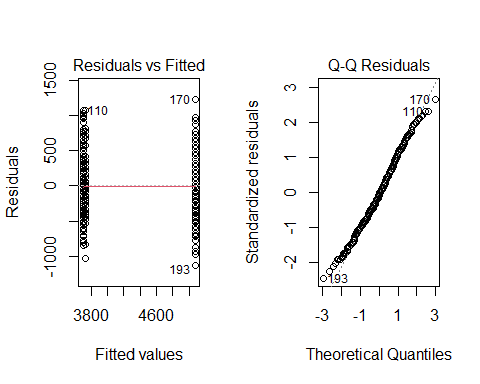
ANOVA undersöker skillnad mellan många grupper och faktorernas effekt på respons variabeln (liknande regression där kontinuerlig variabel kausalt påverkar respons).

#### 2.4.2.1 Krav/antaganden

Anova är nästan samma som t-test,

* Data på intervall eller kvotskala (kontinuerlig)
* Normalfördelad (parametrisk) inom varje grupp (**Bootstrap**)
* Lika varians (homogen varians, ej för Welch t-test)
* Inga outliers
* **Oberoende data** (vi kör inte parat/beroende i denna kurs)
* **Förutsätter lika varians mellan grupper**
* **Kan hantera fler än två grupper samt fler än en faktor**

# Skapa modellen, sedan diagnostik.  
diagnostic\_model <- aov(  
 body\_mass\_g ~ species,  
 data = penguins  
)  
  
par(mfrow = c(1,2))  
  
plot(diagnostic\_model, which = 1) # Lika varians om   
# spridningen/"längden" på varje stapel med punkter är lika.  
  
plot(diagnostic\_model, which = 2) # QQ-plot för normalfördelning.



par(mfrow = c(1,1))

#### 2.4.2.2 Envägs ANOVA

Envägs ANOVA innebär att vi använder en faktor (med flera grupper). Hur hittar man enkelt faktorer och grupper i ett dataset? En faktor är en kolumn med grupper. I penguins data frame är penguins$species en kolumn med grupper, alltså en faktor.

unique(penguins$species) # Unika element i kolumnen,

[1] Adelie Gentoo Chinstrap  
Levels: Adelie Chinstrap Gentoo

# alltså, faktorn 'species' har grupperna 'Adelie', 'Gentoo', 'Chinstrap'

ANOVA med funktionen aov(), summary() och TukeyHSD().

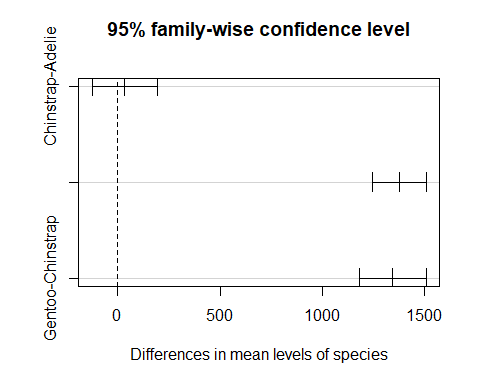
anova\_aov <- aov( # Skapa modell  
 body\_mass\_g ~ species,  
 data = penguins  
)  
summary(anova\_aov) # Kolla om det fanns en skillnad mellan någon av grupperna

Df Sum Sq Mean Sq F value Pr(>F)   
species 2 146864214 73432107 343.6 <2e-16 \*\*\*  
Residuals 339 72443483 213698   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
2 observations deleted due to missingness

TukeyHSD(anova\_aov) # Kolla mellan vilka grupper det fanns en skillnad

Tukey multiple comparisons of means  
 95% family-wise confidence level  
  
Fit: aov(formula = body\_mass\_g ~ species, data = penguins)  
  
$species  
 diff lwr upr p adj  
Chinstrap-Adelie 32.42598 -126.5002 191.3522 0.8806666  
Gentoo-Adelie 1375.35401 1243.1786 1507.5294 0.0000000  
Gentoo-Chinstrap 1342.92802 1178.4810 1507.3750 0.0000000

plot(TukeyHSD(anova\_aov)) # Plot



ANOVA med funktionen lm(), anova() och summary(). Detta ger problem med att göra post-hoc test (TukeyHSD()) för att hitta mellan vilka grupper skillnaden finns. Fördelen skulle vara att få ett intercept och möjligen bygga en matematisk modell.

anova\_lm <- lm( # Skapa modell  
 body\_mass\_g ~ species,  
 data = penguins  
)  
anova(anova\_lm) # Kolla om det fanns en skillnad mellan någon av grupperna

Analysis of Variance Table  
  
Response: body\_mass\_g  
 Df Sum Sq Mean Sq F value Pr(>F)   
species 2 146864214 73432107 343.63 < 2.2e-16 \*\*\*  
Residuals 339 72443483 213698   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(anova\_lm) # Kolla mellan vilka grupper det fanns en skillnad

Call:  
lm(formula = body\_mass\_g ~ species, data = penguins)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-1126.02 -333.09 -33.09 316.91 1223.98   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) 3700.66 37.62 98.37 <2e-16 \*\*\*  
speciesChinstrap 32.43 67.51 0.48 0.631   
speciesGentoo 1375.35 56.15 24.50 <2e-16 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 462.3 on 339 degrees of freedom  
 (2 observations deleted due to missingness)  
Multiple R-squared: 0.6697, Adjusted R-squared: 0.6677   
F-statistic: 343.6 on 2 and 339 DF, p-value: < 2.2e-16

#### 2.4.2.3 Tvåvägs ANOVA

Samma som envägs ANOVA men med två faktorer. Se [Formel notation](#formel-notation).

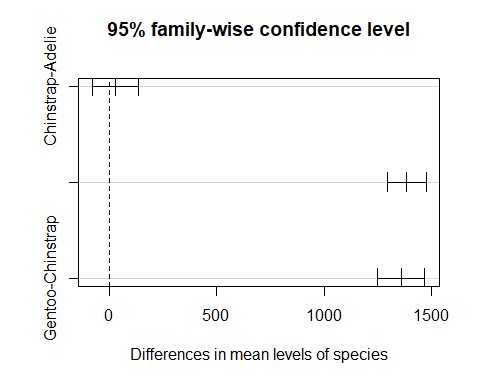
anova2\_aov <- aov( # Skapa modell  
 body\_mass\_g ~ species \* sex,  
 data = penguins  
)  
summary(anova2\_aov) # Kolla om det fanns en skillnad mellan någon av grupperna

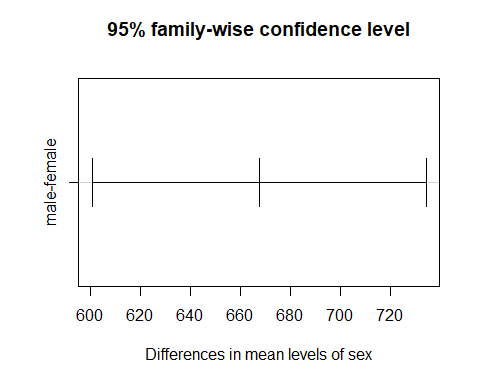
Df Sum Sq Mean Sq F value Pr(>F)   
species 2 145190219 72595110 758.358 < 2e-16 \*\*\*  
sex 1 37090262 37090262 387.460 < 2e-16 \*\*\*  
species:sex 2 1676557 838278 8.757 0.000197 \*\*\*  
Residuals 327 31302628 95727   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
11 observations deleted due to missingness

