RHUL Psychology Statistical modelling notebook

Matteo Lisi

2025-07-16

Contents

Chapter 1

About

This online resource is created and maintained by Matteo Lisi and is designed as a collective repository for the staff and students of the Department of Psychology of Royal Holloway, University of London. It contains a miscellaneous collection of tutorials, examples, case studies, workshop materials, and a variety of additional content focusing on data analysis and modelling. These materials will be updated and expanded regularly in response to the most frequent questions and requests received.

A pdf version is available at this link.

Chapter 2

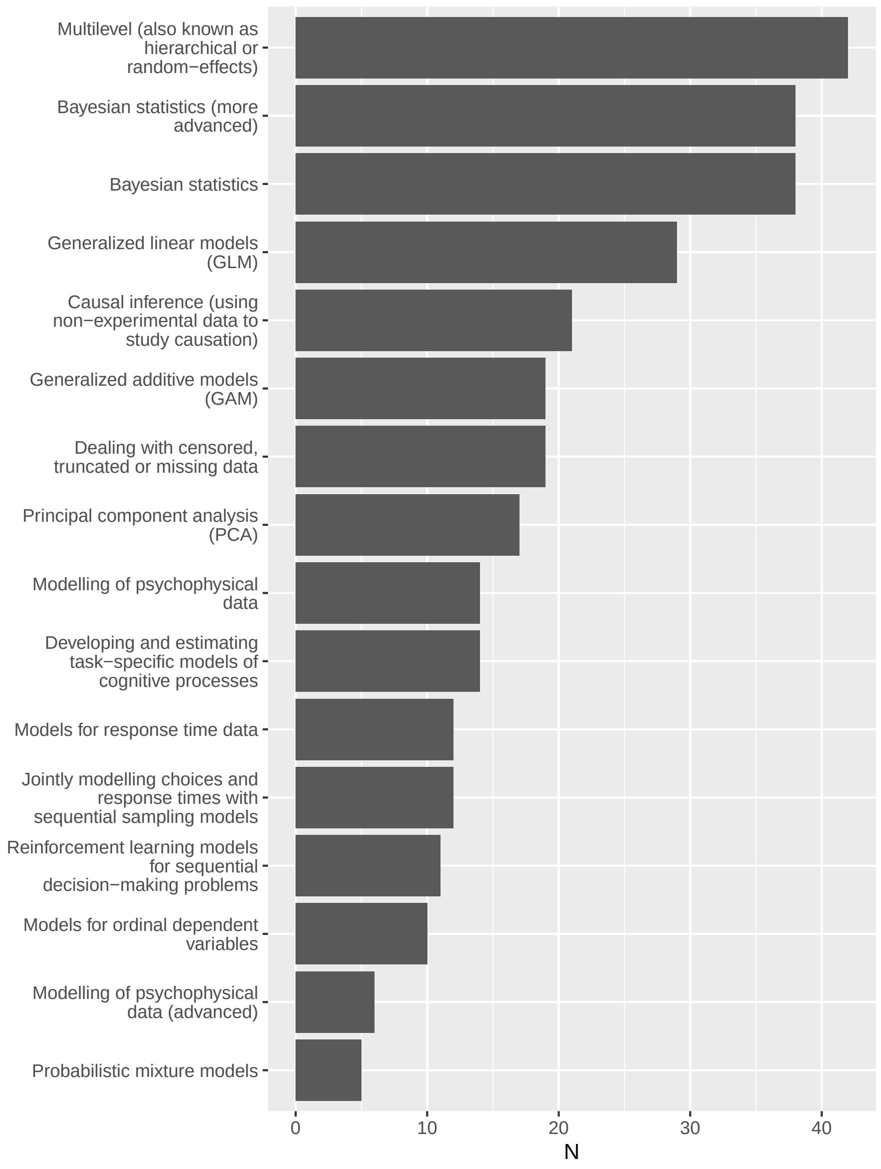
Departmental survey about statistical methods

I used an anonymous survey to ask colleagues some questions about which topics may be more interesting or useful in their research.

2.1 March 2022

2.1.1 Question 1

In the first question people indicated topics of interests. The winner are multi-level models, followed closely by Bayesian statistics.



There were some additional suggestions.

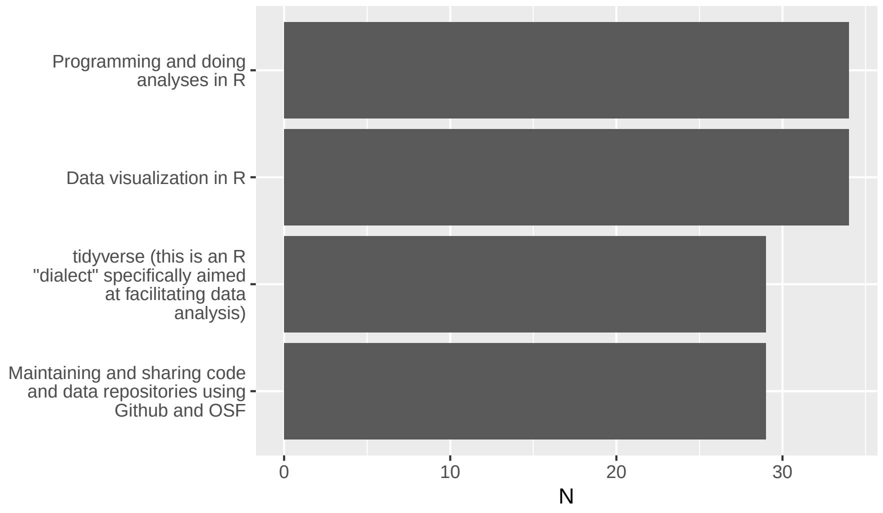
```
#> [1] "power analyses using Shiny apps"
#> [2] "agent-based models"
#> [3] "this may be covered in the above, but approaches to analysing experience sampli
#> [4] "Methods for longitudinal analyses"
#> [5] "Network modelling"
#> [6] "Neural networks, Markov processes"
#> [7] "Random forests and related"
#> [8] "causal modelling using regression models - path models etc"
#> [9] "prediction modelling"
```

A few other topics were mentioned in the comment section:

- Shiny apps
- Network modelling
- Longitudinal analyses
- Random forests
- Neural network

2.1.2 Question 2

Here people indicated their interest for topics related to data analysis.

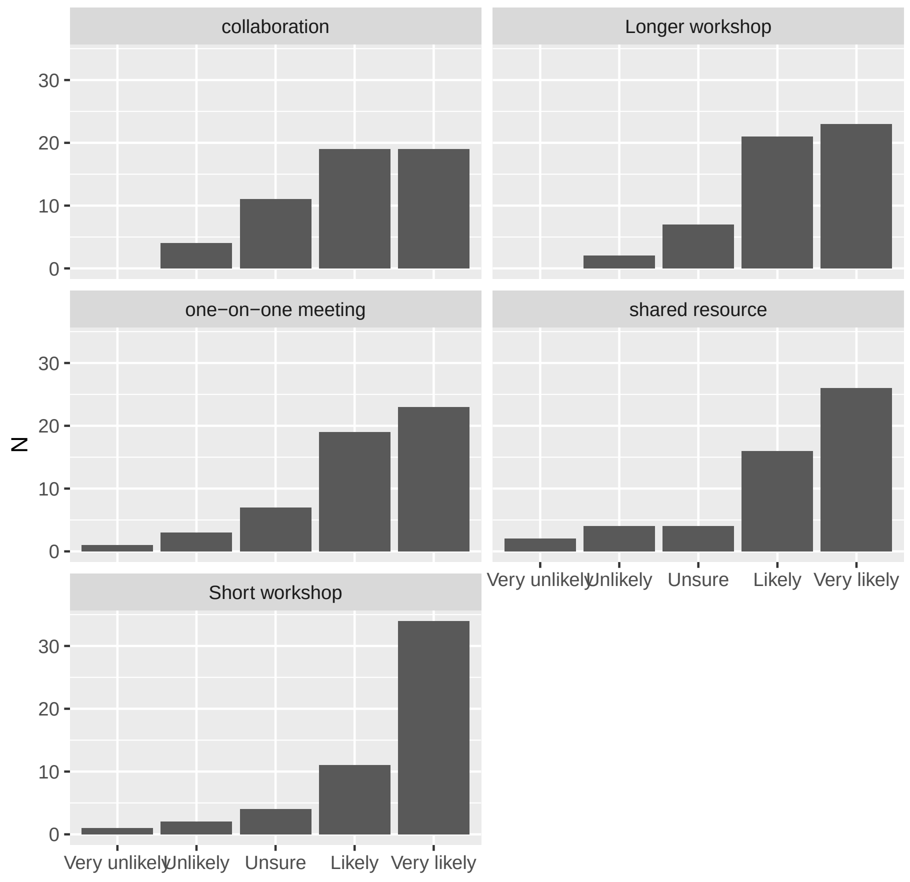


Other things mentioned in the comments were:

- SPM
- Docker
- Python

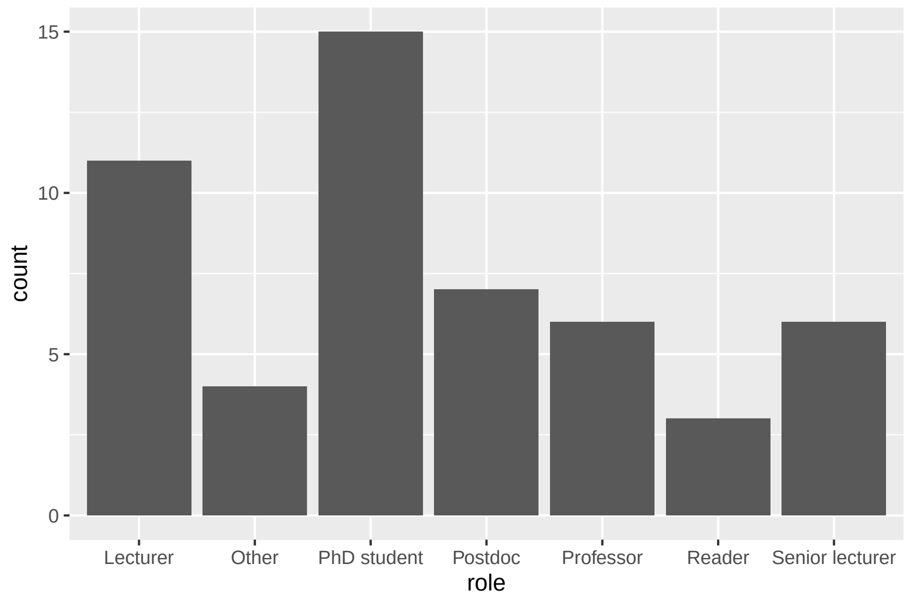
2.1.3 Question 4

This question was about likelihood of using different formats of support



2.1.4 Respondents' status

The final questions asked about the status / career level.


15
10
5
0
count
Lecturer
Other
PhD student
Postdoc
Professor
Reader
Senior lecturer
role

Chapter 3

Introduction to R

Most of the practical statistical tutorials and recipes in this book use the software R, so this section provides some introduction to R for the uninitiated.

3.1 Installing R

The base R system can be downloaded at the following link, which provides installers for both Windows, Mac and Linux:

https://cran.rstudio.com/

In addition to the base R system, it is useful to have also R-studio, which is an IDE (Integrated Development Environment) for R, and provides both an editor, a graphical interface and much more. It can be downloaded from:

https://www.rstudio.com/products/rstudio/download/

3.2 First steps

R is a programming language and free software environment for statistical computing and graphics. It is an *interpreted language*, which means that to give instructions to the computer you do not have to compile it first in machine language, everything is done 'on the fly' through a command line interpreter, e.g. if you type 2+2 in the command line R, the computer will reply with the answer (try this on your computer):

```
2+2
#> [1] 4
```

Typically the normal workflow involve writing and saving a series of instructions in a *script* file (usually saved with the .R extension), which can be executed (either step by step or all at once). Since all steps of the analyes are documented in the script, this makes them transparent and reproducible.

In an R script you can use the # sign to add comments, so that you and others can understand what the R code is about. Commented lines are ignored by R, so they will not influence your result. See the next example:

```
# calculate 3 + 4
3 + 4
#> [1] 7
```

3.2.1 Arithmetic with R

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Exponentiation: ^
- Modulo: %%

The last two might need some explaining:

The ^ operator raises the number to its left to the power of the number to its right: for example 3^2 is 9.

The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 (or 5 %% 3) is 2.

3.2.2 Variable assignment

A basic concept in programming (statistical or not) is called a *variable.*

A variable allows you to store a value (e.g. 2) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

You can assign a value `2` to a variable `my_var` with the command

```
my_var <- 2
```

Note that you would have obtained the same result using:

```
2 -> my_var
```

that is, the *assignment operator* works in both directions `<-` and `->`.

The variable can then be used in any computation, for example:

```
my_var + 2
#> [1] 4
```

3.2.3 Basic data types in R

Variables can be of many types, not just numerical values. For example, they can contain *text* values (e.g. a string of characters). Arithmetic operators such as `+` do no work with these. If you tried to apply them characters R will give you an error message.

```
# Assign a value to the variable apples
apples <- 5

# Assign a text value
oranges <- "six"

#
apples + oranges
#> Error in apples + oranges: non-numeric argument to binary operator
```

In fact R works with numerous data types, and some of these are not numerical (so they can't be added, subtracted, etc.). Some of the most basic types to get started are:

- Decimal values like 4.5 are called numerics.

- Natural numbers like 4 are called integers. Integers are also numerics.
- Boolean values (`TRUE` or `FALSE`, abbreviated `T` and `F`) are called logical[1].
- Text (or string) values are called characters.

3.2.4 Vectors and other data types

Additionally, the simple data types listed above can be combined in more complex 'objects' that can comprise several values. For example, we can obtain a *vector* by concatenating values using the function `c()`. This can be applied both on numerical or character data types, e.g.

```
some_numbers <- c(4,87,10, 0.5, -6)
some_numbers
#> [1]  4.0 87.0 10.0  0.5 -6.0
```

```
my_modules <- c("PS115", "PS509", "PS300", "PS938", "PS9457")
my_modules
#> [1] "PS115"  "PS509"  "PS300"  "PS938"  "PS9457"
```

There are some special handy functions to create specific types of vectors, such as *sequences* (using the function `seq()` or the operator `:`)

```
x <- seq(from = -10, to = 10, by = 2)
x
#>  [1] -10  -8  -6  -4  -2   0   2   4   6   8  10
```

```
y <- seq(-0, 1, 0.1)
y
#>  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
z <- 1:5
z
#> [1] 1 2 3 4 5
```

Another useful type of vector can be obtained by *repetition* of elements, and this can be numerical, character, or even applied to other vectors

[1]Note that you can add or multiply logical Boolean values: internally `FALSE` are treated as zeroes, and `TRUE` as ones.

```
rep(3, 5)
#> [1] 3 3 3 3 3
```

```
x <- 1:3
rep(x, 4)
#>  [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

```
rep(c("leo the cat", "daisy the dog"), 2)
#> [1] "leo the cat"   "daisy the dog" "leo the cat"
#> [4] "daisy the dog"
```

We can combine vectors of different types into a *data frame*, one of the most useful ways of storing data in R. Let's say we have 3 vectors:

```
# create a numeric vector
a <- c(0, NA, 2:4)  # NA means not available

# create a character vector
b <-  c("PS115", "PS509", "PS300", "PS938", "PS9457")

# create a logical vector
c <- c(TRUE, FALSE, TRUE, FALSE, FALSE)  # must all be caps!
```

we can combine them into a data.frame using:

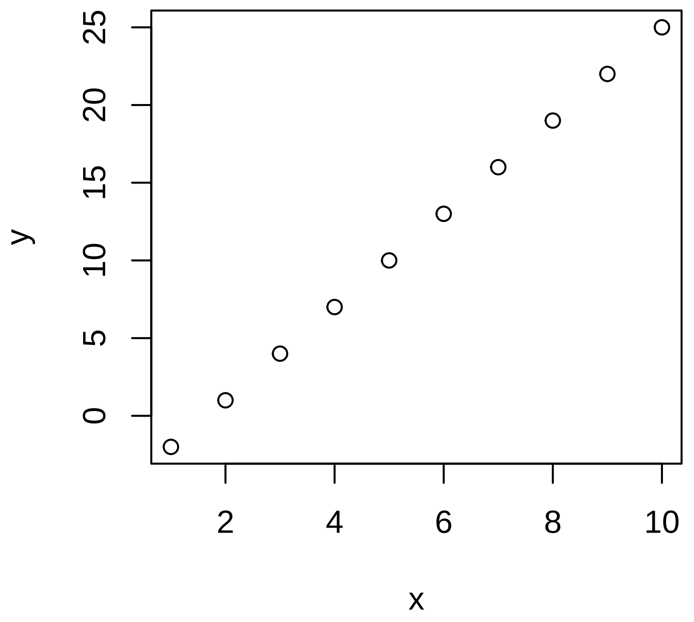
```
# create a data frame with the vectors a, b,and c that we just created
my_dataframe <- data.frame(a,b,c)

# we could also change the column names (currently they are a, b, c)
colnames(my_dataframe) <- c("some_numbers", "my_modules", "logical_values")

# now let's have a look at it
my_dataframe
#>   some_numbers my_modules logical_values
#> 1            0      PS115           TRUE
#> 2           NA      PS509          FALSE
#> 3            2      PS300           TRUE
#> 4            3      PS938          FALSE
#> 5            4     PS9457          FALSE
```

Although note that in most cases we would probably import a dataframe from an external data file, for example using the functions `read.table` or `read.csv`.

3.2.5 Basic plotting in R

We can create plots using the function `plot()`. For example:

