Bayesian Data Analysis: An introduction using brms and rstan

Lecture notes for the Statistical Methods for Linguistics and Psychology summer school, University of Potsdam, Germany

Shravan Vasishth

2019-09-01

Contents

1 Foundational ideas

1.1 Introduction

A few things have been borrowed from the linear modeling and hierarchical linear modeling chapters in https://github.com/vasishth/FGME_Stan_2017. The course notes have benefitted a lot from the lectures and writings of Michael Betancourt. Other sources are acknowledged within the lecture notes.

1.1.1 Intended audience and prerequisites

These notes are intended to accompany a one-week introductory course on Bayesian statistics specifically for linguistics and psychology.

I assume here that the student taking this course has elementary numeracy up to class 10 level. Calculus and linear algebra are occasionally used in the notes but the basic story should be clear even without a full understanding of these topics. No **active** ability in these areas is needed. What I do assume is basic algebra, basic set theory, and arithmetic ability. For example:

- $x^a * x^b = x^{a+b}$
- $\exp(\log(x)) = x$

Some very basic knowledge of R is assumed, but not much. Please be prepared to learn R functionality as needed, either using Google or by reading online help and tutorials.

1.1.2 Software needed

Before starting, please install

- R and RStudio
- The R package rstan:
  - Instructions for Windows
  - Instructions for Mac or Linux
- The R package brms

Please talk to me if you have difficulties installing anything, although be warned that my knowledge of Windows is limited to knowing that it exists.

The central idea we will explore in this course is: given data, how to use Bayes' theorem to quantify uncertainty about a scientific question of interest. In order to understand the methodology, some passive understanding of the following topics needs to be in place:

- From probability theory:
  - conditional probability
  - Bayes' rule
- The theory of random variables
  - the distinction between the probability density/mass function $f(x)$ vs cumulative distribution function $F(x)$
  - inverse CDF, $F^{-1}(x)$
  - expectation and variance of (transformations of) random variables
- Probability density/mass functions
  - Ten common distributions
  - Jointly distributed random variables
  - Sums of random variables
  - Marginal and conditional pdfs
  - Covariance, correlation, and the variance-covariance matrix
  - Multivariate normal distributions
- Maximum likelihood estimation
  - How to find MLEs analytically (some calculus needed here)
  - How to find MLEs using the R function optim
  - Visualization of the log likelihood

Without a clear understanding of these concepts, confusion is an inevitable outcome when we look at more advanced topics.

But before we dive in, it may help to step back and get an overview of where we are going in this course.

1.1.3 Preview: Steps in Bayesian analysis

The way we will conduct data analysis is as follows.

- Given data, specify a *likelihood function*.
- Specify *prior distributions* for model parameters.

- Evaluate whether model makes sense, using *fake-data simulation*, *prior predictive* and *posterior predictive* checks, and (if you want to claim a discovery) calibrating *true* and *false discovery rates*.
- Using software, derive *marginal posterior distributions* for parameters given likelihood function and prior density. I.e., simulate parameters to get *samples from posterior distributions* of parameters using some *Markov Chain Monte Carlo (MCMC) sampling algorithm*, specifically, the Hamiltonian Monte Carlo method.
- Check that the model converged using *model convergence* diagnostics.
- Summarize *posterior distributions* of parameter samples and make your scientific decision.

The above is what you will learn in this course; all the terms introduced above will be explained in these notes and in class. We begin with basic probability theory.

1.2 Bayes' rule and conditional probability

Assume that A and B are events. Conditional probability is defined as follows. $P(A|B)$ below is read as 'the probability that A happens given that B has happened.'

$$P(A|B) = \frac{P(A,B)}{P(B)} \text{ where } P(B) > 0 \tag{1}$$

This means that $P(A,B) = P(A|B)P(B)$.

Since $P(B,A) = P(A,B)$, we can write:

$$P(B,A) = P(B|A)P(A) = P(A|B)P(B) = P(A,B). \tag{2}$$

Rearranging terms:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{3}$$

This gives us Bayes' rule, which is the basis of the whole course and all the statistical inference we will do.

At some point, it will be useful for you to review basic probability theory. A good starting point is Morin (2016). For a deeper understanding, I suggest Kerns (2018) and Blitzstein and Hwang (2014) (a new edition of the Blitzstein book is out or coming out soon).

1.3 Random variable theory

A random variable X is a function $X : S \rightarrow \mathbb{R}$ that associates to each outcome $\omega \in S$ exactly one number $X(\omega) = x$.

S_X is all the x's (all the possible values of X, the support of X). I.e., $x \in S_X$. We can also sloppily write $X \in S_X$.

Good example: number of coin tosses till H

- $X : \omega \rightarrow x$
- ω: H, TH, TTH,... (infinite)
- $x = 0, 1, 2, \dots ; x \in S_X$

Every discrete (continuous) random variable X has associated with it a **probability mass (distribution) function (pmf, pdf)**. I.e., PMF is used for discrete distributions and PDF for continuous. (I will sometimes use lower case for pdf and sometimes upper case. Some books use pdf for both discrete and continuous distributions.)

$$p_X : S_X \rightarrow [0, 1] \tag{4}$$

defined by

$$p_X(x) = P(X(\omega) = x), x \in S_X \tag{5}$$

[**Note**: Books sometimes abuse notation by overloading the meaning of X. They usually have: $p_X(x) = P(X = x), x \in S_X$]

Probability density functions (continuous case) or probability mass functions (discrete case) are functions that assign probabilities or relative frequencies to all events in a sample space.

The expression

$$X \sim f(\cdot) \tag{6}$$

means that the random variable X has pdf/pmf $f(\cdot)$. For example, if we say that $X \sim N(\mu, \sigma)$, we are assuming that the pdf is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp[-\frac{(x-\mu)^2}{2\sigma^2}] \tag{7}$$

We also need a **cumulative distribution function** or cdf because, in the continuous case, P(X=some point value) is zero and we need a way to talk about P(X in a specific range). cdfs serve that purpose.

In the continuous case, the cdf or distribution function is defined as:

$$P(X < x) = F(X < x) = \int_{-\infty}^{X} f(x)\,dx \tag{8}$$

1.3.1 The normalization constant in pdfs

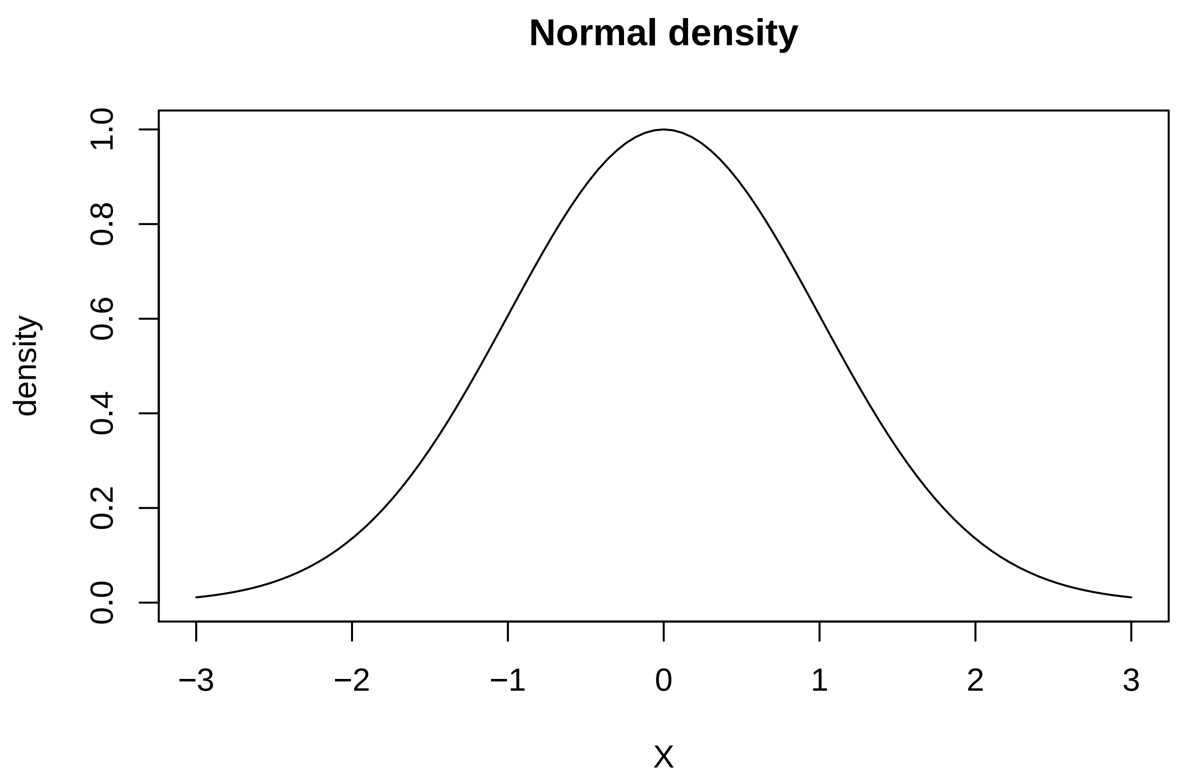
Almost any function can be a pdf as long as the area under the curve sums to 1 over the sample space. Here is an example of a function whose area under the curve doesn't sum to 1:

$$f(x) = \exp[-\frac{(x-\mu)^2}{2\sigma^2}] \tag{9}$$

This is the "kernel" of the normal pdf, and it doesn't sum to 1. We can show that quickly by writing a function in R that expresses this kernel, and then summing up the area under the curve by **integrating** the function in R, from -Infinity to +Infinity.

In what is shown below, integrating the function f(x) is written in mathematics as $\int_a^b f(x)dx$, and simply means that we sum up the area under the continuous function between the ranges a and b.

```
## define function:
normkernel<-function(x,mu=0,sigma=1){
  exp((-(x-mu)^2/(2*(sigma^2))))
}

## plot kernel density function:
plot(function(x) normkernel(x), -3, 3,
      main = "Normal density",ylim=c(0,1),
             ylab="density",xlab="X")
```



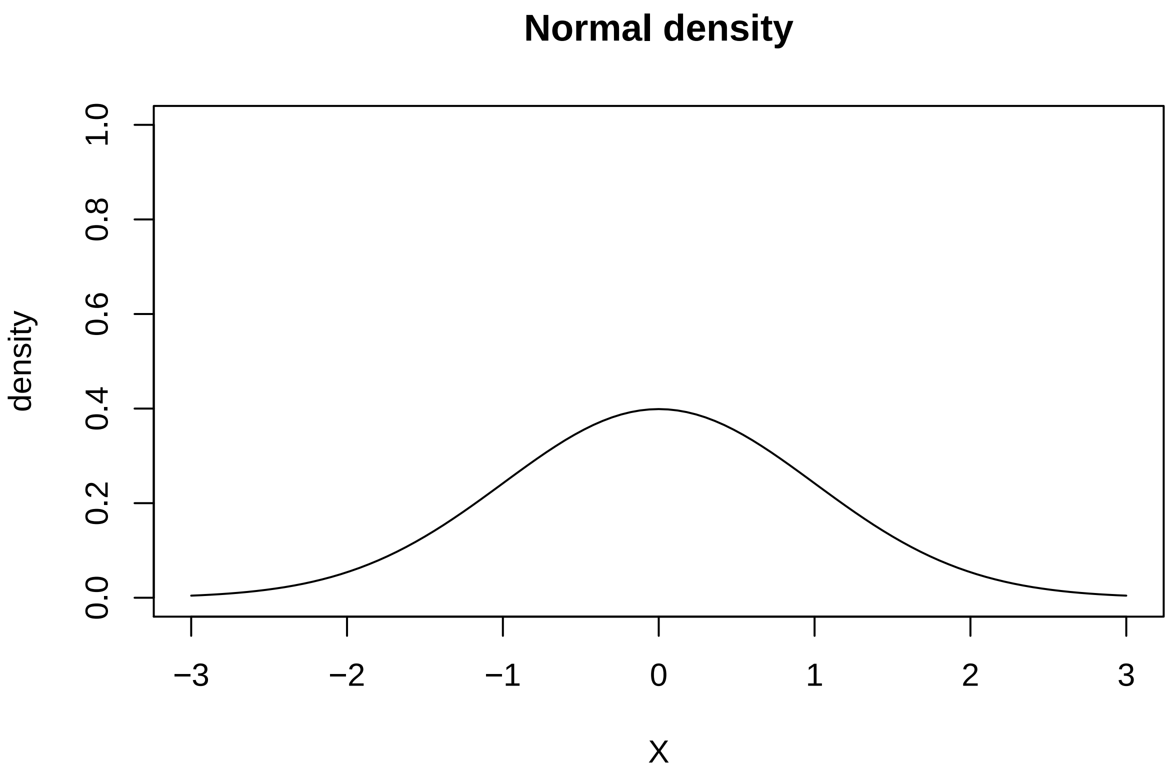
```
## compute area under the curve
## the area under the curve is not equal to 1:
integrate(normkernel,lower=-Inf,upper=Inf)
```

```
## 2.51 with absolute error < 0.00023
```

So, here, $\int_{-\infty}^{\infty} f(x)dx = 2.51$.

Adding a normalizing constant, $\frac{1}{\sqrt{2\pi\sigma^2}}$, makes the above kernel density a pdf.

```
norm<-function(x,mu=0,sigma=1){
  (1/sqrt(2*pi*(sigma^2))) * exp((-(x-mu)^2/(2*(sigma^2))))
}

plot(function(x) norm(x), -3, 3,
      main = "Normal density",ylim=c(0,1),
              ylab="density",xlab="X")
```



```r
### area under the curve sums to 1:
integrate(norm,lower=-Inf,upper=Inf)
```

```
## 1 with absolute error < 0.000094
```

Now, $\int_{-\infty}^{\infty} f(x)dx = 1$.

1.3.2 The pdf f(x) and the cdf F(x)

Recall that a random variable X is a function $X : S \to \mathbb{R}$ that associates to each outcome $\omega \in S$ exactly one number $X(\omega) = x$. S_X is all the x's (all the possible values of X, the support of X). I.e., $x \in S_X$.

X is a continuous random variable if there is a non-negative function f defined for all real $x \in (-\infty, \infty)$ having the property that for any range of real numbers B=[a,b],

$$P\{X \in B\} = \int_a^b f(x)\,dx \tag{10}$$

Kerns has the following to add about the above:

> Continuous random variables have supports that look like

$$S_X = [a,b] \text{ or } (a,b), \tag{11}$$

or unions of intervals of the above form. Examples of random variables that are often taken to be continuous are:

- the height or weight of an individual,
- other physical measurements such as the length or size of an object, and
- durations of time (usually).

E.g., in psychology and linguistics we take as continous:

1. reading or response time in milliseconds: Here the random variable X has possible values ω ranging from 0 ms to some upper bound b ms, and the RV X maps each possible value ω to the corresponding number (0 to 0 ms, 1 to 1 ms, etc.).
2. EEG signals in microvolts.

Every continuous random variable X has a probability density function (PDF) denoted f_X associated with it that satisfies three basic properties:

1. $f_X(x) > 0$ for $x \in S_X$,
2. $\int_{x \in S_X} f_X(x)\,\mathrm{d}x = 1$, and
3. $\mathbb{P}(X \in A) = \int_{x \in A} f_X(x)\,\mathrm{d}x$, for an event $A \subset S_X$.

We can say the following about continuous random variables:

- Usually, the set A in condition 3 above takes the form of an interval, for example, $A = [c,d]$, in which case

$$\mathbb{P}(X \in A) = \int_c^d f_X(x)\,\mathrm{d}x. \tag{12}$$

- It follows that the probability that X falls in a given interval is simply the area under the curve of f_X over the interval.
- Since the area of a line $x = c$ in the plane is zero, $\mathbb{P}(X = c) = 0$ for any value c. In other words, the chance that X equals a particular value c is zero, and this is true for any number c. Moreover, when $a < b$ all of the following probabilities are the same:

$$\mathbb{P}(a \le X \le b) = \mathbb{P}(a < X \le b) = \mathbb{P}(a \le X < b) = \mathbb{P}(a < X < b). \tag{13}$$

$f(x)$ is the probability density function of the random variable X.

Since X must assume some value, f must satisfy

$$1 = P\{X \in (-\infty, \infty)\} = \int_{-\infty}^{\infty} f(x)\,dx \tag{14}$$

If $B = [a,b]$, then

$$P\{a \le X \le b\} = \int_{a}^{b} f(x)\,dx \tag{15}$$

If $a = b$, we get

$$P\{X = a\} = \int_{a}^{a} f(x)\,dx = 0 \tag{16}$$

Hence, for any continuous random variable,

$$P\{X < a\} = P\{X \le a\} = F(a) = \int_{-\infty}^{a} f(x)\,dx \tag{17}$$

F is the **cumulative distribution function**. Differentiating both sides in the above equation:

$$\frac{dF(x)}{dx} = f(x) \tag{18}$$

Just to reiterate this: the density (PDF) is the derivative of the CDF. You can go back and forth between the pdf and the CDF by integrating or differentiating:

$$\int_{a}^{b} f(x)\,dx \Rightarrow F(a) \tag{19}$$

$$dF(x)/dx = f(x) \tag{20}$$

$F(x)$ will give you some probability u. The **inverse of the cdf**, $F^{-1}(u)$ gives us back the quantile x such that $F(x) = u$. *This fact will be of great relevance to us.*

1.3.3 Some basic results concerning random variables

We will only use facts 1 and 4 below in this course, but it can be useful to know the existence of these other results when you read more advanced textbooks.

1. $$E[X] = \int_{-\infty}^{\infty} xf(x)\,dx \tag{21}$$

2. $$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)\,dx \tag{22}$$

3. $$E[aX+b] = aE[X]+b \tag{23}$$

4. $$Var[X] = E[(X-\mu)^2] = E[X^2]-(E[X])^2 \tag{24}$$

5. $$Var(aX+b) = a^2Var(X) \tag{25}$$

So far, we have learnt what a random variable is, and we know that by definition it has a pdf and a cdf associated with it. Why did we go through all this effort to learn all this? The payoff becomes apparent next.

1.3.4 What you can do with a pdf

You can:

1. Calculate the mean:

   Discrete case:

   $$E[X] = \sum_{i=1}^{n} x_i p(x_i) \tag{26}$$

   Continuous case:

   $$E[X] = \int_{-\infty}^{\infty} xf(x)\,dx \tag{27}$$

2. Calculate the variance:

$$Var(X) = E[X^2] - (E[X])^2 \tag{28}$$

3. Compute quartiles: e.g., for some pdf f(x):

$$\int_{-\infty}^{Q} f(x)\,dx \tag{29}$$

   For example, take $f(x)$ to be the normal distribution with mean 0 and sd 1. Suppose we want to know:

$$\int_{0}^{1} f(x)\,dx \tag{30}$$

   We can do this in R as follows:[1]

```
pnorm(1)-pnorm(0)
```

Why are we going through these basic results? We will be using all kinds of distributions in order to build statistical models and to do inference.

1.4 Ten important distributions

These distributions are generally used quite frequently in Bayesian data analyses, especially in psychology and linguistics applications. For the first few distributions, we show the pdf and cdf. The Binomial and Poisson are discrete distributions, the rest are continuous.

You should install the following add-in into RStudio:

```
if ( !('devtools' %in% installed.packages()) ) install.packages("devtools")

devtools::install_github("bearloga/tinydensR")
```

Then, run

```
library(tinydensR)
univariate_discrete_addin()
```

[1]This is a very important piece of R code here. Make sure you understand the relationship between the integral and the R functions used here.

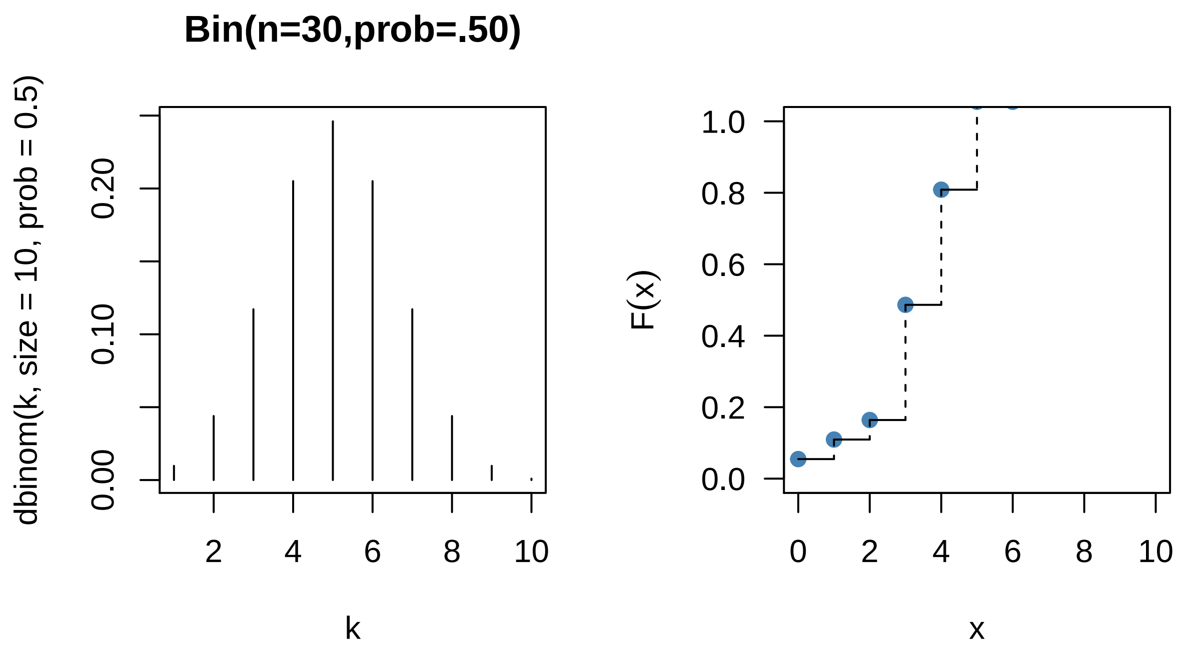


Figure 1: Example of the pmf Bin(n=10,prob=.50).

or

```
univariate_continuous_addin()
```

to visualize the distributions under different parameterizations. Note that RStudio may occasionally crash when this command is run. Just restart RStudio if this happens.

1.4.1 Binomial

Examples: coin tosses, question-response accuracy

Probability mass function:

If we have x successes in n trials, given a success probability p for each trial. If $x \sim Bin(n,p)$.

$$P(x \mid n,p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{31}$$

[Recall that: $\binom{n}{k} = \frac{n!}{(n-r)!r!}$. Hence, given x and n, this term will be a constant.]

Figure 1 shows the pmf and cdf.

Expectation and variance:

The mean is np and the variance $np(1-p)$.

When $n = 1$ we have the Bernoulli distribution.

Relevant functions in R

```
###pmf:
dbinom(x, size, prob, log = FALSE)
### cdf:
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
### inverse cdf:
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
### pseudo-random generation of samples:
rbinom(n, size, prob)
```

Notational conventions: A binomial distribution, n trials each with probability θ of occurring, is written $Bin(\theta, n)$. Given a random variable with this distribution, we can write $R \mid \theta, n \sim Bin(\theta, n)$ or $p(r \mid \theta, n) = Bin(\theta, n)$, where r is the realization of R. We can drop the conditioning in $R \mid \theta, n$, so that we can write: given $R \sim Bin(\theta, n)$, what is $Pr(\theta_1 < \theta < \theta_2 \mid r, n)$.

1.4.2 Poisson

Examples: traffic accidents, typing errors, customers arriving in a bank, number of fixations in reading.

Probability density function

Let λ be the average number of events in the time interval $[0,1]$. Let the random variable X count the number of events occurring in the interval. Then:

$$f_X(x) = \mathbb{P}(X = x) = \mathrm{e}^{-\lambda} \frac{\lambda^x}{x!}, \quad x = 0, 1, 2, \ldots \tag{32}$$

Expectation and variance

$$E[X] = \lambda \tag{33}$$

$$Var(X) = \lambda \tag{34}$$

Relevant functions in R

```
dpois(x, lambda)
ppois(q, lambda, lower.tail = TRUE)
qpois(p, lambda, lower.tail = TRUE)
rpois(n, lambda)
```

1.4.3 Uniform

Example: All outcomes have equal probability.

Probability density function:

A random variable (X) with the continuous uniform distribution on the interval (α, β) has PDF

$$f_X(x) = \begin{cases} \frac{1}{\beta-\alpha}, & \alpha < x < \beta, \\ 0, & \text{otherwise} \end{cases} \tag{35}$$

The associated R function is $\mathsf{dunif}(\mathtt{min} = a, \mathtt{max} = b)$. We write $X \sim \mathsf{unif}(\mathtt{min} = a, \mathtt{max} = b)$. Due to the particularly simple form of this PDF we can also write down explicitly a formula for the CDF F_X:

$$F_X(a) = \begin{cases} 0, & a < 0, \\ \frac{a-\alpha}{\beta-\alpha}, & \alpha \leq t < \beta, \\ 1, & a \geq \beta. \end{cases} \tag{36}$$

The pdf and cdf are show in Figure 2.

Expectation and variance:

$$E[X] = \frac{\beta + \alpha}{2} \tag{37}$$

$$Var(X) = \frac{(\beta - \alpha)^2}{12} \tag{38}$$

Relevant functions in R

```
dunif(x, min = 0, max = 1, log = FALSE)
punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
```

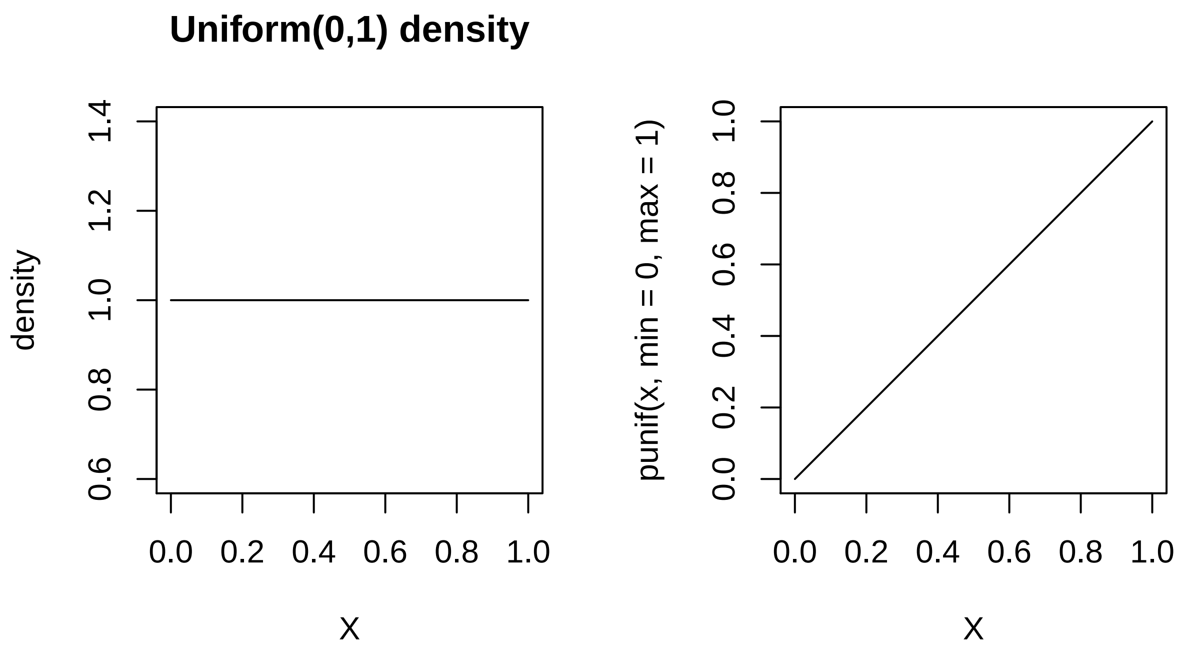


Figure 2: Uniform(0,1) distribution, pdf and cdf.

```
runif(n, min = 0, max = 1)
```

1.4.4 Normal

Examples: heights, weights of people

Probability density function

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty. \tag{39}$$

We write $X \sim \text{norm}(\texttt{mean} = \mu, \texttt{sd} = \sigma)$, and the associated R function is `dnorm(x, mean = 0, sd = 1)`.

If X is normally distributed with parameters μ and σ^2, then $Y = aX + b$ is normally distributed with parameters $a\mu + b$ and $a^2\sigma^2$.

Special case: Standard or unit normal random variable

If X is normally distributed with parameters μ and σ^2, then $Z = (X - \mu)/\sigma$ is normally distributed with parameters $0, 1$.

We conventionally write $\Phi(x)$ for the CDF:

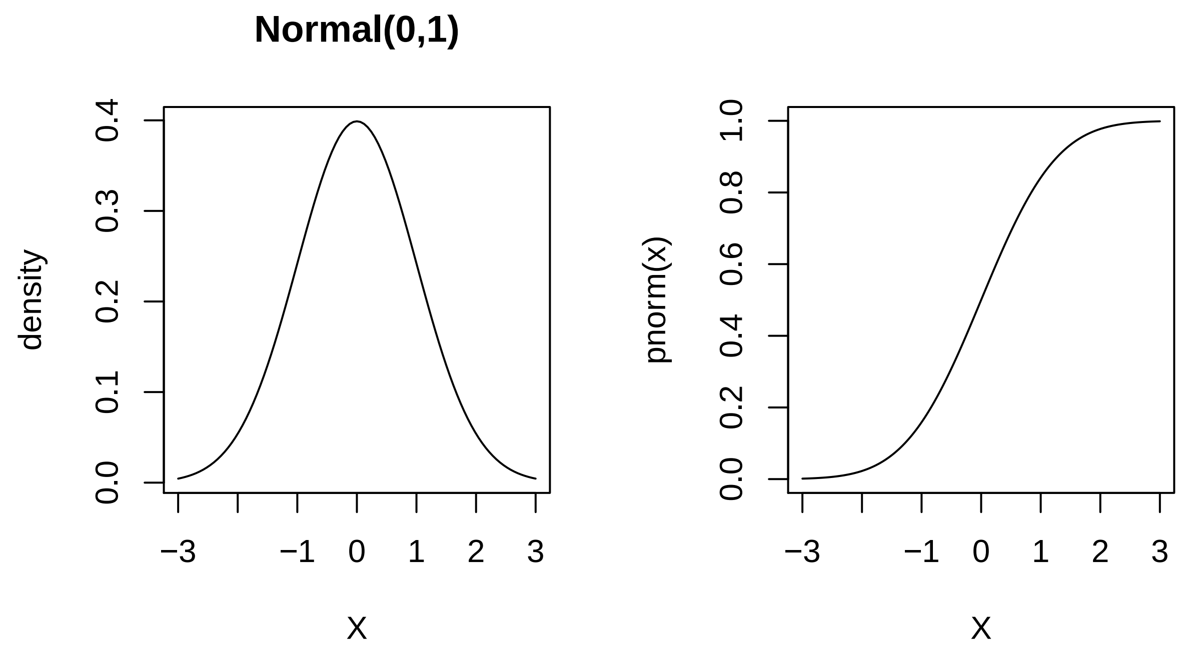


Figure 3: Normal distribution.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-y^2}{2}}\, dy \quad \text{where } y = (x-\mu)/\sigma \tag{40}$$

Old-style (pre-computer era) printed tables give the values for positive x; for negative x we do:

$$\Phi(-x) = 1 - \Phi(x), \quad -\infty < x < \infty \tag{41}$$

If Z is a standard normal random variable (SNRV) then

$$p\{Z \leq -x\} = P\{Z > x\}, \quad -\infty < x < \infty \tag{42}$$

Since $Z = ((X-\mu)/\sigma)$ is an SNRV whenever X is normally distributed with parameters μ and σ^2, then the CDF of X can be expressed as:

$$F_X(a) = P\{X \leq a\} = P\left(\frac{X-\mu}{\sigma} \leq \frac{a-\mu}{\sigma}\right) = \Phi\left(\frac{a-\mu}{\sigma}\right) \tag{43}$$

The standardized version of a normal random variable X is used to compute specific probabilities relating to X (it is also easier to compute probabilities from different CDFs so that the two computations are comparable).

Expectation and variance

$$E[X] = \mu \tag{44}$$

$$Var(X) = \sigma^2 \tag{45}$$

Relevant functions in R

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

1.4.5 Log-Normal

Examples: reaction time, reading time

Probability density function

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(\log x-\mu)^2}{2\sigma^2}}, \quad 0 < x < \infty. \tag{46}$$

```
op<-par(mfrow=c(1,2),pty="s")
plot(function(x) dlnorm(x), 0, 4,
      main = "LogNormal(0,1)",
              ylab="density",xlab="X")
x<-seq(0,4,by=0.001)
plot(x,plnorm(x),main="",type="l",xlab="X")
```

Note:

-μ, σ are on the log scale. -If $X \sim LogNormal(\mu, \sigma)$, this is the same as saying that $\log(X)$ is normally distributed.

Expectation and variance

$$E[X] = \exp(\mu + \sigma^2/2) \tag{47}$$

$$Var(X) = E[X]\exp(\sigma^2 - 1) \tag{48}$$

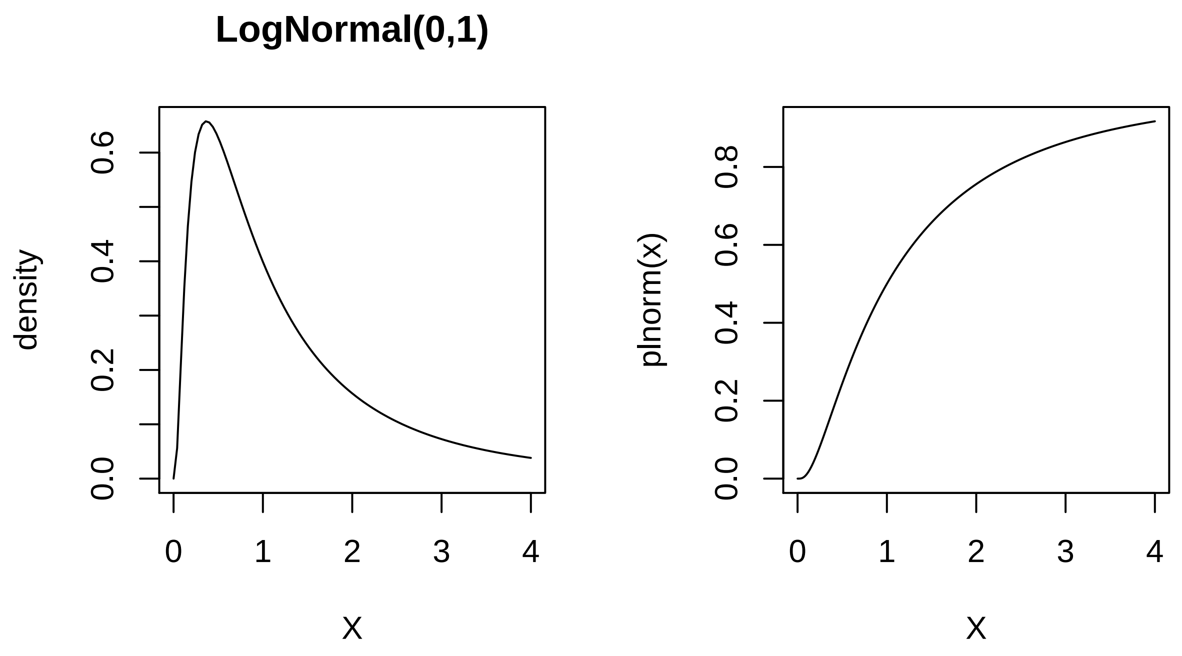


Figure 4: LogNormal(0,1) distribution.

Relevant functions in R

```
dlnorm(x, meanlog = 0, sdlog = 1)
plnorm(q, meanlog = 0, sdlog = 1, lower.tail = TRUE)
qlnorm(p, meanlog = 0, sdlog = 1, lower.tail = TRUE)
rlnorm(n, meanlog = 0, sdlog = 1)
```

1.4.6 Beta

Example: Distribution of probability of giving a correct answer in a yes/no question-response task.

Probability density function

This is a generalization of the continuous uniform distribution. Think of parameter *a* as number of successes, and parameter *b* as number of failures.

$$f(x) = \begin{cases} \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

where

$$Beta(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\, dx$$

We write $X \sim \texttt{beta}(\texttt{shape1} = a, \texttt{shape2} = b)$.

Expectation and variance

$$E[X] = \frac{a}{a+b} \text{ and } Var(X) = \frac{ab}{(a+b)^2\,(a+b+1)}. \tag{49}$$

Relevant functions in R

```
dbeta(x, shape1, shape2)
pbeta(q, shape1, shape2)
qbeta(p, shape1, shape2)
rbeta(n, shape1, shape2)
```

1.4.7 Exponential

Examples: Waiting time for an arrival from a Poisson process (e.g., reading times)

Probability density function

For some $\lambda > 0$,

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases}$$

Expectation and variance

$$E[X] = \frac{1}{\lambda} \tag{50}$$

$$Var(X) = \frac{1}{\lambda^2} \tag{51}$$

Relevant functions in R

```
dexp(x, rate=1)
pexp(q, rate=1)
qexp(p, rate=1)
rexp(n, rate=1)
```

1.4.8 Gamma

Examples: reading times, distribution of inverse of variance.

Connection to Poisson: if X measures the length of time until the first event occurs in a Poisson process with rate λ then $X \sim \texttt{exp}(\texttt{rate} = \lambda)$. If we let Y measure the length of time until the α^{th} event occurs then $Y \sim \texttt{gamma}(\texttt{shape} = \alpha, \texttt{rate} = \lambda)$. When α is an integer this distribution is also known as the **Erlang** distribution.

The Chi-squared distribution is the Gamma distribution with $\lambda = 1/2$ and $\alpha = n/2$, where n is an integer:

Probability density function

This is a generalization of the exponential distribution. We say that X has a Gamma distribution and write $X \sim \texttt{gamma}(\texttt{shape} = \alpha, \texttt{rate} = \lambda)$, where $\alpha > 0$ (called shape) and $\lambda > 0$ (called rate). It has PDF

$$f(x) = \begin{cases} \frac{\lambda e^{-\lambda x}(\lambda x)^{\alpha-1}}{\Gamma(\alpha)} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases}$$

$\Gamma(\alpha)$ is called the gamma **function** (note: it's lower case gamma, and it's a function, not a distribution):

$$\Gamma(\alpha) = \int_0^\infty e^{-y} y^{\alpha-1}\, dy = (\alpha-1)\Gamma(\alpha-1)$$

Note that for integral values of n, $\Gamma(n) = (n-1)!$ (follows from above equation).

Expectation and variance

$$E[X] = \alpha/\lambda \tag{52}$$

$$Var(X) = \alpha/\lambda^2 \tag{53}$$

Relevant functions in R

```
dgamma(x, rate, scale = 1/rate)
pgamma(q, rate, scale = 1/rate)
qgamma(p, rate, scale = 1/rate)
```

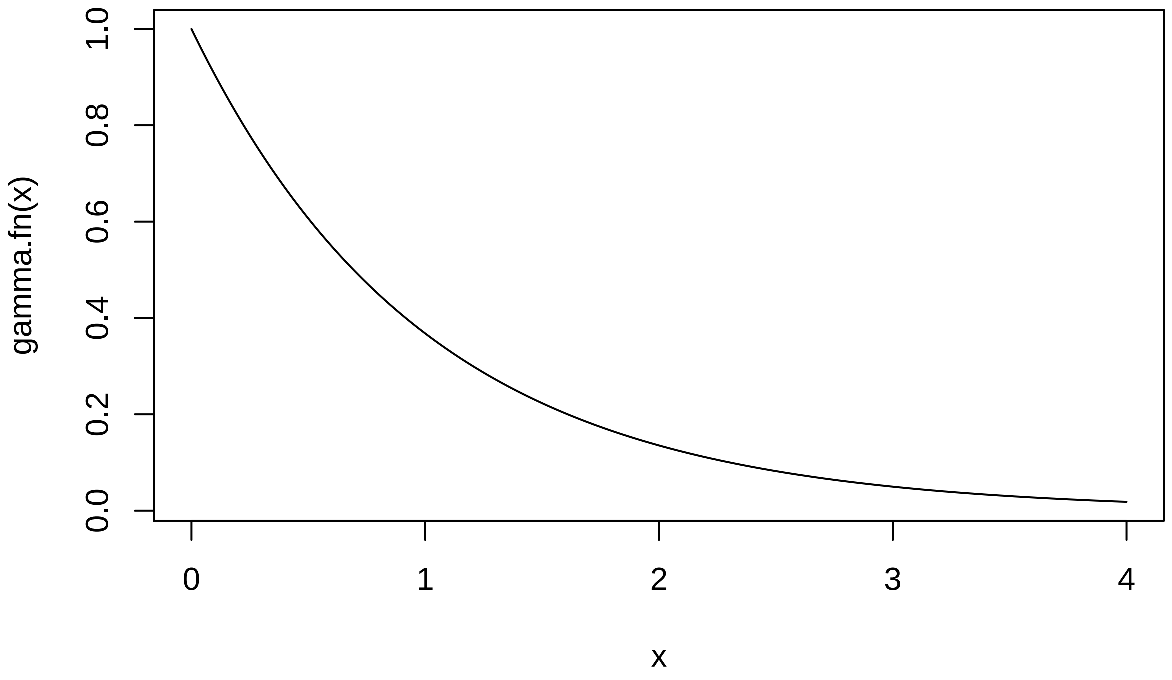


Figure 5: The Gamma distribution.

```
rgamma(n, rate, scale = 1/rate)
```

```
x<-seq(0,4,by=.01)
plot(x,gamma.fn(x),type="l")
```

```
x<-seq(0,100,by=.01)
plot(x,gamma.fn(x),type="l")
```

1.4.9 Student's *t*

Examples: reading times.

Probability density function

A random variable X with PDF

$$f_X(x) = \frac{\Gamma[(r+1)/2]}{\sqrt{r\pi}\,\Gamma(r/2)} \left(1 + \frac{x^2}{r}\right)^{-(r+1)/2}, \quad -\infty < x < \infty \tag{54}$$

is said to have Student's t distribution with r degrees of freedom, and we write $X \sim \mathtt{t}(\mathtt{df} = r)$.

We will write $X \sim t(\mu, \sigma^2, r)$, where r is the degrees of freedom $(n-1)$, where n is sample size.

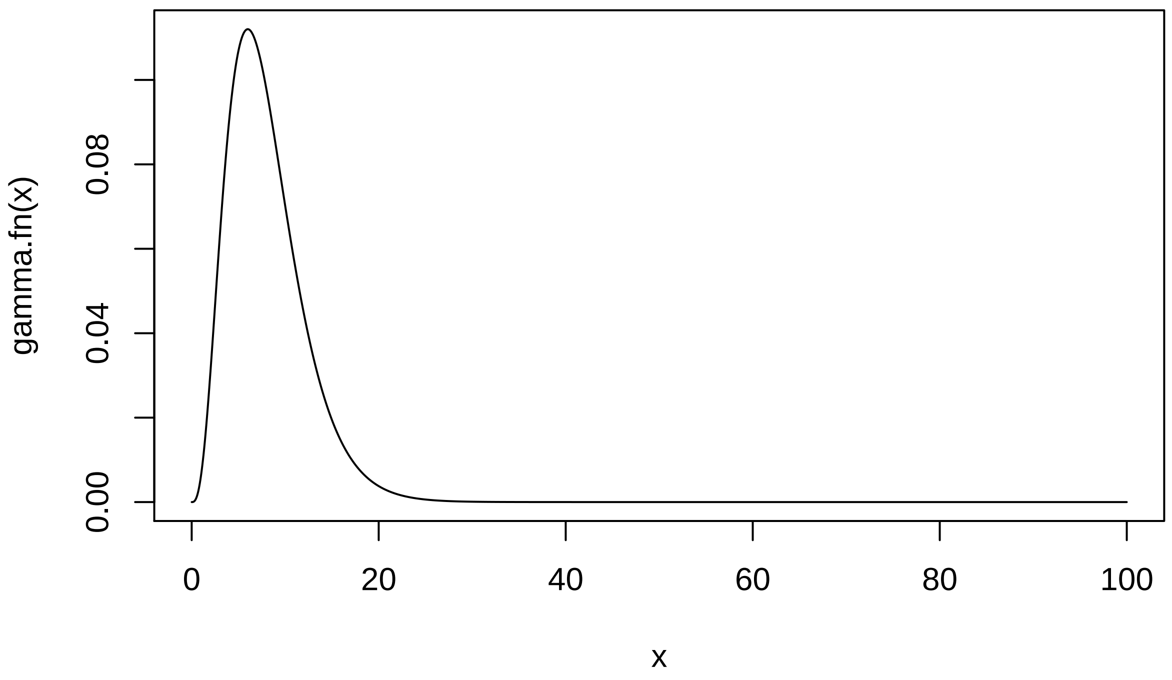


Figure 6: The chi-squared distribution.