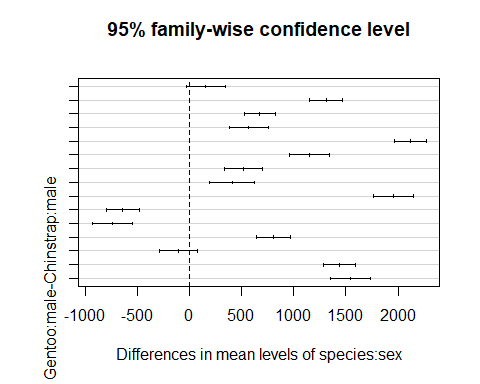
TukeyHSD(anova2\_aov) # Kolla mellan vilka grupper det fanns en skillnad

Tukey multiple comparisons of means  
 95% family-wise confidence level  
  
Fit: aov(formula = body\_mass\_g ~ species \* sex, data = penguins)  
  
$species  
 diff lwr upr p adj  
Chinstrap-Adelie 26.92385 -80.0258 133.8735 0.8241288  
Gentoo-Adelie 1386.27259 1296.3070 1476.2382 0.0000000  
Gentoo-Chinstrap 1359.34874 1248.6108 1470.0866 0.0000000  
  
$sex  
 diff lwr upr p adj  
male-female 667.4577 600.7462 734.1692 0  
  
$`species:sex`  
 diff lwr upr p adj  
Chinstrap:female-Adelie:female 158.3703 -25.7874 342.5279 0.1376213  
Gentoo:female-Adelie:female 1310.9058 1154.8934 1466.9181 0.0000000  
Adelie:male-Adelie:female 674.6575 527.8486 821.4664 0.0000000  
Chinstrap:male-Adelie:female 570.1350 385.9773 754.2926 0.0000000  
Gentoo:male-Adelie:female 2116.0004 1962.1408 2269.8601 0.0000000  
Gentoo:female-Chinstrap:female 1152.5355 960.9603 1344.1107 0.0000000  
Adelie:male-Chinstrap:female 516.2873 332.1296 700.4449 0.0000000  
Chinstrap:male-Chinstrap:female 411.7647 196.6479 626.8815 0.0000012  
Gentoo:male-Chinstrap:female 1957.6302 1767.8040 2147.4564 0.0000000  
Adelie:male-Gentoo:female -636.2482 -792.2606 -480.2359 0.0000000  
Chinstrap:male-Gentoo:female -740.7708 -932.3460 -549.1956 0.0000000  
Gentoo:male-Gentoo:female 805.0947 642.4300 967.7594 0.0000000  
Chinstrap:male-Adelie:male -104.5226 -288.6802 79.6351 0.5812048  
Gentoo:male-Adelie:male 1441.3429 1287.4832 1595.2026 0.0000000  
Gentoo:male-Chinstrap:male 1545.8655 1356.0392 1735.6917 0.0000000

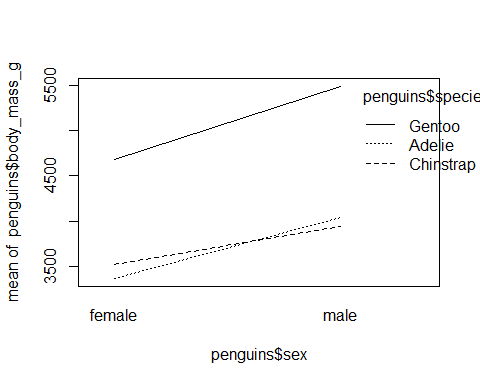
plot(TukeyHSD(anova2\_aov)) # Plot







# Interaktions plot  
interaction.plot(  
 x.factor = penguins$sex, # x-axeln  
 trace.factor = penguins$species, # linjerna  
 response = penguins$body\_mass\_g, # y-axeln  
)



#### 2.4.2.4 Kruskal-Wallis (icke parametrisk)

Icke parametrisk variant av **envägs** ANOVA

kruskal.test(  
 body\_mass\_g ~ species,  
 data = penguins  
)

Kruskal-Wallis rank sum test  
  
data: body\_mass\_g by species  
Kruskal-Wallis chi-squared = 217.6, df = 2, p-value < 2.2e-16

**Detta är utanför kursen!** För att hitta mellan vilka grupper skillnaden finns (post-hoc). Kan inte hantera NA och har krashat R…

pairwise.wilcox.test(  
 x = penguins$body\_mass\_g, # Vektor med mätdata  
 g = penguins$species, #   
 p.adjust.method = "holm"  
)

Pairwise comparisons using Wilcoxon rank sum test with continuity correction   
  
data: penguins$body\_mass\_g and penguins$species   
  
 Adelie Chinstrap  
Chinstrap 0.49 -   
Gentoo <2e-16 <2e-16   
  
P value adjustment method: holm

#### 2.4.2.5 Bootstrap (icke parametriskt)

Om någon av grupperna inte är normalfördelade kan man använda bootstrap. Alla grupper ska fortfarande ha samma varians.

ANOVA.boot() kan inte hantera NA, så se till att filtrera data frame.

# Subset to remove NA  
peng\_boot <- penguins[!is.na(penguins$body\_mass\_g), ]  
peng\_boot <- peng\_boot[!is.na(peng\_boot$species), ]  
peng\_boot <- peng\_boot[!is.na(peng\_boot$sex), ]  
  
anova\_boot <- ANOVA.boot(  
 body\_mass\_g ~ species \* sex,  
 data = peng\_boot  
)  
anova\_boot[c("terms","df","p-values")]

$terms  
[1] "species " "sex " "species:sex" "Residuals "  
  
$df  
[1] 2 1 2 327  
  
$`p-values`  
[1] 0 0 0

Detta är inte särskillt nice output format så vi undviker helst att använda ANOVA.boot().

## 2.5 Kontinuerliga samband

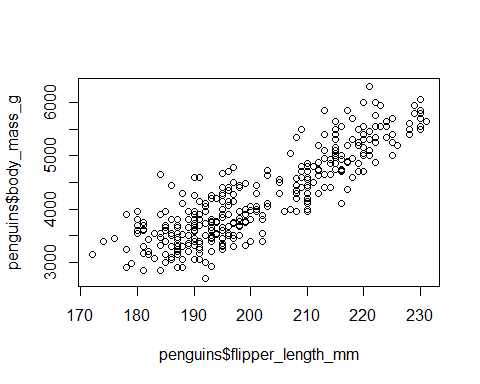
### 2.5.1 Samvariation/korrelation

När två kontinuerliga variabler ökar/minskar tillsammans (positiv korrelation) eller i motsatt riktning (negativ korrelation), möjligtvis till följd av en gemensam tredje (dold) variabel.