```
x = 1:10
y = 3*x - 5
plot(x, y)
```

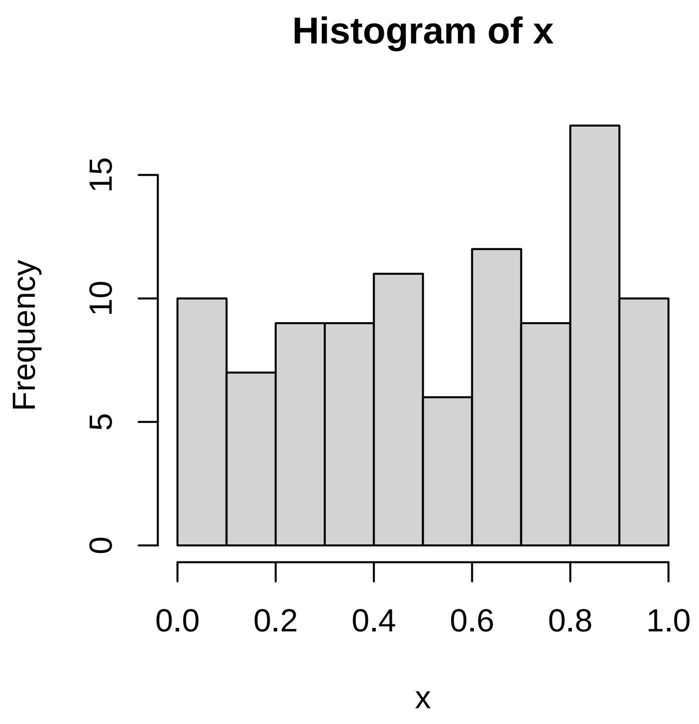


3.2.6 Other operations

3.2.6.1 Random number generation

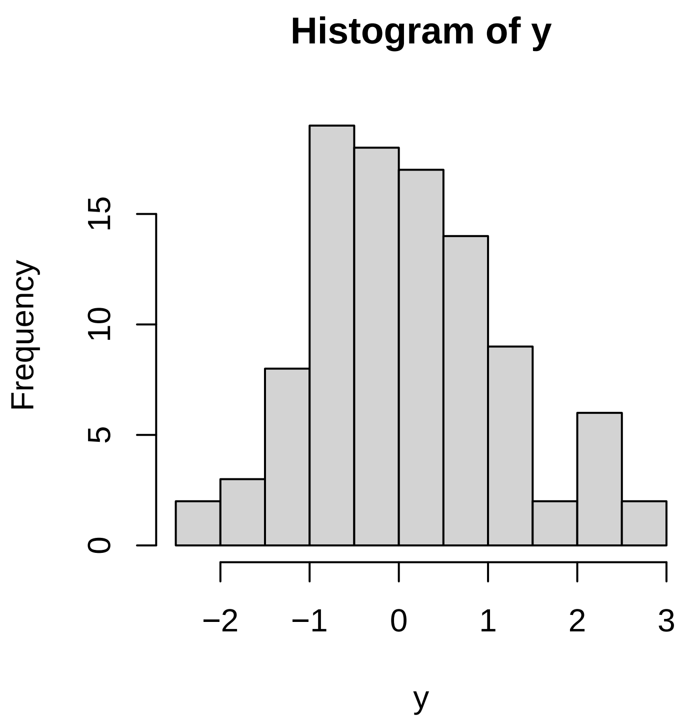
Generate uniformly distributed random numbers (function `runif()`)

```
x <- runif(100, min = 0, max = 1)
hist(x)
```



Generate numbers from a normal distribution

```r
y <- rnorm(100, mean = 0, sd = 1)
hist(y)
```



3.2.7 Getting help

R has a lot of functions, and extra packages that can provides even more. It may seem a bit overwhelming, but it is very easy to get help about how to use a function: just type in a question mark, followed by the name of the function. For example, to see the help of the function we used above to generate the histogram, type

```
?hist
```

3.3 Resources for learning R

There is plenty of resources on the web to learn R. I will recommend a couple that I think are particularly well-done and useful:

- Software Carpentry tutorials on R for Reproducible Scientific Analysis
- The free book Learning Statistics with R by Danielle Navarro

Chapter 4

Correlations

Readers of this page will likely already familiar with the Pearson correlation coefficient and its significance test. The Pearson correlation coefficient measures the strength of association between two variables, denoted as x and y, and is essentially a normalized measure of their covariance. It is calculated by dividing the covariance of x and y by the product of their standard deviations:

$$r_{x,y} = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y}$$

The normalization is achieved by dividing the covariance by the maximum possible covariance that can be attained when the variables are perfectly correlated, represented by the product of their standard deviations. This normalization constrains the Pearson correlation coefficient to a range between -1 and 1.

In R, simple correlation analyses can be run using the function `cor.test`.

In this chapter we will consider some more complex examples and issues that may occur when doing correlation analyses.

4.1 Comparing correlations

Often, we have to compare a correlation value between two conditions (or groups or whatever). One simple approach to do this - if we have access to the raw data - is to run a regression model (with interaction term) with the condition as categorical predictor, and test whether the slope differs between the two conditions. (Note that if all the variables are standardized - i.e. transformed in Z-scores - the regression slope is identical to the Pearson correlation coefficient.)

If we don't have the raw data available, we can still compare the correlations using Fisher's Z transform. This requires first transforming all correlation values, e.g. r_1, r_2, into Z scores using the following:

$$Z = \frac{\log(1 + r) - \log(1 - r)}{2}$$

We can then take the differences in these transformed correlation coefficients, and divide by $\sqrt{\frac{1}{N_1 - 3} + \frac{1}{N_2 - 3}}$, where N_1 and N_2 are the sample size of the two conditions/groups that are compared. We obtain the statistics

$$\Delta_Z = \frac{|Z_1 - Z_2|}{\sqrt{\frac{1}{N_1 - 3} + \frac{1}{N_2 - 3}}}$$

Under the null hypothesis that the two correlations are not different ($H_0 : r_1 = r_2$) the statistics is normally distributed with mean 0 and variance 1, $\Delta_Z \sim \mathcal{N}(0, 1)$. Thus we can get a p value by using the cumulative distribution function of the normal distribution. For example, in R if the statistic is saved in the variable `Delta_Z` we can calculate the p-value using `p_value <- 2*(1-pnorm(Delta_Z))`.

To verify the correctness of this approach, we can perform a simple simulation in R. The code below simulates multiple pairs of datasets, each consisting of two variables, from a population with a given correlation. In this simulation, the 'true' correlation value remains constant, representing the case where the null hypothesis is true. We then use the aforementioned approach to determine whether the difference is statistically significant. Finally, we calculate the proportion of significant tests (i.e., false positives) and verify that it is approximately equal to the desired false positive rate (alpha), conventionally set at 0.05.

```r
# function to simulate correlated normal variables
sim_data <- function(N=100, r=0.3){
  d <- MASS::mvrnorm(N,
              mu = c(0,0),
              Sigma=matrix(c(1,r,r,1), nrow=2, ncol=2))
  return(data.frame(d))
}

# function that calculate F transform and statistics
Fisher_Z_test <- function(r1, r2, N1, N2){
  Z1 <- (log(1+r1) - log(1-r1))/2
  Z2 <- (log(1+r2) - log(1-r2))/2
  denomin <- sqrt(1/(N1-3) + 1/(N2-3))
  Delta_Z <- abs(Z1 - Z2)/denomin
  p_value <- 2*(1-pnorm(Delta_Z))
  return(list(statistic=Delta_Z, p=p_value))
}

# run simulations
set.seed(1)
N_sim <- 1e4
p_vals <- rep(NA, N_sim)
for(i in 1:N_sim){
  d1 <- sim_data()
  d2 <- sim_data()
  r1 <- cor(d1$X1, d1$X2)
  r2 <- cor(d2$X1, d2$X2)
  test.results <- Fisher_Z_test(r1, r2, 100, 100)
  p_vals[i] <- test.results$p
}

# false positive rate
# should be approximately equal to desired alpha
# (here alpha=0.05)
mean(p_vals<0.05)
#> [1] 0.0505
```

4.2 Polychoric and polyserial correlations

Polychoric and polyserial correlations are methods used to estimate correlations between ordinal variables or between ordinal and continuous variables. These methods assume that the ordinal variables can be considered as divisions of underlying latent variables that follow a normal distribution (similar to ordinal models discussed in a later chapter). In R, calculating polychoric correlations is straightforward with the help of the `polycor` package. You can use the `polychor` function to estimate correlations between two variables or the `hetcor` function to compute a correlation matrix for multiple variables simultaneously.

4.3 Partial correlations

Partial correlation is a statistical measure that quantifies the relationship between two variables while controlling for the effects of other variables. It assesses the strength and direction of the linear association between two variables after removing the influence of the remaining variables in the analysis.

Partial correlations are useful in situations where there are multiple variables that may be interconnected, and we want to understand the relationship between two variables while accounting for the effects of other variables. By calculating partial correlations, we can examine the direct association between two variables while holding constant the influence of other variables, thereby revealing the unique relationship between them.

Here are a few scenarios where partial correlations can be useful:

- **Confounding Control**: In observational studies, there may be confounding variables that affect both the dependent and independent variables. By computing partial correlations, we can determine the relationship between the variables of interest while controlling for potential confounders.
- **Multivariate Analysis**: When examining the relationships between multiple variables simultaneously, partial correlations can help identify direct associations between pairs of variables, accounting for the shared influence of other variables in the analysis.
- **Network Analysis**: Partial correlations are commonly employed in network analysis to uncover the underlying structure and connections between variables, such as in gene regulatory networks, social networks, or financial networks.

In R, and for simple Pearson's correlation coefficients, we can use the packages `ppcor` to calculate a matrix of partial correlation coefficients.

4.3.1 Partial correlations using Schur complement

Occasionally, we may need to calculate partial polychoric correlations (e.g. partial correlations between ordinal variables). To the best of my knowledge there isn't a simple package that allows to compute these. However, we can use an approach known as Schur complement.

The approach works as follow. Say we have computed a matrix of partial polychoric correlations using the `polycor` package; in particular Σ is the correlation matrix between the variables of interests. We further have also a set of variables we want to account for (let's call these 'confounders'), and we calculate the matrix of the correlations between them, let's call this C. Finally, we also have a matrix of correlations between the variables of interest and the confounders, here notated as B.

The partial correlation matrix can be computed as the Schur complement of C in the block matrix M, defined as $M = \begin{bmatrix} \Sigma & B \\ B^T & C \end{bmatrix}$. Note that the matrix M simply corresponds to the 'full' correlation matrix - the the matrix including correlations between *all* variables (variables of interest and confounders).

In practice, the Schur complement (partial correlation matrix) is computed as

$$\text{Partial Correlation Matrix} = \Sigma - BC^{-1}B^T.$$

Here is a simple function in R that can calculate the partial correlation matrix when some or all the variables are ordinal:

```r
partial_polychoric <- function(dX, dC, useML = TRUE) {
  # dX is a dataframe containing variables of interest
  # dC is a dataframe containing confounding variables that needs to be partialled out
  # ML estimation is recommended, but may be slow with very large datasets
  combined_data <- cbind(dX, dC)
  combined_polychoric_matrix <- hetcor(combined_data,
                                       parallel=TRUE,
                                       ML=useML,
                                       use="pairwise.complete.obs")$correlations
  cor_dX <- combined_polychoric_matrix[1:ncol(dX), 1:ncol(dX)]
  cor_dC <- combined_polychoric_matrix[1:ncol(dX), (ncol(dX) + 1):ncol(combined_data)]
  cor_yy <- combined_polychoric_matrix[(ncol(dX) + 1):ncol(combined_data), (ncol(dX) + 1):ncol(co
  inv_cor_yy <- solve(cor_yy)
  cor_dX - cor_dC %*% inv_cor_yy %*% t(cor_dC)
}
```

To estimate the p-values of partial correlation coefficients, we can use a permutation test: essentially we shuffle the order of columns in our dataset independently, then re-calculate the correlation matrix, and keep track of how often a correlation higher than the observed ones occur by chance.

The code below is an example of how such a test can be implemented in R

```r
# first, compute the observed matrix of partial correlation
sigma_polychoric <- partial_polychoric(dX, dC)

# custom function to shuffle independently the columns
# of a data.frame containing ordinal variables
shuffle_df <- function(A) {
  A_shuffle <-  as.data.frame(lapply(A, function(x) {
    if (is.ordered(x)) {
      levels <- levels(x)
      x <- as.numeric(x)
      x <- sample(x)
      x <- ordered(x, labels = levels)
    } else {
      x <- sample(x)
    }
    return(x)
  }))
  return(A_shuffle)
}

# Use replicate() to compute 1000 matrix from permutations of the datasets
n_permutations <- 1000
permuted_partial_polychoric <- replicate(n_permutations, {
  permuted_dX <- shuffle_df(dX)
  permuted_dC <- shuffle_df(dC)
  partial_polychoric(permuted_dX, permuted_dC)
})

# Compute p-values by counting how often we observe correlations
# as high as that in the observed matrix from the permuted datasets.
# The output is a mtrix of p-values
p_values <- matrix(0, ncol(dX), ncol(dX))
tot_it <- ncol(dX)*ncol(dX)
for (i in 1:ncol(dX)) {
  for (j in 1:ncol(dX)) {
    p_values[i, j] <- mean(abs(permuted_partial_polychoric[i, j, ]) >= abs(observed_pa
  }
}
```

Chapter 5

Linear models

This section will provide some worked examples of how to do analyses in R.

5.1 Simple linear regression

In this example[1] we will see how to import data into R and perform a simple linear regression analysis.

According to the standard big-bang model, the universe expands uniformly and locally, according to Hubble's law(Hubble, 1929)

$$\text{velocity} = \beta \times \text{distance}$$

where velocity and distance are the relative velocity and distance of a galaxy, respectively; and β is "Hubble's constant"[2]. Note that this is a simple linear equation, in which β indicate how much the variable velocity changes for each unitary increase in the variable distance.

According to this model β^{-1} gives the approximate age of the universe, but β is unknown and must somehow be estimated from observations of velocity and distance, made for a variety of galaxies at different distances from us. Luckily we have available data from the Hubble Space Telescope. Velocities are assessed by measuring the Doppler effect red shift in the spectrum of light that we receive from the Galaxies. Distance is estimated more indirectly, by using the discovery that in certain class of stars (Cepheids), which display fluctuations in diameter

[1]Taken from Simon Wood's book on GAM(Wood, 2017).

[2]Note the Hubble "constant" is a constant only in space, not in time

and temperature over a stable period, there is a systematic relationship between the period and their luminosity.

We can load a dataset of measurements from the Hubble Space Telescope in R using the following code

```
d <- read.table(file="https://raw.githubusercontent.com/mattelisi/RHUL-stats/main/data/
                header=T)
```

`read.table` is a generic function to import dataset in text files (e.g. .csv files) into R. We use the argument `header=T` to specify that the first line of the dataset gives the names of the columns. Note that the argument `file` here is a URL, but it could be also a path to a file in our local folder. To see the help of this function, and what other arguments and features are available type `?read.table` in the R command line.

We can use the command `str()` to examine what we have imported

```
str(d)
#> 'data.frame':    24 obs. of  3 variables:
#>  $ Galaxy  : chr  "NGC0300" "NGC0925" "NGC1326A" "NGC1365" ...
#>  $ velocity: int  133 664 1794 1594 1473 278 714 882 80 772 ...
#>  $ distance: num  2 9.16 16.14 17.95 21.88 ...
```

This tells us that our data frame has 3 variables:

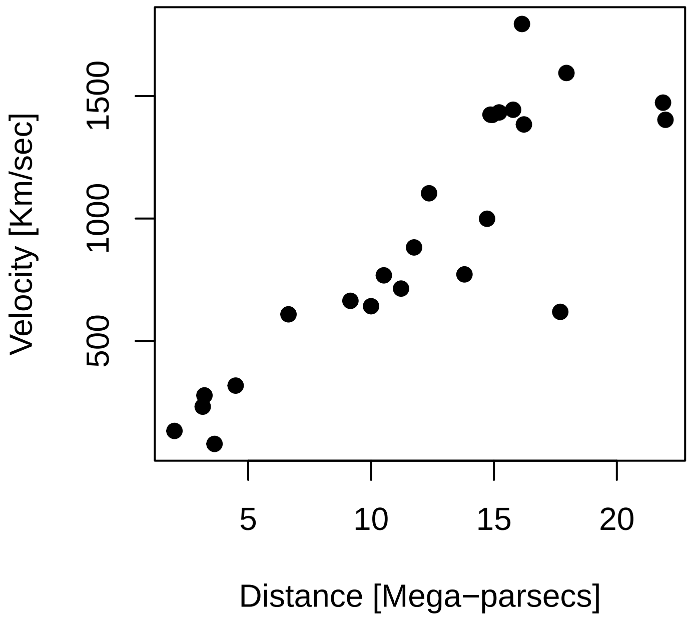
- `Galaxy`, the 'names' of the galaxies in the dataset
- `velocity`, their relative velocity in Km/sec
- `distance`, their distance expressed in Mega-parsecs[3]

We can plot[4] them using the following code:

```
plot(d$distance, # indicate which variable on X axis
     d$velocity, # indicate which variable on Y axis
     xlab="Distance [Mega-parsecs]",
     ylab="Velocity [Km/sec]",
     pch=19) # set the type of point
```

[3] 1Mega-parsec $= 3.09 \times 10^{19}$Km

[4] See `?plot` for more info about how to customize plots in R.



It is clear, from the figure, that the observed data do not follow Hubble's law exactly, but given the how these measurements were obtained (there is uncertainty about the true values of the distance and velocities) it would be surprising if they did. Given the apparent variability, what can be inferred from these data? In particular:

1. what value of β is most consistent with the data?
2. what range of β values is consistent with the data?

In order to make inferences we make some assumptions about the nature of the measurement noise. Specifically, we assume that measurements errors are well-characterized by a Gaussian distribution. This result in the following model:

$$\begin{aligned} y &= \beta x + \epsilon \\ \epsilon &\sim \mathcal{N}\left(0, \sigma_\epsilon^2\right) \end{aligned}$$

which is essentially a linear regression but without the intercept: that is, whereas normally a linear regression model include an additive term that is not multiplied with the predictor (as in $y = \beta_0 + \beta_1 x + \epsilon$), which gives the expected value of the dependent variable when all predictors are set to zero, in this case the theory tells us we can assume the intercept (the term β_0) is zero and we can ignore it.

We can fit the model with the function `lm` in R. Note that to tell R that I don't want to fit the intercept, I include in the formula the term `0 +` - this essentially tells R that the intercept term is set to zero[5]

```
hub.m <- lm(velocity ~ 0 + distance, d)
summary(hub.m)
#>
#> Call:
#> lm(formula = velocity ~ 0 + distance, data = d)
#>
#> Residuals:
#>    Min      1Q Median      3Q     Max
#> -736.5 -132.5  -19.0  172.2  558.0
#>
#> Coefficients:
#>          Estimate Std. Error t value Pr(>|t|)
#> distance   76.581      3.965   19.32 1.03e-15 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 258.9 on 23 degrees of freedom
#> Multiple R-squared:  0.9419, Adjusted R-squared:  0.9394
#> F-statistic: 373.1 on 1 and 23 DF,  p-value: 1.032e-15
```

So, based on this data, **our estimate of the Hubble constant is 76.58 with a standard error of 3.96.** The standard error - which is the standard deviation of the sampling distribution of our estimates - gives an ideas of the range of values that is compatible with our data and could be used to compute a confidence intervals (roughly, we would expect that the 'true' values of the parameters lies in the interval defined by $\pm$ 2 standard errors 95% of the times).

So, how old is the universe?

The Hubble constant estimate have units of $\frac{\text{Km/sec}}{\text{Mega-parsecs}}$. A Mega-parsecs is 3.09×10^{19}Km, so we divide our estimate of $\hat{\beta}$ by this amount. The reciprocal of $\hat{\beta}$ then gives the approximate age of the universe (in seconds). In R we can calculate it (in years) as follow

[5]A similar results would have been obtained using the notation `velocity ~ -1 + distance`.