Expectation and variance

$$E[X] = 0 \text{ if } n > 1 \text{ otherwise undefined} \tag{55}$$

$$Var(X) = \frac{n}{n-2}, \text{ when } n > 2 \text{ otherwise undefined} \tag{56}$$

Relevant functions in R

```
dt(x, df)
pt(q, df)
qt(p, df)
rt(n, df)
```

1.4.10 Summary of distributions

Distribution	PMF/PDF and Support	Expected Value	Variance
Binomial $Bin(n,p)$	$P(X=k)=\binom{n}{k}p^kq^{n-k}$ $k\in\{0,1,2,\dots n\}$	np	npq
Poisson $Pois(\lambda)$	$P(X=k)=\frac{e^{-\lambda}\lambda^k}{k!}$ $k\in\{0, 1, 2, \dots\}$	λ	λ
Uniform $Unif(a,b)$	$f(x)=\frac{1}{b-a}$ $x\in(a,b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Normal $Normal(\mu,\sigma)$	$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/(2\sigma^2)}$ $x\in(-\infty,\infty)$	μ	
Log-Normal $LogNormal(\mu,\sigma)$	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-(\log x-\mu)^2/(2\sigma^2)}$ $x\in(0,\infty)$	$\theta=e^{\mu+\sigma^2/2}$	$\theta^2(e^{\sigma^2}-1)$
Beta Beta(a,b)	$f(x)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$ $x\in(0,1)$	$\mu=\frac{a}{a+b}$	$\frac{\mu(1-\mu)}{(a+b+1)}$
Exponential $Exp(\lambda)$	$f(x)=\lambda e^{-\lambda x}$ $x\in(0,\infty)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Gamma $Gamma(a,\lambda)$	$f(x)=\frac{1}{\Gamma(a)}(\lambda x)^a e^{-\lambda x}\frac{1}{x}$ $x\in(0,\infty)$	$\frac{a}{\lambda}$	$\frac{a}{\lambda^2}$
Student-t $t(n)$	$\frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)}(1+x^2/n)^{-(n+1)/2}$ $x\in(-\infty,\infty)$	0 if $n>1$	$\frac{n}{n-2}$ if $n>2$

The above table is adapted from https://github.com/wzchen/probability_cheatsheet.

1.5 Jointly distributed random variables

1.5.1 Discrete case

[This section is an extract from Kerns.]

Consider two discrete random variables X and Y with PMFs f_X and f_Y that are supported on the sample spaces S_X and S_Y, respectively. Let $S_{X,Y}$ denote the set of all possible observed **pairs** (x,y), called the **joint support set** of X and Y. Then the **joint probability mass function** of X and Y is the function $f_{X,Y}$ defined by

$$f_{X,Y}(x,y) = \mathbb{P}(X = x, Y = y), \quad \text{for } (x,y) \in S_{X,Y}. \tag{57}$$

Every joint PMF satisfies

$$f_{X,Y}(x,y) > 0 \text{ for all } (x,y) \in S_{X,Y}, \tag{58}$$

and

$$\sum_{(x,y)\in S_{X,Y}} f_{X,Y}(x,y) = 1. \tag{59}$$

It is customary to extend the function $f_{X,Y}$ to be defined on all of $\mathbb{R}^2$ by setting $f_{X,Y}(x,y) = 0$ for $(x,y) \notin S_{X,Y}$.

In the context of this chapter, the PMFs f_X and f_Y are called the **marginal PMFs** of X and Y, respectively. If we are given only the joint PMF then we may recover each of the marginal PMFs by using the Theorem of Total Probability: observe

$$\begin{aligned} f_X(x) &= \mathbb{P}(X = x), && (60)\\ &= \sum_{y\in S_Y} \mathbb{P}(X = x, Y = y), && (61)\\ &= \sum_{y\in S_Y} f_{X,Y}(x,y). && (62) \end{aligned}$$

By interchanging the roles of X and Y it is clear that

$$f_Y(y) = \sum_{x\in S_X} f_{X,Y}(x,y). \tag{63}$$

Given the joint PMF we may recover the marginal PMFs, but the converse is not true. Even if we have **both** marginal distributions they are not sufficient to determine the joint PMF; more information is needed.

Associated with the joint PMF is the **joint cumulative distribution function** $F_{X,Y}$ defined by

$$F_{X,Y}(x,y) = \mathbb{P}(X \le x, Y \le y), \quad \text{for } (x,y) \in \mathbb{R}^2.$$

The bivariate joint CDF is not quite as tractable as the univariate CDFs, but in principle we could calculate it by adding up quantities of the form in Equation~57. The joint CDF is typically not used in practice due to its inconvenient form; one can usually get by with the joint PMF alone.

Example:

Roll a fair die twice. Let X be the face shown on the first roll, and let Y be the face shown on the second roll. For this example, it suffices to define

$$f_{X,Y}(x,y) = \frac{1}{36}, \quad x = 1,\ldots,6,\ y = 1,\ldots,6.$$

The marginal PMFs are given by $f_X(x) = 1/6$, $x = 1,2,\ldots,6$, and $f_Y(y) = 1/6$, $y = 1,2,\ldots,6$, since

$$f_X(x) = \sum_{y=1}^{6} \frac{1}{36} = \frac{1}{6}, \quad x = 1,\ldots,6,$$

and the same computation with the letters switched works for Y.

Here, and in many other ones, the joint support can be written as a product set of the support of X "times" the support of Y, that is, it may be represented as a cartesian product set, or rectangle, $S_{X,Y} = S_X \times S_Y$ where $S_X \times S_Y = \{(x,y) :\ x \in S_X,\ y \in S_Y\}$. This form is a necessary condition for X and Y to be **independent** (or alternatively **exchangeable** when $S_X = S_Y$). But please note that in general it is not required for $S_{X,Y}$ to be of rectangle form.

1.5.2 Continuous case

For random variables X and y, the **joint cumulative pdf** is

$$F(a,b) = P(X \leq a, Y \leq b) \quad -\infty < a,b < \infty \tag{64}$$

The **marginal distributions** of F_X and F_Y are the CDFs of each of the associated RVs:

1. The CDF of X:

$$F_X(a) = P(X \leq a) = F_X(a,\infty) \tag{65}$$

2. The CDF of Y:

$$F_Y(a) = P(Y \leq b) = F_Y(\infty,b) \tag{66}$$

Definition 1. ***Jointly continuous***: *Two RVs X and Y are jointly continuous if there exists a function $f(x,y)$ defined for all real x and y, such that for every set C:*

$$P((X,Y) \in C) = \iint\limits_{(x,y)\in C} f(x,y)\, dx\, dy \tag{67}$$

*$f(x,y)$ is the **joint PDF** of X and Y.*

Every joint PDF satisfies

$$f(x,y) \geq 0 \text{ for all } (x,y) \in S_{X,Y}, \tag{68}$$

and

$$\iint\limits_{S_{X,Y}} f(x,y)\, \mathrm{d}x\, \mathrm{d}y = 1. \tag{69}$$

For any sets of real numbers A and B, and if $C = \{(x,y) : x \in A, y \in B\}$, it follows from equation~67 that

$$P((X \in A, Y \in B) \in C) = \int_B \int_A f(x,y)\, dx\, dy \tag{70}$$

Note that

$$F(a,b) = P(X \in (-\infty, a]), Y \in (-\infty, b])) = \int_{-\infty}^{b} \int_{-\infty}^{a} f(x,y)\, dx\, dy \tag{71}$$

Differentiating, we get the joint pdf:

$$f(a,b) = \frac{\partial^2}{\partial a \partial b} F(a,b) \tag{72}$$

One way to understand the joint PDF:

$$P(a < X < a + da, b < Y < b + db) = \int_{b}^{d+db} \int_{a}^{a+da} f(x,y)\, dx\, dy \approx f(a,b) da db \tag{73}$$

Hence, $f(x,y)$ is a measure of how probable it is that the random vector (X,Y) will be near (a,b).

1.5.3 Marginal probability distribution functions

If X and Y are jointly continuous, they are individually continuous, and their PDFs are:

$$\begin{aligned}P(X \in A) =& P(X \in A, Y \in (-\infty,\infty)) \\ =& \int_A \int_{-\infty}^{\infty} f(x,y)\,dy\,dx \\ =& \int_A f_X(x)\,dx\end{aligned} \tag{74}$$

where

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)\,dy \tag{75}$$

Similarly:

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)\,dx \tag{76}$$

1.5.4 Independent random variables

Random variables X and Y are independent iff, for any two sets of real numbers A and B:

$$P(X \in A, Y \in B) = P(X \in A)P(Y \in B) \tag{77}$$

In the jointly continuous case:

$$f(x,y) = f_X(x)f_Y(y) \quad \text{for all } x,y \tag{78}$$

A necessary and sufficient condition for the random variables X and Y to be independent is for their joint probability density function (or joint probability mass function in the discrete case) $f(x,y)$ to factor into two terms, one depending only on x and the other depending only on y. %This can be stated as a proposition:

Example from Kerns:
Let the joint PDF of (X,Y) be given by

$$f_{X,Y}(x,y) = \frac{6}{5}\left(x+y^2\right), \quad 0 < x < 1,\ 0 < y < 1.$$

The marginal PDF of X is

$$\begin{aligned} f_X(x) &= \int_0^1 \frac{6}{5}\left(x+y^2\right) \mathrm{d}y, \\ &= \frac{6}{5}\left(xy+\frac{y^3}{3}\right)\Big|_{y=0}^{1}, \\ &= \frac{6}{5}\left(x+\frac{1}{3}\right), \end{aligned}$$

for $0 < x < 1$, and the marginal PDF of Y is

$$\begin{aligned} f_Y(y) &= \int_0^1 \frac{6}{5}\left(x+y^2\right) \mathrm{d}x, \\ &= \frac{6}{5}\left(\frac{x^2}{2}+xy^2\right)\Big|_{x=0}^{1}, \\ &= \frac{6}{5}\left(\frac{1}{2}+y^2\right), \end{aligned}$$

for $0 < y < 1$.

In this example the joint support set was a rectangle $[0,1]\times[0,1]$, but it turns out that X and Y are not independent. This is because $\frac{6}{5}\left(x+y^2\right)$ cannot be stated as a product of two terms $(f_X(x)f_Y(y))$.

1.5.5 Sums of independent random variables

(Taken nearly verbatim from Ross 2002.)

Suppose that X and Y are independent, continuous random variables having probability density functions f_X and f_Y. The cumulative distribution function of $X+Y$ is obtained as follows:

$$\begin{aligned} F_{X+Y}(a) =& P(X+Y \leq a) \\ =& \iint\limits_{x+y\leq a} f_{XY}(x,y)\,dx\,dy \\ =& \iint\limits_{x+y\leq a} f_X(x)f_Y(y)\,dx\,dy \\ =& \int_{-\infty}^{\infty}\int_{-\infty}^{a-y} f_X(x)f_Y(y)\,dx\,dy \\ =& \int_{-\infty}^{\infty}\int_{-\infty}^{a-y} f_X(x)\,dx f_Y(y)\,dy \\ =& \int_{-\infty}^{\infty} F_X(a-y)f_Y(y)\,dy \end{aligned} \tag{79}$$

The CDF F_{X+Y} is the **convolution** of the distributions F_X and F_Y.

If we differentiate the above equation, we get the pdf f_{X+Y}:

$$\begin{aligned} f_{X+Y} =& \frac{d}{dx}\int_{-\infty}^{\infty} F_X(a-y) f_Y(y)\,dy \\ =& \int_{-\infty}^{\infty} \frac{d}{dx} F_X(a-y) f_Y(y)\,dy \\ =& \int_{-\infty}^{\infty} f_X(a-y) f_Y(y)\,dy \end{aligned} \tag{80}$$

1.5.6 Conditional distributions

1.5.6.1 Discrete case

Recall that the conditional probability of B given A, denoted $\mathbb{P}(B \mid A)$, is defined by

$$\mathbb{P}(B \mid A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(A)}, \quad \text{if } \mathbb{P}(A) > 0. \tag{81}$$

If X and Y are discrete random variables, then we can define the conditional PMF of X given that $Y = y$ as follows:

$$\begin{aligned} p_{X|Y}(x \mid y) =& P(X = x \mid Y = y) \\ =& \frac{P(X = x, Y = y)}{P(Y = y)} \\ =& \frac{p(x,y)}{p_Y(y)} \end{aligned} \tag{82}$$

for all values of y where $p_Y(y) = P(Y = y) > 0$.

The **conditional cumulative distribution function** of X given $Y = y$ is defined, for all y such that $p_Y(y) > 0$, as follows:

$$\begin{aligned} F_{X|Y} =& P(X \leq x \mid Y = y) \\ =& \sum_{a \leq x} p_{X|Y}(a \mid y) \end{aligned} \tag{83}$$

If X and Y are independent then

$$p_{X|Y}(x \mid y) = P(X = x) = p_X(x) \tag{84}$$

See the examples starting p. 264 of Ross (2002).

1.5.6.2 Continuous case

(Taken almost verbatim from Ross 2002.)

If X and Y have a joint probability density function $f(x,y)$, then the conditional probability density function of X given that $Y = y$ is defined, for all values of y such that $f_Y(y) > 0$,by

$$f_{X|Y}(x \mid y) = \frac{f(x,y)}{f_Y(y)} \tag{85}$$

We can understand this definition by considering what $f_{X|Y}(x \mid y)\,dx$ amounts to:

$$\begin{aligned} f_{X|Y}(x \mid y)\,dx =& \frac{f(x,y)}{f_Y(y)}\frac{dxdy}{dy} \\ =& \frac{f(x,y)dxdy}{f_Y(y)dy} \\ =& \frac{P(x < X < x+dx, y < Y < y+dy)}{y < P < y+dy} \end{aligned} \tag{86}$$

1.5.7 Covariance and correlation

There are two very special cases of joint expectation: the **covariance** and the **correlation**. These are measures which help us quantify the dependence between X and Y.
Definition 2. *The **covariance** of X and Y is*

$$Cov(X,Y) = \mathbb{E}(X - \mathbb{E}X)(Y - \mathbb{E}Y). \tag{87}$$

Shortcut formula for covariance:

$$\mathrm{Cov}(X,Y) = \mathbb{E}(XY) - (\mathbb{E}X)(\mathbb{E}Y). \tag{88}$$

The Pearson product moment correlation between X and Y is the covariance between X and Y rescaled to fall in the interval $[-1,1]$. It is formally defined by

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}. \tag{89}$$

The correlation is usually denoted by $\rho_{X,Y}$ or simply ρ if the random variables are clear from context. There are some important facts about the correlation coefficient:

1. The range of correlation is $-1 \le \rho_{X,Y} \le 1$.
2. Equality holds above ($\rho_{X,Y} = \pm 1$) if and only if Y is a linear function of X with probability one.

1.5.7.1 Variance-covariance matrices

The variances in a multivariate distribution will be composed of

- variances for each random variable
- covariances between pairs of random variables, which includes some correlation ρ between pairs of random variables

E.g., for a bivariate distribution with random variables u_0 and u_1, this information is expressed in a so-called variance-covariance matrix.

$$\Sigma = \begin{pmatrix} \sigma_{u0}^2 & \rho\sigma_{u0}\sigma_{u1} \\ \rho\sigma_{u0}\sigma_{u1} & \sigma_{u1}^2 \end{pmatrix} \tag{90}$$

the covariance $Cov(X,Y)$ between two variables X and Y is defined as the product of their correlation ρ and their standard deviations σ_X and σ_Y, such that, $Cov(X,Y) = \rho\sigma_X\sigma_Y$.

$$\Sigma_u = \begin{pmatrix} \sigma_{u_0}^2 & \rho_u\sigma_{u_0}\sigma_{u_1} \\ \rho_u\sigma_{u_0}\sigma_{u_1} & \sigma_{u_1}^2 \end{pmatrix} \tag{91}$$

The covariance matrix can be decomposed into a matrix of standard deviations and a correlation matrix. The correlation matrix looks like this:

$$\rho_u = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \tag{92}$$

This means that we can decompose the covariance matrix into three parts:

$$\Sigma = \begin{pmatrix} \sigma_{u_0} & 0 \\ 0 & \sigma_{u_1} \end{pmatrix} \begin{pmatrix} 1 & \rho_u \\ \rho_u & 1 \end{pmatrix} \begin{pmatrix} \sigma_{u_0} & 0 \\ 0 & \sigma_{u_1} \end{pmatrix} \tag{93}$$

1.5.7.2 The Cholesky decomposition

Assume that we have an $n \times n$ correlation matrix ρ_u. We can obtain a square root of this matrix, which is called the lower triangular matrix $\mathbf{L_u}$. If we multiply $\mathbf{L_u}$ with its transpose, we get the correlation matrix:

$\mathbf{L_u}\mathbf{L_u}^T = \rho_u$.

The matrix $\mathbf{L_u}$ is called the Cholesky factor of $\rho_\mathbf{u}$:

$$\mathbf{L_u} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \tag{94}$$

As an example, let's assume a correlation of 0.8.

```r
rho <- 0.8
#Correlation matrix
rho_u <- matrix(c(1,rho,rho,1),ncol=2)

# Cholesky factor:
L_u <- t(chol(rho_u))
# Verify that we recover rho_u, %*%
# indicates matrix multiplication (=! element-wise multiplication)
L_u %*% t(L_u)
```

```
##      [,1] [,2]
## [1,]  1.0  0.8
## [2,]  0.8  1.0
```

What is the Cholesky factor useful for? We can use the Cholesky factor to generate correlated random variables in the following way.

Step 1. We generate uncorrelated values that will be associated with random variable u from a $Normal(0,1)$. In our case we have u_0 and u_1, so we will generate:

$$z_{u_0} \sim Normal(0,1)$$
$$z_{u_1} \sim Normal(0,1)$$

For example, assuming only 10 data points:

```r
N_subj <- 10
z_u0 <- rnorm(N_subj,0,1)
z_u1 <- rnorm(N_subj,0,1)
```

Step 2. By multiplying the Cholesky factor by our z's we generate a matrix of correlated variables (with standard deviation 1).

A very informal explanation of why this works is that we are making the variable that corresponds to the slope to be a function of a scaled version of the intercept.

For example:

```r
# matrix z_u
z_u <- matrix(c(z_u0,z_u1),ncol=N_subj,byrow=TRUE)

L_u %*% z_u
```

```
##         [,1]   [,2]   [,3]   [,4]  [,5]    [,6]   [,7]   [,8]   [,9] [,10]
## [1,]  0.0488 -0.223 -0.555 0.2129 0.531  0.0764 -0.495 -0.019 -0.997 -1.04
## [2,] -1.0103 -1.505 -1.083 0.0878 1.738 -0.1657 -0.475 -0.173 -0.839 -1.45
```

Step 3. The last step is to scale the previous matrix to the desired standard deviation. We define the diagonalized matrix as before:

$$\begin{pmatrix} \tau_{u_0} & 0 \\ 0 & \tau_{u_1} \end{pmatrix}$$

For example:

```
tau_u0 <- .2
tau_u1 <- .01
(diag_matrix_tau <- matrix(c(tau_u0,0,0,tau_u1),ncol=2))
```

```
##      [,1] [,2]
## [1,]  0.2 0.00
## [2,]  0.0 0.01
```

We pre-multiply it by the correlated variables with SD of 1 from before:

For example:

```
u <- diag_matrix_tau %*% L_u %*% z_u

# We should find that the rows have approx. correlation 0.8
cor(u[1,],u[2,])
```

```
## [1] 0.72
```

```
# We should be able to recover the tau's as well:
sd(u[1,])
```

```
## [1] 0.104
```

```
sd(u[2,])
```

```
## [1] 0.00955
```

This process of generating correlated random variables will be useful when we start fitting hierarchical linear models: we will define a prior on the Cholesky factor of the correlation matrix, and then assemble the variance-covariance matrix as shown above, and then generate correlated random variables (varying intercepts and slopes).

1.5.8 Multivariate normal distributions

This is a very important distribution that we will need in linear mixed models.

Consider the bivariate case first. Suppose we have two univariate random variables $U0 \sim Normal(\mu_0, \sigma_{u0})$ and $U1 \sim Normal(\mu_1, \sigma_{u1})$ that have covariance $\rho\sigma_{u0}\sigma_{u1}$. We write this as:

$$\begin{pmatrix} U_0 \\ U_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_0 \\ \mu_1 \end{pmatrix}, \Sigma\right) \tag{95}$$

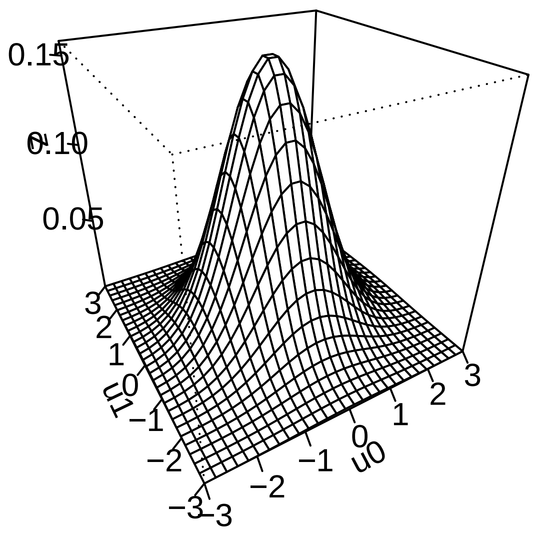


Figure 7: Visualization of two uncorrelated random variables.

The variances and covariances between the two random variables are described by a 2×2 variance-covariance matrix. The diagonals have the variances, and the off-diagonals have the covariance between the two random variables.

$$\Sigma = \begin{pmatrix} \sigma_{u0}^2 & \rho\sigma_{u0}\sigma_{u1} \\ \rho\sigma_{u0}\sigma_{u1} & \sigma_{u1}^2 \end{pmatrix} \tag{96}$$

We develop some graphical intuition about bivariate distributions next.

1.5.8.1 Graphical intuition for the bivariate case

If we have two independent random variables U0, U1, and we examine their joint distribution, we can plot a 3-d plot which shows, u0, u1, and f(u0,u1).

Bivariate distribution with no correlation (independent random variables): E.g., $u0 \sim N(0,1)$ and $u1 \sim N(0,1)$, with two independent random variables. See Figure 7.

```r
library(mvtnorm)
u0 <- u1 <- seq(from = -3, to = 3, length.out = 30)
Sigma1<-diag(2)
f <- function(u0, u1) dmvnorm(cbind(u0, u1), mean = c(0, 0),sigma = Sigma1)
z <- outer(u0, u1, FUN = f)
```

```r
persp(u0, u1, z, theta = -30, phi = 30, ticktype = "detailed")
```

Bivariate distribution with positive correlation: see Figure 8.

```r
Sigma2<-matrix(c(1,.6,.6,1),byrow=FALSE,ncol=2)
f <- function(u0, u1) dmvnorm(cbind(u0, u1), mean = c(0, 0),sigma = Sigma2)
z <- outer(u0, u1, FUN = f)
```

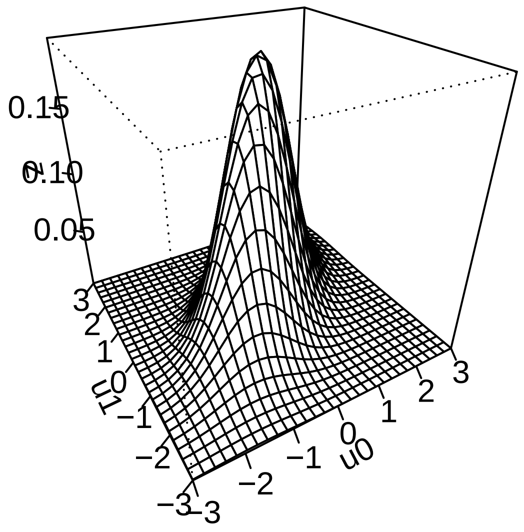


Figure 8: Visualization of two positively correlated random variables.

```r
persp(u0, u1, z, theta = -30, phi = 30, ticktype = "detailed")
```

Bivariate distribution with negative correlation: see Figure 9.

```r
Sigma3<-matrix(c(1,-.6,-.6,1),byrow=FALSE,ncol=2)
f <- function(u0, u1) dmvnorm(cbind(u0, u1), mean = c(0, 0),sigma = Sigma3)
z <- outer(u0, u1, FUN = f)
```

```r
persp(u0, u1, z, theta = -30, phi = 30, ticktype = "detailed")
```

1.5.8.2 Visualizing conditional distributions in a bivariate

You can run the following code to get a visualization of what a conditional distribution looks like when we take “slices” from the conditioning random variable:

```r
bivn<-mvrnorm(1000,mu=c(0,1),Sigma=matrix(c(1,0,0,2),2))
bivn.kde<-kde2d(bivn[,1],bivn[,2],n=50)

for(i in 1:50){
  plot(bivn.kde$z[i,1:50],type="l",ylim=c(0,0.1))
  Sys.sleep(.5)
}
```

If you run this code, you will see “slices” from the bivariate distribution.

1.5.8.3 Formal definition of the multivariate normal

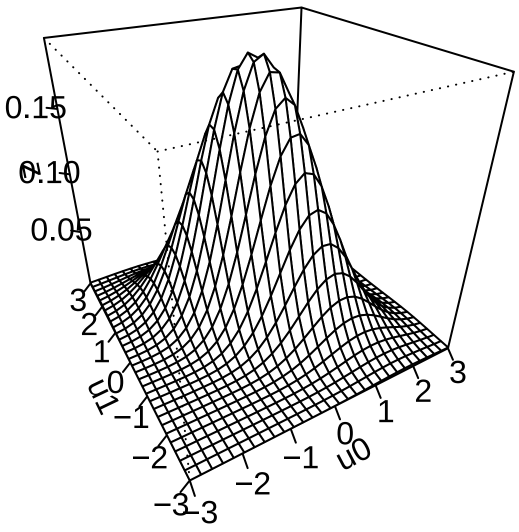


Figure 9: Visualization of two negatively correlated random variables.

Having acquired a graphical intuition, we turn to the formal definitions. Recall that in the univariate normal distribution:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e\{-\frac{(\frac{(x-\mu)}{\sigma})^2}{2}\} \quad -\infty < x < \infty \tag{97}$$

We can write the power of the exponential as:

$$\left(\frac{(x-\mu)}{\sigma}\right)^2 = (x-\mu)(x-\mu)(\sigma^2)^{-1} = (x-\mu)(\sigma^2)^{-1}(x-\mu) = Q \tag{98}$$

Generalizing this to the multivariate case:

$$Q = (x-\mu)'\Sigma^{-1}(x-\mu) \tag{99}$$

Σ is a variance-covariance matrix, and Σ^{-1} is its inverse.

So, for the multivariate case:

$$f(x) = \frac{1}{\sqrt{2\pi det(\Sigma)}} e\{-Q/2\} \quad -\infty < x_i < \infty, i = 1, \dots, n \tag{100}$$

$det(\Sigma)$ is the determinant of the matrix.

Properties of the multivariate normal (MVN) X:

- Linear combinations of X are normal distributions.
- All subsets of X's components have a normal distribution.

- Zero covariance implies independent distributions.
- Conditional distributions are normal.

1.6 Maximum likelihood estimation

1.6.1 Discrete case

Suppose the observed independent sample values are $x_1, x_2, \ldots, x_n$ from some random variable with pmf $P(\cdot)$ that has a parameter θ. The probability of getting these particular values is

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = f(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n; \theta) \tag{101}$$

i.e., the function f is the value of the joint probability **distribution** of the random variables $X_1, \ldots, X_n$ at $X_1 = x_1, \ldots, X_n = x_n$. Since the sample values have been observed and are fixed, $f(x_1, \ldots, x_n; \theta)$ is a function of θ. The function f is called a **likelihood function**.

1.6.2 Continuous case

Here, f is the joint probability **density**, the rest is the same as above.
Definition 3. *If $x_1, x_2, \ldots, x_n$ are the values of a random sample from a population with parameter θ, the **likelihood function** of the sample is given by*

$$L(\theta) = f(x_1, x_2, \ldots, x_n; \theta) \tag{102}$$

for values of θ within a given domain. Here, $f(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n; \theta) = f(x_1; \theta) \cdot f(x_2; \theta) \ldots f(x_n; \theta)$ is the joint likelihood of the random variables $X_1, \ldots, X_n$ at $X_1 = x_1, \ldots, X_n = x_n$.

So, the method of maximum likelihood consists of maximizing the likelihood function with respect to θ. The value of θ that maximizes the likelihood function is the **MLE** (maximum likelihood estimate) of θ.

1.6.3 Finding maximum likelihood estimates for different distributions

Example 1

Let X_i, $i = 1, \ldots, n$ be a random variable with PDF $f(x; \sigma) = \frac{1}{2\sigma} exp(-\frac{|x|}{\sigma})$. Find $\hat{\sigma}$, the MLE of σ.

$$L(\sigma) = \prod f(x_i; \sigma) = \frac{1}{(2\sigma)^n} exp(-\sum \frac{|x_i|}{\sigma}) \tag{103}$$

Let ℓ be log likelihood (log lik). The log likelihood is much easier to work with, because products become sums. Then:

$$\ell(x;\sigma) = \sum \left[-\log 2 - \log \sigma - \frac{\mid x_i \mid}{\sigma} \right] \tag{104}$$

Differentiating and equating to zero to find maximum:

$$\ell'(\sigma) = \sum \left[-\frac{1}{\sigma} + \frac{\mid x_i \mid}{\sigma^2} \right] = -\frac{n}{\sigma} + \frac{\mid x_i \mid}{\sigma^2} = 0 \tag{105}$$

Rearranging the above, the MLE for σ is:

$$\hat{\sigma} = \frac{\sum \mid x_i \mid}{n} \tag{106}$$

Example 2: Poisson

$$\begin{aligned} L(\mu;x) \qquad &= \prod \tfrac{\exp^{-\mu} \mu^{x_i}}{x_i!} && (107)\\ &= \exp^{-\mu} \mu^{\sum x_i} \tfrac{1}{\prod x_i!} && (108) \end{aligned}$$

Log lik:

$$\ell(\mu;x) = -n\mu + \sum x_i \log \mu - \sum \log y! \tag{109}$$

Differentiating:

$$\ell'(\mu) = -n + \frac{\sum x_i}{\mu} = 0 \tag{110}$$

Therefore:

$$\hat{\lambda} = \frac{\sum x_i}{n} \tag{111}$$

Example 3: Binomial

$$L(\theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x} \tag{112}$$

Log lik:

$$\ell(\theta) = \log \binom{n}{x} + x \log \theta + (n-x) \log(1-\theta) \tag{113}$$

Differentiating:

$$\ell'(\theta) = \frac{x}{\theta} - \frac{n-x}{1-\theta} = 0 \tag{114}$$

Thus:

$$\hat{\theta} = \frac{x}{n} \tag{115}$$

Example 4: Normal

Let $X_1, \dots, X_n$ constitute a random variable of size n from a normal population with mean μ and variance σ^2, find joint maximum likelihood estimates of these two parameters.

$$L(\mu;\sigma^2) \quad = \prod N(x_i;\mu,\sigma) \tag{116}$$

$$= (\tfrac{1}{2\pi\sigma^2})^{n/2} \exp(-\tfrac{1}{2\sigma^2} \sum (x_i-\mu)^2) \tag{117}$$

$$\tag{118}$$

Taking logs and differentiating with respect to μ and σ^2, we get:

$$\hat{\mu} = \frac{1}{n}\sum x_i = \bar{x} \tag{119}$$

and

$$\hat{\sigma}^2 = \frac{1}{n}\sum (x_i - \bar{x})^2 \tag{120}$$

1.6.4 Visualizing likelihood and maximum log likelihood for normal

For simplicity consider the case where $N(\mu = 0, \sigma^2 = 1)$.

```
op<-par(mfrow=c(1,2),pty="s")
plot(function(x) dnorm(x,log=FALSE), -3, 3,
      main = "Normal density",
             ylab="density",xlab="X")
abline(h=0.4)
plot(function(x) dnorm(x,log=TRUE), -3, 3,
      main = "Normal density (log)",
             ylab="density",xlab="X")
abline(h=log(0.4))
```

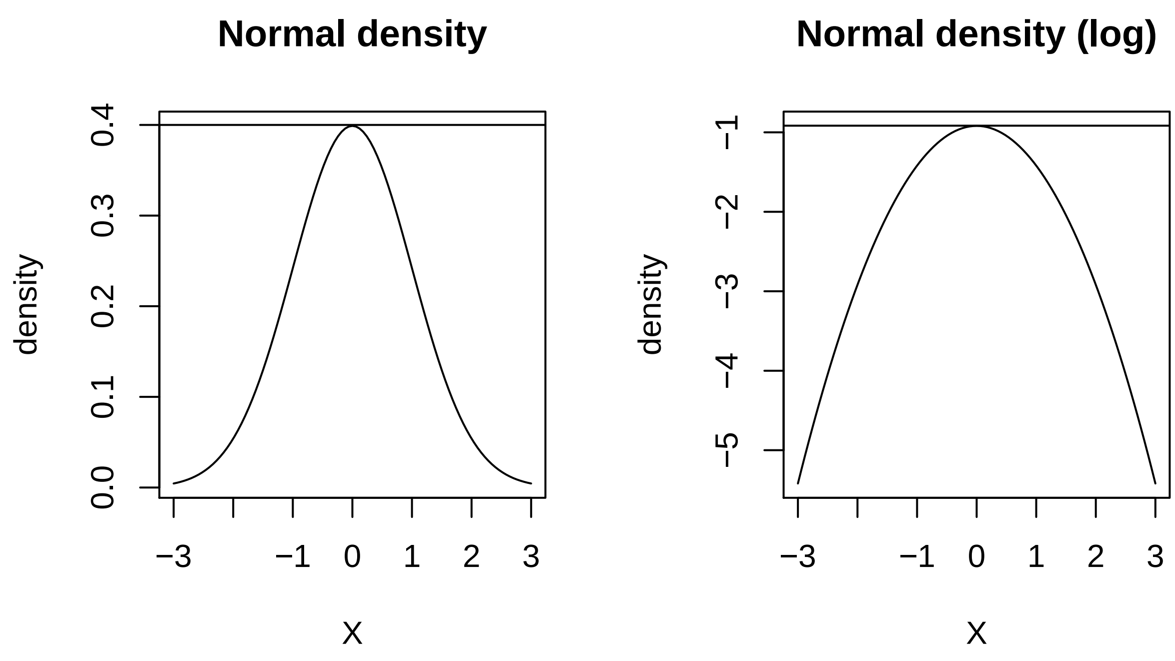


Figure 10: Maximum likelihood and log likelihood.

1.6.5 MLE using R

1.6.5.1 One-parameter case

Estimating θ for the binomial distribution: Let's assume we have the result of 10 coin tosses. We know that the MLE is the number of successes divided by the sample size:

```
x<-rbinom(10,1,prob=0.5)
sum(x)/length(x)
```

```
## [1] 0.1
```

We will now get this number using MLE. We do it numerically to illustrate the principle. First, we define a negative log likelihood function for the binomial. Negative because the function we will use to optimize does minimization by default, so we just flip the sign on the log likelihood to convert the maximum to a minimum.

```
negllbinom <- function(p, x){
  -sum(dbinom(x, size = 1, prob = p,log=T))
}
```

Then we run the optimization function:

```
optimize(negllbinom,
         interval = c(0, 1),
         x = x)
```

```
## $minimum
## [1] 0.1
##
## $objective
## [1] 3.25
```

1.6.5.2 Two-parameter case

Here is an example of MLE using R. Note that in this example, we could have analytically figured out the MLEs. Instead, we are doing this numerically. The advantage of the numerical approach becomes obvious when the analytical way is closed to us.

Assume that you have some data that was generated from a normal distribution, with mean 500, and standard deviation 50. Let's say you have 100 data points.

```
data<-rnorm(100,mean=500,sd=50)
```

Let's assume we don't know what the mean and standard deviation are. Now, of course you know how to estimate these using the standard formulas. But right now we are going to estimate them using MLE.

We first write down the negation of the log likelihood function. We take the negation because the optimization function we will be using (see below) does minimization by default, so to get the maximum with the default setting, we just change the sign.

The function `nllh.normal` takes a vector `theta` of parameter values, and a data frame `data`.

```
nllh.normal<-function(theta,data){
  ## decompose the parameter vector to
  ## its two parameters:
  m<-theta[1]
  s<-theta[2]
  ## read in data
  x <- data
  n<-length(x)
  ## log likelihood:
  logl<- sum(dnorm(x,mean=m,sd=s,log=TRUE))
  ## return negative log likelihood:
  -logl
  }
```

Here is the negative log lik for mean = 40, sd 4, and for mean = 800 and sd 4:

```
nllh.normal(theta=c(40,4),data)
```

```
## [1] 655888
```

```
nllh.normal(theta=c(800,4),data)
```

```
## [1] 299015
```

As we would expect, the negative log lik for mean 500 and sd 50 is much smaller (due to the sign change) than the two log liks above:

```
nllh.normal(theta=c(500,50),data)
```

```
## [1] 537
```

Basically, you could sit here forever, playing with combinations of values for mean and sd to find the combination that gives the optimal log likelihood. R has an optimization function that does this for you. We have to specify some sensible starting values:

```
opt.vals.default<-optim(theta<-c(700,40),
                        nllh.normal,
      data=data,
      hessian=TRUE)
```

Finally, we print out the estimated parameter values that maximize the likelihood:

```
opt.vals.default$par
```

```
## [1] 495.1  51.6
```

Knowledge of maximum likelihood estimation will be needed in the next chapter.

2 Introduction to Bayesian data analysis

2.1 Introduction

Recall Bayes' rule:

When A and B are observable events, we can state the rule as follows:

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)} \tag{121}$$

Note that $p(\cdot)$ is the probability of an event.

When looking at probability distributions, we will encounter the rule in the following form. Let y be a vector of data.

$$f(\theta \mid y) = \frac{f(y \mid \theta)f(\theta)}{f(y)} \tag{122}$$

Here, $f(\cdot)$ is a probability density, not the probability of a single event. $f(y)$ is called a "normalizing constant" (recall the earlier discussion in the Foundations chapter), which makes the left-hand side a probability distribution. Without the normalizing constant, we have the relationship:

$$f(\theta \mid y) \propto f(y \mid \theta)f(\theta) \tag{123}$$

Note that the likelihood is proportional to $f(y \mid \theta)$, and that's why we can refer to $f(y \mid \theta)$ as the likelihood in the following Bayesian incantation:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{124}$$

Our central goal is going to be to derive the posterior distribution and then summarize its properties (mean, median, 95% confidence intervals, etc.).

Usually, we don't need the normalizing constant to understand the properties of the posterior distribution. That's why Bayes' theorem is often stated in terms of the proportionality shown above.

Two examples will clarify how we can use Bayes' rule to obtain the posterior. Both examples involve so-called conjugate priors, which are defined as follows:
Definition 4. *Given the likelihood $f(x \mid \theta)$, if the prior $f(\theta)$ results in a posterior $f(\theta \mid x)$ that has the same form as $f(\theta)$, then we call $f(\theta)$ a conjugate prior.*

2.2 Example 1: Binomial Likelihood, Beta prior, Beta posterior

This is a contrived example, just meant to provide us with a smooth entry into Bayesian data analysis. Suppose that an individual with aphasia answered 46 out of 100 questions correctly in a particular sentence comprehension task. The research question is, what is the probability that their average response is greater than 0.5, i.e., above chance.

The likelihood function will tell us $P(\text{data} \mid \theta)$:

```r
dbinom(46, 100, 0.5)
```

```
## [1] 0.058
```

Note that

$$P(\text{data} \mid \theta) \propto \theta^{46}(1-\theta)^{54} \tag{125}$$

So, to get the posterior, we just need to work out a prior distribution $f(\theta)$.

$$f(\theta \mid \text{data}) \propto f(\text{data} \mid \theta) f(\theta) \tag{126}$$

For the prior, we need a distribution that can represent our uncertainty about the probabiliy θ of success. The Beta distribution is commonly used as prior for proportions. We say that the Beta distribution is conjugate to the binomial density; i.e., the two densities have similar functional forms.

The pdf is

$$f(x) = \begin{cases} \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

where

$$B(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\, dx$$

In R, we write $X \sim \texttt{beta}(\texttt{shape1} = \alpha, \texttt{shape2} = \beta)$. The associated R function is $\texttt{dbeta}(\texttt{x}, \texttt{shape1}, \texttt{shape2})$.

The mean and variance are

$$E[X] = \frac{a}{a+b} \text{ and } Var(X) = \frac{ab}{(a+b)^2(a+b+1)}. \tag{127}$$

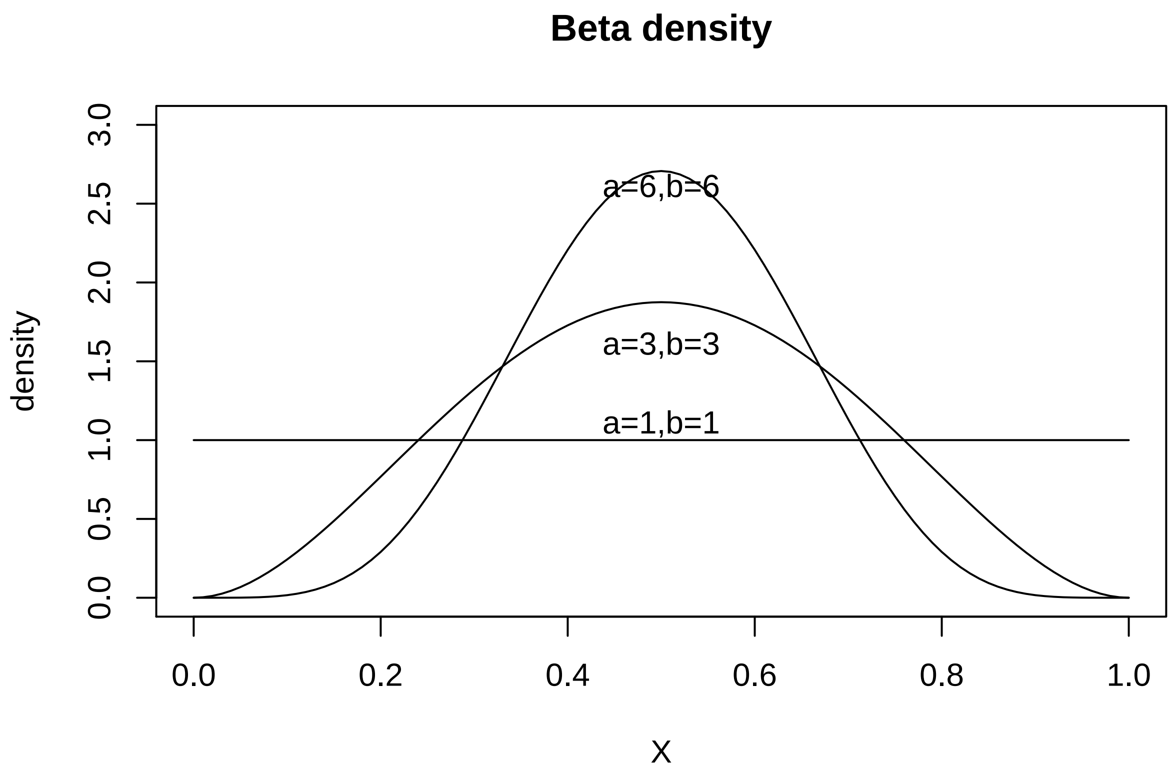


Figure 11: Examples of the beta distribution with different parameter values.

The Beta distribution's parameters a and b can be interpreted as (our beliefs about) prior successes and failures, and are called **hyperparameters**. Once we choose values for a and b, we can plot the Beta pdf. Here, we show the Beta pdf for three sets of values of a,b:

```
plot(function(x)
  dbeta(x,shape1=1,shape2=1), 0,1,
      main = "Beta density",
              ylab="density",xlab="X",ylim=c(0,3))

text(.5,1.1,"a=1,b=1")

plot(function(x)
  dbeta(x,shape1=3,shape2=3),0,1,add=T)
text(.5,1.6,"a=3,b=3")


plot(function(x)
  dbeta(x,shape1=6,shape2=6),0,1,add=T)
text(.5,2.6,"a=6,b=6")
```

As Figure 11 shows, as the a,b values are increased, the spread decreases.