#### 2.5.1.1 Krav/antaganden

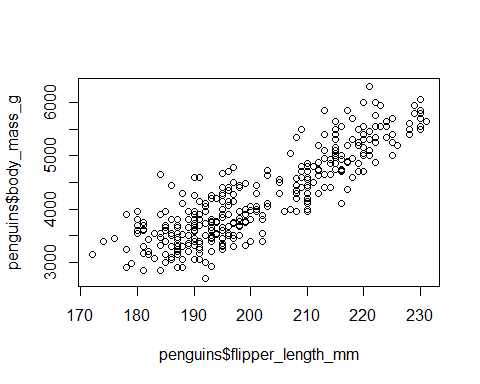
* Två kontinuerliga variabler/mätdata
* Linjärt samband (kan göras linjär genom transformation om nödvändigt)

plot( # Ögonmåtta att sambandet inte är uppenbarligen icke-linjärt  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g  
)



#### 2.5.1.2 Pearson korrelation

plot( # Kolla alltid data  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g  
)



cor.test(  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g  
)

Pearson's product-moment correlation  
  
data: penguins$flipper\_length\_mm and penguins$body\_mass\_g  
t = 32.722, df = 340, p-value < 2.2e-16  
alternative hypothesis: true correlation is not equal to 0  
95 percent confidence interval:  
 0.843041 0.894599  
sample estimates:  
 cor   
0.8712018

p-värdet tolkas som vanligt, cor är r2-värdet som beskriver styrka och riktning av korrelationen.

#### 2.5.1.3 Spearman (icke parametrisk)

cor.test(  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g,  
 method = "spearman" # Specificera spearman  
)

Warning in cor.test.default(penguins$flipper\_length\_mm, penguins$body\_mass\_g, :  
Cannot compute exact p-value with ties

Spearman's rank correlation rho  
  
data: penguins$flipper\_length\_mm and penguins$body\_mass\_g  
S = 1066875, p-value < 2.2e-16  
alternative hypothesis: true rho is not equal to 0  
sample estimates:  
 rho   
0.8399741

För denna kurs, ignorera Warning: cannot compute exact p-value with ties. Det betyder att det finns två av samma mätvärde vilket inte går att rangordna. (Om data är c(25, 26, 26, 27) är det (1,2,3,4) eller (1,3,2,4)?)

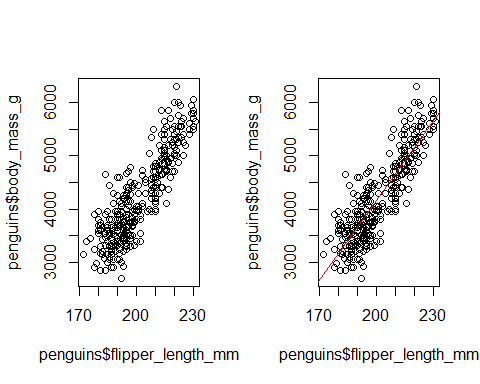
### 2.5.2 Regression

Regression används då en kontinuerlig variabel kausalt orsakar en annan kontinuerlig variabel.

#### 2.5.2.1 Krav/antaganden

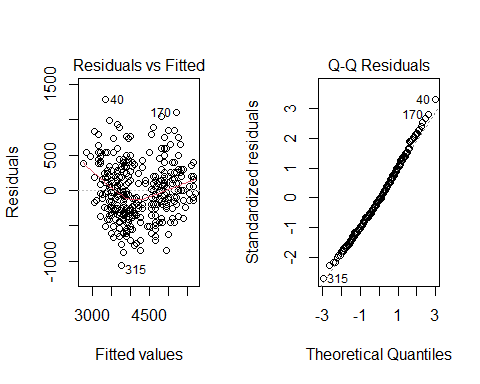
* **Linjärt samband**
* Normalfördelad (parametrisk)
* Lika varians

par(mfrow = c(1,2))  
  
plot( # Ögonmåtta om det finns en linjär trend.  
 x = penguins$flipper\_length\_mm,  
 y = penguins$body\_mass\_g  
)  
  
# Skapa modellen, sedan diagnostik.  
diagnostic\_model <- aov(  
 body\_mass\_g ~ flipper\_length\_mm,  
 data = penguins  
)  
  
plot( # Plot med linjen från modellen som hjälp  
 x = penguins$flipper\_length\_mm,  
 y = penguins$body\_mass\_g  
)  
abline(diagnostic\_model,  
 col = "red")



par(mfrow = c(1,1))

par(mfrow = c(1,2))  
  
plot(diagnostic\_model, which = 1) # Lika varians om   
# spridningen av punkter är uniform & röda linjen är platt.  
  
plot(diagnostic\_model, which = 2) # QQ-plot för normalfördelning.



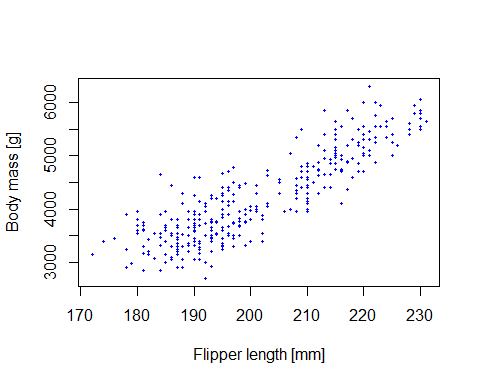
par(mfrow = c(1,1))

#### 2.5.2.2 Linjär regression

Skapas med en formel liknande för ANOVA.

Med det högst biologiska antagandet att längre fenor orsakar högre kroppsvikt (genom att pingvinerna simmar fortare och därmed kan äta mer, säger vi) enligt en linjär trend, väljer vi att göra linjär regression.

plot( # Scatterplot för att kolla sambandet  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g,  
 pch = 16,  
 cex = 0.4,  
 col = "blue",  
 xlab = "Flipper length [mm]",  
 ylab = "Body mass [g]"  
)



mass\_by\_flipper <- lm( # Skapa model  
 body\_mass\_g ~ flipper\_length\_mm,  
 data = penguins  
)  
summary(mass\_by\_flipper)

Call:  
lm(formula = body\_mass\_g ~ flipper\_length\_mm, data = penguins)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-1058.80 -259.27 -26.88 247.33 1288.69   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) -5780.831 305.815 -18.90 <2e-16 \*\*\*  
flipper\_length\_mm 49.686 1.518 32.72 <2e-16 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 394.3 on 340 degrees of freedom  
 (2 observations deleted due to missingness)  
Multiple R-squared: 0.759, Adjusted R-squared: 0.7583   
F-statistic: 1071 on 1 and 340 DF, p-value: < 2.2e-16

#### 2.5.2.3 Bootstrap

Alternativt med lm() och sedan boot\_summary().

boot.anova\_lm <- lm( # Skapa model  
 body\_mass\_g ~ species \* sex,   
 data = peng\_boot )   
boot\_summary(boot.anova\_lm)