```r
# transform in Km
hubble.const <- coef(hub.m)/(3.09 * 10^(19))

# invert to get age in seconds
age <- 1/hubble.const

# use unname() to avoid carrying over
# the label "distance" from the model
age <- unname(age)

# transform age in years
age <- age/(60^2 * 24 * 365)

# age in billion years
age/10^9
#> [1] 12.79469
```

giving an estimate of about 13 billion years.

Chapter 6

Models for count data

This section will provide some examples of models that can deal with *count* data. Typically, count data occurs when the dependent variable is the counted number of occurrences of an event - for example the number of patients arriving in an emergency department (A&E) in a given time of the day - e.g. between 10:00 and 11:00. In this case, the dependent variable (the number of patients) has several characteristics that make it unsuitable for analysis with standard linear models such as linear regression: their distribution is discrete, composed only of non-negative integers, and is often positively skewed, with many observations having a value of 0.

Another characteristic is that the variance of the observations (e.g. the variance of the number of counts across observations within the same condition) increases with their expected value (e.g. the average number of counts for that condition)[1].

6.1 Poisson model

The simplest model that account for the characteristics mentioned above is a generalized linear model that assume a dependent variable with a Poisson distribution. The Poisson distribution has a single free parameter, usually notated with λ, which gives both the expected value (the mean) and the variance of the count variable. In fact it assumes that the variance has the same value as the mean.

Formally, a Poisson model is usually formulated as follow: let $y = (y_1, \dots, y_n)$ be the dependent variable, consisting of counts (non-negative integers) and x the independent variable (predictor). Then

[1]This makes sense if you think that when the expected number of counts is very low, say ≈ 1, there cannot be many observations with very high counts - otherwise their average wouldn't be as low (recall that counts are strictly non-negative). In other words, the variance must be low when the average is also low.

$$y_i \sim \text{Poisson}\left(\lambda_i\right)$$

where[2]

$$\log(\lambda_i) = \beta_0 + \beta_1 x_i$$

or alternatively

$$\lambda_i = e^{\beta_0 + \beta_1 x_i}$$

Which indicates that we can use the exponential function (in R the function `exp()`) to calculate the predicted values of the mean (more precisely, the expected value, $\mathbb{E}(y)$) and variance ($Var(y)$) of the dependent variable. This relies on the property of the Poisson distribution[3]

$$y_i \sim \text{Poisson}\left(\lambda\right) \implies \mathbb{E}(y) = Var(y) = \lambda$$

However, in practice, data often do not conform to the constraint of having identical mean and variance. Often the observed variance of the count is higher than what predicted according to the Poisson model (we say in this case that the data is *over-dispersed*).

How does over-dispersion look like?

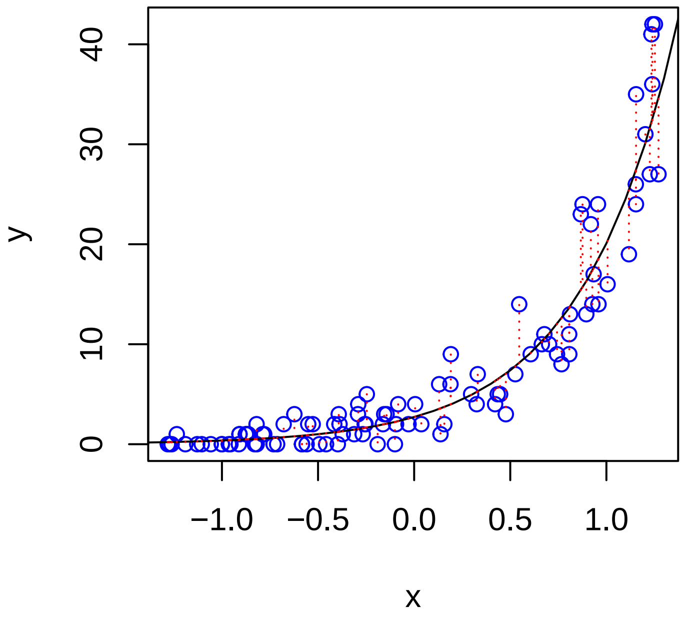
The type of data expected under a Poisson model is illustrated in the figure below, which shows 100 datapoints simulated from the model $y_i \sim \text{Poisson}\left(e^{1+2x_i}\right)$. The vertical deviations of the datapoints from the line are consistent with the property of the Poisson distribution that the variance of hte count has the same value as their expected value, formally, $Var(y) = \mathbb{E}(y)$.

```r
set.seed(2)
n <- 100
x <- runif(n, -1.3, 1.3)
a <- 1
b <- 2
linpred <- a + b*x # linear predictor part
y <- rpois(n, exp(linpred)) # simulate Poisson observations

plot(x,y,col="blue") # plot
```

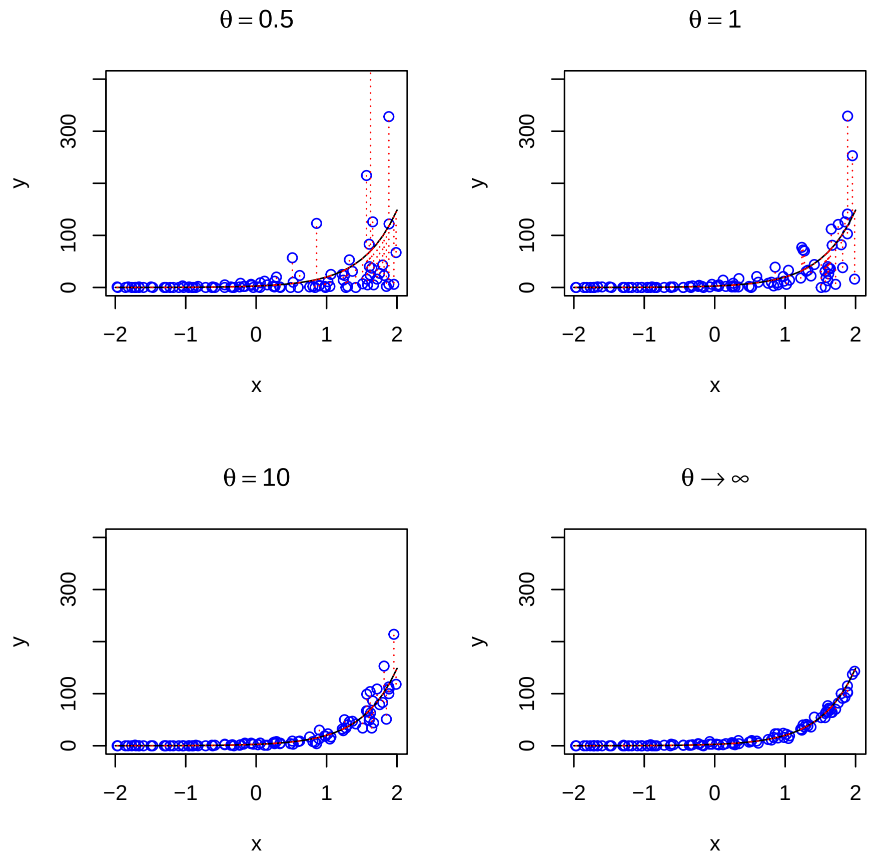
[2]This is the most common formulation, and is sometime referred to as log 'link' function, to indicate the fact that we have a function (the logarithm function) that 'link' the linear part of the model (the linear combination of the independent variables, here $\beta_0 + \beta_1 x_i$) with the parameter of the distribution of the dependent variable (here the Poisson rate parameter λ). This is a common component to all generalized linear models, for example for the logistic regression we have a 'logit' link function - the quantile function of the standard logistic distribution - that link the linear predictor part with the parameter p of the binomial distribution.

[3]The symbol $\implies$ is used to denote logical implication, e.g. $(A = B) \implies (B = A)$ - symmetry of logical equivalence.

```r
x_ <- seq(-2,2,0.1)
lines(x_, exp(a+b*x_))
segments(x,exp(a+b*x),x,y, lwd=1,lty=3, col="red")
```



We can adjust the code above to simulate the same data with some degree of over-dispersion, using a negative binomial distribution, for different values of the precision parameter theta (θ), which regulate the degree of overdispersion. Importantly, these datapoints are simulated assuming the same function fo the average (expected) number of counts (same also as the previous figure), they just differe in the amount of overdispersion relative to a Poissone model. Note that for value arbitrarily large of the precision parameter $\theta \to \infty$ (bottom-right panel) the negative binomial converges to the Poisson.



6.2 Negative binomial model

As mentioned above, data are often overdispersed relative to the Poisson (that is, their variance is larger than the mean). This is an issue because when the data are overdispersed then our results may be largely influences by few extreme datapoints. Moreover, we will have the wrong estimated about the variability of the data. To account for overdispersion we can use the negative binomial which can be seen as a generalization of the Poisson model[4].

[4]The name *negative binomial* comes from a sampling procedure that give rise to this distribution: basically the negative binomial gives the probability of the number of successes in a sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of failures occur (this pre-fixed number of failures is a parameter of the negative binomial under an alternative parametrization of this distribution).

Formally, the negative binomial model with 1 predictor x can be notated as

$$y_i \sim \text{NegBinomial}(\lambda_i, \theta)$$
$$\log(\lambda_i) = \beta_0 + \beta_1 x_i$$

The negative binomial is very similar to the Poisson - in particular it still the case that $\mathbb{E}(y) = \lambda$. However, it includes an additional precision (or "reciprocal dispersion") parameter which I referred to as θ[5]. Essentially, whereas for the Poisson we had that $Var(y) = \mathbb{E}(y)$, now we have that

$$Var(y) = \mathbb{E}(y) + \frac{\mathbb{E}(y)^2}{\theta}$$

6.3 Examples

6.3.1 Anchoring and alcohol units

To see how this would work in practice, I use negative binomial to analyse a dataset that is based on an experiment run by Ryan McKay and his MSc students[6].

The goal of the study was to use the anchoring effect to:

> *…to see if we could attenuate under-reporting of alcohol consumption (which is a medical problem). Participants in a high-anchor condition were asked "do you drink more or less than 40 units of alcohol a week", and were then asked to estimate exactly how many units they'd consumed. Those in a low anchor condition were initially asked "do you drink more or less than 4 units of alcohol a week" before giving their precise estimate, and those in a control condition just gave their precise estimate.*

In R we begin by loading the data

```
d <- read_csv("../data/nb_units.csv", show_col_types = F) # data availabel in the data folder of
d
#> # A tibble: 930 x 3
#>    gender condition units
#>    <chr>  <chr>     <dbl>
#>  1 Male   high         10
```

[5] I choose 'theta' for consistency with the R output but note that this is sometime referred to as ϕ in the literature ¯_( )_/¯

[6] Note that this is not the actual dataset that they collected but just some data that I simulated based on their research idea.

```
#>  2 Male   low          0
#>  3 Female high         1
#>  4 Female high         2
#>  5 Male   high         0
#>  6 Female low          2
#>  7 Male   control     23
#>  8 Female low          2
#>  9 Female high         1
#> 10 Male   control      4
#> # i 920 more rows
```

We can calculate the mean and variance of the number of units reported in each conditions. This reveal that the variance across participants in the number of units reported is many times higher than the mean in the number of reported units.

```
d %>%
  group_by(condition) %>%
  summarise(Mean = mean(units),
            Variance = var(units)) %>%
  knitr::kable(digits=2,
               caption="Mean and Variance of weekly units of alcohol reported.")
```

Mean and Variance of weekly units of alcohol reported.

condition

Mean

Variance

control

7.53

138.35

high

11.97

363.69

low

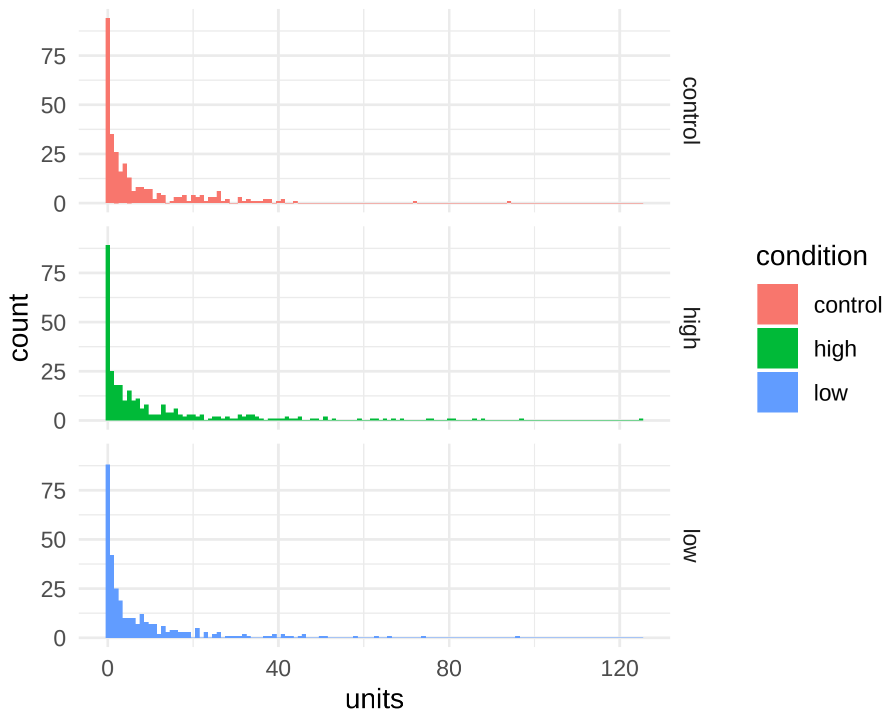
8.12

177.05

We can use ggplot2 library to visualize the distributions of reported units in each condition. We can see tha the distribution are skewed and contains many

0, which would make them unsuitable for an analysis with a linear regression model.

```
d %>%
  ggplot(aes(x=units, fill=condition)) +
  geom_histogram(binwidth=1)+
  facet_grid(condition~.) +
  theme_minimal()
```



To estimate the negative-binomial model, we can use the function `glm.nb()` from available in the `MASS` package. Our predictor condition is categorical with 3 levels and therefore it is coded internally as a set of 2 dummy variables. We can see how the contrast is set using

```
d$condition <- factor(d$condition) # tell R that this is a categorical factor
contrasts(d$condition)
#>         high low
#> control    0   0
#> high       1   0
#> low        0   1
```

This indicate that `control` is our baseline condition and the model will have 2 coefficients that code for the difference in the `high` and `low` anchoring condition relative to the control one.

Note also that for this analysis the variable `units` must contain only integer values - if participants reported non-integer values (e.g. a bottle of lager is about 1.7 units) we could divide everything by the minimum common denominator so that we end up with integer values.

The following command can be used to estimate the model and examine the results

```
library(MASS)
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>     select
```

```
nb01 <- glm.nb(units ~ condition, data = d)
summary(nb01)
#>
#> Call:
#> glm.nb(formula = units ~ condition, data = d, init.theta = 0.3852966632,
#>     link = log)
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    2.01856    0.09396  21.482  < 2e-16 ***
#> conditionhigh  0.46365    0.13218   3.508 0.000452 ***
#> conditionlow   0.07564    0.13255   0.571 0.568251
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(0.3853) family taken to be 1)
#>
#>     Null deviance: 1040.0  on 929  degrees of freedom
#> Residual deviance: 1025.2  on 927  degrees of freedom
#> AIC: 5656.8
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  0.3853
#>           Std. Err.:  0.0196
```

```
#>
#>  2 x log-likelihood:  -5648.7830
```

We can see from output that the condition `high` anchoring elicited reports with higher number of alcohol units than the `control` condition.

We can use the model to make a more precise statement about the size of the difference. We can use the value of the coefficients to calculate the predicted values of counts. The exact values of the coefficients can be accessed from the fitted model using the `$` operator

```
nb01$coefficients
#>   (Intercept) conditionhigh  conditionlow
#>    2.01856406    0.46365080    0.07563937
```

The values are combined together according to the dummy variables coding for the condition and represents the linear predictor part of the model:

$$\lambda_i = \exp\left(\beta_0 + \beta_1 \times D_{\text{high}} + \beta_2 \times D_{\text{low}}\right)$$

where I have used the notation D_{high} and D_{low} to indicate the two dummy variable, whose value is 1 for observation in the `high` and `low` conditions, respectively, and zero otherwise.

β_0 is a common notation for the intercept parameter - in this case it gives the expected number of alcohol units in the control condition (because for observations in the control condition we have that $D_{\text{high}} = D_{\text{low}} = 0$). Thus our model predict an average number of counts in the control condition of

```
exp(nb01$coefficients["(Intercept)"]) # equivalent to exp(nb01$coefficients[1])
#> (Intercept)
#>    7.527508
```

(Compare this value with the table above).

Furthermore, our models tells us also that the number of reported alcohol units increase multiplicatively in the `high` condition by a factor of

```
exp(nb01$coefficients["conditionhigh"])
#> conditionhigh
#>      1.589868
```

In fact the predicted number of counts in the `high` condition can be derived from the model as

```r
exp(nb01$coefficients["(Intercept)"])  * exp(nb01$coefficients["conditionhigh"])
#> (Intercept)
#>    11.96774
```

or equivalently

```r
exp(nb01$coefficients["(Intercept)"] + nb01$coefficients["conditionhigh"])
#> (Intercept)
#>    11.96774
```

Finally, note that we can use the `sjPlot` library to prepare a fancy version of the model output, and we can see that the multiplicative factor that describe the increase in reported units is called here an *incidence ratio*[7].

```r
library(sjPlot)
#> Install package "strengejacke" from GitHub (`devtools::install_github("strengejacke
```

```r
tab_model(nb01)
```

units

Predictors

Incidence Rate Ratios

CI

p

(Intercept)

7.53

6.29 – 9.10

<0.001

condition [high]

1.59

1.23 – 2.06

<0.001

condition [low]

1.08

[7] Although honestly I am not sure how common is this terminology

0.83 – 1.40

0.568

Observations

930

R2 Nagelkerke

0.023

6.3.1.1 Adding predictors

The dataset include also information about the gender of the participants. We may hypothesize that male participants drink more than female ones[8]. Does taking this into account improve the accuracy of our modelling?

To test this, we can estimate an additional model with also gender as predictor. We can compare this to the previous one using a *likelihood-ratio test*. This is based on a theorem which states that the difference in log-likelihood[9] between nested models is (asymptotically) distributed according to a Chi-squared distribution, therefore allowing the calculation of a p-value. In R this can be done using the function `anova()`.

First, let's fit an additional model with the extra predictor `gender`

```
nb02 <- glm.nb(units ~ condition + gender, data = d)
summary(nb02)
#>
#> Call:
#> glm.nb(formula = units ~ condition + gender, data = d, init.theta = 0.4212084665,
#>     link = log)
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   1.495941   0.105452  14.186  < 2e-16 ***
#> conditionhigh 0.426116   0.127159   3.351 0.000805 ***
#> conditionlow  0.006701   0.127720   0.052 0.958156
#> genderMale    0.909721   0.103969   8.750  < 2e-16 ***
#> ---
#> Signif. codes:
```

[8]For the sake of the example we use only 2 gender categories, but in a real study we should be mindful to include more options for non-binary / third gender participants, as McKay's students did in the real study.

[9]In this model the parameters are estimated via maximum likelihood, which amounts to choosing the values of the parameters that maximize the probability (likelihood) of the data under the model. Thus when we refer to the log-likelihood of a model we indicate the logarithm of the maximized value of the likelihood function.

