Bayesian Inference and Computation

Dr Rowland Seymour

Semester 2, 2024

Contents

Practicalities

0.1 Module Aims

Bayesian inference is a set of methods where the probability of an event occurring can be updated as more information becomes available. It is fundamentally different from frequentist methods, which are based on long running relative frequencies. This module gives an introduction to the Bayesian approach to statistical analysis and the theory that underpins it.

Students will be able to explain the distinctive features of Bayesian methodology, understand and appreciate the role of prior distributions and compute posterior distributions. It will cover the derivation of posterior distributions, the construction of prior distributions, and inference for missing data. Extensions are considered to models with more than a single parameter and how these can be used to analyse data. Computational methods have greatly advanced the use of Bayesian methods and this module covers, and allows students to apply, procedures for the sampling and analysis of intractable Bayesian problems.

By the end of the course, students should be able to:

1. Demonstrate a full and rigorous understanding of all definitions associated with Bayesian inference and understand the differences between the Bayesian and frequentist approaches to inference
2. Demonstrate a sound understanding of the fundamental concepts of Bayesian inference and computational sampling methods
3. Understand how to make inferences assuming various population distributions while taking into account expert opinion and the implications of weak prior knowledge and large samples
4. Demonstrate an understanding of the principles of Markov Chain Monte Carlo and be able to programme an MCMC algorithm
5. Engage in Bayesian data analysis in diverse situations drawn from physics, biological, engineering and other mathematical contexts.

0.2 Module Structure

The module is split between theory and computation. Each week will have three lectures, one computer lab and one guided study. In the labs, you will need to bring your own laptop. The timetable for this module is

Week	Monday (1200)	Tuesday (1000)	Thursday (1100)	Friday (1200)
2	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G19
3	Physics West 115	Nuffield G18	Arts LR5	Nuffield G19
4	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G13
5	Physics West 115	Nuffield G18	Arts LR5	Nuffield G19
6	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G13
7	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G13
8	Physics West 115	Nuffield G18	Arts LR5	Nuffield G19
9	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G19
10	Physics West 115	Nuffield G18	Nuffield G13	Nuffield G13

0.3 Assessment

Assessment for this module is 50% via an exam and 50% via coursework assignments during the semester. The exam will last 1h 30m and take place during the summer exam period. There will be three coursework assignment – assignment 1 will be worth 10% of the final mark, with assignments 2 and 3 counting for 20% each. More details about the assignments will be made available during the semester.

0.4 Getting Help

There are lots of ways of getting help throughout the module. You can visit my office hour (Watson 317) on Thursdays at 0900-1030 or email me at r.g.seymou r@bham.ac.uk.

0.5 Recommended Books and Videos

No books are required for this course and the whole material is contained in these notes. However, you may find it useful to use other resources in your studies. I recommend the following:

1. A First Course in Bayesian Statistical Methods - Peter D. Hoff. This is a short book that covers the basics of Bayesian inference and computation. To the point and well written, it's a useful place to look topics up.
2. Bayesian Data Analysis - Andrew Gelman, John Carlin, Hal Stern, David Dunson, Aki Vehtari, and Donald Rubin. This is a thorough book explaining everything you'd need to know to carry out Bayesian data analysis. It's a fairly long and in-depth book, but the authors are authoritative and give good advice throughout. Example code on the website is in R, Python and Stan.
3. Statistical Rethinking - Richard McElrath. This book provides a friendly intuitive understanding of Bayesian inference and computation. Aimed at social and natural scientists, it has less theory that the other two books but is perhaps more approachable. A set of video lectures for this book can be found on YouTube.

0.6 Common Distributions

For many Bayesian inference problems, it is useful to be able to identify probability density functions (for continuous random variables) and probability mass functions (for discrete random variables) up to proportionality. Some common density/mass functions are given below.

Normal distribution

$$\pi(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \qquad x \in \mathbb{R},$$

where $\mu \in \mathbb{R}$ and $\sigma > 0$.

Beta distribution

$$\pi(x \mid \alpha, \beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \qquad x \in [0,1],$$

where $\alpha, \beta > 0$ and $B(\alpha, \beta)$ is the beta function.

Gamma distribution

$$\pi(x \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \qquad x > 0,$$

where $\alpha, \beta > 0$ and $\Gamma(\alpha)$ is the gamma function.

Exponential distribution

$$f(x \mid \lambda) = \lambda e^{-\lambda x} \qquad x > 0,$$

where $\lambda > 0$.

Poisson distribution

$$\pi(x = k \mid \lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad k \in \{1, 2, ...\},$$

where $\lambda > 0$.

Binomial distribution

$$\pi(x = k \mid N, p) = \binom{N}{k} p^k (1-p)^{N-k} \qquad k \in \{1, ..., N\}$$

where $p \in [0, 1]$.

Chapter 1

Fundamentals of Bayesian Inference

Bayesian inference is built on a different way of thinking about parameters of probability distributions than methods you have learnt so far. In the past 30 years or so, Bayesian inference has become much more popular. This is partly due to increased computational power becoming available. In this first chapter, we are going to set out to answer:

1. What are the fundamental principles of Bayesian inference?
2. What makes Bayesian inference different from other methods?

1.1 Statistical Inference

The purpose of statistical inference is to "draw conclusions, from numerical data, about quantities that are not observed" (Bayesian Data Analysis, chapter 1). Generally speaking, there are two kinds of inference:

1. Inference for quantities that are unobserved or haven't happened yet. Examples of this might be the size of a payout an insurance company has to make, or a patients outcome in a clinical trial had they been received a certain treatment.
2. Inference for quantities that are not possible to observe. This is usual because they are part of modelling process, like parameters in a linear model.

In this module, we are going to look at a different way of carrying out statistical inference, one that doesn't depend on long run events. Instead, we're going to introduce the definition of probability that allows us to interpret the subjective chance that an event occurs.

1.2 Frequentist Theory

Frequentist probability is built upon the theory on long run events. Probabilities must be interpretable as frequencies over multiple repetitions of the experiment that is being analysed, and are calculated from the sampling distributions of measured quantities.

Definition 1.1. The long run relative frequency of an event is the **probability** of that event.

Example 1.1. If a frequentist wanted to assign a probability to rolling a 6 on a particular dice, then they would roll the dice a large number of times and compute the relative frequency.

Definition 1.2. The **sampling distribution** of a statistic is the distribution based on a long run of samples of a fixed size from the population.

The sampling distribution is an important concept in frequentist theory as it describes the randomness in the process. From a frequentist standpoint, we have a model containing some parameter θ and some data y. All the evidence in the data y about θ is contained in the likelihood function $\pi(y \mid \theta)$. The parameter θ is fixed and the likelihood function describes the probability of observing the data y given the parameter θ.

The most common way to estimate the value of θ is using maximum likelihood estimation. Although other methods do exist (e.g. method of moments, or generalised maximum likelihood estimation).

Definition 1.3. The maximum likelihood estimate of θ, $\hat{\theta}$, is the value such that $\hat{\theta} = \max_\theta \pi(y \mid \theta)$.

Uncertainty around the maximum likelihood estimate is based on the theory of long running events that underpin frequentist theory.

Definition 1.4. Let Y be a random sample from a probability distribution θ. A $100(1-\alpha)$ **confidence interval** for θ is an interval $(u(Y), v(Y))$ such that

$$\mathbb{P}(u(Y) < \theta < v(Y)) = 1 - \alpha$$

This means that if you had an infinite number of samples for Y and the corresponding infinite number of confidence intervals, then $100(1-\alpha)\%$ of them would contain the true value of θ. It does *not* mean that there is a $100(1-\alpha)$ probability a particular interval contains the true value of θ.

Given that we want to understand the properties of θ given the data we have observed y, then you might think it makes sense to investigate the distribution $\pi(\theta \mid y)$. This distribution says what are the likely values of θ given the information we have observed from the data y. We will talk about Bayes' theorem in more detail later on in this chapter, but, for now, we will use it to write down this distribution

$$\pi(\theta \mid y) = \frac{\pi(y \mid \theta)\pi(\theta)}{\pi(y)}.$$

This is where frequentist theory cannot help us, particularly the term $\pi(\theta)$. Randomness can only come from the data, so how can we assign a probability distribution to a constant θ? The term $\pi(\theta)$ is meaningless under this philosophy. Instead, we turn to a different philosophy where we can assign a probability distribution to θ.

1.3 Bayesian Probability

The Bayesian paradigm is built around a different definition of probability. This allows us to generate probability distirbutuions for parameters values.

Definition 1.5. The subjective belief of an event is the **probability** of that event.

This definition means we can assign probabilities to events that frequentists do not recognise as valid.

Example 1.2. Consider the following:

1. The probability that I vote for the labour party at the next election
2. A photo taken from the James Watt telescope contains a new planet.
3. The real identify of Banksy is Robin Gunningham.

These are not events that can be repeated in the long run.

1.4 Conditional Probability and Exchangability

Before we derive Bayes' theorem, we recap some important definitions in probability.

Definition 1.6. Given two events A and B, the **conditional probability** that event A occurs given the event B has already occurred is

$$\pi(A \mid B) = \frac{\pi(A \cap B)}{\pi(B)},$$

when $\pi(B) > 0$.

Definition 1.7. Two events A and B are **independent** given event C if and only if

$$\pi(A \cap B \mid C) = \pi(A \mid C)\pi(B \mid C).$$

Definition 1.8. Let $\pi(y_1, \dots, y_N)$ be the joint density of $Y_1, \dots, Y_N$. If $\pi(y_1, \dots, y_N) = \pi(y_{\pi_1}, \dots, y_{\pi_N})$ for a permutations π of $\{1, \dots, N\}$, then $Y_1, \dots, Y_N$ are **exchangeable**.

Exchangability means that the labels of the random variables don't contain any information about the outcomes. This is an important idea in many areas of probability and statistics, and we often model exchangeable events as iid.

Example 1.3. If $Y_i \sim Bin(n, p)$ are independent and identically distributed for $i = 1, 2, 3$, then $\pi(Y_1, Y_2, Y_3) = \pi(Y_3, Y_1, Y_2)$.

Example 1.4. Let (X, Y) follow a bivariate normal distribution with mean $\mathbf{0}$, variances $\sigma_x = \sigma_y = 1$ and a correlation parameter $\rho \in [-1, 1]$. (X, Y) are exchangable, but only independent if $\rho = 0$.

Proposition 1.1. *If $\theta \sim \pi(\theta)$ and $(Y_1, \dots, Y_N)$ from a sample space $\mathcal{Y}$ are conditionally iid given some parameter θ, then marginally $Y_1, \dots, Y_N$ are exchangable.*

Proof. Suppose $(Y_1, \dots, Y_N)$ are conditionally iid given some parameter θ. Then for any permutation σ of $\{1, \dots, N\}$ and observations $\{y_1, \dots, y_N\}$

$$\begin{aligned}\pi(y_1, \dots, y_N) &= \int \pi(y_1, \dots, y_N \mid \theta)\pi(\theta)\, d\theta \qquad \text{(definition of marginal distribution)}\\ &= \int \left\{\prod_{i=1}^{N} \pi(y_i \mid \theta)\right\} \pi(\theta)\, d\theta \qquad \text{(definition of conditionally iid)}\\ &= \int \left\{\prod_{i=1}^{N} \pi(y_{\sigma_i} \mid \theta)\right\} \pi(\theta)\, d\theta \qquad \text{(product is commutative)}\\ &= \pi(y_{\pi_1}, \dots, y_{\sigma_N}) \qquad \text{(definition of marginal distribution)}\end{aligned} \tag{1.1}$$

□

This tells us that if we have some conditionally iid random variables and a subjective prior belief about some parameter θ, then we have exchangeability. This is nice to have, but the implication in the other direction is much more interesting and powerful.

Theorem 1.1 (de Finetti)**.** *If a sequence of random variables $(Y_1, \dots, Y_N)$ from a sample space $\mathcal{Y}$ is exchangeable, then its joint distribution can be written as*

$$\pi(y_1, \dots, y_N) = \int \left\{\prod_{i=1}^{N} \pi(y_i \mid \theta)\right\} \pi(\theta)\, d\theta$$

for some parameter θ, some distribution on θ, and some sampling model $\pi(y_i \mid \theta)$.

This is a kind of existence theorem for Bayesian inference. It says that if we have exchangeable random varibales, then a parameter θ must exist and a subjective probability distribution $\pi(\theta)$ must also exist. The argument against Bayesian inference is that it doesn't guarantee a *good* subjective probability distribution $\pi(\theta)$ exists.

1.5 Bayes' Theorem

Now we have an understanding of conditional probability and exchangeability, we can put these two together to understand Bayes' Theorem. Bayes' theorem is concerned with the distribution of the parameter θ given some observed data y. It tries to answer the question: what does the data tell us about the model parameters?

Theorem 1.2 (Bayes)**.** *The distribution of the model parameter θ given the data y is*

$$\pi(\theta \mid y) = \frac{\pi(y \mid \theta)\pi(\theta)}{\pi(y)}$$

Proof.

$$\pi(\theta \mid y) = \frac{\pi(\theta, y)}{\pi(y)} \tag{1.2}$$

$$\implies \pi(\theta, y) = \pi(\theta \mid y)\pi(y) \tag{1.3}$$

Analogously, using $\pi(y \mid \theta)$ we can derive

$$\pi(\theta, y) = \pi(y \mid \theta)\pi(\theta)$$

Putting these two terms equal to each other and dividing by $\pi(y)$ gives

$$\pi(\theta \mid y) = \frac{\pi(y \mid \theta)\pi(\theta)}{\pi(y)}$$

□

There are four terms in Bayes' theorem:

1. The **posterior distribution** $\pi(\theta \mid y)$. This tells us our belief about the model parameter θ given the data we have observed y.
2. The **likelihood function** $\pi(y \mid \theta)$. The likelihood function is common to both frequentist and Bayesian methods. By the likelihood principle, the likelihood function contains all the information the data can tell us about the model parameter θ.
3. The **prior distribution** $\pi(\theta)$. This is the distribution that describes our prior beliefs about the value of θ. The form of θ should be decided before we see the data. It may be a vague distribution (e.g. $\theta \sim N(0, 10^2)$) or a specific distribution based on prior information from experts (e.g. $\theta \sim N(5.5, 1.3^2)$).

4. The **evidence of the data** $\pi(y)$. This is sometimes called the average probability of the data or the marginal likelihood. In practice, we do not need to derive this term as it can be back computed to ensure the posterior distribution sums/integrates to one.

A consequence of point four is that posterior distributions are usually derived proportionally, and (up to proportionality) Bayes' theorem

$$\pi(\theta \mid y) \propto \pi(y \mid \theta)\pi(\theta).$$

> **Some history of Thomas Bayes**. Thomas Bayes was an English theologean born in 1702. His "Essay towards solving a problem in the doctrine of chances" was published posthumously. It introduces theroems on conditional probability and the idea of prior probability. He discusses an experiment where the data can be modelled using the Binomial distribution and he guesses (places a prior distribution) on the probability of success.
>
> Richard Price sent Bayes' work to the Royal Society two years after Bayes had died. In his commentary on Bayes' work, he suggested that the Bayesian way of thinking proves the existance of God, stating: The purpose I mean is, to show what reason we have for believing that there are in the constitution of things fixt laws according to which things happen, and that, therefore, the frame of the world must be the effect of the wisdom and power of an intelligent cause; and thus to confirm the argument taken from final causes for the existence of the Deity.
>
> It's not clear how Bayesian Thomas Bayes actually was, as his work was mainly about specific forms of probability theory and not his intepretation of it. The Bayesian way of thinking was really popularised by Laplace, who wrote about deductive probability in the early 19th century.

Example 1.5. We finish this chapter with a very simple example. The advantage of the example being so simple is that we can obtain plots in R that show what's going on.

Suppose we have a model $y \sim N(\theta, 1)$ and we want to estimate θ. To do this we need to derive the posterior distribution. By Bayes' theorem,

$$\pi(\theta \mid y) \propto \pi(y \mid \theta)\pi(\theta).$$

We know the form of $\pi(y \mid \theta) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(y-\theta)^2}$, but how should we describe our prior beliefs about θ? Here are three options:

1. We can be very vague about θ – we genuinely don't know about its value. We assign a uniform prior distribution to θ that takes values between -1,000 and +1,000, i.e. $\theta \sim u[-1000, 1000]$. Up to proportionality $\pi(\theta) \propto 1$ for $\theta \in [-1000, 1000]$.
2. After thinking hard about the problem, or talking to an expert, we decide that the only thing we know about θ is that it can't be negative. We adjust our prior distribution from 1. to be $\theta \sim u[0, 1000]$. Up to proportionality $\pi(\theta) \propto 1$ for $\theta \in [0, 1000]$.

3. We decide to talk to a series of experts about θ asking for their views on likely values of θ. Averaging the experts opinions gives $\theta \sim N(3, 0.7^2)$. This is a method known as prior elicitation.

We now go and observe some data. After a lot of time and effort, we collect one data point: $y = 0$.

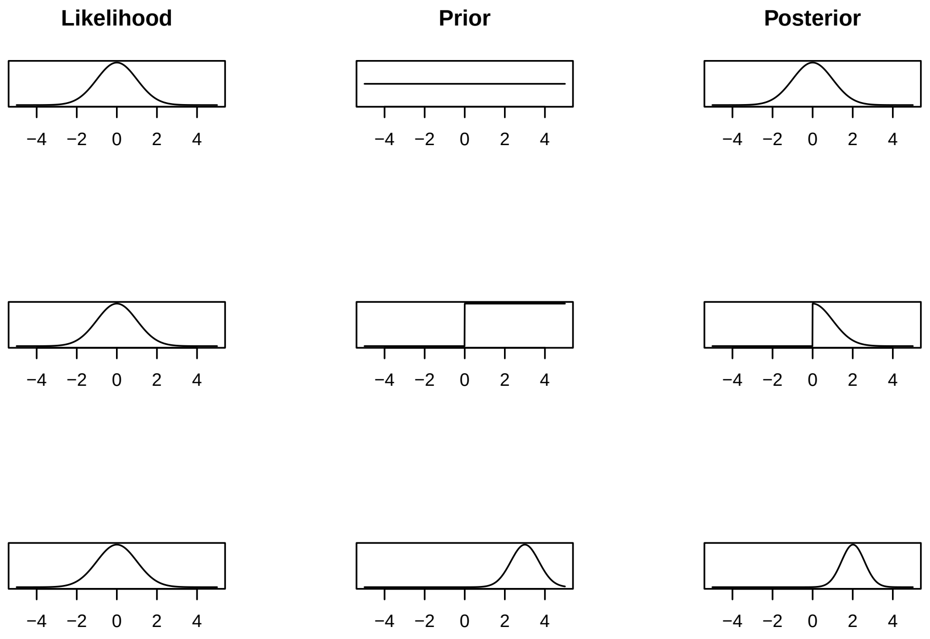
Now we have all the ingredients to construct the posterior distribution. We multiply the likelihood function evaluated at $y = 0$ by each of the three prior distributions. This gives us the posterior distributions. These are

1. For the uniform prior distribution, the posterior distribution is $\pi(\theta \mid y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\theta^2\right)$ for $\theta \in [-1000, 1000]$.

2. For the uniform prior distribution, the posterior distribution is $\pi(\theta \mid y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\theta^2\right)$ for $\theta \in [-1000, 1000]$.
3. For the normal prior distribution, as we are only interested in the posterior distribution up to proportionality, we can write it as $\pi(\theta \mid y) \propto \exp\left(-\frac{1}{2}\theta^2\right) \exp\left(-\frac{1}{2}\left(\frac{\theta-3}{0.7}\right)^2\right)$. Combining like terms, gives $\pi(\theta \mid y) \propto \exp\left(-\frac{1}{2}\left(\frac{1.7\theta^2-6\theta}{0.7^2}\right)\right)$ for $\theta \in \mathbb{R}$.

```r
#The likelihood function is the normal PDF
#To illustrate this, we evaluate this from [-5, 5].
x <- seq(-5, 5, 0.01)
likelihood <- dnorm(x, mean = 0, sd = 1)

#The first prior distribution we try is a
#uniform [-1000, 1000] distribution. This is a
#vague prior distribution.
uniform.prior <- rep(1, length(x))
posterior1 <- likelihood*uniform.prior


#The second prior distribution we try is a uniform
#[0, 1000] distribution, i.e. theta is non-negative.
step.prior <- ifelse(x >= 0, 1, 0)
posterior2 <- likelihood*step.prior


#The third prior distribution we try is a
#specific normal prior distribution. It
#has mean 3 and variance 0.7.
normal.prior <- dnorm(x, mean = 3, sd = 0.7)
posterior3 <- likelihood*normal.prior

#Now we plot the likelihoods, prior and posterior distributions.
#Each row corresponds to a different prior distribution. Each
```

```
#column corresponds to a part in Bayes' theorem.
par(mfrow = c(3, 3))
plot(x, likelihood, type = 'l', xlab = "", ylab = "", yaxt = "n", main = "Likelihood")
plot(x, uniform.prior, type = 'l', yaxt = "n", xlab = "", ylab = "", main = "Prior")
plot(x, posterior1, type = 'l', yaxt = "n", xlab = "", ylab = "", main = "Posterior")
plot(x, likelihood, type = 'l', xlab = "", ylab = "", yaxt = "n")
plot(x, step.prior, type = 'l', yaxt = "n", xlab = "", ylab = "")
plot(x, posterior2, type = 'l', yaxt = "n", xlab = "", ylab = "")
plot(x, likelihood, type = 'l', xlab = "", ylab = "", yaxt = "n")
plot(x, normal.prior, type = 'l', yaxt = "n", xlab = "", ylab = "")
plot(x, posterior3, type = 'l', yaxt = "n", xlab = "", ylab = "")
```



1. The posterior distribution is proportional to the likelihood function. The prior distribution closely matches frequentist inference. Both the MLE and posterior mean are 0.
2. We get a lopsided posterior distribution, that is proportional to the likelihood function for positive values of θ, but is 0 for negative values of θ.
3. We get some sort of average of the likelihood function and the prior distribution. Had we collected more data, the posterior distribution would have been weighted toward the information from the likelihood function more.

Chapter 2

Programming in R

2.1 Random Numbers, For Loops and R

This first computer lab is about getting used to R. The first step is to download R and Rstudio.

- Download R
- Download RStudio IDE

The easiest way to learn R is by using it to solve problems. The lab contains four exercises and three ways of approaching the exercise (easy, medium and hard). If you're new to R, use the easy approach and copy and paste the code straight into R – you'll need to fill in a few blanks though. If you've used R before, or a similar programming language, stick to the medium and hard approaches. This is also an exercise in using Google. Googling around a problem of for specific commands can allow you to quickly find examples (most likely on Stack Overflow) with code you can use.

There are three aims of this lab:

1. Getting used to programming in R.
2. Generating random numbers in R.
3. Creating for loops in R.

Example 2.1. Computationally verify that the Poisson distribution with rate $\lambda = 100$ can be approximated by a normal distribution with mean and variance 100.

To do this, we can generate lots of samples from a Poisson(100) distribution and plot them on top of the density function of the normal distribution with mean and variance 100.

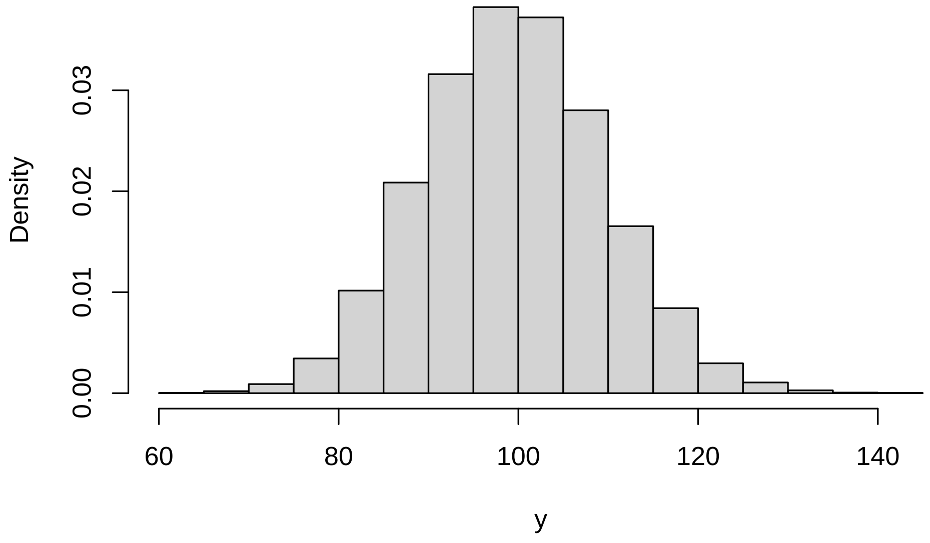
R has four built-in functions for working with distributions. They take the form rdist, ddist, pdist, and qdist. You replace the dist part with the name of the distribution you want to work with, for example unif for the uniform distribution or norm for the normal distribution. As we are working with the the Poisson distribution, we will use pois. The prefixes allow you to work with the distribution in different ways: r gives you random numbers sampled form the distribution, d evaluates the density function, p evaluates the density function, and q evaluates the inverse density function (or quantile function).