Estimate Lower.bound Upper.bound p.value  
(Intercept) 3368.8356 3299.70679 3441.78794 0.000  
speciesChinstrap 158.3703 28.68293 284.80896 0.018  
speciesGentoo 1310.9058 1204.76994 1418.14926 0.000  
sexmale 674.6575 575.42004 769.17767 0.000  
speciesChinstrap:sexmale -262.8928 -446.01658 -79.50548 0.007  
speciesGentoo:sexmale 130.4372 -12.84620 283.67054 0.070

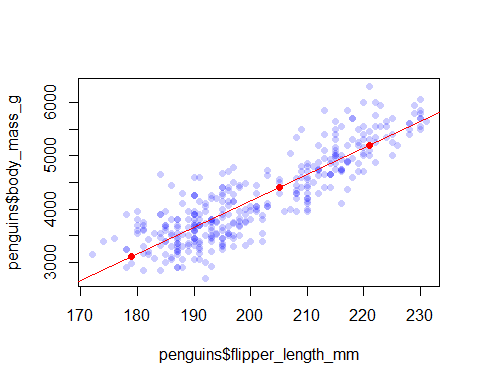
#### 2.5.2.4 Predict

Om längden på fenorna orsakar kroppsmassa kan vi använda modellen för att förutse vikten på en pingvin baserat på fenornas längd.

Argumentet newdata = data.frame() måste innehålla samma kolumn namn som x-axeln och en vektor med de värden på x-axeln man vill förutspå y-värdet för.

# Data frame med nya värden att förutspå.   
# Måste ha en kolumn med samma namn som x-axeln i modellen!  
measured\_flipper\_length <- data.frame(  
 flipper\_length\_mm = c(205, 221, 179)  
)  
  
# De värden vi gissar med hjälp av modellen  
estimated\_weight <- predict(  
 object = mass\_by\_flipper, # Modellen  
 newdata = measured\_flipper\_length) # nya x-värden

plot(  
 penguins$flipper\_length\_mm,  
 penguins$body\_mass\_g,  
   
 pch = 16,  
 col = alpha("blue", 0.2)  
)  
points(  
 x = measured\_flipper\_length$flipper\_length\_mm, # newdata  
 y = estimated\_weight, # Gissade värden  
   
 pch = 16,  
 col = "red"  
)  
abline( # Lägg till linjen från modellen.  
 reg = mass\_by\_flipper,  
 col = "red",  
 lwd = 1  
)



## 2.6 Fördelning av frekvens data

### 2.6.1 Chi-square (X2-test)

Det enda testet på kursen som hanterar frekvenser/räknedata. Principiellt behövs tre saker, en faktor med grupper som blir kolumner, en faktor med grupper som blir rader, individer/föremål att räkna.

Till ära av Tobbes doktorsavhandling används här data om Gemsbock och Red hartbeest och deras observerade fördelning på fyra områden.

bockar <- rbind(  
 Gemsbock = c(FR = 5, CGA = 7, WMA = 12, NP = 67),  
 Red\_hartbeest = c( 6, 8, 54, 13)  
)  
  
bockar

FR CGA WMA NP  
Gemsbock 5 7 12 67  
Red\_hartbeest 6 8 54 13

#### 2.6.1.1 Krav/antaganden

* Frekvens data (nominal skala)
* Väntevärdet/expected i varje cell ska vara minst 5 (Fishers exact test)

low\_expected <- rbind(  
 c(2,3,4),  
 c(4,5,6)  
)  
low\_expected\_test <- chisq.test(  
 low\_expected  
)

Warning in chisq.test(low\_expected): Chi-squared approximation may be incorrect

low\_expected\_test$expected

[,1] [,2] [,3]  
[1,] 2.25 3 3.75  
[2,] 3.75 5 6.25

#### 2.6.1.2 Anpassning

Kollar om en rad data “kan anpassas” efter en fördelning, vanligen lika många i varje kategori. Så om data ser ut såhär,

bockar["Gemsbock", , drop = FALSE] # Det extra ', drop = FALSE' gör att vi behåller namnet på raden (som annars försvinner mystiskt när matrisen har bara en rad).

FR CGA WMA NP  
Gemsbock 5 7 12 67

# Detta kan ignoreras och skulle då vara 'bockar["Gemsbock", ]'.  
  
bockar\_anpassning <- chisq.test(  
 bockar["Gemsbock", ]  
)  
bockar\_anpassning # Visa resultatet

Chi-squared test for given probabilities  
  
data: bockar["Gemsbock", ]  
X-squared = 115.9, df = 3, p-value < 2.2e-16

bockar\_anpassning$expected # Visa väntevärden

FR CGA WMA NP   
22.75 22.75 22.75 22.75

Alternativt, anpassning mot en specifik fördelning.

bockar\_anpassning <- chisq.test(  
 bockar["Gemsbock", ],  
 p = c(0.10, 0.15, 0.35, 0.40) # Totalt ska dessa bli 1  
)  
bockar\_anpassning # Visa resultatet

Chi-squared test for given probabilities  
  
data: bockar["Gemsbock", ]  
X-squared = 43.182, df = 3, p-value = 2.251e-09

bockar\_anpassning$expected # Visa väntevärden

FR CGA WMA NP   
 9.10 13.65 31.85 36.40

#### 2.6.1.3 Oberoende

Testa slumpmässiga observationer från två olika faktorer samtidigt för att se om de kommer från samma fördelning. Innebär att alla kolumner och alla rader kan ha olika summor. Så om data ser ut såhär,

# Skapa ny kolumn med summan av varje rad  
bockar\_sums <- cbind(  
 bockar,  
 Row\_sum = rowSums(bockar)  
)  
# Skapa ny rad med summan av varje kolumn  
bockar\_sums <- rbind(  
 bockar\_sums,  
 Col\_sum = colSums(bockar\_sums)  
)  
bockar\_sums

FR CGA WMA NP Row\_sum  
Gemsbock 5 7 12 67 91  
Red\_hartbeest 6 8 54 13 81  
Col\_sum 11 15 66 80 172

Varken alla rader eller alla kolumner har samma summor, alltså gör vi ett oberoende test.

bockar\_oberoende <- chisq.test(  
 bockar  
)  
bockar\_oberoende

Pearson's Chi-squared test  
  
data: bockar  
X-squared = 62.966, df = 3, p-value = 1.365e-13

bockar\_oberoende$expected

FR CGA WMA NP  
Gemsbock 5.819767 7.936047 34.9186 42.32558  
Red\_hartbeest 5.180233 7.063953 31.0814 37.67442

#### 2.6.1.4 Homogenitet

Testa slumpmässiga observationer från **en** faktor och kontrollera den andra. Innebär att **antingen** alla rader **eller** kolumner har samma summa (medan den andra varierar).