```
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(0.4212) family taken to be 1)
#>
#>     Null deviance: 1115.5  on 929  degrees of freedom
#> Residual deviance: 1025.4  on 926  degrees of freedom
#> AIC: 5587.9
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  0.4212
#>           Std. Err.:  0.0219
#>
#>  2 x log-likelihood:  -5577.9060
```

This indicate that indeed male participants report on average

```
exp(nb02$coefficients["genderMale"])
#> genderMale
#>    2.48363
```

times more units of alcohol per week than females.

We can already see that the difference due to gender is significant, but nevertheless let's compare them using a likelihood ratio test.

```
anova(nb01, nb02)
#> Likelihood ratio tests of Negative Binomial Models
#>
#> Response: units
#>                Model     theta Resid. df    2 x log-lik.
#> 1          condition 0.3852967       927       -5648.783
#> 2 condition + gender 0.4212085       926       -5577.906
#>     Test    df LR stat. Pr(Chi)
#> 1
#> 2 1 vs 2     1 70.87633       0
```

Here the value of the likelihood ratio statistic is NA, 70.88 and under the null hypothesis that any improvement of model fit obtained after adding gender as predictor is due to chance is distributed as a Chi-square with 1 degree of freedom.

6.3.1.2 Plotting model fit

It's not straightforward to visualize the model fit to the data - the code below give one possibility:

```r
# here I make a new data matric for claculating the prediction of the model
nd <- expand.grid(condition=unique(d$condition),
                  units = 0:max(d$units),
                  KEEP.OUT.ATTRS = F)

# use the predict() function to calculate the predicted counts for each condition
nd$predicted_units <- predict(nb01, newdata=nd, type="response")

# here I use the dnbinom() function - which gives the probability density of
# the negative binomial - to calculate the probability of the observations under the model
nd$pred_density <- dnbinom(nd$units, mu=nd$predicted_units, size=nb01$theta)

# finally take all together and plot
d %>%
  ggplot(aes(x=units, fill=condition)) +
  geom_histogram(aes(y=..density..),binwidth=1, color="white")+
  geom_line(data=nd, aes(x=units, y=pred_density),size=1) +
  facet_grid(condition~.) +
  theme_minimal() +
  coord_cartesian(xlim=c(0,40))
#> Warning: Using `size` aesthetic for lines was deprecated in ggplot2
#> 3.4.0.
#> i Please use `linewidth` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
#> Warning: The dot-dot notation (`..density..`) was deprecated in
#> ggplot2 3.4.0.
#> i Please use `after_stat(density)` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
```

Admittedly the probability of the data under the model (the black lines) looks quite similar across the three panels, however, the model does assign higher probability to higher count values in the `high` condition compared to the other ones - we can see this by putting them together in the same panel, and by plotting the logarithm of the probability instead of the probability itself. These changes in the probability of the data may not seems large when looked at in this way, but they amount to quite substantial changes in the average number

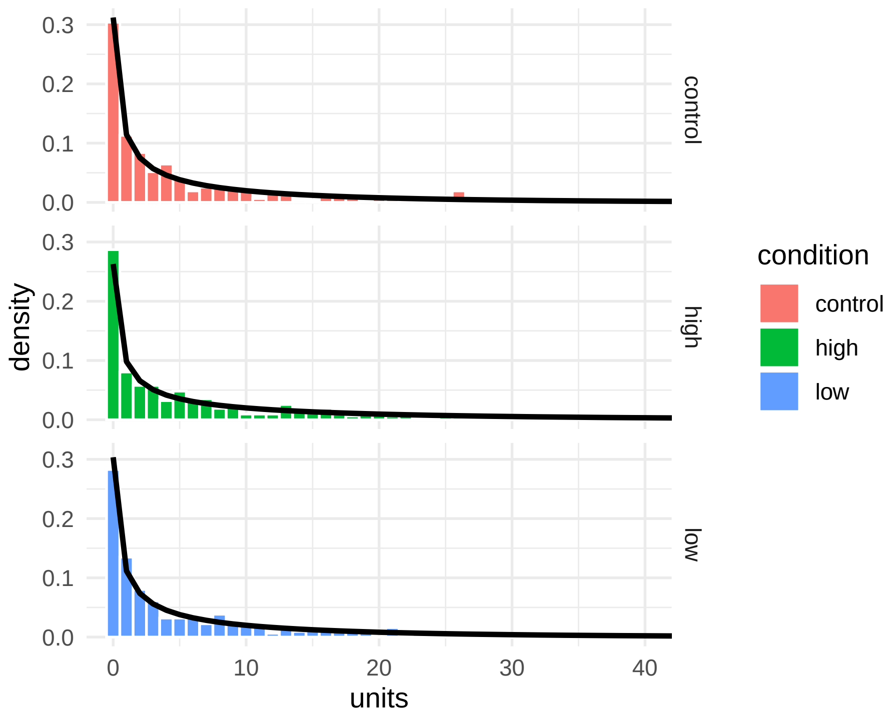
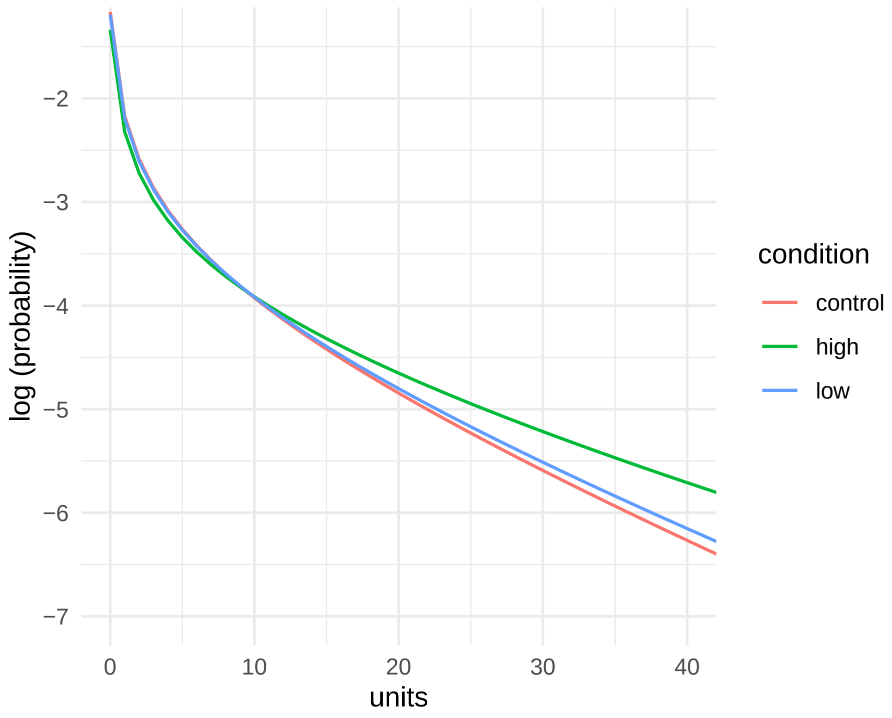


Figure 6.1: The black line represent the predicted probability of the data (note that I clipped the x-axis at 40)

of counts - recall that in the `high` condition participants reported on average nearly 1.6 times the number of alcohol units than in the control condition.

```
nd %>%
  ggplot(aes(x=units, y=log(pred_density), color=condition))+
  geom_line(size=0.6)+
  theme_minimal() +
  coord_cartesian(xlim=c(0,40),ylim=c(-7,-1.4))+
  labs(y="log (probability)")
```



Chapter 7

Models for ordinal data

Ordinal-type of variable arise often in psychology. One common example are responses to Likert scales. Although it is very common practice that these are analyzed with a linear model, it is know that this approach can lead to serious inference errors (Liddell and Kruschke, 2018). For this reason, the recommended approach is to use a model appropriate for ordinal data. Here I will describe an approach to this, using an ordered logistic regression model (also know as proportional odds model).

7.1 Ordered logistic regression

One way to think about this model is by assuming the existence of a continuous latent quantity, call it y, specified by a logistic probability density function. The latent distribution is partitioned into a series of k intervals, where k is the number of ordered choice options available to respondents, using $k+1$ latent cut-points, $c_1, \dots, c_{k+1}$. By integrating the latent density function within each interval we obtain the ordinal response probabilities $p_1, \dots, p_k$. Other choices are possible (e.g. assuming a normally distributed latent variable would yield an ordered *probit* model). Beyond mathematical convenience, one advantage of the ordered logit is that coefficient can be interpreted as ordered log-odds, implementing the proportional odds assumption (McCullagh, 1980).

Formally, the model can be notated as

$$\begin{aligned} p_k &= p\left(c_{k-1} < y \leq c_k \mid \mu\right) \\ &= \text{logit}^{-1}\left(c_k - \mu\right) - \text{logit}^{-1}\left(c_{k-1} - \mu\right) \end{aligned}$$

where

$$\text{logit}^{-1}(\alpha) = \frac{1}{1+e^{-\alpha}}$$

is the cumulative function of the logistic distribution (also known as inverse-logit), and

$$\mu = \beta_1 x_1 + \ldots + \beta_n x_n$$

is the linear part of the model (a linear combination of the n predictor variables).

This is the general approach and the formalism used - below I present few examples that illustrates how this work in practice in R.

7.1.1 Mixed-effects ordinal regression

R libraries used in this example

```
library(ggplot2)
library(ordinal)
library(tidyverse)
library(DescTools)
```

In addition to the above libraries, here I will create a handy R function that gives the probabilities of the categorical responses given a mean value of the latent quantity (indicated with μ above) and a set of cutpoints $c_1, \ldots, c_{k+1}$. This will be used both for simulating the data and for plotting the fit of the model.

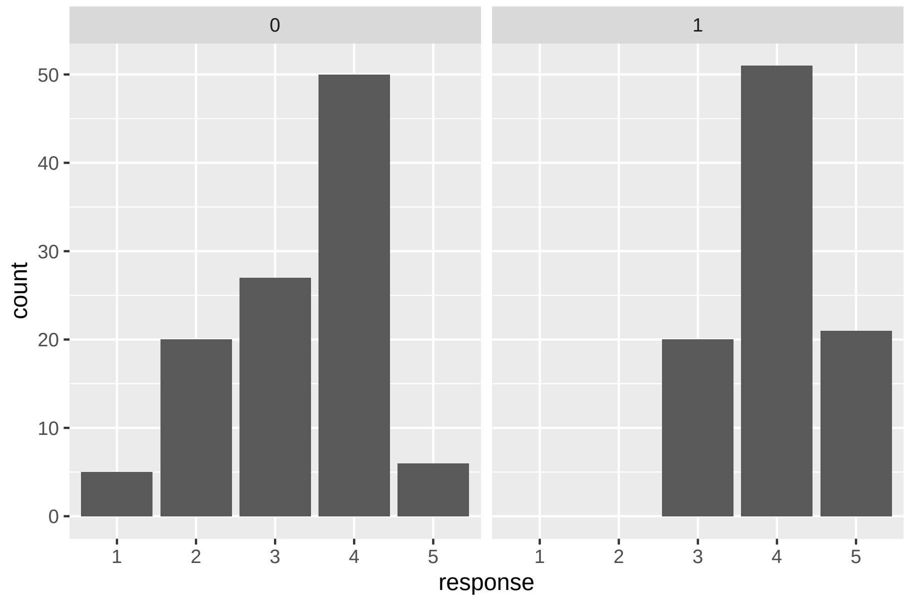
```
ordered_logistic <- function(eta, cutpoints){
  cutpoints <- c(cutpoints, Inf)
  k <- length(cutpoints)
  p <- rep(NA, k)
  p[1] <- plogis(cutpoints[1], location=eta, scale=1, lower.tail=TRUE)
  for(i in 2:k){
    p[i] <- plogis(cutpoints[i], location=eta, scale=1, lower.tail=TRUE) -
      plogis(cutpoints[i-1], location=eta, scale=1, lower.tail=TRUE)
  }
  return(p)
}
```

For this example we simulate some data. We have two predictors: `x1`, a continuous predictor that vary with each observation, and `d1` a dummy variable that indicate a categorical predictor with 2 levels (e.g. two experimental conditions). The conditions are within-subject, meaningthat each participant (identified by the variable `id`) is being tested in both conditions.

```
set.seed(5)
N <- 200
N_id <- 10
dat <- data.frame(
  id = factor(sample(1:N_id,N, replace = T)),
  d1 = rbinom(N,1,0.5), # dummy variable (0,1) indicate 2 conditions
  x1 = runif(n = N, min = 1, max = 10)
)
rfx <- rnorm(length(unique(dat$id)), mean=0, sd=5)
LP <- 0.5*dat$x1 + 2*dat$d1 + rfx[dat$id]
for(i in 1:N){
  dat$response[i] <- which(rmultinom(1,1, ordered_logistic(LP[i], c(0,2.5, 5,10)))==1)
}
dat$response <- factor(dat$response)
str(dat)
#> 'data.frame':    200 obs. of  4 variables:
#>  $ id      : Factor w/ 10 levels "1","2","3","4",..: 2 9 9 9 5 7 7 3 3 6 ...
#>  $ d1      : int  0 0 0 0 0 0 0 1 0 0 ...
#>  $ x1      : num  6.81 1.49 6.94 3.09 3.97 ...
#>  $ response: Factor w/ 5 levels "1","2","3","4",..: 4 1 3 2 2 3 3 4 3 2 ...
```

The dependent variable is categorical with 5 levels - here is a plot of the number of responses per category in the two conditions. We are interested in testing whether the distribution differ across the conditions.

```
ggplot(dat,aes(x=response))+
  geom_bar()+
  facet_grid(.~d1)
```



We use the `clmm()` function in the package `ordinal` to estimate the model. The syntax is similar to what we would use for a linear mixed effect model. Note that in the output the `Threshold coefficients` are the latent cutpoints $c_1, \dots, c_4$

```
model <- clmm(response ~ x1 + d1 + (1|id), data = dat)
summary(model)
#> Cumulative Link Mixed Model fitted with the Laplace approximation
#>
#> formula: response ~ x1 + d1 + (1 | id)
#> data:    dat
#>
#>  link  threshold nobs logLik  AIC    niter    max.grad
#>  logit flexible  200  -175.75 365.49 291(919) 6.57e-05
#>  cond.H
#>  8.6e+02
#>
#> Random effects:
#>  Groups Name        Variance Std.Dev.
#>  id     (Intercept) 4.224    2.055
#> Number of groups:  id 10
#>
#> Coefficients:
#>    Estimate Std. Error z value Pr(>|z|)
#> x1  0.55685    0.07516   7.409 1.27e-13 ***
#> d1  2.29521    0.36658   6.261 3.82e-10 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Threshold coefficients:
#>     Estimate Std. Error z value
#> 1|2  -2.1850     0.8512  -2.567
#> 2|3   0.2853     0.7664   0.372
#> 3|4   2.9453     0.8059   3.655
#> 4|5   7.8126     1.0072   7.757
```

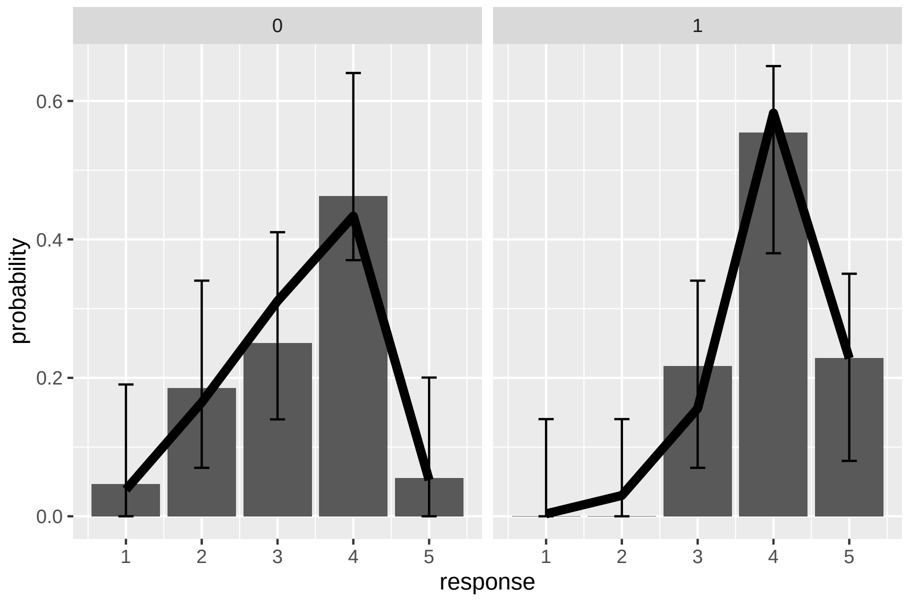
There is no function that can out-of the box calculate the predictions of the model for us, so this will need some coding. I also use the library `DescTools` to calculate simultaneous multinomial confidence intervals. In the resulting plot the black line are model fit, and bar the observed responses.