If we don't have much prior information, we could use a=b=1; this gives us a uniform prior; this is called an uninformative prior or non-informative prior (although having no prior knowledge is, strictly speaking, not uninformative). If we have a lot of prior knowledge and/or a strong belief that θ has a particular value, we can use a larger a,b to reflect our greater certainty about the parameter. Notice that the larger our parameters a and b, the narrower the spread of the distribution; this makes sense because a larger sample size (a greater number of successes a, and a greater number of failures b) will lead to more precise estimates.

The central point is that the Beta distribution can be used to define the prior distribution of θ.

Just for the sake of illustration, let's take four different beta priors, each reflecting increasing certainty.

1. Beta(a=2,b=2)
2. Beta(a=3,b=3)
3. Beta(a=6,b=6)
4. Beta(a=21,b=21)

Each reflects a belief that $\theta = 0.5$, with varying degrees of (un)certainty. Now we just need to plug in the likelihood and the prior:

$$f(\theta \mid \text{data}) \propto f(\text{data} \mid \theta) f(\theta) \tag{128}$$

The four corresponding posterior distributios would be:

$$f(\theta \mid \text{data}) \propto [\theta^{46}(1-\theta)^{54}][\theta^{2-1}(1-\theta)^{2-1}] = \theta^{48-1}(1-\theta)^{56-1} \tag{129}$$

$$f(\theta \mid \text{data}) \propto [\theta^{46}(1-\theta)^{54}][\theta^{3-1}(1-\theta)^{3-1}] = \theta^{49-1}(1-\theta)^{57-1} \tag{130}$$

$$f(\theta \mid \text{data}) \propto [\theta^{46}(1-\theta)^{54}][\theta^{6-1}(1-\theta)^{6-1}] = \theta^{52-1}(1-\theta)^{60-1} \tag{131}$$

$$f(\theta \mid \text{data}) \propto [\theta^{46}(1-\theta)^{54}][\theta^{21-1}(1-\theta)^{21-1}] = \theta^{67-1}(1-\theta)^{75-1} \tag{132}$$

We can now visualize each of these triplets of priors, likelihoods and posteriors. Note that I use the beta to model the likelihood because this allows me to visualize all three (prior, lik., posterior) in the same plot. The likelihood function is actually as shown in Figure 12:

```r
theta=seq(0,1,by=0.01)

plot(theta,dbinom(x=46,size=100,prob=theta),
     type="l",main="Likelihood")
```

We can represent the likelihood in terms of the Beta as well, as shown in Figure 13:

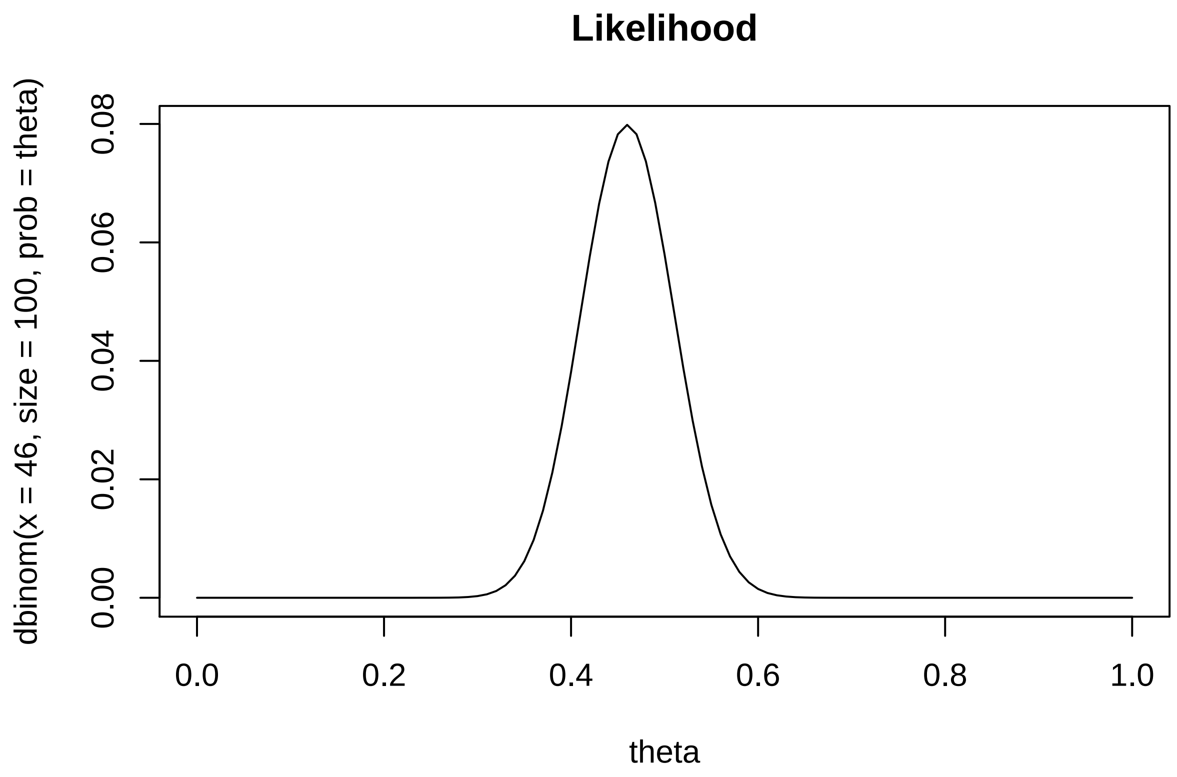


Figure 12: Binomial likelihood function.

```
plot(function(x)
  dbeta(x,shape1=46,shape2=54),0,1,
                ylab="",xlab="X")
```

2.3 Example 2: Poisson Likelihood, Gamma prior, Gamma posterior

This is also a contrived example. Suppose we are modeling the number of times that a speaker says the word "the" per day.

The number of times x that the word is uttered in one day can be modeled by a Poisson distribution:

$$f(x \mid \theta) = \frac{\exp(-\theta)\theta^x}{x!} \tag{133}$$

where the rate θ is unknown, and the numbers of utterances of the target word on each day are independent given θ.

We are told that the prior mean of θ is 100 and prior variance for θ is 225. This information could be based on the results of previous studies on the topic.

In order to visualize the prior, we first fit a Gamma density prior for θ based on the above information.

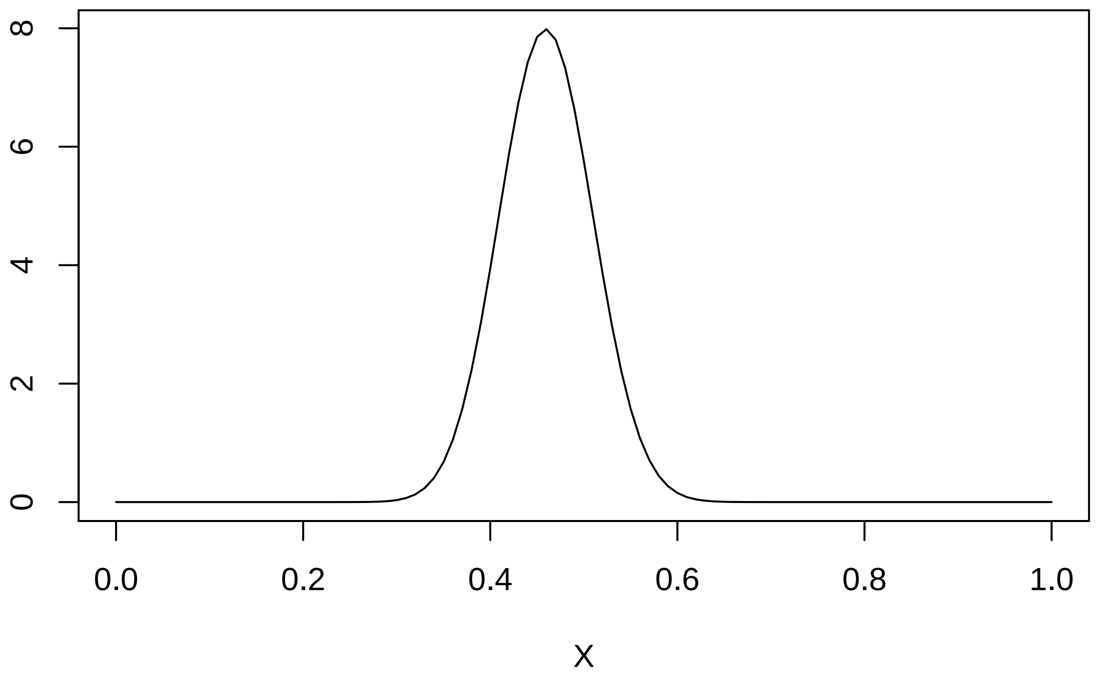


Figure 13: Using the beta distribution to represent a binomial.

Note that we know that for a Gamma density with parameters a, b, the mean is $\frac{a}{b}$ and the variance is $\frac{a}{b^2}$. Since we are given values for the mean and variance, we can solve for a,b, which gives us the Gamma density.

If $\frac{a}{b} = 100$ and $\frac{a}{b^2} = 225$, it follows that $a = 100 \times b = 225 \times b^2$ or $100 = 225 \times b$, i.e., $b = \frac{100}{225}$.

This means that $a = \frac{100\times 100}{225} = \frac{10000}{225}$. Therefore, the Gamma distribution for the prior is as shown below (also see Figure 14):

$$\theta \sim Gamma(\frac{10000}{225}, \frac{100}{225}) \tag{134}$$

```
x<-0:200
plot(x,dgamma(x,10000/225,100/225),type="l",lty=1,
     main="Gamma prior",ylab="density",
     cex.lab=2,cex.main=2,cex.axis=2)
```

A distribution for a prior is **conjugate** if, multiplied by the likelihood, it yields a posterior that has the distribution of the same family as the prior.

It turns out that the Gamma distribution is a conjugate prior for the Poisson distribution. For the Gamma distribution to be a conjugate prior for the Poisson, the posterior needs to have the general form of a Gamma distribution. We derive this conjugacy result below. The proof just involves very simple algebra.

Given that

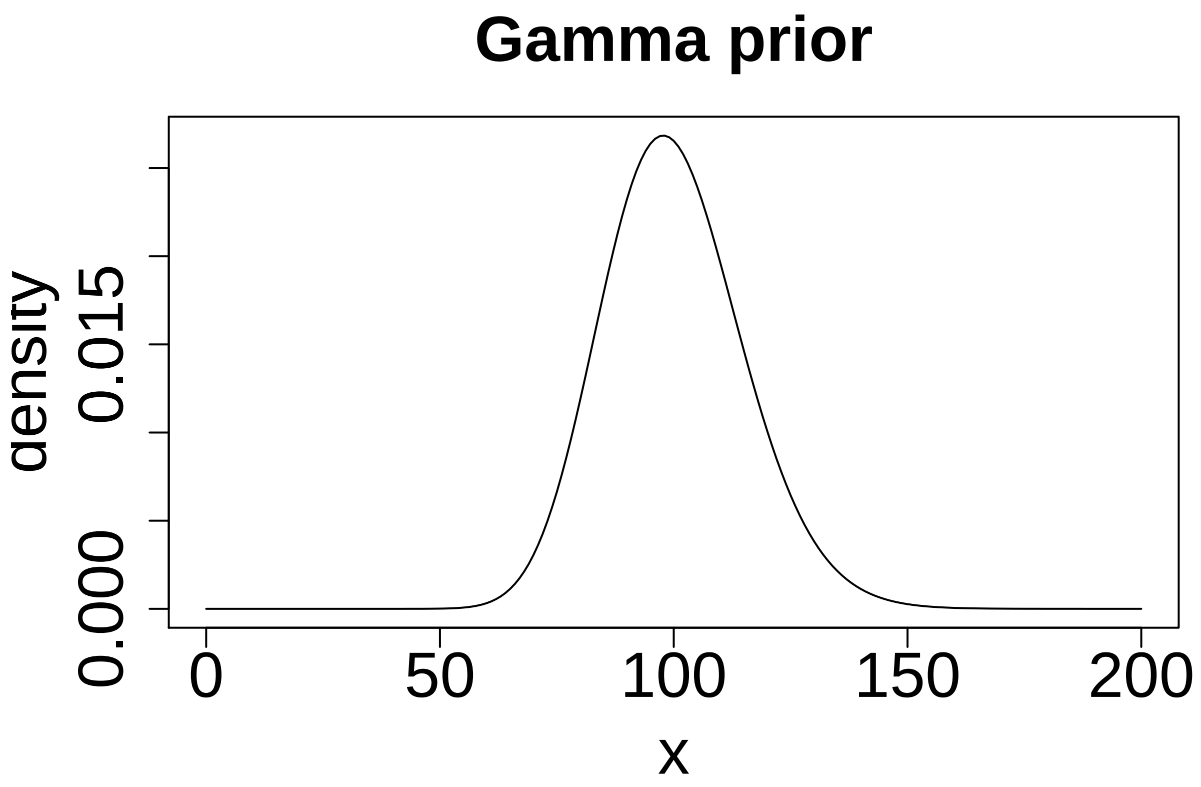


Figure 14: The Gamma prior for the parameter theta.

$$\text{Posterior} \propto \text{Prior} \times \text{Likelihood} \tag{135}$$

and given that the likelihood is:

$$\begin{aligned} L(\mathbf{x} \mid \theta) &= \prod_{i=1}^{n} \frac{\exp(-\theta)\theta^{x_i}}{x_i!} \\ &= \frac{\exp(-n\theta)\theta^{\sum_i^n x_i}}{\prod_{i=1}^{n} x_i!} \end{aligned} \tag{136}$$

we can compute the posterior as follows:

$$\text{Posterior} = \left[\frac{\exp(-n\theta)\theta^{\sum_i^n x_i}}{\prod_{i=1}^{n} x_i!}\right] \left[\frac{b^a \theta^{a-1} \exp(-b\theta)}{\Gamma(a)}\right] \tag{137}$$

We can disregard the terms $x!, \Gamma(a), b^a$, which do not involve θ, we have

$$\begin{aligned} \text{Posterior} &\propto \exp(-n\theta)\theta^{\sum_i^n x_i}\theta^{a-1}\exp(-b\theta) \\ &= \theta^{a-1+\sum_i^n x_i}\exp(-\theta(b+n)) \end{aligned} \tag{138}$$

We can now figure out the parameters of the posterior distribution, and show that it will be a Gamma distribution.

First, note that the Gamma distribution in general is $Gamma(a,b) \propto \theta^{a-1}\exp(-\theta b)$. So it's enough to state the above as a Gamma distribution with some parameters a, b.

If we equate $a^* - 1 = a - 1 + \sum_i^n x_i$ and $b^* = b + n$, we can rewrite the above as:

$$\theta^{a^*-1}\exp(-\theta b^*) \tag{139}$$

This means that $a^* = a + \sum_i^n x_i$ and $b^* = b + n$. We can find a constant k such that the above is a proper probability density function, i.e.:

$$\int_{-\infty}^{\infty} k\theta^{a^*-1}\exp(-\theta b^*) = 1 \tag{140}$$

Thus, the posterior has the form of a Gamma distribution with parameters $a^* = a + \sum_i^n x_i, b^* = b + n$. Hence the Gamma distribution is a conjugate prior for the Poisson.

2.3.1 Concrete example given data

Suppose we know that the number of "the" utterances is $115, 97, 79, 131$. We can derive the posterior distribution as follows.

The prior is Gamma(a=10000/225,b=100/225). The data are as given; this means that $\sum_i^n x_i = 422$ and sample size $n = 4$. It follows that the posterior is

$$\begin{aligned} Gamma(a^* = a + \sum_i^n x_i, b^* = b + n) =& Gamma(10000/225 + 422, 4 + 100/225) \\ =& Gamma(466.44, 4.44) \end{aligned} \tag{141}$$

The mean and variance of this distribution can be computed using the fact that the mean is $\frac{a*}{b*} = 466.44/4.44 = 104.95$ and the variance is $\frac{a*}{b*^2} = 466.44/4.44^2 = 23.66$.

```
### load data:
data<-c(115,97,79,131)

a.star<-function(a,data){
  return(a+sum(data))
}

b.star<-function(b,n){
  return(b+n)
}
```

```
new.a<-a.star(10000/225,data)
new.b<-b.star(100/225,length(data))

### post. mean
(post.mean<-new.a/new.b)
```

```
## [1] 105
```

```
### post. var:
(post.var<-new.a/(new.b^2))
```

```
## [1] 23.6
```

Now suppose you get one new data point:

```
new.data<-c(200)
```

We can compute the parameters of the posterior Gamma distributions using the function we wrote above:

```
new.a.2<-a.star(new.a,new.data)
new.b.2<-b.star(new.b,length(new.data))

### new mean
(new.post.mean<-new.a.2/new.b.2)
```

```
## [1] 122
```

```
### new var:
(new.post.var<-new.a.2/(new.b.2^2))
```

```
## [1] 22.5
```

Thus, new data can be used with the previous posterior distribution as prior, to compute the new posterior.

2.3.2 The posterior is a weighted mean of prior and likelihood

Here we come to a very important insight. We can express the posterior mean as a weighted sum of the prior mean and the maximum likelihood estimate of θ.

The posterior mean is:

$$\frac{a*}{b*} = \frac{a + \sum x_i}{n + b} \tag{142}$$

This can be rewritten as

$$\frac{a*}{b*} = \frac{a+n\bar{x}}{n+b} \tag{143}$$

Dividing both the numerator and denominator by b:

$$\frac{a*}{b*} = \frac{(a+n\bar{x})/b}{(n+b)/b} = \frac{a/b+n\bar{x}/b}{1+n/b} \tag{144}$$

Since a/b is the mean m of the prior, we can rewrite this as:

$$\frac{a/b+n\bar{x}/b}{1+n/b} = \frac{m+\frac{n}{b}\bar{x}}{1+\frac{n}{b}} \tag{145}$$

We can rewrite this as:

$$\frac{m+\frac{n}{b}\bar{x}}{1+\frac{n}{b}} = \frac{m\times 1}{1+\frac{n}{b}} + \frac{\frac{n}{b}\bar{x}}{1+\frac{n}{b}} \tag{146}$$

This is a weighted average: setting $w_1 = 1$ and $w_2 = \frac{n}{b}$, we can write the above as:

$$m\frac{w_1}{w_1+w_2} + \bar{x}\frac{w_2}{w_1+w_2} \tag{147}$$

A n approaches infinity, the weight on the prior mean m will tend towards 0, making the posterior mean approach the maximum likelihood estimate of the sample.

In general, in a Bayesian analysis, as sample size increases, the likelihood will dominate in determining the posterior mean.

Regarding variance, since the variance of the posterior is:

$$\frac{a*}{b*^2} = \frac{(a+n\bar{x})}{(n+b)^2} \tag{148}$$

as n approaches infinity, the posterior variance will approach zero: more data will reduce variance (uncertainty).

2.4 Summary: conjugate form analyses

We saw two examples where we can do the computations to derive the posterior using simple algebra. There are several other such simple cases. However, in realistic data analysis settings, we cannot specify the posterior distribution as a particular density. We can only specify the priors and the likelihood.

For such cases, we need to use MCMC sampling techniques so that we can sample from the posterior distributions of the parameters.

We will discuss two approaches for sampling:

- Gibbs sampling using inversion sampling
- Hamiltonian Monte Carlo

But before we do that, we introduce some basic ideas about Monte Carlo sampling and Markov chain processes.

2.5 MCMC sampling

2.5.1 Monte Carlo integration

The term Monte Carlo integration sounds fancy, but basically this amounts to sampling from a distribution f(x), and computing summaries like the mean. Formally, we calculate E(f(X)) by drawing samples $\{X_1, \dots, X_n\}$ and then approximating:

$$E(f(X)) \approx \frac{1}{n}\sum f(X) \tag{149}$$

For example:

```
x<-rnorm(1000,mean=0,sd=1)
mean(x)
```

```
## [1] -0.00955
```

We can increase sample size to as large as we want.

We can also compute quantities like $P(X < 1.96)$ by sampling:

```
mean((x<1.96))
```

```
## [1] 0.977
```

```
## theoretical value:
pnorm(1.96)
```

```
## [1] 0.975
```

So, we can compute summary statistics using simulation. However, if we only know up to proportionality the form of the distribution to sample from, how do we get these samples to summarize from? Monte Carlo Markov Chain (MCMC) methods provide that capability: they allow you to sample from distributions you only know up to proportionality.

2.5.2 Markov chain sampling

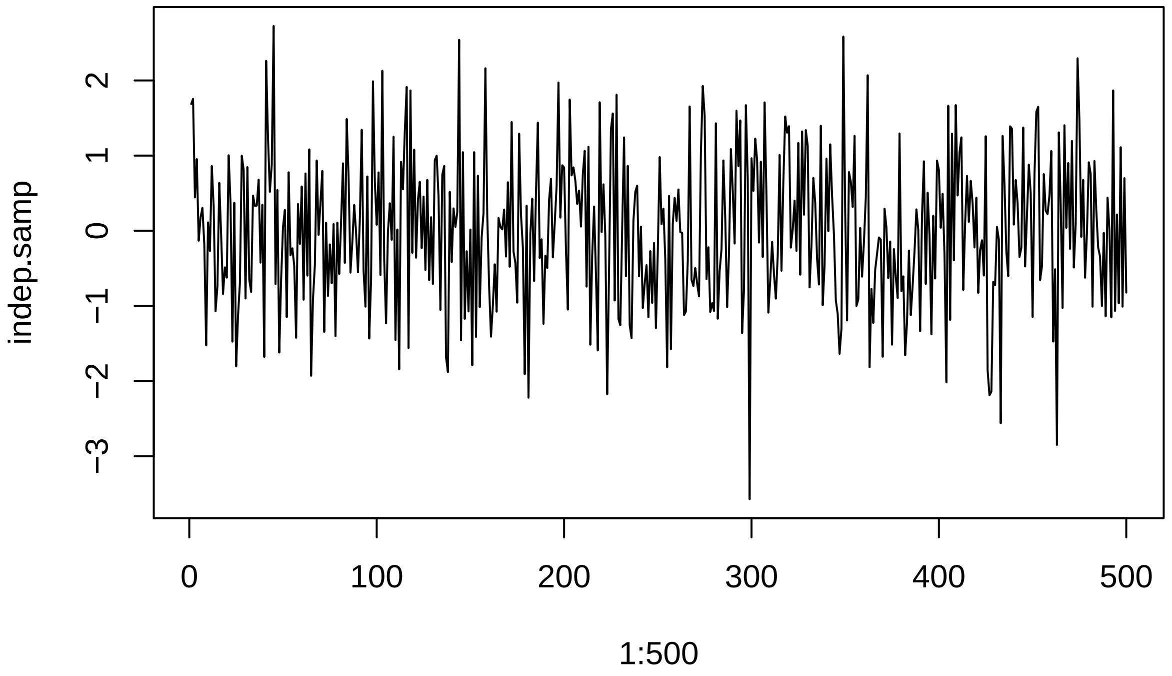
We have been doing non-Markov chain sampling when we started this course. Taking independent samples is an example of non-Markov chain sampling:

```
indep.samp<-rnorm(500,mean=0,sd=1)
head(indep.samp)
```

```
## [1]  1.686  1.754  0.443  0.953 -0.133  0.176
```

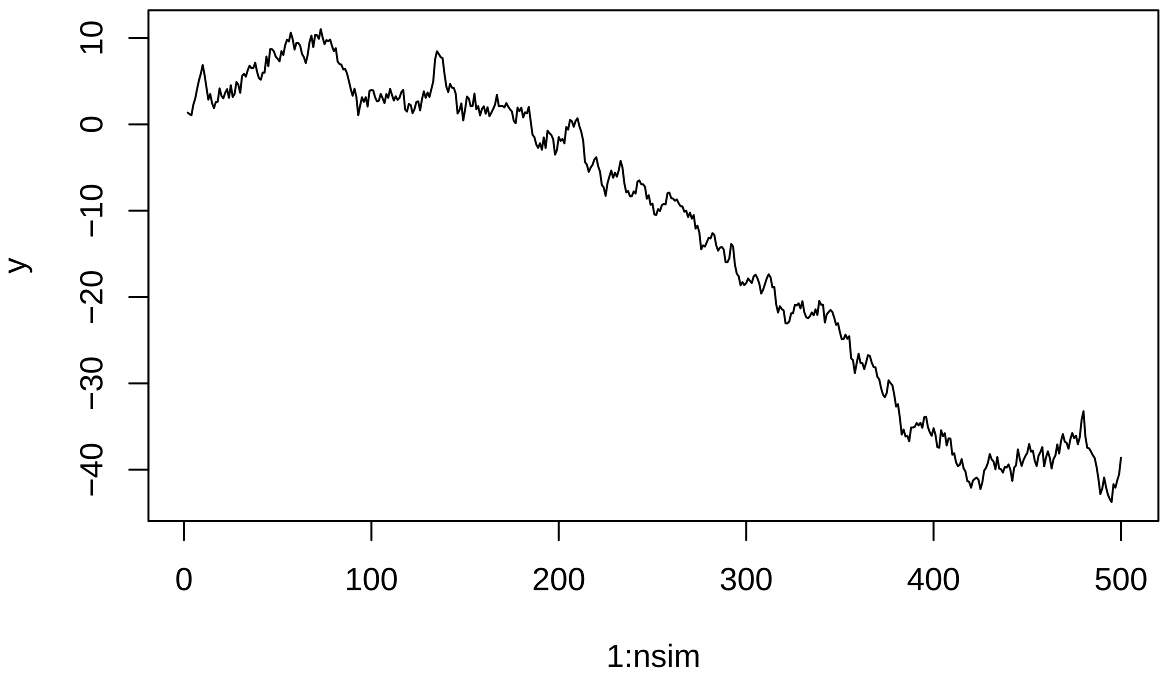
The vector of values sampled here are statistically independent.

```
plot(1:500,indep.samp,type="l")
```



If the current value influences the next one, we have a Markov chain. Here is a simple Markov chain: the i-th draw is dependent on the i-1 th draw:

```
nsim<-500
x<-rep(NA,nsim)
y<-rep(NA,nsim)
x[1]<-rnorm(1) ## initialize x
for(i in 2:nsim){
## draw i-th value based on i-1-th value:
y[i]<-rnorm(1,mean=x[i-1],sd=1)
x[i]<-y[i]
}
plot(1:nsim,y,type="l")
```



2.5.3 What is convergence in a Markov chain? An illustration using discrete Markov chains

A discrete Markov chain defines probabilistic movement from one state to the next. Suppose we have 6 states; a **transition matrix** can define the probabilities:

```
## Set up transition matrix:
T<-matrix(rep(0,36),nrow=6)
diag(T)<-0.5
offdiags<-c(rep(0.25,4),0.5)
for(i in 2:6){
T[i,i-1]<-offdiags[i-1]
}
offdiags2<-c(0.5,rep(0.25,4))
for(i in 1:5){
T[i,i+1]<-offdiags2[i]
}
T
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.50 0.50 0.00 0.00 0.00 0.00
## [2,] 0.25 0.50 0.25 0.00 0.00 0.00
## [3,] 0.00 0.25 0.50 0.25 0.00 0.00
## [4,] 0.00 0.00 0.25 0.50 0.25 0.00
## [5,] 0.00 0.00 0.00 0.25 0.50 0.25
## [6,] 0.00 0.00 0.00 0.00 0.50 0.50
```

Note that the rows sum to 1, i.e., the probability mass is distributed over all possible transitions from any one location:

```
rowSums(T)
```

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

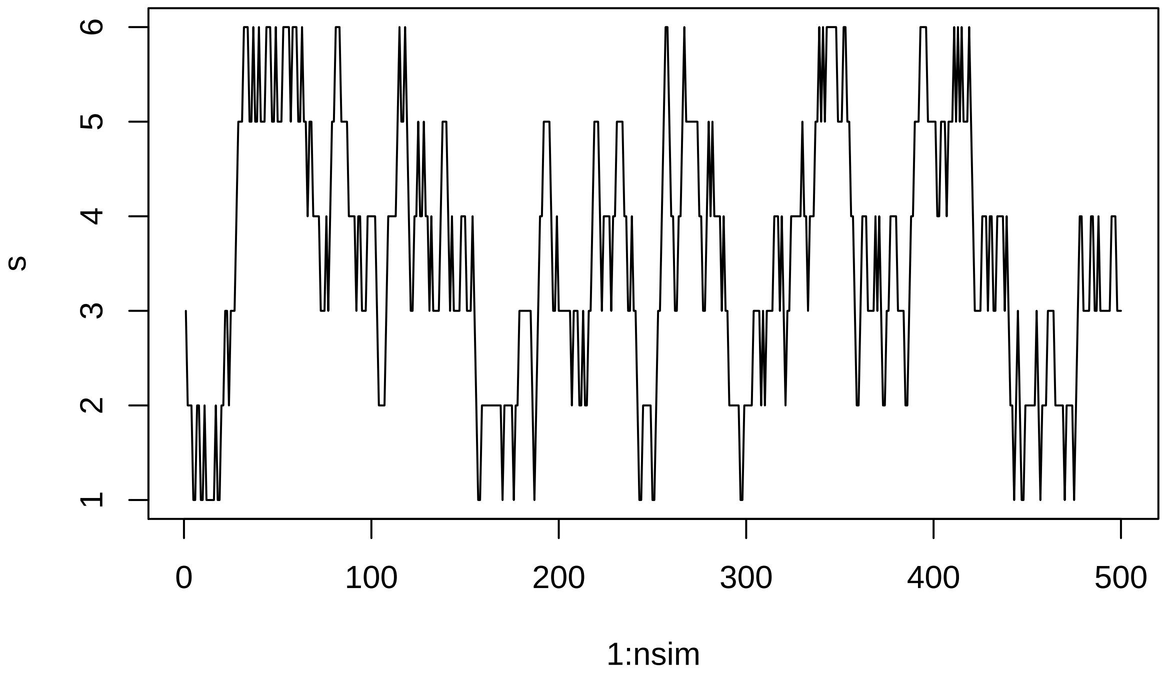
We can represent a current location as a probability vector: e.g., in state one, the transition probabilities for possible moves are:

```
T[1,]
```

```
## [1] 0.5 0.5 0.0 0.0 0.0 0.0
```

We can also simulate a random walk based on T:

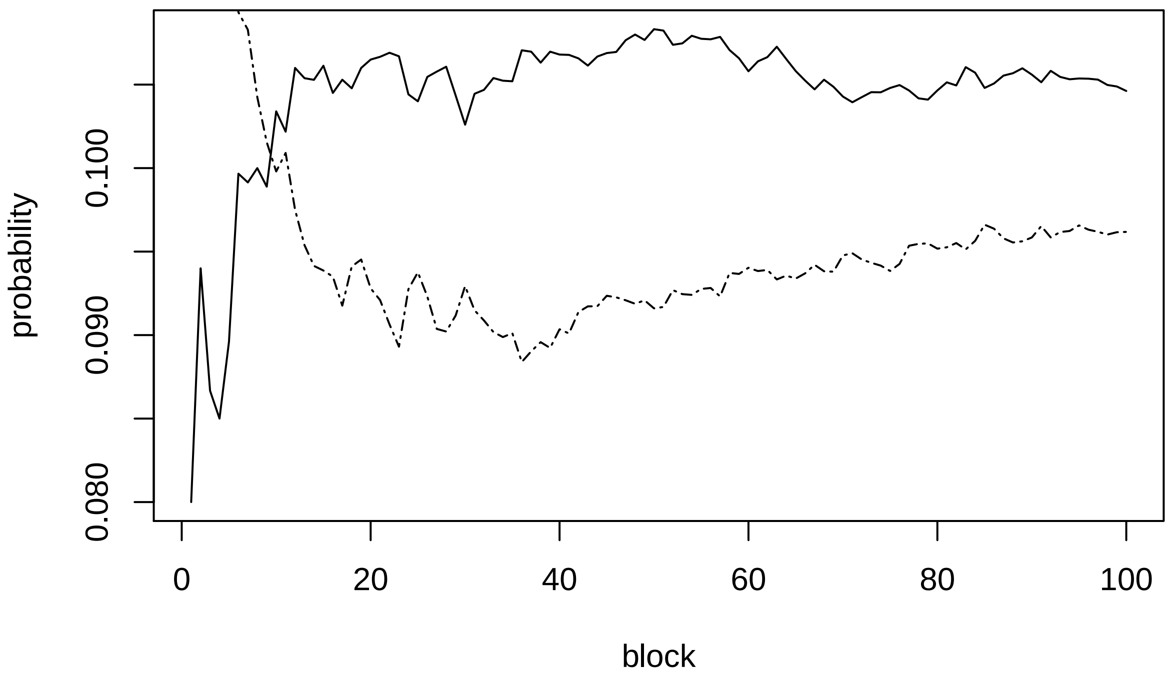
```
nsim<-500
s<-rep(0,nsim)
## initialize:
s[1]<-3
for(i in 2:nsim){
  s[i]<-sample(1:6,size=1,prob=T[s[i-1],])
}

plot(1:nsim,s,type="l")
```



Note that the above random walk is non-deterministic: if we rerun the above code multiple times, we will get different patterns of movement.

This Markov chain converges to a particular distribution of probabilities of visiting states 1 to 6. We can see the convergence happen by examining the proportions of visits to each state after blocks of steps that increase by 500 steps:

```r
nsim<-50000
s<-rep(0,nsim)
## initialize:
s[1]<-3
for(i in 2:nsim){
  s[i]<-sample(1:6,size=1,prob=T[s[i-1],])
}


blocks<-seq(500,50000,by=500)
n<-length(blocks)
## store transition probs over increasing blocks:
store.probs<-matrix(rep(rep(0,6),n),ncol=6)
## compute relative frequencies over increasing blocks:
for(i in 1:n){
  store.probs[i,]<-table(s[1:blocks[i]])/blocks[i]
}

## convergence for state 1:
for(j in 1:6){
if(j==1){
plot(1:n,store.probs[,j],type="l",lty=j,xlab="block",
     ylab="probability")
} else {
  lines(1:n,store.probs[,j],type="l",lty=j)
}
}
```



Each of the rows of the store.probs matrix is a probability mass function, which defines the probability distribution for the 6 states:

```r
store.probs[1,]
```

```
## [1] 0.080 0.196 0.184 0.208 0.192 0.140
```

This distribution is settling down to a particular set of values; that's what we mean by convergence. This particular set of values is:

```r
(w<-store.probs[n,])
```

```
## [1] 0.1046 0.2065 0.1990 0.1984 0.1953 0.0962
```

w is called a **stationary** distribution. If $wT = w$, then then w is the stationary distribution of the Markov chain. The R command %*% stands for matrix multiplication.

```r
round(w%*%T,digits=2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.1 0.21  0.2  0.2  0.2  0.1
```

```r
round(w,digits=2)
```

```
## [1] 0.10 0.21 0.20 0.20 0.20 0.10
```

This discrete example gives us an intuition for what will happen in continuous distributions: we will devise a Markov chain such that the chain will converge to the distribution we are interested in sampling from.

2.5.4 Monte Carlo sampling

Suppose we have a posterior that has the form of a Beta distribution. We can sample from the posterior distribution easily:

```
x<-rbeta(5000,1498,1519)
```

Once we have these samples, we can compute any kind of useful summary, e.g., the posterior probability p (given the data) that $p > 0.5$:

```
table(x>0.5)[2]/ sum(table(x>0.5))
```

```
##  TRUE
## 0.361
```

Or we can compute an interval over which we are 95% sure that the true parameter value lies:

```
##lower bound:
quantile(x,0.025)
```

```
##  2.5%
## 0.478
```

```
## upper bound:
quantile(x,0.975)
```

```
## 97.5%
## 0.515
```

Since we can integrate the Beta distribution analytically, we could have done the same thing with the `qbeta` function (or simply using calculus):

```
(lower<-qbeta(0.025,shape1=1498,shape2=1519))
```

```
## [1] 0.479
```

```
(upper<-qbeta(0.975,shape1=1498,shape2=1519))
```

```
## [1] 0.514
```

Using calculus (well, we are still using R; I just mean that one could do this by hand, by solving the integral):

```
integrate(function(x) dbeta(x,shape1=1498,shape2=1519),
          lower=lower,upper=upper)
```

```
## 0.95 with absolute error < 0.0000000000099
```

However—and here we finally get to the crucial point—integration of posterior densities is often impossible (e.g., because they may have many dimensions). In those situations we use sampling methods called Markov chain Monte Carlo, or MCMC, methods.

First, let's look at a relatively simple method of sampling: the inversion method.

2.5.5 The inversion method for sampling

This method works when we know the closed form of the pdf we want to simulate from and can derive the inverse of that function.

Steps:

1. Sample one number u from $Unif(0,1)$. Let $u = F(z) = \int_L^z f(x)\,dx$ (here, L is the lower bound of the pdf f).
2. Then $z = F^{-1}(u)$ is a draw from $f(x)$.

2.5.5.1 Example 1: Samples from Standard Normal

Take a sample from the Uniform(0,1):

```
u<-runif(1,min=0,max=1)
```

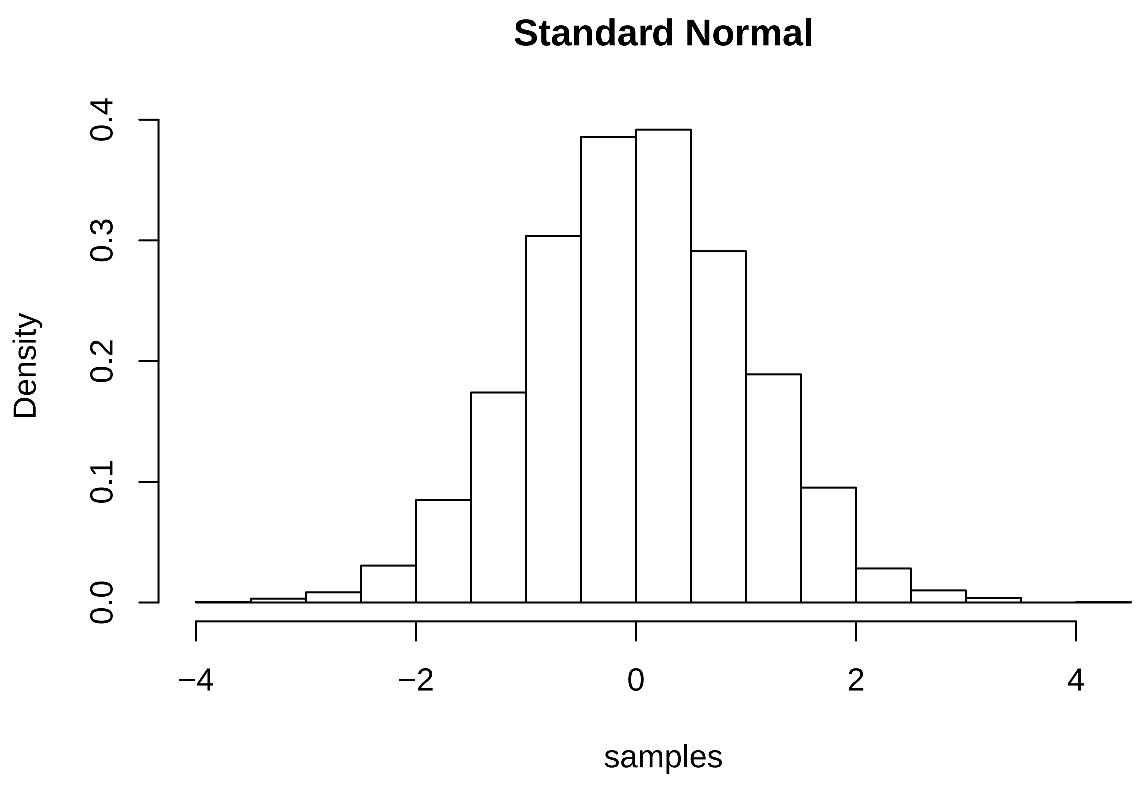
Let f(x) be a Normal density—we want to sample from this density. The inverse of the CDF in R is qnorm. It takes as input a probability and returns a quantile.

```
qnorm(u)
```

```
## [1] -0.887
```

If we do this repeatedly, we will get samples from the Normal distribution (here, the standard normal).

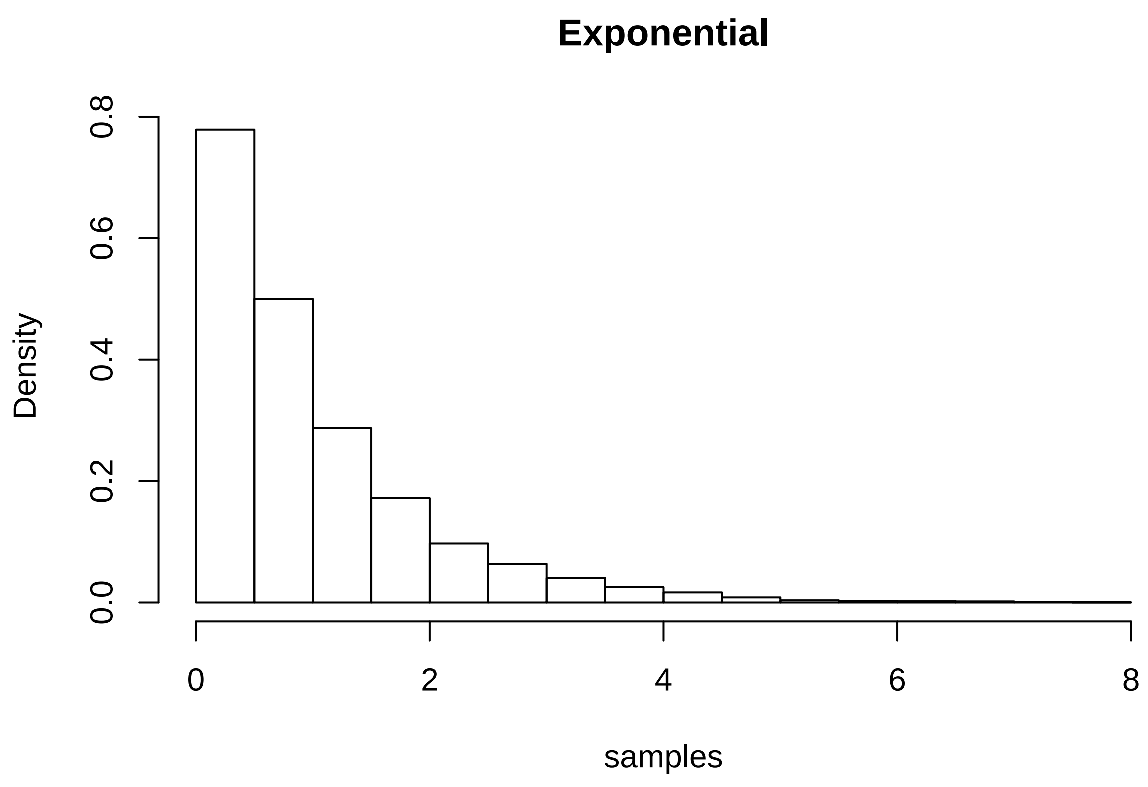
```
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qnorm(u)
}
hist(samples,freq=FALSE,main="Standard Normal")
```



2.5.5.2 Example 2: Samples from Exponential or Gamma

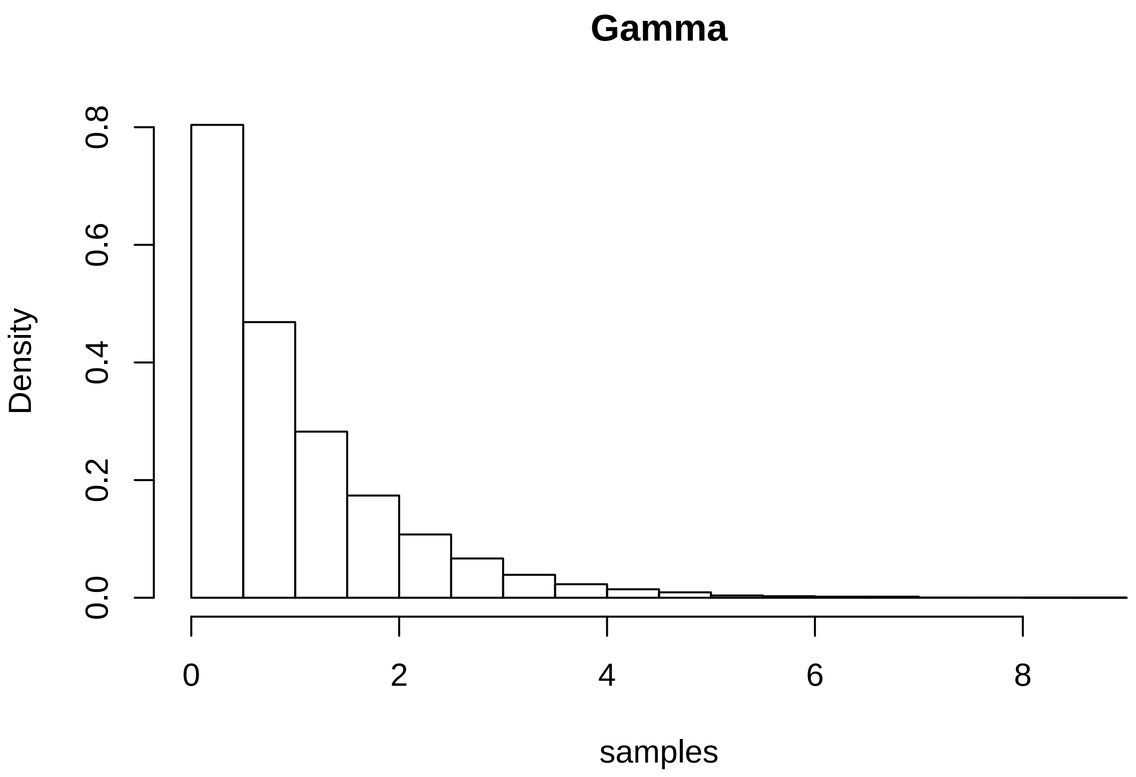
Now try this with the exponential with rate 1:

```r
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qexp(u)
}
hist(samples,freq=FALSE,main="Exponential")
```



Or the Gamma with rate and shape 1:

```r
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qgamma(u,rate=1,shape=1)
}
hist(samples,freq=FALSE,main="Gamma")
```



2.5.5.3 Example 3: Samples from an arbitrarily defined distribution

Let $f(x) = \frac{1}{40}(2x+3)$, with $0 < x < 5$. Now, we can't just use the family of q functions in R, because this density is not defined in R.