För detta behöver vi ny data. Ett stort företag vill undersöka om antalet vaccinerade anställda skiljer sig mellan avdelningarna. För att kontrollera för de olika avdelningarna slumpades 100 anställda från varje avdelning och frågade om de var vaccinerade eller inte.

uppsala\_site <- rbind( # Data  
 vaccinerad = c(RnD = 99, HR = 45, Lön = 63, IT = 76),  
 ej\_vaccinerade = c( 1, 55, 37, 24)  
)  
  
uppsala\_site

RnD HR Lön IT  
vaccinerad 99 45 63 76  
ej\_vaccinerade 1 55 37 24

Vi kan utforska företagets data lite mer genom att lägga till rad och kolumn summor.

# Skapa ny kolumn med summan av varje rad  
uppsala\_sums <- cbind(  
 uppsala\_site,   
 Row\_sum = rowSums(uppsala\_site)  
)  
# Skapa ny rad med summan av varje kolumn  
uppsala\_sums <- rbind(  
 uppsala\_sums,   
 Col\_sum = colSums(uppsala\_sums)  
)  
uppsala\_sums

RnD HR Lön IT Row\_sum  
vaccinerad 99 45 63 76 283  
ej\_vaccinerade 1 55 37 24 117  
Col\_sum 100 100 100 100 400

Detta säger oss att den faktorn är kontrollerad, alltså gör vi ett Homogenitetstest.

vaccin\_homogen <- chisq.test(  
 uppsala\_site  
)  
  
vaccin\_homogen

Pearson's Chi-squared test  
  
data: uppsala\_site  
X-squared = 74.839, df = 3, p-value = 3.923e-16

vaccin\_homogen$expected

RnD HR Lön IT  
vaccinerad 70.75 70.75 70.75 70.75  
ej\_vaccinerade 29.25 29.25 29.25 29.25

### 2.6.2 Fisher’s exact (icke parametrisk)

Ifall en eller flera celler med väntevärden/expected är < 5 kan man använda Fisher’s exact test.

uppsala\_fisher <- fisher.test(  
 uppsala\_site  
)  
  
uppsala\_fisher # Inga expected values

Fisher's Exact Test for Count Data  
  
data: uppsala\_site  
p-value < 2.2e-16  
alternative hypothesis: two.sided

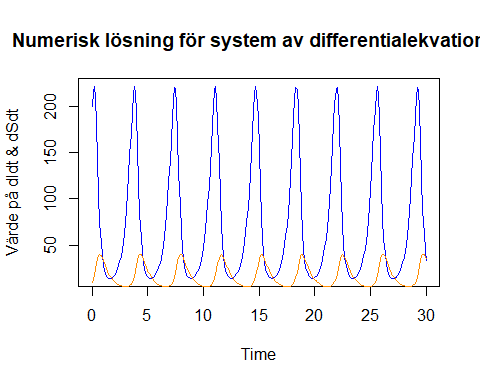
# 3. Matte

## 3.1 Matematiska beräkningar

1 + 1 # addition  
1 - 1 # subtraktion  
5 \* 4 # multiplikation  
6 / 2 # division  
5^2 # upphöjt med  
  
sqrt(25) # roten ur, (positiv)  
exp(3) # e^3  
log(7) # log(7) med basen e  
log10(7) # log(7) med basen 10

## 3.2 Differentialekvationer

# Funktion med ekvationssystemet att lösa  
DE\_fun <- function(times, # Intervall  
 y, # Begynnelsevärden  
 parms # Andra värden  
) {  
 # variabler att använda (från parms)  
 alfa <- parms["alfa"]  
 beta <- parms["beta"]  
 delta <- parms["delta"]  
 gamma <- parms["gamma"]  
   
 # Begynnelsevärden (från y)  
 B <- y["B"]  
 P <- y["P"]  
   
 # Differentialekvationen att lösa numeriskt  
 # (Minst 1 ekv. men sedan är det bara att lägga till fler ekv.  
 # på var sin rad. Varje ekv. måste ha ett begynnelsevärde &  
 # sedan returneras i result\_vec)  
 dB <- alfa \* B - beta \* B \* P  
 dP <- delta \* B \* P - gamma \* P  
   
 # Spara & returnera resultatet (ode() är lite speciell...)  
 result\_vec <- c( # Först som vektor  
 dB,  
 dP  
 )  
 result\_list <- list(result\_vec) # Sedan som lista  
 return(result\_list) #Returnera  
}  
  
# Vektor med alla tidssteg (t) att sätta in  
DE\_time\_span <- seq(  
 from = 0, # Början på intervall  
 to = 30, # Slut på intervall  
 by = 0.1 # Steglängd  
)  
  
# Vektor med begynnelsevärden   
# (enklast att namnge dem och sedan indexera med namnet i fun)  
DE\_init <- c(  
 B = 200,  
 P = 10  
)  
  
# Vektor med andra värden för formeln  
# (enklast att namnge dem och sedan indexera med namnet i fun)  
DE\_params <- c(  
 alfa = 2.5,  
 beta = 0.15,  
 delta = 0.02,  
 gamma = 1.5  
)  
  
# Lösa ekvationerna numeriskt  
DE\_sol <- ode(  
 y = DE\_init, # Begynnelsevärden  
 times = DE\_time\_span, # Intervall  
 func = DE\_fun, # Funktionen  
 parms = DE\_params, # Andra värden  
 method = "rk4" # Runge-Kutta version 4  
)  
# Konvertera till data frame för att göra den lättare att hantera  
DE\_sol <- as.data.frame(DE\_sol)  
  
  
# Plotta dB respektive dP mot tid  
plot( # Plot fönster & dB mot tid  
 DE\_sol$time,  
 DE\_sol$B,  
 type = "l",  
   
 col = "blue",  
   
 xlab = "Time",  
 ylab = "Värde på dIdt & dSdt",  
 main = "Numerisk lösning för system av differentialekvationer"  
)  
lines( # Lägg till dP mot tid  
 DE\_sol$time,  
 DE\_sol$P,  
   
 col = "darkorange"  
)



# 4. KvantBio

## 4.1 Fibonacci

n <- c(1,2)  
for (i in 3:25) {  
 n[i] <- n[i-1] + n[i-2]  
}  
n

[1] 1 2 3 5 8 13 21 34 55 89  
[11] 144 233 377 610 987 1597 2584 4181 6765 10946  
[21] 17711 28657 46368 75025 121393

## 4.2 Hantera modeller

Kunna använda modeller och lösa för saknade parametrar. Översätt de matematiska funktionerna till funktioner i R. Funktionen uniroot() är användbar för att hitta nollställen.

### 4.2.1 Allometri & energiförbrukning

Bioenergetisk modell där K = konsumtion, P = produktion/ökning i massa, M = metabolism, E = exkretion.

Modell för citronhajars metabolism baserat på sambandet mellan simhastighet och syreförbrukning (för hajar som väger 1kg).

### 4.2.2 Individuell tillväxt (von Bertalanffys tillväxtekvation)

Där L\_inf är maxlängd vid oändlig tid, L\_noll är längden vid t = 0 , k är en tillväxtfaktor och t är tid/ålder.

För att hitta längden vid givet t.