```
# pre-allocate a matrix to store model predictions
# note that these are a vector of 5 probabilities for each trial
pred_mat <- matrix(NA, nrow=N, ncol=length(unique( dat$response)))
```

```r
for(i in 1:N){

  # first calculate the linear predictor
  # by summing all variable as indicated
  # in the model formulate, weighted by the coefficients
  eta <- dat$x1[i]*model$beta['x1'] +  dat$d1[i]*model$beta['d1'] + model$ranef[dat$id[i]]
  # note that + model$ranef[dat$id[i]] adds
  # the random intercept for the subjects of observation i

  # calculate vector of predicted probabilities
  pred_mat[i,] <- ordered_logistic(eta, model$Theta)

}

# add predictions to dataset
pred_dat <- data.frame(pred_mat)
colnames(pred_dat) <- paste("resp_",1:ncol(pred_mat),sep="")
pred_dat <- cbind(dat, pred_dat)

# in order to visalize the predictions,
# we first average them for each condition
pred_dat %>%
  pivot_longer(cols=starts_with("resp_"),
               names_prefix="resp_",
               values_to = "prob",
               names_to ="response_category") %>%
  group_by(d1, response_category) %>%
  summarise(prob = mean(prob),
            n=sum(response==response_category)) %>%
  group_by(d1) %>%
  mutate(prop_obs = n/sum(n),
    response=as.numeric(response_category)) -> pred_d1
#> `summarise()` has grouped output by 'd1'. You can override
#> using the `.groups` argument.
```

```r
# cimpute the multinomial interval
pred_d1$CI_lb <- MultinomCI(pred_d1$n)[,"lwr.ci"] *2
pred_d1$CI_ub <- MultinomCI(pred_d1$n)[,"upr.ci"] *2
# note that I multiply for 2 because in the plot each condition
# will be in a different panel and the probability will sum to 1 in each panel

# visualize (aggregated) ordinal response & prediction
# the black line are the predictions of the model
```

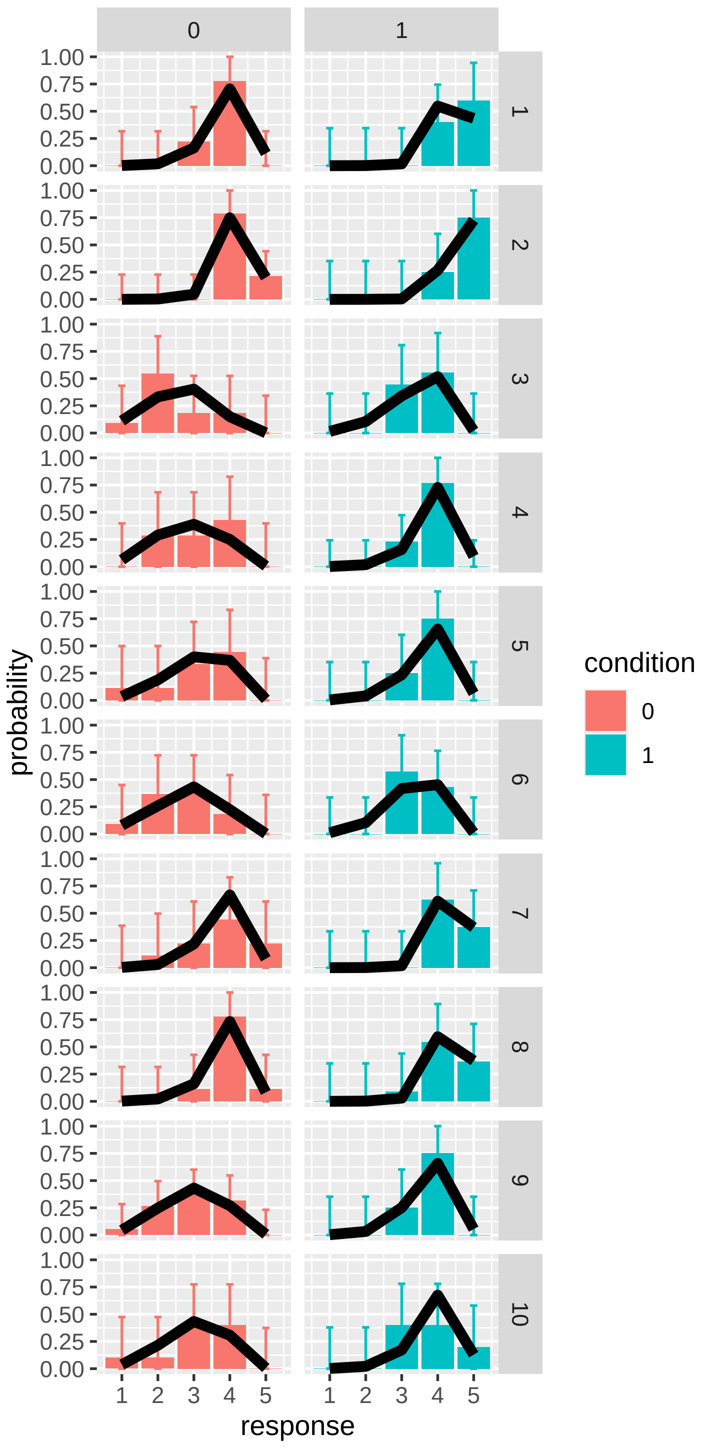
```
ggplot(pred_d1,aes(x=response, y=prop_obs))+
  geom_col()+
  geom_errorbar(data=pred_d1, aes(ymin=CI_lb, ymax=CI_ub),width=0.2)+
  facet_grid(.~d1)+
  geom_line(data=pred_d1, aes(y=prob), size=2)+
  labs(y="probability")
#> Warning: Using `size` aesthetic for lines was deprecated in ggplot2
#> 3.4.0.
#> i Please use `linewidth` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
```



We can repeat the same process also for calculating predictions for individual participants

```
# split by ID
pred_dat %>%
  pivot_longer(cols=starts_with("resp_"),
               names_prefix="resp_",
               values_to = "prob",
               names_to ="response_category") %>%
  group_by(id, d1, response_category) %>%
  summarise(prob = mean(prob),
            n=sum(response==response_category)) %>%
  group_by(d1, id) %>%
  mutate(prop_obs = n/sum(n),
         response=as.numeric(response_category))  -> pred_d1
#> `summarise()` has grouped output by 'id', 'd1'. You can
#> override using the `.groups` argument.
```

```
# calculate multinomial CI
# we do a loop over all participants and conditions
pred_d1$CI_lb <- NA
pred_d1$CI_ub <- NA
for(i in unique(pred_d1$id)){
  for(cond in c(0,1)){
    pred_d1$CI_lb[pred_d1$id==i & pred_d1$d1==cond] <- DescTools::MultinomCI(pred_d1$n[pred_d1$ic
    pred_d1$CI_ub[pred_d1$id==i & pred_d1$d1==cond] <- DescTools::MultinomCI(pred_d1$n[pred_d1$ic
  }
}

pred_d1 %>%
  mutate(condition = factor(d1)) %>%
  ggplot(aes(x=response, y=prop_obs, fill=condition))+
    geom_col()+
    geom_errorbar(aes(ymin=CI_lb, ymax=CI_ub, color=condition), width=0.2)+
    facet_grid(id~d1)+
    geom_line(aes(y=prob), size=2)+
    labs(y="probability")
```


0
1
1
2
3
4
5
6
7
8
9
10
1.00
0.75
0.50
0.25
0.00
probability
response
1 2 3 4 5
condition
0
1

Chapter 8

Meta-analyses

For running meta-analyses, we recommend the `metafor` package for R (see link 1, link 2).

A comprehensive, hands-on guide on how to use this package is provided in the book by Harrer and colleagues (Harrer et al., 2021), freely available at this link.

An alternative to the `metafor` package is to Bayesian multilevel modelling (also discussed in the book by Harrer and colleagues). A more technical discussion of Bayesian multilevel modelling for meta-analyes is provided in this paper by Williams, Rast and Bürkner (Williams et al., 2018).

Note: the slides of a workshop on meta-analyses using `metafor` package are included in the workshop section of this website 13

Chapter 9

Missing data

9.1 Types of missing data

Following the work of Rubin(RUBIN, 1976), missing data are typically grouped in 3 categories:

- Missing completely at random (**MCAR**). This assumes that the probability of being missing is the same for all cases; this implies that the mechanisms that causes missingness is not related in any way to the data. For example, say, there's a known unpredictable error on the server side that prevented recording some responses for some respondents to a survey. As the missingness is entirely independent on the respondents' characteristics, this would be MCAR. When the data are MCAR we can ignore a lot of the complexities and just do a *complete-case* analysis (that is, simply exclude incomplete observations from the dataset). A part from possible loss of information, doing a complete case analysis should not introduce bias in the results. In practice, however, it is difficult to establish whether the data are truly MCAR. Ideally, to argue that data are MCAR, one should have a good idea of the mechanisms that caused missigness (more on this below). Formally, data is MCAR if

  $$\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi) = \Pr(R = 0|\psi)$$

  where R is an indicator variable that is set to 0 for missing data and 1 otherwise; $Y_{\text{obs}}, Y_{\text{mis}}$ indicate observed and missing data, respectively; and ψ is simply a parameter that determine the overall (fixed) probability of being missing.

- Missing at random (**MAR**). A less strong assumption about missingness is that it is systematically related to the observed but not the unobserved data. For example, data are MAR if in a study male respondents are less likely to complete a survey on depression severity than female respondents - that is, the probability of reaching the end of the survey is related to their sex (fully observed) but not the severity of their symptoms. Formally, data is MAR if

$$\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi) = \Pr(R = 0|Y_{\text{obs}}, \psi)$$

When data are missing at random (MAR) the results of complete case analyses may be biased and a common approach to deal with this is to use imputation. Stef van Buuren has a freely available online book on this topic(van Buuren, 2018). Among other things, it illustrates how to do multiple imputation in R with examples.

- Missing not at random (**MNAR**). This means that the probability of being missing varies for reasons that are unknown to us, and may depends on the missing values themselves. Formally this means that $\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi)$ does not simplify in any way. This case is the most hard to handle: a complete case analyses may or may not be biased, but there is no way of knowing it and we may have to find more information about what caused missingness.

9.2 Deciding whether the data are MCAR

As MCAR is the only scenario in which it is safe to do a complete case analysis, it would seem useful to have way to test this assumption. Some approaches have been proposed to test whether the data are MCAR, but they are not widely used and it's not clear how useful they are in practice. For example one could run a logistic regression with "missingness" as dependent variable (e.g. an indicator variable set to 1 if data is missing and 0 otherwise), and all other variables as predictors - if the data are MCAR then none of the predictors should predict missingness. A popular alternative, implemented in several software packages is Little's test(Little, 1988).

Technically, these approaches can help determine whether the missingness depends on some observed variables (that is, if they are MAR), but strictly speaking cannot exclude missingness due to unobserved variables (MNAR scenario). Nevertheless, if one has good reasons to believe that the data are MCAR, and want to add some statistical test that corroborate this assumption, these could be reasonable tests to do. However, it remains important to also discuss openly possible reasons and mechanisms of missingness, and explain why we deem it a priori plausible that the data are MCAR. In fact, *statistical tests alone cannot tell whether data are missing completely at random.* The terms MCAR, MAR and MNAR refers to the *causal* mechanisms that is responsible for missing data and, strictly speaking, causal claims cannot be decided uniquely on the basis of a simple statistical test. If the data "pass" the test it would provide some additional support to the assumption that they are MCAR, but in and of itself the test alone does not fully satisfy the assumptions of MCAR. To see why note that MCAR (as formally defined above) assumes also that there should be no relationship between the missingness on a particular variable and the values of that same variable: but since this is a question about what is missing from the data, it cannot be tested with any quantitative analysis of the available data. Finally, it should be added that as these are null-hypothesis significance test, a failure to reject the null hypothesis does not, in and of itself, provide evidence for the null hypothesis (that the data are MCAR). It may be also that we don't have enough power to reliably detect the pattern in the missingness.

9.3 Causal analysis and Bayesian imputation

The best and most principled approach to deal with missingness (at least in my opinion) is to think hard about the causal mechanisms that may determine missingness, and use our assumption about the causal mechanisms to perform a full Bayesian imputation (that is, treating the missing data as parameter and estimating them).

I plan to create and include here a worked example of how to do this; in the meantime interested readers are referred to Chapter 15 (in particular section 15.2) of the excellent book by Richard McElreath *Statistical Rethinking*(McElreath, 2020) which present a very accessible worked example of how to do this in R.

Chapter 10

Multilevel Bayesian models

10.1 Overdispersed poisson GLMM example: police stops in NYC

In this section, we analyze an example dataset of police stop-and-frisk data, referred to in `police_BDA3rd_p435.pdf` available at this repository.
The data contains counts of stops by precinct, ethnicity, and crime type, along with population and DCJS arrests.
We'll fit an **overdispersed Poisson** regression with random intercepts by precinct and an observation-level random effect, following Gelman & Hill's approach.

10.1.1 1. Load and Explore the Data

```r
# Read the data from a text file
d <- read_delim("../data/police_stops.txt", delim = " ", col_names = TRUE)

# Examine the structure
str(d)
#> spc_tbl_ [900 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
#>  $ stops       : num [1:900] 75 36 74 17 37 39 23 3 26 32 ...
#>  $ pop         : num [1:900] 1720 1720 1720 1720 1368 ...
#>  $ past.arrests: num [1:900] 191 57 599 133 62 27 149 57 135 16 ...
#>  $ precinct    : num [1:900] 1 1 1 1 1 1 1 1 1 1 ...
#>  $ eth         : num [1:900] 1 1 1 1 2 2 2 2 3 3 ...
#>  $ crime       : num [1:900] 1 2 3 4 1 2 3 4 1 2 ...
#>  - attr(*, "spec")=
#>   .. cols(
#>   ..   stops = col_double(),
#>   ..   pop = col_double(),
#>   ..   past.arrests = col_double(),
#>   ..   precinct = col_double(),
#>   ..   eth = col_double(),
#>   ..   crime = col_double()
#>   .. )
#>  - attr(*, "problems")=<externalptr>
```

```
# Note:
# precincts are numbered 1-75
# ethnicity 1=black, 2=hispanic, 3=white
# crime type 1=violent, 2=weapons, 3=property, 4=drug
```

10.1.2 2. Summary Statistics

We'll group by ethnicity to see how many stops, how many arrests, and what fraction of population each group represents across all precincts/crime categories.

```
d_summaries <- d %>%
  mutate(eth = case_when(
    eth == 1 ~ "black",
    eth == 2 ~ "hispanic",
    eth == 3 ~ "white"
  )) %>%
  group_by(eth) %>%
  summarise(
    pop           = sum(pop),
    total_stops   = sum(stops),
    total_arrests = sum(past.arrests),
    .groups = "drop"
  ) %>%
  mutate(
    prop_of_all_stops = total_stops / sum(total_stops),
    pop_fraction      = pop / sum(pop)
  )

d_summaries
#> # A tibble: 3 x 6
#>   eth         pop total_stops total_arrests prop_of_all_stops
#>   <chr>     <dbl>       <dbl>         <dbl>             <dbl>
#> 1 black    7.46e6        69823         125719                0.531
#> 2 hispan~ 6.93e6        44623          74898                0.340
#> 3 white    1.27e7        16974          35922                0.129
#> # i 1 more variable: pop_fraction <dbl>
```

10.1.3 3. Model Definition

We define the overdispersed Poisson model:

$$y_{ep} \sim \text{Poisson}\big(\underbrace{\text{past.arrests}_{ep} \times \tfrac{15}{12}}_{\text{offset term}} \times e^{\alpha_e+\beta_p+\epsilon_{ep}}\big),$$

where:

- y_{ep} = number of stops for ethnicity e in precinct p.
- α_e = fixed effect for each ethnicity ($e = 1, 2, 3$).
- β_p = random effect for precinct p, with $\beta_p \sim N(0, \sigma_\beta^2)$.
- ϵ_{ep} = observation-level overdispersion, $\epsilon_{ep} \sim N(0, \sigma_\epsilon^2)$.

Hence, the log-rate is: $\log(\lambda_{ep}) = \alpha_e + \beta_p + \epsilon_{ep}$, and we multiply λ_{ep} by the offset $\text{past.arrests}_{ep} \times \frac{15}{12}$.

10.1.3.1 Non-Centered Parametrization

We call β_p *non-centered* if we write:

$$\beta_p = \sigma_\beta\, \beta_{\text{raw},p}, \quad \beta_{\text{raw},p} \sim N(0, 1),$$

instead of sampling β_p directly from $N(0, \sigma_\beta^2)$. This often improves sampling efficiency in hierarchical models. The same logic applies to ϵ_{ep}.

10.1.4 4. Stan Code

Below is the Stan code (which you can save in a file named `overdispersed_poisson.stan`):

```
data {
  int<lower=1> N;           // total # of (eth, precinct) data points
  int<lower=1> n_eth;       // number of ethnicity categories, e.g. 3
  int<lower=1> n_prec;      // number of precincts

  int<lower=0> y[N];        // outcome counts y_{ep}
  vector[N] past_arrests;   // baseline/reference rate

  int<lower=1, upper=n_eth> eth[N];       // ethnicity ID for each row
  int<lower=1, upper=n_prec> precinct[N]; // precinct ID for each row
}

parameters {
  // Ethnicity effects (fixed)
  vector[n_eth] alpha;

  // Random intercepts for precinct
  real<lower=0> sigma_beta;
  vector[n_prec] beta_raw;   // non-centered param for precinct

  // Overdispersion
  real<lower=0> sigma_eps;
  vector[N] eps_raw;         // non-centered param for each (e,p) observation
}

transformed parameters {
  vector[n_prec] beta;
  beta = sigma_beta * beta_raw;

  vector[N] eps;
  eps = sigma_eps * eps_raw;
}

model {
  // Priors (adjust as appropriate)
  alpha ~ normal(0, 5);
  sigma_beta ~ exponential(1);
  beta_raw ~ normal(0, 1);

  sigma_eps ~ exponential(1);
  eps_raw ~ normal(0, 1);

  // Likelihood
  for (i in 1:N) {
    // lambda = alpha[ eth[i] ] + beta[ precinct[i] ] + eps[i]
    // Multiply by (past_arrests[i] * 15/12) to get Poisson mean
    y[i] ~ poisson(past_arrests[i] * (15.0 / 12.0) * exp(alpha[eth[i]] + beta[precinct[i]] + eps[
  }
}
```

10.1.5 5. Prior Predictive Check

Before fitting the model, it's good practice to simulate data under the prior to see if the implied distribution is reasonable. For brevity, here's a minimal example:

```
# Suppose we do a quick simulation from the prior in R:
# (We won't run Stan's generated quantities block, but just do a rough check.)

set.seed(123)
N_sim <- 10
sigma_beta_sim <- rexp(1, 1)
sigma_eps_sim <- rexp(1, 1)
alpha_sim <- rnorm(3, 0, 5) # 3 ethnicity groups

beta_raw_sim <- rnorm(75, 0, 1)
beta_sim <- sigma_beta_sim * beta_raw_sim

eps_raw_sim <- rnorm(N_sim, 0, 1)
eps_sim <- sigma_eps_sim * eps_raw_sim

cat("Simulated sigma_beta =", sigma_beta_sim, "\n")
#> Simulated sigma_beta = 0.8434573
```

```
cat("Simulated sigma_eps  =", sigma_eps_sim, "\n")
#> Simulated sigma_eps  = 0.5766103
```

```
cat("Simulated alpha =", alpha_sim, "\n")
#> Simulated alpha = -1.150887 7.793542 0.352542
```

We could draw some random `past_arrests` values (e.g., 1–50) and see how big the Poisson means might get. If these are unreasonably large or small, we might tighten the priors.

10.1.6 6. Fit the Model

Important: If `past.arrests == 0` for some rows but `stops` are nonzero, we need to avoid a zero Poisson mean. One quick fix is to add a small constant (e.g. 0.5) to `past.arrests`.