We have to draw a number from the uniform distribution and then solve for z, which amounts to finding the inverse function:

$$u = \int_0^z \frac{1}{40}(2x+3) \tag{150}$$

```
u<-runif(1000,min=0,max=1)

z<-(1/2) * (-3 + sqrt(160*u +9))
```

This method can't be used if we can't find the inverse, and it can't be used with multivariate distributions.

If the inverse can't be computed, we can use the rejection method.

2.5.6 The rejection method

If the inverse $F^{-1}(u)$ can't be computed, we sample from $f(x)$ as follows:

1. Sample a value z from a distribution $g(z)$ from which sampling is easy, and for which

$$mg(z) > f(z) \quad m \text{ a constant} \tag{151}$$

   $mg(z)$ is called an envelope function because it envelops $f(z)$.

2. Compute the ratio

$$R = \frac{f(z)}{mg(z)} \tag{152}$$

3. Sample $u \sim Unif(0,1)$.

4. If $R > u$, then z is treated as a draw from $f(x)$. Otherwise return to step 1.

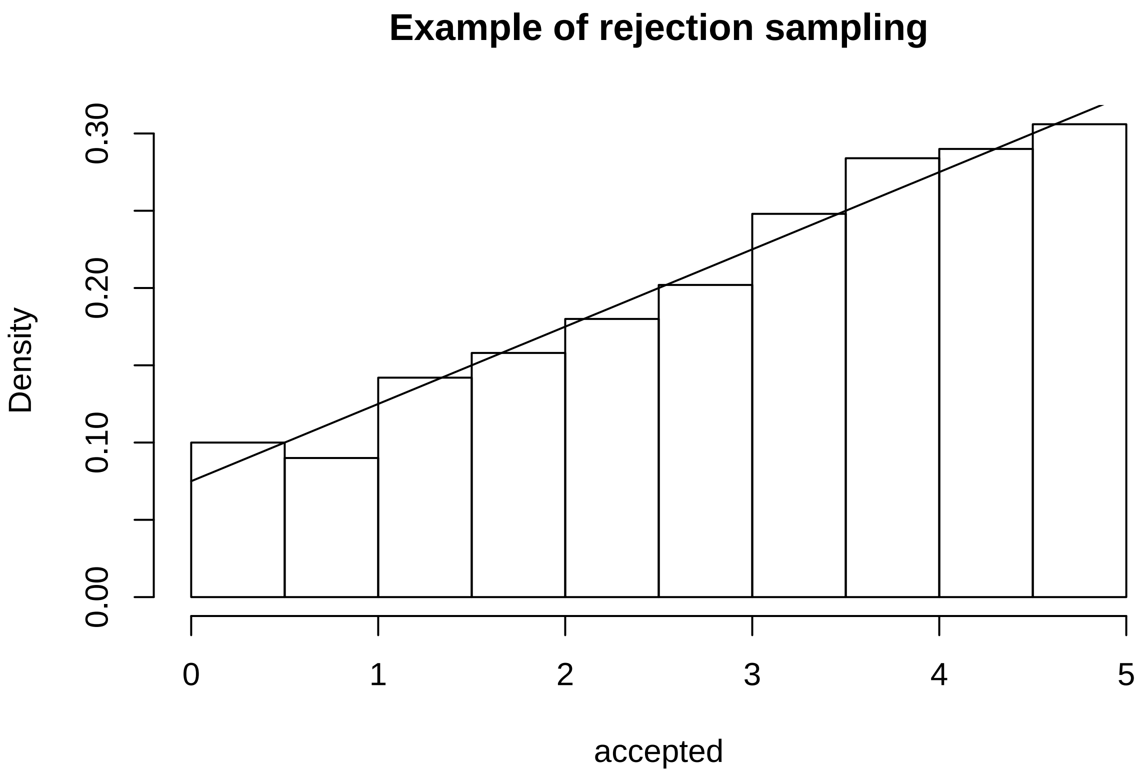
To take an example from Lynch (2007), consider f(x) to be: $f(x) = \frac{1}{40}(2x+3)$, with $0 < x < 5$. The maximum height of $f(x)$ is 0.325 (why? Try evaluating the function at x=5). So we need an envelope function that exceeds 0.325. The uniform density $Unif(0,5)$ has maximum height 0.2 (why is that?), so if we multiply it by 2 we have maximum height 0.4, which is greater than 0.325.

In the first step, we sample a number x from a uniform distribution Unif(0,5). This serves to locate a point on the x-axis between 0 and 5 (the domain of x). The next step involves locating a point in the y direction once the x coordinate is fixed. If we draw a number u from Unif(0,1), then $mg(x)u = 2*0.2u$ is a number between 0 and $2*0.2$. If this number is less than f(x), that means that the y value falls within f(x), so we accept it, else reject. Checking whether $mg(x)u$ is less than $f(x)$ is the same as checking whether

$$R = f(x)/mg(z) > u \tag{153}$$

```
#R program for rejection method of sampling
## From Lynch book, adapted by SV.
count<-0
k<-1
accepted<-rep(NA,1000)
rejected<-rep(NA,1000)
while(k<1001)
{
z<-runif(1,min=0,max=5)
r<-((1/40)*(2*z+3))/(2*.2)
if(r>runif(1,min=0,max=1)) {
  accepted[k]<-z
  k<-k+1} else {
    rejected[k]<-z
  }
count<-count+1
}
```

```r
hist(accepted,freq=F,
     main="Example of rejection sampling")

fn<-function(x){
  (1/40)*(2*x+3)
}

x<-seq(0,5,by=0.01)

lines(x,fn(x))
```



Here is the acceptance rate:

```r
## acceptance rate:
table(is.na(rejected))[2]/
  sum(table(is.na(rejected)))
```

```
##  TRUE
## 0.511
```

Question: If you increase m, will acceptance rate increase or decrease? Stop here and come up with an answer before you read further.

Rejection sampling can be used with multivariate distributions.

Some limitations of rejection sampling: finding an envelope function may be difficult; the acceptance rate would be low if the constant m is set too high and/or if the envelope function is too high relative to f(x), making the algorithm inefficient.

2.5.7 Gibbs sampling

Gibbs sampling is a very commonly used method in Bayesian statistics. Here is how it works.

Let Θ be a vector of parameter values, let length of Θ be k. Let j index the j-th iteration.

Algorithm:

1. Assign some starting values to Θ:

   $\Theta^{j=0} \leftarrow S$
2. Update the iteration index by 1: $j \leftarrow j+1$
3. 1. Sample $\theta_1^j \mid \theta_2^{j-1} \dots \theta_k^{j-1}$.
    2. Sample $\theta_2^j \mid \theta_1^j \theta_3^{j-1} \dots \theta_k^{j-1}$.

⋮

   k. Sample $\theta_k^j \mid \theta_1^j \dots \theta_{k-1}^j$.

4. Return to step 1.

The idea is simple. Repeat the following many times: for a vector of parameters, $\theta_1, \dots, \theta_k$, (1) take one sample from the conditional distribution of the first parameter given the values of the other parameters that were fixed in the previous iteration, (2) take one sample from the conditional distribution of the second parameter given the value of the first parameter from the current iteration, and the values of the third parameter onwards that were fixed in the previous iteration, and so on till you sample a value for the k-th parameter.

2.5.7.1 Example: A simple bivariate distribution

Assume that our bivariate (joint) density is:

$$f(x,y) = \frac{1}{28}(2x+3y+2) \tag{154}$$

Using the methods discussed in the Foundations chapter, it is possible to analytically work out the conditional distributions from the joint distribution. Just take my word for it here, we don't need to do these derivations ourselves.

$$f(x \mid y) = \frac{f(x,y)}{f(y)} = \frac{(2x+3y+2)}{6y+8} \tag{155}$$

$$f(y \mid x) = \frac{f(x,y)}{f(x)} = \frac{(2x+3y+2)}{4y+10} \tag{156}$$

The Gibbs sampler algorithm is:

1. Set starting values for the two parameters $x = -5, y = -5$. Set j=0.
2. Sample x^{j+1} from $f(x \mid y)$ using inversion sampling. You need to work out the inverse of $f(x \mid y)$ and $f(y \mid x)$ first. To do this, for $f(x \mid u)$, we have find z_1:

$$u = \int_0^{z_1} \frac{(2x+3y+2)}{6y+8}\, dx \tag{157}$$

   And for $f(y \mid x)$, we have to find z_2:

$$u = \int_0^{z_2} \frac{(2x+3y+2)}{4y+10}\, dy \tag{158}$$

   It doesn't matter if you can't solve this integral; the solution is given in the code below.

```
#R program for Gibbs sampling using inversion method
## program by Scott Lynch, modified by SV:
x<-rep(NA,2000)
y<-rep(NA,2000)
x[1]<- -5
y[1]<- -5

for(i in 2:2000)
{ #sample from x | y
  u<-runif(1,min=0, max=1)
  x[i]<-sqrt(u*(6*y[i-1]+8)+(1.5*y[i-1]+1)*(1.5*y[i-1]+1))-
    (1.5*y[i-1]+1)
  #sample from y | x
u<-runif(1,min=0,max=1)
y[i]<-sqrt((2*u*(4*x[i]+10))/3 +((2*x[i]+2)/3)*((2*x[i]+2)/3))-
    ((2*x[i]+2)/3)
}
```

You can run this code to visualize the simulated posterior distribution. See Figure 15.

```
library(MASS)
bivar.kde<-kde2d(x,y)
persp(bivar.kde,phi=10,theta=90,shade=0,border=NA,
      main="Simulated bivariate density using Gibbs sampling")
```

A central insight here is that knowledge of the conditional distributions is enough to simulate from the joint distribution, provided such a joint distribution exists.

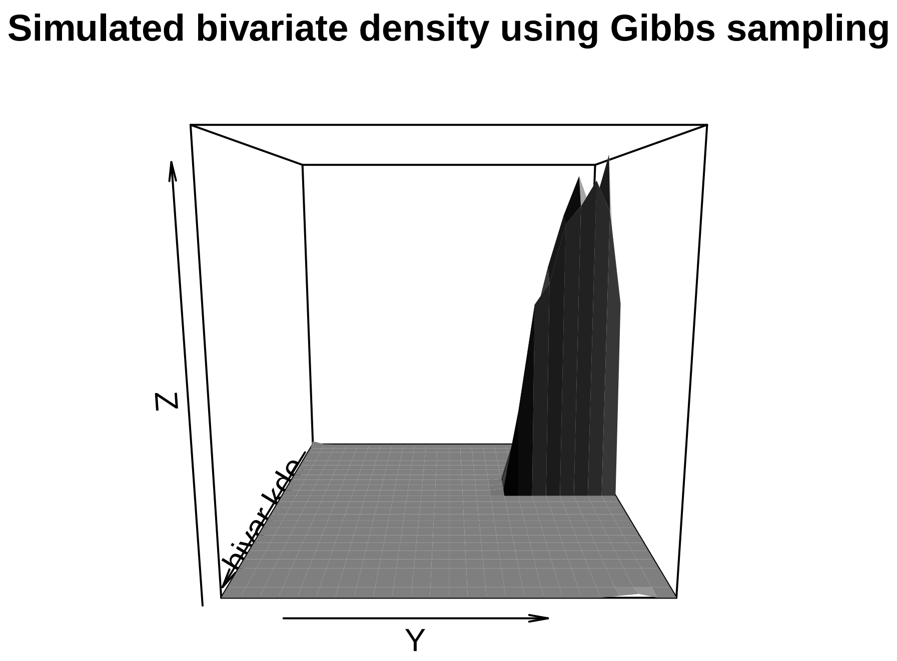


Figure 15: Example of posterior distribution of a bivariate distribution.

This section provided a short introduction to MCMC sampling, just to give a flavor for the way it works. For more details, read Lynch (2007) and Lambert (2018). We now turn to the sampling method used in Stan.

2.6 Hamiltonian Monte Carlo

Instead of Gibbs sampling, Stan uses this more efficient sampling approach. HMC works well for the high-dimensional models we will fit (hierarchical models). Gibbs sampling faces difficulties with some of the complex hierarchical models we will be fitting later. HMC will always succeed for these complex models.

One limitation of HMC (which Gibbs sampling does not have) is that HMC only works with continuous parameters (not discrete parameters).

For our purposes, it is enough to know what sampling using MCMC is, and that HMC gives us posterior samples efficiently. A good reference explaining HMC is Neal and others (2011). However, this paper is technically very demanding. More intuitively accessible introductions are available via Michael Betancourt's home page: https://betanalpha.github.io/. In particular, this video is helpful: https://youtu.be/jUSZboSq1zg. Another good resource is this visualization: https://chi-feng.github.io/mcmc-demo/app.html (this last one doesn't work well on my Firefox browser).

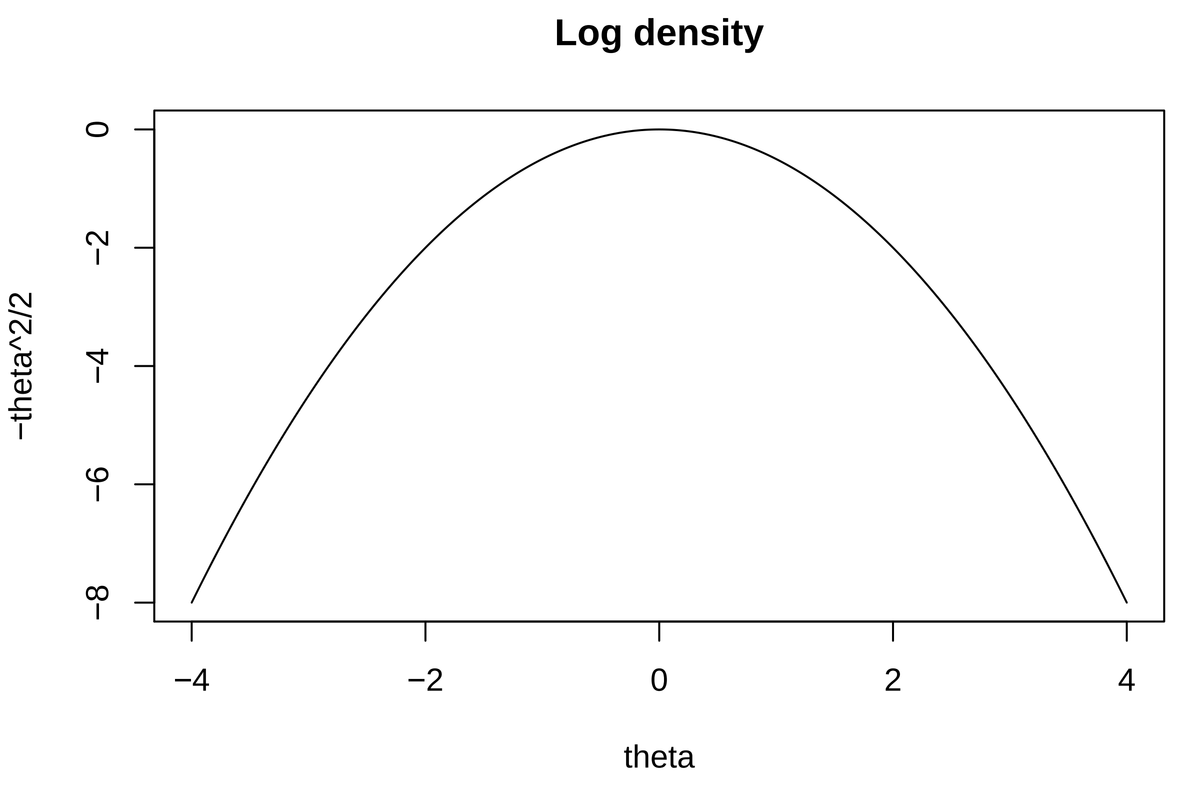


Figure 16: Example log density.

2.6.1 HMC demonstration

The HMC algorithm takes as input the log density and the gradient of the log density. In Stan, these will be computed internally; the user doesn't need to do any computations.

For example, suppose the log density is $f(\theta) = -\frac{\theta^2}{2}$. Its gradient is $f'(\theta) = -\theta$. Setting this gradient to 0, and solving for θ, we know that the maximum is at 0. We know it's a maximum because the second derivative, $f''(\theta) = -1$, is negative. See Figure 16.

This is the machinery we learnt in the foundations chapter (recall how we found MLEs in particular).

```
theta<-seq(-4,4,by=0.001)
plot(theta,-theta^2/2,type="l",main="Log density")
```

We first write down the Radford Neal algorithm for HMC; e and L are tuning parameters and can be ignored for now. The code is from Jarad Niemi's github repository.

```
## Radford Neal algorithm:
HMC_neal <- function(U, grad_U, e, L, current_theta) {
theta = current_theta
omega = rnorm(length(theta),0,1)
current_omega = omega
omega = omega - e * grad_U(theta) / 2
for (i in 1:L) {
```

```r
theta = theta + e * omega
if (i!=L) omega = omega - e * grad_U(theta)
}
omega = omega - e * grad_U(theta) / 2
omega = -omega
current_U = U(current_theta)
current_K = sum(current_omega^2)/2
proposed_U = U(theta)
proposed_K = sum(omega^2)/2
if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K))
{
return(theta)
}
else {
return(current_theta)
}
}

HMC <- function(n_reps, log_density, grad_log_density, tuning, initial) {
theta = rep(0, n_reps)
theta[1] = initial$theta
for (i in 2:n_reps) theta[i] = HMC_neal(U = function(x) -log_density(x),
grad_U = function(x) -grad_log_density(x),
e = tuning$e,
L = tuning$L,
theta[i-1])
theta
}
```

Then, we use the HMC function above to take 2000 samples from the posterior. We drop the first few (typically, the first half) samples, which are called warm-ups. The reason we drop them is that the initial samples may not yet be sampling from the posterior.

```r
theta <- HMC(n_reps=2000,
             log_density=function(x) -x^2/2,
             grad_log_density=function(x) -x,
             tuning=list(e=1,L=1),
             initial=list(theta=0))
```

Figure 17 shows a **trace plot**, the trajectory of the samples over 2000 iterations. This is called a **chain**. When the sampler is correctly sampling from the posterior, we see a characteristic "fat hairy caterpillar" shape, and we say that the sampler has **converged**. You will see later what a failed convergence looks like.

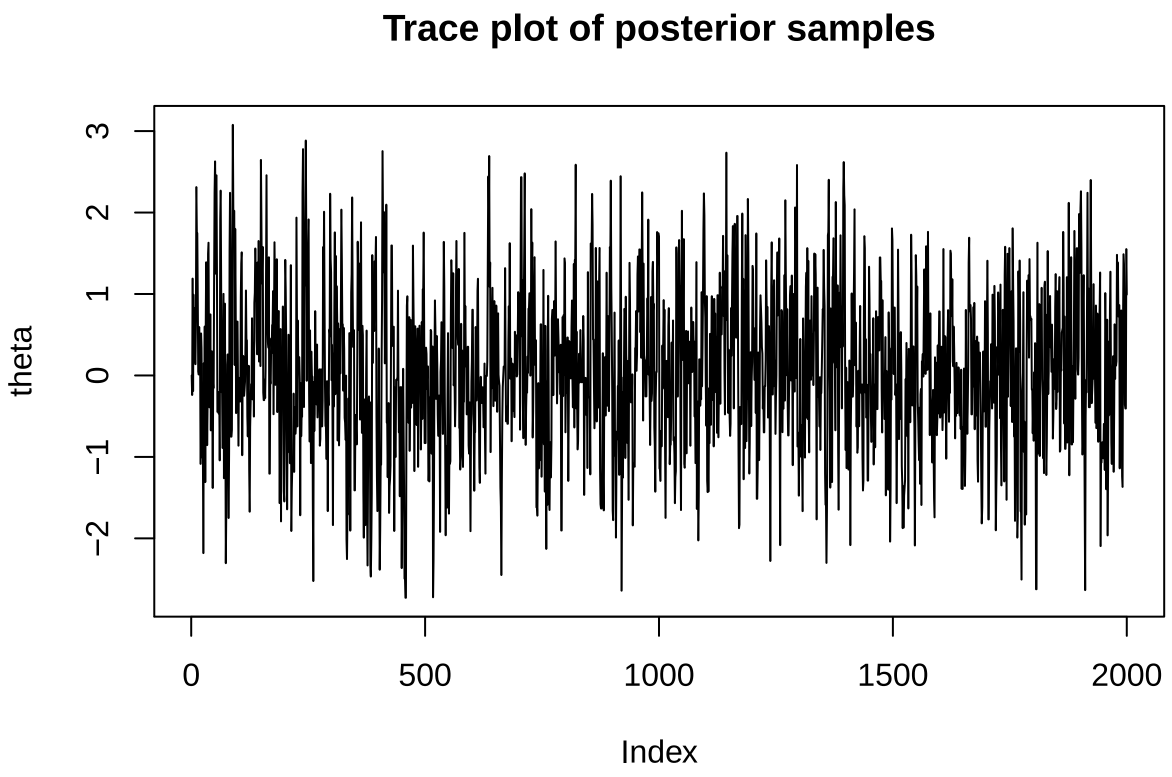


Figure 17: An example of a trace plot.

```r
plot(theta,type="l",main="Trace plot of posterior samples")
```

When we fit Bayesian models, we will always run four parallel chains. This is to make sure that even if we start with four different initial values chosen randomly, the chains all end up sampling from the same distribution. When this happens, we see that the chains overlap visually, and we say that the chains are **mixing**.

Figure 18 shows the posterior distribution of θ. We are not discarding samples here because the sampler converges quickly in this simple example.

```r
hist(theta, freq=F, 100,

     main="Posterior distribution of the parameter.",
     xlab=expression(theta))
curve(dnorm, add=TRUE, col='red', lwd=2)
```

In the modeling we do in the following pages, the Stan software will do the sampling for us.

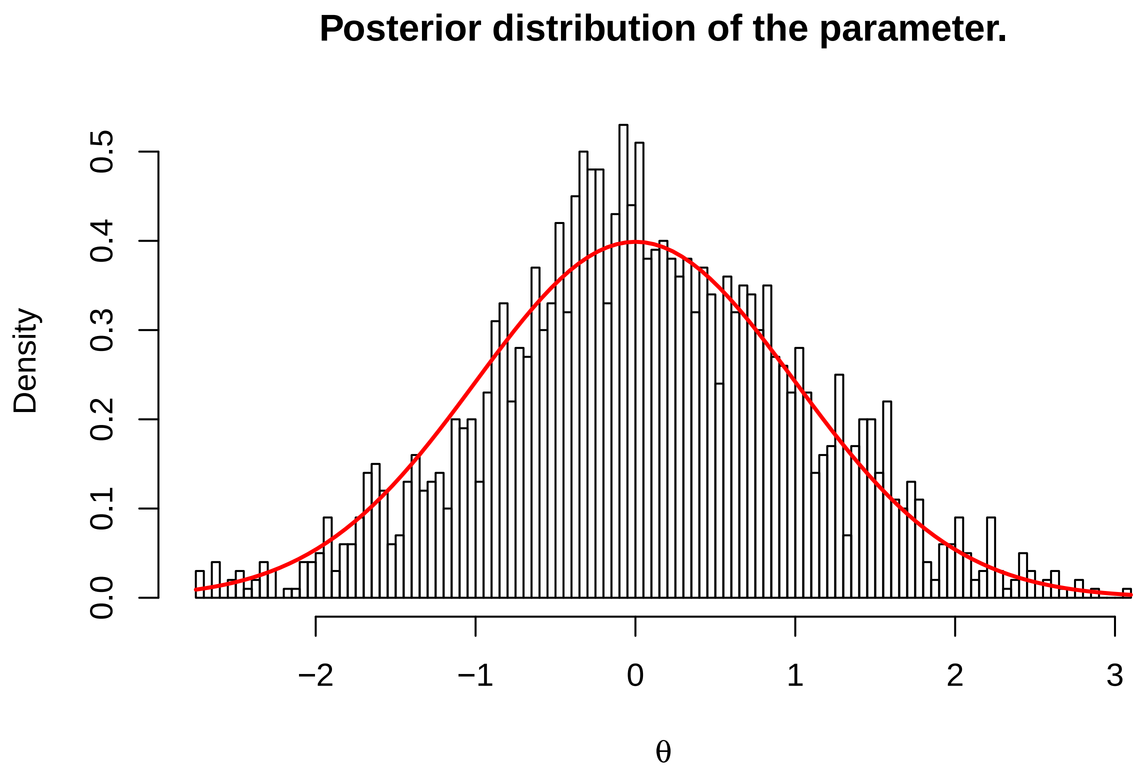


Figure 18: Sampling from the posterior using HMC. The red curve shows the true distribution, Normal(0,1).

2.7 Further readings

Some good books introducing Bayesian data analysis methods are the following:

- McElreath, R. (2015). Statistical Rethinking. Texts in Statistical Science. This book is currently the best textbook in existence, and assumes no calculus or linear algebra knowledge.
- Kruschke, J. (2014). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. Academic Press. This book is specifically designed for a psychology audience. I have only read parts of it and I find it very useful as a reference.
- Lynch, S. M. (2007). Introduction to applied Bayesian statistics and estimation for social scientists. Springer Science & Business Media. This book assumes knowledge of calculus and linear algebra. It gives a classical, statistician's introduction to Bayes. I highly recommend this book to people who know some (undergraduate math level) calculus and linear algebra.

3 Linear modeling

In this chapter, we will fit linear models of the following type. Suppose y is a vector of continuous responses; assume for now that it is coming from a normal distribution:

$y \sim Normal(\mu, \sigma)$

This is the simple linear model:

$y = \mu + \varepsilon$ where $\varepsilon \sim Normal(0, \sigma)$

There are two parameters, μ, σ, so we need priors on these. We expand on this simple model next.

Recall from the foundations chapter that the way we will conduct data analysis is as follows.

- Given data, specify a *likelihood function*.
- Specify *prior distributions* for model parameters.
- Evaluate whether model makes sense, using fake-data simulation, *prior predictive* and *posterior predictive* checks, and (if you want to claim a discovery) calibrating true and false discovery rates.
- Using software, derive *marginal posterior distributions* for parameters given likelihood function and prior density. I.e., simulate parameters to get *samples from posterior distributions* of parameters using some *Markov Chain Monte Carlo (MCMC) sampling algorithm*.
- Check that the model converged using *model convergence* diagnostics,
- Summarize *posterior distributions* of parameter samples and make your scientific decision.

We will now work through some specific examples to illustrate how the data analysis process works.

3.1 Example 1: A single subject pressing a button repeatedly

As a first example, we will fit a simple linear model to some reaction time data.

The file `button_press.dat` contains data of a subject pressing the space bar without reading in a self-paced reading experiment.

3.1.1 Preprocessing of the data

```
noreading_data <- read.table(header = F,"data/button_press.dat",
                             encoding="latin1")
noreading_data <- noreading_data[c("V2","V3","V5","V6","V8")]
colnames(noreading_data) <- c("type","item","wordn","word","y")
tail(noreading_data)
```

```
##       type item wordn        word   y
## 356 filler    3     0  Vielleicht 214
## 357 filler    3     1       haben 182
```

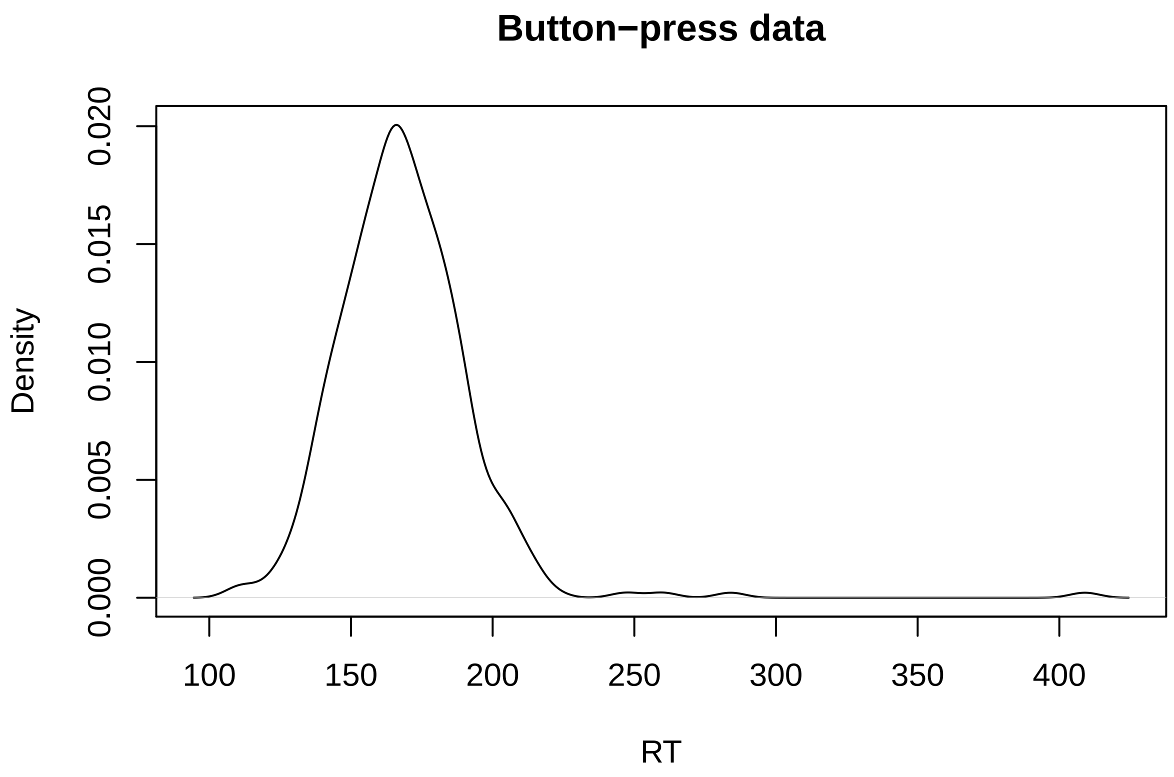


Figure 19: Visualizing the data.

```
## 358 filler    3     2 die_Zahnärztin 179
## 359 filler    3     3       aus_Bonn 177
## 360 filler    3     4  die_Patienten 183
## 361 filler    3     5      verklagt. 162
```

```
summary(noreading_data$y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     110     156     166     169     181     409
```

```
class(noreading_data)
```

```
## [1] "data.frame"
```

3.1.2 Visualizing the data

It is a good idea to look at the distribution of the data before doing anything else. See Figure 19.

```
plot(density(noreading_data$y),
     main="Button-press data",xlab="RT")
```

The data looks a bit skewed, but we ignore this for the moment.

3.1.3 Define the likelihood function

Let's model the data with the following assumptions:

- There is a true underlying time, μ, that the participant needs to press the space-bar.
- There is some noise in this process.
- The noise is normally distributed (this assumption is questionable given the skew but; we fix this assumption later).

This means that the likelihood for each observation i will be:

$$y_i \sim Normal(\mu, \sigma) \tag{159}$$

where $i = 1 \dots N$.

This is just the simple linear model:

$$y = \mu + \varepsilon \text{ where } \varepsilon \sim Normal(0, \sigma) \tag{160}$$

3.1.4 Define the priors for the parameters

We are going to use the following priors for the two parameters in this model:

$$\begin{aligned} &\mu \sim Normal(0, 2000) \\ &\sigma \sim Normal(0, 500) \text{ truncated so that } \sigma > 0 \end{aligned} \tag{161}$$

In order to decide on a prior for the parameters, always visualize them first. See Figure 20.

```
op<-par(mfrow=c(1,2),pty="s")
x<-seq(-4000,4000,by=1)
plot(x,dnorm(x,mean=0,sd=2000),type="l",
     main=expression(paste("Prior for ",mu)),xlab=expression(mu))
x<-seq(0,4000,by=1)
plot(x,dnorm(x,mean=0,sd=500),type="l",
     main=expression(paste("Prior for ",sigma)),xlab=expression(sigma))
```

The prior for μ expresses the belief that the underlying mean button-pressing time could be both positive and negative, and given that the scale of the prior (in this case the standard deviation of the normal distribution) is 2000, we are $\approx 68\%$ certain that the true value would be between 2000 ms and -2000 ms and $\approx 95\%$ certain that it would be between -4000 ms and 4000 ms (two standard deviations away from zero). We know this because:

```
pnorm(2000,mean=0,sd=2000)-pnorm(-2000,mean=0,sd=2000)
```

```
## [1] 0.683
```

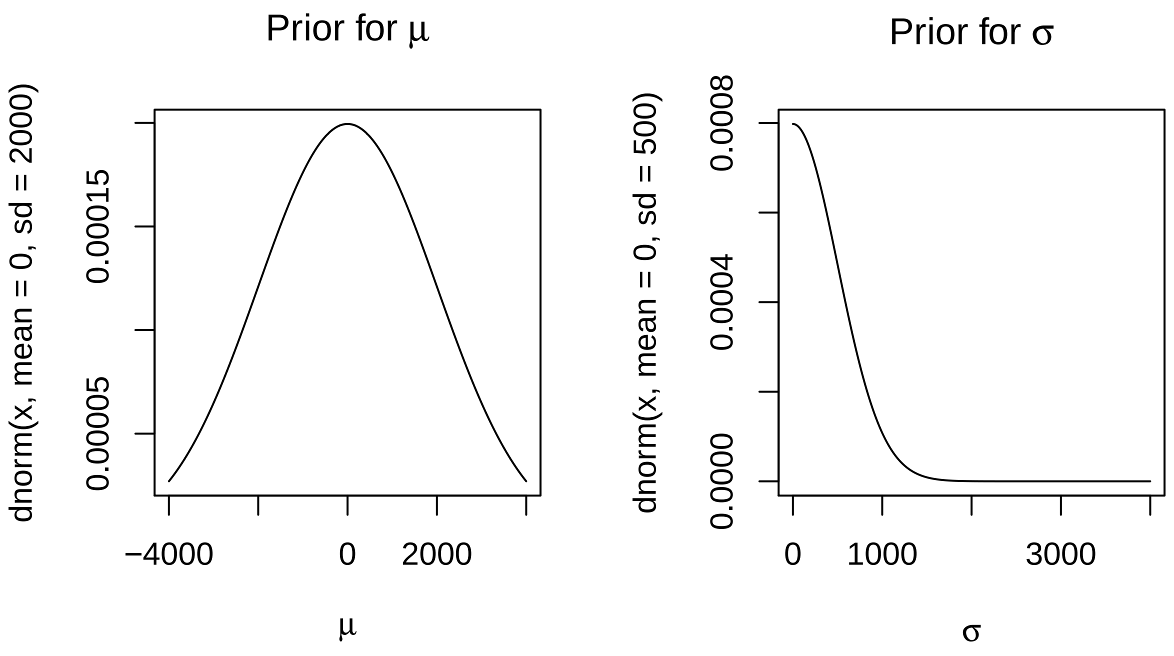


Figure 20: Visualizing the priors for example 1.

```
pnorm(4000,mean=0,sd=2000)-pnorm(-4000,mean=0,sd=2000)
```

```
## [1] 0.954
```

But we obviously know that button-pressing time can't be negative! So we have more prior information than what we are using for informing the model. We'll discuss this below.

Regarding the prior for σ: The values must be positive, and we are $\approx 68\%$ certain that the true value would be between 0 ms and 500 ms and $\approx 95\%$ certain that it would be between 0 ms and 1000 ms. **How would you check this using pnorm?**

3.1.5 Prior predictive checks

With these priors, we are going to generate something called the **prior predictive distribution**. This helps us check whether the priors make sense.

Formally, we want to know the density $f(\cdot)$ of data points $y_1,\dots,n$, given a vector of priors Θ. In our example, $\Theta = \langle \mu, \sigma \rangle$. The prior predictive density is:

$$f(y_1,\dots,y_n) = \int f(y_1)\cdot f(y_2)\cdots f(y_n) f(\Theta)\, d\Theta \tag{162}$$

In essence, we integrate out the parameters. Here is one way to do it in R:

- Take one sample from each of the priors
- Generate data using those samples

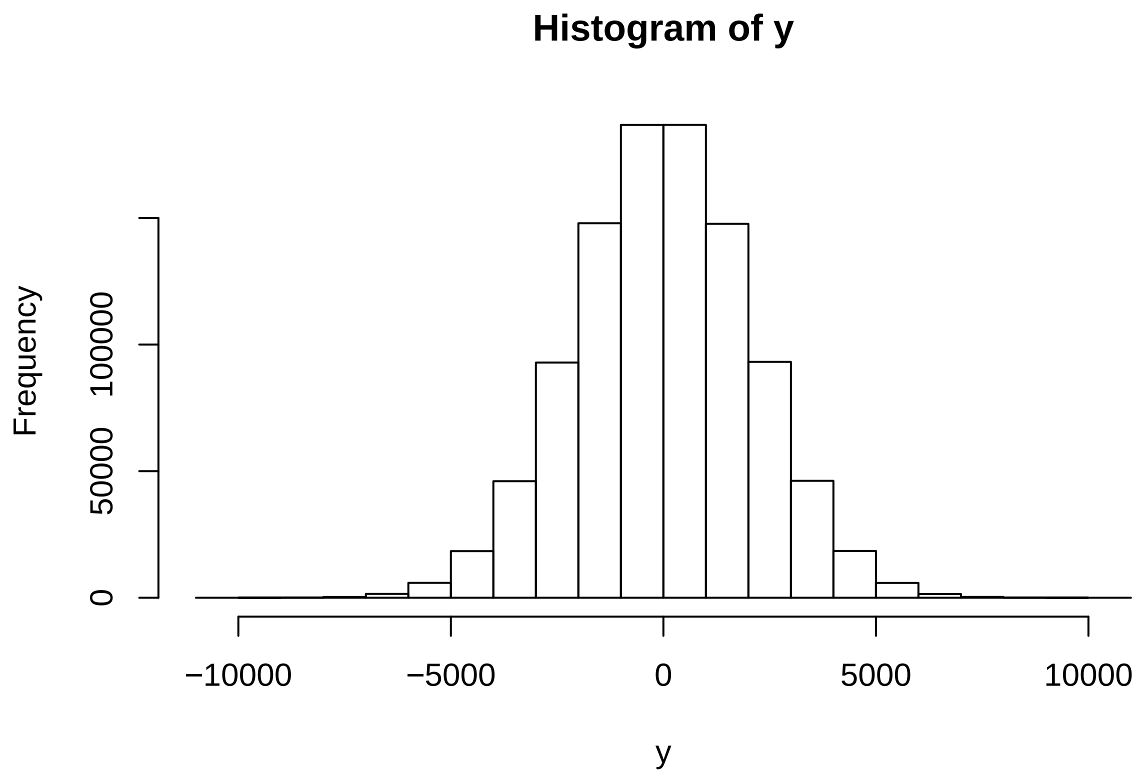


Figure 21: First attempt at prior predictive distribution of the data, model m1.

- Repeat until you have a substantial amount of data
- Plot the prior predictive density

```
nsim<-1000000
y<-rep(NA,nsim)
mu<-rnorm(nsim,mean=0,sd=2000)
sigma<-abs(rnorm(nsim,mean=0,sd=500))

for(i in 1:nsim){
y[i]<-rnorm(1,mean=mu[i],sd=sigma[i])
}

hist(y)
```

This prior predictive distribution in Figure 21 shows that the prior for μ is not realistic: how can button press time have negative values?

We can try to redefine the prior for μ to have only positive values, and then check again (Figure 22). We still get some negative values, but that is because we are assuming that

$y \sim Normal(\mu, \sigma)$

which will have negative values for small μ and large σ.

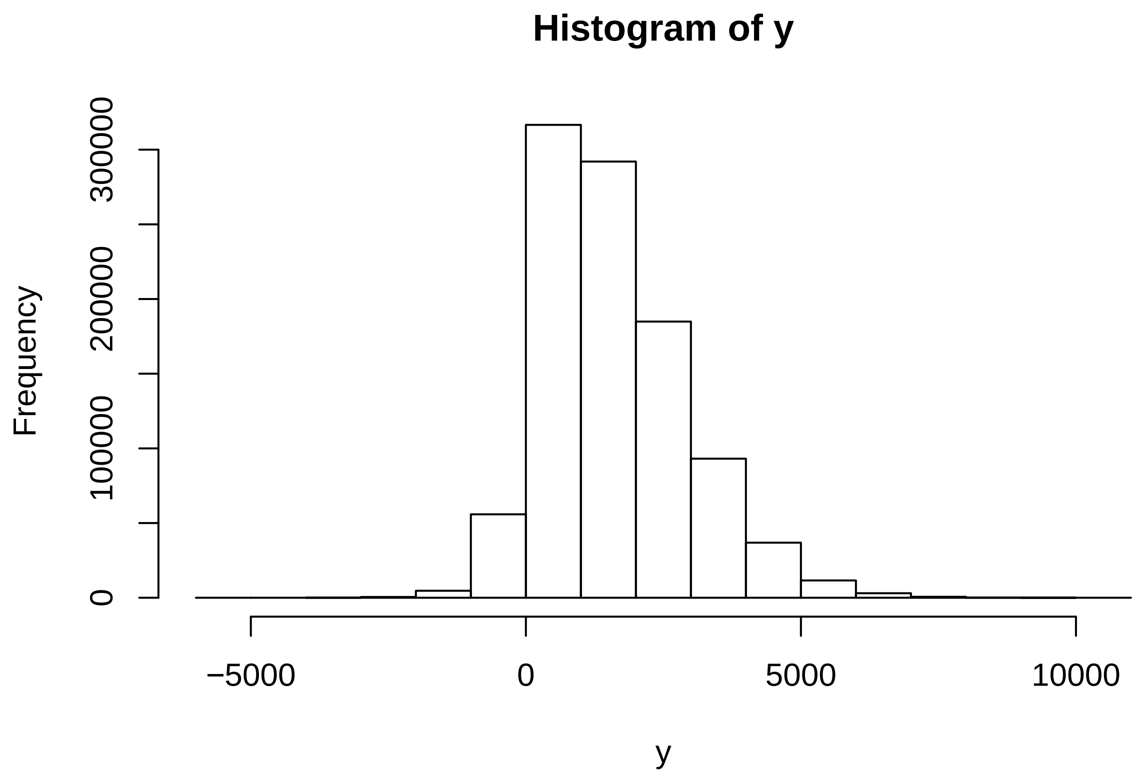


Figure 22: Second attempt at prior predictive distribution of the data, model m1.

```
nsim<-1000000
y<-rep(NA,nsim)
mu<-abs(rnorm(nsim,mean=0,sd=2000))
sigma<-abs(rnorm(nsim,mean=0,sd=500))

for(i in 1:nsim){
y[i]<-rnorm(1,mean=mu[i],sd=sigma[i])
}

hist(y)
```

This prior predictive distribution in Figure 22 looks reasonable for now.