# Parametrar  
L\_inf <- 85  
k <- 0.00082  
L\_noll <- 30  
  
# Funktion av t  
growth\_av\_t <- function(t) {  
 L\_t <- L\_inf + (L\_noll - L\_inf) \* exp(-k \* t)  
 return(L\_t)  
}  
growth\_av\_t(t = 10)

[1] 30.44916

För att hitta L\_inf (eller annan okänd från funktionen) kan vi möblera om till

och använda uniroot().

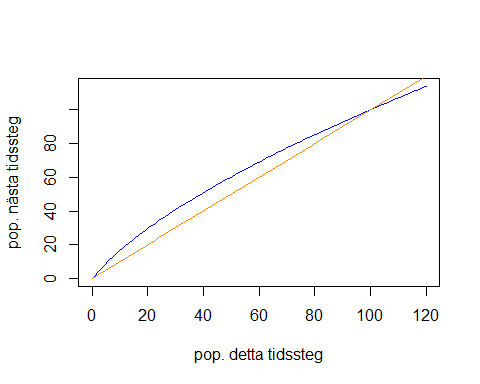
# Parametrar  
k <- 0.00082  
L\_noll <- 30  
L\_t <- 95  
t <- 2000  
  
# Funktion av L\_inf  
growth\_av\_L\_inf <- function(L\_inf) {  
 noll\_punkt <- L\_inf + (L\_noll - L\_inf) \* exp(-k \* t) - L\_t  
 return(noll\_punkt)  
}  
  
# Hitta nollpunkten med uniroot()  
uniroot(  
 f = growth\_av\_L\_inf,  
 interval = c(0,200)  
)$root

[1] 110.6432

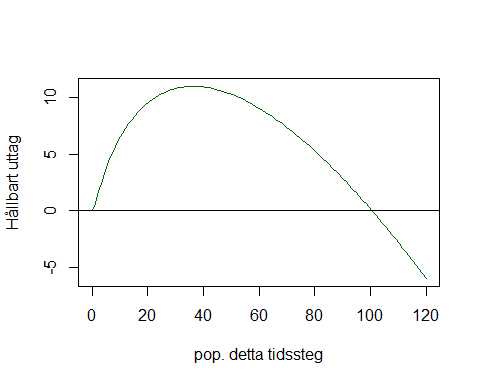
## 4.3 Hållbart uttag

Hur tillväxer en population och vid vilken storlek kan man plocka ut så många individer som möjligt och samtidigt få en stabil population?

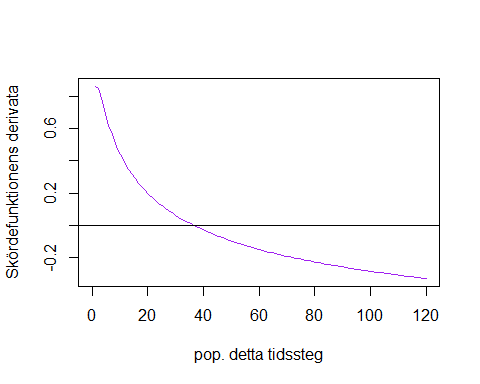
# Rekryteringsfunktion, n individer detta år ger n+x individer nästa år  
recruitment <- function(n) {  
 recruit <- 2.17 \* sqrt(n) \* log(n+1)  
 return(recruit)  
}  
# Plot  
curve(  
 recruitment,  
 xlim = c(0,120),  
 col = "blue",  
 xlab = "pop. detta tidssteg",  
 ylab = "pop. nästa tidssteg"  
)  
curve(  
 1\*x,  
 add = TRUE,  
 col = "darkorange"  
)



# Skördekurva  
harvest <- function(n) {  
 harvest <- recruitment(n) - n  
 return(harvest)  
}  
  
# Plot  
curve(  
 harvest,  
 xlim = c(0,120),  
 xlab = "pop. detta tidssteg",  
 ylab = "Hållbart uttag",  
 col = "darkgreen"  
)  
abline(  
 h = 0  
)



# Hitta x-värde för maxpunkt, alltså pop. storlek för max hållbart uttag.  
  
# Derivata av skördefunktionen  
harvest\_deriv <- function(n) {  
 harvest\_deriv <- fderiv(harvest, n)  
 return(harvest\_deriv)  
}  
  
# Plot (indikerar intervallet för uniroot())  
curve(  
 harvest\_deriv,  
 xlim = c(0,120),  
 xlab = "pop. detta tidssteg",  
 ylab = "Skördefunktionens derivata",  
 col = "purple"  
)  
abline(  
 h = 0  
)



# Hitta nollpunk numeriskt = pop. storlek för max hållbart uttag  
harvest\_deriv\_root <- uniroot(  
 harvest\_deriv,  
 interval = c(20,60)  
)

# Insättning av nollpunkt i skördefunktion ger max hållbart uttag  
harvest(harvest\_deriv\_root$root)

[1] 11.0155

## 4.4 Diskret logistisk tillväxt

Funktion för diskret logistisk tillväxt.

dlt <- function(n, r, k) {  
 n\_ny <- n + r\*n\*(1-n/k)  
 return(n\_ny)  
}

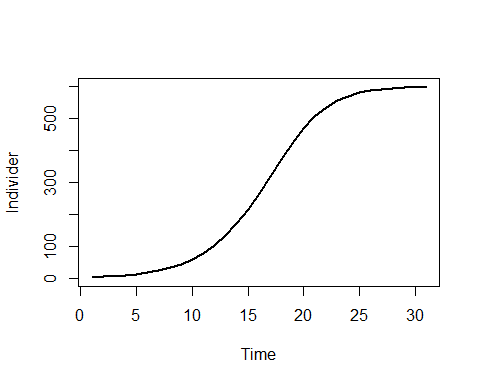
### 4.4.1 Modellera en population

Definiera värden på parametrar.

r <- 0.36 # perkapitareproduktion (egenvärde från Leslie-matris)  
k <- 600 # Bärkraft  
  
sluttid <- 30  
t <- 1:(sluttid+1)

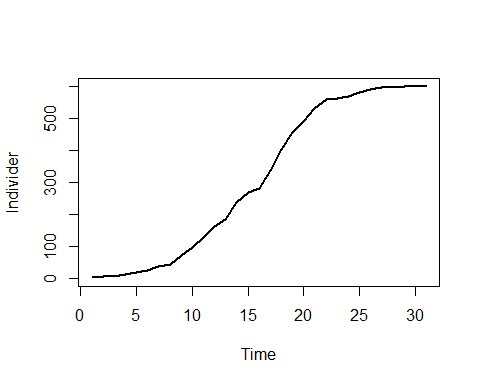
En populationsutveckling med konstant r-värde.

n <- c(4) # Startvärde & vektor att lagra resultat i  
  
for (i in 1:sluttid) { # Loop för pop. utveckling  
 n[i+1] <- dlt(n = n[i],  
 r = r,  
 k = k)  
}  
  
plot(  
 x = t,  
 y = n,  
 ylim = c(0,k),  
   
 type = "l",  
 lwd = 2,  
   
 xlab = "Time",  
 ylab = "Individer"  
)



Samma, men med varierande r-värde.