```r
# Correction for zero arrests
d$past.arrests <- ifelse(d$past.arrests == 0, 0.5, d$past.arrests)
```

10.1.6.1 Prepare data for Stan

```r
stan_data <- list(
  N          = nrow(d),
  n_eth      = length(unique(d$eth)),
  n_prec     = length(unique(d$precinct)),
  y          = d$stops,
  eth        = d$eth,
  past_arrests  = d$past.arrests,
  precinct   = d$precinct
)
```

10.1.6.2 Compile and Run Stan

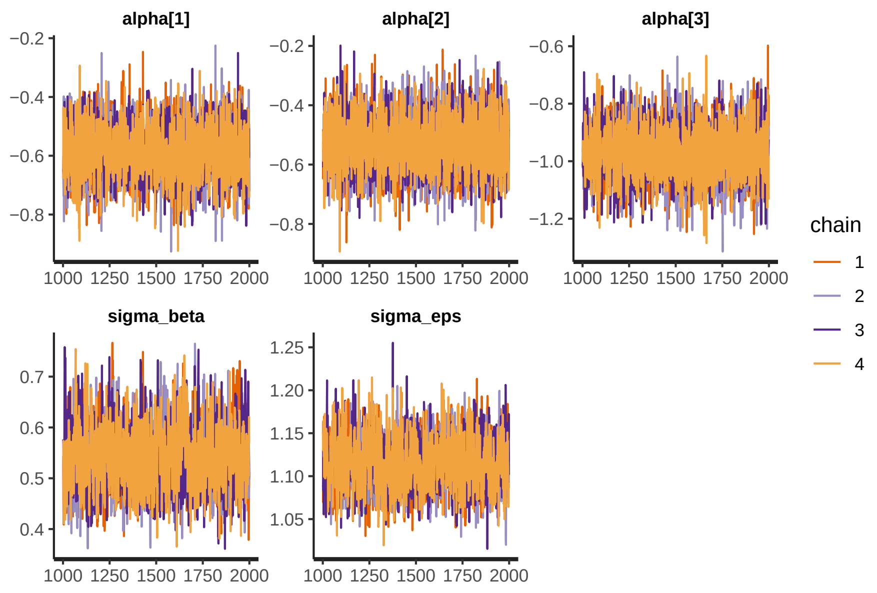
```r
# Make sure you have the Stan model saved as "overdispersed_poisson.stan".
# Then run:
fit <- stan(
  file = "../models/overdispersed_poisson.stan",
  data = stan_data,
  iter = 2000,
  chains = 4,
  cores = 4
)
```

```r
print(fit, pars = c("alpha", "sigma_beta", "sigma_eps"))
#> Inference for Stan model: anon_model.
#> 4 chains, each with iter=2000; warmup=1000; thin=1;
#> post-warmup draws per chain=1000, total post-warmup draws=4000.
#>
#>             mean se_mean   sd  2.5%   25%   50%   75% 97.5%
#> alpha[1]   -0.58       0 0.09 -0.76 -0.64 -0.58 -0.52 -0.41
#> alpha[2]   -0.53       0 0.09 -0.71 -0.59 -0.53 -0.47 -0.34
#> alpha[3]   -0.97       0 0.09 -1.15 -1.03 -0.97 -0.91 -0.80
#> sigma_beta  0.54       0 0.06  0.43  0.50  0.54  0.58  0.68
#> sigma_eps   1.11       0 0.03  1.06  1.10  1.11  1.13  1.17
#>            n_eff Rhat
#> alpha[1]    7809    1
#> alpha[2]    6275    1
#> alpha[3]     7447    1
#> sigma_beta   1065    1
#> sigma_eps    1688    1
#>
#> Samples were drawn using NUTS(diag_e) at Tue Mar 25 16:31:01 2025.
#> For each parameter, n_eff is a crude measure of effective sample size,
#> and Rhat is the potential scale reduction factor on split chains (at
#> convergence, Rhat=1).
```

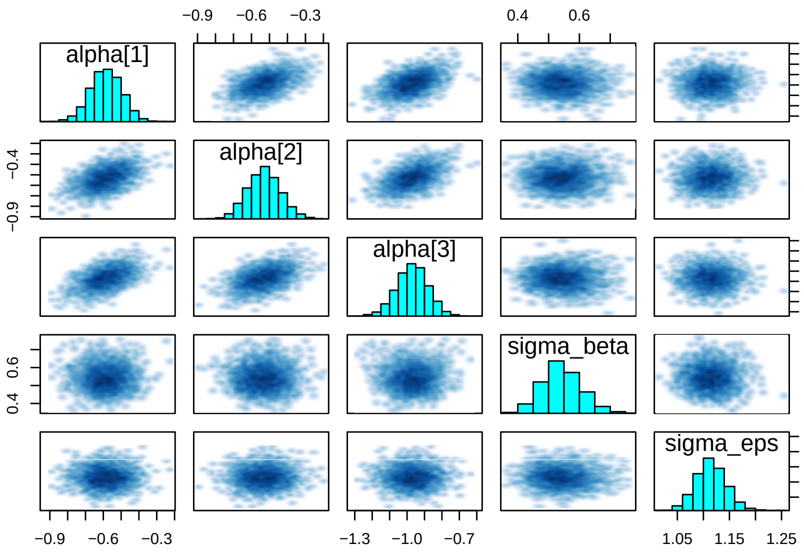
Here, `alpha` are the ethnic-group log-effects, `sigma_beta` is the precinct-level standard deviation, and `sigma_eps` is the overdispersion. You can also examine random intercepts `beta` or the `eps` in detail.

10.1.7 7. Diagnostic Checks

```
# Traceplots
traceplot(fit, pars = c("alpha", "sigma_beta", "sigma_eps"))
```



```
# Pairs plot for some key parameters
pairs(fit, pars = c("alpha[1]", "alpha[2]", "alpha[3]", "sigma_beta", "sigma_eps"))
```



Check for good mixing (no major issues in traceplots) and no suspicious funnel shapes in pairs plots.

10.1.8 8. Posterior Analysis

We use **tidybayes** to extract samples and summarize. Suppose we want to look at the exponentiated ethnic effects (relative stops vs. arrests).

```
library(tidybayes)

posterior <- fit %>%
  spread_draws(alpha[e]) %>%
  mutate(rate = exp(alpha))   # exponentiate to interpret as a multiplicative factor

# Summarize the rate by ethnicity
posterior %>%
  group_by(e) %>%
  mean_hdi(rate, .width = 0.95)
#> # A tibble: 3 x 7
#>       e  rate .lower .upper .width .point .interval
#>   <int> <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
#> 1     1 0.560  0.467  0.660   0.95 mean   hdi
#> 2     2 0.591  0.488  0.697   0.95 mean   hdi
#> 3     3 0.381  0.316  0.449   0.95 mean   hdi
```

If `e=1` = black, `e=2` = hispanic, `e=3` = white, these give the average ratio of stops to arrests (relative to some baseline) for each ethnicity, controlling for precinct and overdispersion.

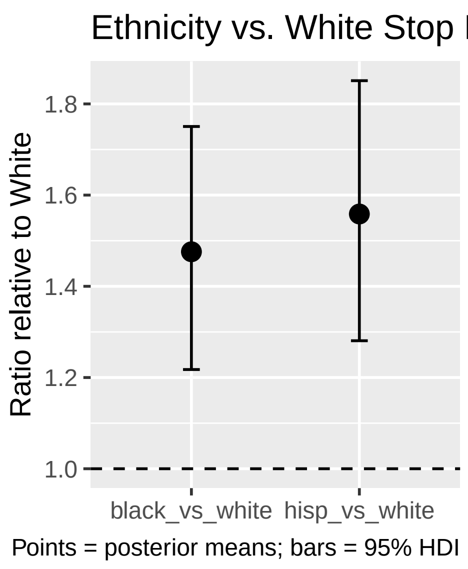
You might then compare black vs. white, etc., by calculating differences or ratios in the posterior draws.

```r
posterior_ratios <- posterior %>%
  pivot_wider(
    id_cols = c(.chain, .iteration, .draw),  # these 3 columns identify each draw
    names_from = e,                          # which column to pivot on
    values_from = rate,                      # which column holds the values
    names_prefix = "eth_"
  ) %>%
  mutate(
    black_vs_white = eth_1 / eth_3,
    hisp_vs_white  = eth_2 / eth_3
  ) %>%
   # Pivot longer so each ratio is in its own row
  pivot_longer(
    cols = c(black_vs_white, hisp_vs_white),
    names_to = "contrast",
    values_to = "ratio"
  ) %>%
  # Summarize with mean_hdi -> returns columns .mean, .lower, .upper
  group_by(contrast) %>%
  mean_hdi(ratio, .width = 0.95)

posterior_ratios
#> # A tibble: 2 x 7
#>   contrast       ratio .lower .upper .width .point .interval
#>   <chr>          <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
#> 1 black_vs_white  1.48   1.22   1.75   0.95 mean   hdi
#> 2 hisp_vs_white   1.56   1.28   1.85   0.95 mean   hdi
```

This gives an estimate of how many times more likely black (or hispanic) stops are relative to white stops, after controlling for arrests, precinct differences, and overdispersion.

We can use this for plotting e.g.:

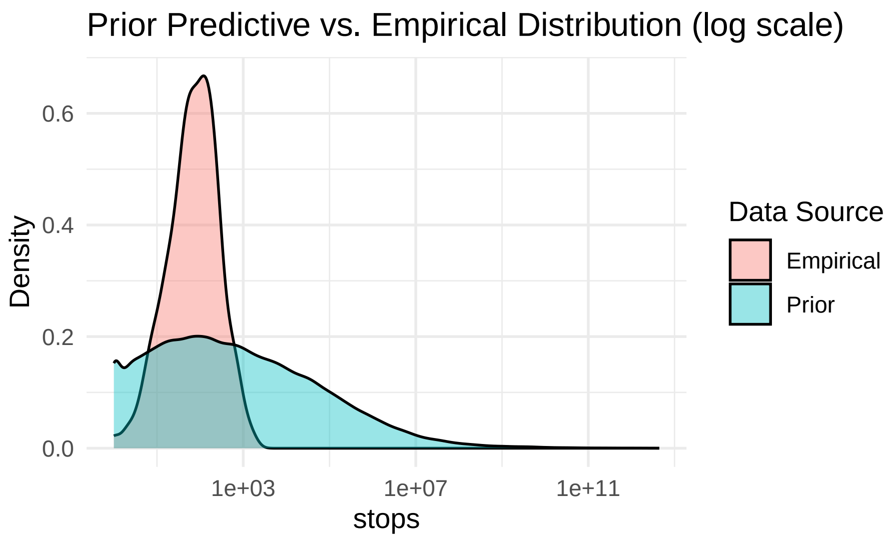
```r
ggplot(posterior_ratios, aes(x = contrast, y = ratio)) +
  geom_point(size=3) +
  geom_hline(yintercept=1, lty=2)+
  geom_errorbar(aes(ymin = .lower, ymax = .upper), width = 0.1) +
  labs(
    x = NULL,
    y = "Ratio relative to White",
    title = "Ethnicity vs. White Stop Ratios",
    caption = "Points = posterior means; bars = 95% HDI"
  )
```



10.1.9 Prior predictive check

```
N <- nrow(d)
n_eth <- length(unique(d$eth))
n_prec <- length(unique(d$precinct))

# Decide how many draws from the prior to simulate
n_sims <- 200

# We'll store all simulated Y values in a single big vector
# so we can compare them to the observed distribution
all_prior_stops <- numeric(0)

set.seed(101)  # reproducibility

for (s in 1:n_sims) {

  # 1) Sample the parameters from their priors
```

```r
  alpha_draw <- rnorm(n_eth, mean = 0, sd = 5) # alpha: one for each ethnicity
  sigma_beta_draw <- rexp(1, rate = 1)
  beta_raw_draw <- rnorm(n_prec, 0, 1)
  beta_draw <- sigma_beta_draw * beta_raw_draw

  sigma_eps_draw <- rexp(1, rate = 1)          # Overdispersion
  eps_raw_draw <- rnorm(N, 0, 1)
  eps_draw <- sigma_eps_draw * eps_raw_draw

  # 2) Generate y (stops) from Poisson for each observation i

  past_arrests_offset <- ifelse(d$past.arrests == 0, 0.5, d$past.arrests)

  # compute the linear predictor for each row i
  lambda_vec <- numeric(N)
  for (i in seq_len(N)) {
    e <- d$eth[i]
    p <- d$precinct[i]
    lambda_vec[i] <- alpha_draw[e] + beta_draw[p] + eps_draw[i]
  }

  # Poisson means
  mu_vec <- past_arrests_offset * (15/12) * exp(lambda_vec)

  # sample from Poisson
  y_sim <- rpois(N, mu_vec)

  # 3) Store in a big vector for later comparison
  all_prior_stops <- c(all_prior_stops, y_sim)
}

# Now we have n_sims * N simulated 'stops' from the prior
# Compare with the empirical distribution:
observed_stops <- d$stops

df_plot <- tibble(
  value = c((observed_stops ), (all_prior_stops )),
  type  = rep(c("Empirical", "Prior"), c(length(observed_stops), length(all_prior_stops)))
)

# Plot densities
ggplot(df_plot, aes(x = value, fill = type)) +
  geom_density(alpha = 0.4) +
  labs(
    x = "stops",
    y = "Density",
    title = "Prior Predictive vs. Empirical Distribution (log scale)",
    fill = "Data Source"
  ) +
  theme_minimal()+
  scale_x_log10()
```



10.1.10 References

- Gelman, Andrew, and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. 2007.

Chapter 11

Signal Detection Theory

Signal Detection Theory (hereafter abbreviated as SDT) is probably the most important and influential framework for modelling perceptual decisions in forced-choice tasks, and has wide applicability also beyond purely perceptual decision tasks. Here I review and derive some fundamental concepts of equal-variance and unequal variance signal detection theory, and present some R code to simulate the confidence of an optimal observer/decision-maker.

11.1 Equal-variance Gaussian SDT

SDT relies on the idea that information available to the observer / decision-maker can be modeled as a single random variable on a latent decision space. SDT is typically (but not only) applied to detection tasks: each trials a stimulus is presented, consisting of some background noise with or without a signal. The observer is tasked with deciding whether the signal was present or absent (thus there are 2 possible responses, *yes* and *no*). SDT assumes that in each trial the observer makes their decision on the basis of a random variable, call it X, which may be drawn from the signal distribution or from the noise distribution.

In the simplest case, the signal distribution is a Gaussian with variance $\sigma^2 = 1$ and mean $d' > 0$.

$$\begin{aligned} f_S(x) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-d')^2}{2\sigma^2}} \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-d')^2}{2}} \end{aligned}$$

And the noise distribution is a second normal distribution with mean 0 and variance $\sigma^2 = 1$

$$f_N(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Note that the *prior probability* of signal and noise may not be equal. Let's define the probability of a signal-present trial as $p(S) = \alpha$; we have thus that $p(N) = 1 - p(S) = 1 - \alpha$.

11.1.1 Optimal decision rule

The optimal way to decide whether a particular value of x was drawn from a signal or noise distribution is by using a likelihood-ratio, that is one should responde "yes" whenever

$$\begin{aligned} \frac{f_S(x)\, p(S)}{f_N(x)\, p(N)} &\geq 1 \\ \frac{f_S(x)\, \alpha}{f_N(x)\, (1-\alpha)} &\geq 1 \end{aligned}$$

With some algebraic manipulations, it can be shown that the likelihood ratio decision rule amounts to comparing the value of x to a criterion c:

$$\begin{aligned} \frac{f_S(x)\alpha}{f_N(x)(1-\alpha)} &\geq 1 \\ \frac{f_S(x)}{f_N(x)} &\geq \frac{1-\alpha}{\alpha} \\ \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{(x-d')^2}{2}}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}} &\geq \frac{1-\alpha}{\alpha} \\ \frac{e^{-\frac{(x-d')^2}{2}}}{e^{-\frac{x^2}{2}}} &\geq \frac{1-\alpha}{\alpha} \end{aligned}$$

taking the log of both sides

$$
\begin{aligned}
\log\left(\frac{e^{-\frac{(x-d')^2}{2}}}{e^{-\frac{x^2}{2}}}\right) &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
\log\left(e^{-\frac{(x-d')^2}{2}}\right) - \log\left(e^{-\frac{x^2}{2}}\right) &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
-\frac{(x-d')^2}{2} + \frac{x^2}{2} &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
\frac{-x^2-(d')^2+2d'x+x^2}{2} &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
\frac{-(d')^2+2d'}{2}x &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
d'x - \frac{(d')^2}{2} &\geq \log\left(\frac{1-\alpha}{\alpha}\right)\\
d'x &\geq \log\left(\frac{1-\alpha}{\alpha}\right) + \frac{(d')^2}{2}\\
x &\geq \frac{1}{d'}\log\left(\frac{1-\alpha}{\alpha}\right) + \frac{d'}{2}
\end{aligned}
$$

The optimal criterion is thus found as $c = \frac{1}{d'}\log\left(\frac{1-\alpha}{\alpha}\right) + \frac{d'}{2}$. Whenever the x exceed the criterion, $x \geq c$, the observer should respond "signal present" and respond "signal absent" otherwise.

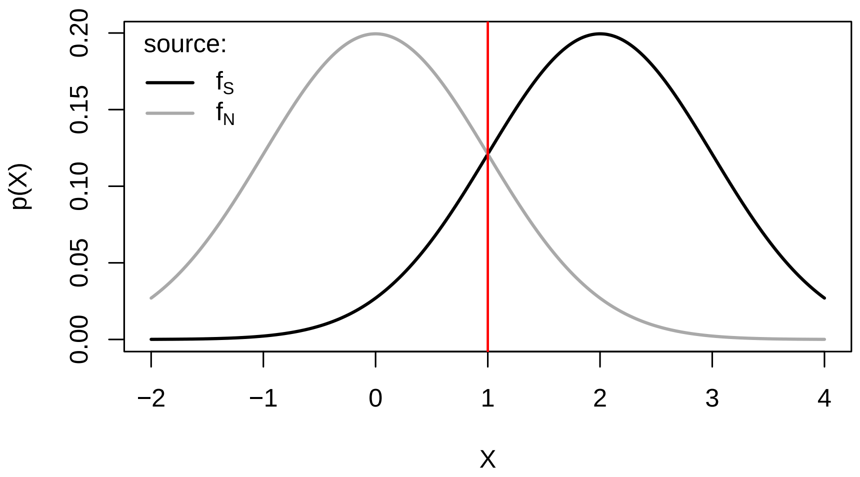
It is easy to verify that when the signal and noise trials are equiprobable, that is $p(s) = p(N) \implies \alpha = 0.5$, the optimal criterion becomes $c = \frac{d'}{2}$.

11.1.1.1 Visualizing signal detection theory in R

We use R to verify visually that the optimal criterion (represented by the vertical red line) is located at horizontal coordinates of the crossover point of the two probability densities:

```r
# settings
d_prime <- 2
sigma <- 1
alpha <- 0.5

# support of random variable X (for plotting)
supp_x <- seq(-2,4,length.out=500)

# calculate optima criterion
optimal_c <- 1/d_prime * log((1-alpha)/alpha) + d_prime/2
```

```
# calculate probability density and scale by prior probability
fS <- alpha*dnorm(supp_x, mean=d_prime, sd=sigma)
fN <- (1-alpha)*dnorm(supp_x, mean=0, sd=sigma)

# plot
plot(supp_x, fS, type="l",lwd=2,col="black",xlab="X",ylab="p(X)")
lines(supp_x, fN, lwd=2,col="dark grey")
abline(v=optimal_c,lwd=1.5,lty=1,col="red")
legend("topleft",c(expression("f"["S"]),expression("f"["N"])),col=c("black","dark grey'
```



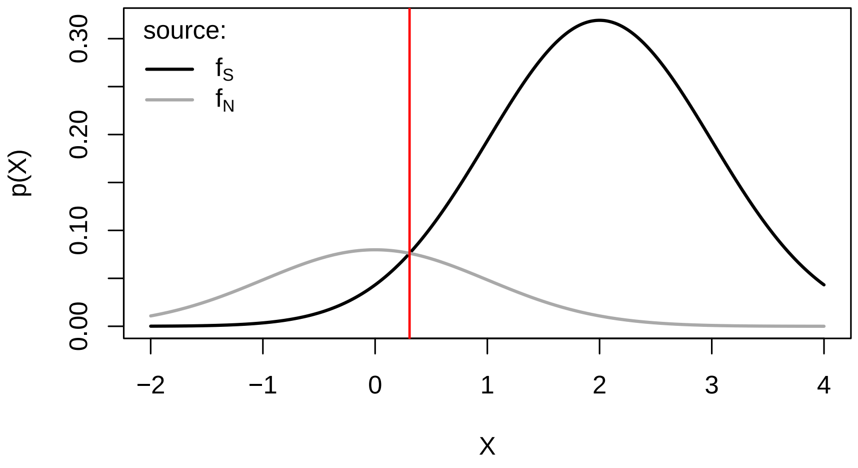
(Note that the noise distribution, in dark grey, is centered on zero. The signal distribution is centered on d'.)

What if we have unequal prior probabilities, e.g. $p(S) = \alpha = 0.8$?

The optimal criterion is always at the crossover point, which however is in a different location since the two distribution are scaled by their prior proability.