The function rpois allows us to generate samples from a Poisson distribution. We store 10,000 samples in a vector y by calling

```
y <- rpois(n = 10000, lambda = 100)
```

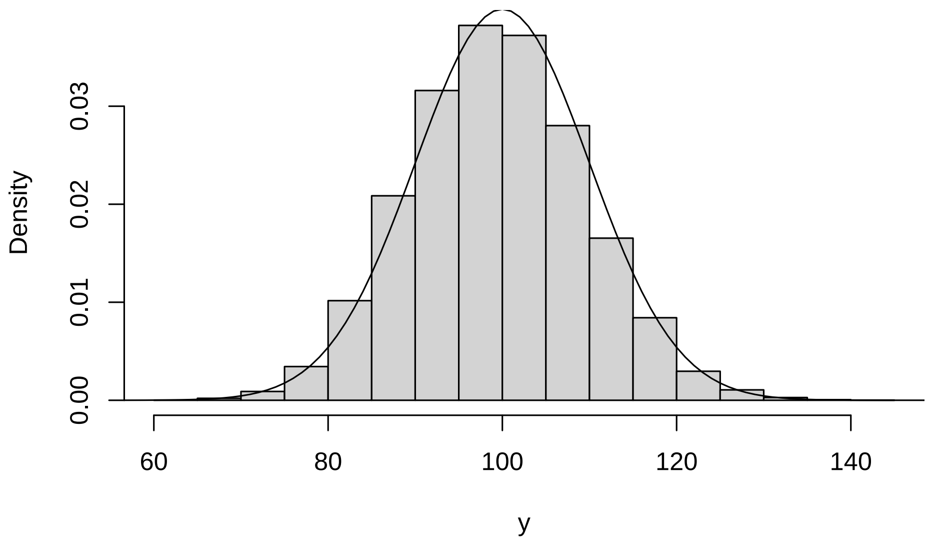
We can generate a histogram of y using the hist command. Setting freq = FALSE, makes R plot a density histogram instead of a frequency histogram. Typing ?hist will give you more information about this

```
hist(y, freq = FALSE, xlab = "y", main = "")
```



The last thing to do is to plot the normal density on top. There are a couple of ways of doing this. The way below generates a uniform grid of points and then evaluates the density at each point. Finally, it adds a line graph of these densities on top.

```
x <- seq(from = 50, to = 150, by = 1)          #create uniform grid on [50, 150]
density <- dnorm(x, mean = 100, sd = sqrt(100)) #compute density

#plot together
hist(y, freq = FALSE, xlab = "y", main = "")
lines(x, density)
```



The two match up well, showing the normal distribution is a suitable approximation here.

Over the next two sessions, you will need to solve the following four problems in R. You can type `?` before any function in R (e.g. `?rnorm`) to bring up R's helpage on the function. Googling can also bring up lots of information, possible solutions and support.

Exercise 2.1. The changes in the Birmingham stock exchange each day can be modelled using a normal distribution. The price on day i, X_i is given by

$$X_i = \alpha X_{i-1}, \qquad \alpha \sim N(1.001, 0.005^2).$$

The index begins at $X_0 = 100$. Investigate the distribution of the value of the stock market on days 50 and 100.

Hard. Use a simulation method to generate the relevant distributions.

Medium. Simulate the value for α for each of the 100 days and use the `cumprod` command to plot a trajectory. Use a for loop to repeat this 100 times and investigate the distribution of the value of the stock market on days 50 and 100.

Easy. Fill in the blanks in the following code.

```
# Plot one ---------------------------------------------------------------
x <- rnorm(n = , mean = , sd = ) #Simulate daily change for 100 days
plot(, type = 'l') #multiply each day by the previous days

# Plot 100 realisations --------------------------------------------------
market.index <- matrix(NA, 100, 100) #Initialise a matrix to store trajectories
for(i in 1:100){
  x <- rnorm(n = , mean = , sd = )
  market.index [, i] <-
}

#Plot all trajectories
matplot(market.index, type = 'l')

#Get distribution of days 50 and 100
hist()
hist()
quantile(, )
quantile(, )
```

Exercise 2.2. You are an avid lottery player and play the lottery twice a week, every week for 50 years (a total of 5,200 times). The lottery has 50 balls labeled 1, ..., 50 and you play the same 6 numbers each time. Six out of the 50 balls are chosen uniformly at random and the prize money is shown in the table below.

Numbers Matched	Prize Amount
0-2	£0
3	£30
4	£140
5	£1,750
6	£1,000,000

It costs you £2 to play each time. Simulate one set of 5,200 draws. How much do you win? What is your total profit/loss?

Hard. Use a for loop and sequence of if else statements to generate your prize winnings.

Medium. Use a for loop to generate the lottery numbers and prize winnings for each draw. Use the `sample` function to generate a set of lottery numbers and check they match against your numbers using the `%in%` function. Finally, use if else statements to check how much you have won each time.

Easy. Fill in the blanks in the following code.

```
my.numbers <-

#For loop to generate lottery numbers and prize winnings
prize <- numeric(5200)
for(i in 1:5200){

  #Generate lottery numbers
  draw <- sample(, )

  #Check how many match my numbers
  numbers.matched <- #use %in% function

  #Compute prize winings
  if(numbers.matched < 3)
    prize[i] <- 0
  else if()
    prize[i] <-  30
  else if()
    prize[i] <-  140
  else if()
    prize[i] <-  1750
  else
    prize[i] <- 1000000
}

#Summarise prize winnings
table(prize)
hist(prize)
sum(prize) - 2*5200
```

Exercise 2.3. Estimate π.

Hard. Use a rejection sampling algorithm.

Medium. Generate lots of points (x, y) on the unit square $[0, 1]^2$. Check each point to see if it lies within the unit circle. Use the proportion of points that lie within the unit circle to estimate π.

Easy. Fill in the blanks in the following code.

```
#Sample on unit square
N <- 10000      #number of points
x <-            #sample N points uniformly at random on [0, 1]
y <-            #sample N points uniformly at random on [0, 1]

#Estimate pi
r.sq                <- x^2 + y^2               #check how far from origin
number.inside.circle <-          #count how many points inside unit cirlce
pi.estimate         <-

#Plot points
par(pty = "s")         #make sure plot is square
plot(x, y, cex = 0.1)  #plot points
theta <- seq(0, pi/2, 0.01) #plot unit circle
lines(x = cos(theta), y = sin(theta), col = "red")
```

Extra. Use a for loop to repeat this for $N = \{1, \ldots, 10000\}$. Record the estimate for π for each value and the relative error.

Exercise 2.4. A linear congruential generator (LCG) is a simple algorithm for generating random integers. Given a starting value X_0, it generates a sequence of integers according to

$$X_{i+1} = aX_i + c \mod m.$$

Software that generates numbers using an LCG Setting $a = 3$, $c = 2$, $m = 7$ and $X_0 = 0$, generate 20 samples from this generator.

1. Investigate the 'randomness' of this generator by creating the delay plot, where X_{i-1} is plotted against X_i
2. One way to improve the quality of these generators is to shuffle the sequence generated. Generate rate two sequences X and Y from two different LCGs, and report the shuffled sequence $Z_j = X_{Y_j}$. For the sequence Y use the values $a = 5$, $c = 1$, $m = 8$ and $Y_0 = 2$.
3. As the past two exercises show, LCGs are notoriously poor. in the 1960s and 70s, RANDU was a widely used LCG developed by IBM. According to Wikipedia > IBM's RANDU is widely considered to be one of the most ill-conceived random number generators ever designed, and was described as "truly horrible" by Donald Knuth.

The RANDU LCG uses $a = 2^{16} + 3$, $c = 0$, $m = 2^{31}$ and $Y_0 = 1$. Generate a sequence of 10,000 pseudorandom variables from the RANDU LCG and create the delay plot.

The delay plot seems to show little relationship between X_i and X_{i+1}. The third order delay plot is a 3d-plot with coordinate (X_i, X_{i+1}, X_{i+2}) and this plot shows a different picture. Create this plot using the code

```
#install.packages("scatterplot3d") #you may need to install this package
scatterplot3d::scatterplot3d(X[1:9998], X[2:9999], X[3:10000], angle=154,
                             xlab = expression(X[i]), ylab = expression(X[i+1]), zlab =
```

This is what makes the RANDU LCG so poor. Write down X_{i+1} and X_{i+2} in terms of X_i. Show that $X_{i+2} = \alpha X_{i+1} + \beta X_i$.

Hard. Use a for loop to construct sequences from the LCGs X and Y.

Medium. Create a for loop to generate the value for the sequence X_i for $i = 1, \dots, 20$. Modular arithmetic can be performed using the `%%` function. Create a new for loop to construct the sequence Y. To shuffle the sequence X using Y, you will need to subset X by Y in R.

Easy. Fill in the blanks in the code below

```
# 1. Shuffling ----------------------------------------------------------------
X <- numeric(21) #initialise vector to store X

#Set values for LCG
a <-
```

```
c <-
m <-
X[1]<-


#Run Generator
for(i in 2:21){
  X[i] <-
}
X

#Delay plot
plot( , , xlab = expression(X[i-1]), ylab = expression(X[i]), type = 'l')



# 2. Shuffling ---------------------------------------------------------------

Y <- numeric(21) #initialise vector to store Y

#Set values for LCG
a <-
c <-
m <-
Y[1]<-


#Run Generator
for(i in 2:50){
  Y[i] <-
}

#report sequence
Y
X[Y]

#Plot delay plot
plot(x = ,y = , xlab = expression(Z[i-1]), ylab = expression(Z[i]), type = 'l')
```

2.2 Functions in R

The purpose of this lab is to learn how to write functions is R. Functions are wrappers that allow you to easily repeat commands, as well as customise specific pieces of code.

2.2.1 Built in commands

R has many build in commands and you used these in Computer Lab I. An example is the `runif` command from the second exercise. This function generates random numbers from an interval. The code chunk below shows it in action:

```
u <- runif(n = 10, min = -1, max = 1)
u
```

```
##  [1] -0.87157252 -0.14145068  0.08933733 -0.90894291  0.99791288  0.62516311
##  [7] -0.92640731  0.58073921 -0.23192204  0.20427438
```

The functions has three **arguments**: *n* the number of samples to be generated, *min* the lower limit of the interval, *max* the upper limit of the interval. In the code chunk above 10 random numbers were generated from the interval [-1, 1]. In R, you don't need to label the arguments, so the following will sample the same number of samples from the same interval:

```
u <- runif(10, -1, 1)
```

Although in most cases it helps to label the arguments for readability and avoiding undefined behaviour. Note that if you decide to omit the argument names in the function call, the arguments must appear exactly in the order defined by the function prototype (check the documentation ?function for specific cases).

2.2.2 User defined functions

In many cases, we will need to repeat the same piece of code over and over again, or we will need to run it again with different values. In this case, we can write our own function. In R, there are two ways to type your own function. The first is to write a full function definition. The basic template is

```
name.of.function <- function(arguments){

  #do something
  #produce result

  return(result)

}
```

The second way is an in-line function, which is sometimes useful for short functions. The template is

```
name.of.function <- function(arguments) #do something
```

In this module, we're going to use the full function way of writing functions.

Example 2.2. In this example, we're going to write a function to evaluate the normal density function. The density function is given by

$$\pi(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}}.$$

We will need our function to take three arguments, the value at which the density function needs to be evaluated, and the mean and standard deviation of the distribution.

```
normal.density <- function(x, mu, sigma){

  fraction.term <- 1/sqrt(2*pi*sigma^2)
  exponent.term <- -1/(2*sigma^2)*(x-mu)^2

  result <- fraction.term*exp(exponent.term)
  return(result)

}
```

We have split up the density into two parts to make it easier to code up and read. R has its own inbuilt normal density function `dnorm` and we can compare our function against R's. Although R's is faster and more reliable, we should get the same results.

```
normal.density(x = 0.5, mu = 1, sigma = 0.5)
```

```
## [1] 0.4839414
```

```
dnorm(x = 0.5, mean = 1, sd = 0.5)
```

```
## [1] 0.4839414
```

Why might R's function be faster and more reliable than ours?

Exercise 2.5. Write a function to evaluate the log of the probability density function of a Poisson distribution with rate λ.

Exercise 2.6. Consider the stock exchange problem in Exercise 2.1. Write a function that simulates 100 days of the stock exchange. Use the `replicate` function to call this function 10,000 times.

2.3 Good Coding Practices

In the past two labs, we've written code to solve different problems. In this lab, we're going to take a step back and think about what good R code does and doesn't look like.

2.3.1 Code Style

Code should be both efficient and easy to read. In most cases it's better to write code that's easy to read and less efficient, than highly efficient code that's difficult to read. Some basic principles to make code easy to read are:

1. Write short functions names, e.g. `buy.loot.box` is better than `player.buys.one.loot.boox` or `blb`.
2. Document and comment code. In R comments start with `#`.
3. Multiple short functions are better than long functions that do multiple things.
4. Be consistent.

Example 2.3. Review the tidyverse Style Guide.

Example 2.4. Review Google's R Style Guide.

One way to ensure code style is consistent and bug free is to carry out code reviews. These are common both in academia and industry. A code review is where someone else goes through your code line-by-line ensuring it conforms to the company style and doesn't have any bugs.

Exercise 2.7. The following code is for Exercises 2.1 about the stock exchange. Restyle the code so it is easy to read.

```
# Plot one ------------------------------------------------------------------
rnorm(100,1.001,0.5) -> x
plot(100*cumprod(x),type ='l')
# Plot 100 realisations ----------------------------------------------------
X <- matrix(NA, 100, 100)
for(i in 1:100){
  x <- rnorm(100, 1.001, 0.005)
  X[,i] <- 100*cumprod(x)
}
matplot(X,type ='l')
hist(X[50,]);hist(X[100,]);quantile(X[50,], c(0.25, 0.5, 0.75));quantile(X[100,], c(0.
```

Exercise 2.8. In pairs or groups, carry out a code review for one of your solutions to an exercise from a previous lab. Remember to

1. Make sure the coding style is consistent.
2. Identify any bugs.
3. Be respectful and constructive in your feedback.

Chapter 3

Bayesian Inference

Whereas Chapter 1 dealt with the fundamentals of Bayesian inference and definitions, Chapter 3 is much more practical. We are going to be deriving posterior distributions and proving when it does and doesn't work.

3.1 The Binomial Distribution

The first example we are going to go through is with the Binomial distribution.

Example 3.1. A social media company wants to determine how many of its users are bots. A software engineer collects a random sample of 200 accounts and finds that three are bots. She uses a Bayesian method to estimate the probability of an account being a bot. She labels the accounts with a 1 if they are a bot and 0 if there is are a real person. The set of account labels is given by $y = \{y_1, \dots, y_{200}\}$ and the probability an account is a bot is θ. By Bayes' theorem, we obtain the following,

$$\pi(\theta \mid y) \propto \pi(y \mid \theta)\pi(\theta).$$

Likelihood function $\pi(y \mid \theta)$. We observe 200 trials each with the same probability of success (denoted by θ) and probability of failure (given by $1 - \theta$). The Binomial distribution seems the most suitable way of modelling this. Therefore, the likelihood function is given by,

$$\pi(y \mid \theta) = \binom{200}{3} \theta^3 (1-\theta)^{197},$$

assuming that any two accounts being a bot are independent of one another.

Prior distribution $\pi(\theta)$. We now need to describe our prior beliefs about θ. We have no reason to suggest θ takes any specific value, so we use a uniform prior distribution $\theta \sim U[0,1]$, where $\pi(\theta) = 1$ for $\theta \in [0,1]$.

Posterior distribution $\pi(\theta \mid y)$. We can now derive the posterior distribution up to proportionality

$$\pi(\theta \mid y) \propto \theta^3(1-\theta)^{197}.$$

This functional dependence on θ identifies the $\pi(\theta \mid y)$ is a Beta distribution. The PDF for the beta distribution with shape parameters α and β is

$$\pi(x \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}.$$

The posterior distribution is therefore $\theta \mid y \sim \text{Beta}(4, 198)$.

3.2 Reporting Conclsuions from Bayesian Inference

In the previous example, we derived the posterior distribution $\theta \mid y \sim \text{Beta}(4, 198)$. But often, we want to share more descriptive information about our beliefs given the observed data. In this example, the posterior mean given the data is $\frac{4}{198} = \frac{2}{99}$. That is to say given the data, we expect that for every 99 accounts, two to be bots. The posterior mode for θ is $\frac{3}{200}$ or 1.5%.

It is important to share the uncertainty about out beliefs. In a frequentist framework, this would be via a confidence interval. The Bayesian analogues is a credible interval.

Definition 3.1. A **credible interval** is a central interval of posterior probability which corresponds, in the case of a $100(1-\alpha)\%$ interval, to the range of values that capture $100(1-\alpha)\%$ of the posterior probability.

Example 3.2. The 95% credible interval for the Binomial example is given by

```r
cred.int.95 <- qbeta(c(0.025, 0.975), 4, 198)
round(cred.int.95, 3)
```

```
## [1] 0.005 0.043
```

This says that we believe there is a 95% chance that the probability of an account being a bot lies between 0.005 and 0.043. This is a much more intuitive definition to the confidence interval, which says if we ran the experiment an infinite number of times and computed an infinite number of confidence intervals, 95% of them would contain the true value of θ.

3.3 The Exponential Distribution

Example 3.3. An insurance company want to estimate the time until a claim is made on a specific policy. They describe the rate at which claims come in by λ. The company provides a sample of 10 months at which a claim was made

$y = \{14, 10, 6, 7, 13, 9, 12, 7, 9, 8\}$. By Bayes' theorem, the posterior distribution for λ is

$$\pi(\lambda \mid y) \propto \pi(y \mid \lambda)\pi(\lambda).$$

Likelihood function $\pi(y \mid \lambda)$. The exponential distribution is a good way of modelling lifetimes or the length of time until an event happens. Assuming all the claims are independent of one another, the likelihood function is given by

$$\begin{aligned}\pi(y \mid \lambda) &= \prod_{i=1}^{10} \lambda e^{-\lambda y_i} \\ &= \lambda^{10} e^{-\lambda \sum_{i=1}^{10} y_i} \\ &= \lambda^{10} e^{-95\lambda}.\end{aligned}$$

Prior distribution $\pi(\lambda)$. As we are modelling a rate parameter, we know it must be positive and continuous. We decide to use an exponential prior distribution for λ, but leave the choice of the rate parameter up to the insurance professionals at the insurance company. The prior distribution is given by $\lambda \sim \text{Exp}(\gamma)$.

Posterior distribution $\pi(\lambda \mid y)$. We now have all the ingredients to derive the posterior distribution. It is given by

$$\begin{aligned}\pi(\lambda \mid y) &\propto \lambda^{10} e^{-95\lambda} \times e^{-\gamma\lambda} \\ &\propto \lambda^{10} e^{-(95+\gamma)\lambda}\end{aligned}$$

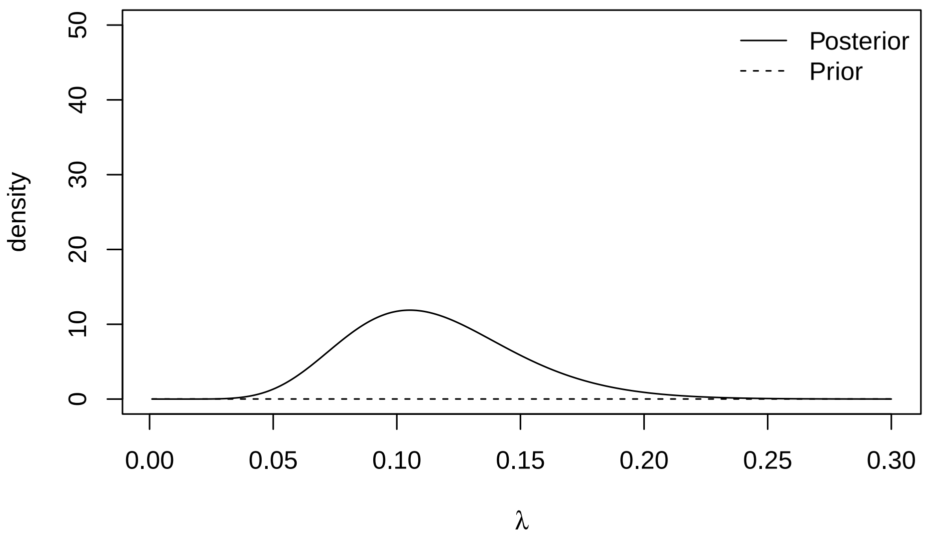
The functional form tells us that the posterior distribution is a Gamma distribution. The PDF of a gamma random variable with shape α and rate β is

$$\pi(x \mid \alpha, \beta) = \frac{\alpha^\beta}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

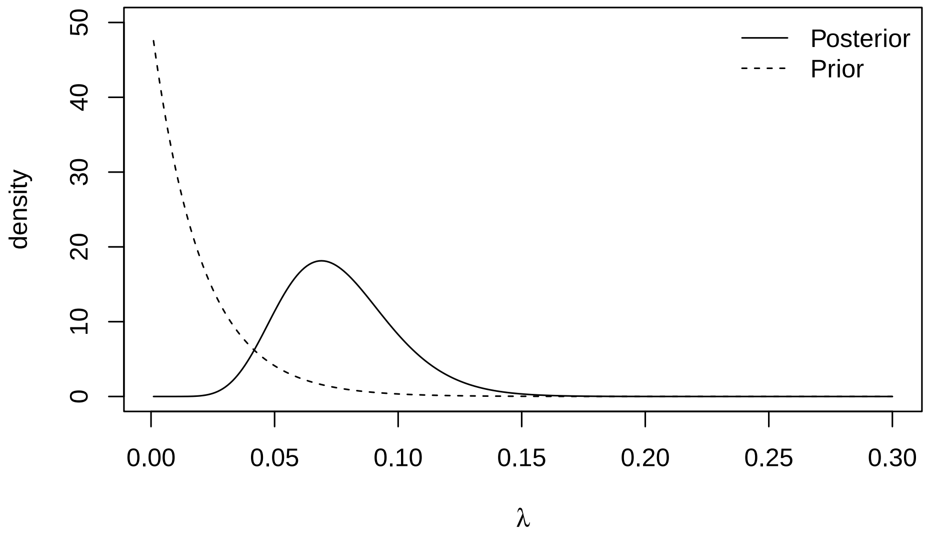
The distribution of the rate of the claims given the observed data is $\lambda \mid y \sim \text{Gamma}(11, 95 + \gamma)$.

The posterior mean months until a claim is $\frac{11}{95+\gamma}$. We can see the effect of the choice of rate parameter in this mean. Small values of γ yield vague prior distribution, which plays a minimal role in the posterior distribution. Large values of γ result in prior distributions that contribute a lot to the posterior distribution. The plots below show the prior and posterior distributions for $\gamma = 0.01$ and $\gamma = 50$.

```
plot.distributions <- function(gamma.prior){
  #evaluate at selected values of lambda
  lambda <- seq(0.001, 0.3, 0.001)

  #evaluate prior density
  prior <- dexp(lambda, rate = gamma.prior)

  #evaluate posterior density
  posterior <- dgamma(lambda, shape = 11, rate = 95 + gamma.prior)


  #plot
  plot(lambda, posterior, type= 'l',
       ylim = c(0, 50), xlab = expression(lambda), ylab = "density")
  lines(lambda, prior, lty = 2)
  legend('topright', lty = c(1, 2), legend = c("Posterior", "Prior"),
         bty = "n")
}

plot.distributions(0.01)
```



```
plot.distributions(50)
```



The insurance managers recommend that because this is a new premium, a vague prior distribution be used and $\gamma = 0.01$. The posterior mean is $\frac{11}{95.01} \approx 0.116$ and the 95% credible interval is

```
round(qgamma(c(0.025, 0.975), 11, 95.01), 3)
```

```
## [1] 0.058 0.194
```

3.4 The Normal Distribtuion

The Normal distribution is incredibly useful for modelling a wide range of natural phenomena and in its own right. We're now going to derive posterior distributions for the normal distribution. As we're going to see, the concepts behind deriving posterior distributions are the same as in the previous two examples. However, the algebraic accounting is a lot more taxing.

Example 3.4. Suppose we observe N data points $y = \{y_1, \dots, y_N\}$ and we assume $y_i \sim N(\mu, \sigma^2)$ and each observation is independent. Suppose that, somehow, we know the population standard deviation and we wish to estimate the population mean μ. By Bayes' theorem, the posterior distribution is

$$\pi(\mu \mid y, \sigma^2) \propto \pi(y \mid \mu, \sigma^2)\pi(\mu)$$

Likelihood function. As the observations are independent, the likelihood function is given by the product of the N normal density functions as follows,

$$\begin{aligned}\pi(y \mid \mu, \theta^2) &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y_i-\mu)^2}{2\sigma^2}\right\} \\ &= (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left\{-\sum_{i=1}^{N} \frac{(y_i-\mu)^2}{2\sigma^2}\right\}.\end{aligned}$$

Prior distribution We suppose we have no prior beliefs about the values that μ can take. We assign a normal prior distribution to $\mu \sim N(\mu_0, \sigma_0^2)$ despite it being a time. We will set $\mu = 0$ and $\sigma_0^2 = 1000$ to signify our vague prior beliefs, but, for ease, we will use the symbolic values during the derivation of the posterior distribution. We have

$$\pi(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right\}.$$

Posterior distribution. To derive the posterior distribution, up to proportionality, we multiply the prior distribution by the likelihood function. As the fractions out the front of both terms do not depend on μ, we can ignore these.

$$\begin{aligned}\pi(\mu \mid y, \sigma^2) &\propto \exp\left\{-\sum_{i=1}^{N} \frac{(y_i-\mu)^2}{2\sigma^2}\right\} \exp\left\{\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right\} \\ &= \exp\left\{-\sum_{i=1}^{N} \frac{(y_i-\mu)^2}{2\sigma^2} - \frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right\} \\ &= \exp\left\{-\frac{\sum_{i=1}^{N} y_i^2}{2\sigma^2} + \frac{\mu\sum_{i=1}^{N} y_i}{\sigma^2} - \frac{N\mu^2}{2\sigma^2} - \frac{\mu^2}{2\sigma_0^2} + \frac{\mu\mu_0}{\sigma_0^2} - \frac{\mu_0^2}{2\sigma_0^2}\right\}.\end{aligned}$$

We can drop the first and last term as they do not depend on μ. With some arranging, the equation becomes

$$\pi(\mu \mid y, \sigma^2) \propto \exp\left\{-\mu^2\left(\frac{N}{2\sigma^2} + \frac{1}{2\sigma_0^2}\right) + \mu\left(\frac{\sum_{i=1}^{N} y_i}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right)\right\}$$

Defining $\mu_1 = \left(\frac{\sum_{i=1}^{N} y_i}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right)$ and $\sigma_1^2 = \left(\frac{N}{\sigma^2} + \frac{1}{\sigma_0^2}\right)^{-1}$ tidies this up and gives

$$\pi(\mu \mid y, \sigma^2) \propto \exp\left\{-\frac{\mu^2}{2\sigma_1^2} + \mu\mu_1\right\}.$$

Our last step to turning this into a distribution is completing the square. Consider the exponent term, completing the square becomes

$$-2\sigma_1^2\mu^2 + \mu\mu_1 = -\frac{1}{2\sigma_1^2}\left(\mu - \frac{\mu_1}{\sigma_1^2}\right)^2.$$

Therefore, the posterior distribution, up to proportionality, is given by

$$\pi(\mu \mid y, \sigma^2) \propto \exp\left\{-\frac{1}{2\sigma_1^2}\left(\mu - \frac{\mu_1}{\sigma_1^2}\right)^2\right\},$$

and so the posterior distribution of μ is $\mu \mid y, \sigma^2 \sim N(\mu_1, \sigma_1^2)$.