3.2 Generating prior predictive data using Stan

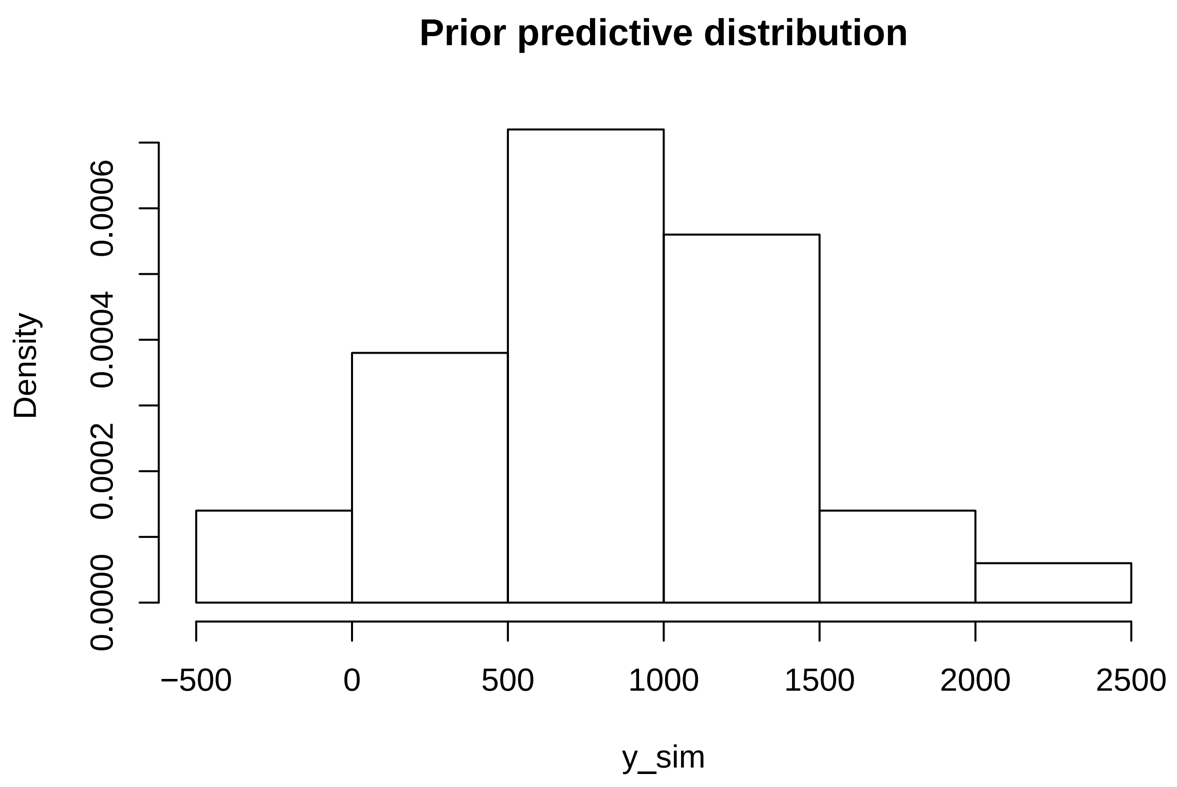
Incidentally, we can generate a prior predictive distribution using Stan as follows.

First, we define a Stan model that defines the priors and defines how the data are to be generated. The details of the code below will be explained in class. Documentation on Stan is available at mc-stan.org.

```
priorpred<-"data {
  int N;
}
parameters {
real<lower=0> mu;
real<lower=0> sigma;
}
model {
  // priors
    mu ~ normal(0,2000);
    sigma ~ normal(0,500);
}
generated quantities {
  vector[N] y_sim;
  // prior predictive
  for(i in 1:N) {
    y_sim[i] = normal_rng(mu,sigma);
  }
}
"
```

Then we generate the data:

```
## load rstan
library(rstan)
options(mc.cores = parallel::detectCores())

## generate 100 data points
dat<-list(N=100)

## fit model:
m1priorpred<-stan(model_code=priorpred,
                  data=dat,
                  chains = 4,
                iter = 2000)

## extract and plot one of the data-sets:
y_sim<-extract(m1priorpred,pars="y_sim")
dim(y_sim)
hist(y_sim$y_sim[1,],
     main="Prior predictive distribution",
     xlab="y_sim",freq=FALSE)
```



```
##compare with real data:
#y_fake<-rnorm(100,mean=1543,sd=1291)
#hist(y_fake,col="red",add=TRUE,freq=FALSE)
## compare with mean:
#mn<-rep(NA,100)
#for(i in 1:100){
#mn[i]<-mean(y_sim$y_sim[,i])
#}
#hist(mn)
#abline(v=1550)
```

Having satisfied outselves that the priors mostly make sense, we now fit the model to fake data. The goal here is to ensure that the model recovers the true underlying parameters.

3.2.1 Fake-data simulation and modeling

Next, we write the Stan model, adding a likelihood in the model block:

```
m1<-"data {
  int N;
  real y[N]; // data
}
parameters {
real<lower=0> mu;
real<lower=0> sigma;
}
model {
// priors
mu ~ normal(0,2000);
sigma ~ normal(0,500);
// likelihood
y ~ normal(mu,sigma);
}
generated quantities {
  vector[N] y_sim;
 // posterior predictive
  for(i in 1:N) {
    y_sim[i] = normal_rng(mu,sigma);
  }
}
"
```

Then generate fake data with known parameter values (we decide what these are):

```
set.seed(123)
N <- 500
true_mu <- 400
true_sigma <- 125
y <- rnorm(N, true_mu, true_sigma)

y <- round(y)
fake_data <- data.frame(y=y)
dat<-list(y=y,N=N)
```

Finally, we fit the model:

```
## fit model:
m1rstan<-stan(model_code=m1,
                  data=dat,
                  chains = 4,
                iter = 2000)

## extract posteriors:
posteriors<-extract(m1rstan,pars=c("mu","sigma"))
```

Figure 23 shows that the true parameters that we defined when generating the fake data fall within the posterior distributions. This shows that the model can in principle recover the parameters. (One should do several fake data simulations to check that the model consistently recovers the true parameters.)

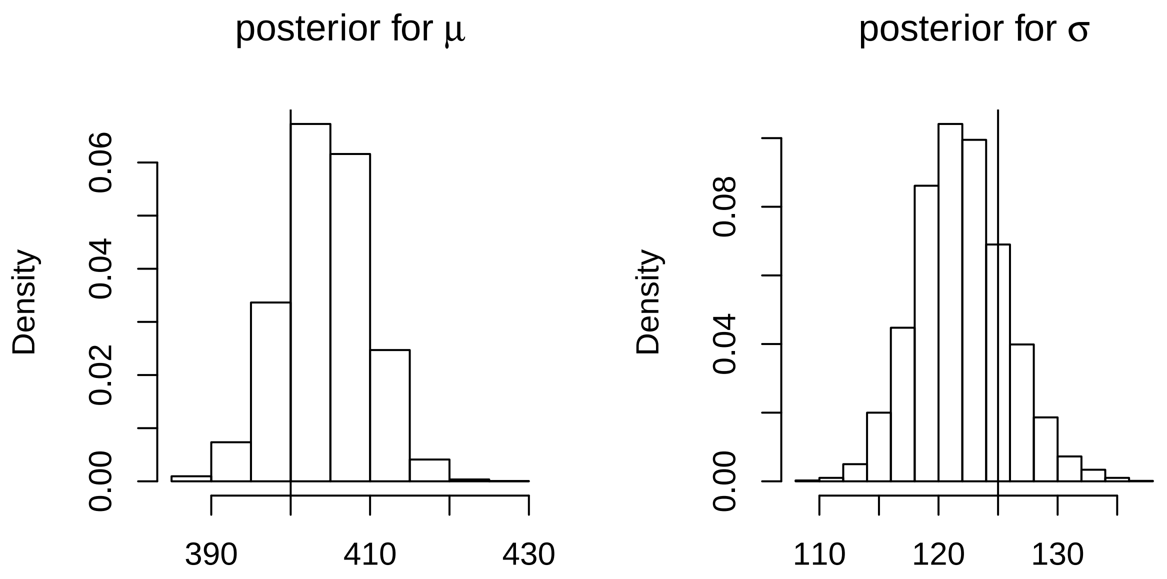


Figure 23: Posteriors from fake data, model m1. Vertical lines show the true values of the parameters.

```
op<-par(mfrow=c(1,2),pty="s")
hist(posteriors$mu,
     main=expression(paste("posterior for ",mu)),freq=FALSE,xlab="")
abline(v=400)
hist(posteriors$sigma,
     main=expression(paste("posterior for ",sigma)),freq=FALSE,xlab="")
abline(v=125)
```

3.2.2 Posterior predictive checks

Once we have the posterior distribution $f(\Theta \mid y)$, we can derive the predictions based on this posterior distribution:

$$p(y_{pred} \mid y) = \int p(y_{pred}, \Theta \mid y)\, d\Theta = \int p(y_{pred} \mid \Theta, y) p(\Theta \mid y)\, d\Theta \tag{163}$$

Assuming that past and future observations are conditionally independent given Θ, i.e., $p(y_{pred} \mid \Theta, y) = p(y_{pred} \mid \Theta)$, we can write:

$$p(y_{pred} \mid y) = \int p(y_{pred} \mid \Theta) p(\Theta \mid y)\, d\Theta \tag{164}$$

Note that we are conditioning y_{pred} only on y, we do not condition on what we don't know (Θ); **we integrate out the unknown parameters**.

This posterior predictive distribution is different from the frequentist approach, which gives only a predictive distribution of y_{pred} given our estimate of θ (a point value).

In the Stan code above, we have already generated the posterior predictive distribution, in the generated quantities block:

```
generated quantities {
  vector[N] y_sim;
 // posterior predictive
  for(i in 1:N) {
    y_sim[i] = normal_rng(mu,sigma);
  }
}
```

3.2.3 Implementing model in brms

An alternative to using rstan as we did above is the package brms. The advantage with using brms is that many of the details of model-specification are hidden from the user; the price paid is loss of transparency, and reduced flexibility in modeling. brms is a good software for fitting canned models, but for customized models you will always need Stan, so it is good to know both syntaxes. I personally do all my research using Stan almost exclusively.

First, load the brms package; this packge runs Stan (Stan Development Team 2017) in the background. For an introduction to this package, see Bürkner (2017), and https://github.com/paul-buerkner/brms.

```
library(brms)
```

This model is expressed in brms in the following way. First, define the priors:

```
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "sigma"))
```

Then, define the generative process assumed:

```
m1brms<-brm(y~1,noreading_data,prior = priors,
       iter = 2000,
       warmup = 1000,
       chains = 4,
       family = gaussian(),
       control = list(adapt_delta = 0.99))
```

1. The term `family = gaussian()` makes explicit the underlying likelihood function that is implicit in `lme4`. Other linking functions are possible, exactly as in the `glmer` function in `lme4`.

2. The term prior takes as argument the list of priors we defined earlier. Although this specification of priors is optional, the researcher should always explicitly specify each prior. Otherwise, brms will define a prior by default, which may or may not be appropriate.
3. The term iter refers to the number of iterations that the sampler makes to sample from the posterior distribution of each parameter (by default 2000). See the discussion on HMC earlier.
4. The term warmup refers to the number of iterations from the start of sampling that are eventually discarded (by default half of iter).
5. The term chains refers to the number of independent runs for sampling (by default four).
6. The term control refers to some optional control parameters for the sampler, such as adapt_delta. Briefly, increasing adapt_delta tells the sampler to take smaller steps when moving in the parameter space; it slows down sampling but that is a price one should be willing to pay if one is having convergence warnings.

3.2.4 Summarizing the posteriors, and convergence diagnostics

The summary displayed below show summary statistics over the marginal posterior distributions of the parameters in the model. The summary shows posterior means, standard deviations (sd), quantiles, Monte Carlo standard errors (se_mean), split Rhats, and effective sample sizes (n_eff).

The summaries are computed after removing the warmup and merging together all chains. Notice that the se_mean is unrelated to the se of an estimate in the frequentist model, and we will ignore it in this course.

```
summary(m1brms)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ 1
##    Data: noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept   168.68      1.34   166.06   171.34       2349 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma    25.00      0.94    23.24    26.93       2225 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

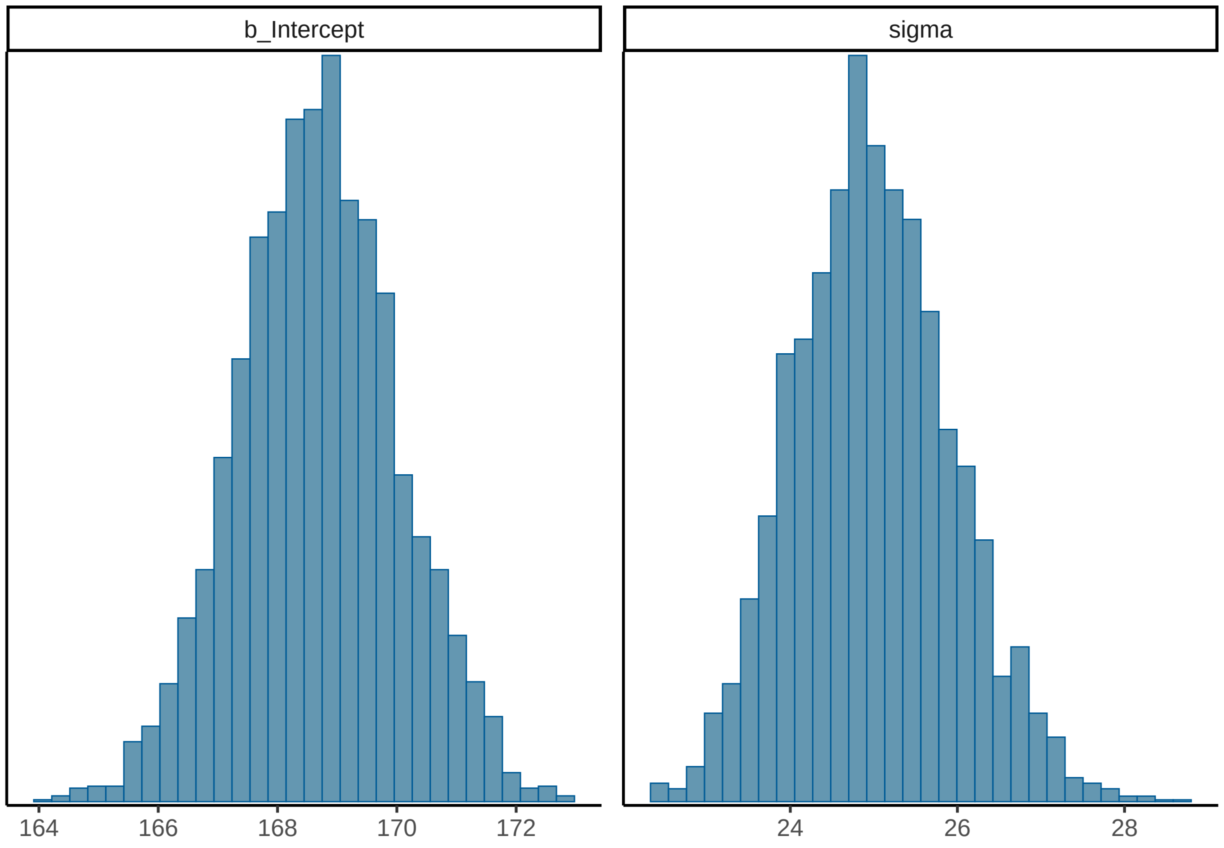


Figure 24: Posterior distributions of the parameters in model m1.

A graphical summary of posterior distributions of model m1 is shown in Figure 24:

```
stanplot(m1brms,type="hist")
```

The trace plots in Figure 25 show how well the four chains are mixing:

```
stanplot(m1brms,type="trace")
```

An alternative way to plot is shown in Figure 26.

```
plot(m1brms)
```

3.2.5 MCMC diagnostics: Convergence problems and Stan warnings

Because we are using MCMC methods to sample from the posterior distributions, we need to make sure that the model has converged.

The most important checks or MCMC diagnostics are the following:

- The chains should look like a straight “fat hairy caterpillar”: the chains should bounce around the same values and with the same variance.

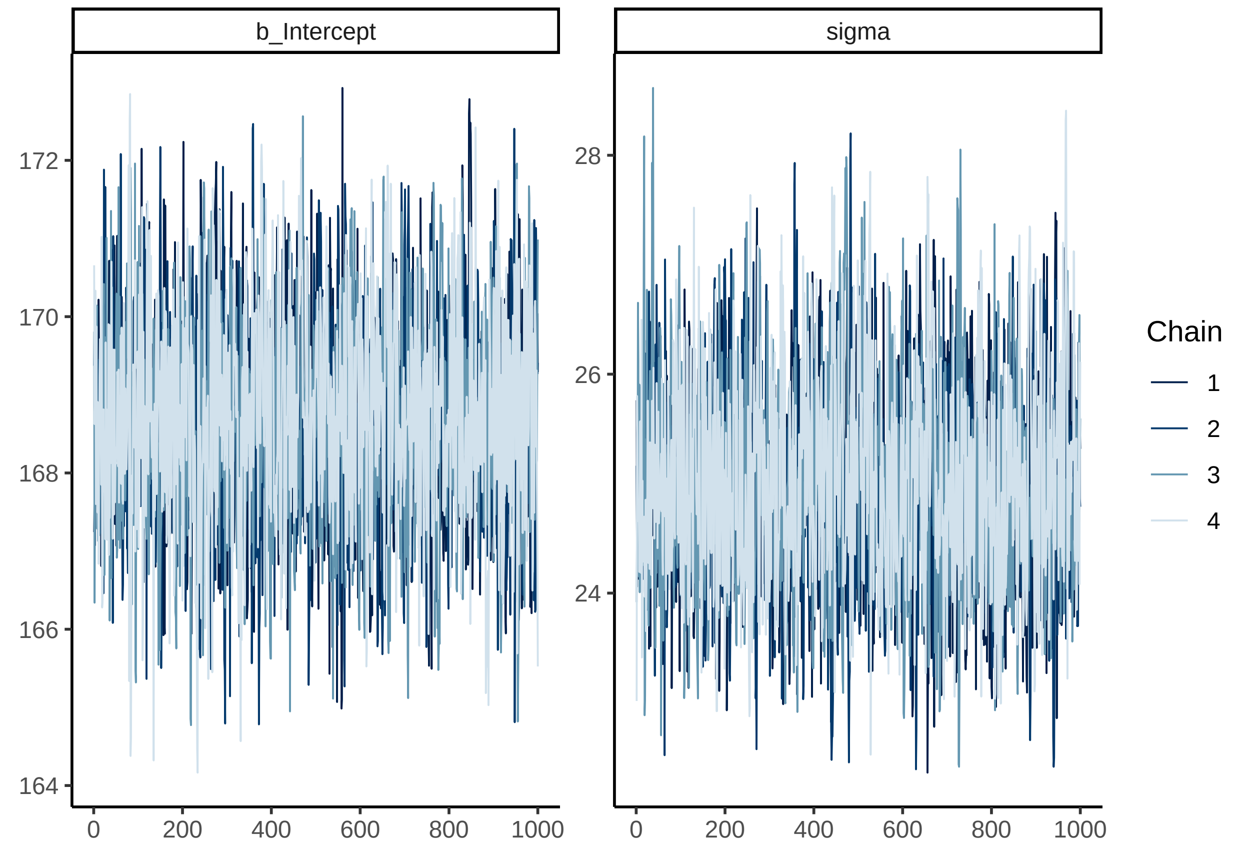


Figure 25: Trace plots in model m1.

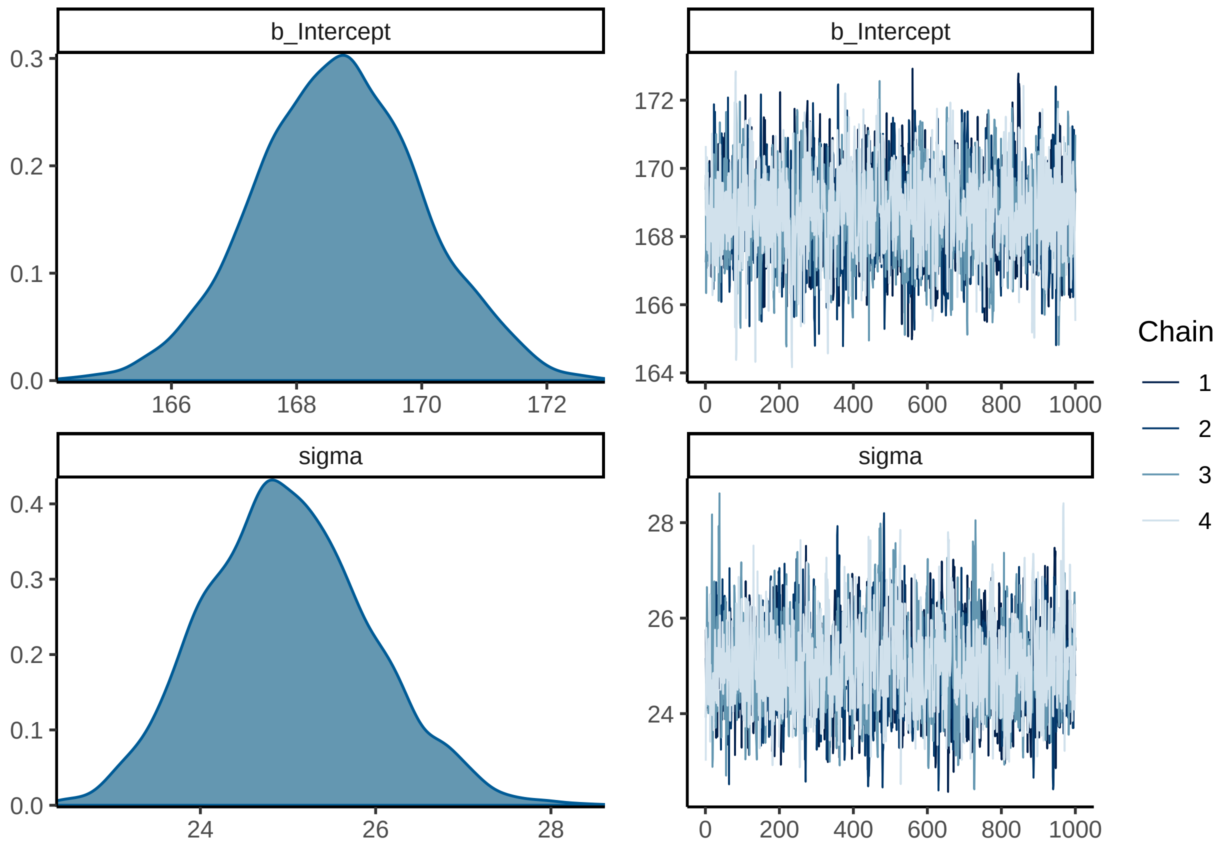


Figure 26: Posterior distributions and trace plots in model m1.

- The R-hat statistic, $\hat{R}$s, of the parameters should be close to one (as a rule of thumb less than 1.1). This indicates that the chains have mixed and they are traversing the same distribution. $\hat{R}$s are printed in the summary in the column `Rhat` (see section 11.4 of Gelman et al. 2014). These are the ratio of between to within chain variance.
- The effective sample size, n_{eff}, is an estimate of the number of independent draws from the posterior distribution. Since the samples are not independent, n_{eff} will generally be smaller than the total number of samples, N. How large n_{eff} should be depends on the summary statistics that we want to use. But as a rule of thumb, $n_{eff}/N > 0.1$.
- Check that the software does not warnings such as divergent transitions, Bayesian fraction of missing information (BFMI) that was too low, etc. These warning may indicate that the sampler is not adequately exploring the parameter space. If you see these warnings, consult http://mc-stan.org/misc/warnings.html

For useful graphical visualizations see https://mc-stan.org/users/interfaces/shinystan

Any warnings **should not be ignored**! See the Appendix 3.4 for some troubleshooting ideas to solve them.

3.2.6 Summarizing the posterior distribution: posterior probabilities and the credible interval

We are assuming that there's a true underlying time it takes to press the space bar, μ, and there is normally distributed noise with distribution Normal(0,σ) that generates the different RTs. All this is encoded in our likelihood by assuming that RTs are distributed with an unknown true mean μ (and an unknown standard deviation σ).

The objective of the Bayesian model is to learn about the plausible values of μ, or in other words, to get a distribution that encodes what we know about the true mean of the distribution of RTs, and about the true standard deviation, σ, of the distribution of RTs.

Our model allows us to answer questions such as:

What is the probability that the underlying value of the mindless press of the space bar would be over, say 170 ms?

As an example, consider this model that we ran above:

```r
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "sigma"))

m1brms<-brm(y~1,noreading_data,prior = priors,
       iter = 2000,
       warmup = 1000,
       chains = 4,
       family = gaussian(),
       control = list(adapt_delta = 0.99))
```

```
## Compiling the C++ model
## recompiling to avoid crashing R session
## Start sampling
```

We now compute the posterior probability $Prob(\mu > 170)$:

```
mu_post<-posterior_samples(m1brms,pars=c("b_Intercept"))$b_Intercept
mean(mu_post>170)
```

```
## [1] 0.155
```

The credible interval

The 95% credible interval can be extracted for μ as follows:

```
posterior_interval(m1brms,pars=c("b_Intercept"))
```

```
##              2.5% 97.5%
## b_Intercept  166   171
```

This type of interval is also known as a *credible interval*. A credible interval demarcates the range within which we can be certain with a certain probability that the "true value" of a parameter lies given the data and the model. This is very different from the frequentist confidence interval! See for discussion, Hoekstra et al. (2014) and Morey et al. (2016).

The percentile interval is a type of credible interval (the most common one), where we assign equal probability mass to each tail. We generally report 95% credible intervals. But we can extract any interval, a 73% interval, for example, leaves 13.5% of the probability mass on each tail, and we can calculate it like this:

```
quantile(mu_post,prob=c(0.135,0.865))
```

```
## 13.5% 86.5%
##   167   170
```

3.2.6.1 Influence of priors and sensitivity analysis

Everything was normally distributed in our example (or truncated normal), but the fact that we assumed that RTs were normally distributed is completely unrelated to our (truncated) normally distributed priors. Let's try a uniform prior on μ with a low boundary of 0 and a high boundary of 5000. Here, we assume that every value between 0 and 5000 is equally likely. In general, this is a bad idea for two reasons: (i) it is computationally expensive (the sampler has a larger parameter space to search), and (ii) it is providing information that we know is not sensible (every value between 0 and 5000 cannot be equally likely). But in our very simple example these priors will give use the same posterior as with the normal priors.

$$\begin{aligned}\mu &\sim Uniform(0,5000)\\ \sigma &\sim Uniform(0,5000)\end{aligned} \tag{165}$$

```r
priors <- c(set_prior("uniform(0, 5000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "sigma"))
```

```r
m2<-brm(y~1,noreading_data,prior = priors,
       iter = 2000, chains = 4,family = gaussian(),
       control = list(adapt_delta = 0.99))
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```r
summary(m2)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ 1
##    Data: noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept   168.59      1.34   165.96   171.23       2018 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma    25.01      0.95    23.26    26.99       2198 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

In general, we don't want our priors to have too much influence on our posterior. This is unless we have *very* good reasons for having informative priors, such as a very small sample and a lot of prior information; an example would be if we have data from an impaired population, which makes it hard to increase our sample size.

We usually center the priors on 0 and we let the likelihood dominate in determining the posterior. This type of prior is called *weakly informative prior*. Notice that a uniform prior is not a weakly informative prior, it assumes that every value is equally likely, zero is as likely as 5000.

You should always do a *sensitivity analysis* to check how influential the prior is: try different priors and verify that the posterior doesn't change drastically (for a published example, see Vasishth et al. 2013).

3.3 Example 2: Investigating adaptation effects

More realistically, we might have run the small experiment to find out whether the participant tended to speedup (practice effect) or slowdown (fatigue effect) while pressing the space bar.

3.3.1 Preprocessing the data

We need to have data about the number of times the space bar was pressed for each observation, and add it to our list. It's a good idea to center the number of presses (a covariate) to have a clearer interpretation of the intercept. In general, centering predictors is always a good idea, for interpretability and for computational reasons. See Schad et al. (2019) for details on this point.

```
# Create the new column in the data frame
noreading_data$presses <- 1:nrow(noreading_data)
# Center the column
noreading_data$c_presses <- noreading_data$presses - mean(noreading_data$presses)
```

3.3.2 Probability model

Our model changes, because we have a new parameter.

$$y_i \sim Normal(\alpha + presses_i \cdot \beta, \sigma) \tag{166}$$

where $i = 1 \dots N$

And we are going to use the following priors:

$$\begin{aligned} &\alpha \sim Normal(0, 2000) \\ &\beta \sim Normal(0, 500) \\ &\sigma \sim Normal(0, 500) \text{ truncated so that } \sigma > 0 \end{aligned} \tag{167}$$

We are basically fitting a linear model, α represents the intercept (namely, the grand mean of the RTs), and β represents the slope. What information are the priors encoding? Do the priors make sense?

We'll write this in brms as follows.

```
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "b",
                      coef="presses"),
            set_prior("normal(0, 500)",
                      class = "sigma"))
```

```
m2<-brm(y~1+presses,noreading_data,prior = priors,
       iter = 2000, chains = 4,family = gaussian(),
       control = list(adapt_delta = 0.99))
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```
summary(m2)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ 1 + presses
##    Data: noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept   152.54      2.41   147.66   157.28       3109 1.00
## presses       0.09      0.01     0.07     0.11       3342 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma    23.24      0.86    21.63    25.02       2710 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

3.3.3 Summarizing the posterior and inference

How can we answer our research question? What is the effect of pressing the bar on the participant's reaction time?

We'll need to examine what happens with β. The summary gives us the relevant information:

```
m2_post_samp_b <- posterior_samples(m2, "^b")
beta_samples <- m2_post_samp_b$b_presses
beta_mean<-mean(beta_samples)
quantiles_beta <- quantile(beta_samples,prob=c(0.025,0.975))
```

```
beta_low<-quantiles_beta[1]
beta_high<-quantiles_beta[2]
```

We learn that the most likely values of β will be around the mean of the posterior 0.089, and we can be 95% certain that the true value of β *given the model and the data* lies between 0.066 and 0.111.

We see that as the number of times the space bar is pressed increases, the participant becomes slower. If we want to determine how likely it is that the participant was slower rather than faster, we can examine the proportion of samples above zero:

```
mean(beta_samples > 0)
```

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

We would report this in a paper as $\hat{\beta} = 0.089$, 95% CrI = $[0.066, 0.111]$, $P(\beta > 0) \approx 1$. Plotting the posterior as a histogram is always a good idea.

Can we really conclude that there is a fatigue effect? It depends on how much we expect the fatigue to affect the RTs. Here we see that only after 100 button presses do we see a slowdown of 9 ms on average (0.09×100).

We need to consider whether the size of this effect has any scientific relevance by considering the previous literature. Sometimes this requires a meta-analysis. See Jäger, Engelmann, and Vasishth (2017), Nicenboim, Roettger, and Vasishth (2018) for examples. For examples of the use of this prior knowledge, see (**???**).

3.3.4 Posterior predictive checks

Let's say we know that our model is working as expected, since we already used fake data to test the recovery of the parameters (this will be a homework assignment).

We will now examine the *descriptive adequacy* of the models (Shiffrin et al. 2008; Gelman et al. 2014, Chapter 6): the observed data should look plausible under the *posterior predictive distribution*, as discussed above. The posterior predictive distribution is composed of one dataset for each sample from the posterior. (So it will generate as many datasets as iterations we have after the warm-up.) Achieving descriptive adequacy means that the current data could have been predicted by the model. Passing a test of descriptive adequacy is not strong evidence in favor of a model, but a major failure in descriptive adequacy can be interpreted as strong evidence against a model (Shiffrin et al. 2008).

To do posterior predictive checks for our last example, using brms, we need to do:

```
pp_check(m2, nsamples = 100)+
  theme(text = element_text(size=16),
        legend.text=element_text(size=16))
```

We'll use the values generated by our model to verify whether the general shape of the actual distribution matches the distributions from some of the generated datasets. Let's compare the real data against 100 of the predicted 4000 datasets.

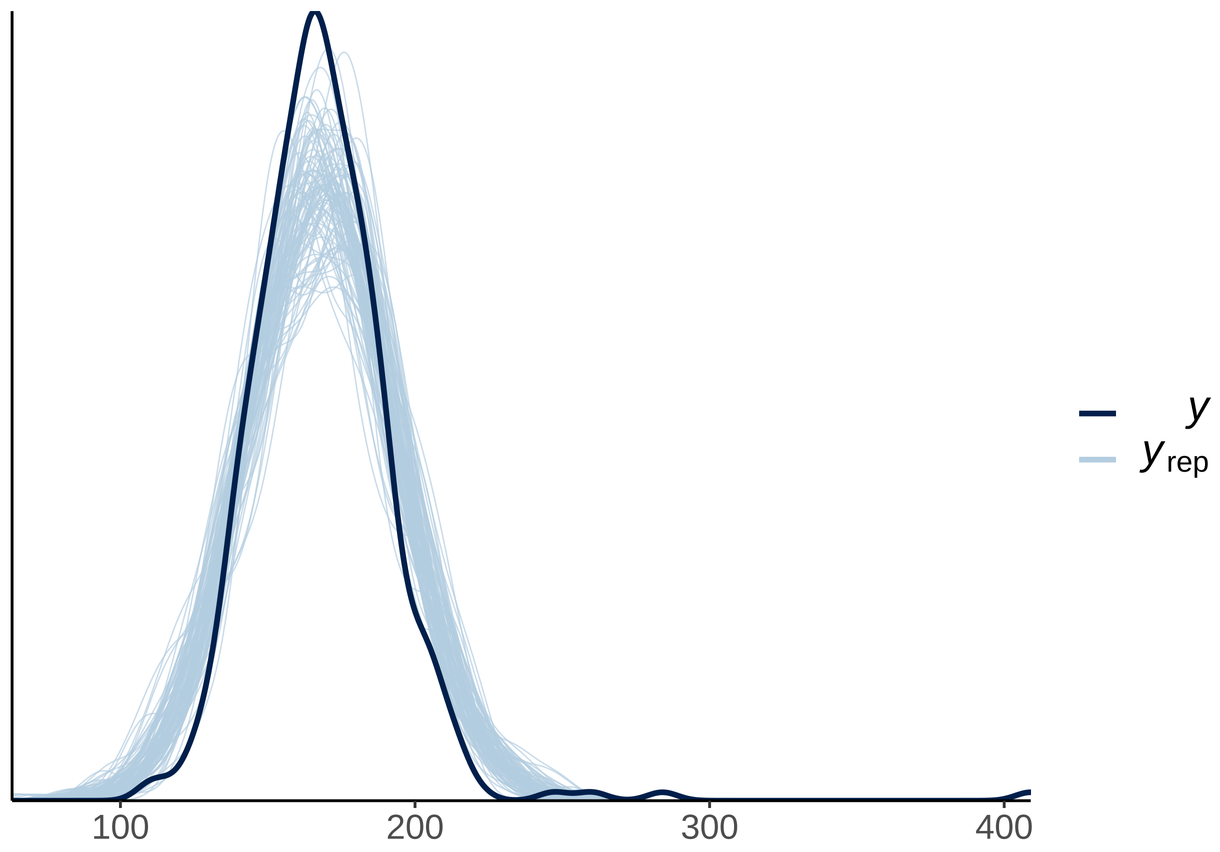


Figure 27: Posterior predictive check of model m2.

Are the posterior predicted data similar to the real data?

Figure 27 shows that our dataset seems to be more skewed to the right than our predicted datasets. This is not too surprising, we assumed that the likelihood was a normal distribution, but latencies are not very normal-like, they can't be negative and can be arbitrarily long.

3.3.5 Using the log-normal likelihood

Since we know that the latencies shouldn't be normally distributed, we can choose a more realistic distribution for the likelihood. A good candidate is the log-normal distribution since a variable (such as time) that is log-normally distributed takes only positive real values.

If Y is log-normally distributed, this means that $log(Y)$ is normally distributed.[2] Something important to notice is that the log-normal distribution is defined using again μ and σ, but this corresponds to the mean and standard deviation of the normally distributed logarithm $log(Y)$. Thus μ and σ are on a different scale than the variable that is log-normally distributed.

This also means that you can create a log-normal distribution by exponentiating the samples of a normal distribution. See Figure 28.

```r
mu <- 6
sigma <- 0.5
N <- 100000
# Generate N random samples from a log-normal distribution
sl <- rlnorm(N, mu, sigma)
lognormal_plot <- ggplot(data.frame(samples=sl), aes(sl)) + geom_histogram() +
      ggtitle("Log-normal distribution\n") +
  ylim(0,25000) + xlim(0,2000)
# Generate N random samples from a normal distribution,
# and then exponentiate them
sn <- exp(rnorm(N, mu, sigma))
normalplot <- ggplot(data.frame(samples=sn), aes(sn)) + geom_histogram() +
      ggtitle("Exponentiated samples of\na normal distribution") + ylim(0,25000) + xlim(

source("R/multiplot.R")
multiplot(lognormal_plot,normalplot,cols=)
```

3.3.6 Re-fit the model assuming a log-normal likelihood

If we assume that RTs are log-normally distributed, we'll need to change our model:

$$Y_i \sim LogNormal(\alpha + presses_i \cdot \beta, \sigma) \tag{168}$$

[2] In fact, $log_e(Y)$ or $ln(Y)$, but we'll write it as just $log()$

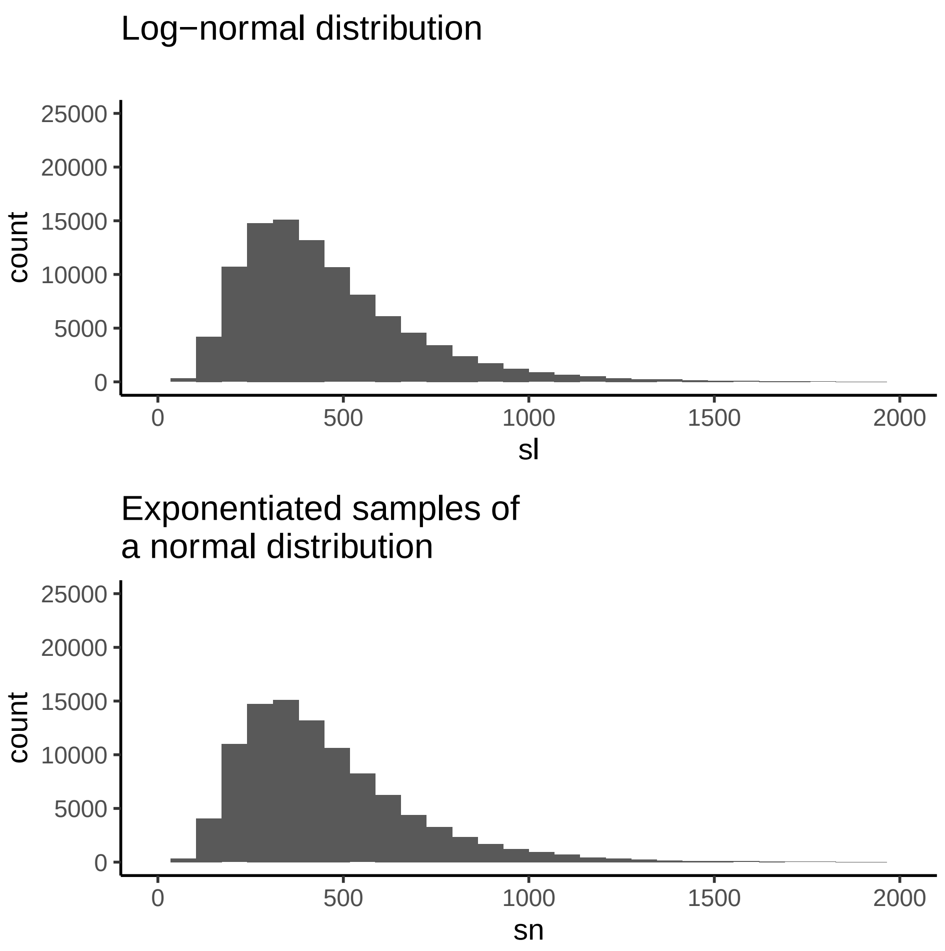


Figure 28: The log-normal distribution.

where $i = 1 \dots N$

But now the scale of our priors needs to change! They are no longer in milliseconds.

$$\begin{aligned}
\alpha &\sim Normal(0, 10) \\
\beta &\sim Normal(0, 1) \\
\sigma &\sim Normal(0, 2) \text{ truncated so that } \sigma > 0
\end{aligned} \tag{169}$$

The interpretation of the parameters changes and it is more complex than if we were dealing with a linear model that assumes a normal likelihood:

- α. In our previous linear model, α represented the grand mean (or the grand median since in a normal distribution both coincide), and was equivalent to our previous μ (since β was multiplied by 0). But now, the grand mean needs to be calculated in the following way, $\exp(\alpha + \sigma^2/2)$. Interestingly, the grand median will just be $exp(\alpha)$,[3] and we could assume that this represents the underlying time it takes to press the space bar if there would be no noise, that is, if σ had no effect. This also means that the prior of α is not in milliseconds, but in log(milliseconds).
- β. In a linear model, β represents the slowdown for each time the space bar is pressed. Now β is the effect on the log-scale, and the effect in milliseconds depends on the intercept α: $\exp(\alpha + \beta) - \exp(\alpha)$. Notice that the log is not linear and the effect of β will have more impact on milliseconds as the intercept grows. For example, if we start with (i) $\exp(5) = 148$, and we add 0.1 in log-scale, $\exp(5 + 0.1) = 164$, we end up with a difference of 15 ms; if we start with (ii) $\exp(6) = 400$, and we add 0.1, $\exp(6 + 0.1) = 445$, we end up with a difference of 45 ms. You can also see this graphically below, in Figure 29.
- σ. This is the standard deviation of the normal distribution of $log(y)$.