```
# Set a different prior probability
alpha <- 0.8

# calculate optima criterion
optimal_c <- 1/d_prime * log((1-alpha)/alpha) + d_prime/2
```

```
# calculate probability density and scale by prior probability
fS <- alpha*dnorm(supp_x, mean=d_prime, sd=sigma)
fN <- (1-alpha)*dnorm(supp_x, mean=0, sd=sigma)

# plot
plot(supp_x, fS, type="l",lwd=2,col="black",xlab="X",ylab="p(X)")
lines(supp_x, fN, lwd=2,col="dark grey")
abline(v=optimal_c,lwd=1.5,lty=1,col="red")
legend("topleft",c(expression("f"["S"]),expression("f"["N"])),col=c("black","dark grey"),lwd=2,ti
```



11.1.2 Estimating the parameters from data

The parameters can be easily estimated form the proportions of hits, p_{H}, and false alarms p_{FA}. (Hits are correct 'yes' responses and faalse alarms are incorrect 'yes' responses.)

Consider first that the probability of a false alarm is just the probability of observing $X \geq c$ when X is drawn from the noise distribution f_N.

Thus

$$p_{\text{FA}} = 1 - \Phi(c)$$

where Φ is the cumulative distribution function of the standard normal distribution It words, $\Phi(c)$ is the area that lies to the left of c and under a Gaussian function with mean 0 and variance 1.

Similarly when X is drawn from the signal distribution, the probability of a hit response is the probability that a x drawn from the signal distribution f_S is greater than the criterion c,

$$p_{\text{H}} = 1 - \Phi(c - d')$$

Thus, if we know the proportion of hits and false alarms, we can estimate c and d' using the inverse of Φ, which can be notated as Φ^{-1} and its often referred to as the *quantile function*

$$c = \Phi^{-1}\left(1 - p_{\text{FA}}\right) = -\Phi^{-1}\left(p_{\text{FA}}\right) d' = c - \Phi^{-1}\left(1 - p_{\text{H}}\right) = \Phi^{-1}\left(p_{\text{H}}\right) - \Phi^{-1}\left(p_{\text{FA}}\right)$$

11.1.3 GLM formulation of equal-variance SDT models

Note that the above SDT model could be reformulated a *probit* generalized linear model. This could be expressed as

$$\Phi^{-1}(p_{\text{yes}}) = \beta_0 + \beta_1 x$$

where p_{yes} is the probability of the observer responding that the signal was present, and x is a variable that indicates the presence/absence of the signal as 1/0, respectively. The similarity with the SDT model is evident if we consider that, in the GLM, the probability of a hit or a false alarm correspond to p_{yes} when the signal is present (that is $x = 1$) or absent (that is $x = 0$), respectively. This allows mapping the signal detection theory parameters, d' and c to the GLM intercept and slope parameters, β_0 and β_1

$$c = -\Phi^{-1}(p_{\text{FA}}) = -\beta_0$$

and

$$d' = \Phi^{-1}(p_{\text{H}}) - \Phi^{-1}(p_{\text{FA}}) = \beta_0 + \beta_1 - \beta_0 = \beta_1$$

Recognizing this identity makes it easier to use statistical packages such as R to easily analyse complex design with multiple conditions and interaction effects. It also makes it possible to estimate multi-level (or hierarchical, random-effects) SDT models.

11.2 Bayesian confidence in equal-variance SDT

Formally, confidence should be the Bayesian (that is, subjective) posterior probability that a decision was correct given the evidence available to the observer.

First some notation. Let's use S to indicate the event that a signal was present (an event is something to which we can assign a probability) and N to indicate the event that only noise was presented. Say the observer respond yes, their confidence should correspond to the posterior probability $p(S \mid x)$, that is the probability that a signal was present given that we observed x.

This can be calculated applying Bayes theorem:

$$\begin{aligned} p(S \mid x) &= \frac{p(x \mid S) \times p(S)}{p(x)} \\ &= \frac{p(x \mid S)p(S)}{p(x \mid S)p(S) + p(x \mid N)p(N)} \\ &= \frac{p(x \mid S)\alpha}{p(x \mid S)\alpha + p(x \mid N)(1-\alpha)} \end{aligned}$$

This can be simplified further in cases where we have equal probabilities $p(s) = p(N) \implies \alpha = 0.5$

$$p(S \mid x)_{\alpha=0.5} = \frac{p(x \mid S)}{p(x \mid S) + p(x \mid N)}$$

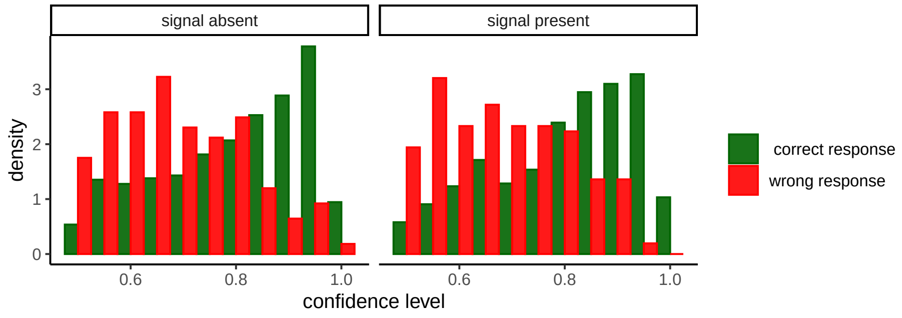
Note also that the confidence for the *signal absent* trials N is calculated in the same way:

$$p(N \mid x)_{\alpha=0.5} = \frac{p(x \mid S)}{p(x \mid N) + p(x \mid N)}$$

One question we may ask at this point is how the distribution of confidence levels of the observers changes in correct vs wrong responses, and also in signal absent vs signal present responses. The simplest way to get at this is by simulation - see the following R code

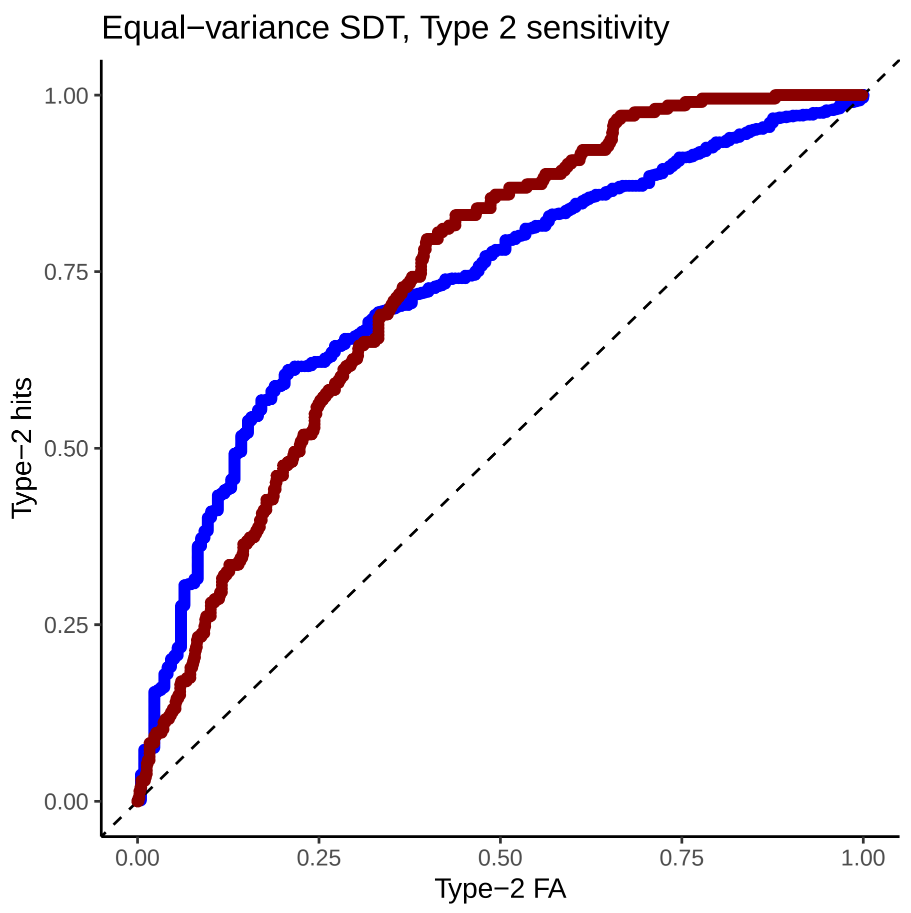
```r
# load ggplot library for plotting
library(ggplot2)

# settings
d_prime <- 1.5
sigma <- 1
alpha <- 0.5
```

```
# calculate optimal criterion
optimal_c <- 1/d_prime * log((1-alpha)/alpha) + d_prime/2

# simulate 2*10^4 trials and calculate the confidence
N_trials <- 2*10^3
tar_pres <- c(rep(0,N_trials/2),rep(1,N_trials/2))

# simulate X by adding Gaussian noise (function rnorm())
x <- tar_pres*d_prime + rnorm(length(tar_pres), mean=0, sd=1)
resp_yes <- ifelse(x >= optimal_c,1,0)

# define a custom function to calculate confidence
confidenceSDT1 <- function(x,resp,d_prime, alpha=0.5){
  conf <- ifelse(
    resp==1,
    dnorm(x,mean=d_prime,sd=1)/(dnorm(x,mean=d_prime,sd=1)+dnorm(x,mean=0,sd=1)),
    dnorm(x,mean=0,sd=1)/(dnorm(x,mean=d_prime,sd=1)+dnorm(x,mean=0,sd=1))
  )
  return(conf)
}

# calculate confidence
confidence <- confidenceSDT1(x, resp_yes, d_prime=1.5)

# put into a dataframe for plotting
d <- data.frame(confidence, x, tar_pres, resp_yes)

# check which simulated responses are correct
d$correct <- ifelse((d$tar_pres==1 & d$resp_yes==1)|(d$tar_pres==0 & d$resp_yes==0),1,

# plot
d$tar <- ifelse(d$tar_pres==1,"signal present","signal absent")
d$correct <- ifelse(d$correct==1," correct response","wrong response")
ggplot(d,aes(x=confidence,group=correct,color=correct,fill=correct))+
  geom_histogram(position = 'dodge',aes(y=..density..), binwidth=0.05,alpha=0.9)+
  facet_grid(.~tar)+
  scale_color_manual(values=c("dark green","red"),name="")+
  scale_fill_manual(values=c("dark green","red"),name="")+
  labs(x="confidence level")+
  theme_classic()
#> Warning: The dot-dot notation (`..density..`) was deprecated in
#> ggplot2 3.4.0.
#> i Please use `after_stat(density)` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
```



The distribution of confidence is - as expected - different from correct and wrong response: it is peaked near 1 for correct responses, and peaked near 0.5 for errors. Importantly, the separation between confidence distributions in correct and wrong responses is similar in both signal absent (left panel) and signal present (right panel) trials. This suggest that metacognitive sensitivity - the ability to discriminate between correct and incorrect responses - should not change across signal present and signal absent answers. This can be visualized with Type-2 ROC (Receiver Operating Characteristic) curves, which are obtained by plotting the proportion of “type-2 hits” as a function of the “type-2 false alarms” - these are the fraction of correct and wrong responses that are classified as correct for each possible threshold setting on the confidence distribution. The term “Type-2” is used to indicate that is a metacognitive task - a decision about a decision(Galvin et al., 2003).

```
# functions to compute true and false positive rates
TPR <- function(d,th){ sum(d$tar_pres==1 & d$x>th) / sum(d$tar_pres==1)}
FPR <- function(d,th){ sum(d$tar_pres==0 & d$x>th) / sum(d$tar_pres==0)}

# use all the sorted values are possible threshods
thresholds <- sort(d$x)

roc <- data.frame(y=sapply(thresholds, function(th){TPR(d[d$resp_yes==1,],th)}),
                  x=sapply(thresholds, function(th){FPR(d[d$resp_yes==1,],th)}) )

roc0 <- data.frame(y=sapply(thresholds, function(th){TPR(d[d$resp_yes==0,],th)}),
                  x=sapply(thresholds, function(th){FPR(d[d$resp_yes==0,],th)}) )

ggplot(roc,aes(x,y))+geom_point(color="blue")+theme_classic()+labs(y="Type-2 hits", x="Type-2 FA'
```



The ROC curve derived from confidence ratings discriminate equally well correct vs wrong responses (in the plot, blue is the curve for target present responses) - thus indicating similar metacognitive or "type-2" sensitivity. However, empiricaly it has been found that metacognitive sensitivite seems to be worse for signal absent responses - e.g. see (Mazor et al., 2020). As we see below, this finding can be accomodated by relaxing the assumption that signal and noise distribution have the same standard deviation, and assuming instead that the standard deviation of the signal present distribution is larger.

11.3 Unequal-variance SDT

Signal detection theory can be extended to account for cases in which signal and noise distribution have a different variance. For many types of random processes, the mean and the variance are related such that signals with higher mean have also higher variance (the firing rate of neurons is an example). Thus the signal distribution is now given by

$$f_S(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-d')^2}{2\sigma^2}}$$

with the variance $\sigma^2 \neq 1$.

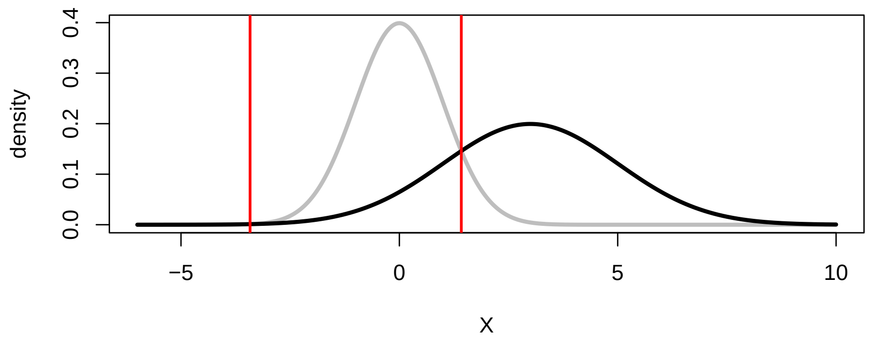
One important consequence of the different noise level, is that now the log-likelihood ratio is a quadratic function of the signal x

$$\log\left(\frac{f_S(x)}{f_N(x)}\right) = -\frac{1}{2\sigma^2}\left[\left(1-\sigma^2\right)x^2 - 2d'x + d'^2 + 2\sigma^2\log\sigma\right]$$

As a result, the log-likelihood ratio crosses zero in 2 points, thus yielding 2 decision criteria - see next figure.

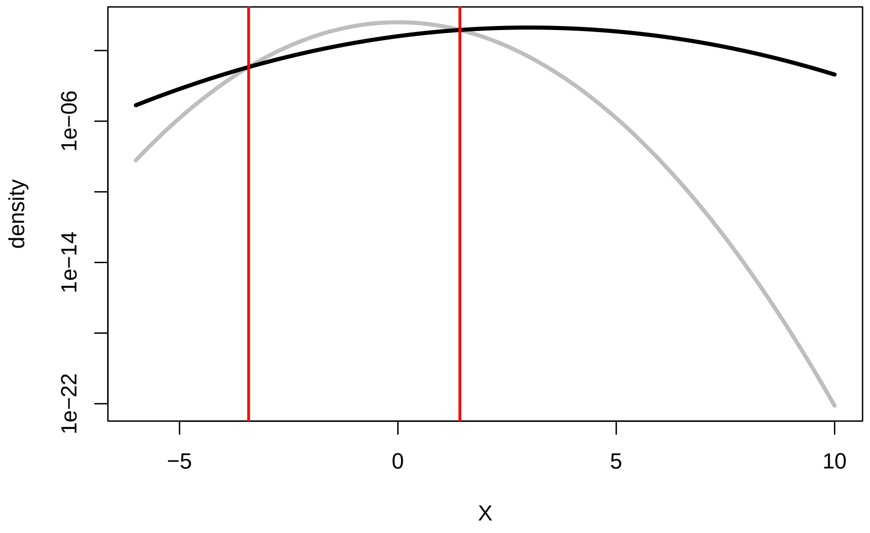
```r
# settings
d_prime <- 3
sigmaS <- 2
sigmaN <- 1 # fixed
alpha <- 0.5


# calculate optimal UEV-SDT criterion
UVGSDTcrit <- function(dp, sig, logbeta=0){
  TwoSigSq <- 2 * sig^2

  minLam <- optimize(function(X, dp, sig){-((1 - sig^2) * X^2 - 2 * dp * X + dp^2 + TwoSigSq * lo

  if(logbeta < minLam){ warning("complex roots")}

  cf <- -c(dp^2 + TwoSigSq * log(sig) + logbeta * TwoSigSq,-2 * dp, 1 - sig^2)/TwoSigSq

  proot <- polyroot(cf)

  return(sort(Re(proot)))
}

#
UE_c <- UVGSDTcrit(d_prime, sigmaS)

# simulate 2*10^3 trials and calculate the confidence
N_trials <- 2*10^3
tar_pres <- c(rep(0,N_trials/2),rep(1,N_trials/2))
```

```r
# simulate X by adding Gaussian noise (function rnorm())
# first generate internal responses
x <- rep(NA,length(tar_pres))
x[tar_pres==0] <- rnorm(N_trials/2, mean=0,sd=1)
x[tar_pres==1] <- rnorm(N_trials/2, mean=d_prime,sd=sigmaS)


# plot unequal variance and criterion
Xi <- seq(-6,10,length.out=500)
fS = dnorm(Xi,mean=d_prime,sd=sigmaS)
fN = dnorm(Xi,mean=0,sd=1)
plot(Xi,fN,type="l",col="grey",lwd=3,ylab="density",xlab="X")
lines(Xi,fS,lwd=3)
abline(v=UE_c,col="red",lwd=2)
```



The reason why there are two criteria may be seen more clearly if we plot the logarithm of the probability density, as this makes it evident that there are two regions, to the left and the right of the noise distribution, in which the probability of signal present is larger than that of no-signal (i.e noise only).