It may help to consider the meaning of μ_1 and σ_1^2. The variance of the posterior distribution can be thought of as the weighted average of the population and sample precision, where the weight is the number of data points collected. The interpretation of the posterior mean can be seen more easily by writing is as

$$\mu = \sigma_1^2\left(\frac{N\bar{y}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right).$$

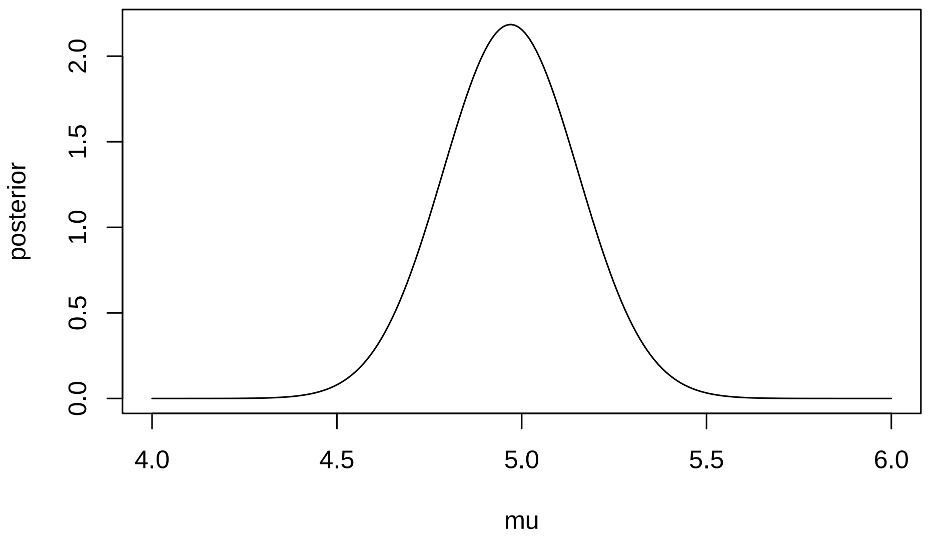
The posterior mean is partially defined through the weighted average of the population and prior means, where the weighting depends on the number of data points collected and how precise the distributions are.

Now we have derived the posterior distribution, we can explore it using R. We simulate some data with $N = 30$, $\mu = 5$ and $\sigma^2 = 1$.

```
#data
N <- 30
sigma <- 1
y <- rnorm(N, 5, sigma)

#prior
sigma0 <- 1000
mu0     <- 0

#posterior
sigma1.sq <- (1/(sigma0^2)  + N/(sigma^2))^-1
mu1       <- sigma1.sq*(sum(y)/(sigma^2) + mu0/(sigma0^2))

c(mu1, sigma1.sq) #output mean and variance
```

```
## [1] 4.96982697 0.03333333
```

```
#Create plot
mu <- seq(4, 6, 0.01)
posterior <- dnorm(mu, mean = mu1, sd = sqrt(sigma1.sq))
plot(mu, posterior, type ='l')
```



The 95% credible interval for the population's mean reaction time is

```r
qnorm(c(0.025, 0.975), mu1, sqrt(sigma1.sq))
```

```
## [1] 4.611988 5.327666
```

When the prior distribution induces the same function form in the posterior distribution, this is known as conjugacy.

Definition 3.2. If the prior distribution $\pi(\theta)$ has the same distributional family as the posterior distribution $\pi(\theta \mid y)$, then the prior distribution is a **conjugate prior distribution**.

3.5 Hierarchical Models

In many modelling problems, there will be multiple parameters each related to one another. These parameters may be directly related to the model, or they may be parameters we introduce through prior distributions. We can form a hierarchy of these parameters, from closest to further from the data, to construct our model.

Example 3.5. Let's consider 3.3 again. We have some data y that are assumed to have been generated from an Exponential distribution with rate parameter λ. We placed an Exponential prior distribution with rate γ on λ and the posterior distribution was $\lambda \mid y \sim \text{Gamma}(11, 95 + \gamma)$.

In that example, we discussed how the choice of γ can affect the posterior distribution and conclusions presented to the company. One option is to place a prior distribution on γ – a hyperprior distribution. The hierachy formed is

$$\begin{aligned} y \mid \lambda &\sim \text{Exp}(\lambda) && \text{(likelihood)} \\ \lambda \mid \gamma &\sim \text{Exp}(\gamma) && \text{(prior distribution)} \\ \gamma \mid \nu &\sim \text{Exp}(\nu) && \text{(hyperprior distribution)} \end{aligned}$$

. By Bayes' theorem, we can write the posterior distribution as

$$\begin{aligned} \pi(\lambda, \gamma \mid y) &\propto \pi(y \mid \lambda)\pi(\lambda \mid \gamma)\pi(\gamma) \\ &\propto \lambda^{10} e^{-\lambda(95+\gamma)} \nu e^{-\nu\gamma}. \end{aligned}$$

To derive the full conditional distributions, we only consider the terms that depends on the parameters we are interested in. The full conditional distribution for λ is

$$\pi(\lambda \mid y,\, \gamma) \propto \lambda^{10} e^{-\lambda(95+\gamma)}.$$

This is unchanged and shows that $\lambda \mid y, \gamma \sim \text{Gamma}(11, 95 + \gamma)$. The full conditional distribution for γ is

$$\pi(\gamma \mid y,\, \lambda) \propto e^{-\nu\gamma}.$$

Therefore the full conditional distribution of γ is $\gamma \mid y,\, \lambda \sim \text{Exp}(\lambda + \nu)$. In the next chapter, we will look at how to sample from these distributions.

3.6 Prediction

In many cases, although we are interested in drawing inference for the model parameters, what we may also be interested in is predicting new values, whose distribution is determined by the model parameters and observed data.

Definition 3.3. Suppose we observe some data y given some model parameters θ and assign a prior distribution to θ and hence derive the posterior distribution $\pi(\theta \mid y)$. The quantity we are interested in is some future observation z, we would like to the distribution of z given the observed data y, denoted by $\pi(z \mid y)$. This distribution, known as the **posterior predictive distribution** of z must be exhibited as a mixture distribution over the possible values of θ and is written as,

$$\pi(z \mid y) = \int \pi(z \mid \theta)\pi(\theta \mid y)\, d\theta.$$

Example 3.6. Students have to submit coursework for a particular statistical modules. However, each semester a number of students miss the deadline and hand in their coursework late. Last year, three out of 20 students handed their coursework in late. This year, the course has thirty students in. How many students can we expect to hand in their coursework late?

We can model the number of students handing their coursework in late, denoted by Y, using a Binomial distribution, i.e. $Y \sim \text{Bin}(n, \theta)$ where n is the number of students and θ is the probability of any particular student handing in their coursework late. As in Example 3.1, we assign a uniform prior distribution to $\theta \sim U[0, 1]$. Given then observed data, we can derive $\theta \mid y \sim Beta(4, 28)$ (See problem sheets for derivation).

Now we can derive the posterior predictive distribution of Z, the number of students who hand in late. We model Z using a Binomial distribution, $Z \sim \text{Bin}(30, \theta)$. The distribution of Z given the observed data is

$$\begin{aligned}
\pi(z \mid y) &= \int_0^1 \pi(z \mid \theta)\pi(\theta \mid y)\, d\theta \\
&= \int_0^1 \binom{30}{z} \theta^z (1-\theta)^{30-z} \frac{\Gamma(32)}{\Gamma(4)\Gamma(28)} \theta^3 (1-\theta)^{27}\, d\theta \\
&= \binom{30}{z} \frac{\Gamma(32)}{\Gamma(4)\Gamma(28)} \int_0^1 \theta^{z+3}(1-\theta)^{57-z}\, d\theta
\end{aligned}$$

This integral is difficult to evaluate immediately. But by multiplying (and dividing outside the integral) by a constant, we can turn it into the density function of a Beta$(5 + z, 58 - z)$ random variable. This integrates to 1.

$$\begin{aligned}
\pi(z \mid y) &= \binom{30}{z} \frac{\Gamma(32)}{\Gamma(4)\Gamma(28)} \frac{\Gamma(z+4)\Gamma(58-z)}{\Gamma(62)} \int_0^1 \frac{\Gamma(62)}{\Gamma(z+4)\Gamma(58-z)} \theta^{z+3}(1-\theta)^{57-z}\, d\theta \\
&= \binom{30}{z} \frac{\Gamma(32)\Gamma(z+4)\Gamma(58-z)}{\Gamma(4)\Gamma(28)\Gamma(62)} \qquad \text{for } z \in \{0, 1, ..., 30\}.
\end{aligned}$$

This code implements the distribution

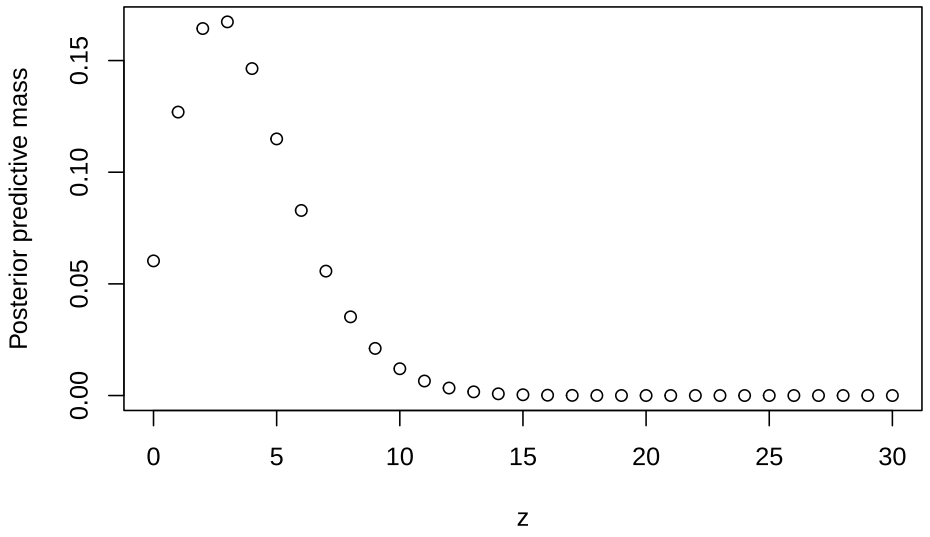
```r
beta.binom.posterior.predictive.distribution <- function(z){


  numerator <- gamma(32)*gamma(z + 4)*gamma(58-z)
  denominator <- gamma(4)*gamma(28)*gamma(62)

  output <- choose(30, z)*numerator/denominator
  return(output)

}
```

We can check that our posterior predictive distribution is a valid probability mass function by checking that the probabilities sum to one.

```
z <- 0:30
ppd <- beta.binom.posterior.predictive.distribution(z)
sum(ppd)
```

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

```
plot(z, ppd, xlab = "z", ylab = "Posterior predictive mass")
```



The expected number of students who hand in late is 3.75 and there's a 95% chance that up to 8 hand in late.

```
z%*%ppd #expectation
```

```
##      [,1]
## [1,] 3.75
```

```
cbind(z, cumsum(ppd)) #CDF
```

```
##        z
##  [1,]  0 0.06029453
##  [2,]  1 0.18723037
##  [3,]  2 0.35156696
##  [4,]  3 0.51889148
##  [5,]  4 0.66530044
##  [6,]  5 0.78021765
##  [7,]  6 0.86309065
##  [8,]  7 0.91880359
##  [9,]  8 0.95404202
## [10,]  9 0.97513714
## [11,] 10 0.98713498
## [12,] 11 0.99363285
## [13,] 12 0.99698773
## [14,] 13 0.99863936
## [15,] 14 0.99941423
## [16,] 15 0.99976022
## [17,] 16 0.99990696
## [18,] 17 0.99996591
## [19,] 18 0.99998826
## [20,] 19 0.99999622
## [21,] 20 0.99999887
## [22,] 21 0.99999969
## [23,] 22 0.99999992
## [24,] 23 0.99999998
## [25,] 24 1.00000000
## [26,] 25 1.00000000
## [27,] 26 1.00000000
## [28,] 27 1.00000000
## [29,] 28 1.00000000
## [30,] 29 1.00000000
## [31,] 30 1.00000000
```

3.7 Non-informative Prior Distibrutions

We have seen in a few examples how the choice of the prior distribution (and prior parameters) can impact posterior distributions and the resulting conclusions. As the choice of prior distribution is subjective, it is the main criticism of Bayesian inference. A possible way around this is to use a prior distribution that reflects a lack of information about θ.

Definition 3.4. A **non-informative prior distribution** is a prior distribution that places equal weight on the every possible value of θ.

Example 3.7. In Example 3.1, we assigned a uniform prior distribution to the parameter θ.

Such a prior distribution can have interesting and perhaps unintended side effects. Suppose we do indeed have some parameter θ and we place a uniform prior distribution on θ such that $\theta \sim U[0,1]$. This means, for example, our prior beliefs about θ are that it is equally likely to be in $[0, 0.1]$ as it is to lie in $[0.8, 0.9]$ or any other interval of size 0.1. However, our prior beliefs about θ^2 are not uniform. Letting $\psi = \theta^2$, changing variables gives $\pi(\psi) = \frac{1}{2\sqrt{\psi}}$, something that is not uniform. That raises the question, if we have little to say about θ *a priori*, shouldn't we have little to say about any reasonable transformation of θ?

Theorem 3.1 (Jeffrey). *Given some observed data $y = \{y_1, \dots, y_N\}$, an invariant prior distribution is*

$$\pi(\theta) \propto \sqrt{I_\theta(y)},$$

where $I_\theta(y)$ is the Fisher information for θ contained in y.

Jeffrey argues that if there are two ways of parameterising a model, e.g. via θ and ψ, then the priors on these parameters should be equivalent. In other words, the prior distribution should be invariant under sensible (one-to-one) transformations.

Proof. Recall that the distribution of $\psi = h(\theta)$, for some one-to-one function h, is invariant to the distribution of θ if

$$\pi(\psi) = \pi(\theta)\left|\frac{d\theta}{d\psi}\right|.$$

Transforming the Fisher information for ψ shows

$$\begin{aligned}
I_\psi(y) &= -\mathbb{E}\left(\frac{d^2 \log \pi(y \mid \psi)}{d\psi^2}\right) \\
&= -\mathbb{E}\left(\frac{d}{d\psi}\left(\frac{d \log \pi(y|\theta(\psi))}{d\theta}\frac{d\theta}{d\psi}\right)\right) && \text{(chain rule)} \\
&= -\mathbb{E}\left(\left(\frac{d^2 \log \pi(y|\theta(\psi))}{d\theta d\psi}\right)\left(\frac{d\theta}{d\psi}\right) + \left(\frac{d \log \pi(y|\theta(\psi))}{d\theta}\right)\left(\frac{d^2\theta}{d\psi^2}\right)\right) \\
& && \text{(prod. rule)} \\
&= -\mathbb{E}\left(\left(\frac{d^2 \log \pi(y|\theta(\psi))}{d\theta^2}\right)\left(\frac{d\theta}{d\psi}\right)^2 + \left(\frac{d \log \pi(y|\theta(\psi))}{d\theta}\right)\left(\frac{d^2\theta}{d\psi^2}\right)\right) \\
& && \text{(chain rule)} \\
&= -\mathbb{E}\left(\left(\frac{d^2 \log \pi(y \mid \theta)}{d\theta^2}\left(\frac{d\theta}{d\psi}\right)^2\right)\right) \\
&= I_\theta(y)\left(\frac{d\theta}{d\psi}\right)^2.
\end{aligned}$$

Thus $\sqrt{I_\psi(y)} = \sqrt{I_\theta(y)}\left|\frac{d\theta}{d\psi}\right|$ and $\sqrt{I_\psi(y)}$ and $\sqrt{I_\theta(y)}$ are invariant prior distributions. □

Example 3.8. In Example 3.1, we modelled the number of bot accounts on a social media website by $Y \sim \text{Bin}(n, \theta)$. To construct Jeffrey's prior distribution for θ, we must first derive the Fisher information.

$$
\begin{aligned}
&\pi(y \mid \theta) = \binom{n}{y} \theta^y (1-\theta)^{n-y} \\
\Longrightarrow\ & \log \pi(y \mid \theta) = \log \binom{n}{y} + y \log\theta + (n-y)\log(1-\theta) \\
\Longrightarrow\ & \frac{\partial \log \pi(y \mid \theta)}{\partial\theta} = \frac{y}{\theta} - \frac{n-y}{1-\theta} \\
\Longrightarrow\ & \frac{\partial^2 \log \pi(y \mid \theta)}{\partial\theta^2} = -\frac{y}{\theta^2} + \frac{n-y}{(1-\theta)^2} \\
\Longrightarrow\ & \mathbb{E}\left(\frac{\partial \log \pi(y \mid \theta)}{\partial\theta}\right) = -\frac{\mathbb{E}(y)}{\theta^2} + \frac{n-\mathbb{E}(y)}{(1-\theta)^2} \\
\Longrightarrow\ & \mathbb{E}\left(\frac{\partial \log \pi(y \mid \theta)}{\partial\theta}\right) = -\frac{n\theta}{\theta^2} + \frac{n-n\theta}{(1-\theta)^2} \\
\Longrightarrow\ & \mathbb{E}\left(\frac{\partial \log \pi(y \mid \theta)}{\partial\theta}\right) = -\frac{n}{\theta} + \frac{n}{1-\theta} \\
\Longrightarrow\ & \mathbb{E}\left(\frac{\partial \log \pi(y \mid \theta)}{\partial\theta}\right) = -\frac{n}{\theta(1-\theta)} \\
\Longrightarrow\ & I_\theta(y) \propto \frac{1}{\theta(1-\theta)}.
\end{aligned}
$$

Hence Jeffrey's prior is $\pi(\theta) \propto \theta^{-\frac{1}{2}}(1-\theta)^{-\frac{1}{2}}$. This functional dependency on θ shows that $\theta \sim \text{Beta}(\frac{1}{2}, \frac{1}{2})$.

3.8 Bernstein-von-Mises Theorem

So far, we have considered Bayesian methods in contrast to frequentist ones. The Bernstein-von-Mises theorem is a key theorem linking the two inference methods.

Theorem 3.2 (Bernstein-von-Mises)**.** *For a well-specified model $\pi(y \mid \theta)$ with a fixed number of parameters, and for a smooth prior distribution $\pi(\theta)$ that is non-zero around the MLE $\hat{\theta}$, then*

$$
\left\| \pi(\theta \mid y) - N\left(\hat{\theta}, \frac{I(\hat{\theta})^{-1}}{n}\right) \right\|_{TV} \to 0,
$$

where $||p-q||_{TV}$ is the total variation distance between distributions p and q:

$$
||p-q||_{TV} = \frac{1}{2}\int |\pi(x) - q(x)|\, dx.
$$

The Berstein-von-Mises theorem says that as the number of data points approaches infinity, the posterior distribution tends to a Normal distribution centered around the MLE and variance dependent on the Fisher information. The proof of this theorem is out of the scope of this module, but can be found in Asymptotic Statistics (2000) by A. W. van der Vaart.

3.9 Lab

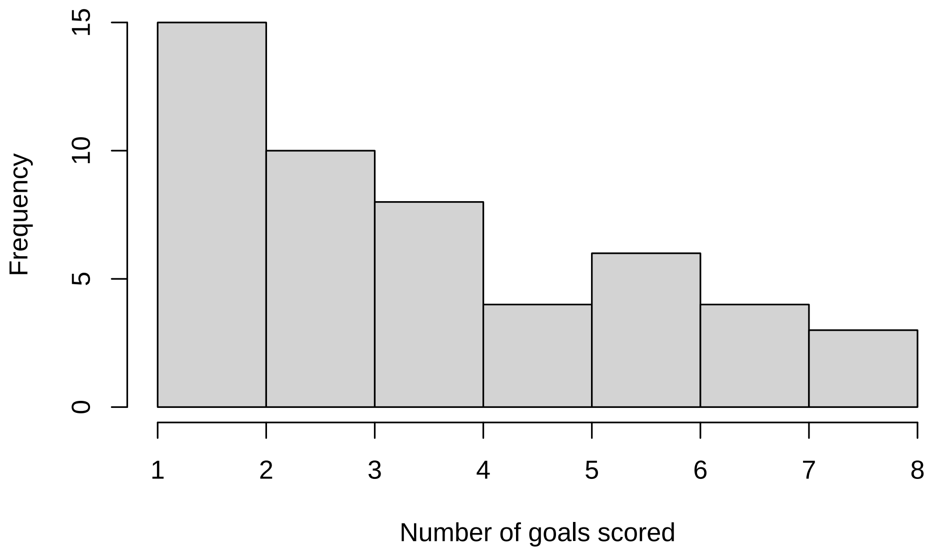
The aim of this lab is to work with some posterior distributions in cases when the prior distribution is or is not conjugate. Recall the definition of a conjugate prior distribution:

If the prior distribution $\pi(\theta)$ has the same distributional family as the posterior distribution $\pi(\theta \mid y)$, then the prior distribution is a **conjugate prior distribution**.

Working with conjugate prior distributions often makes the analytical work much easier, as we can work with the posterior distribution. But sometimes, conjugate prior distributions may not be appropriate. This is where R can help, as we do not need a closed form to carry out computations.

Example 3.9. The total number of goals scored in 50 games of a low level football league is shown below.

```
y <- c(2, 6, 2, 3, 4, 3, 4, 3, 1, 2, 3, 2, 6, 6, 2, 3, 5, 1, 2, 2, 4, 2, 5, 3,
       6, 4, 1, 2, 7, 8, 4, 3, 7, 3, 3, 5, 2, 6, 1, 3, 7, 4, 2, 6, 8, 8, 4, 5,
       7, 4)
hist(y, main = "", xlab = "Number of goals scored")
```



```
mean(y)
```

```
## [1] 3.92
```

We can model the number of goals scored using a Poisson distribution

$$y \sim \text{Po}(\lambda).$$

By Bayes' theorem, the posterior distribution is given by

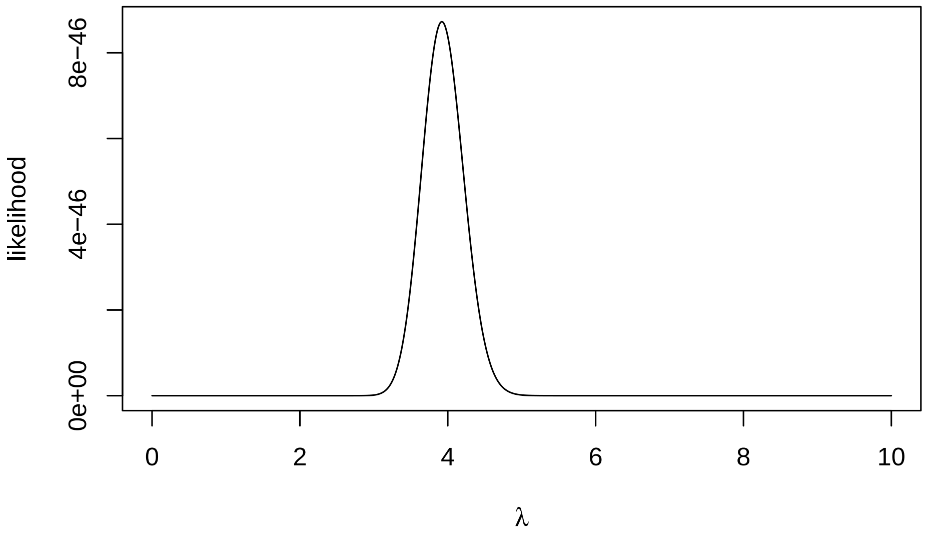
$$\pi(\lambda \mid y) \propto \pi(y \mid \lambda)\pi(\lambda).$$

The likelihood function is given by

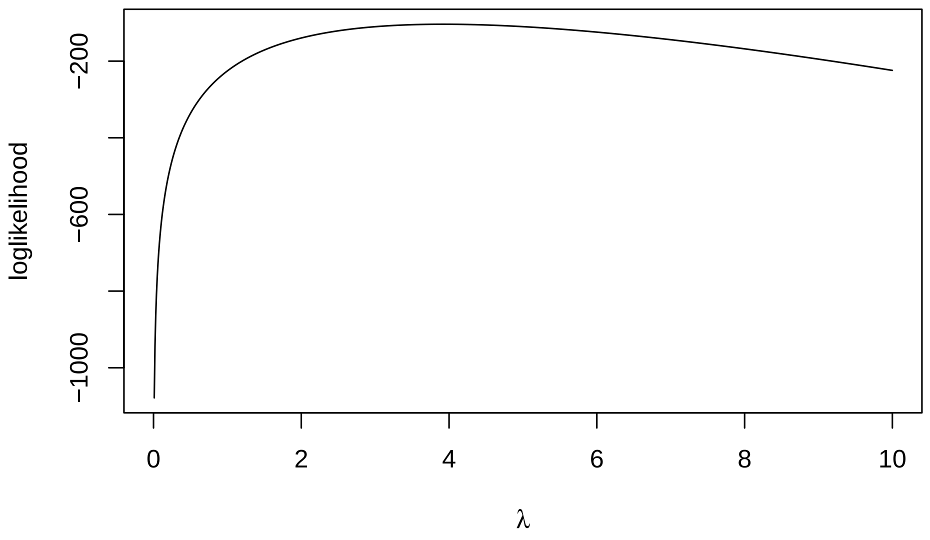
$$\begin{aligned} \pi(y \mid \lambda) &= \prod_{i=1}^{50} \frac{e^{-\lambda}\lambda^{y_i}}{y_i!} \\ &= \frac{e^{-50\lambda}\lambda^{\sum y_i}}{\prod_{i=1}^{50} y_i!} \end{aligned}$$

R has a set of inbuilt functions for working with the Poisson distribution so we can rely on those to write functions for the likelihood and loglikelihood.

```r
lambda <- seq(0, 10, 0.01) #grid of lambda values
likelihood.function <- function(lambda, y) prod(dpois(y, lambda)) #compute likelihood
log.likelihood.function  <- function(lambda, y) sum(dpois(y, lambda, log = TRUE)) #com
likelihood <- sapply(lambda,  likelihood.function, y) #evaluate at grid of points
log.likelihood <- sapply(lambda,  log.likelihood.function, y) #evaluate at grid of poi

#Plot likelihood
plot(lambda, likelihood,
     xlab = expression(lambda), ylab = "likelihood", type = 'l')
```



```
plot(lambda, log.likelihood,
     xlab = expression(lambda), ylab = "loglikelihood", type = 'l')
```



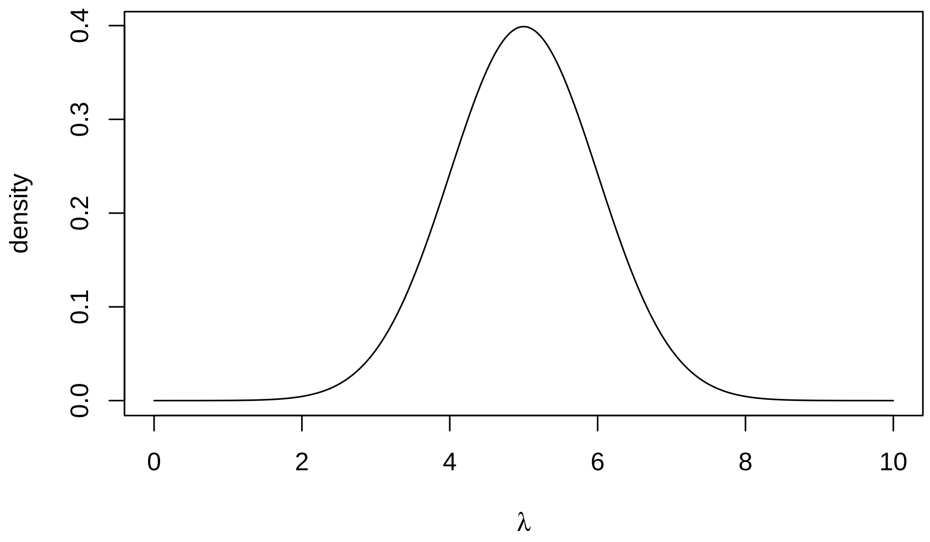
When coding posterior distributions, we often work on the log scale because the numbers can be smaller that R can deal with. The denominator with the factorial can get very large very quickly.