```r
ms_diff <- function(Intercept){
  exp(Intercept + .1) - exp(Intercept)
}
df <- tibble::data_frame(Intercept=seq(.1,15,.01),
                         ms= ms_diff(Intercept))
ggplot(df, aes(x=Intercept,y=ms)) + geom_point() +
  scale_y_continuous("Effect in milliseconds \n  (log-scale effect: 0.1)")
```

3.3.7 What kind of information are the priors encoding?

- For α: We are 95% certain that the grand median of the RTs will be between ≈ 0 and 485165195 milliseconds. This is a (very) weakly regularizing prior because it won't affect our results, but it will down-weight values for the grand median of the RTs that are extremely large, and won't allow the grand median to be negative. We calculate the previous range by

[3] You can check in Wikipedia (https://en.wikipedia.org/wiki/Log-normal_distribution) why.

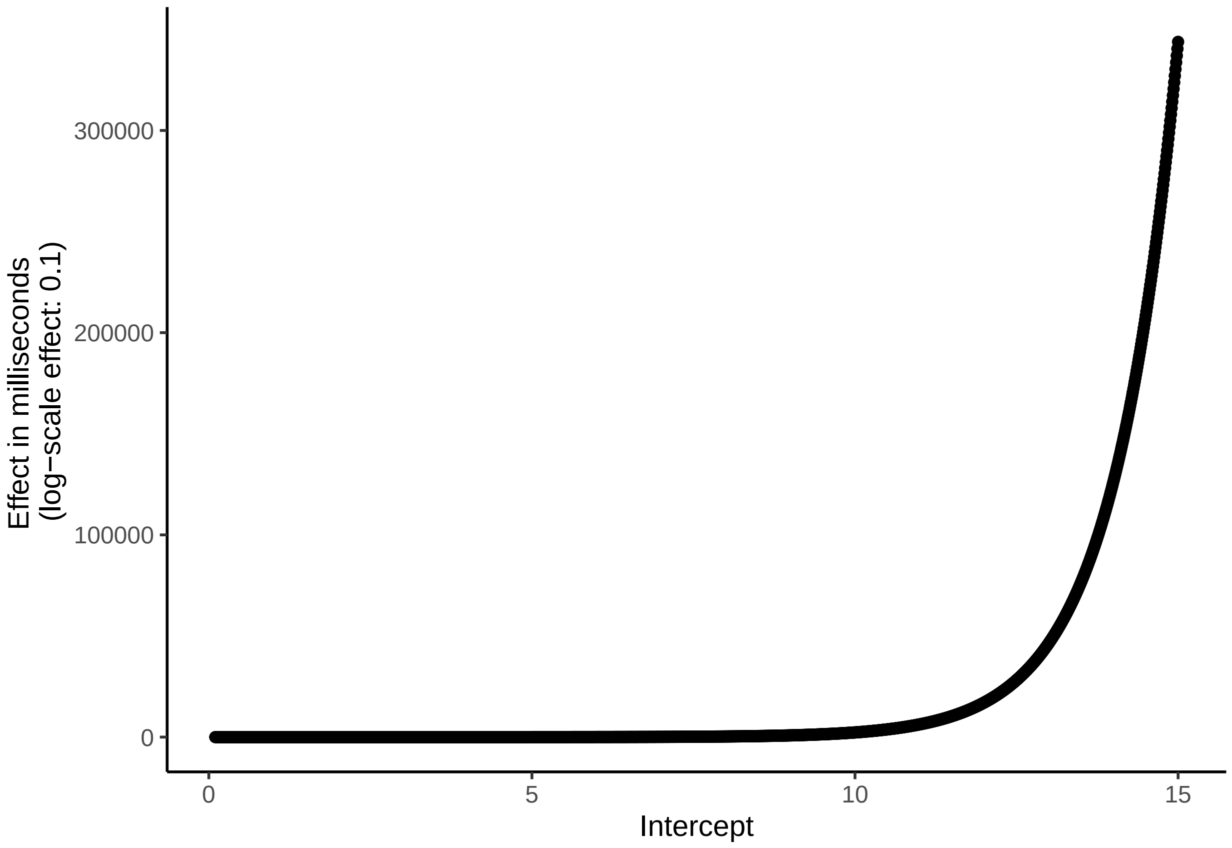


Figure 29: Back-transforming an effect of 0.1 log milliseconds to milliseconds.

back-transforming the values that lie between two standard deviations of the prior (2×10) to millisecond scale: $exp(-10 \times 2)$ and $exp(10 \times 2)$).

- For β: This is more complicated, because the effect on milliseconds will depend on the estimate of α. However, we can assume some value for α and it will be enough to be in the right order of magnitude. So let's assume 500 ms. That will mean that we are 95% certain that the effect of pressing the space bar will be between -491 and 26799 milliseconds. It is asymmetric because the log-scale is asymmetric. But the prior is weak enough so that if we assume 1000 or 100 instead of 500, the possible estimates of β will still be contained in the prior distribution. We calculate this by first finding out the value in milliseconds when we are two standard deviations away in both directions: (2×2), that is $\exp(\log(500) - 2 \times 2)$ and $\exp(\log(500) + 2 \times 2)$, and we subtract from that the value of α that we assumed, 500: $\exp(\log(500) - 2 \times 2) - 500$ and $\exp(\log(500) + 2 \times 2) - 500$.
- For σ. This indicates that we are 95% certain that the standard deviation of $log(y)$ will be between 0 and 2. So 95% of the RTs will be between $\exp(\log(500) - 1 \times 2) = 68$ and $exp(log(500) + 1 \times 2) = 3695$.

What happens if we replace 500 by 100, and by 1000? What happens if it is 10 instead? Does it still makes sense?

We'll code the model as follows.

```r
priors_log <- c(set_prior("normal(0, 10)",
                          class = "Intercept"),
            set_prior("normal(0, 1)",
                          class = "b",
                          coef="presses"),
            set_prior("normal(0, 2)",
                          class = "sigma"))
```

```r
m2_logn<-brm(y~1+presses,noreading_data,prior = priors_log,
       iter = 2000, chains = 4,family = lognormal(),
       control = list(adapt_delta = 0.99,max_treedepth=15))
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```r
summary(m2_logn)
```

```
##  Family: lognormal
##   Links: mu = identity; sigma = identity
## Formula: y ~ 1 + presses
##    Data: noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
```

```
##           Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept     5.02      0.01     5.00     5.05       2771 1.00
## presses       0.00      0.00     0.00     0.00       3768 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     0.12      0.00     0.11     0.13       1266 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

We fit the model, and check its convergence as usual (this will be a homework assignment).

3.3.8 Summarizing the posterior and inference

Next, we turn to the question of what we can report as our results, and what we can conclude from the data.

We can summarize the posterior and do inference as discussed in Example 1. If we want to talk about the effect estimated by the model, we summarize the posterior of β in the following way: $\hat{\beta} = 0.001$, 95% CrI = $[0, 0.001]$.

But in most cases, the effect is easier to interpret in milliseconds. We generated the effect of 1 press in the generated quantities block, which is not the same as the linear model's β. Our generated estimate will tell us the estimate of the slowdown produced by pressing the space bar in the middle of the experiment once, assuming that the RTs are log-normally distributed: 0.079 ms, 95% CrI = $[0.062, 0.096]$. Coincidentally, it is close to the same value as before, but this is not always the case, and since it's not linear the effect won't be the same across the whole experiment.

3.3.9 Posterior predictive checks and distribution of summary statistics

We can now verify whether our predicted datasets look more similar to the real dataset. See Figure 30.

```
pp_check(m2_logn, nsamples = 100)+
  theme(text = element_text(size=16),legend.text=element_text(size=16))
```

Are the posterior predicted data now more similar to the real data?

It seems so, but it's not easy to tell. Another way to examine this would be to look at the *distribution of summary statistics*. The idea is to compare the distribution of representative summary statistics for the datasets generated by different models and compare them to the observed statistics. Since we suspect that the log-normal distribution may capture the long tail, we could use the maximum as a summary statistics. We could generate 100 posterior predictive data-sets, and then compute the

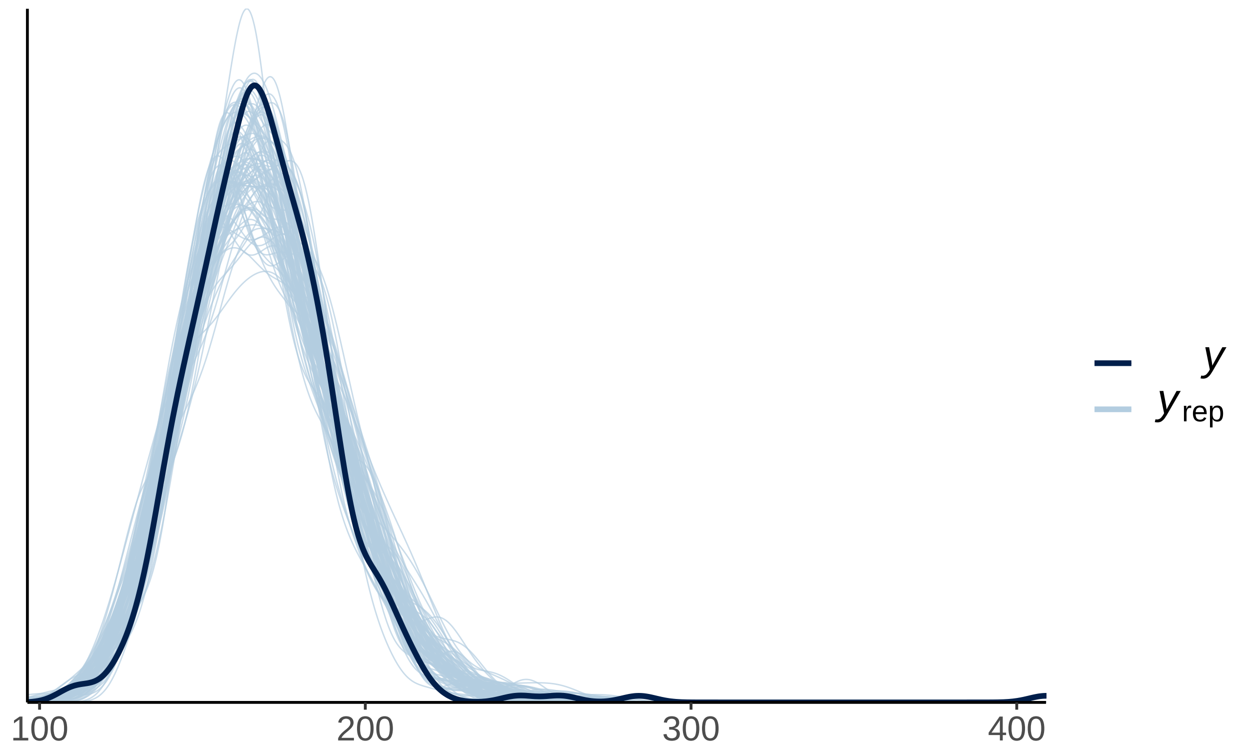


Figure 30: Posterior predictive check.

maximum each time and plot the distribution of the maximum, comparing it with the maximum value (409) in the data.

```
m2_logn<-brm(y~1+presses,noreading_data,prior = priors_log,
       iter = 2000, chains = 4,
       family = lognormal(),
       control = list(adapt_delta = 0.99,max_treedepth=15))
```

```
## Compiling the C++ model
```

```
## recompiling to avoid crashing R session
```

```
## Start sampling
```

```
postsamp<-posterior_samples(m2_logn,"^b",add_chain=TRUE)

nsim<-100
maximum<-rep(NA,nsim)
for(i in 1:nsim){
ppm2_logn<-predict(m2_logn)
maximum[i]<-max(as.vector(ppm2_logn))
}
```

```
hist(maximum,main="Distribution of maximum values",freq=FALSE)
```

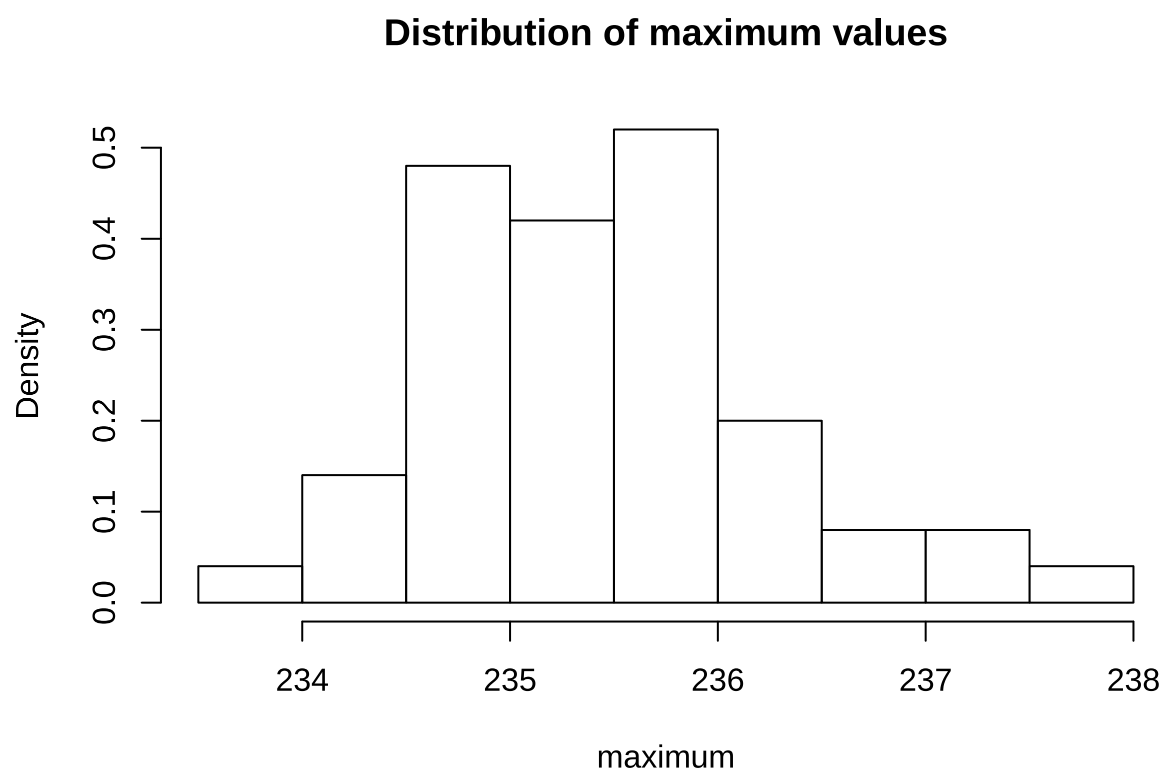


Figure 31: Distribution of maximum values in a posterior predictive check. The maximum in the data is 409 ms.

Figure 31 shows that we are unable to capture the maximum value of the observed data, so there's still room for improving the model. We will return to this question later in the course.

3.3.10 General workflow

This is the general workflow that we suggest when fitting a Bayesian model.

1. Define the full probability model:
    a. Decide on the likelihood.
    b. Decide on the priors.
    c. Write the brms or Stan model.
2. Do prior predictive checks to determine if priors make sense.

3. Check model using fake data simulations:
    a. Simulate data with known values for the parameters.
    b. Fit the model and do MCMC diagnostics.
    c. Verify that it recovers the parameters from simulated data.
4. Fit the model with real data and do MCMC diagnostics.
5. Evaluate the model's fit (e.g., posterior predictive checks, distribution of summary statistics). This may send you back to 1.
6. Inference/prediction/decisions.

7. Conduct model comparison if there’s an alternative model (to be discussed later).

3.4 Appendix - Troubleshooting problems of convergence

1. Rhat > 1.1 First of all check that there are no silly mistakes in the model. Forgetting to put parenthesis, multiplying the wrong parameters, using the wrong operation, etc. can create a model that can't converge. As our models grow in complexity there are more places where to make mistakes. Start simple, see if the model works, add complexity slowly, checking if the model converges at every step. In very rare occasions, when Rhat >1.1 and the model is correct, it may help to increase the number of iterations, but then it's usually a better idea to re-parametrize the model, see 3.

2. Stan gives a warning. The solution may also be point 1. But if the model is correctly specified, you should check Stan's website, there is a very good guide to solve problems in: http://mc-stan.org/misc/warnings.html. If this doesn't work, you may need to re-parametrize the model, see 3.

3. Some models have convergence issues because the sampler struggles to explore the parameters space. This is specially relevant in complex hierarchical models. In this case, the solution might be to re-parametrize the model. This is by no means trivial. However, the simplest parametrization trick to try is to have all the priors on the same rough scale, that is priors shouldn't have different orders of magnitude. You can find some suggestions in the chapter 21 of Stan manual (Stan Development Team 2017), and the following case study: http://mc-stan.org/users/documentation/case-studies/qr_regression.html.

4 Hierarchical linear modeling

4.1 Example 1: Reading time differences in subject vs object relatives in English

We begin with a classic question from the psycholinguistics literature: are subject relatives easier to process than object relatives? The data come from Experiment 1 in a paper by Grodner and Gibson (2005).

4.1.1 Scientific question: Is there a subject relative advantage in reading?

In two important papers, Gibson (2000) and Grodner and Gibson (2005) suggest that object relative clause sentences are more difficult to process than subject relative clause sentences because the distance between the relative clause verb *sent* and the head noun phrase of the relative clause, *reporter*, is longer in object vs subject relatives. Examples are shown below.

(1a) The *reporter* who the photographer *sent* to the editor was hoping for a good story. (object gap)

(1b) The *reporter* who *sent* the photographer to the editor was hoping for a good story. (subject gap)

The underlying explanation has to do with memory processes: shorter linguistic dependencies are easier to process due to either reduced interference or decay, or both. For implemented computational models that spell this point out, see Lewis and Vasishth (2005) and Engelmann, Jäger, and Vasishth (2018).

In the Grodner and Gibson data, the dependent measure is reading time at the relative clause verb, in milliseconds. We are expecting longer reading times in object gap sentences compared to subject gap.

4.1.2 Load data and reformat

```
library(dplyr)
gg05e1 <- read.table("data/GrodnerGibson2005E1.csv",sep=",", header=T)
```

```
## Warning in if (!header) rlabp <- FALSE: the condition has length > 1 and
## only the first element will be used

## Warning in if (header) {: the condition has length > 1 and only the first
## element will be used
```

```
gge1 <- gg05e1 %>% filter(item != 0)

gge1 <- gge1 %>% mutate(word_positionnew = ifelse(item != 15 &
                                                    word_position > 10,
                                                  word_position-1,
                                                    word_position))
#there is a mistake in the coding of word position,
#all items but 15 have regions 10 and higher coded
#as words 11 and higher

## get data from relative clause verb:
gge1crit <- subset(gge1, ( condition == "objgap" & word_position == 6 ) |
             ( condition == "subjgap" & word_position == 4 ))
```

4.1.3 Experiment design

Two important properties of these data are worth noticing.

4.1.3.1 Latin-square design

First, the design is the classic repeated measure Latin square set-up. To see what this means, first look at the number of subjects and items, and the number of rows in the data frame:

```
length(unique(gge1crit$subject))
```

```
## [1] 42
```

```
length(unique(gge1crit$item))
```

```
## [1] 16
```

```
dim(gge1crit)[1]
```

```
## [1] 672
```

There are 42 subjects and 16 items. There are $42 \times 16 = 672$ rows in the data frame. Notice also that each subject sees exactly eight object gap and eight subject gap sentences:

```
head(xtabs(~subject+condition,gge1crit),n=4)
```

```
##         condition
## subject objgap subjgap
##       1      8       8
##       2      8       8
##       3      8       8
##       4      8       8
```

The researchers created 16 sets of subject and object relatives; one set is the pair of sentences shown in (1a) and (1b) above. In the data frame, both these two items have the same id 1, but no one subject will see both sentences in any one set. For example, item 1 is seen by subject 1 in the object gap condition (1a) and item 1 is seen by subject 2 in the subject gap condition (1b):

```r
subset(gge1crit,item==1)[1:2,]
```

```
##     subject item condition word_position region_position rawRT residRT
## 6         1    1    objgap             6               3   320   -21.4
## 250       2    1   subjgap             4               3   357  -111.5
##     qcorrect experiment word_positionnew
## 6          0     tedrg3                6
## 250        1     tedrg2                4
```

Table 1: The Latin-square design in repeated measures experiments.

item id	group 1	group 2
1	objgap	subjgap
2	subjgap	objgap
3	objgap	subjgap
4	subjgap	objgap
⋮	⋮	⋮
16	subjgap	objgap

This is called a Latin-square design because of the following layout. See Table 1. Each subject is randomly assigned to Group 1 or 2, and one should have an even number of subjects in order to have a balanced data-set. Hence the 42 subjects in the Grodner and Gibson data: 21 in each group.

A useful way to ensure that you have balanced assignments of subjects to each group is to randomize the order of incoming participants in advance, such that pairs of subjects are assigned to group 1 and 2. Let order1 be such that the first subject gets group 1 and the second gets group 2, and order 2 that the first subject gets group 2 and the second group 1. Then just generate a random ordering to ensure that each pair of subjects lands in a balanced way across groups:

```r
sample(rep(c("order1","order2"),11))
```

```
##  [1] "order2" "order2" "order2" "order2" "order2" "order1" "order1"
##  [8] "order1" "order1" "order2" "order1" "order1" "order1" "order2"
## [15] "order1" "order2" "order2" "order1" "order2" "order2" "order1"
## [22] "order1"
```

Latin square designs are used in psychology and linguistics (and other areas) because they are optimal in several ways.

Soon we will need to generate a fake data-frame with a repeated measures Latin square design. We can do this using R as follows (source: Vasishth et al. 2018):

```r
library(MASS)
nitem <- 16
nsubj <- 42
## prepare data frame for two condition in a latin square design:
g1<-data.frame(item=1:nitem,
                 cond=rep(c("objgap","subjgap"),nitem/2))
g2<-data.frame(item=1:nitem,
                 cond=rep(c("objgap","subjgap"),nitem/2))

## assemble data frame in long format:
gp1<-g1[rep(seq_len(nrow(g1)),
              nsubj/2),]
gp2<-g2[rep(seq_len(nrow(g2)),
              nsubj/2),]

fakedat<-rbind(gp1,gp2)
## sanity check:
dim(fakedat)
```

```
## [1] 672   2
```

```
## add subjects:
fakedat$subj<-rep(1:nsubj,each=nitem)
fakedat<-fakedat[,c(3,1,2)]
fakedat$so<-ifelse(fakedat$cond=="objgap",1,-1)
```

For example, subject 1 sees the following conditions and items:

```
head(fakedat,n=16)
```

```
##    subj item    cond so
## 1     1    1  objgap  1
## 2     1    2 subjgap -1
## 3     1    3  objgap  1
## 4     1    4 subjgap -1
## 5     1    5  objgap  1
## 6     1    6 subjgap -1
## 7     1    7  objgap  1
## 8     1    8 subjgap -1
## 9     1    9  objgap  1
## 10    1   10 subjgap -1
## 11    1   11  objgap  1
## 12    1   12 subjgap -1
## 13    1   13  objgap  1
## 14    1   14 subjgap -1
## 15    1   15  objgap  1
## 16    1   16 subjgap -1
```

We will need this code later for fake data simulation.

4.1.3.2 Fully crossed subjects and items

In the data, because of the Latin square design, each subject sees exactly one item in one of the two conditions:

```
xtabs(~subject+item,gge1crit)
```

```
##        item
## subject 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
##      1  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      2  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      3  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      4  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      5  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      6  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      7  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      8  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      9  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      10 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      11 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      12 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      13 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      14 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      15 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      16 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      17 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      18 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      19 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      20 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      21 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      22 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      23 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      24 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      25 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      26 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      27 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      28 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      29 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      30 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      31 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      32 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      33 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      34 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      35 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      36 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      37 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##      38 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       39 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       40 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       41 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       42 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
```

If there were some zeros in the above matrix, we would have an imbalance, and this would then be *partially crossed*. This kind of imbalance arises in data-sets due to missing data, where missingness can happen due to different reasons. E.g., in eyetracking, subjects sometimes skip the critical word entirely, or there is tracking loss; these events lead to a 0 ms reading time being recorded, and this could be treated as missing data (marked as NA). We return to this point in an advanced course on Bayesian modeling, to be offered at some later date.

4.1.4 The implied generative model

The above design implies a particular statistical model that takes us beyond the linear model.

To remind you, a simple linear model of the above data would be:

$$y = \alpha + \beta * x + \varepsilon \text{ where } \varepsilon \sim Normal(0, \sigma) \tag{170}$$

Here, object gaps are coded +1, subject gaps -1. See Schad et al. (2019) for an explanation of contrast coding.

```
gge1crit$so<-ifelse(gge1crit$condition=="objgap",1,-1)
```

As figure 32 shows, a Normal likelihood doesn't seem well motivated, so we will use the log-normal.

```
plot(density(gge1crit$rawRT),main="Grodner and Gibson Expt 1",xlab="RTs (ms)")
```

4.1.4.1 Between subject variability in mean reading time

The simple linear model above would ignore the fact that we have repeated measures from multiple subjects—the iid assumption is violated. Also, different subjects that may have different mean reading times (α differ for each subject) and different object gap processing costs (β differs for each subject). Some subjects will be slower and some faster, and some may suffer more with object gaps because of lower working memory capacity, lower attention, etc. (of course, we are speculating here, we have no measurements of these individual difference variables). See Figure 33 for a visualization of between subject variability in mean reading times.

```
hist(with(gge1crit,tapply(rawRT,subject,mean)),
     main="Between subject variability",
     xlab="mean RTs",freq=FALSE)
```

In the linear model, we can express the assumption that the grand mean intercept α needs an adjustment by subject, where subjects are indexed from $j = 1, \dots, J$:

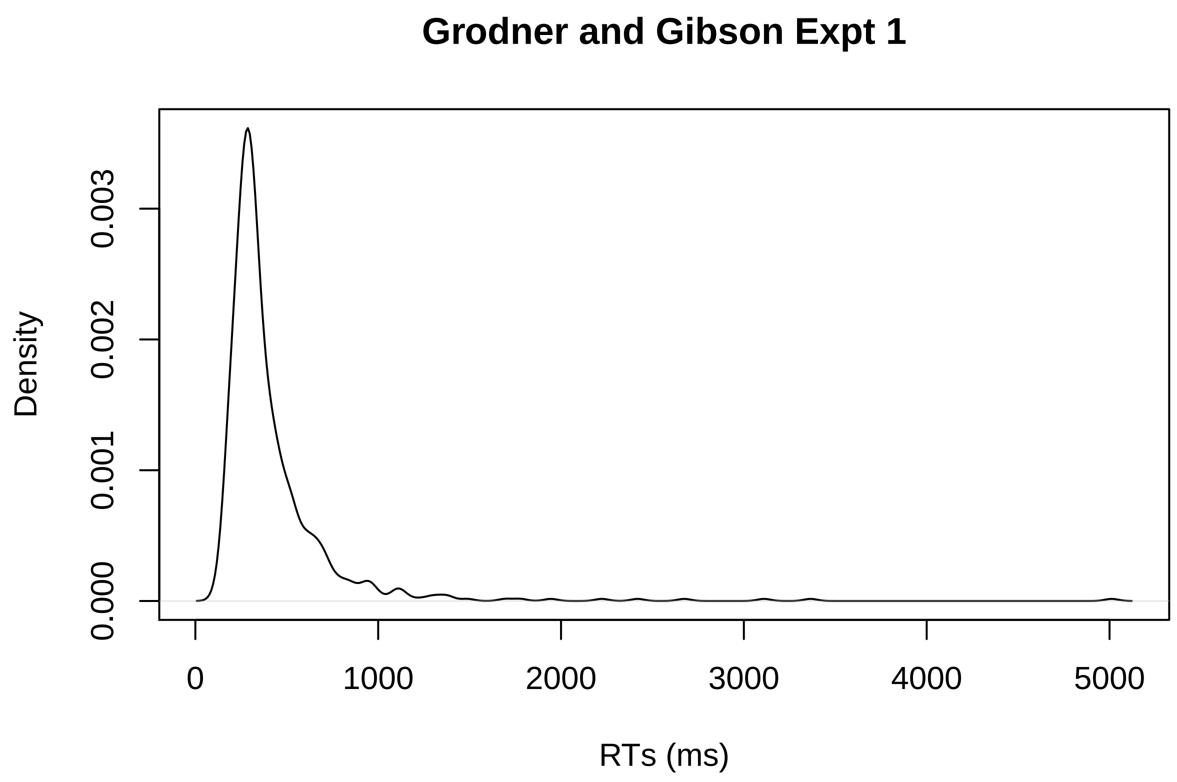


Figure 32: Distribution of reading times in the Grodner and Gibson Experiment 1 data, at the critical region.

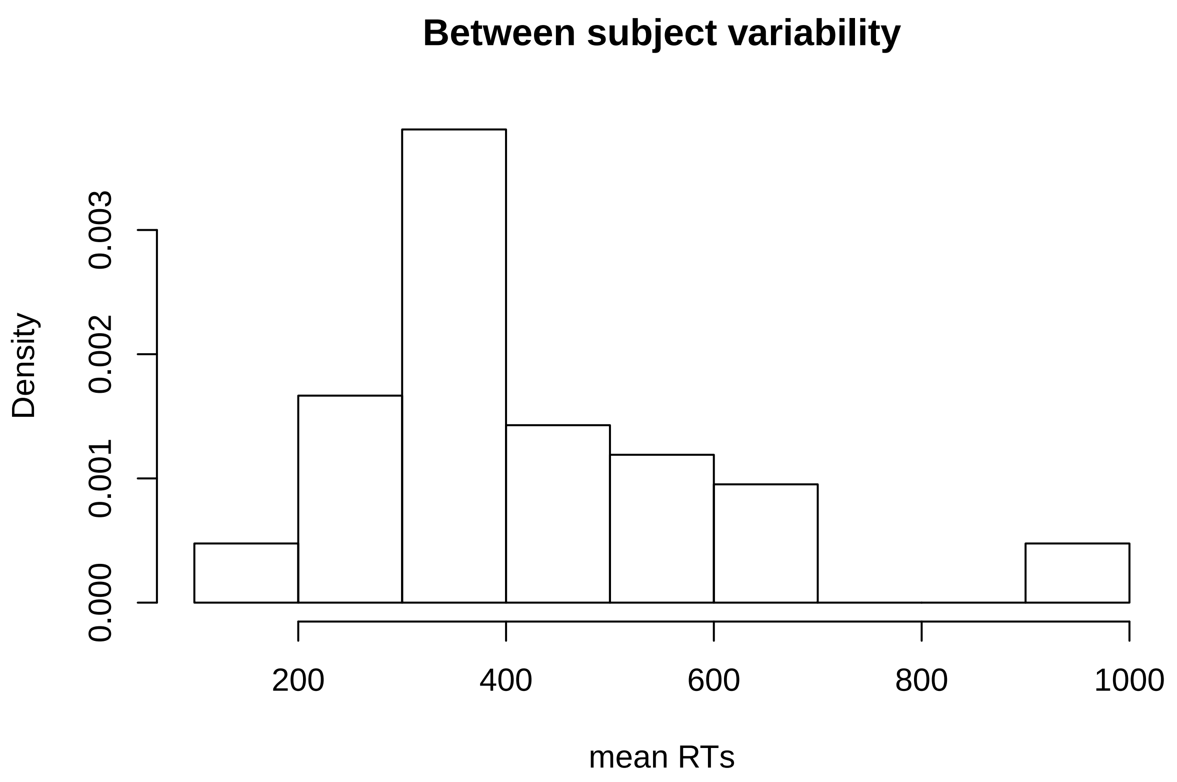


Figure 33: Between subject variability in mean reading times.

$$y_j = \alpha + u_{0j} + \beta * x_j + \varepsilon_j \tag{171}$$

where we now have two sources of variance:

- within subject variance, $\varepsilon_j \sim Normal(0, \sigma)$
- between subject variance in mean reading times, $u_{0j} \sim Normal(0, \sigma_{u0})$

In Bayes, the adjustment u_{0j} is a parameter. This is not the case in the frequentist paradigm (Bates et al. 2015).

4.1.4.2 Between item variability in mean reading time

We also see that items also differ, some would be read faster and some slower:

```
hist(with(gge1crit,tapply(rawRT,item,mean)),
     main="Between item variability",
     xlab="mean RTs",freq=FALSE)
```

For items ranging from $k = 1, \dots, K$, we can add this assumption to the model:

$$y_{kj} = \alpha + u_{0j} + w_{0k} + \beta * x_{kj} + \varepsilon_{kj} \tag{172}$$

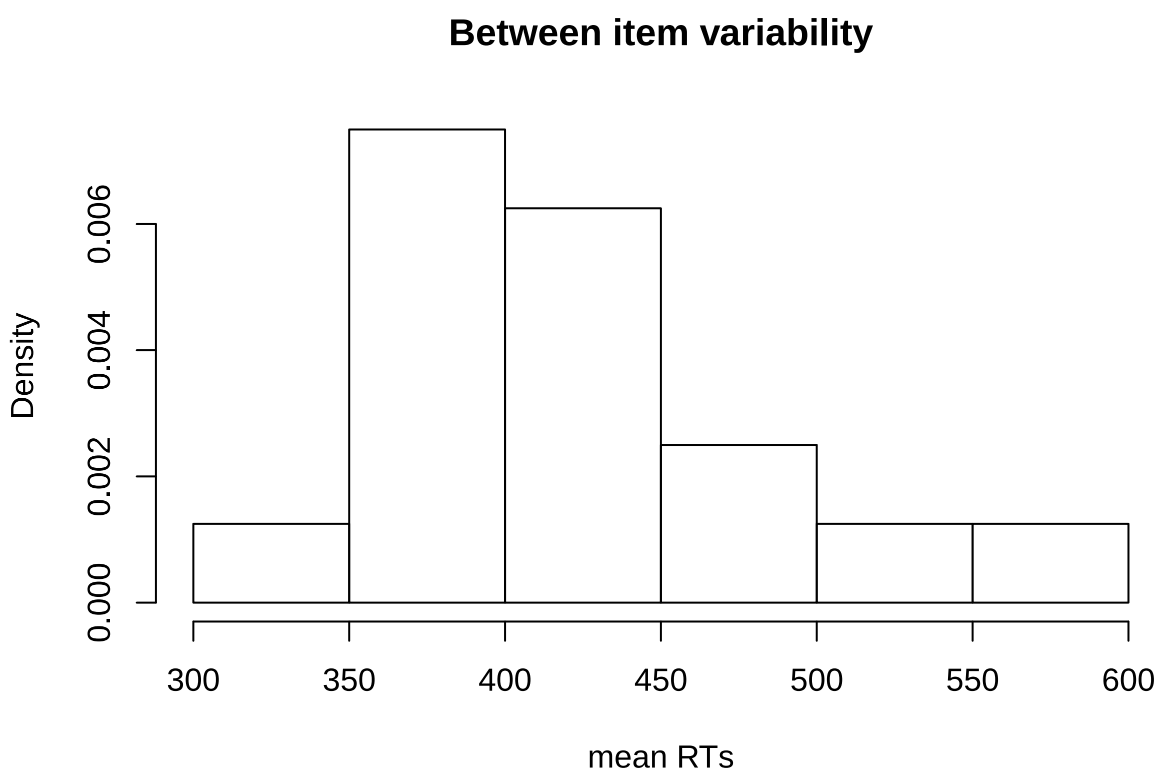


Figure 34: Between item variability in mean reading times.

where there are now three variance components:

- $\varepsilon_{kj} \sim Normal(0,\sigma)$
- $u_{0j} \sim Normal(0,\sigma_{u0})$
- between item variability in mean reading time, $w_{0k} \sim Normal(0,\sigma_{w0})$

This model is called a *varying intercepts model* with crossed varying intercepts for subjects and for items.

4.1.4.3 Between subject and between item variability in objgap cost

The object gap cost can also vary by subject and by item. See Figure 35.

```
op<-par(mfrow=c(1,2),pty="s")
meanssubj<-with(gge1crit,
                tapply(rawRT,IND=list(subject,condition),mean))
diff<-meanssubj[,2]-meanssubj[,1]

hist(diff,
     main="Between subject variability",xlab="mean objgap cost",freq=FALSE)

meansitem<-with(gge1crit,tapply(rawRT,IND=list(item,condition),mean))
diff<-meansitem[,2]-meansitem[,1]
```

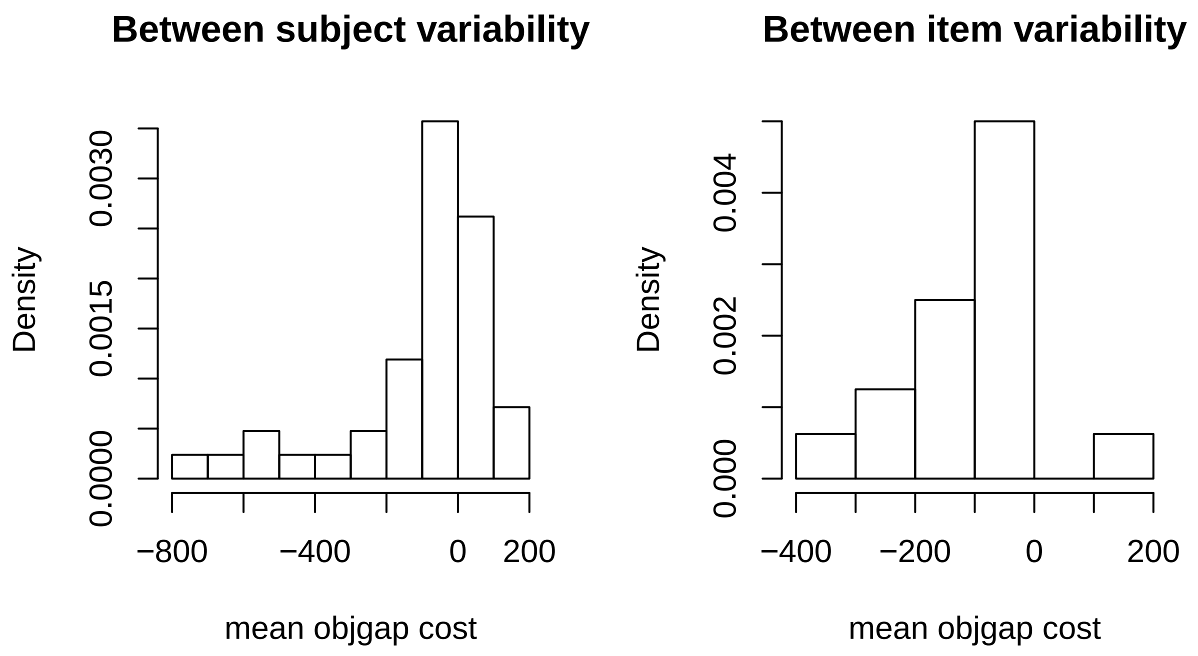


Figure 35: Between subject and item variability in object gap vs subject gap reading times.

```
hist(diff,
     main="Between item variability",xlab="mean objgap cost",freq=FALSE)
```

We can incorporate this assumption into the model by adding adjustments to the β parameter:

$$y_{kj} = \alpha + u_{0j} + w_{0k} + (\beta + u_{1j} + w_{1k}) * x_{kj} + \varepsilon_{kj} \tag{173}$$

where

- $\varepsilon_{kj} \sim Normal(0, \sigma)$
- $u_{0j} \sim Normal(0, \sigma_{u0})$
- $u_{1j} \sim Normal(0, \sigma_{u1})$
- $w_{0k} \sim Normal(0, \sigma_{w0})$
- $w_{1k} \sim Normal(0, \sigma_{w1})$

This is called the *varying intercepts and slopes* model with *no correlation* between the intercepts and slopes.

4.1.5 The maximal model

There is one detail still missing in the model: the adjustments to the intercept and slope are correlated for subjects, and also for items. In other words, we have a bivariate distribution for the subject and item random effects:

$$y_{kj} = \alpha + u_{0j} + w_{0k} + (\beta + u_{1j} + w_{1k}) * x_{kj} + \varepsilon_{kj} \tag{174}$$

where $\varepsilon_{kj} \sim Normal(0,\sigma)$ and

$$\Sigma_u = \begin{pmatrix} \sigma_{u0}^2 & \rho_u\sigma_{u0}\sigma_{u1} \\ \rho_u\sigma_{u0}\sigma_{u1} & \sigma_{u1}^2 \end{pmatrix} \quad \Sigma_w = \begin{pmatrix} \sigma_{w0}^2 & \rho_w\sigma_{w0}\sigma_{w1} \\ \rho_w\sigma_{w0}\sigma_{w1} & \sigma_{w1}^2 \end{pmatrix} \tag{175}$$

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u\right), \quad \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w\right) \tag{176}$$

This is a varying intercepts and slopes model with fully specified variance-covariance matrices for the subject and item random effects. It is sometimes called the **maximal model** (Barr et al. 2013).

4.1.6 Implementing the model

The above model is simple to implement in the Bayesian framework.

4.1.6.1 Specify and visualize priors

We define some priors first:

1. $\alpha \sim Normal(0,10)$
2. $\beta \sim Normal(0,1)$
3. Residual standard deviation: $\sigma \sim Normal_+(0,1)$
4. All other standard deviations: $\sigma \sim Normal_+(0,1)$
5. Correlation matrix: $\rho \sim LKJ(2)$.

The LKJ prior needs some explanation.

4.1.6.2 The LKJ prior on the correlation matrix

In this model, we assume that the vector $\mathbf{u} = \langle u_0, u_1 \rangle$ comes from a bivariate normal distribution with a variance-covariance matrix Σ_u. This matrix has the variances of the adjustment to the intercept and to the slope respectively along the diagonal, and the covariance on the off-diagonals.

Recall that the covariance $Cov(X,Y)$ between two variables X and Y is defined as the product of their correlation ρ and their standard deviations σ_X and σ_Y, such that, $Cov(X,Y) = \rho\sigma_X\sigma_Y$.

$$\Sigma_u = \begin{pmatrix} \sigma_{u_0}^2 & \rho_u\sigma_{u_0}\sigma_{u_1} \\ \rho_u\sigma_{u_0}\sigma_{u_1} & \sigma_{u_1}^2 \end{pmatrix} \tag{177}$$

The covariance matrix can be decomposed into a vector of standard deviations and a correlation matrix. The correlation matrix looks like this:

$$\begin{pmatrix} 1 & \rho_u \\ \rho_u & 1 \end{pmatrix} \tag{178}$$

In Stan, we write a matrix that has 0's on the off-diagonals as:

$$diag\_matrix(\sigma_{u0}, \sigma_{u1}) = \begin{pmatrix} \sigma_{u0} & 0 \\ 0 & \sigma_{u1} \end{pmatrix} \tag{179}$$

This means that we can decompose the covariance matrix into three parts:

$$\begin{aligned} \Sigma_u &= diag\_matrix(\sigma_{u0}, \sigma_{u1}) \cdot \rho_u \cdot diag\_matrix(\sigma_{u0}, \sigma_{u1}) \\ &= \begin{pmatrix} \sigma_{u0} & 0 \\ 0 & \sigma_{u1} \end{pmatrix} \begin{pmatrix} 1 & \rho_u \\ \rho_u & 1 \end{pmatrix} \begin{pmatrix} \sigma_{u0} & 0 \\ 0 & \sigma_{u1} \end{pmatrix} \end{aligned} \tag{180}$$

So we need priors for the σ_us and for ρ_u:

The basic idea of the LKJ prior is that its parameter (usually called *eta*, η, here it has value 2) increases, the prior increasingly concentrates around the unit correlation matrix (i.e., favors smaller correlation: ones in the diagonals and values close to zero in the lower and upper triangles). At $\eta = 1$, the LKJ correlation distribution is uninformative (similar to $Beta(1,1)$), at $\eta < 1$, it favors extreme correlations (similar to $Beta(a < 1, b < 1)$).