# r <- 0.36 # perkapitareproduktion (egenvärde från Leslie-matris)  
k <- 600 # Bärkraft  
  
sluttid <- 30  
t <- 1:(sluttid+1)  
  
n <- c(4) # Startvärde & vektor att lagra resultat i  
  
for (i in 1:sluttid) { # Loop för pop. utveckling  
 n[i+1] <- dlt(n = n[i],  
 r = rnorm(n = 1,  
 mean = 0.36,  
 sd = 0.2),  
 k = k)  
}  
  
plot(  
 x = t,  
 y = n,  
 ylim = c(0,k),  
   
 type = "l",  
 lwd = 2,  
   
 xlab = "Time",  
 ylab = "Individer"  
)



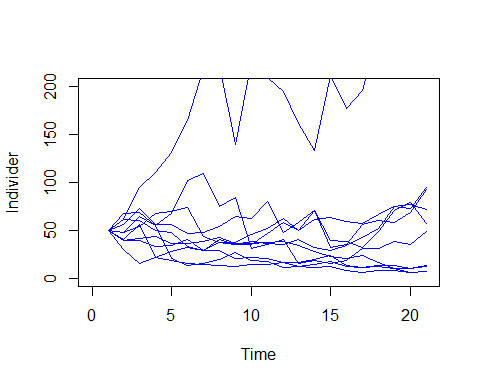
### 4.4.2 Modellera många populationer & beräkna utdöende risk

Definiera värden på parametrar.

# r defineras i for-loop  
k <- 1000 # Bärkraft  
  
sluttid <- 20  
t <- 1:(sluttid+1)  
  
collapse <- 10 # Färre individer = pop. dör ut

Skapa många linjer för hypotetiska populationers öden.

plot( # Skapa tom plot  
 NULL,  
 xlim = c(0, sluttid+1),  
 ylim = c(0, 200),  
 type = "l",  
 xlab = "Time",  
 ylab = "Individer"  
)  
  
n <- c(50) # Startvärde & vektor att lagra resultat i  
pop\_collapse <- 0 # Räknare för hur många populationer har dött ut  
attempts <- 10  
  
for (i in 1:attempts) { # Loop för upprepade försök/linjer  
   
 for (j in 1:sluttid) { # Loop för en linje/population  
 n[j+1] <- dlt(n = n[j],  
 r = rnorm(n = 1, # slumpad reproduktion  
 mean = 0,  
 sd = 0.25),  
 k = k)  
 }  
   
 if (min(n) < collapse) { # Kolla om pop. collapsade  
 pop\_collapse <- pop\_collapse +1 # Addera 1 till räknare  
 print(paste("Försök",i,"dog ut. Totalt",pop\_collapse,"collapsade populationer."))  
 }  
   
 lines( # Addera linje för ett försök  
 x = t,  
 y = n,  
 lwd = 1,  
 col = "blue"  
 )  
   
}



[1] "Försök 3 dog ut. Totalt 1 collapsade populationer."  
[1] "Försök 5 dog ut. Totalt 2 collapsade populationer."  
[1] "Försök 9 dog ut. Totalt 3 collapsade populationer."

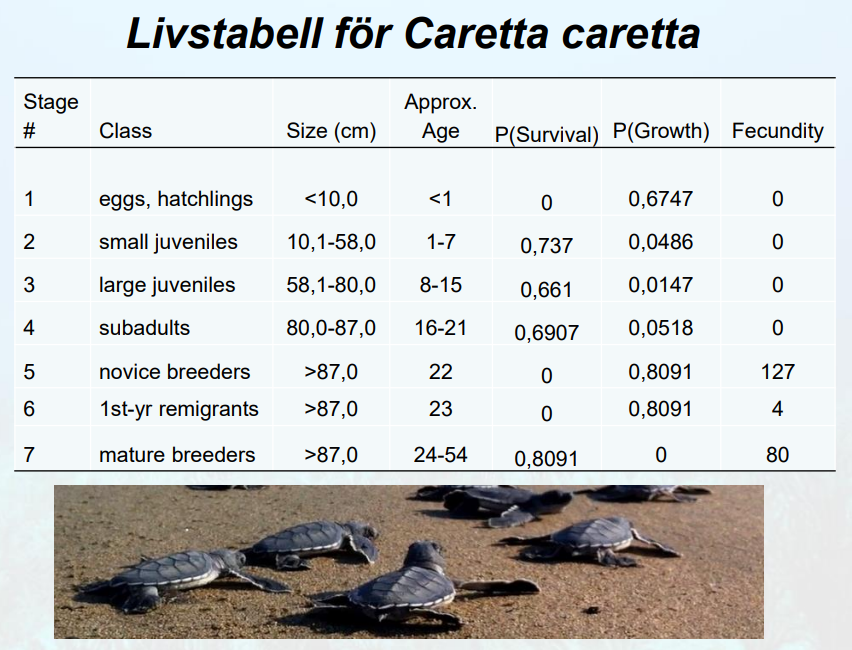
För att beräkna utdöenderisk i procent.

extiction\_rate <- (pop\_collapse / attempts) \*100  
print(paste("Utdöenderisken är",extiction\_rate,"%"))

[1] "Utdöenderisken är 30 %"

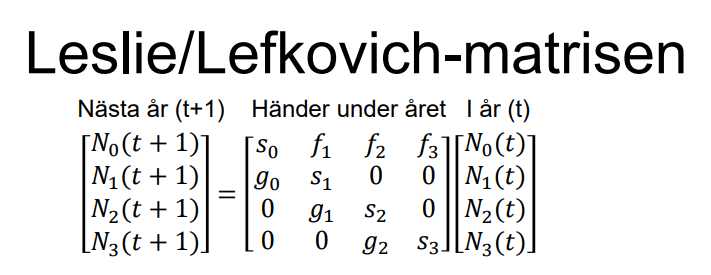
## 4.5 Lesliematriser

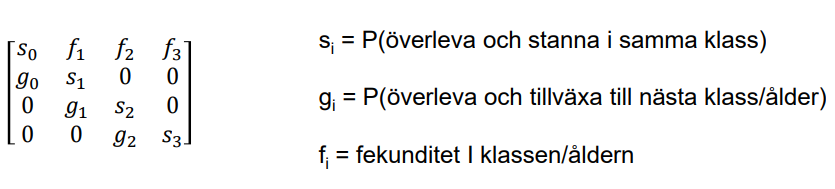
För användning av en livstabell översätts den till en Leslie/Lefkovitch matris.



En Leslie-matris använder åldersklasser där ingen kan “stanna” i klassen, alltså är diagonalen 0.