After speaking to football experts, we decide to place a normal prior distribution on λ with mean 5 goals and standard deviation one goal, i.e.

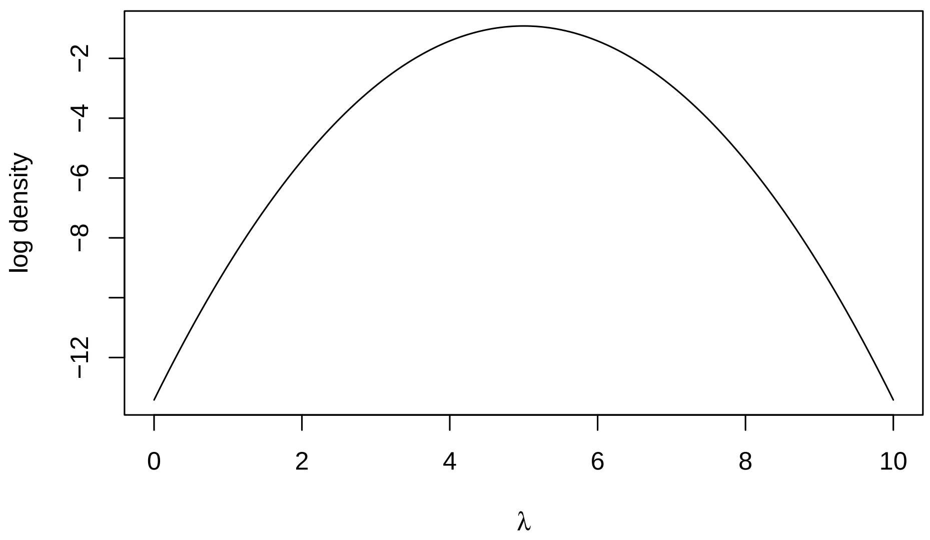
$$\lambda \sim N(5, 1).$$

The prior distribution can be plotted by

```
lambda    <- seq(0, 10, 0.01) #grid of lambda values
prior     <- dnorm(lambda, 5, 1)
log.prior <- dnorm(lambda, 5, 1, log = TRUE)
plot(lambda, prior, type = 'l', xlab = expression(lambda), ylab = "density")
```



```
plot(lambda, log.prior, type = 'l',
     xlab = expression(lambda), ylab = "log density")
```

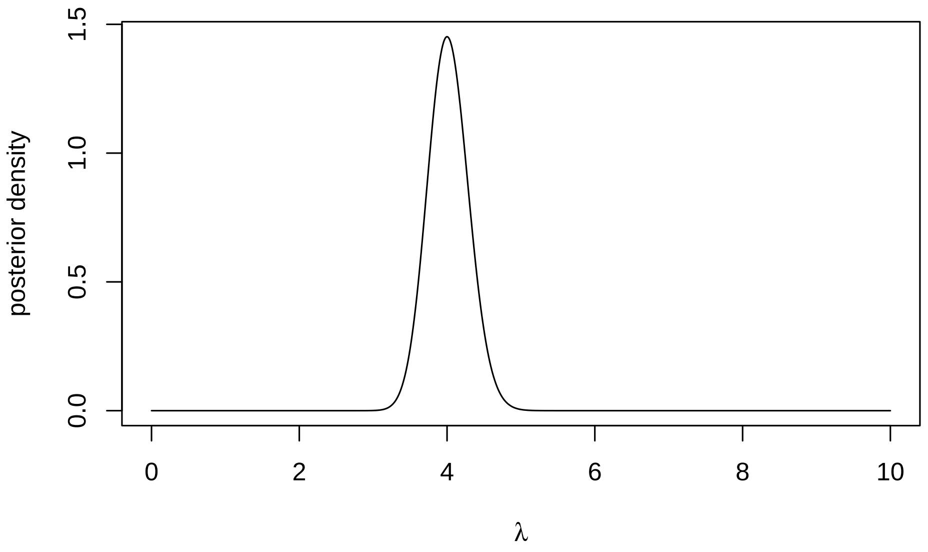


Writing the posterior distribution up to proportionality, we get

$$\pi(\lambda \mid y) \propto \exp\left(-50\lambda - \frac{1}{2}(\lambda - 5)^2\right) \lambda^{\sum y_i}.$$

There is no closed form for this distribution and it is not that nice to work with. But with R, we can easily evaluate the posterior distribution at a grid of points.

```
posterior <- prior*likelihood
integrating.factor <- 0.5*0.01*(posterior[1] + posterior[1001] + 2*sum(posterior[-c(1,
posterior <- posterior/integrating.factor #normalise
plot(lambda, posterior, type = 'l', xlab = expression(lambda),
     ylab = "posterior density")
```



We can now visually inspect the posterior distribution and see that it has a strong peak around 4. One important statistic is the **maximum a posteriori estimation** or MAP estimate, this is the mode of the posterior distribution and it is a similar principle to the maximum likelihood estimate.

We can compute this using the command

```
lambda[which.max(posterior)]
```

```
## [1] 4
```

which shows the MAP estimate is exactly 4.

Exercise 3.1. Adapt the code in the Example above to use an exponential prior distribution with rate 0.1. Then derive the posterior distribution analytically and compare to the numerical version.

Exercise 3.2. You are given that the data are exponentially distributed with rate λ, i.e. $Y_1, \dots, Y_N \sim \text{Exp}(\lambda)$. Your prior belief is that $\lambda \in (0, 1)$. Show that the posterior distribution $\pi(\lambda \mid y)$ has no closed form when the prior distribution for $\lambda \sim \text{Beta}(\alpha, \beta)$.

The data is given by

```
y <- c(1.101558, 1.143953, 1.287348, 1.181010, 1.139132, 1.148631, 1.133201, 1.361229, 1.332540,
```

By writing an R function to evaluate the likelihood function, evaluate the posterior distribution for λ over a grid of points.

Exercise 3.3. Suppose you have $X_1, ..., X_N \sim \text{Bin}(100, p)$. Using $p \sim \text{Beta}(\alpha, \beta)$ as the prior distribution, derive the posterior distribution and the posterior mean (Wikipedia is a helpful place for properties of distributions).

1. (Large data scenario) Fix $\alpha = 2$, $N = 150$ and $\Sigma x_i = 2,971$. Plot the prior and posterior distributions for different values of β on the same figure. Plot the posterior mean against $\beta \in (0, 10)$. Plot the prior mean against the posterior mean for $\beta \in (0, 10)$.
2. (Small data scenario) Fix $\alpha = 2$, $N = 10$ and $\Sigma x_i = 101$ Plot the prior and posterior distributions for different values of β on the same figure.Plot the posterior distribution for different values of β. Plot the posterior mean against $\beta \in (0, 10)$. Plot the prior mean against the posterior mean for $\beta \in (0, 10)$.

Exercise 3.4. Code up the posterior distribution in question 4 of problem sheet 2 (the Pareto distribution). Set $a = 1$, $b = 2$ and let the data be

```
y <- c(1.019844, 1.043574, 1.360953, 1.049228, 1.491926, 1.192943, 1.323738, 1.262572,
```

Find the MAP estimate for β

Chapter 4

Sampling

4.1 Uniform Random Numbers

What we won't be doing in this module is generating true uniform random numbers. This is incredibly difficult and usually requires lots of expensive hardware. This is because computers aren't good at being random, they require algorithmic instructions. True random number generation often uses physical methods, such as the radioactive decay of atoms, or atmospheric noise.

Throughout this module, we will be using R's built in random number generation. This is a pseudo random number generator that has excellent random properties, but will eventually repeat. A basic random number generation tool that we will repeatedly use in the module involves sampling from a uniform distribution on the unit interval, which can be done in R using

```r
runif(1, 0, 1)
```

```
## [1] 0.7769918
```

4.2 Inverse Transform Sampling

Suppose we want to sample from a non-uniform one-dimensional distribution. The inverse transform theorem allows us to do this using the distribution's inverse function.

Definition 4.1. Let X be a real-valued random variable with a distribution function F. Then the **inverse function** of a distribution function F, denoted F^{-1}, is defined for all $u \in (0,1)$ by

$$F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) > u\}.$$

Theorem 4.1. *Let $F : \mathbb{R} \to [0,1]$ be a continuous distribution function, $U \sim U[0,1]$ and $Y = F^{-1}(U)$. Then Y has distribution function F.*

Proof. We have

$$\mathbb{P}(Y \leq a) = \mathbb{P}(F^{-1}(U) \leq a) = \mathbb{P}(\inf\{x \in \mathbb{R} : F(x) > u\} \leq a).$$

Since $\inf\{x \in \mathbb{R} : F(x) > u\} \leq a$ can only hold if $F(a) \geq U$, we have

$$\mathbb{P}(Y \leq a) = \mathbb{P}(F(a) \geq U)$$

As $U \sim U[0,1]$, we have $\mathbb{P}(F(a) \geq U) = F(a)$. □

This theorem says that if we have a random variable $U \sim U[0,1]$ and we want to get $Y \sim F$, then we can use $F^{-1}(U)$. Viewing this theorem graphically can provide a much more intuitive understanding.

Example 4.1. We would like to sample from an exponential distribution with rate λ, i.e. $Y \sim \text{Exp}(\lambda)$. The density function is given by

$$\pi(y \mid \lambda) = \begin{cases} \lambda e^{\lambda y} & y \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The distribution function can be derived by

$$\begin{aligned} F(y \mid \lambda) &= \int_0^y \lambda e^{\lambda t}\, dt \\ &= 1 - e^{\lambda y}. \end{aligned}$$

Finally, the inverse function is given by

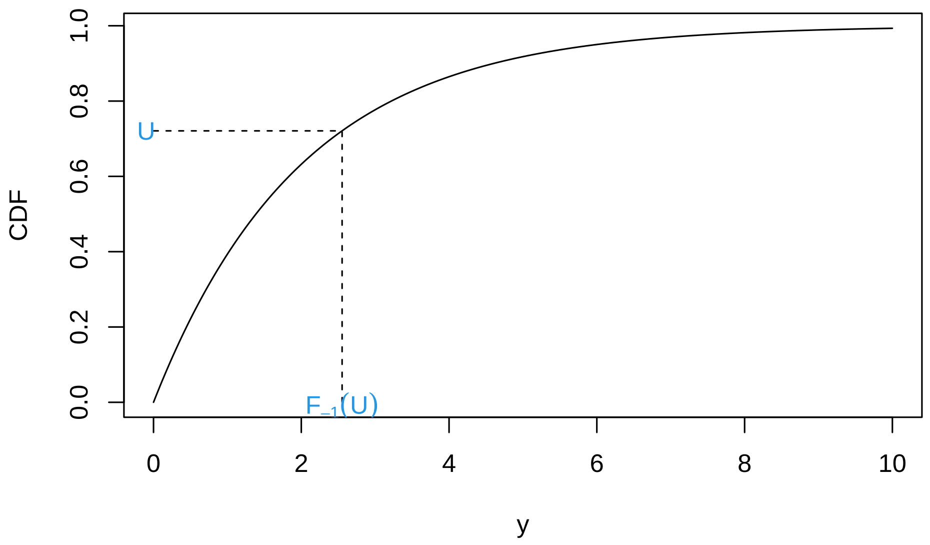
$$F^{-1}(y \mid \lambda) = -\frac{1}{\lambda}\log(1-y).$$

Therefore, if $U \sim U[0,1]$, then it follows that $-\frac{1}{\lambda}log(1-U) \sim \text{Exp}(\lambda)$.

The R code below generates a plot to show this (with $\lambda = 0.5$). We can plot the CDF for most one parameter distributions straightforwardly. We can think of this theorem as allowing us to sample a point on the y-axis and then computing the quantile this corresponds to.

```r
set.seed(12345) # to reproduce
y <- seq(0, 10, 0.01) #Show on the interval [0, 5]
f <- 1 - exp(-0.5*y)     #Construct the cumulative density
                         #function (CDF)
plot(y, f, type ='l', xlab = "y", ylab= "CDF")

#Sample u
```

```
u <- runif(1)

#Get the corresponding y value
f.inv <- -2*log(1-u)

#plot
segments(x0 = 0, y0 = u, x1 = f.inv, y1 = u, lty = 2)
segments(x0 = f.inv, y0 = 0, x1 = f.inv, y1 = u, lty = 2)
text(x = f.inv, y = -0.01, expression(F[-1](U)), col = 4)
text(x = -.1, y = u, "U", col = 4)
```

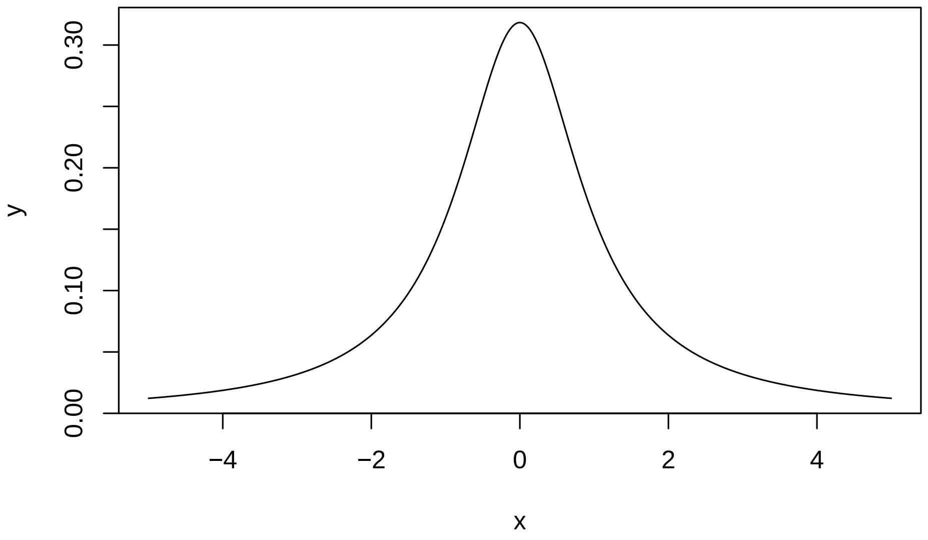


Example 4.2. Suppose we want to generate samples from the Cauchy distribution with location 0 and scale 1. This has density function

$$\pi(x) = \frac{1}{\pi(1+x^2)}, \quad x \in \mathbb{R}.$$

A plot of this function is shown below.

```
x <- seq(-5, 5, 0.01)
y <- 1/(pi*(1 + x^2))
plot(x, y, type = 'l')
```



To use the inverse transform method, we first need to find the CDF:

$$F(x) = \int_{-\infty}^{x} \frac{1}{\pi(1+t^2)} dt$$

Letting $t = \tan\theta$, we can write $dt = \sec^2(\theta)d\theta$. The integral becomes

$$F(x) = \int_{-\frac{\pi}{2}}^{\arctan(x)} \frac{\sec^2(\theta)}{\pi(1+\tan^2(\theta))} d\theta$$

As $1 + \tan^2(\theta) = \sec^2(\theta)$, we can write the integral as

$$F(x) = \int_{-\frac{\pi}{2}}^{\arctan(x)} \frac{1}{\pi} du = \left[\frac{\theta}{\pi}\right]_{-\frac{\pi}{2}}^{\arctan(x)} = \frac{\arctan(x)}{\pi} + \frac{1}{2}.$$

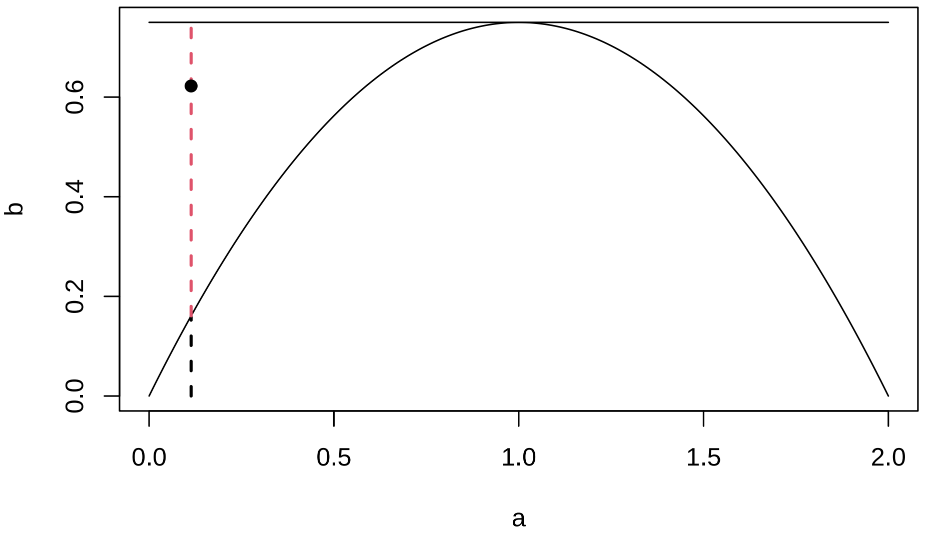
The inverse of the distribution function is

$$F^{-1}(x) = \tan\left(\pi\left(x - \frac{1}{2}\right)\right).$$

Hence, if $U \sim U[0,1]$, then $\tan\left(\pi\left(U - \frac{1}{2}\right)\right) \sim \text{Cauchy}(0,1)$.

4.3 Rejection Sampling

We now have a way of sampling realisations from distributions where we can analytically derive the inverse distribution function. We can use this to sample from more complex densities, or simple densities more efficiently. Rejection sampling works by sampling according to a density we can sample from and then rejecting or accepting that sample based on the density we're actually interested in. The plot below shows an example from this. We would like to generate a sample from the distribution with the curved density function, which is challenging. Instead, we find a distribution whose density function bounds the one we are interested in a sample from that. In this case we can use the uniform distribution. Once we have generated our sample from he uniform distribution, we choose to accept or reject it based on the distribution we are interested in. In this case we reject it.



Suppose we want to sample from a density π, but can only generate samples from a density q. If there exists some constant $c > 0$, such that $\frac{\pi(y)}{q(y)} \leq c$ for all y, then we can generate samples from π by

1. Sampling $Y \sim Q$
2. Sampling $U \sim U[0, 1]$
3. Computing $k = \frac{\pi(u)}{cq(y)}$
4. Accepting y if $U < k$ and rejecting otherwise.

This says draw sample a point y according to the density q. Draw a vertical line at y from the x-axis to $cq(y)$. Sample uniformly on this line. If the uniformly random sample is below q, then accept it. Otherwise, reject it. The theory behind this is as follows. Suppose we sample some point y according to this algorithm and we want to work out its density f, then

$$f(y) \propto q(y)\pi(U < k) = q(y)\frac{\pi(u)}{cq(y)} = \frac{\pi(u)}{c}.$$

Therefore, $f = p$.

Example 4.3. Suppose we want to sample from a distribution that has the density

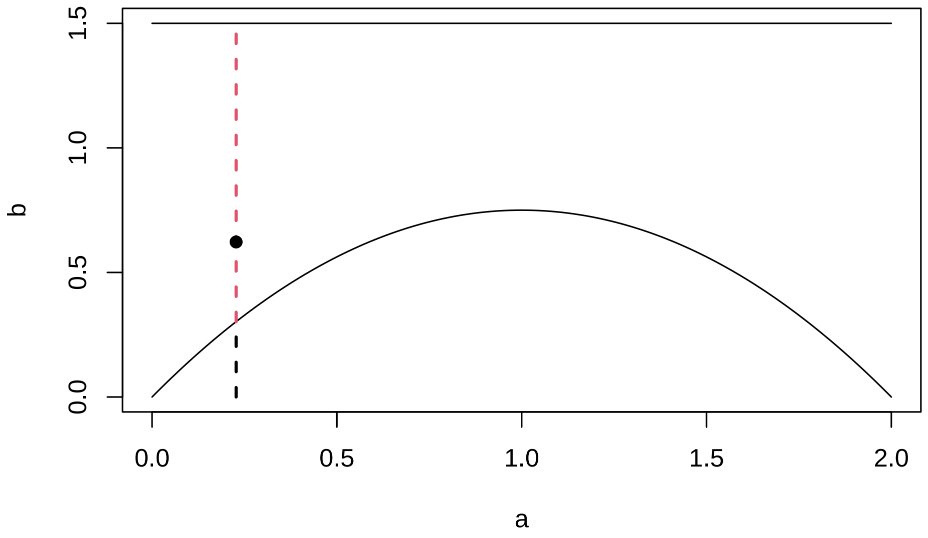
$$\pi(y) = \begin{cases} \frac{3}{4}y(2-y), & y \in [0,2] \\ 0, & \text{otherwise} \end{cases} .$$

This has a maximum at $\frac{3}{4}$. We choose $p \sim U[0,1]$ and $c = \frac{3}{2}$. The R code below shows a pictorial version of how one sample is generated.

```r
set.seed(1234)   #to reproduce
scaling.c <- 3/2          #set c
y <- runif(1, 0, 2)    #sample Y ~ Q
p <- 3/4*y*(2-y) #compute pi(y)
k <- p/(scaling.c*1/2)     #compute k
u <- runif(1)    #sample U ~ U[0, 1]
ifelse(u < k, 'accept', 'reject') #Accept if  u < k
```

```
## [1] "reject"
```

```r
#Create nice plot
a <- seq(0, 2, 0.01)
b <- 3/4*a*(2-a)
scaling.c  <- scaling.c*rep(1, length(a))
plot(a, b, ylim = c(0, 3/2), type = 'l')
lines(a, scaling.c)
segments(x0 = y, y0 = 0, x1 = y,  y1 =3/4*y*(2-y) ,
         lty = 2, lwd = 2)
segments(x0 = y,  y0 =3/4*y*(2-y), x1 = y, y1 = 3/2, lty = 2,
         col = 2, lwd = 2)
points(x = y, y = u, pch = 19)
```



The plot also shows how the choices of c and q can make the sampling more or less efficient. In our example, the rejection space is large, meaning many of our proposed samples will be rejected. Here, we could have chosen a better q to minimise this space.