4.1.6.3 Visualize the priors

As always, it is a good idea to visualize these priors. See Figure 36.

```
priors_alpha <- c(0,10)
priors_beta <- c(0,1)
priors_sigma_e <- c(0,1)
priors_sigma_u <- c(0,1)
priors_sigma_w <- c(0,1)

## code for visualizing lkj priors:
fake_data <- list(x = rnorm(30,0,1),
                  N = 30, R = 2)

stancode <- "
data {
  int<lower=0> N;
  real x[N];
  int R;
  }
```

```
parameters {
  real mu;
  real<lower=0> sigma;
}
model {
  x ~ normal(mu,sigma);
}
generated quantities {
  corr_matrix[R] LKJ05;
  corr_matrix[R] LKJ1;
  corr_matrix[R] LKJ2;
  corr_matrix[R] LKJ4;
  LKJ05 = lkj_corr_rng(R,.5);
  LKJ1 = lkj_corr_rng(R,1);
  LKJ2 = lkj_corr_rng(R,2);
  LKJ4 = lkj_corr_rng(R,4);
}
"

fitfake <- stan(model_code = stancode, pars = c("LKJ05","LKJ1","LKJ2","LKJ4"),
                data = fake_data, chains = 4,
                iter = 2000)

corrs<-extract(fitfake,pars=c("LKJ05[1,2]","LKJ1[1,2]","LKJ2[1,2]","LKJ4[1,2]"))
```

```
op<-par(mfrow=c(2,3),pty="s")
par(oma = rep(0, 4),
    mar = c(2.7, 2.7, 0.1, 0.1),
    mgp = c(1.7, 0.4, 0))
b<-seq(-priors_alpha[2]*2,
       priors_alpha[2]*2,by=0.01)
plot(b,dnorm(b,mean=priors_beta[1],
             sd=priors_beta[2]),
     type="l",ylab="density",
     xlab=expression(alpha),ylim=c(0, 0.5))
plot(b,dnorm(b,mean=priors_beta[1],
             sd=priors_beta[2]),
     type="l",ylab="density",
     xlab=expression(beta),ylim=c(0, 0.5))
sig<-seq(0,priors_sigma_e[2]*3,by=0.01)
plot(sig,dnorm(sig,mean=priors_sigma_e[1],
               sd=priors_sigma_e[2]),type="l",ylab="density",
     xlab=expression(sigma[e]))
plot(sig,dnorm(sig,mean=priors_sigma_u[1],
```

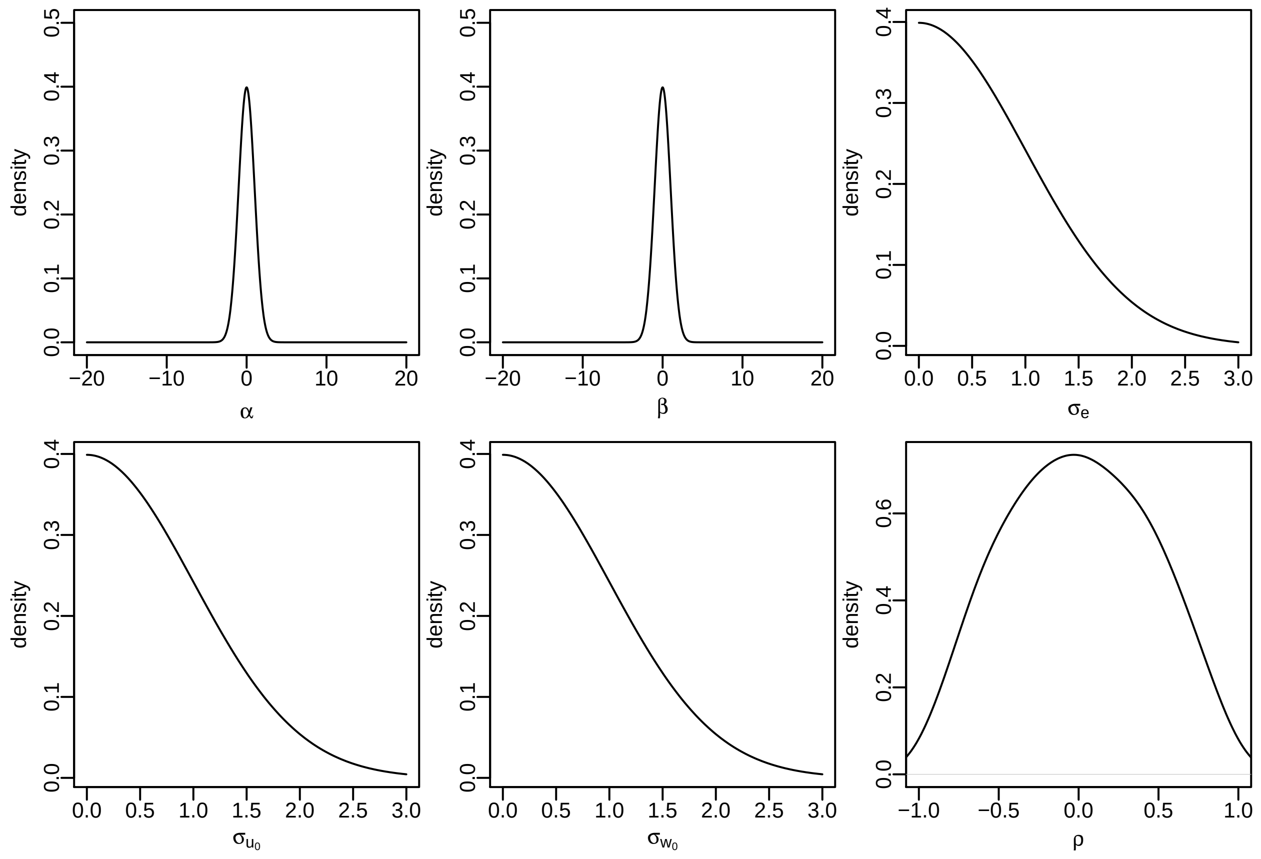


Figure 36: Priors for the Godner and Gibson data.

```
               sd=priors_sigma_u[2]),type="l",ylab="density",
     xlab=expression(sigma[u[0]]))
plot(sig,dnorm(sig,mean=priors_sigma_u[1],
               sd=priors_sigma_u[2]),type="l",ylab="density",
     xlab=expression(sigma[w[0,1]]))
plot(density(corrs[[3]],bw=0.15),
     ylab="density",xlab=expression(rho),
     xlim=c(-1,1),main="")
```

We will return later to the question of whether these priors are appropriate and reasonable (short answer: no).

4.1.7 Fit the model using brms

```
priors <- c(set_prior("normal(0, 10)", class = "Intercept"),
                      set_prior("normal(0, 1)", class = "b",
                                coef = "so"),
                      set_prior("normal(0, 1)", class = "sd"),
                      set_prior("normal(0, 1)", class = "sigma"),
                      set_prior("lkj(2)", class = "cor"))

m_gg<-brm(rawRT~so + (1+so|subject) + (1+so|item),gge1crit,family=lognormal(),
    prior=priors)
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```
summary(m_gg)
```

```
##  Family: lognormal
##   Links: mu = identity; sigma = identity
## Formula: rawRT ~ so + (1 + so | subject) + (1 + so | item)
##    Data: gge1crit (Number of observations: 672)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~item (Number of levels: 16)
##                     Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sd(Intercept)           0.04      0.02     0.00     0.09        909 1.01
## sd(so)                  0.04      0.02     0.00     0.10       1372 1.00
## cor(Intercept,so)       0.25      0.42    -0.67     0.89       1347 1.01
##
## ~subject (Number of levels: 42)
##                     Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sd(Intercept)           0.33      0.04     0.26     0.42        777 1.00
## sd(so)                  0.11      0.02     0.07     0.16       1543 1.00
## cor(Intercept,so)       0.51      0.17     0.15     0.79       1763 1.00
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept       5.88      0.05     5.78     5.99        473 1.00
## so              0.06      0.03     0.01     0.11       1484 1.00
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## sigma     0.36      0.01     0.34     0.39       3021 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

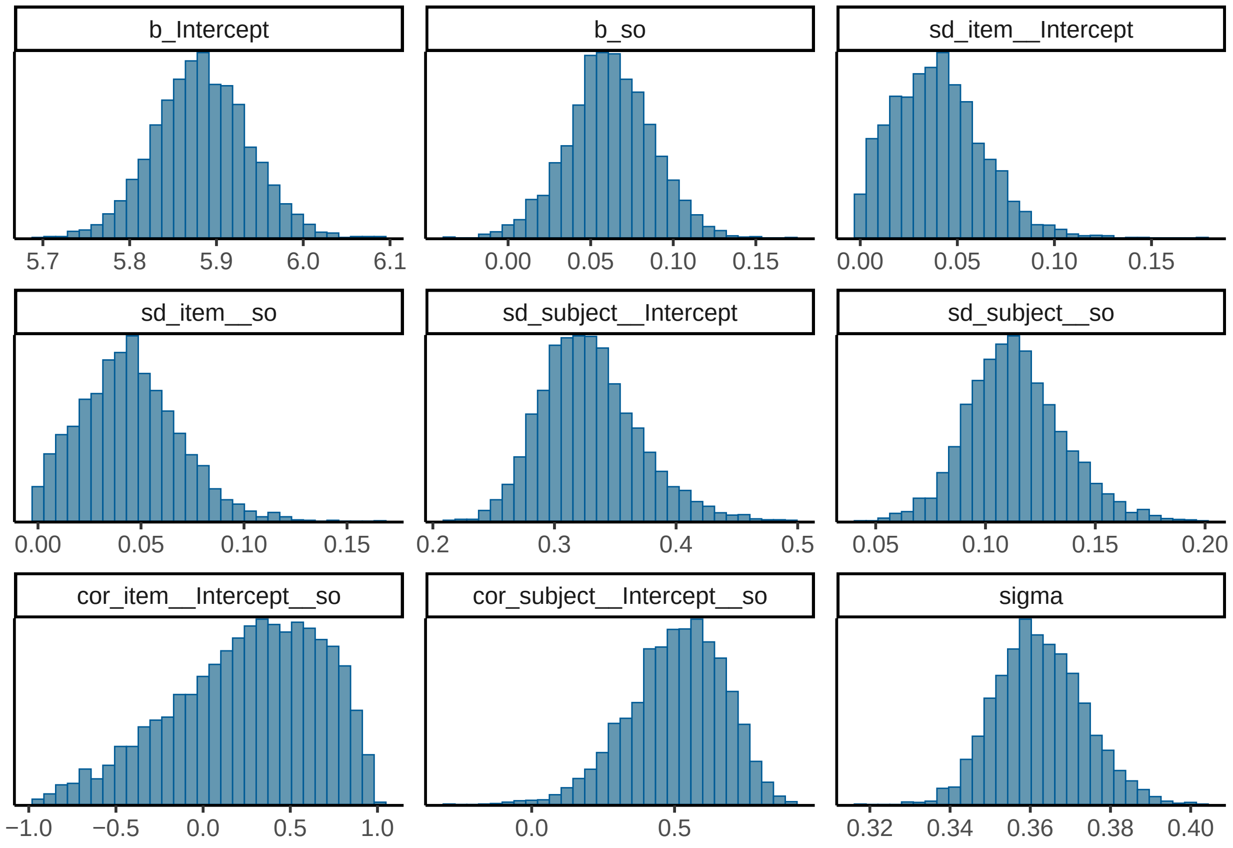


Figure 37: Posterior distributions of parameters in the Grodner and Gibson data.

```
stanplot(m_gg,type="hist")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Figure 37 shows the posterior distributions of the parameters on the log ms scale (for the coefficients and standard deviations). Notice that

- The object relative takes longer to read than the subject relative, as predicted. We know this because the parameter b_so is positive.
- The largest sources of variance are the subject intercepts, slopes, and the residual standard deviation. Look at the sd_subject parameters, and sigma.
- The by-item variance components are relatively small. Look at the sd_item parameters.
- The correlations have very wide uncertainty—the prior is dominating in determining the posteriors as there isn't that much data to obtain accurate estimates of these parameters. Look at the cor parameters.

4.1.8 Examine by subject random effects visually

First, extract the posterior samples of the parameters that we will need to compute individual differences.

```r
library(bayesplot)
postgg<-posterior_samples(m_gg)
## extract variances:
alpha<-postgg$b_Intercept
beta<-postgg$b_so
cor<-posterior_samples(m_gg,"^cor")
sd<-posterior_samples(m_gg,"^sd")
sigma<-posterior_samples(m_gg,"sigma")

## item random effects won't be used below
item_re<-posterior_samples(m_gg,"^r_item")
subj_re<-posterior_samples(m_gg,"^r_subj")
```

4.1.8.1 By subject intercept adjustments

Figure 38 shows the adjustments to the intercept (α) by subject. This is on the log scale. Here, we are looking at the parameters u_0 in the model, which are assumed to be generated from $Normal(0, \sigma_{u0})$. The between subject variability is being captured here by this subject level parameter.

```r
subjint<-subj_re[,1:42]
colnames(subjint)<-c(paste("u0,",1:42,sep=""))
intmns <- colMeans(subjint)
subjint<-subjint[,order(intmns)]
mcmc_areas(subjint)
```

```
## Warning: Unquoting language objects with `!!!` is deprecated as of rlang 0.4.0.
## Please use `!!` instead.
##
##   # Bad:
##   dplyr::select(data, !!!enquo(x))
##
##   # Good:
##   dplyr::select(data, !!enquo(x))    # Unquote single quosure
##   dplyr::select(data, !!!enquos(x))  # Splice list of quosures
##
## This warning is displayed once per session.
```

4.1.8.2 By subject slope adjustments

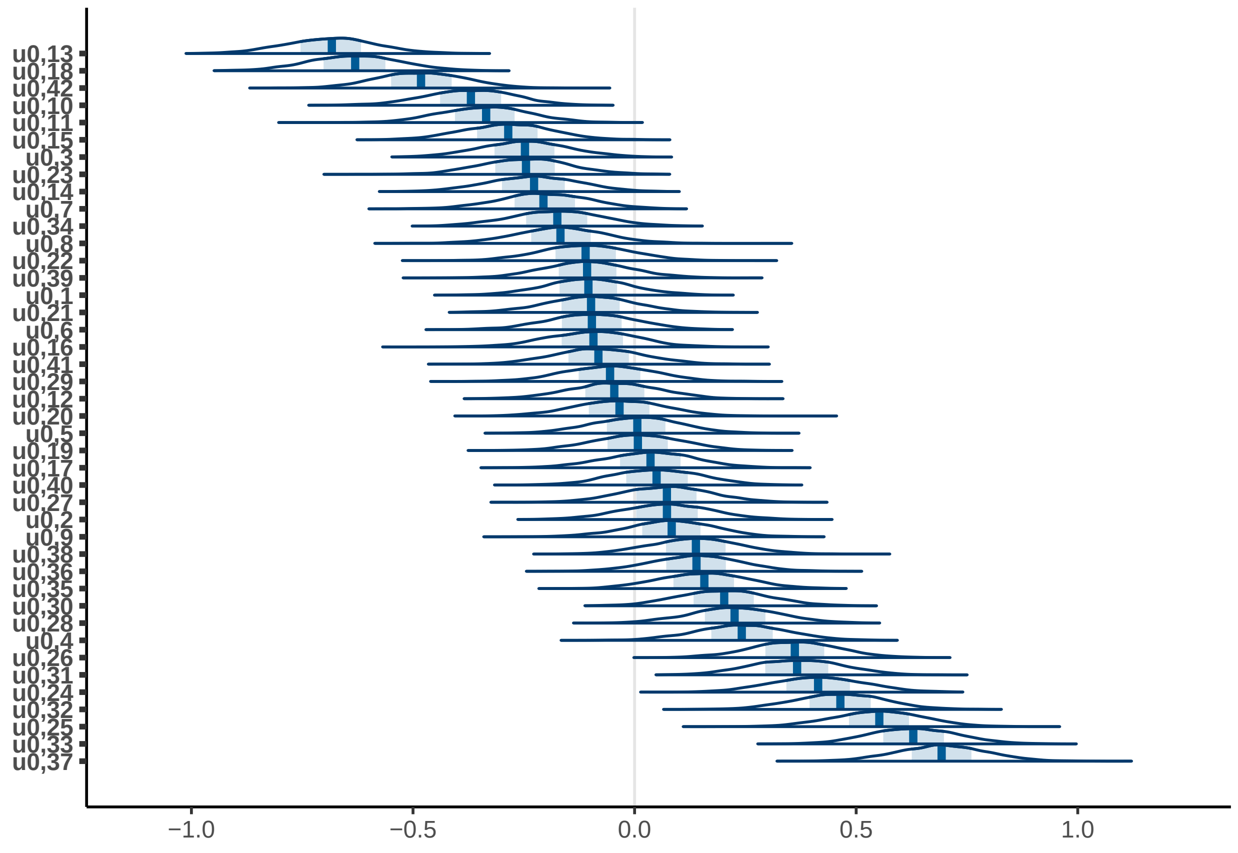


Figure 38: Variability in subject intercept adjustments in the Grodner and Gibson data.

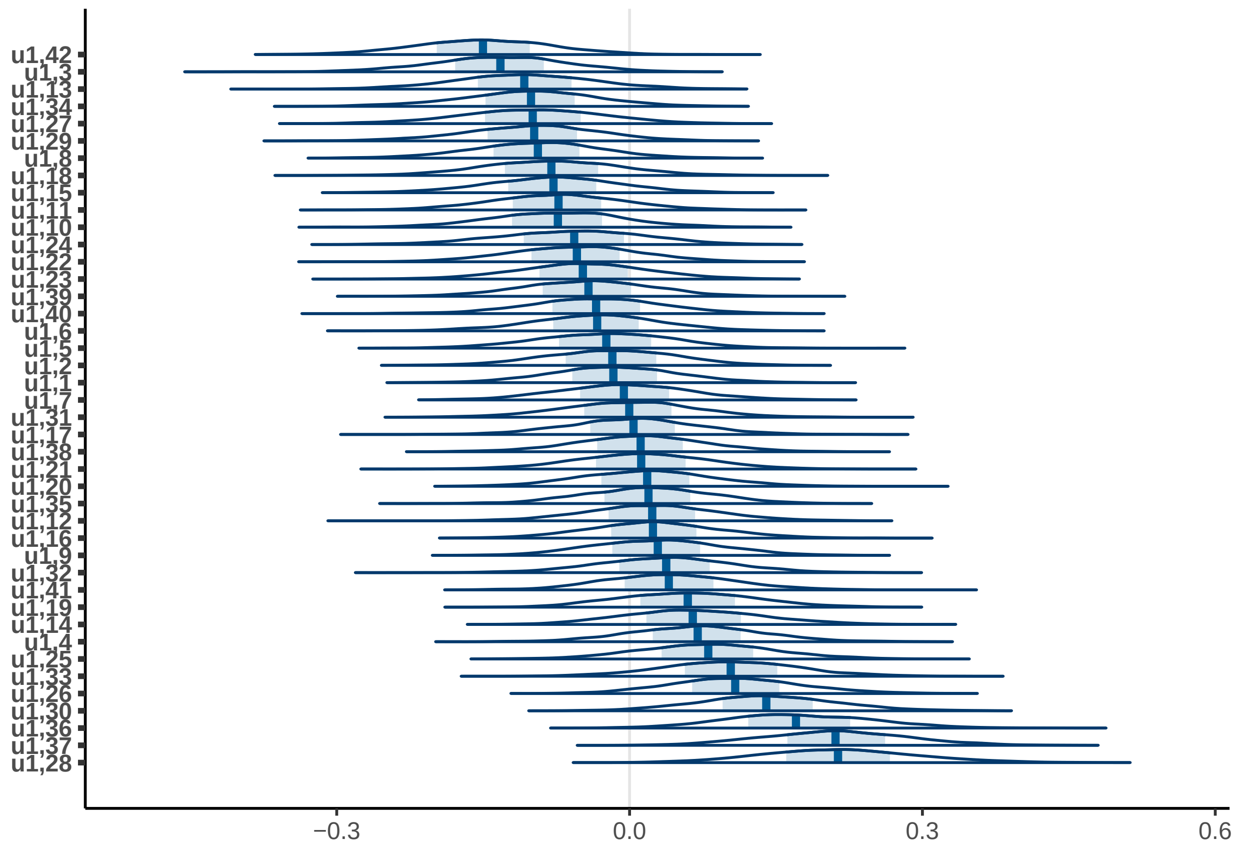


Figure 39: Variability in subject slope adjustments in the Grodner and Gibson data.

Figure 39 shows the by-subject adjustments to the OR processing cost effect (β). Here, the assumption is that these adjustments are $u_1 \sim Normal(0, \sigma_{u_1})$.

```
subjslope<-subj_re[,(1:42)+42]
colnames(subjslope)<-c(paste("u1,",1:42,sep=""))
slopemns <- colMeans(subjslope)
subjslope<-subjslope[,order(slopemns)]
mcmc_areas(subjslope)
```

The correlation between u_0 and u_1 is represented by the correlation parameter ρ_u.

```
stanplot(m_gg,type="hist",
         pars="cor_subject__Intercept__so")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

4.1.9 Examine mean and individual differences on the raw ms scale

It is useful to see the effects on the raw ms scale. The log ms scale is difficult to interpret.

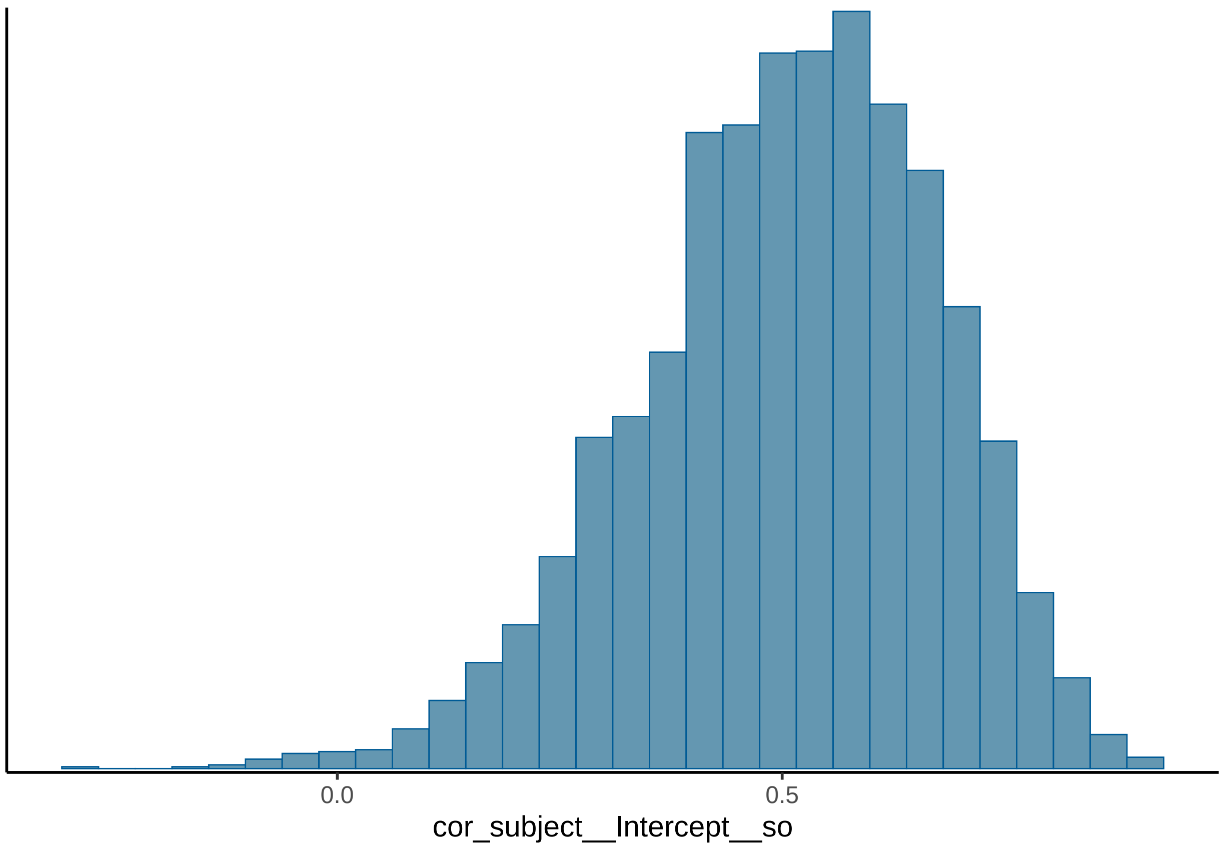


Figure 40: Posterior distributions of subject varying intercept and slope correlation parameter in the Grodner and Gibson data.

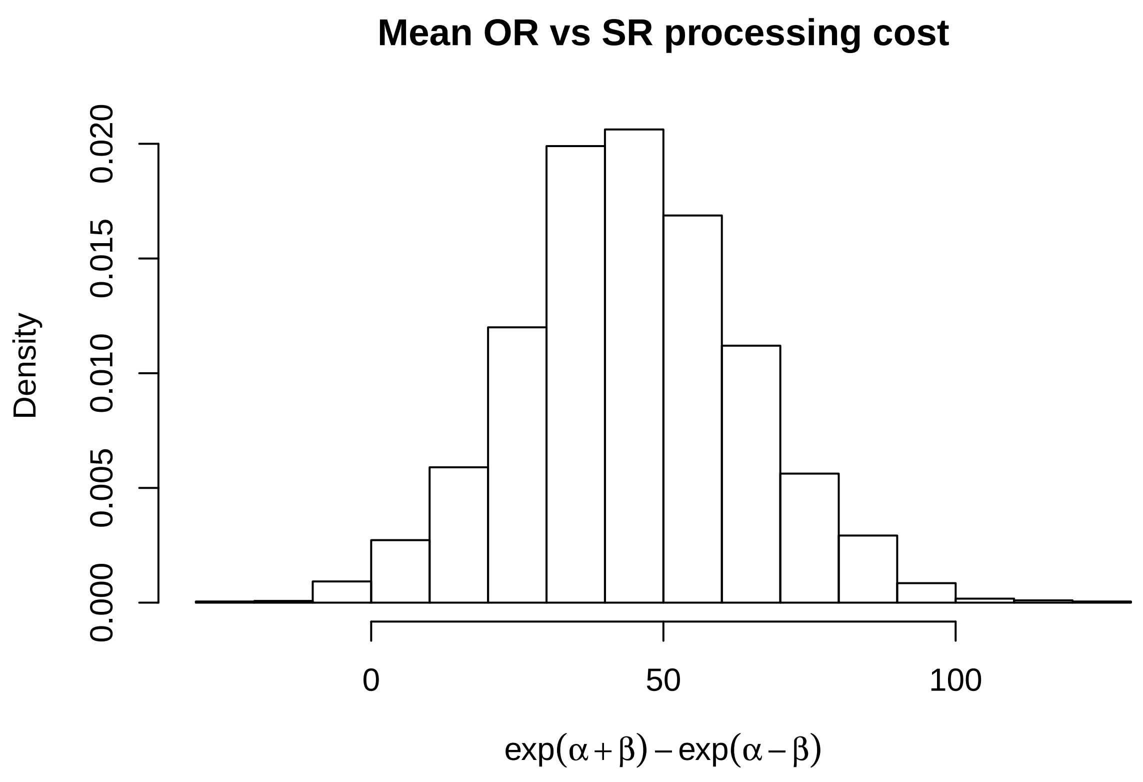


Figure 41: Mean OR processing cost effect in the Grodner and Gibson data.

4.1.9.1 Mean difference

In psychology and linguistics, undue emphasis is paid on reporting the overall mean effect. Figure 41 shows this—there seems to be a clear OR processing cost effect.

```
meandiff<- exp(alpha + beta) - exp(alpha - beta)
mean(meandiff)
```

```
## [1] 44.7
```

```
round(quantile(meandiff,prob=c(0.025,0.975)),0)
```

```
##  2.5% 97.5%
##     7    84
```

Plot the histogram of the object relative processing cost in ms:

```
hist(meandiff,freq=FALSE,
     main="Mean OR vs SR processing cost",
     xlab=expression(exp(alpha + beta)- exp(alpha - beta)))
```

4.1.9.2 Individual effects of OR processing cost

However, only three out of 42 subjects show clear evidence for OR processing cost. See Figure 42.

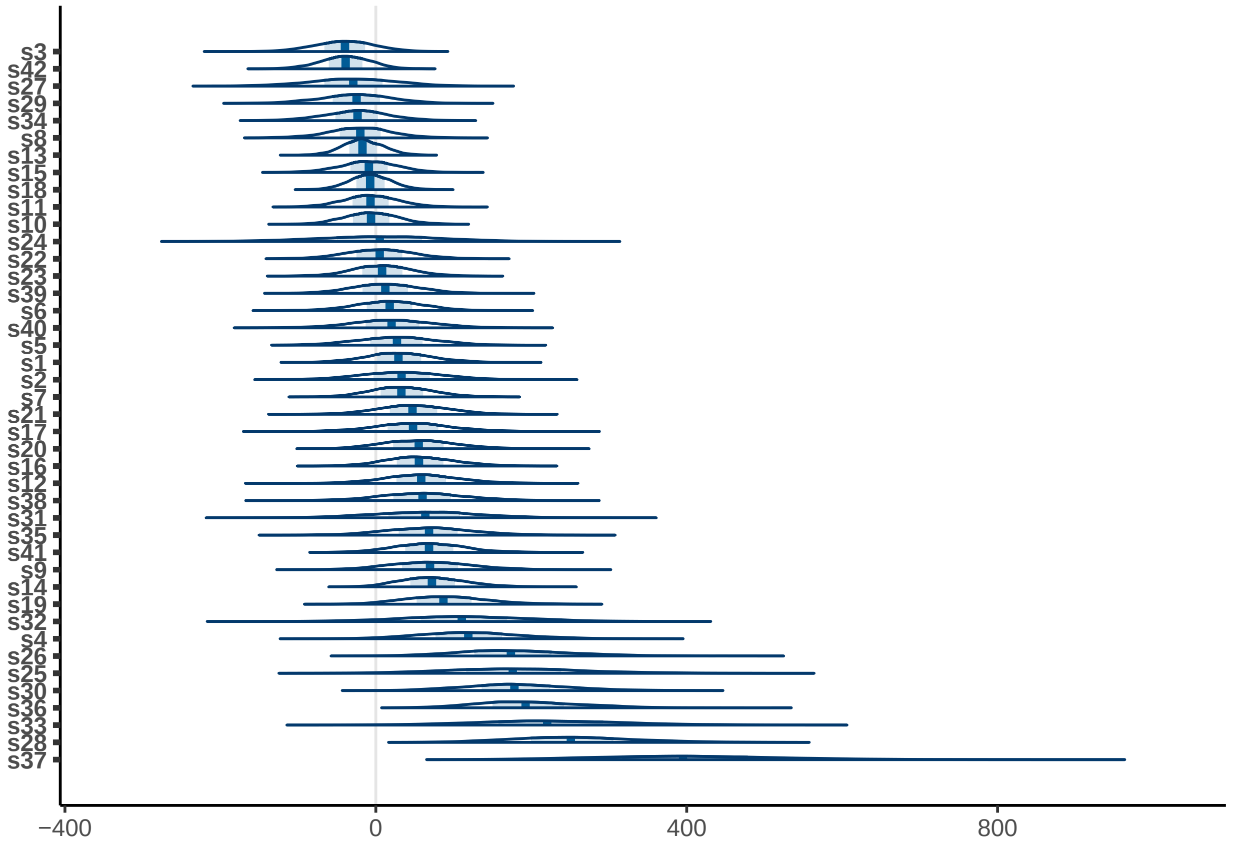


Figure 42: Variability in subject OR processing cost effect in the Grodner and Gibson data.

```
subjdiff<-matrix(rep(NA,42*4000),nrow=42)
for(i in 1:42){
subjdiff[i,]<-exp(alpha + subj_re[,i]  + (beta+subj_re[,i+42])) -
  exp(alpha + subj_re[,i] -
        (beta+subj_re[,i+42]))
}

subjdiff<-t(subjdiff)

subjdiff<-as.data.frame(subjdiff)
colnames(subjdiff)<-paste("s",c(1:42),sep="")
mns <- colMeans(subjdiff)
subjdiff<-subjdiff[,order(mns)]
mcmc_areas(subjdiff)
```

This illustrates a point that Blastland and Spiegelhalter (2014) make: “The average is an abstraction. The reality is variation.” Any theory of sentence processing would have to be able to reproduce the pattern of individual differences observed, with only a few participants showing an OR processing cost and the majority showing equivocal conclusions. The average effect doesn’t represent the behavior of many individuals in this sample.

4.1.10 To make discovery claims, calibrate the true and false discovery rate

Suppose that, based on these data and this model, we want to claim that there is a mean OR processing cost in English. In order to make a discovery claim, we need to understand the **true discovery rate** of this effect. In the frequentist world, this would be the *statistical power*, the probability of detecting an effect if there is in fact one.

First, we write a function to generate fake data:

```r
library(MASS)
gen_fake_lnorm <- function(nitem=16,nsubj=42,
alpha=NULL,beta=NULL,
                        Sigma_u=NULL,Sigma_w=NULL,sigma_e=NULL){
  ## prepare data frame for two condition in a latin square design:
g1<-data.frame(item=1:nitem,
                 cond=rep(c("objgap","subjgap"),nitem/2))
g2<-data.frame(item=1:nitem,
                 cond=rep(c("objgap","subjgap"),nitem/2))

## assemble data frame in long format:
gp1<-g1[rep(seq_len(nrow(g1)),
              nsubj/2),]
gp2<-g2[rep(seq_len(nrow(g2)),
              nsubj/2),]

fakedat<-rbind(gp1,gp2)

## add subjects:
fakedat$subj<-rep(1:nsubj,each=nitem)
fakedat<-fakedat[,c(3,1,2)]
fakedat$so<-ifelse(fakedat$cond=="objgap",1,-1)

## subject random effects:
  u<-mvrnorm(n=length(unique(fakedat$subj)),
            mu=c(0,0),Sigma=Sigma_u)
  ## item random effects
  w<-mvrnorm(n=length(unique(fakedat$item)),
            mu=c(0,0),Sigma=Sigma_w)
  ## generate data row by row:
  N<-dim(fakedat)[1]
  rt<-rep(NA,N)
  for(i in 1:N){
    rt[i] <- rlnorm(1,alpha +
                      u[fakedat[i,]$subj,1] +
                      w[fakedat[i,]$item,1] +
                      (beta+u[fakedat[i,]$subj,2]+
                         w[fakedat[i,]$item,2])*fakedat$so[i],
                   sigma_e)}
  fakedat$rt<-rt
  fakedat$subj<-factor(fakedat$subj); fakedat$item<-factor(fakedat$item)
  fakedat}
```

Next, we extract the parameter means from the Bayesian model, and assemble the variance covariance matrices for the subject and item random effects.

```
sds<-colMeans(sd)
cors<-colMeans(cor)
sig<-mean(sigma$sigma)
Sigma_u<-diag(sds[3:4]^2)
Sigma_u[1,2]<-Sigma_u[2,1]<-cors[2]*sds[3]*sds[4]
Sigma_w<-diag(sds[1:2]^2)
Sigma_w[1,2]<-Sigma_w[2,1]<-cors[1]*sds[1]*sds[2]
```

Then, we run 50 simulations, computing the 95% credible interval of the OR processing cost effect. Because this is a very time-consuming calculation, we are going to use previously computed values.

```
nsim<-50
betaquants<-matrix(rep(NA,nsim*2),ncol =2)
betameans<-matrix(rep(NA,nsim),ncol =2)

for(i in 1:nsim){
gg_fake<-gen_fake_lnorm(alpha=mean(alpha),
                        beta=mean(beta),
               Sigma_u=Sigma_u,Sigma_w=Sigma_w,
               sigma_e=sig)

m_gg_fake<-brm(rt~so + (1+so|subj) + (1+so|item),gg_fake,family=lognormal(),
    prior=priors,
    control = list(adapt_delta = 0.99,max_treedepth=15))
betapost<-posterior_samples(m_gg_fake)$b_so
betaquants[i,]<-quantile(betapost,prob=c(0.025,0.975))
betameans[i]<-mean(betapost)
}
save(betameans,
     file="data/truediscoverymeans.Rda")
save(betaquants,
     file="data/truediscoveryquants.Rda")
```

This simulation gives us an estimate of the proportion of times we would be able to detect an effect with 16 items and 42 subjects, assuming that the Grodner and Gibson data reflect a real difference between the two relative clause types.

Assuming that we are willing to declare an effect just in case 0 is not included in the 95% credible interval of the effect, the above simulation shows that we would detect the effect in only half of the repeated experiments.

```
> length(which(betaquants[,1]>0))/50
[1] 0.5
```

Thus, the true discovery rate is quite low. One would want the true discovery rate to be at least 80%.

We can also investigate the false discovery rate—the proportion of times we would declare that we found an effect, when there is none. In frequentist statistics, this is called Type I error. The only change needed in the above simulation is to set β to 0, to reflect the assumption that there is no effect.

4.1.11 Posterior predictive checks

Figure 43 shows that we have reasonable coverage over the observed data.

```
pp_check(m_gg, nsamples = 100)+
  theme(text = element_text(size=16),
        legend.text=element_text(size=16))
```

4.2 Example 2: Question-response accuracies (Logistic regression)

The Grodner and Gibson (2005) data also has question-response accuracies: 1 if the response to a question following the sentence was correct, 0 otherwise. We show only the relevant columns below:

```
head(gge1crit[,c(1,2,3,8,11)])
```

```
##    subject item condition qcorrect so
## 6        1    1    objgap        0  1
## 19       1    2   subjgap        1 -1
## 34       1    3    objgap        0  1
## 49       1    4   subjgap        1 -1
## 68       1    5    objgap        1  1
## 80       1    6   subjgap        1 -1
```

One could aggregate the accuracy by item, and then just fit a hierarchical linear model:

```
meanp<-with(gge1crit,tapply(qcorrect,
                     IND=list(condition,subject),
                     mean))
q_df<-data.frame(subj=rep(c(1:42),2),
```

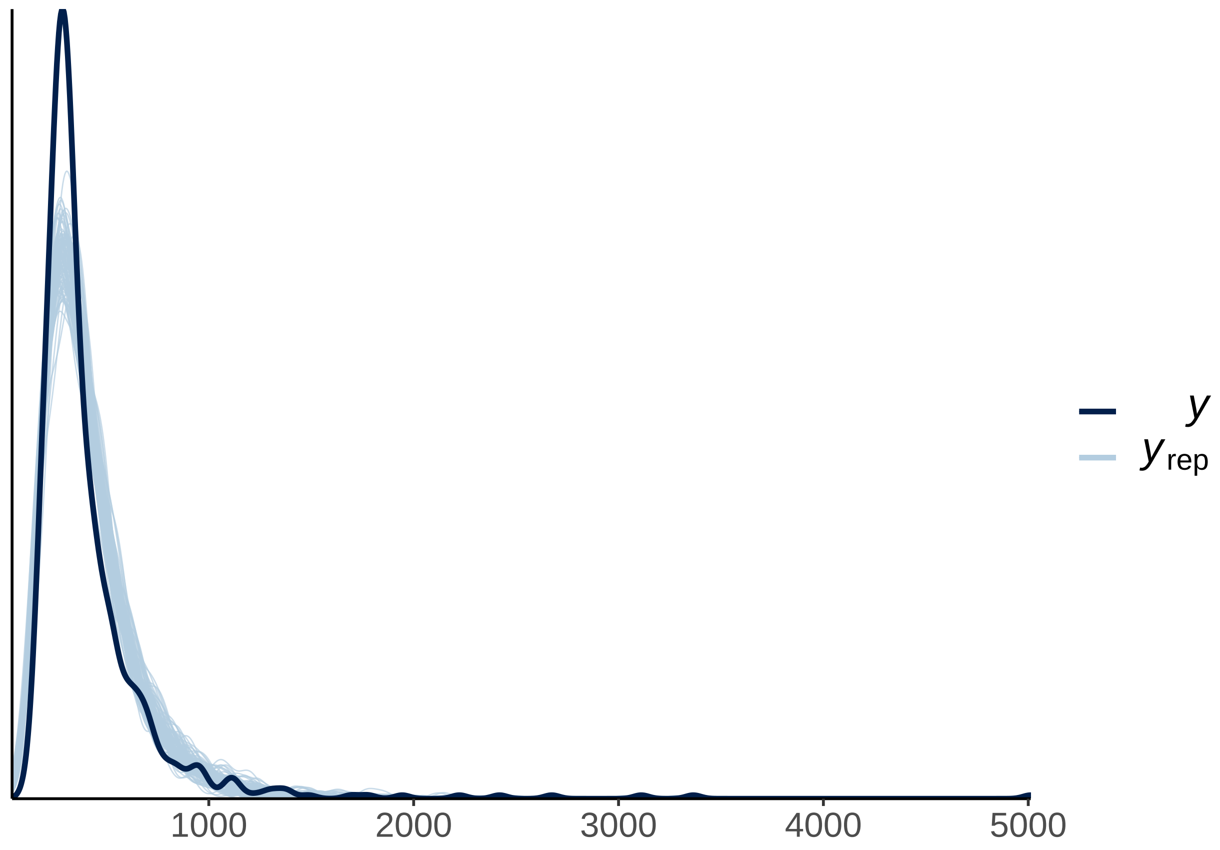


Figure 43: Posterior predictive check for the Grodner and Gibson data.

```
           so=rep(c(1,-1),each=42),
           p=c(meanp[1,],meanp[2,]))

head(q_df)
```

```
##   subj so     p
## 1    1  1 0.750
## 2    2  1 0.875
## 3    3  1 1.000
## 4    4  1 1.000
## 5    5  1 0.875
## 6    6  1 1.000
```

```
mqlmer<-lmer(p~so+(1|subj),q_df)
```

```
## boundary (singular) fit: see ?isSingular
```

```
summary(mqlmer)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: p ~ so + (1 | subj)
##    Data: q_df
##
## REML criterion at convergence: -97.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0372 -0.7770 -0.0706  0.9182  1.2008
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subj     (Intercept) 0.000    0.000
##  Residual             0.016    0.126
## Number of obs: 84, groups:  subj, 42
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   0.8661     0.0138   62.79
## so            0.0179     0.0138    1.29
##
## Correlation of Fixed Effects:
##    (Intr)
## so 0.000
## convergence code: 0
## boundary (singular) fit: see ?isSingular
```

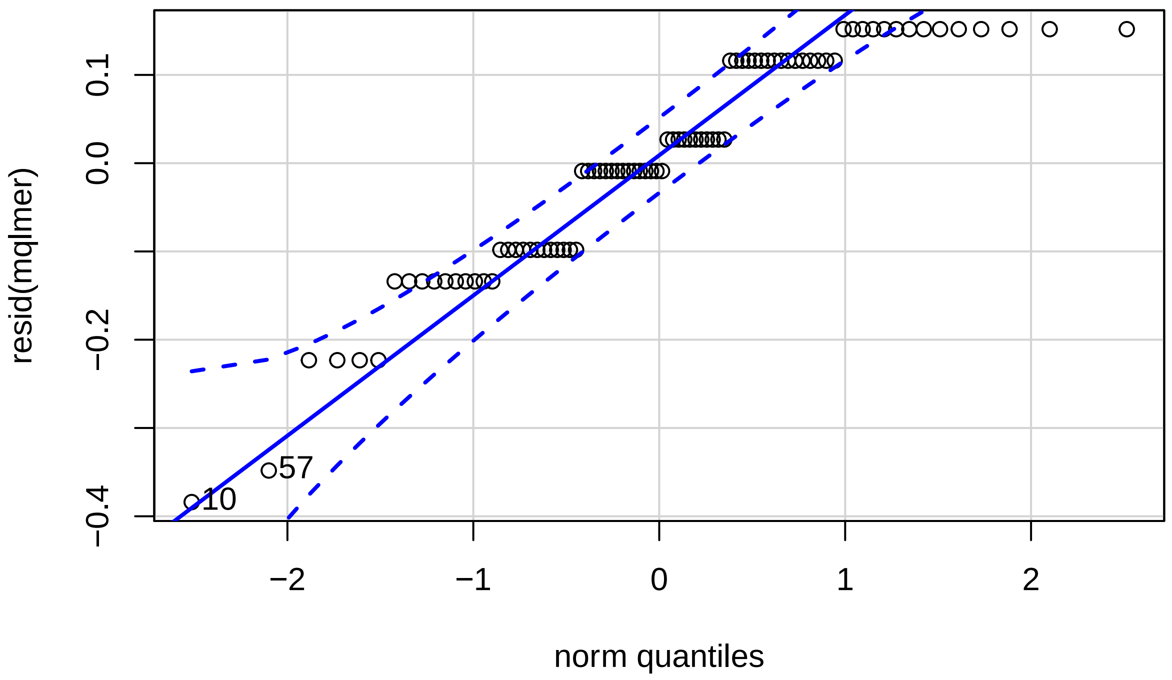


Figure 44: Residuals of the aggregated accuracy model for the question responses in the Grodner and Gibson data.

Figure 44 shows that the residuals don't really look particularly normal: the model assumption that $\varepsilon \sim Normal(0, \sigma)$ is difficult to defend.