```r
# plot unequal variance and criterion
Xi <- seq(-6,10,length.out=500)
fS = dnorm(Xi,mean=d_prime,sd=sigmaS)
fN = dnorm(Xi,mean=0,sd=1)
plot(Xi,fN,type="l",col="grey",lwd=3,ylab="density",xlab="X", log="y")
lines(Xi,fS,lwd=3)
abline(v=UE_c,col="red",lwd=2)
```

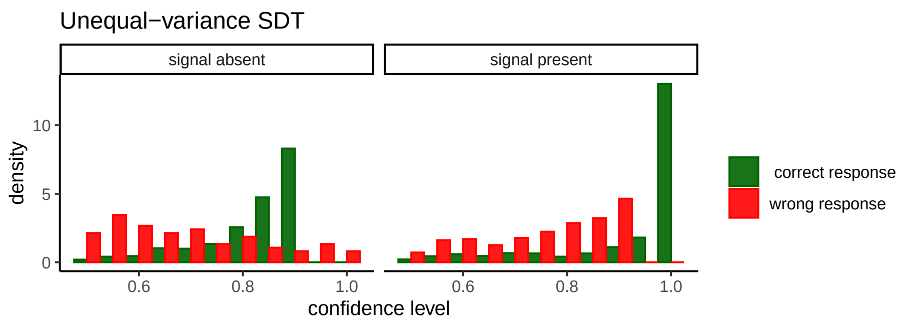


11.3.1 Optimal confidence in unequal-variance SDT

The confidence can be computed in the same way (applying Bayes rule).

```
# now apply decision rule
if(length(UE_c)==1){
  resp_yes <- ifelse(x > UE_c,1,0)
}else{
  resp_yes <- ifelse((x <= UE_c[1])|(x>UE_c[2]),1,0)
}


# define a custom function to calculate confidence
confidenceSDT1 <- function(x,resp,dp=d_prime, alpha=0.5){
  conf <- ifelse(
    resp==1,
    dnorm(x,mean=dp,sd=sigmaS)/(dnorm(x,mean=dp,sd=sigmaS)+dnorm(x,mean=0,sd=1)),
    dnorm(x,mean=0,sd=1)/(dnorm(x,mean=dp,sd=sigmaS)+dnorm(x,mean=0,sd=1))
  )
  return(conf)
}
```

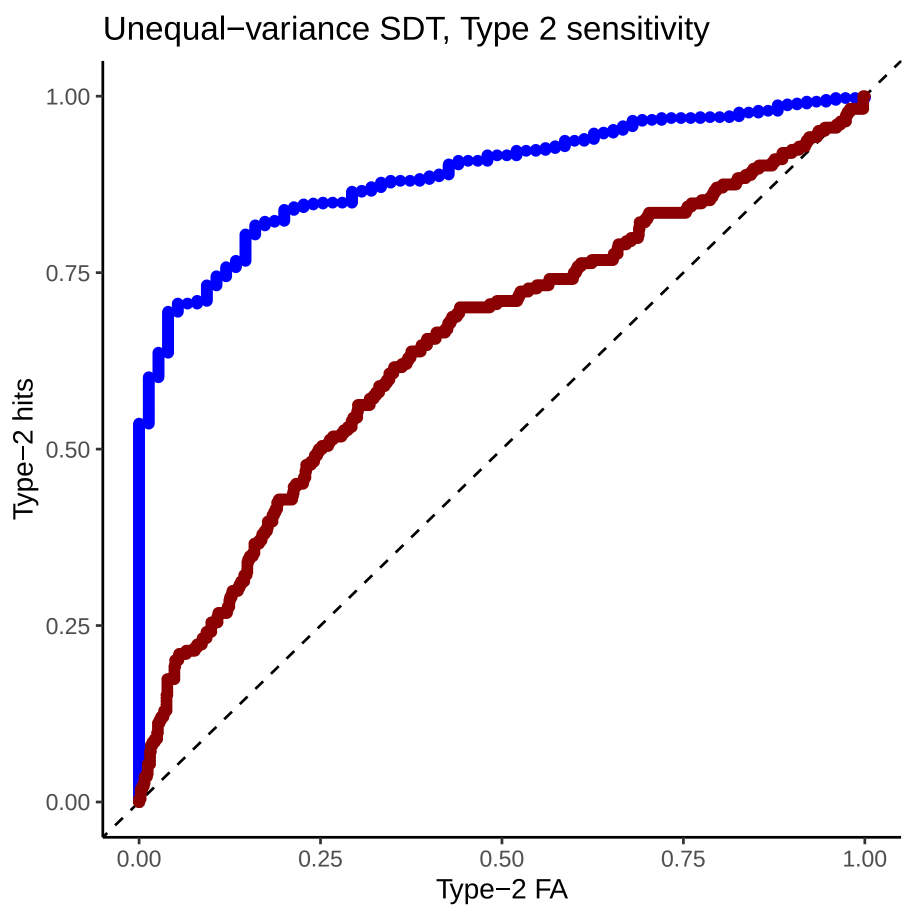
```
# calculate confidence
confidence <- confidenceSDT1(x, resp_yes)

# put into a dataframe for plotting
d <- data.frame(confidence, x, tar_pres, resp_yes)

# check which simulated responses are correct
d$correct <- ifelse((d$tar_pres==1 & d$resp_yes==1)|(d$tar_pres==0 & d$resp_yes==0),1,

# plot
d$tar <- ifelse(d$tar_pres==1,"signal present","signal absent")
d$correct <- ifelse(d$correct==1," correct response","wrong response")
ggplot(d,aes(x=confidence,group=correct,color=correct,fill=correct))+
  geom_histogram(position = 'dodge',aes(y=..density..), binwidth=0.05,alpha=0.9)+
  facet_grid(.~tar)+
  scale_color_manual(values=c("dark green","red"),name="")+
  scale_fill_manual(values=c("dark green","red"),name="")+
  labs(x="confidence level")+
  theme_classic()+
  ggtitle("Unequal-variance SDT")
```



As can be seen from the ROC curve, confidence levels (even estimated optimally using Bayes rule) reveals an asymmetry (again, target present responses are represented by the blue curve). That is, the unequal-variance signal detection theory model predict worse metacognitive sensitivity for “signal absent” responses.

```
# functions to compute true and false positive rates
TPR <- function(d,th){ sum(d$tar_pres==1 & d$x>th) / sum(d$tar_pres==1)}
FPR <- function(d,th){ sum(d$tar_pres==0 & d$x>th) / sum(d$tar_pres==0)}

# use all the sorted values are possible thresholds
```

```
thresholds <- sort(d$x)

roc <- data.frame(y=sapply(thresholds, function(th){TPR(d[d$resp_yes==1,],th)}),
                 x=sapply(thresholds, function(th){FPR(d[d$resp_yes==1,],th)}) )

roc0 <- data.frame(y=sapply(thresholds, function(th){TPR(d[d$resp_yes==0,],th)}),
                 x=sapply(thresholds, function(th){FPR(d[d$resp_yes==0,],th)}) )

ggplot(roc,aes(x,y))+geom_point(color="blue")+theme_classic()+labs(y="Type-2 hits", x="Type-2 FA'
```



Chapter 12

Fitting Zipf's law to word frequency data

Zipf's law predicts that the frequency of any word is inversely proportional to its rank in the frequency table. This phenomenon is observed across many natural languages and can be described by the Zipf-Mandelbrot law. Here I demonstrate how to apply maximum likelihood estimation and the binomial split approach, as described by Piantadosi (Piantadosi, 2014), to fit Zipf's law to the word frequency data of "Moby Dick".

12.1 Background

Zipf's law can be mathematically represented as:

$$P(r) \propto \frac{1}{r^a}$$

where $P(r)$ is the probability of the r-th most common word, and a is a parameter that typically lies close to 1 for natural language.

An extension of this is the Zipf-Mandelbrot law, which introduces a parameter s to account for a finite-size effect:

$$P(r) \propto \frac{1}{(r+s)^a}$$

where s is a positive parameter that shifts the rank.

12.2 Data preparation

We use the word frequency data from "Moby Dick" available in the `languageR` package. First, we load and clean the data:

```
# Clean environment and set working directory
rm(list=ls())

# Load necessary libraries
library(tidyverse)
#> -- Attaching core tidyverse packages ---- tidyverse 2.0.0 --
#> v dplyr     1.1.4     v readr     2.1.5
#> v forcats   1.0.0     v stringr   1.5.1
#> v ggplot2   3.5.1     v tibble    3.2.1
#> v lubridate 1.9.3     v tidyr     1.3.1
#> v purrr     1.0.2
#> -- Conflicts ---------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
#> i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflict
```

```
library(languageR)

# Load and prepare Moby Dick word frequency data
data(moby)
words <- moby[which(moby != "")]

# create a table with word frequencies
word_freq <- table(words)
word_freq_df <- as.data.frame(word_freq, stringsAsFactors=FALSE)
names(word_freq_df) <- c("word", "frequency")
head(word_freq_df)
#>       word frequency
#> 1        -         3
#> 2    -west         1
#> 3  -wester         1
#> 4 -Westers         1
#> 5       [A         1
#> 6  [SUDDEN         1
```

We then rank the words by frequency

```
ranked_words <- word_freq_df %>%
  arrange(desc(frequency)) %>%
```

```r
  mutate(word = factor(word, levels = word),
         rank = rank(-frequency, ties.method = "random"))
```

12.3 Estimation

We define custom functions for the binomial split, the negative log-likelihood, and the Zipf–Mandelbrot law probability mass function:

```r
# Binomial split - randomly split the corpus to ensure independence of rank and frequencies estin
binomial_split <- function(data, p=0.5){
  f1 <- rep(NA, nrow(data))
  f2 <- rep(NA, nrow(data))
  for(i in 1:nrow(data)){
    f1[i] <- rbinom(1, size=data$frequency[i], prob=p)
    f2[i] <- data$frequency[i] - f1[i]
  }
  data_split <- data.frame(
    word = data$word,
    frequency = f1,
    rank = rank(-f2, ties.method = "random")
  )
  return(data_split)
}

# negative log-likelihood function (for optimization)
neglog_likelihood <- function(params, data) {
  s <- params[1]
  a <- params[2]

  N <- length(data$frequency)
  M <- sum(data$frequency)
  logP_data <- M*log(sum(((1:N)+s)^(-a)))+a*sum(data$frequency[data$rank]*log(data$rank+s))
  return(logP_data)
}

# probability mass function for Zipf-Mandelbrot law
# as defined in: https://en.wikipedia.org/wiki/Zipf%E2%80%93Mandelbrot_law
dzipf <- function(rank, params, N){
  s <- params[1]
  a <- params[2]
  p <- ((rank + s)^(-a))/(sum((rank + s)^(-a)))
  return(p)
}
```

We then fit the model using the `optim` function:

```r
# Initial parameter values for optimization
init_params <- c(s = 1, a = 1)

# Fitting model to the data
fit_zipf <- optim(par = init_params,
                  fn = neglog_likelihood,
                  data = ranked_words,
                  method = "L-BFGS-B",
                  lower = c(0, 0),
                  upper = c(100, 100),
                  hessian = TRUE)
```

12.4 Results

Here are the estimated values of the parameters s and a

```r
print(fit_zipf$par)
#>        s        a
#> 1.817667 1.072602
```

We can get standard errors of the parameters estimates from the Hessian matrix:

```r
sqrt(diag(solve(fit_zipf$hessian)))
#>           s           a
#> 0.032707790 0.001272293
```

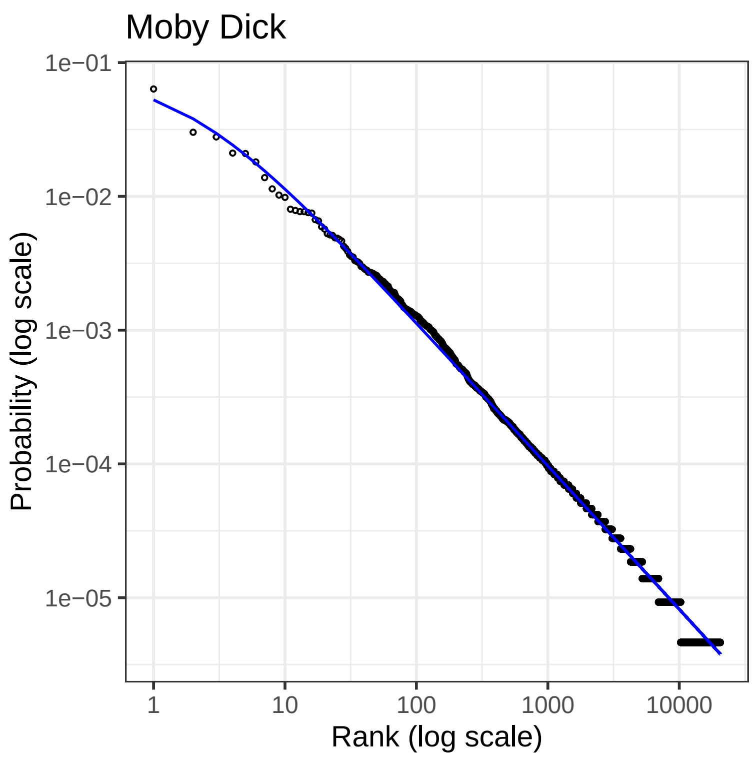
Note that these standard errors do not take into account the additional sampling variability due to the binomial split. In order to take this into account we can re-estimate the model for multiple random splits, and examine how much parameters vary across spits.

12.5 Visualization

Finally, we can visualize the fitted model against the actual data in the classical rank-frequency plot:

```r
#computed predicted probabilities
ranked_words$probability_predicted <- dzipf(rank=ranked_words$rank,
                                            params=fit_zipf$par,
                                            N=length(ranked_words$frequency))
```

```r
# Make rank-frequency plot using ggplo2
ggplot() +
  geom_point(data=ranked_words, pch=21, size=0.6,
             aes(x = rank, y = frequency/sum(frequency))) +
  geom_line(data=ranked_words,
            aes(x = rank,y = probability_predicted),
            color="blue") +
  scale_x_log10() +
  scale_y_log10() +
  theme_bw() +
  labs(
    title = "Moby Dick",
    x = "Rank (log scale)",
    y = "Probability (log scale)"
  )
```



Chapter 13

Workshops

Either click on the links or click on the embedded slides and press 'F' to go full screen. Advance through slides with arrow keys, and press 'O' for an overview of all slides.

13.1 Linear multilevel models (LMM) workshop (9th Sept 2022)

Part 1 (link)

Part 2 (link)

Practicals

Exercises with solution (link)

13.2 Introduction to meta-analyses in R

22nd February 2023

(link)

13.3 Power analyses via data simulation

15th November 2023

Links: (slides), (script)

13.4 Introduction to Bayesian data analysis using R and Stan

29th November 2023

Link: (slides),

See this Github repository for code examples and datasets: https://github.com/mattelisi/intro-Bayes.

13.5 Introduction to causal reasoning with graphical models

15th January 2025

Link: (slides),

13.6 Introduction to linear algebra

Lecture for MSc module PS5210 (version from: 7th December 2022)

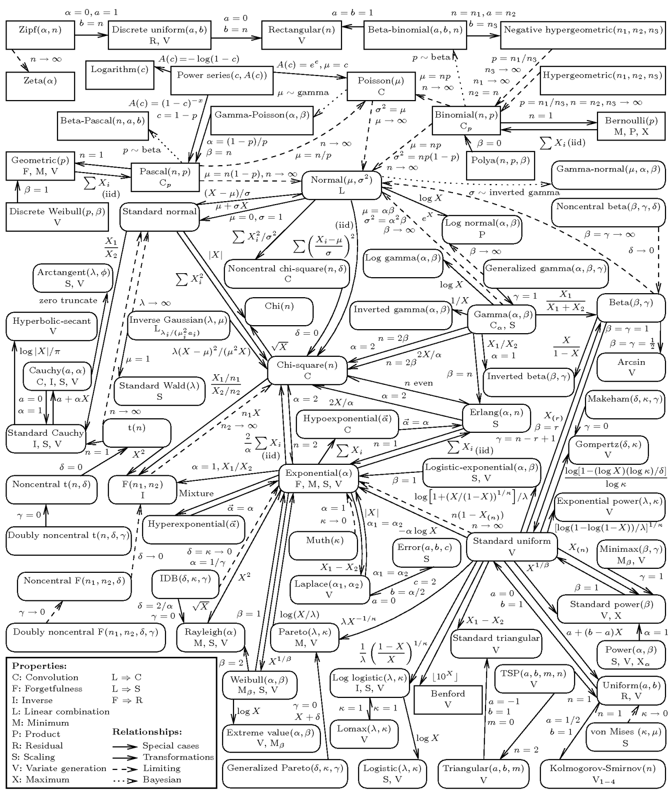
Part 2 (link)

Chapter 14

Useful links & resources

14.1 Statistical theory

14.1.1 Map of univariate statistical distributions



Bibliography

Galvin, S. J., Podd, J. V., Drga, V., and Whitmore, J. (2003). Type 2 tasks in the theory of signal detectability: Discrimination between correct and incorrect decisions. *Psychonomic Bulletin & Review*, 10(4):843–876.

Harrer, M., Cuijpers, P., A, F. T., and Ebert, D. D. (2021). *Doing Meta-Analysis With R: A Hands-On Guide*. Chapman & Hall/CRC Press, Boca Raton, FL and London, 1st edition.

Hubble, E. (1929). A relation between distance and radial velocity among extra-galactic nebulae. *Proceedings of the National Academy of Sciences*, 15(3):168–173.

Liddell, T. M. and Kruschke, J. K. (2018). Analyzing ordinal data with metric models: What could possibly go wrong? *Journal of Experimental Social Psychology*, 79:328–348.

Little, R. J. A. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, 83(404):1198–1202.

Mazor, M., Friston, K. J., and Fleming, S. M. (2020). Distinct neural contributions to metacognition for detecting, but not discriminating visual stimuli. *eLife*, 9:e53900.

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):109–127.

McElreath, R. (2020). *Statistical Rethinking, A Course in R and Stan*. Chapman & Hall/CRC Press, 2nd edition.

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21:1112–1130.

RUBIN, D. B. (1976). Inference and missing data. *Biometrika*, 63(3):581–592.

van Buuren, S. (2018). *Flexible imputation of missing data*. Chapman & Hall/CRC Press, 2nd edition.

Williams, D. R., Rast, P., and Bürkner, P. C. (2018). Bayesian meta-analysis with weakly informative prior distributions.

Wood, S. (2017). *Generalized Additive Models: An Introduction with R.* Chapman & Hall/CRC Press, 2nd edition.