Example 4.4. Suppose we want to sample from a Beta(4, 8) distribution. This distribution looks like

```
x <- seq(0, 1, 0.001)
y <- dbeta(x, 4, 8)
```

A uniform distribution on [0, 1] will cover the Beta distribution and we can then use a rejection sampling algorithm. First, we need to find c, the maximum of the Beta distribution. We can find this by differentiating the pdf and setting it equal to 0:

$$\pi(x) = \frac{1}{B(4,8)} x^3(1-x)^7 \implies \frac{d\pi(x)}{dx} = \frac{1}{B(4,8)}(3x^2(1-x)^7 - 7x^3(1-x)^6) \implies \frac{d\pi(x)}{dx} = \frac{1}{B(4,8)}(x^2(1-x)^7(3-10x)).$$

Setting this equal to 0 gives us the maximum at $x = \frac{3}{10}$. This means we can set $\pi(3/10) = \frac{1}{B(4,8)} \frac{3^7 7^{10}}{10^{10}}$. Our rejection sampling algorithm is therefore

1. Sample $u \sim U[0,1]$
2. Compute $k = \pi(u)/cq(u)$.
3. Accept u with probability k.

Example 4.5. In this example, we want to sample from a Gamma(3, 2) distribution. The density function is shown below.

```
x <- seq(0, 6, 0.01)
y <- dgamma(x, 3, 2)
```

We will use an Exp(1) distribution as our proposal distribution. To find the value of c, consider $R(x) = \frac{\pi(x)}{q(x)}$

$$R(x) = \frac{\frac{2^3}{\Gamma(3)} x^2 \exp(-2x)}{\exp(-x)} = \frac{2^3}{\Gamma(3)} x^2 \exp(-x)$$

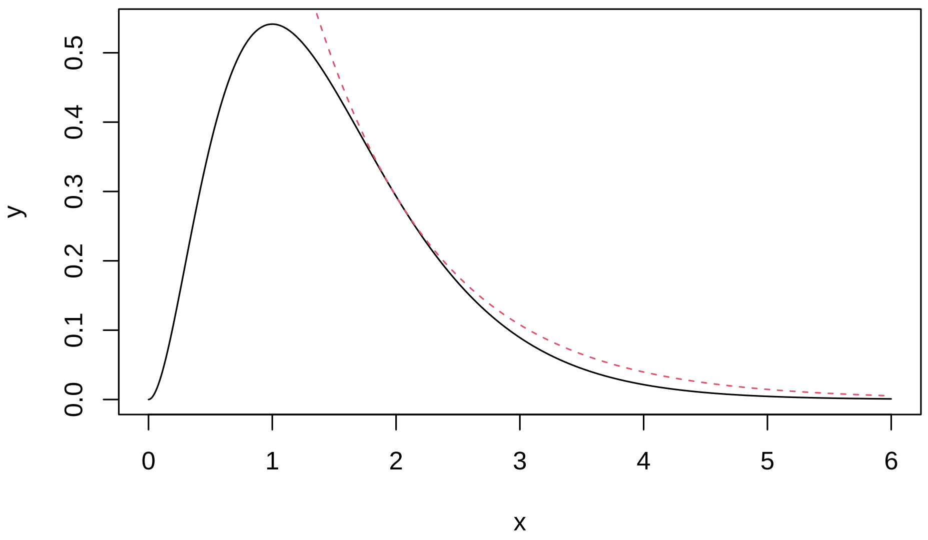
To find the maximum of this ratio, we differentiate and set the result equal to 0.

$$\frac{dR}{dx} = 2x\exp(-x) - x^2\exp(-x) = x\exp(-x)(2-x)$$

The maximum is therefore at 2. The value of the ratio at $x = 2$ is $R(2) = \frac{2^3}{\Gamma(3)} 4\exp(-2)$. This is therefore our value of c. We can see how $\pi(x)$ and $cq(x)$ look of the graph below.

```
x <- seq(0, 6, 0.01)
y <- dgamma(x, 3, 2)
scaling.c <- 2^3/gamma(3)*4*exp(-2)
q <- dexp(x, 1)
```

```r
plot(x, y, type = 'l')
lines(x, q*scaling.c, col = 2, lty = 2)
```



The exponential distribution completely encloses the gamma distribution using this value of c, wiht the two densities just touching at $x = 2$. Our rejection sampling algorithm is therefore

1. Sample $u \sim \exp(1)$
2. Compute $k = \frac{\pi(u)}{cq(u)}$
3. Accept u with probability k.

4.3.1 Rejection Sampling Efficiency

Suppose we are using a rejection sampling algorithm to sample from $f(y)$ using the proposal distribution $g(y)$. How good is our rejection sampling algorithm? What does it mean to be a 'good' rejection sampling algorithm. One measure of the efficiency of a sampler is, on average, how many samples to we need to generate until one is accepted.

Proposition 4.1. *The number of samples proposed in a rejection sampling algorithm before one is accepted is distributed geometrically with mean* $\frac{1}{M}$.

Proof. Given a proposed value y, and let A be the event of sample is accepted. The probability a sample is accepted, given it takes the value y is

$$A \mid y \sim \text{Bernoulli}\left(\frac{f(y)}{Mg(y)}\right).$$

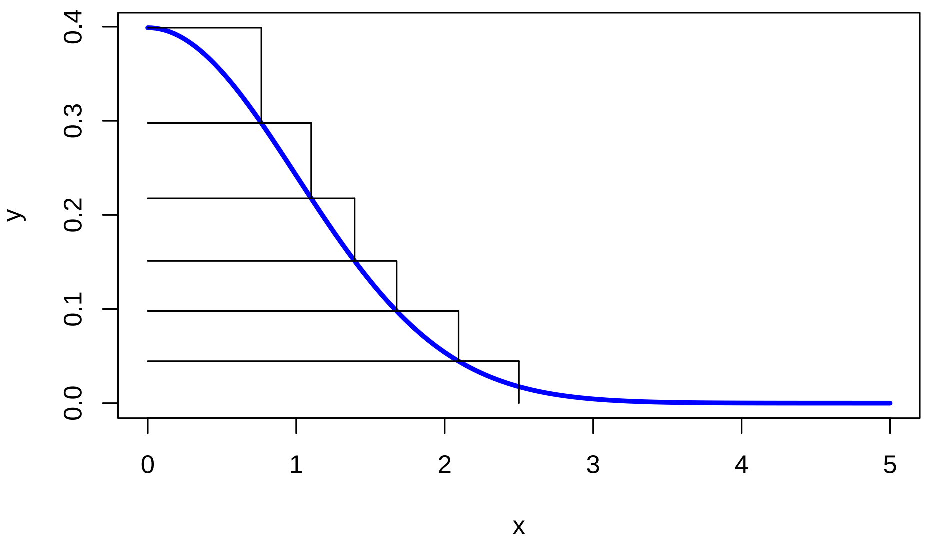
By the tower property

$$\begin{aligned} E(A) &= E(E(A \mid y)) \\ &= E\left(\frac{f(y)}{Mg(y)}\right) \\ &= \int \frac{f(y)}{Mg(y)} g(y)dy \\ &= \frac{1}{M}\int f(y)dy \\ &= \frac{1}{M}. \end{aligned}$$

Therefore, the number of samples proposed before one is accepted is distributed geometrically with mean $\frac{1}{M}$. $\square$

4.4 Ziggurat Sampling

The final method we are going to look at in this chapter is Ziggurat sampling. It is used to sample from a $N(0,1)$ distribution using samples from a $U(0,1)$ distribution. A Ziggurat is a kind of stepped pyramid. To start the sampling algorithm, we approximate the normal distribution by a series of horizontal rectangles, each with the same area. An example of this is shown below.



To generate a sample, we choose a rectangle at random – we might do this by sampling on the y-axis uniformly at random. We then sample uniformly at random within the rectangle. We accept the sample if it is ‘clearly’ in the box. By clearly, we mean that is is also contained in the rectangle above. If it is, not contained in the rectangle above, we generate a new uniform random number to accept or reject the sample.

Suppose the bottom right corner of the i^{th} rectangle has coordinates (x_i, y_i). The Ziggurat algorithm is

1. Sample a rectangle i uniformly at random
2. Sample $u_0 \sim U[0,1]$ and set $x = u_0 x_i$
3. If $x < x_{i+1}$ accept the sample
4. Sample $u_1 \sim U[0,1]$ and set $y = y_i + u_1(y_{i+1} - y_i)$
5. If $y < f(y)$, where f is the density function of a standard normal distribution, then accept the sample.

This only generates samples for the positive side of the normal distribution. To generate samples from the full distribution, we multiply each sample by -1 with probability a half.

4.5 Lab

The aim of this lab is to code up some sampling methods. You have already done some similar work in lab 1 (e.g. estimating π).

Exercise 4.1. Visit random.org to generate some truly random numbers. How does this site generate numbers that are truly random?

Exercise 4.2. A random variable X has density $\pi(x) = ax^2$ for $x \in [0,1]$ and 0 otherwise. Find the value a. Use the inverse transform method to generate 10,000 samples from this distribution. Plot a histogram against the true density function.

Exercise 4.3. Let X have the density $\pi(x) = \frac{1}{\theta} x^{\frac{1-\theta}{\theta}}$ for $x \in [0,1]$. Use the inverse transform method to generate 10,000 samples from this distribution with $\theta = \{1, 5, 10\}$.

Exercise 4.4. Let X have the density $\pi(x) = 3x^2$ for $x \in [0,1]$ and 0 otherwise. Use a rejection sampling method to generate 10,000 from this distribution. Plot the results against the true density to check you have the same distribution.

Exercise 4.5. The half normal distribution with mean 0 and variance 1 has density function

$$\pi(x) = \frac{2}{\sqrt{2\pi}} \exp\left(-x^2/2\right)$$

for $x \geq 0$ and 0 otherwise.

1. Denote the exponential density function by $q(x) = \lambda \exp(-\lambda x)$. Find the smallest c such that $\pi(x)/q(x) < c$.
2. Use a rejection sampling algorithm with an exponential proposal distribution to generate samples from the half normal distibrution with mean 0 and variance 1.

Chapter 5

Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a class of algorithms that produce samples from a probability distribution. These methods combine the idea of rejection sampling with the theory of Markov chains. Before we set out the theory of Markov chains, we'll go through an example to show how MCMC works.

Example 5.1. (Adapted from Statistical Rethinking §9) Consider an eccentric King whose kingdom consists of a ring of 10 islands. Directly north is island one, the smallest island. Going clockwise around the archipelago, next is island two, which is twice the size of island one, then island three, which is three times as large as island one. Finally, island 10 is next to island one and ten times as large.

The King wanted to visit all of his islands, but spending time on each one according to its size. That is he should spend the most time on island ten and the least on island one. Being climate conscious, he also decided that flying from one side of the archipelago to the other was not allowed. Instead, he would only sail from one island to either of its neighbors. So from island one, he could reach islands two and ten.

He decided to travel according to these rules:

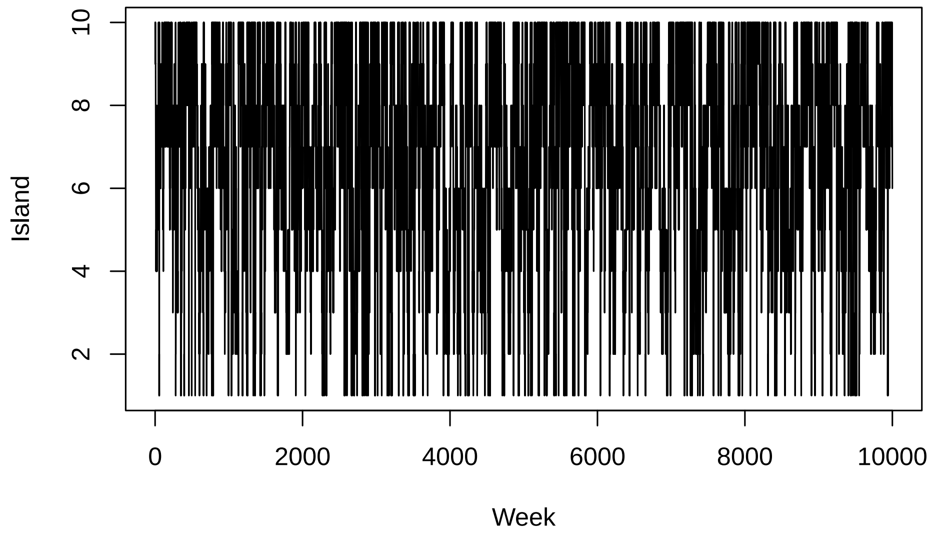
1. At the end of each week, he decides to stay on the same island or move to a neighboring island according to a coin toss. If it's heads he proposes moving clockwise, and tails anti-clockwise. The island he is considering moving to is called the proposal island.

2. To decided if he is going to move to the proposal island, the King counts out a number of shells equal to the number of size of the island. So if island five is the proposal island, he counts out five shells. He then counts out a number of stones equal to the size of the current island.

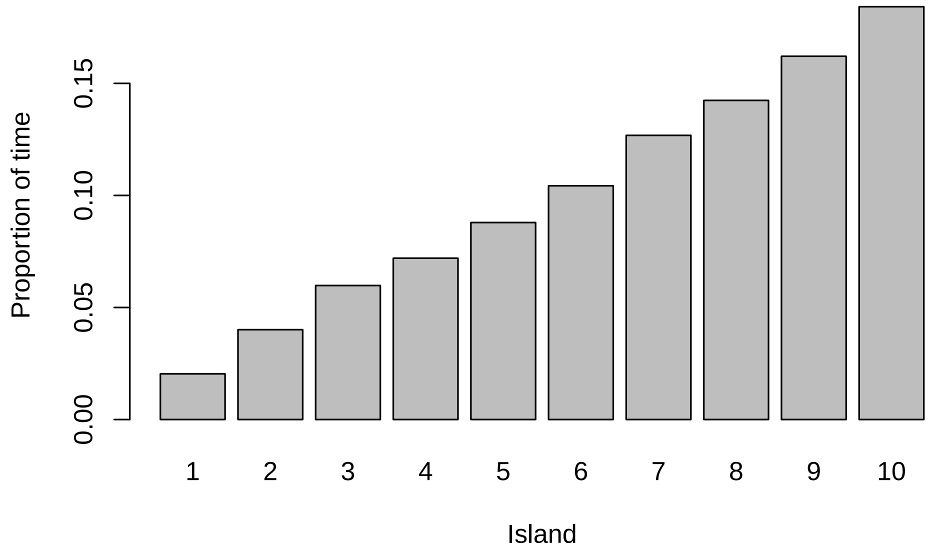
3. If the number of seashells is greater than the number of stones, he moves to the proposed island. If the number of seashells is less than the number of stones, he takes a different strategy. He discards the number of stones equal to the number of seashells. So if there are six stone and five seashells, he ends up with 6-5=1 stones. He then places the stones and seashells into a bag a chooses one at random. If he picks a seashell, he moves to the proposed island, otherwise if he picks a shell, he stays put.

This is a complex way of moving around, but it produces the required result; the time he spends on each island is proportionate to the size of the island. The code below shows an example of this over 10,000 weeks.

```r
weeks <- 10000
island <- numeric(weeks)
current <- 10
for(i in 1:weeks){
  ## record current position
  island[i] <- current

  #Flip a coin to move to a propose a new island
  proposed <- current + sample(c(1, -1), size = 1)

  #Ensure he loops round the island
  if(proposed < 1)
    proposed <- 10
  if(proposed > 10)
    proposed <- 1

  #Decide to move
  p <- proposed/current
  u <- runif(1)
  if(u < p)
    current <- proposed
}

#Plot results
plot(island, type = 'l', xlab = "Week", ylab = "Island")
```



```r
barplot(table(island)/weeks, xlab = "Island",
        ylab = "Proportion of time")
```



We can recognise several different statistical principles in this example. The King decides to move islands dependent on where he is currently, not based on where he has been previously (Markov property). He proposes an island to move to and accepts or rejects this decision based on some distribution (rejection principle). We are now going to describe some of the properties of Markov chains, including the Markov property.

5.1 Properties of Markov Chains

Definition 5.1. A sequence of random variables $\{Y_1, Y_2, ...\}$ is a **Markov chain** if $\mathbb{P}(Y_{n+1} \mid Y_n, \dots, Y_1) = \mathbb{P}(Y_{n+1} \mid Y_n)$. That is that distribution of the next state Y_{n+1} only depends on the current state Y_n and not any previous states.

Definition 5.2. The probability of transitioning from state i to state j in a Markov chain is given by p_{ij}. The **transition matrix** for a Markov chain with N states is the $N \times N$ matrix P, where the $\{i, j\}^{th}$ entry denoted by p_{ij} is probability is moving from state i to state j.

These two properties make Markov chains nice to work with, especially the Markov property (Definition 5.1). Two other important definitions are

Definition 5.3. The **period** of a state i is given by $d_i = \gcd\{n > 0; p_{ii} > 0\}$. A state is **aperiodic** if $d_i = 1$. An **aperiodic chain** is a chain where all states are a periodic.

Definition 5.4. A Markov chain is **irreducible** if there exists an $n \in \mathbb{N}$ such that $\mathbb{P}(Y_n = i \mid Y_0 = j)$ for all pairs i and j. In other words, it is possible to move from any state to any other state in a finite number of steps.

We can use these definitions to start working with distributions. Suppose, the state we start at is drawn from some distribution $Y_1 \sim q$. Then the distributions of the second state Y_2 depends on the distribution of Y_1 and the transition probabilities

$$\mathbb{P}(Y_2 = j) = \sum_i q_i p_{ij}.$$

If we denote the distribution of $Y_2 \sim q^{(2)}$, then we can write it in terms of the transition matrix $q' = qP$. Now suppose we would like the distribution of $Y_3 \sim q^{(3)}$, thanks to the Markov property, this is the distribution for Y_2 multiplied by the transition matrix, so $Y_3 \sim qP^2$. Inductively, $P_k \sim qP^{k-1}$. To use Markov chains to sample from distributions, we need to identify the Eigenvalues of the transition matrix.

Proposition 5.1. *A transition matrix P always has at least one eigenvalue equal to one.*

Proof. The columns of P sum to 1 as they are probability distributions. Therefore, 1 is an eigenvalue. $\square$

Definition 5.5. If a transition matrix P has a unique Eigenvalue that takes the value 1, there is a unique distribution π such that

$$\pi P = \pi.$$

This distribution π, is known as the **stationary distribution**.

This important concept underpins MCMC methods. It says that no matter where we start our chain, we'll eventually end up sampling states according to the distribution π. It make take a long time to reach the stationary distribution, but it will eventually get there.

In order to check whether our Markov chain will converge to a stationary distribution, we need to check:

1. the Markov chain is aperiodic,
2. the Markov chain is irreducible, and
3. that there exists a unique distribution π such that $\pi P = \pi$.

Example 5.2. In Example 5.1, the King wanted to visit the islands according to how large they are. We can think of the islands as the states and the stationary distribution as $p(Y = i) \propto i$. The eccentric method the King used allowed him to construct a transition matrix for an aperiodic Markov chain. He also never visited islands regularly using this method.

When designing a Markov chain, it is usually straightforward to design one that meets conditions one and two. Condition three is more difficult to prove, but for some chains it is possible to show they satisfy detailed balance.

Definition 5.6. The Markov chain with transition matrix P satisfies the **detailed balance** equation with respect to the distribution π if

$$\pi_i p_{ij} = \pi_j p_{ji}.$$

Theorem 5.1 (Detailed Balance)**.** *Let P be a transition matrix that satisfies detailed balance with respect to the distribution π. Then $\pi P = \pi$.*

Proof. The j^{th} row of πP is

$$\begin{aligned}\sum_i \pi_i p_{ij} &= \sum_i \pi_j p_{ji} \quad \text{(detailed balance)} \\ &= \pi_j \sum_i p_{ji} \\ &= \pi_j. \qquad \text{(probaility sums to 1)}\end{aligned}$$

Hence $\pi P = \pi$. □

The section has shown us that we can use a Markov chain theory to simulate from a probability distribution π. All we need is for the Markov chain to be irreducible, aperiodic, and for the transition matrix to satisfy $\pi P = \pi$. This provides the foundation theory for MCMC and allows us to sample from a posterior distribution π. What it doesn't tell us is how to design the Markov chain, and that is what the next sections deal with.

5.2 Metropolis-Hastings

We're now going to look at MCMC algorithms. The first algorithm we are going to look at is the Metropolis-Hasting algorithm. This is a useful algorithm if we cannot sample directly from the posterior distribution and if the conditional distributions do not have a closed form. The Metropolis-Hastings algorithm is like the island example we saw earlier. At each iteration, we propose a new sample and then accept or reject it based on the likelihood function, the prior and how likely we are to propose this new sample given the current one.

Suppose we want to sample from the posterior distribution $\pi(\theta \mid y)$. The Metropolis-Hastings works as follows:

1. Set the initial value $\theta^{(0)}$.
2. Set $i = 1$.
3. Propose a new value of θ' from some distribution q
4. Accept θ' with probability
$$p_{\text{acc}} = \min\left\{\frac{\pi(\theta' \mid y)}{\pi(\theta \mid y)}\frac{q(\theta \mid \theta')}{q(\theta' \mid \theta)}, 1\right\}.$$
5. Repeat steps 3 to 4 for $i = 2, \dots, M$.

There are two parts to the acceptance probability in step 4. The first is the posterior ratio, similar to saying the likelihood of θ' given the observed data over the likelihood of θ given the data. The second is the proposal ratio. It is similar to saying the likelihood of proposing θ given the current value θ', over the likelihood of proposing θ' given the current value θ.

In practice, we don't need to evaluate the full posterior distribution. Recall
$$\pi(\theta \mid y) = \frac{\pi(y \mid \theta)\pi(\theta)}{\pi(y)}$$
As the the denominator doesn't depend on θ, it cancels in the ration. The ratio becomes
$$\frac{\pi(\theta' \mid y)}{\pi(\theta \mid y)} = \frac{\pi(y \mid \theta')\pi(\theta')}{\pi(y \mid \theta)\pi(\theta)}.$$
This is the likelihood ratio multiplied by the prior ratio.

Proposition 5.2. *The Markov chain generated by the Metropolis-Hastings algorithm satisfies detailed balance with respect to the posterior distribution.*

Proof. Denote the current state θ and the proposed state θ'. We would like to show
$$\pi(\theta \mid y)\pi(\theta' \mid \theta) = \pi(\theta' \mid y)\pi(\theta \mid \theta').$$

The density of θ' given the proposed state θ is the proposal density multiplied by the acceptance probability. It is given by

$$\begin{aligned}\pi(\theta' \mid \theta) &= q(\theta' \mid \theta)p_{acc} \\ &= q(\theta' \mid \theta)\min\left\{\frac{\pi(\theta' \mid y)}{\pi(\theta' \mid y)}\frac{q(\theta \mid \theta')}{q(\theta' \mid \theta)}, 1\right\} \\ &= \min\left\{\frac{\pi(\theta' \mid y)}{\pi(\theta' \mid y)}q(\theta \mid \theta'), \, q(\theta' \mid \theta)\right\}.\end{aligned}$$

The left hand side of the detailed balance equation becomes

$$\pi(\theta \mid y)\pi(\theta' \mid \theta) = \min\{\pi(\theta' \mid y)q(\theta \mid \theta'), \, \pi(\theta \mid y)q(\theta' \mid \theta)\}.$$

Analogously, we can show the right hand side is

$$\pi(\theta' \mid y)\pi(\theta \mid \theta') = \min\{\pi(\theta' \mid y)q(\theta \mid \theta'), \, \pi(\theta \mid y)q(\theta' \mid \theta)\}.$$

Hence, $\pi(\theta \mid y)\pi(\theta' \mid \theta) = \pi(\theta' \mid y)\pi(\theta \mid \theta')$ and the Markov chain satisfies detailed balance with respect to the posterior disquisition. □

Example 5.3. A counter monitors the time until atoms decays. It collects the data $X_1, \dots, X_N$ and we assume $X_i \sim \text{Exp}(\lambda)$. The time until these atom decays is short, less than 1 second, so we assume $\lambda \sim \text{Beta}(\alpha, \beta)$.

The prior distribution is given by

$$\pi(\lambda \mid x) \propto \lambda^{N+\alpha-1}(1-\lambda)^{\beta-1}\exp\left(-\lambda \sum x_i\right)$$

This doesn't have a closed form, so we need to use a Metropolis-Hastings algorithm to generate samples from this distribution. A suitable algorithm is

1. Decide on an starting value the Markov chain and denote it $\lambda^{(1)}$. Set $i = 1$.
2. Propose a new value for the parameter and denote it λ'. In this example, we are going to propose values using a random walk method where $\lambda' \sim N(\lambda^{(i)}, \sigma^2)$.
3. Accept λ' as the value $\lambda^{(i+1)}$ with probability p_{acc}. Otherwise reject this value and set $\lambda^{(i+1)} = \lambda^{(i)}$.
4. Set $i = i + 1$.
5. Repeat steps 2, 3, 4, for $i = 2, \dots, M$.