En Lefkovitch-matris använder ontogenetiska stadier där en individ kan antingen “stanna” i samma klass eller “gå vidare” till nästa klass, alltså kan diagonalen vara >0.





caretta\_matris <- rbind( # En Lefkovitch matris  
 r1 = c(c1 = 0, c2 = 0, c3 = 0, c4 = 0, c5 = 127, c6 = 4, c7 = 80),  
 r2 = c(0.6747, 0.737, 0, 0, 0, 0, 0),  
 r3 = c( 0, 0.0486, 0.661, 0, 0, 0, 0),  
 r4 = c( 0, 0, 0.0147, 0.6907, 0, 0, 0),  
 r5 = c( 0, 0, 0, 0.0518, 0, 0, 0),  
 r6 = c( 0, 0, 0, 0, 0.8091, 0, 0),  
 r7 = c( 0, 0, 0, 0, 0, 0.8091, 0.8091)  
)  
caretta\_matris

c1 c2 c3 c4 c5 c6 c7  
r1 0.0000 0.0000 0.0000 0.0000 127.0000 4.0000 80.0000  
r2 0.6747 0.7370 0.0000 0.0000 0.0000 0.0000 0.0000  
r3 0.0000 0.0486 0.6610 0.0000 0.0000 0.0000 0.0000  
r4 0.0000 0.0000 0.0147 0.6907 0.0000 0.0000 0.0000  
r5 0.0000 0.0000 0.0000 0.0518 0.0000 0.0000 0.0000  
r6 0.0000 0.0000 0.0000 0.0000 0.8091 0.0000 0.0000  
r7 0.0000 0.0000 0.0000 0.0000 0.0000 0.8091 0.8091

För att få ut egenvärdet (r-värdet) och egenvektorn (stabil stadiefördelning). Om är 0.90 innebär det att populationen minskar med 10% (förändrings faktor för populationen). Den relativa fördelningen ( ) indikerar hur många procent av den totala populationen bör finnas i varje klass. Kan användas som p = relativ\_dist i chisq.test() för att göra ett anpassningstest med en observerad population.

caretta\_eigen <- eigen(caretta\_matris)  
  
# Egentligen det största egenvärdet, men det bör vara första.  
Re(caretta\_eigen$values[1])

[1] 0.9450846

# Samma index för egenvektor som egenvärdet.  
Re(caretta\_eigen$vectors[,1])

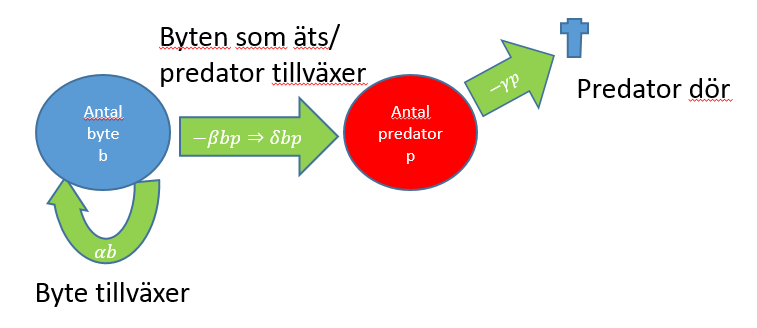
[1] 0.2908382596 0.9430228664 0.1613283678 0.0093226030 0.0005109710  
[6] 0.0004374493 0.0026027957

# Relativ fördelning i procent  
caretta\_dist <- Re(caretta\_eigen$vectors[,1])  
caretta\_dist\_procent <- (caretta\_dist/sum(caretta\_dist))\*100  
caretta\_dist\_procent

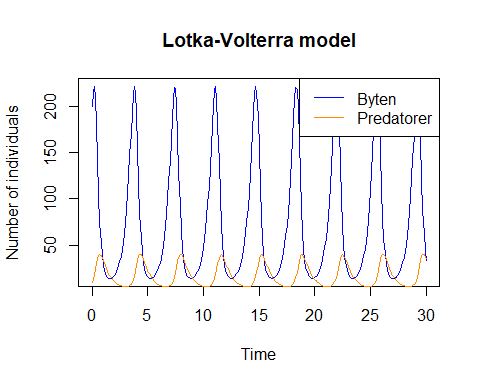
[1] 20.65519760 66.97304431 11.45746547 0.66208692 0.03628892 0.03106745  
[7] 0.18484934

## 4.6 Lotka-Volterra byte-predator modell

En specifik model som bygger på att lösa ett ekvationssystem av differentialekvationer.



# Funktion med ekvationssystemet att lösa  
LV\_fun <- function(times, # Intervall  
 y, # Begynnelsevärden  
 parms # Andra värden  
) {  
 # variabler att använda (från parms)  
 alfa <- parms["alfa"]  
 beta <- parms["beta"]  
 delta <- parms["delta"]  
 gamma <- parms["gamma"]  
# K <- parms["K"] # alt. där K = bärkraft  
   
 # Begynnelsevärden (från y)  
 B <- y["B"]  
 P <- y["P"]  
   
 # Differentialekvationen att lösa numeriskt  
 dBdt <- (alfa \* B - beta \* B \* P)  
# dBdt <- (alpha \* B(1 - B/K) - beta \* B \* P) # alt. där K = bärkraft  
   
 dPdt <- (delta \* B \* P - gamma \* P)  
   
 # Spara & returnera resultatet  
 result\_vec <- c(  
 dBdt,  
 dPdt  
 )  
 result\_list <- list(result\_vec)  
 return(result\_list)  
}  
  
# Vektor med alla tidssteg (t) att sätta in  
time\_span\_LV <- seq(  
 0,  
 30,  
 by = 0.1  
)  
  
# Vektor med begynnelsevärden  
init\_LV <- c(  
 B = 200,  
 P = 10  
)  
  
# Vektor med andra värden för formeln  
params\_LV <- c(  
 alfa = 2.5, # Tillväxt byten  
 beta = 0.15, # Byten som äts  
 delta = 0.02, # Tillväxt predator  
 gamma = 1.5 # Död predator  
# K = 500 # alt. där K = bärkraft  
)  
  
# Lösa ekvationerna numeriskt  
sol\_LV <- ode(  
 y = init\_LV, # Begynnelsevärden  
 times = time\_span\_LV, # Intervall  
 func = LV\_fun, # Funktionen  
 parms = params\_LV, # Andra värden  
 method = "rk4" # Runge-Kutta version 4  
)  
sol\_LV <- as.data.frame(sol\_LV)  
  
  
# Plotta antal insekter [milljoner] respektive spindlar [tusen] mot tid  
plot(  
 sol\_LV$time,  
 sol\_LV$B,  
 type = "l",  
   
 col = "blue",  
   
 xlab = "Time",  
 ylab = "Number of individuals",  
 main = "Lotka-Volterra model"  
)  
lines(  
 sol\_LV$time,  
 sol\_LV$P,  
   
 col = "darkorange"  
)  
legend(  
 "topright",  
 legend = c("Byten",   
 "Predatorer"),  
   
 lty = c(1, 1),  
 col = c("blue",  
 "darkorange")  
)



# "Fasporträtt" (två pop. mot varandra där tiden blir att följa linjen)  
plot(  
 sol\_LV$B,  
 sol\_LV$P,  
 type = "l",  
   
 col = "darkgreen",  
   
 xlab = "Byten",  
 ylab = "Predatorer",  
 main = "Fasporträtt"  
)