```
library(car)
qqPlot(resid(mqlmer))
```

```
## [1] 10 57
```

Furthermore, think about the generative process; a 0,1 response is best seen as generated by a Bernoulli distribution with probability of success p: response $\sim Bernoulli(p)$.

One can therefore model each 0,1 response as being generated from a Bernoulli distribution, which is just a Binomial with a single trial. Thus, what is of interest is the probability of correct responses in subject vs object relatives:

```
round(100*with(gge1crit,
               tapply(qcorrect,condition,mean)))
```

```
##  objgap subjgap
##      88      85
```

We will transform the probability p of a correct response to a log-odds:

$$\log \frac{p}{1-p} \tag{181}$$

and assume that the log-odds of a correct response is affected by the relative clause type:

$$\log \frac{p}{1-p} = \alpha + \beta * so \tag{182}$$

This model is called a *logistic* regression because it uses the logistic or logit function to transform p to log odds space. Notice that there is no residual term in this model.

We can fit the above model easily using brms:

```
m_gg_q1<-brm(qcorrect~so,gge1crit,family=bernoulli(link="logit"))
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```
summary(m_gg_q1)
```

Obviously, because the question-response data are also repeated measures, we must use a hierarchical linear model, with varying intercepts and slopes for subject and item, as in Example 1:

```
m_gg_q2<-brm(qcorrect~so+(1+so|subject) + (1+so|item),
             gge1crit,family=bernoulli(link="logit"))
```

```
## Compiling the C++ model
```

```
## Start sampling
```

```
summary(m_gg_q2)
```

This model is not especially good because many of the response accuracies are at ceiling. However, in principle this kind of model is appropriate for binary responses.

4.2.1 Convert posteriors back to probability space

What is theoretically important is the posterior distribution of the difference between object and subject relative response accuracy. That is on the probability scale. We can go from log-odds space to probability space by solving this equation for p.

Using simple algebra, we can go from:

$$\log \frac{p}{1-p} = \alpha + \beta * so = \mu \tag{183}$$

to:

$$p = \exp(\mu)/(1+\exp(\mu)) \tag{184}$$

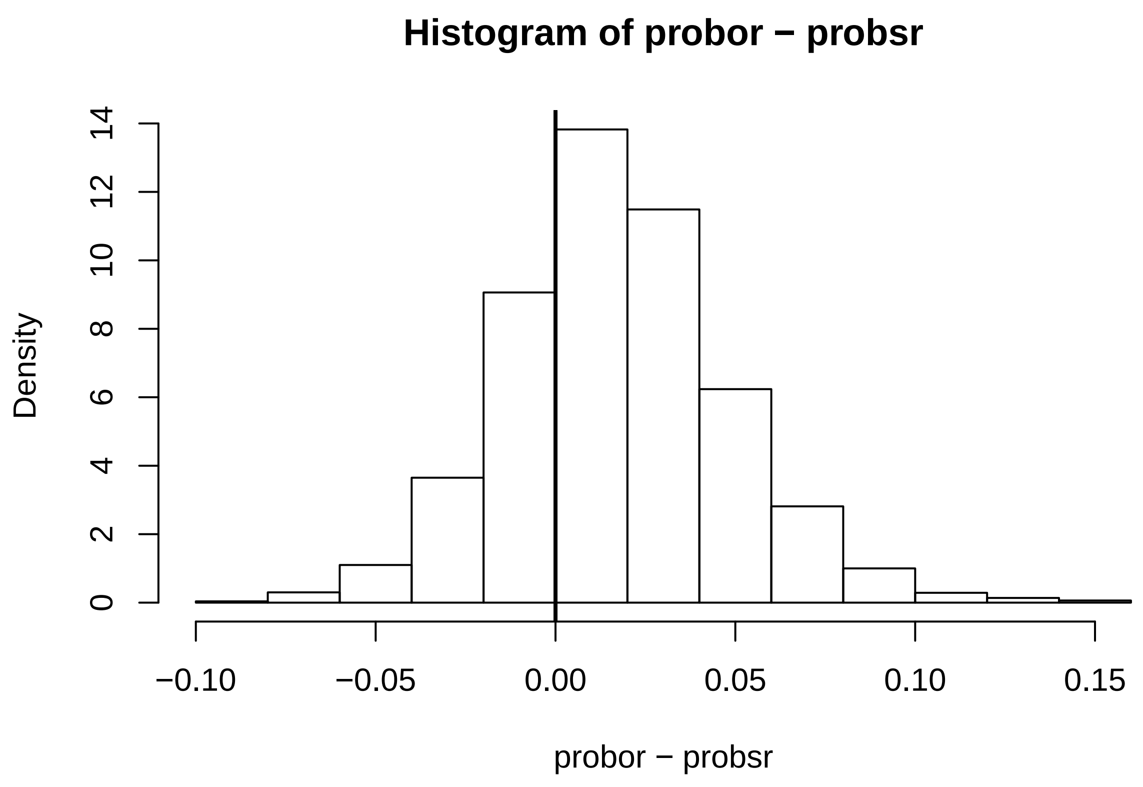


Figure 45: The difference in question-response accuracies between object and subject relatives.

For object gap sentences, the factor *so* is coded as 1, so we have $\mu = \alpha + \beta$. For subject gap sentences, *so* is coded as -1, so we have $\mu = \alpha - \beta$. Therefore, we just need to plug in the expression for μ for object and subject relatives.

We can now straightforwardly plot the posterior distribution of the difference between object and subject relatives. See Figure 45. We see that there isn't any important difference between the two relative clause types.

```
postq<-posterior_samples(m_gg_q2)
alpha<-postq$b_Intercept
beta<-postq$b_so
mu_or<-alpha+beta
probor<-exp(mu_or)/(1+exp(mu_or))
mu_sr<-alpha-beta
probsr<-exp(mu_sr)/(1+exp(mu_sr))

hist(probor-probsr,freq=FALSE)
abline(v=0,lwd=2)
```

4.3 Shrinkage in hierarchical linear models

Recall that the posterior mean is a compromise between the prior mean m and the sample mean $\bar{x}$, weighted by the precision of the prior and the data.

$$m\frac{w_1}{w_1+w_2}+\bar{x}\frac{w_2}{w_1+w_2} \tag{185}$$

Here:

- w_1 is the precision (1/variance) of the prior distribution
- w_2 is the precision of the data

This fact has a practical consequence for us.

The simple varying intercepts model we are considering in the lecture notes (Example 1 in the lecture notes on hierarchical models) is:

$y_{jk} \sim Normal(\beta_0 + \beta_1 \times so_{jk} + u_{0j}, \sigma)$

where j indexes subject id's, k indexes item, and $u_{0j} \sim Normal(0, \sigma_{u0})$. The predictor so is coded as -1/+1, as in the lecture notes.

In the Bayesian framework, each of the u_{0j} is a parameter. When there is too little data to estimate a subject j's parameter, then the posterior is determined by the prior $Normal(0, \sigma_{u0})$ and the parameter u_{0j} shrinks to 0. When there is a lot of data from subject j, then the parameter u_{0j} gets closer and closer to the maximum likelihood estimate for that subject. This phenomenon is called shrinkage or partial pooling. We illustrate this next.

First we load the data from our running example, the Grodner and Gibson experiment 1 data.

```r
library(dplyr)
gg05e1 <- read.table("data/GrodnerGibson2005E1.csv",sep=",", header=TRUE)
gge1 <- gg05e1 %>% filter(item != 0)

gge1 <- gge1 %>% mutate(word_positionnew = ifelse(item != 15 & word_position > 10,
                                                  word_position-1, word_position))
#there is a mistake in the coding of word position,
#all items but 15 have regions 10 and higher coded
#as words 11 and higher

## get data from relative clause verb:
gge1crit <- subset(gge1, ( condition == "objgap" & word_position == 6 ) |
            ( condition == "subjgap" & word_position == 4 ))
gge1crit<-gge1crit[,c(1,2,3,6)]
gge1crit$so<-ifelse(gge1crit$condition=="objgap",1,-1)

dat<- gge1crit
```

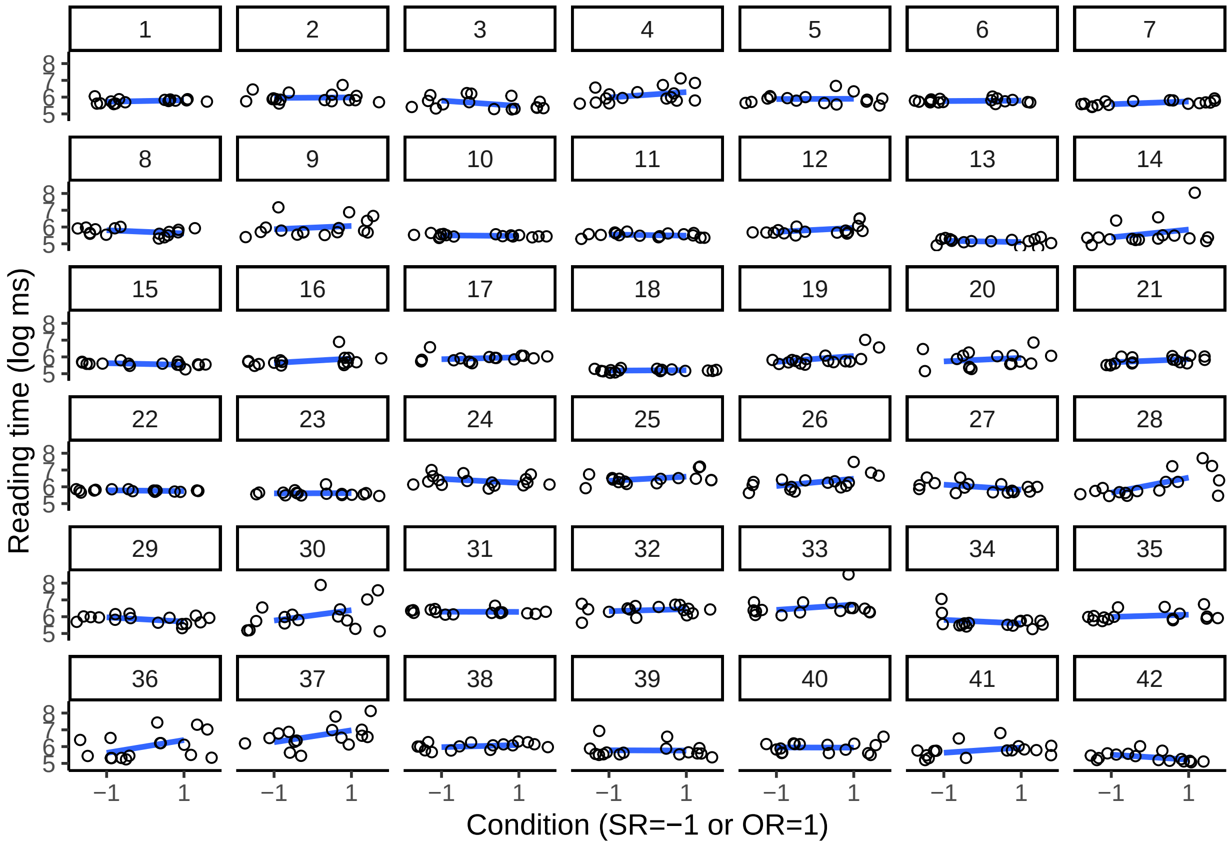


Figure 46: Relative clause effect by subject.

4.3.1 Visualizing the data by subject

The figure below shows that participants don't all show the same difference between object and subject relatives.

```r
library(ggplot2)

xlab <- "Condition (SR=-1 or OR=1)"
ylab <- "Reading time (log ms)"

ggplot(dat) +
  aes(x = so, y = log(rawRT)) +
  stat_smooth(method = "lm", se = FALSE) +
  geom_point(shape=1,position="jitter") +
  facet_wrap("subject") +
  labs(x = xlab, y = ylab) + scale_x_continuous(breaks = c(-1,1))
```

```r
ggplot(dat) +
  aes(x = so, y = log(rawRT)) +
  stat_smooth(method = "lm", se = FALSE) +
```

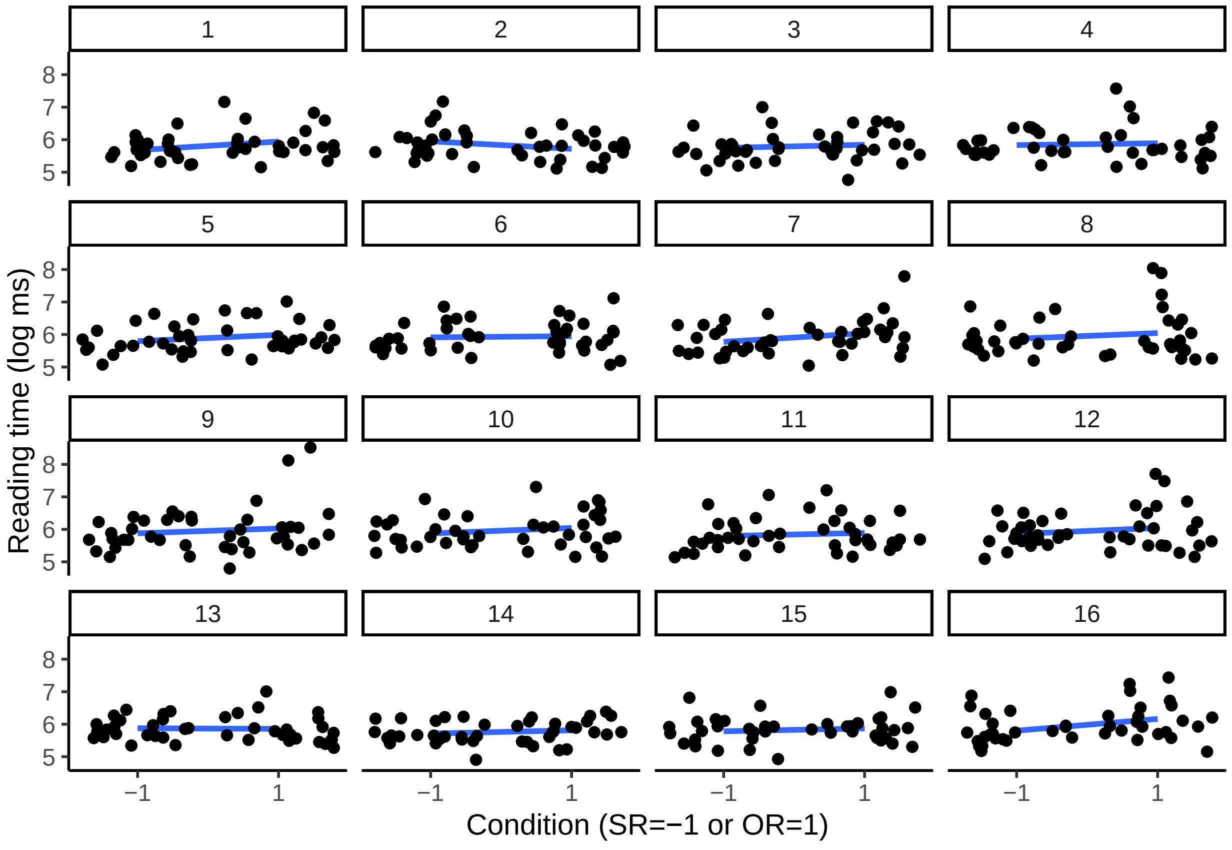


Figure 47: Relative clause effect by item.

```
  geom_point(position="jitter") +
  facet_wrap("item") +
  labs(x = xlab, y = ylab) + scale_x_continuous(breaks = c(-1,1))
```

4.3.2 Complete pooling model

We could fit a single linear model for all subjects (this was a homework assignment):

```
m<-lm(log(rawRT)~so,dat)

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
abline(m,lwd=3,col="red")
```

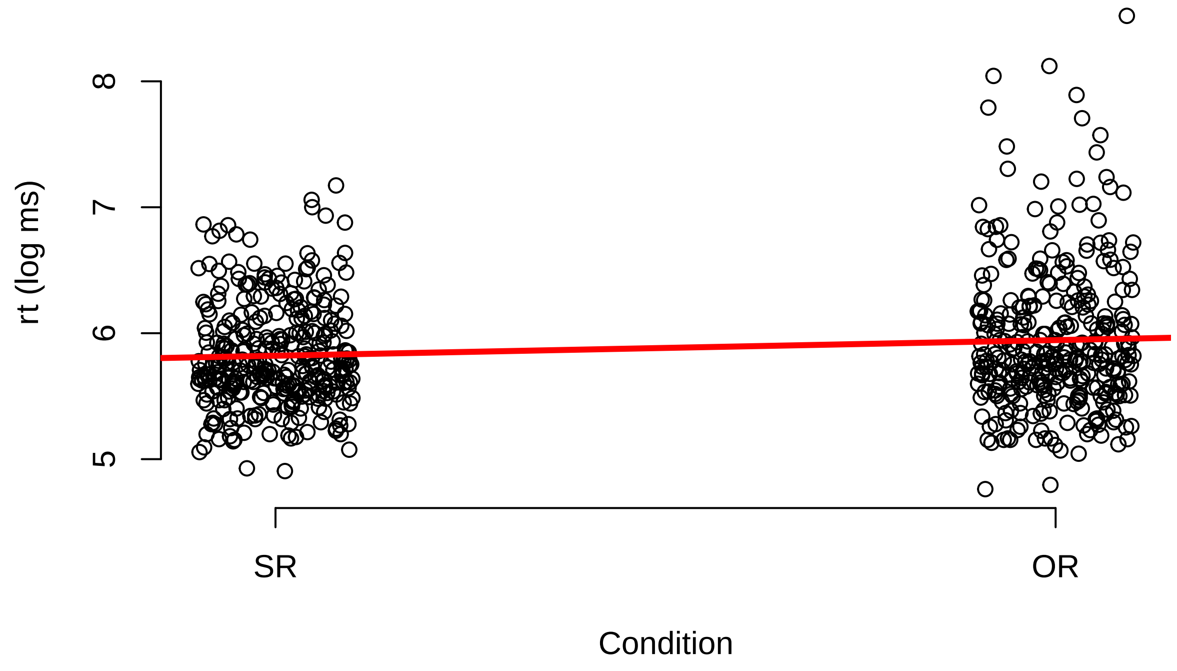


Figure 48: The complete pooling model (the simple linear model).

4.3.3 The no-pooling model

We could fit a **separate** linear model for each subject:

```r
library(lme4)
(nopoolingLM<-lmList(log(rawRT)~so|subject,dat))
```

```
## Call: lmList(formula = log(rawRT) ~ so | subject, data = dat)
## Coefficients:
##    (Intercept)       so
## 1         5.77  0.04352
## 2         5.97  0.01403
## 3         5.63 -0.16101
## 4         6.14  0.16137
## 5         5.89  0.00807
## 6         5.78  0.01197
## 7         5.66  0.08284
## 8         5.71 -0.09474
## 9         5.97  0.09754
## 10        5.49 -0.01712
## 11        5.52 -0.03213
## 12        5.83  0.11808
## 13        5.14 -0.03950
## 14        5.62  0.23244
## 15        5.58 -0.04776
## 16         5.77  0.12939
## 17         5.92  0.05700
## 18         5.20  0.01296
## 19         5.88  0.17398
## 20         5.84  0.10520
## 21         5.77  0.09371
## 22         5.77 -0.02335
## 23         5.62  0.00525
## 24         6.35 -0.11729
## 25         6.48  0.12416
## 26         6.26  0.21653
## 27         5.98 -0.14896
## 28         6.10  0.44814
## 29         5.84 -0.12657
## 30         6.08  0.30988
## 31         6.29 -0.00511
## 32         6.39  0.05924
## 33         6.56  0.15638
## 34         5.71 -0.10665
## 35         6.05  0.06662
## 36         6.01  0.38253
## 37         6.62  0.35537
## 38         6.03  0.06383
## 39         5.77 -0.00769
## 40         5.94 -0.00699
## 41         5.79  0.15446
## 42         5.37 -0.14489
##
## Degrees of freedom: 672 total; 588 residual
## Residual standard error: 0.365
```

Compare the no-pooling estimates of the intercepts and slopes with the complete pooling estimates:

```
## extract intercepts and slopes by subject:
coefs<-coef(nopoolingLM)
intercepts<-coefs[1]
colnames(intercepts)<-"intercept"
slopes<-coefs[2]

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
subjects<-1:42
for(i in subjects){
```

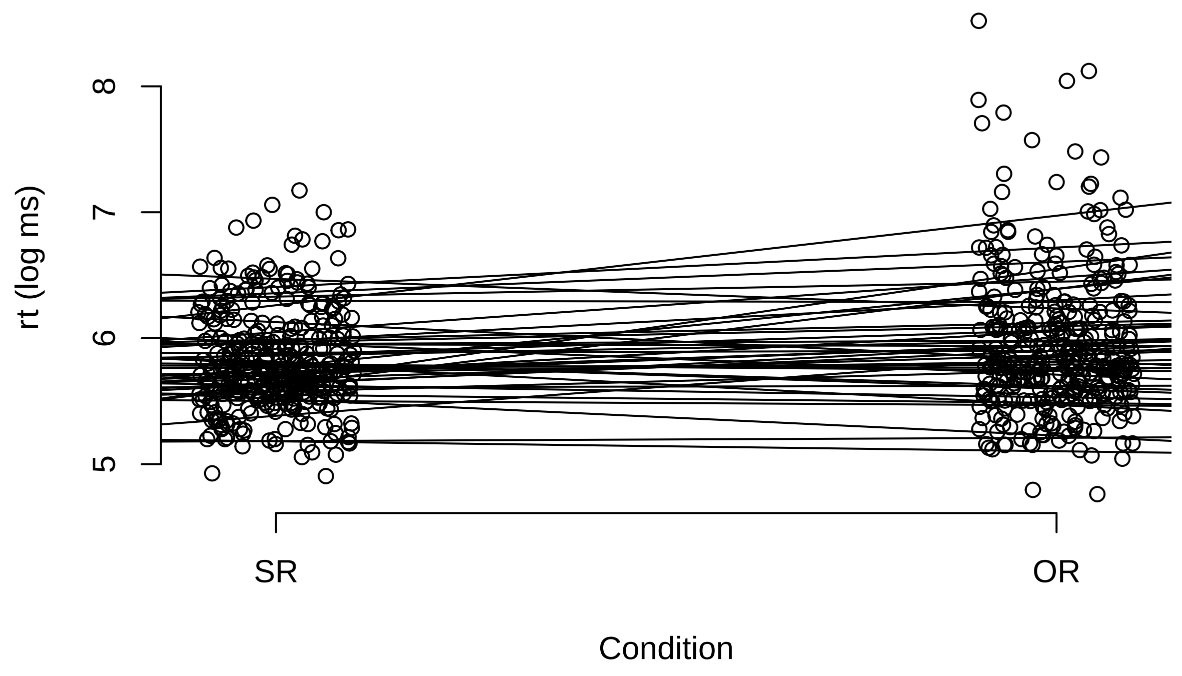


Figure 49: No pooling model's estimates for each subject (also shown is the complete pooling estimate, in red).

```
   abline(intercepts$intercept[i],slopes$so[i])
   }
abline(lm(rawRT~so,dat),lwd=3,col="red")
```

4.3.4 Hierarchical models: partial pooling

4.3.4.1 Varying intercepts by subject

First, consider the varying intercepts model. We will now visualize this model and compare it graphically to the no pooling and complete pooling models.

```
m1<-lmer(log(rawRT)~so+(1|subject),dat)
## estimates of intercept and slope
b0<-fixef(m1)[1]
b1<-fixef(m1)[2]

## intercept adjustments by subject
u0<-ranef(m1)
plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
```

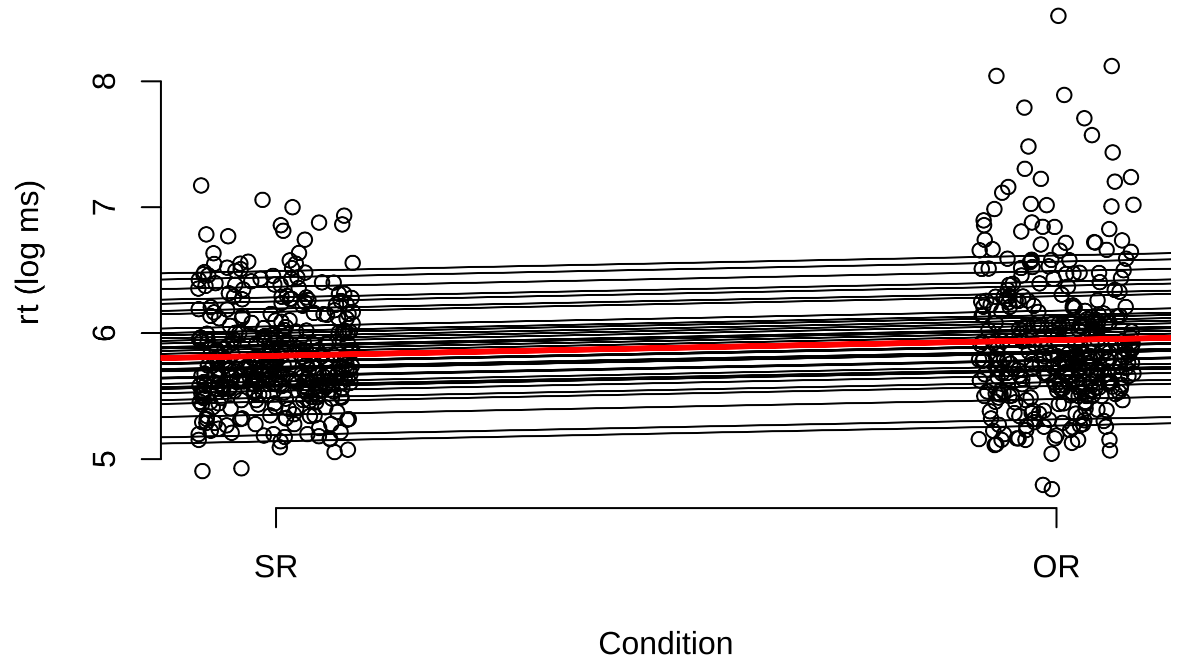


Figure 50: Partial pooling by subject (also shown is the complete pooling estimate, in red).

```
subjects<-1:42
for(i in subjects){
   abline(b0+u0$subject[i,],b1)
   }
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")
```

4.3.4.2 Varying intercepts and slopes by subject (the maximal model)

```
m2<-lmer(log(rawRT)~so+(1+so|subject),dat)
## estimates of fixed effects intercept and slope
b0<-fixef(m2)[1]
b1<-fixef(m2)[2]

## intercept adjustments by subject
u0<-ranef(m2)$subject[,1]
## slope adjustments by subject
u1<-ranef(m2)$subject[,2]

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
for(i in subjects){
```

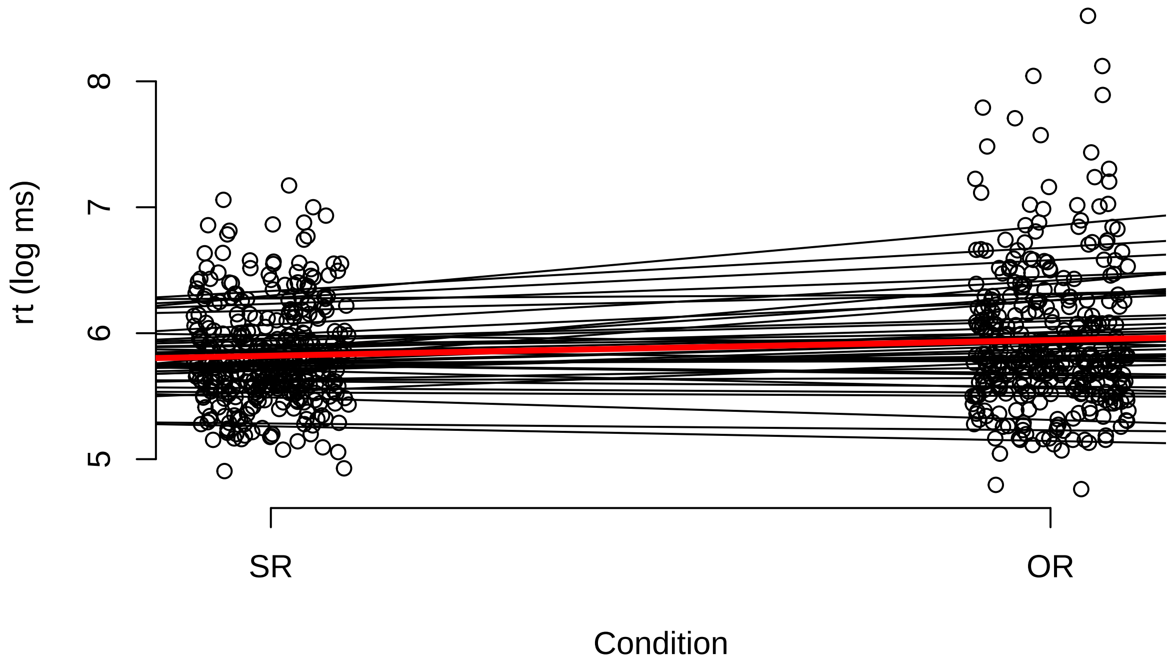


Figure 51: The partial pooling model with adjustments to intercepts and slopes by subject.

```
    abline(b0+u0[i],b1+u1[i])
  }
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")
```

4.3.5 The practical implications of shrinkage or partial pooling

4.3.5.1 Regularization 1: Extreme behavior of individual subjects is shrunk towards the grand mean

Compare both the no pooling and partial pooling estimates for subjects 28, 36, 37. These subjects show the most extreme effects of relative clause type:

```
coef(nopoolingLM)[2][28,]
```

```
## [1] 0.448
```

```
coef(nopoolingLM)[2][36,]
```

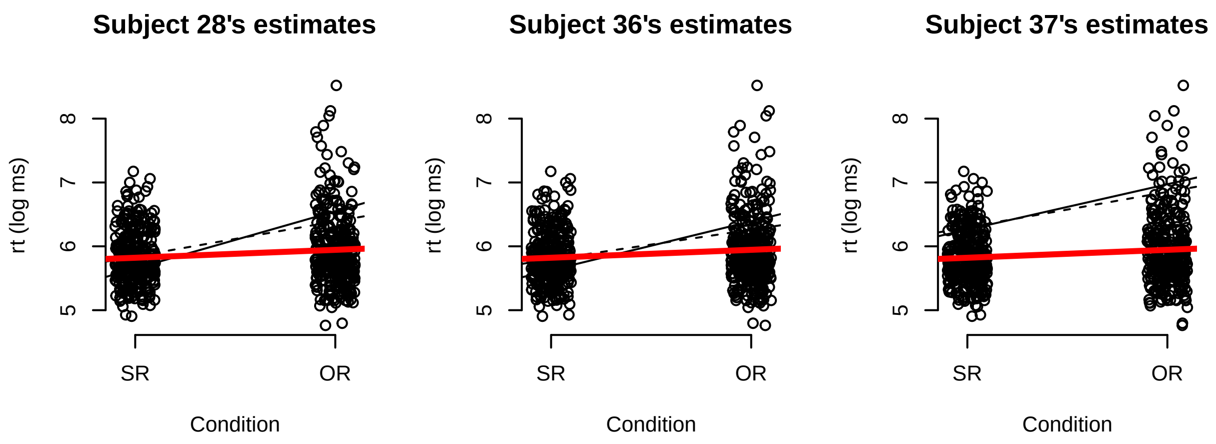
```
## [1] 0.383
```

```
coef(nopoolingLM)[2][37,]
```

```
## [1] 0.355
```

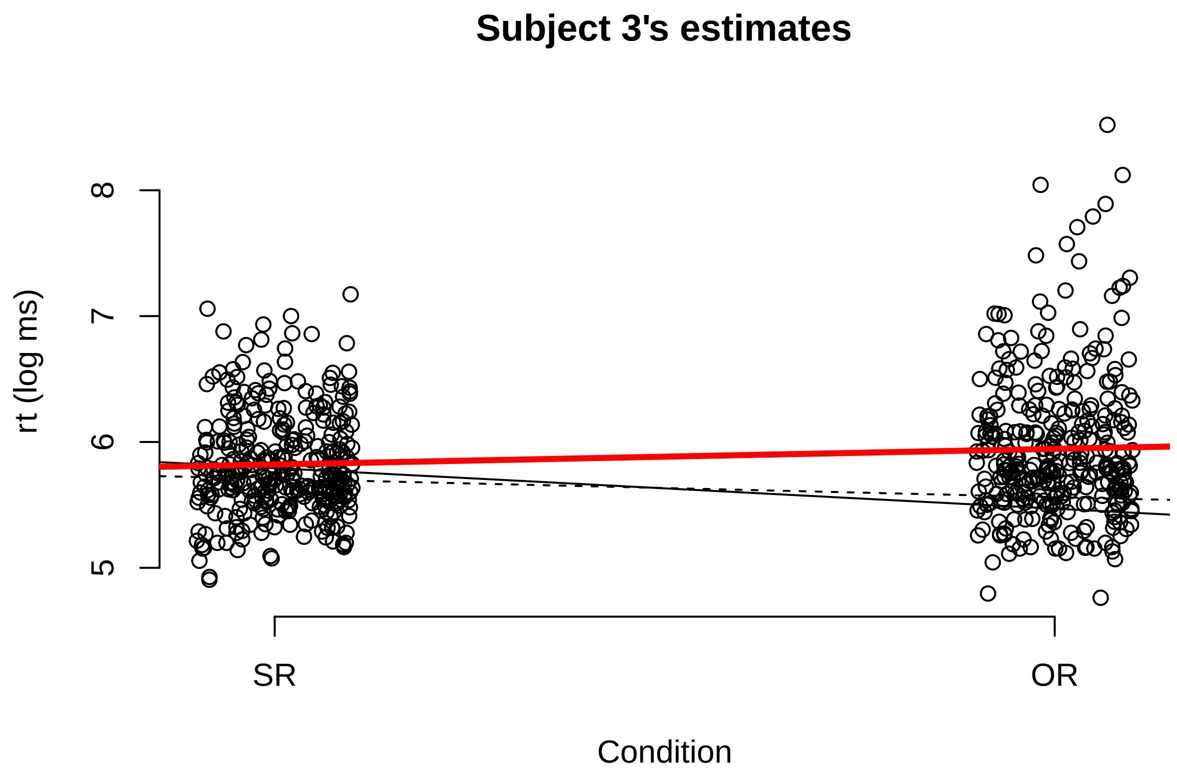
```
op<-par(mfrow=c(1,3),pty="s")
coefs<-coef(nopoolingLM)
```

```r
intercepts<-coefs[1]
colnames(intercepts)<-"intercept"
slopes<-coefs[2]

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)",
        main="Subject 28's estimates")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
## no pooling estimate:
abline(intercepts$intercept[28],slopes$so[28])
## partial pooling:
abline(b0+u0[28],b1+u1[28],lty=2)
## complete pooling:
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)",
        main="Subject 36's estimates")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
## no pooling estimate:
abline(intercepts$intercept[36],slopes$so[36])
## partial pooling:
abline(b0+u0[36],b1+u1[36],lty=2)
## complete pooling:
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")

plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)",
        main="Subject 37's estimates")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
## no pooling estimate:
abline(intercepts$intercept[37],slopes$so[37])
## partial pooling:
abline(b0+u0[37],b1+u1[37],lty=2)
## complete pooling:
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")
```



In the no-pooling analysis, subject 3 shows the biggest negative slope, which goes against the hypothesis that object relatives are harder to process than subject relatives. Compare the no-pooling estimate for this subject with the partial pooling or shrunk estimates from the hierarchical model:

```
plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)",
        main="Subject 3's estimates")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
## no pooling estimate:
abline(intercepts$intercept[3],slopes$so[3])
## partial pooling:
abline(b0+u0[3],b1+u1[3],lty=2)
## complete pooling:
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")
```



4.3.5.2 Regularization 2: When data from a subject are sparse, the estimates for a subject shrinks to the grand mean

First save the original estimates of the intercept and slope for subject 37:

```
intercept37orig<-intercepts$intercept[37]
slope37orig<-slopes$so[37]
```

Then delete about half the data from this subject:

```
set.seed(4321)
## choose some data randomly to remove:
rand<-rbinom(1,n=16,prob=0.5)

dat[which(dat$subject==37),]$rawRT
```

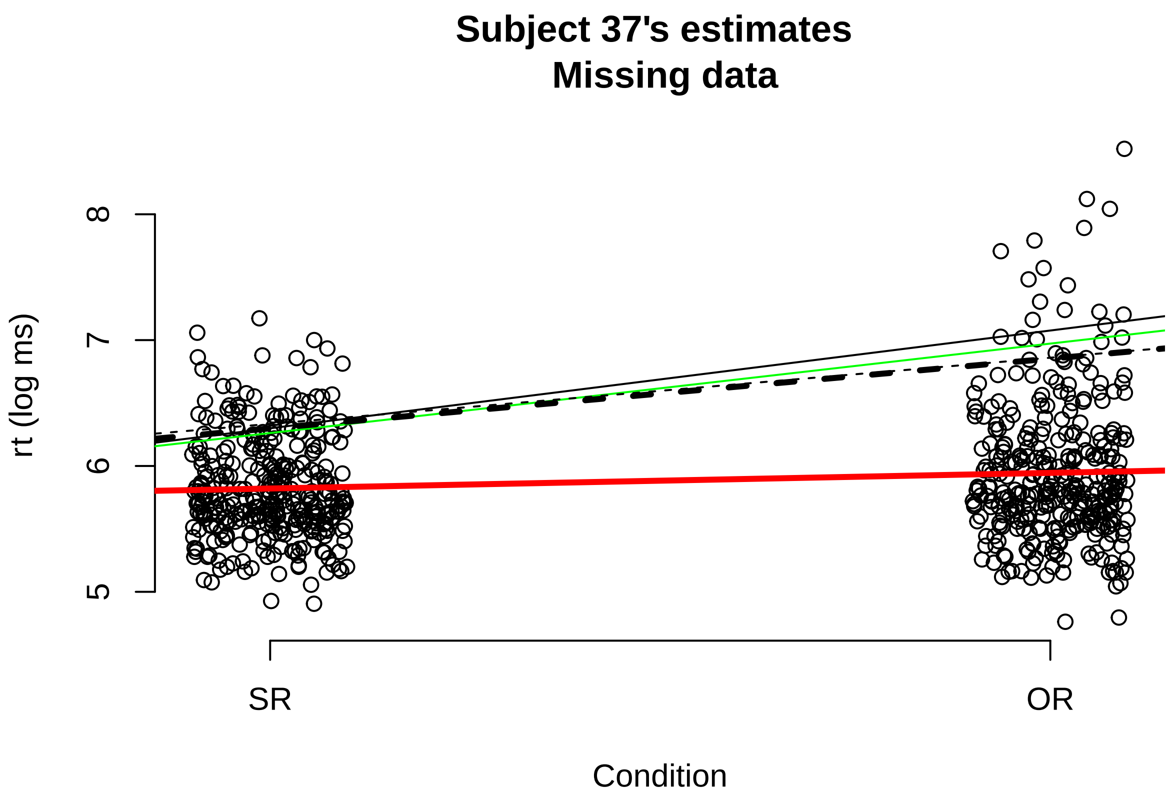
```
##  [1]  770  536  686  578  457  487 2419  884 3365  233  715  671 1104  281
## [15] 1081  971
```

```
dat$deletedRT<-dat$rawRT
dat[which(dat$subject==37),]$deletedRT<-ifelse(rand,NA,dat[which(dat$subject==37),]$rawR
```

Now fit the hierarchical model and extract the estimates by subject:

```
b0orig<-fixef(m2)[1]
b1orig<-fixef(m2)[2]
```

```r
## intercept adjustments by subject
u0orig<-ranef(m2)$subject[,1]
## slope adjustments by subject
u1orig<-ranef(m2)$subject[,2]


m3<-lmer(log(deletedRT)~so+(1+so|subject),dat)

b0<-fixef(m3)[1]
b1<-fixef(m3)[2]
## intercept adjustments by subject
u0<-ranef(m3)$subject[,1]
## slope adjustments by subject
u1<-ranef(m3)$subject[,2]

## No pooling estimates for deleted data
nopoolingLMdeleted<-lmList(log(deletedRT)~so|subject,dat)
coefs<-coef(nopoolingLMdeleted)
intercepts<-coefs[1]
colnames(intercepts)<-"intercept"
slopes<-coefs[2]


plot(jitter(dat$so,.5),
        log(dat$rawRT),axes=FALSE,
        xlab="Condition",ylab="rt (log ms)",
        main="Subject 37's estimates \n Missing data")
axis(1,at=c(-1,1),labels=c("SR","OR"))
axis(2)
## no pooling estimate from deleted data:
abline(intercepts$intercept[37],slopes$so[37])
abline(intercept37orig,slope37orig,col="green")
## partial pooling deleted data:
abline(b0+u0[37],b1+u1[37],lty=2)
## partial pooling original data:
abline(b0orig+u0orig[37],b1orig+u1orig[37],lty=2,lwd=3)
## complete pooling:
abline(lm(log(rawRT)~so,dat),lwd=3,col="red")
```



What we see here is that the estimates from the hierarchical model are barely affected by the missingness, but the estimates from the no-pooling model are heavily affected.

4.3.6 Summary

- When an individual subject's data has sparse measurements, the hierarchical model "shrinks" the subject's estimates to the grand mean (complete pooling model) estimates. **This is because the subject's adjustments to intercept and slope shrink towards 0, which is the prior mean.**
- When the subject provides sufficient data, the estimates for the adjustment to the intercept and slope for that subject are closer to the individual subject's means (the sample mean for that subject).

References

Barr, Dale J, Roger Levy, Christoph Scheepers, and Harry J Tily. 2013. "Random Effects Structure for Confirmatory Hypothesis Testing: Keep It Maximal." *Journal of Memory and Language* 68 (3). Elsevier: 255–78.

Bates, D., M. Maechler, B.M. Bolker, and S. Walker. 2015. "Fitting Linear Mixed-Effects Models Using Lme4." *Journal of Statistical Software* 67 (1): 1–48.

Blastland, Michael, and David Spiegelhalter. 2014. *The Norm Chronicles: Stories and Numbers About Danger and Death*. Basic Books (AZ).

Blitzstein, Joseph K, and Jessica Hwang. 2014. *Introduction to Probability*. Chapman; Hall/CRC.

Bürkner, Paul-Christian. 2017. "brms: An R Package for Bayesian Multilevel Models Using Stan." *Journal of Statistical Software* 80 (1): 1–28. https://doi.org/10.18637/jss.v080.i01.

Engelmann, Felix, Lena A. Jäger, and Shravan Vasishth. 2018. "The Effect of Prominence and Cue Association in Retrieval Processes: A Computational Account."

Gelman, Andrew, John B Carlin, Hal S Stern, and Donald B Rubin. 2014. *Bayesian Data Analysis*. Third Edition. Taylor & Francis.

Gibson, Edward. 2000. "Dependency Locality Theory: A Distance-Based Theory of Linguistic Complexity." In *Image, Language, Brain: Papers from the First Mind Articulation Project Symposium*, edited by Alec Marantz, Yasushi Miyashita, and Wayne O'Neil. Cambridge, MA: MIT Press.

Grodner, Daniel, and Edward Gibson. 2005. "Consequences of the Serial Nature of Linguistic Input." *Cognitive Science* 29: 261–90.

Hoekstra, Rink, Richard D Morey, Jeffrey N. Rouder, and Eric-Jan Wagenmakers. 2014. "Robust Misinterpretation of Confidence Intervals." *Psychonomic Bulletin & Review* 21 (5). Springer: 1157–64. https://doi.org/10.3758/s13423-013-0572-3.

Jäger, Lena A., Felix Engelmann, and Shravan Vasishth. 2017. "Similarity-Based Interference in Sentence Comprehension: Literature Review and Bayesian Meta-Analysis." *Journal of Memory and Language* 94: 316–39.

Kerns, G. Jay. 2018. *Introduction to Probability and Statistics Using R*. https://www.nongnu.org/ipsur/.

Lambert, Ben. 2018. *A Student's Guide to Bayesian Statistics*. Sage.

Lewis, Richard L., and Shravan Vasishth. 2005. "An Activation-Based Model of Sentence Processing as Skilled Memory Retrieval." *Cognitive Science* 29: 1–45.

Lynch, Scott M. 2007. *Introduction to Applied Bayesian Statistics and Estimation for Social Scientists*. Springer Science & Business Media.

Morey, Richard D, Rink Hoekstra, Jeffrey N. Rouder, Michael D Lee, and Eric-Jan Wagenmakers. 2016. "The Fallacy of Placing Confidence in Confidence Intervals" 23 (1): 103–23. https://doi.org/

10.3758/s13423-015-0947-8.

Morin, David J. 2016. *Probability for the Enthusiastic Beginner*. Createspace Independent Publishing Platform.

Neal, Radford M, and others. 2011. "MCMC using Hamiltonian dynamics." *Handbook of Markov Chain Monte Carlo* 2 (11): 2.

Nicenboim, Bruno, Timo B. Roettger, and Shravan Vasishth. 2018. "Using Meta-Analysis for Evidence Synthesis: The case of incomplete neutralization in German." *Journal of Phonetics* 70: 39–55. https://doi.org/https://doi.org/10.1016/j.wocn.2018.06.001.

Ross, Sheldon. 2002. *A First Course in Probability*. Pearson Education.

Schad, Daniel J., Sven Hohenstein, Shravan Vasishth, and Reinhold Kliegl. 2019. "How to Capitalize on a Priori Contrasts in Linear (Mixed) Models: A Tutorial." *Journal of Memory and Language*.

Shiffrin, Richard M., Michael Lee, Woojae Kim, and Eric-Jan Wagenmakers. 2008. "A Survey of Model Evaluation Approaches with a Tutorial on Hierarchical Bayesian Methods." *Cognitive Science* 32 (8). Wiley-Blackwell: 1248–84. https://doi.org/10.1080/03640210802414826.

Stan Development Team. 2017. "Stan: A C++ Library for Probability and Sampling, Version 2.16.0." http://mc-stan.org/.

Vasishth, Shravan, Zhong Chen, Qiang Li, and Gueilan Guo. 2013. "Processing Chinese Relative Clauses: Evidence for the Subject-Relative Advantage." *PLoS ONE* 8 (10). Public Library of Science: 1–14.

Vasishth, Shravan, Daniela Mertzen, Lena A. Jäger, and Andrew Gelman. 2018. "The Statistical Significance Filter Leads to Overoptimistic Expectations of Replicability." *Journal of Memory and Language* 103: 151–75. https://doi.org/https://doi.org/10.1016/j.jml.2018.07.004.