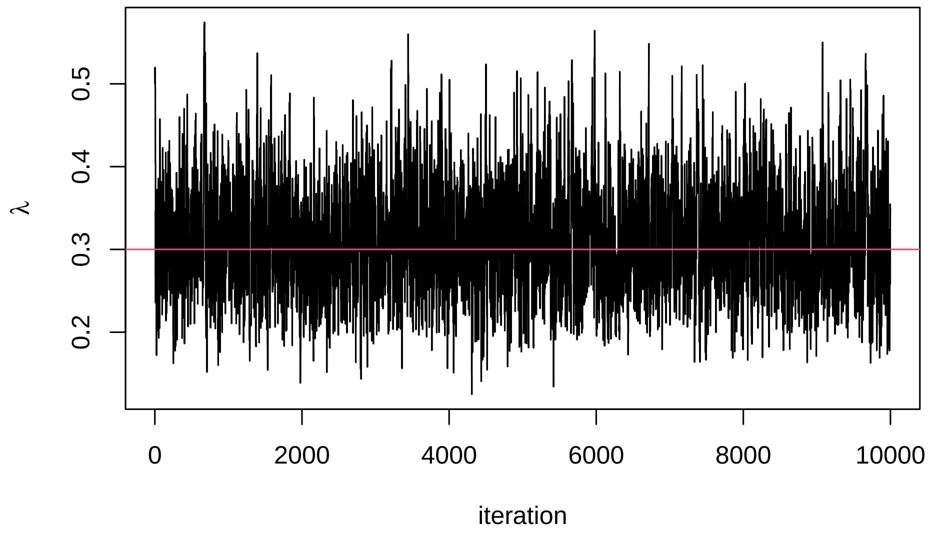
The value p_{acc} is given by

$$p_{\text{acc}} = \min\left\{1, \, \frac{\lambda'^{N+\alpha-1}(1-\lambda')^{\beta-1}\exp\left(-\lambda' \sum x_i\right)}{\lambda^{N+\alpha-1}(1-\lambda)^{\beta-1}\exp\left(-\lambda \sum x_i\right)}\right\} = \min\left\{1, \, \left(\frac{\lambda'}{\lambda}\right)^{N+\alpha-1}\left(\frac{1-\lambda'}{1-\lambda}\right)^{\beta-1}\exp\left((\lambda-\lambda')\sum x\right)\right\}$$

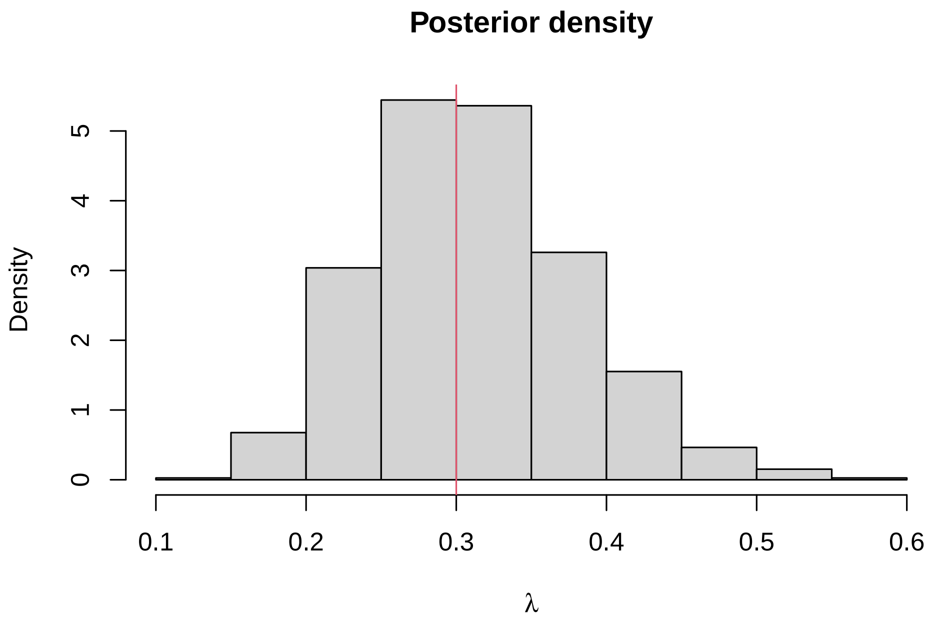
As we our proposal distribution is symmetric, $q(\lambda \mid \lambda') = q(\lambda' \mid \lambda)$ and this term cancels.

The code below shows this algorithm in action

```
# Set Up MCMC Algorithm ---------------------------------------------------

n.iter <- 10000
lambda.store <- numeric(n.iter) #Store value of Markov chain at end of every iteration

#Initialise Prior Parameters and Data
lambda <- 0.5
sum.x <- 67.6
N <- 20
alpha <- 1
beta <- 1


# Run MCMC Algorithm ------------------------------------------------------
for(i in 1:n.iter){

  #Propose new value of lambda
  lambda.prop <- rnorm(1, lambda, 0.1)

  #Check lambda \in [0, 1]
  if(lambda.prop > 0 & lambda.prop < 1){

    #Compute p_acc
    log.p.acc <- (N + alpha - 1)*log(lambda.prop/lambda) +
      (beta - 1)*log((1-lambda.prop)/(1-lambda)) +
      (lambda - lambda.prop)*sum.x

    #Accept/Reject step
    if(log(runif(1)) < log.p.acc)
      lambda <- lambda.prop

  }

  #Store current value of Markov Chain
  lambda.store[i] <- lambda

}

#Plot trace plot (Markov chain values)
plot(lambda.store, type = 'l', xlab = "iteration", ylab = expression(lambda))
abline(h=0.3, col = 2) #the value I used to simulate the data
```



```
#Plot posterior density
hist(lambda.store, prob = TRUE, xlab = expression(lambda), main = "Posterior density")
abline(v=0.3, col = 2) #the value I used to simulate the data
```



```
mean(lambda.store) #posterior mean
```

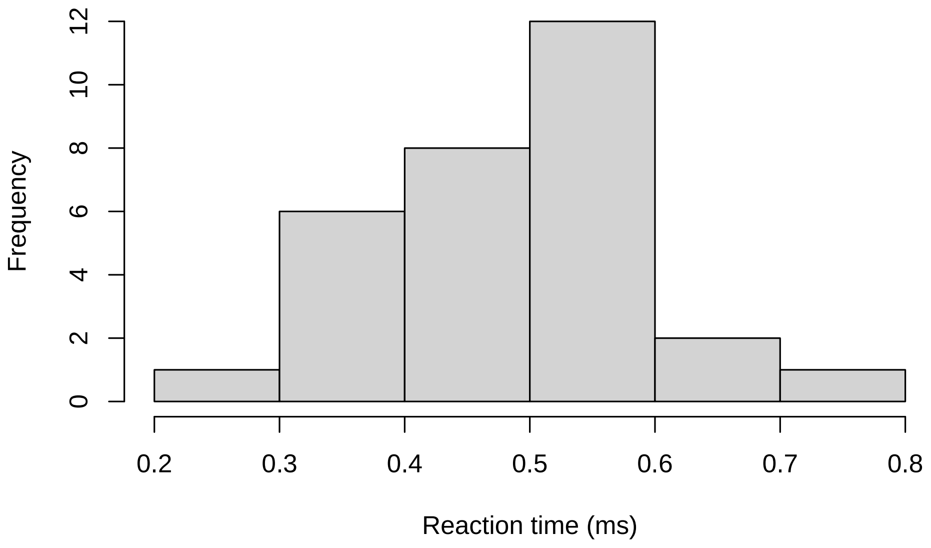
```
## [1] 0.3116649
```

```r
quantile(lambda.store, c(0.025, 0.975)) #95% CI
```

```
##      2.5%     97.5%
## 0.1937659 0.4582714
```

Example 5.4. The time until lorry drivers react (in milliseconds) to an obsticle in the road is

```r
y <- c(0.34, 0.47, 0.58, 0.27, 0.74, 0.44, 0.46, 0.65, 0.36, 0.55, 0.58, 0.55,
       0.53, 0.56, 0.54, 0.61, 0.43, 0.52, 0.45, 0.49, 0.32, 0.33, 0.47, 0.58,
       0.34, 0.60, 0.59, 0.43, 0.57, 0.34)
hist(y, main = "", xlab = "Reaction time (ms)")
```



Assuming $Y_i \sim N(\mu, \sigma^2)$ are independent and identically distributed for $i = 1, ..., n$, by Bayes' theorem, the posterior distribution is

$$\pi(\mu \mid y, \sigma^2) \propto \pi(y \mid \mu, \sigma^2)\pi(\mu).$$

One of the issues here is that we have assigned a normal prior distribution to the population mean parameter μ. The advantage previously was that we could derive a posterior distribution with closed form. The disadvantage however is that the choice of prior distribution assigns some positive probability to impossible values of μ, i.e. reaction times less than zero.

Now we have a tool to sample from posterior distributions that don't have a closed form. We can instead assign an exponential prior distribution, a distribution which only has non-negative support. Letting $\mu \sim \text{Exp}(0.01)$ sets a vague prior distribution on μ. It can be shown that the posterior distribution (exercise) is therefore

$$\pi(\mu \mid y, \sigma^2) \propto \exp\left\{-0.01\mu - \sum_{i=1}^{30} \frac{(y_i - \mu)^2}{\sigma^2}\right\}$$

We can use the Metropolis-Hasting algorithm to sample from this posterior distribution. But how should we propose new value of μ? A common method is a Metropolis-Hastings Random Walk proposal distribution. The proposal distribution is symmetric and centered on μ. The two most common methods are $\mu' \mid \mu \sim U[\mu - \varepsilon, \mu + \varepsilon]$ and $\mu' \mid \mu \sim N(\mu, \tau^2)$. We choose the uniform proposal distribution, with

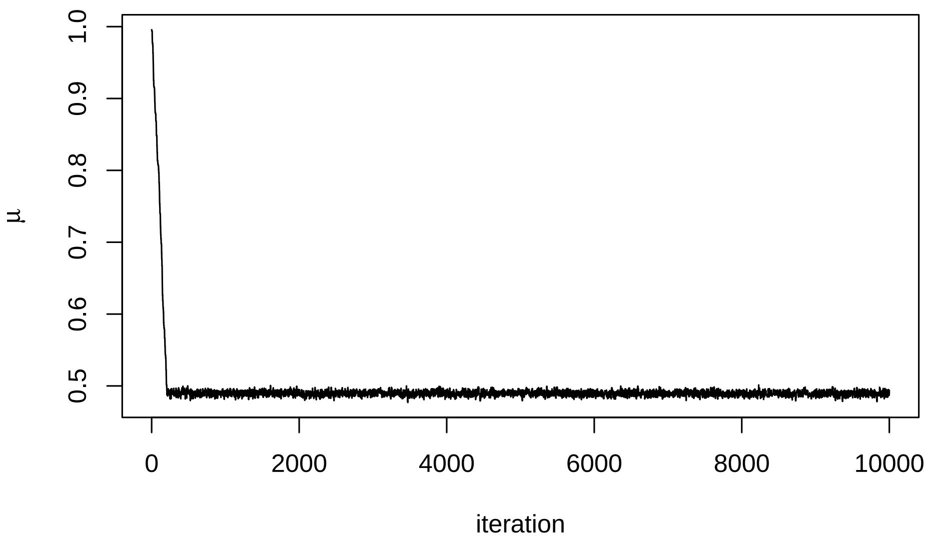
$$q(\mu' \mid \mu) = \frac{1}{2\varepsilon}.$$

The acceptance probability is therefore

$$p_{\text{acc}} = \min\left\{\frac{\exp\left\{-0.01\mu' - \sum_{i=1}^{30}\frac{(y_i-\mu')^2}{\sigma^2}\right\}}{\exp\left\{-0.01\mu - \sum_{i=1}^{30}\frac{(y_i-\mu)^2}{\sigma^2}\right\}}, 1\right\}$$

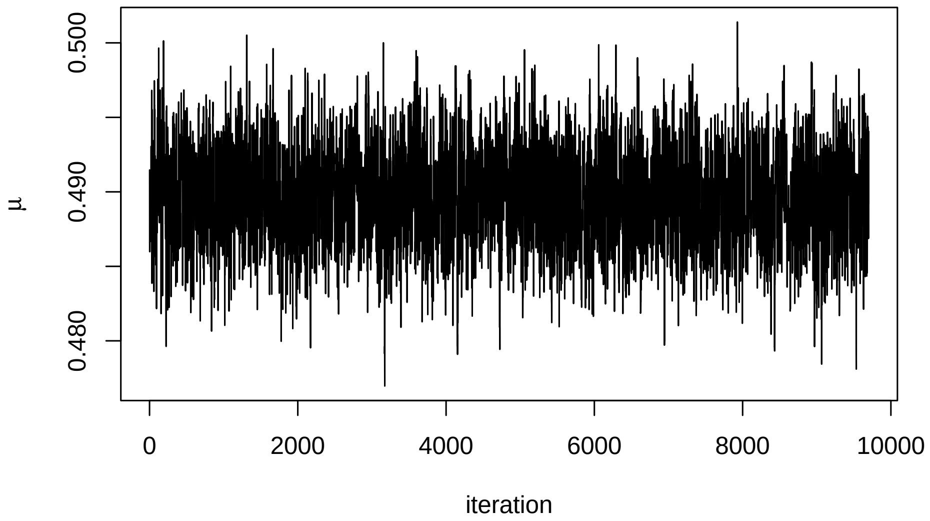
We can implement a sampler in R as follows:

```r
#Set up elements for MCMC
set.seed(123) #to reproduce
n.iter   <- 10000
mu.store <- numeric(n.iter)

#Initial values
mu <- 1
sigma <- 0.1 #known

for(i in 1:n.iter){

  #Propose value for mu
  mu.proposed <- runif(1, mu - 0.01, mu + 0.01)

  if(mu.proposed > 0){ #If mu < 0 we can reject straight away

    #Compute (log) acceptance probability
    log.numerator   <- -0.01*mu.proposed -
                       sum(y - mu.proposed)^2/(2*sigma^2)
    log.denominator <- -0.01*mu - sum(y - mu)^2/(2*sigma^2)

    log.p.acc <- log.numerator - log.denominator
    u <- runif(1)
    #Accept/Reject step
    if(log(u) < log.p.acc){
      mu <- mu.proposed
    }
  }

  #Store mu at each iteration
  mu.store[i] <- mu
}
plot(mu.store, type = 'l', xlab = "iteration",
     ylab = expression(mu))
```

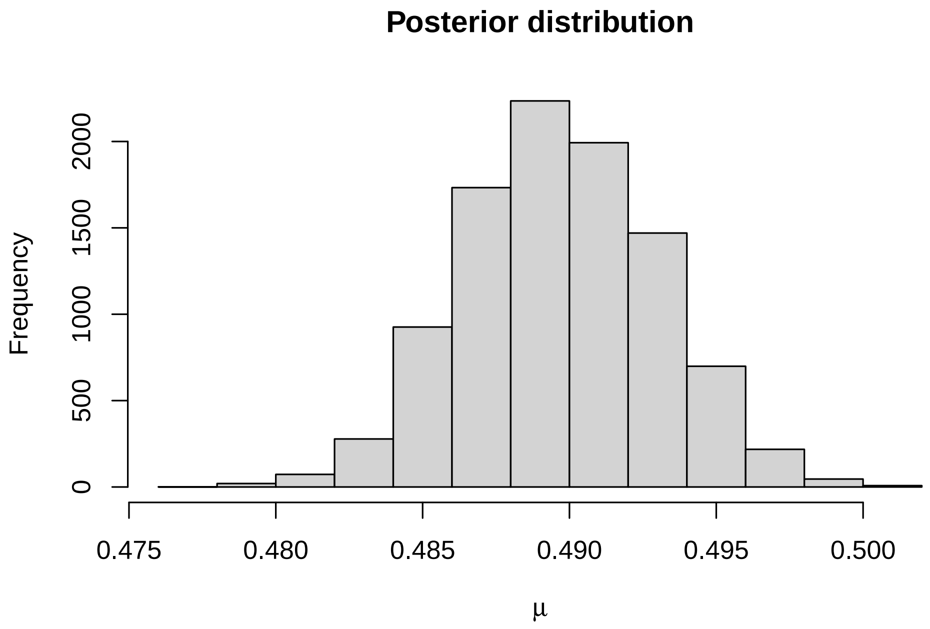


We can see that after about 300 iterations, the Markov chain has converged to its stationary distribution, the posterior distribution. We can see this more clearly by removing the first 300 iterations.

```
plot(mu.store[-c(1:300)], type = 'l', xlab = "iteration", ylab = expression(mu))
```



```r
hist(mu.store[-c(1:300)], xlab = expression(mu), main = "Posterior distribution")
```



The 95% credible interval for μ using this prior distribution is

```r
quantile(mu.store[-c(1:300)], c(0.025, 0.975))
```

```
##      2.5%     97.5% 
## 0.4832146 0.4961484
```

Using a normal prior distribution, it was

```
0.486 0.493
```

It seems that the posterior distribution is very similar when using these two prior distributions. This is because the data are very informative.

Example 5.5. In Lab 3.9, we computed the MAP estimate for a parameter from the Pareto distribution. The density function of this distribution is

$$\pi(x \mid \beta) = \frac{\beta}{x^{\beta+1}}, \quad x > 1.$$

Placing a Gamma prior distribution on β such that $\beta \sim \Gamma(a, b)$. The posterior distribution given the data $y = \{y_1, \ldots, y_N\}$ is

$$\pi(\beta \mid y) \propto \frac{\beta^{N+a-1}e^{-b\beta}}{\prod y_i^{\beta+1}}.$$

We can't sample from this directly, so wee need to use a Metropolis-Hastings algorithm to generate samples from the posterior distribution. We will use a normal proposal distribution.

The acceptance probability is

$$p_{acc} = \min\left\{1, \frac{\beta'^{N+a-1}e^{-b\beta'}}{\prod y_i^{\beta'+1}} \frac{\prod y_i^{\beta+1}}{\beta^{N+a-1}e^{-b\beta}}\right\} = \min\left\{1, \left(\frac{\beta'}{\beta}\right)^{N+a-1} \prod y_i^{\beta-\beta'} \exp((\beta-\beta')b)\right\}.$$

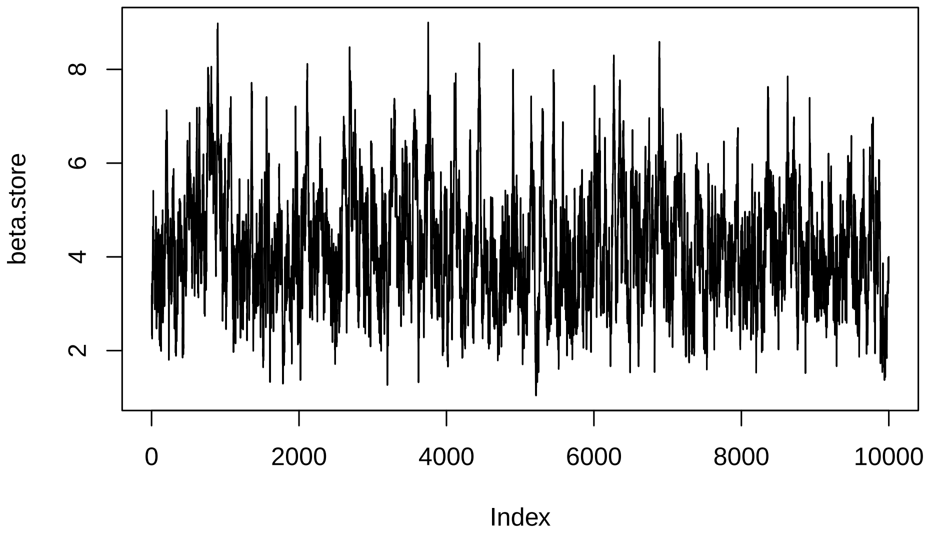
The MCMC algorithm will be 1. Set an initial value for β_1 and set $i = 1$. 2. Propose a new value $\beta' \sim N(\beta_i, \sigma^2)$ 3. Accept β' with probability p_{acc} and set $\beta i + 1 = \beta'$, otherwise reject and set$\beta i + 1 = \beta_i$. 4. Repeat steps 2, 3, and 4 for \$ i = 2, …, M\$.

We fix $b = 0.01$ and use the data

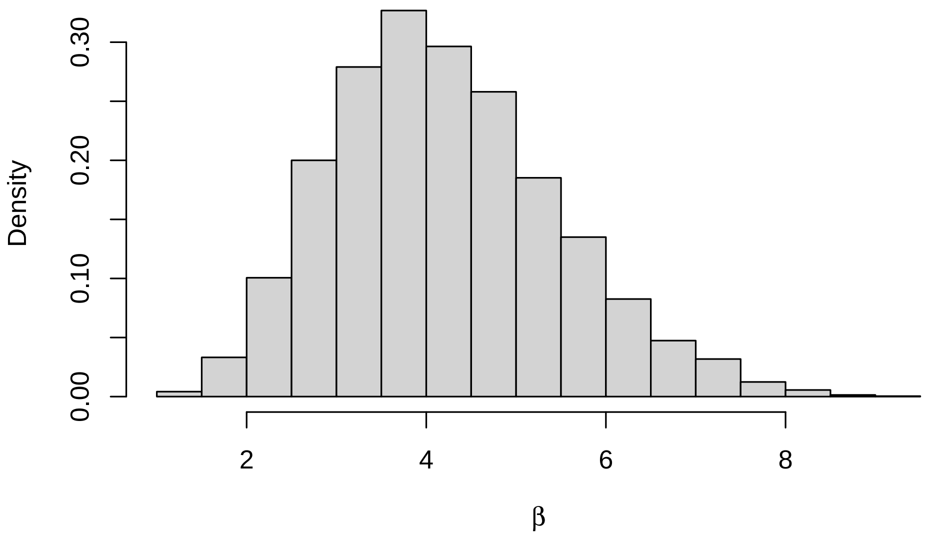
```
x <- c(1.019844, 1.043574, 1.360953, 1.049228, 1.491926, 1.192943, 1.323738, 1.262572,
```

to code up our algorithm.

```r
#Function that evaluates Pareto loglikelihood
log.likelihood <- function(x, beta){

  log.value <- length(x)*log(beta) - (beta + 1)*sum(log(x))

  return(log.value)

}

# MCMC Sampler ------------------------------------------------------------

#Initialise Values
x <- c(1.019844, 1.043574, 1.360953, 1.049228, 1.491926, 1.192943, 1.323738, 1.262572,
n.iter <- 10000 #number of iterations
beta.current <- 2 #initial value for beta
beta.store <- numeric(n.iter) #empty vecotr to store beta at each iteration

#Run MCMC For Loop
for(i in 1:n.iter){

  #Propose prop value for beta
  beta.prop <- rnorm(1, beta.current, 0.5)

  #Compute current and prop loglikelihood
  loglike.prop     <- log.likelihood(x, beta.prop)
  loglike.current <- log.likelihood(x, beta.current)

  #Compute Log acceptance probability
  log.p.acc <- loglike.prop - loglike.current +
    dgamma(beta.prop, 1, 0.01, log = TRUE) - dgamma(beta.current, 1, 0.01, log = TRUE)

  #Accept/Reject
  u <- runif(1)
  if(log(u) < log.p.acc){
    beta.current <- beta.prop
  }

  #Store Current Value
  beta.store[i] <- beta.current


}

#Plot trace plots
plot(beta.store, type = 'l')
```



```
#Investigate posterior
hist(beta.store, freq = FALSE, main = "", xlab = expression(beta))
```



```
quantile(beta.store, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 2.102268 7.027826
```

5.3 Gibbs Sampler

When we can sample directly from conditional posterior distributions, we can use a Gibbs sampler. Suppose we have a distribution with parameters $\{\theta_1, \dots, \theta_N\}$, a Gibbs sampler works as follows:

1. Set initial values $\{\theta_1^{(0)}, \dots, \theta_N^{(0)}\}$
2. Set $i = 1$.
3. Draw a value for $\theta_1^{(i)}$ from $\pi(\theta_1 \mid \theta_2^{(i-1)}, \dots, \theta_N^{(i-1)})$).
4. Draw a value for $\theta_2^{(i)}$ from $\pi(\theta_2 \mid \theta_1^{(i-1)}, \theta_3^{(i-1)}, \dots, \theta_N^{(i-1)})$).
5. Repeat steps 3 and 4 for parameters $\{\theta_3^{(i)}, \dots, \theta_N^{(i)}\}$.
6. Repeat steps 3, 4, and 5, for $i = 2, \dots M$.

In code, this might look like

```
M #number of iterations
N #number of parameters
theta.store   <- matrix(NA, N, M)
theta         <- numeric(N)

for(j in 1:M){
  for(j in 1:N){
    theta[i] <- #sample from conditional with theta[-i]
  }
  theta.store[, j] <- theta.current #store current values
}
```

Example 5.6. In Example 3.5, we had a hierarchical model with

$$
\begin{aligned}
y \mid \lambda &\sim \text{Exp}(\lambda) && \text{(likelihood)}\\
\lambda \mid \gamma &\sim \text{Exp}(\gamma) && \text{(prior distribution)}\\
\gamma \mid \nu &\sim \text{Exp}(\nu) && \text{(hyperprior distribution)}
\end{aligned}
$$

. To derive the full conditional distributions, we only consider the terms in the posterior distributions that depends on the parameters we are interested in. The full conditional distribution for λ is

$$\pi(\lambda \mid y, \gamma) \propto \lambda^{10} e^{-\lambda(95+\gamma)}.$$

This is unchanged and shows that $\lambda \mid y, \gamma \sim \text{Gamma}(11, 95 + \gamma)$. The full conditional distribution for γ is

$$\pi(\gamma \mid y, \lambda) \propto e^{-\nu\gamma}.$$

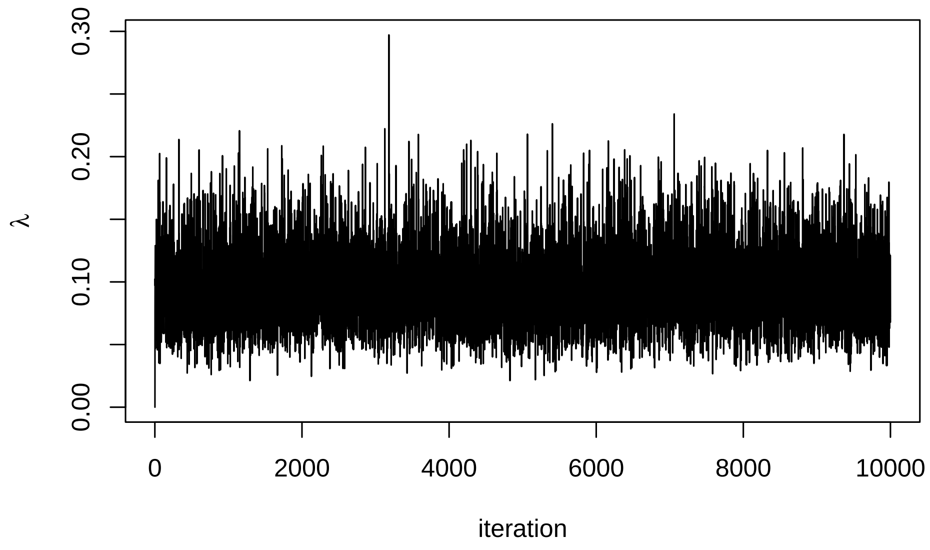
Therefore the full conditional distribution of γ is $\gamma \mid y,\, \lambda \sim \text{Exp}(\lambda + \nu)$.

We can set up a Metropolis-Hastings algorithm using Gibbs samplers to generate samples for λ and γ.

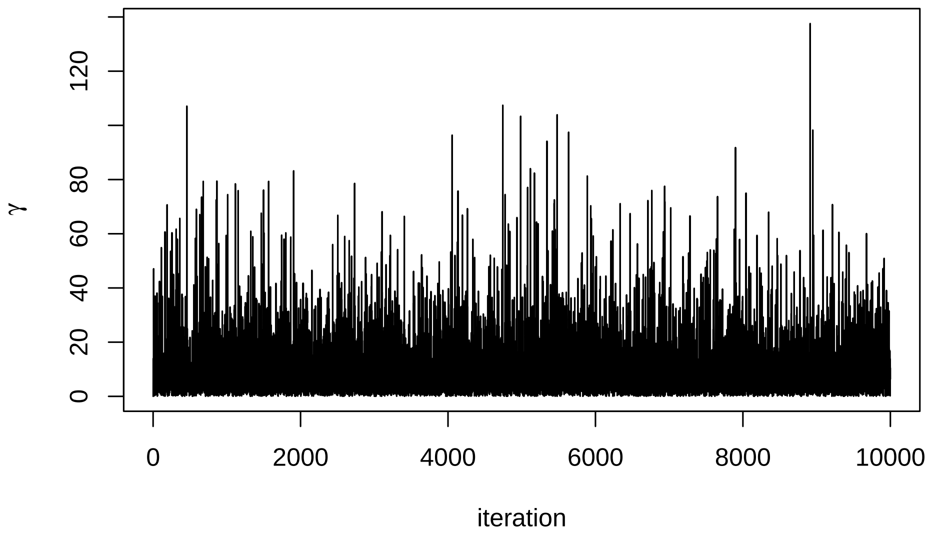
1. Set initial values $\{\lambda^{(0)}, \gamma^{(0)}\}$
2. Set $i = 1$.
3. Draw a value for $\lambda^{(i)} \mid y, \gamma^{(i-1)} \sim \text{Gamma}(10, 95 + \gamma^{(i-1)})$
4. Draw a value for $\gamma^{(i)} \mid y,\, \lambda^{(i)} \sim \text{Exp}(\lambda^{(i)} + \nu)$.
5. Repeat steps 3 and 4 for $i = 2, \dots M$.

We can now code this up and run the algorithm.

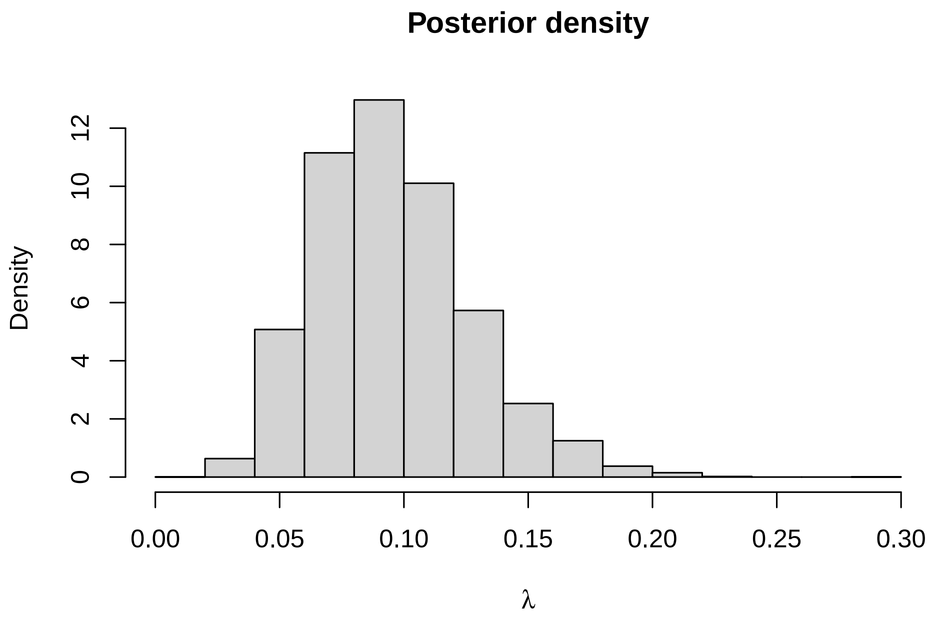
```r
# Set Up MCMC Algorithm ----------------------------------------------------

n.iter <- 10000
lambda.store <- numeric(n.iter) #Store value of Markov chain at end of every iteration
gamma.store <- numeric(n.iter) #Store value of Markov chain at end of every iteration



# Run MCMC Algorithm -------------------------------------------------------
for(i in 2:n.iter){

  #Store current value of Markov Chain
  lambda.store[i] <- rgamma(1, 10, 95 + gamma.store[i-1])
  gamma.store[i]  <- rexp(1, 0.01 + lambda.store[i])

}

#Plot trace plot (Markov chain values)
plot(lambda.store, type = 'l', xlab = "iteration", ylab = expression(lambda))
```



```r
plot(gamma.store, type = 'l', xlab = "iteration", ylab = expression(gamma))
```



```r
#Plot posterior density
hist(lambda.store, prob = TRUE, xlab = expression(lambda), main = "Posterior density")
```



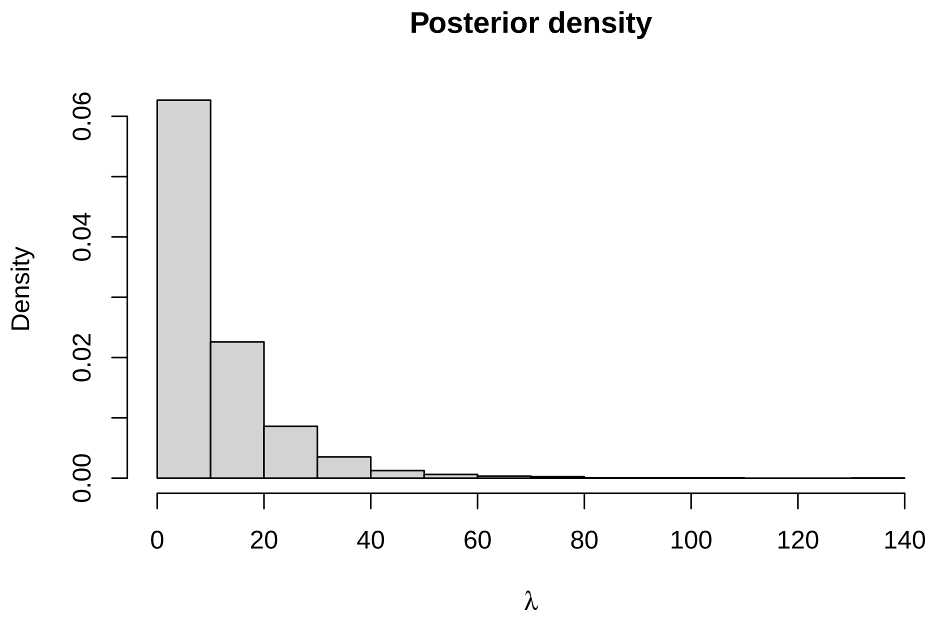
```r
mean(lambda.store) #posterior mean
```

```
## [1] 0.09561264
```

```r
quantile(lambda.store, c(0.025, 0.975)) #95% CI
```

```
##      2.5%     97.5%
## 0.0454177 0.1658069
```

```r
hist(gamma.store, prob = TRUE, xlab = expression(lambda), main = "Posterior density")
```



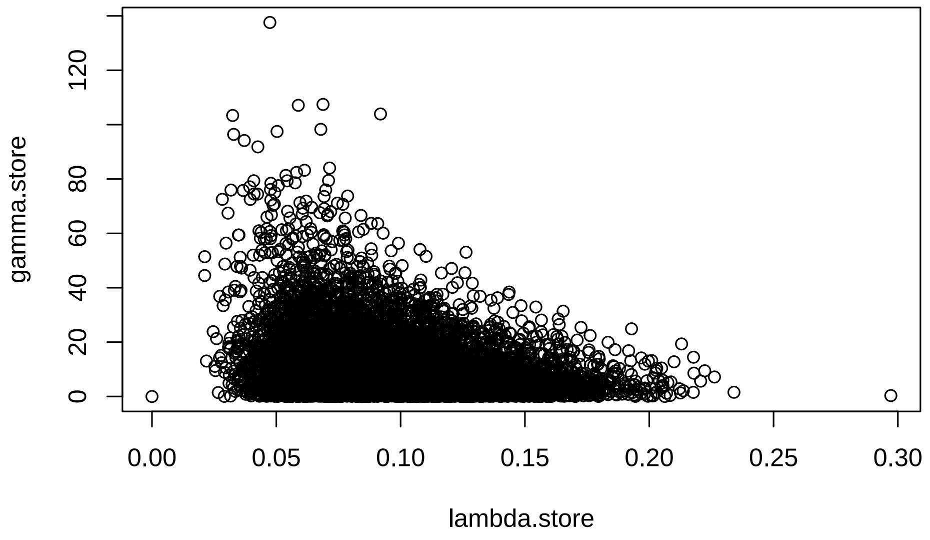
```
mean(gamma.store) #posterior mean
```

```
## [1] 10.4679
```

```
quantile(gamma.store, c(0.025, 0.975)) #95% CI
```

```
##      2.5%      97.5%
##  0.2437085 40.4251992
```

```
#Investogate correlation between parameters
plot(lambda.store, gamma.store)
```



5.4 MCMC Diagnostics

When running an MCMC algorithm, it is always important to check that the Markov chain has converged and is mixing well. For our purposes, mixing well means the chain is exploring the space of possible values of θ effectively and effectively and not getting stuck on the same value for a long time.

A key way of doing this is by looking at the trace plot, which is a time series of the posterior samples simulated by the algorithm. The trace plot should look like it has converged to the stationary distribution and exploring the stationary distribution efficiently. What it shouldn't look like is a long series of small steps, or being stuck in one spot for a long time. There are two definitions that help us isolate an efficient Markov chain.

Definition 5.7. The **burn-in period** is the number of iterations the Markov chain takes to reach the stationary distribution.

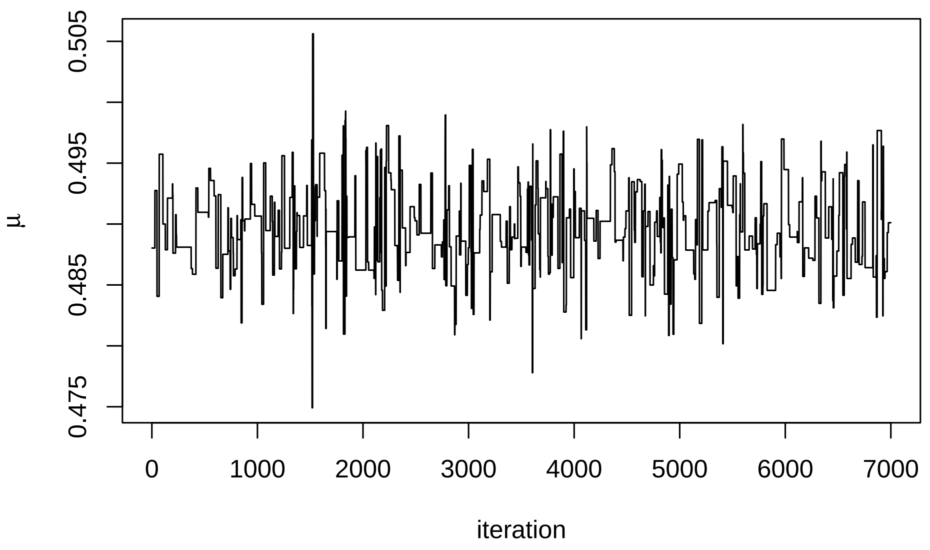
Definition 5.8. The **thinning parameter** is the period of iterations of the Markov chain that are stored.

Example 5.7. In Example 5.4, we saw a Markov chain that mixes well. We took the burn-in period to be 3,000 iterations, which was how long it took to for the chain to converge. Although the posterior distribution is invariant to the choice of the proposal distribution, it still has a large effect of the efficiency of the algorithm and how well the chain mixes. The ideal trace plot looks like white noise, or a hairy caterpillar.

In a Metropolis-Hasting random walk algorithm, the proposal distribution often has a large impact on how well the Markov chain mixes. The variance, or step size, of the proposal distribution can be tuned to ensure the chain mixes well.

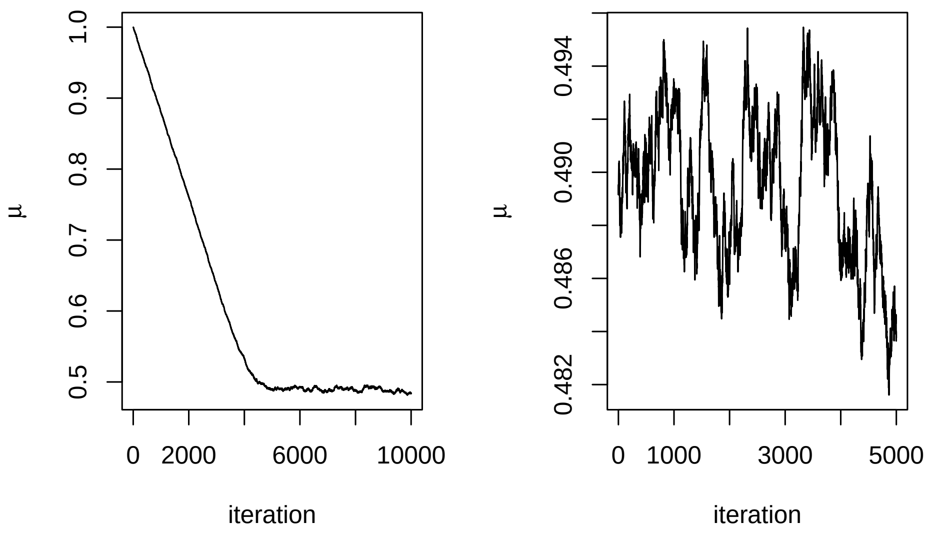
The following two examples show poorly mixing Markov chains. The first is where the step size is too big and the chain frequently gets stuck for several hundred iterations.

```r
set.seed(123) #to reproduce
n.iter   <- 10000
mu.store <- numeric(n.iter)

#Initial values
mu <- 1
sigma <- 0.1 #known

for(i in 1:n.iter){

  #Propose value for mu
  mu.proposed <- runif(1, mu - 0.1, mu + 0.1) #Step size too big

  if(mu.proposed > 0){ #If mu < 0 we can reject straight away

    #Compute (log) acceptance probability
    log.numerator   <- -0.01*mu.proposed -
                        sum(y - mu.proposed)^2/(2*sigma^2)
    log.denominator <- -0.01*mu - sum(y - mu)^2/(2*sigma^2)

    log.p.acc <- log.numerator - log.denominator
    u <- runif(1)

    #Accept/Reject step
    if(log(u) < log.p.acc){
      mu <- mu.proposed
    }
  }

  #Store mu at each iteration
  mu.store[i] <- mu
}
plot(mu.store[-c(1:3000)], type = 'l', xlab = "iteration",
     ylab = expression(mu))
```



The next is where the step size is too small. It takes a long time for the chain to converge (~50% of the run time). When the chain does converge, it is inefficient at exploring the space.

```
set.seed(123) #to reproduce
n.iter   <- 10000
mu.store <- numeric(n.iter)

#Initial values
mu <- 1
sigma <- 0.1 #known

for(i in 1:n.iter){

  #Propose value for mu
  mu.proposed <- runif(1, mu - 0.0005, mu + 0.0005) #Step size too small

  if(mu.proposed > 0){ #If mu < 0 we can reject straight away

    #Compute (log) acceptance probability
    log.numerator   <- -0.01*mu.proposed -
                       sum(y - mu.proposed)^2/(2*sigma^2)
    log.denominator <- -0.01*mu - sum(y - mu)^2/(2*sigma^2)

    log.p.acc <- log.numerator - log.denominator
    u <- runif(1)

    #Accept/Reject step
    if(log(u) < log.p.acc){
      mu <- mu.proposed
    }
  }

  #Store mu at each iteration
  mu.store[i] <- mu
}
par(mfrow = c(1, 2))
plot(mu.store, type = 'l', xlab = "iteration", ylab = expression(mu))
plot(mu.store[-c(1:5000)], type = 'l', xlab = "iteration",
     ylab = expression(mu))
```



The final diagnostic issue we are going to think about is the **curse of dimensionality**. In general, the more parameters we try and update, the less likely we are to accept them. This makes exploring the proposal distribution hard if we are trying to update lots of parameters simultaneously. We can see this if we consider a hypersphere :::{.example} Suppose we have a density function that is uniformly distributed over the area of a sphere in N dimensions with radius r. The sphere has volume

$$V = \frac{\pi^{N/2}}{\Gamma(\frac{N}{2}+1)} r^N.$$

Now consider a smaller sphere inside of our original sphere. This still has dimension N but has radius $r_1 < r$. This small sphere has volume

$$V_1 = \frac{\pi^{N/2}}{\Gamma(\frac{N}{2}+1)} r_1^N.$$

The difference between these two volumes is

$$V - V_1 = \frac{\pi^{N/2}}{\Gamma(\frac{N}{2}+1)}(r^N - r_1^N).$$

For large N, even when $r - r_1$ is small, $(r^N - r_1^N)$ is large. This means that lots of the probability mass is concentrated away from the mode into the outer shell of the sphere.

The hypersphere example shows that in large dimensions we need our Markov chain to spend lots of time away from the posterior mode and in the tails of the distribution, but this is where the proposal distribution has lowest mass. We can avoid this by updating each parameter individually, but this means we need to use the full conditional distributions, which have highest mass near the mode. This 'curse' makes MCMC algorithms inefficient for large dimensions.

##Beyond MCMC MCMC is not the only method available to generate samples from the posterior distribution. MCMC is often slow and inefficient. Much of the work in computational statistics research is about developing fast and efficient methods for sampling from posterior distributions. We are going to look at another method, called approximate Bayesian computation, in the next chapter. Two other methods, beyound the scope of this module, are Sequential Monte Carlo and Hamiltonian Monte Carlo.

Sequential Monte Carlo (SMC) methods for Bayesian inference aim to estimate the posterior distribution of a state space model recursively based on the observed data. Initially, a set of particles representing possible states is sampled from the prior distribution. Then particles are propagated forward using the system dynamics and updated according to their likelihood given the observed data. This update step involves reweighting particles based on how well they explain the observed data. To ensure that the particle set accurately represents the posterior distribution, resampling is performed, where particles with higher weights are more likely to be retained. By iteratively repeating these steps, SMC effectively tracks the evolution of the posterior distribution over time, providing a flexible and computationally efficient framework for Bayesian inference in dynamic systems.

Hamiltonian Monte Carlo (HMC) is a sophisticated Markov chain Monte Carlo (MCMC) method for sampling from complex, high-dimensional target distributions, commonly used in Bayesian inference. Unlike traditional MCMC methods, which rely on random walk proposals, HMC employs Hamiltonian dynamics to guide the exploration of the state space. By introducing auxiliary momentum variables, HMC constructs a joint distribution over the original target variables and the momentum variables. This augmented space enables the use of Hamiltonian dynamics, which can efficiently explore the target distribution by simulating the evolution of the system's energy function. The key idea is to use the gradient of the target distribution's log-probability to determine the momentum dynamics, leading to more effective proposals that can traverse the state space more efficiently. HMC samples are obtained by simulating Hamiltonian dynamics over a trajectory and then accepting or rejecting the proposed states based on Metropolis-Hastings criteria. Overall, HMC offers significant improvements in exploration efficiency compared to traditional MCMC methods, particularly in high-dimensional spaces, making it a powerful tool for Bayesian inference.

5.5 Lab

Exercise 5.1. You observe the following draws from a Binomial distribution with 25 trials and probability of success p.

```
y <- c(20, 16, 20, 17, 18, 19, 19, 18, 21, 20, 19, 22, 23, 19, 20, 19, 21, 20, 25, 15)
```

Use a normal prior distribution with mean 0.5 and variance 0.1^2. Write a Metropolis-Hastings Random Walk algorithm to obtain samples from the posterior distribution (you can use R's built in function for the likelihood function and prior distribution, but if you don't take logs you will run into small number errors).

Exercise 5.2. In a medical trial, to investigate the proportion p of the population who have a particular disease a random sample of 20 individuals is taken. Ten of these are subject to a diagnostic test (Test A) which detects the disease when it is present with 100% certainty. The remaining 10 are given a test (test B) which only detects the disease with probability 0.8 when it is present. Neither test can give a false positive result. before collecting the data your prior belief about p is represented by a U(0,1) distribution. Suppose that, for Test A, 5 out of 10 test positive while for test B, 3 out of 10 test positive. Use an MCMC algorithm to investigate the posterior density of p and estimate its posterior mean and variance.

Exercise 5.3. Code up an MCMC algorithm for Example 3.5 using Gibbs samplers.

Exercise 5.4. The density function for the inverse-gamma distribution is

$$\pi(x \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta/x}$$

Using independent Gamma prior distributions on the model parameters, $\alpha \sim \Gamma(a, b)$ and $\beta \sim Gamma(c, d)$, write down the posterior distribution for the model parameters. Only one will have a closed form.

Develop a code an MCMC algorithm to sample from the posterior distribution by alternating between sampling α and then β.

Simulate some data from the inverse-gamma distribution and see if you can recover the parameters used to simulate the data. Is there any correlation between the samples for α and β.

Chapter 6

Advanced Computation

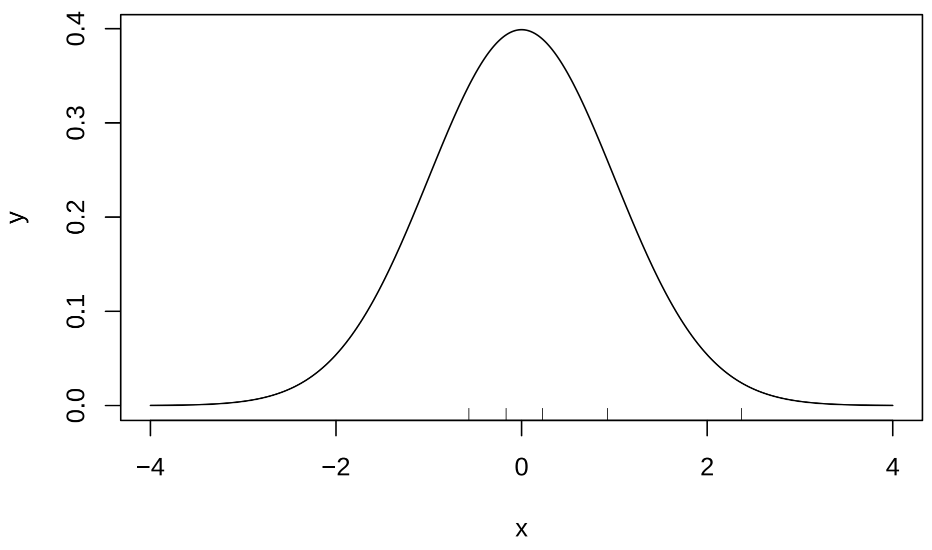
Now we have the tools of Bayesian inference and methods to sample from complex posterior distributions, we can start to look at more advanced methods and models. This chapter is split into three distinct parts, each showing a different method in Bayesian inference.

6.1 Gaussian Processes

So far in the module, we have considered prior distribution on parameters. These parameters have taken values (mostly real) or real-valued vectors. In this section, we're going to extend this idea further to place prior distributions on functions. That is, we're going to describe a prior distribution that when sampled gives us functions. The method we're going to use is called a Gaussian Process (GP).

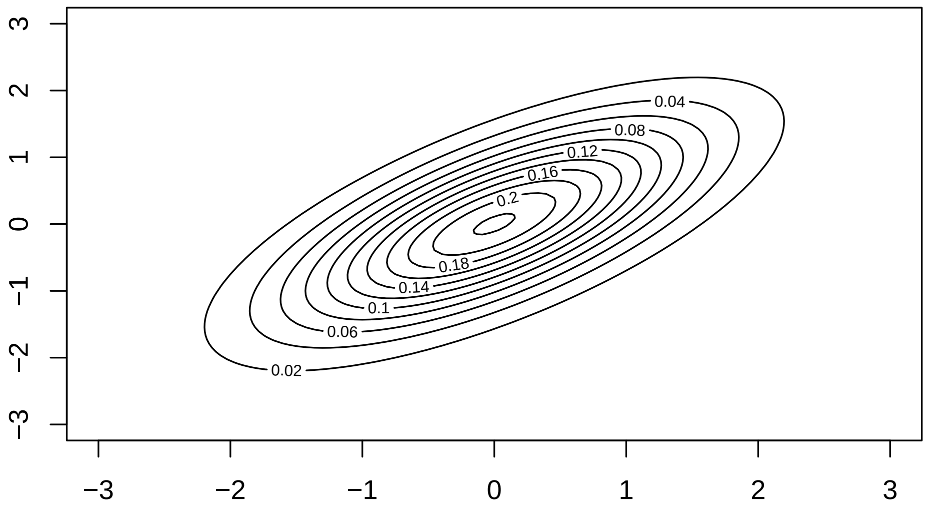
Before, we define a GP, we're going to build an intuitive definition of it. Recall the normal distribution with mean μ and variance σ^2, $N(\mu, \sigma^2)$. It assigns probabilities to values on the real line – when we sample from it, we get real values. The plot below shows the density function for a $N(0, 1)$ distribution and five samples.

```r
#Plot N(0, 1)
x <- seq(-4, 4, 0.01)
y <- dnorm(x)
plot(x, y, type = 'l')

#Add samples
samples <- rnorm(5)
rug(samples)
```

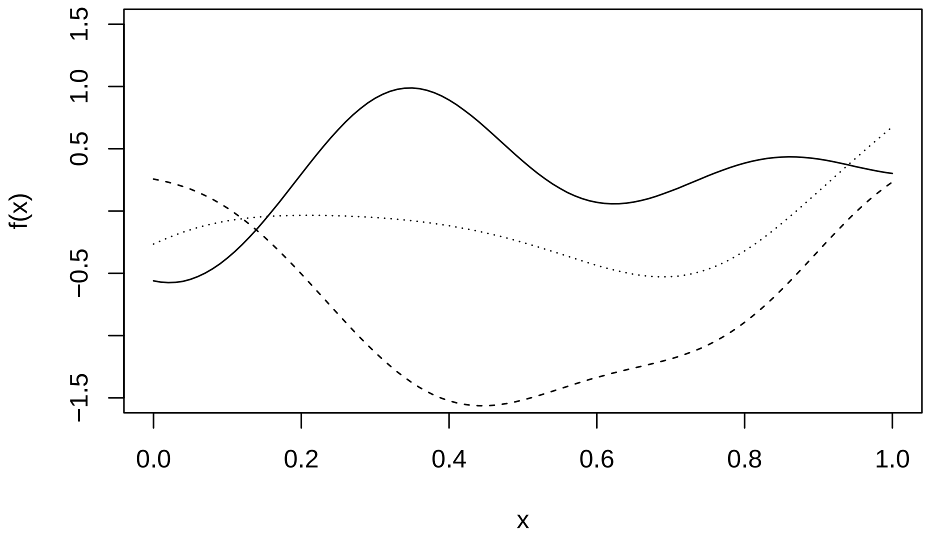


The multivariate normal distribution extends this to a vector space, $\mathbb{R}^N$. Instead of having a mean and variance value, the distribution is defined through a mean vector and covariance matrix. The mean vector describes the expected value of each component of the vector and the covariance matrix describes the relationship between each pair of components in the vector. When we draw samples, we get vectors. The plot below shows the density of the multivariate normal distribution with $N = 2$, zero mean, $\sigma_x^2 = \sigma_y^2 = 1$ and $\rho = 0.7$.

```r
#Create Grid
x <- seq(-3,3,length.out=100)
y <- seq(-3,3,length.out=100)

#Evaluate density at grid
z <- matrix(0,nrow=100,ncol=100)
mu <- c(0,0)
sigma <- matrix(c(1, 0.7, 0.7, 1),nrow=2)
for (i in 1:100) {
  for (j in 1:100) {
    z[i,j] <- mvtnorm::dmvnorm(c(x[i],y[j]),
                               mean=mu,sigma=sigma)
  }
}

#Generate contour plot
contour(x, y ,z)
```



A GP takes this one step further and puts a prior distribution on a function space. It is specified by a mean function, $\mu(\cdot)$ and covariance function $k(\cdot, \cdot)$. The mean function describes the expected value of each point the function can be evaluated at, and the covariance function describes the relationship between each point on the function. The plot below shows three samples from a GP distribution with mean function the zero function $\mu(x) = 0 \, \forall x$ and a covariance function that supports smooth functions.



Definition 6.1. A **Gaussian Process** is a collection of random variables, any finite number of which have a joint Gaussian distribution.

This says that is we think of a function as an infinite collection of points, then if any finite subset of those points following a Gaussian distribution, we have a Gaussian process. In reality, we set up the function so that is meets this definition. More formally,

Definition 6.2. A **GP distribution on a function** $f(x)$ is defined through its mean function $\mu(x) = \mathbb{E}(x)$ and covariance function $k(x, x') = \mathbb{E}(x)\left((f(x) - \mu(x))(f(x') - \mu(x'))\right)$. We write it as $f(x) \sim \mathcal{GP}(\mu(x), k(x, x'))$.

Before we go any further, it is worth proceeding with caution. Those with good memories will recall Bernstein-von-Mises' theorem from Chapter 3.

Theorem 6.1 (Bernstein-von-Mises)**.** *For a well-specified model $\pi(y \mid \theta)$ with a fixed number of parameters, and for a smooth prior distribution $\pi(\theta)$ that is non-zero around the MLE $\widehat{\theta}$, then*

$$\left\| \pi(\theta \mid y) - N\left(\widehat{\theta}, \frac{I(\widehat{\theta})^{-1}}{n}\right) \right\|_{TV} \to 0.$$

Bernstein-von-Mises' theorem only holds when the model has a fixed (i.e. finite) number of parameters. A GP is defined on an infinite collection of points, and so this theorem does not hold. This is the first time in this module we have encountered a distribution where Bernstein-von-Mises' theorem does not hold. Fortunately, various forms of Bernstein-von-Mises' theorems for GPs exist, with many coming about in the early 2010s. However, this is still an ongoing area of research.

6.1.1 Covariance Functions

One issue when using GPs is describing the covariance function. How do we decide how each pair of points (there being an infinite number of them)? There are lots of standard choices of covariance functions that we can choose from, each one making different assumptions about the function we are interested in.

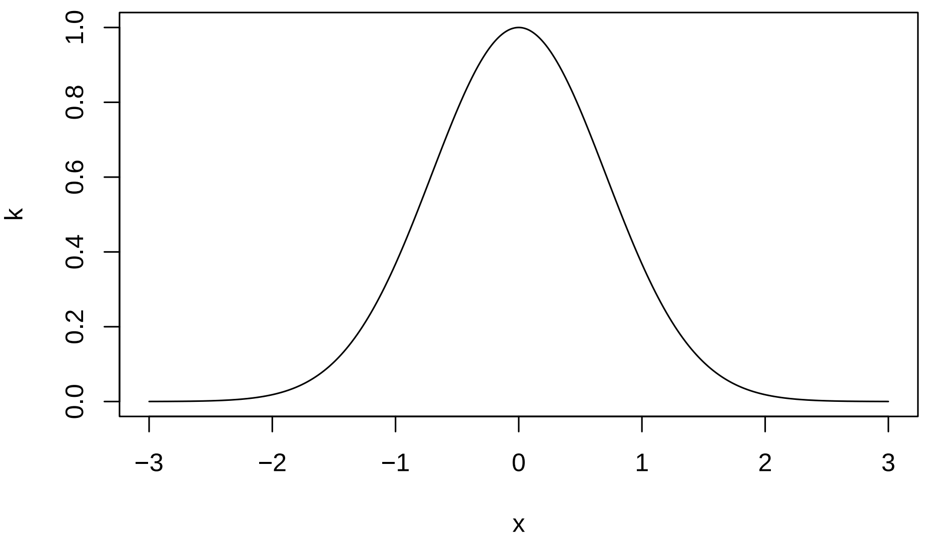
The most common covariance function is the squared exponential functions. It is used to model functions that are 'nice', i.e. they are smooth, continuous and infinitely differentiable.

Definition 6.3. The **squared exponential covariance function** takes the form

$$k(x, x') = \alpha^2 \exp\left\{-\frac{1}{l}(x - x')^2\right\},$$

where α^2 is the signal variance and $l > 0$ is the length scale parameter.

For now, consider $\alpha = l = 1$. What is the covariance between the function evaluated at 0 and the function evaluated at x? The plot below shows the covariance.



The covariance is highest when the x is near to 0, i.e. the points are immediately next to each other. If the value of x is ± 2, the covariance is 0. As we are dealing with a joint normal distribution, a covariance of 0 implies independence. So with this covariance function, the value of $f(x)$ is independent of $f(0)$ if $|x|$ is larger than about two. The parameter l is called the length scale parameter and dictates how quickly the covariance decays. Small values of l mean that the value of the function at nearby points are independent of each other, resulting in functions that look like white noise. Large values of l mean that even if points are far away, they are still highly dependent on each other. This gives very flat functions.

The choice of covariance function is a modelling choice – it depends completely on the data generating process you are trying to model. The following properties are useful when deciding which covariance function to use.

Definition 6.4. A **stationary** covariance function is a function of $x - x'$. That means it is invariant to translations in space.

Definition 6.5. An **isotropic** covariance function is a function only of $|x - x'|$. That means it is invariant to rigid translations in space.

Definition 6.6. An **dot product** covariance function is a function only of $x \cdot x'$. That means it is invariant to rigid rotations in space, but not translations.

What is most important is that the matrix resulting from a covariance function is positive semi-definite. This is because covariance matrices must be positive semi-definite.

Definition 6.7. An $N \times N$ matrix Σ is positive semi-definite if it is symmetric and

$$x^T \Sigma x \geq 0 \quad \text{for all } x \in \mathbb{R}^N.$$

The squared exponential covariance function is isotropic and produces functions that are continuous and differentiable. There are many other types of covariance functions, including ones that don't produce functions that are continuous or differentiable. Two more are given below.

Definition 6.8. The **M'atern covariance function** models functions that are differentiable only once:

$$k(x,x') = \left(1 + \frac{\sqrt{3}(x-x')^2}{l}\right) \exp\left\{-\frac{\sqrt{3}(x-x')^2}{l}\right\}.$$

Definition 6.9. The periodic covariance function models functions that are periodic and it is given by

$$k(x,x') = \alpha^2 \exp\left\{-\frac{2}{l}\sin^2\frac{(x-x')^2}{p}\right\},$$

where the period is p.

Definition 6.10. The dot product covariance function models functions that are rotationally invariant and it is given by

$$k(x,x') = \alpha^2 + x \cdot x'.$$

These are just some covariance functions. In addition to the covariance functions defined, we can make new covariance funcitons by combining existing ones.

Proposition 6.1. *If k_1 and k_2 are covariance functions, then so is $k_1 + k_2$.*

Proof. Let f_1 be a function with covariance function k_1 and f_2 be a function with covariance function k_2, then $f = f_1 + f_2$ has covariance function $k_1 + k_2$. □

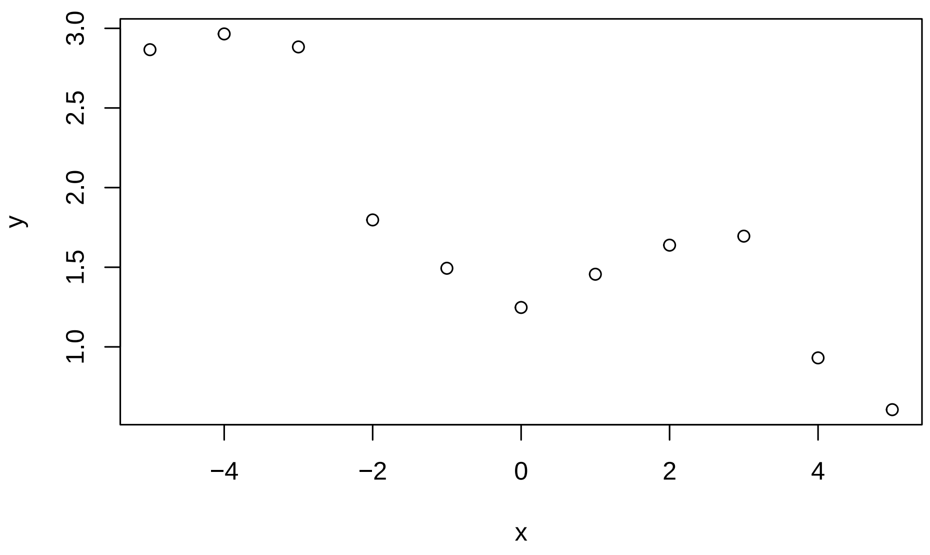
Proposition 6.2. *If k_1 and k_2 are covariance functions, then so is k_1k_2.*

Proof. See problem sheet. □

6.1.2 Gaussian Process Regression

One of the main applications of GPs in in regression. Suppose we observe the points below $y = \{y_1, \dots, y_N\}$ and want to fit a curve through them. One method is to write down a set of functions of the form $y = X^T\beta + \varepsilon$, where X is the design matrix and β a vector of parameters. For each design matrix X, construct the posterior distributions for β and use some goodness-of-fit measure to choose the most suitable design matrix.

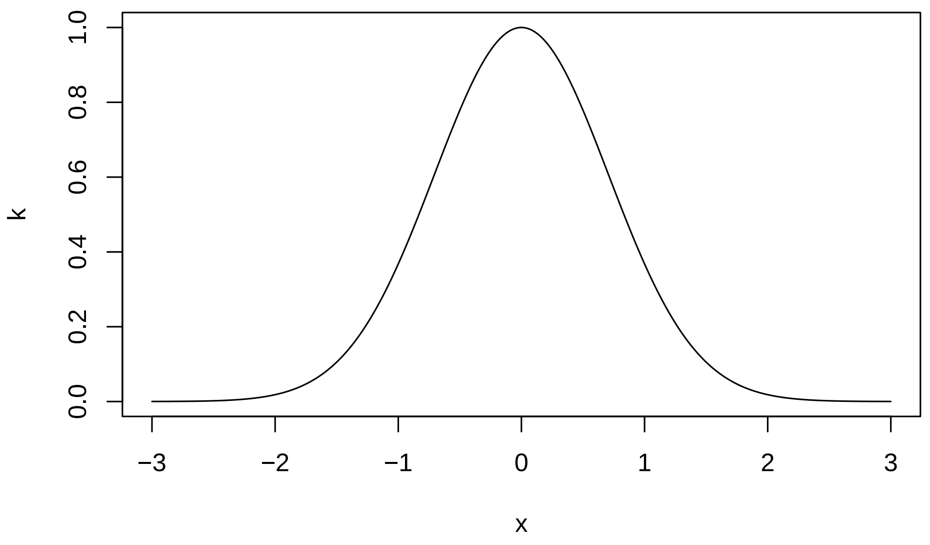
```r
x <- -5:5
y <- sin(x/2)^2 + exp(-x/5) + rnorm(length(x), 0, 0.2)
plot(x, y)
```



One difficulty is writing down the design matrices X, it is often not straightforward to propose or justify these forms GPs allow us to take a much less arbitrary approach, simply saying that $y_i = f(x_i)+\varepsilon_i$ and placing a GP prior distribution on f.

Although we're placing an prior distribution with an infinite dimension on f, we only ever need to work with a finite dimensional object, making this much easier. We only observe the function at finite number of points $f = \{f(x_1), \dots, f(x_N)\}$ and we will infer the value of the function at points on a fine grid, $f^* = \{f(x_1^*), \dots, f(x_N^*)\}$. By the definition of a GP, the distribution of these points is a multivariate normal distribution.

Example 6.1. Suppose we observe $y = \{y_1, \dots, y_N\}$ at $x = \{x_1, \dots, x_N\}$. The plot below shows these points.



Using the model $y_i = f(x_i) + \varepsilon_i$, where $\varepsilon_i \sim N(0, \sigma^2)$, we want to infer the function f evaluated at a gird of points $f^* = \{f(x_1^*), \dots, f(x_N^*)\}$. We place a GP prior distribution on $f \sim \mathcal{GP}(0, k)$, where k is the squared exponential covariance function. Using the model, the covariance between points y_i and y_j is

$$\text{cov}(y_i, y_j) = k(x_i, x_j) + \sigma^2 1_{i=j}.$$

That is the covariance function evaluated at x_i and x_j plus σ^2 if $i = j$. We can write this in matrix form as $K(x, x) + \sigma^2 I$ where I is the identity matrix. The distribution of y is therefore $y \sim N(0,\ K(x, x) + \sigma^2 I)$. By definition of the GP, the distribution of the function evaluated at the fine grid is $f^* \sim N(0, K(x^*, x^*))$.

We can now write the joint distribution as

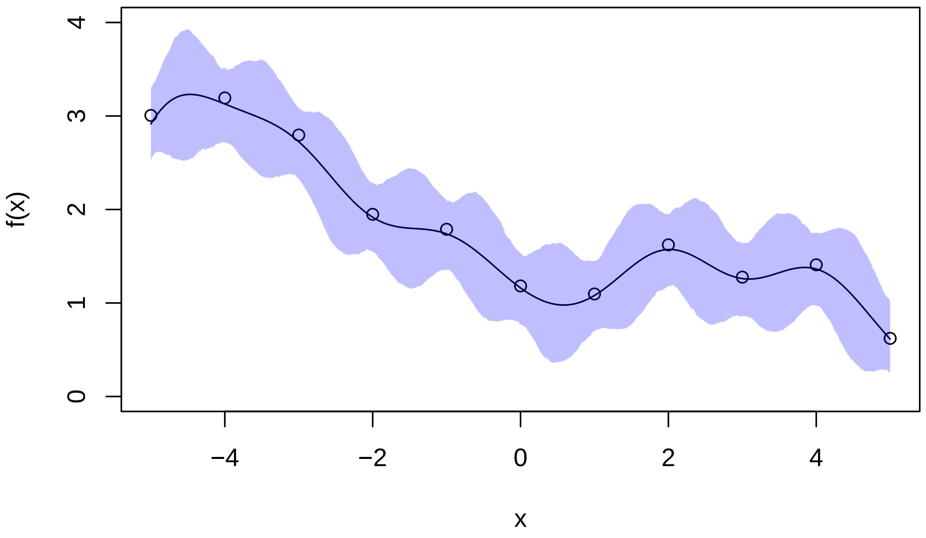
$$\begin{pmatrix} y \\ f^* \end{pmatrix} \sim N\left(0, \begin{pmatrix} K(x, x) + \sigma^2 I & K(x, x^*) \\ K(x^*, x) & K(x^*, x^*) \end{pmatrix}.\right)$$

The off-diagonal terms in the covariance matrix describe the relationship between the observed points y and the points of interest f^*. We can now write down the distribution of f^* given the observed points y and σ^2.

$$f^* \mid y, \sigma^2 \sim N(\mu^*,\ K^*),$$

where $\mu^* = K(x^*, x)(K(x, x)+\sigma^2 I)^{-1} y$ and $K^* = K(x^*, x^*) - K(x^*, x)(K(x, x) + \sigma^2 I)^{-1} K(x, x^*)$.

We set the fine gird to be $x^* = \{-5, -4.99, -4.98, \dots, 5\}$, the GP parameters $\alpha = l = 1$ and $\sigma = 0.2$. The posterior mean and 95% credible interval are shown be-



low.
The posterior mean for f is a smooth line passing near each point. The 95% credible interval for f has the smallest variance near each point, and largest furthest away from the points.

6.2 Data Augmentation

Real world data are often messy with data points missing which may mean they are partially or completely unobserved. One common example of this is in clinical trials where people drop out of the trial before their treatment is complete. Another example is crime data, where only a fraction of crimes are reported and many crimes go unobserved. Two common ways to deal with partially or completely unobserved are:

- Remove data points that are not completely observed. This throws away information and is likely to increase the overall uncertainty in the estimates.
- Replace data points that are not completely observed with estimates such as the sample mean. This is likely to underestimate the uncertainty as we are treating the observation as completely observed when it is not.

The Bayesian framework provides a natural way for dealing with missing, partially, or completely unobserved data. It allows us to treat the missing data points as random variables and infer the data points alongside the model parameters. This provides us with a method to quantify the uncertainty around our estimates of the missing data points.

In data augmentation, we distinguish between two likelihood functions.

Definition 6.11. The **observed data likelihood function** is the likelihood function of the observed data.

Definition 6.12. The **complete data likelihood function** is the likelihood function of the observed data and any missing or censored data had they been fully observed.

The difference between the two likelihood functions is that the complete data likelihood function is the functions had we observed everything we want to observe. However, as the complete data likelihood function contains data we didn't fully observe, we can't compute it. Instead we can only evaluate the observed data likelihood function. A simple probability based example of this is if there are two events X and Y, where the outcome of X is observed and Y unobserved. The complete data likelihood is $pi(X = x, Y = y)$ because we are considering all the events, observed or not. However, we can only compute $\pi(x) = \int_{y \in Y} \pi(X = x, Y = y)$ or $\pi(x) = \sum_{y \in Y} \pi(X = x, Y = y)$, since y is unobserved.

In data augmentation, we start off with the observed data likelihood function and then augment this function by introducing variables that we want to have fully observed. This then gives us the complete data likelihood function.

6.2.1 Imputing censored observations

The first example we will look at is when data is censored. Instead of throwing away these observations, we will instead treat them as random variables and infer their values.

Example 6.2. A bank checks transactions for suspicious activities in batches of 1000. Denote the probability a transaction is suspicious by p and the number of suspicious transactions in a batch by Y.

The bank checks five batches and observes $y_1, \dots, y_4$ suspicious transactions in the first four batches. Due to a computer error, the number of suspicious transactions in the final batch is not properly recorded, but is known to be less than 6.

The observed data likelihood functions is

$$\pi(y_1, \dots, y_4, \tilde{y}_5 \mid p) = \left(\prod_{i=1}^{4} \binom{1000}{y_i} p^{y_i} (1-p)^{1000-y_i} \right) \left(\sum_{j=0}^{5} \binom{1000}{j} p^{j} (1-p)^{1000-j} \right).$$

This is known as marginalising over the missing variable, just as we did in the simple probability example earlier. Placing a uniform prior distribution on $p \sim U[0, 1]$ give the posterior distribution

$$\pi(p \mid y_1, \dots, y_4, \tilde{y}_5) = \left(\prod_{i=1}^{4} \binom{1000}{y_i} p^{y_i} (1-p)^{y_i} \right) \left(\sum_{j=0}^{5} \binom{1000}{j} p^{j} (1-p)^{1000-j} \right).$$

Although we could sample from this distribution, it is not easy to work with. Instead, we can write down the complete data likelihood. Suppose that y_5 was observed, then the complete data likelihood may be written as

$$\pi(y_1, \dots, y_5 \mid p) = \prod_{i=1}^{5} \binom{1000}{y_i} p^{y_i}(1-p)^{1000-y_i},$$

The posterior distribution is therefore

$$p \mid y_1, \dots, y_5 \sim \text{Beta}\left(\sum_{i=1}^{5} y_i + 1, 5000 + 1 - \sum_{i=1}^{5} y_i\right).$$

The full conditional distribution of y_5 given p, the other data points an $y_5 < 6$ is

$$\pi(y_5 = y \mid y_1, \dots, y_4, y_5 < 6, p) = \frac{\binom{1000}{y} p^{y}(1-p)^{1000-y}}{\sum_{j=0}^{5} \binom{1000}{j} p^{j}(1-p)^{1000-j}}, \qquad y < 6$$

We can use a Gibbs sampler alternating between sampling p and y_5.

6.2.2 Imputing Latent Variables

Often there are variables that are cannot be observed, these may be hidden somehow or introduced to help with the modelling. Instead we can learn about this variable indirectly from the data.

A **latent variable** is a variable that cannot be observed.

A mixture model is an example of latent variables being useful.

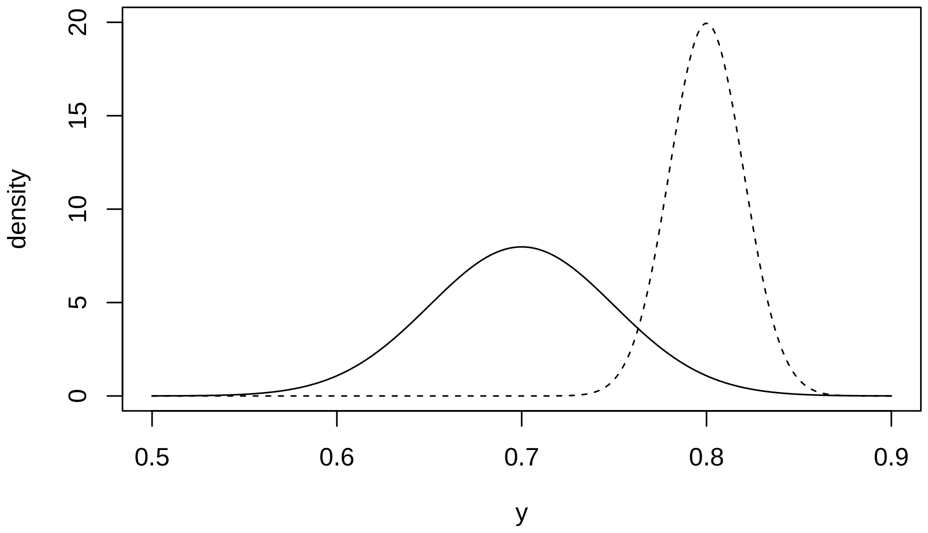
Example 6.3. Royal Mail use image detection software to read postcodes on letters. A camera scans the front of an envelope and then records the barcode. This example is a very simplified version of how the system could work.

Suppose the machine is processing a bag of letters addressed to people in either B1 or B2 postcodes. The camera scans the first two characters of the postcode (B1 or B2) and records the proportion of the scanned image that is taken up by the characters. The picture below shows an example of what the scanned image looks like.

We introduce a latent variable $z_i \sim \text{Bernoulli}(p)$ that describes if the characters on the i^{th} image are B1 or B2. The observation y_i is the proportion of the i^{th}

image that is taken up by the characters. We observe y_i, but want to estimate z_i. The difficultly is there lack of one-to-one correspondence between the values y_i can take and the value z_i. Due to the different handwriting and fonts used on envelopes, if the letter is going to B1 ($Z = 1$), then $Y_i \sim N(0.7, 0.05^2)$ and if it is going to B2 ($Z = 2$), then $Y_i \sim N(0.8, 0.02^2)$. The plot below shows the two densities and the overlap between them.

```r
a <- seq(0.5, 0.9, 0.001)
x <- dnorm(a, 0.7, 0.05)
y <- dnorm(a, 0.8, 0.02)
plot(a, x, type = 'l', ylim = c(0, 20), xlab = expression(y),
     ylab = "density")
lines(a, y, lty = 2)
```



As the variables z are latent, the observed data likelihood function is

$$\pi(y \mid p) = \prod_{i=1}^{N} \left[p\pi(y_i \mid \mu = 0.7, \sigma^2 = 0.05^2) + (1-p)\pi(y_i \mid \mu = 0.8, \sigma^2 = 0.02^2) \right].$$

Instead, it's easier to work with the complete data likelihood function, supposing we had observed the variables z. This is given by

$$\pi(y, z \mid p) = \binom{N_1 + N_2}{N_1} p^{N_1}(1-p)^{N_2} \prod_{i; z_i = 1} \pi(y_i \mid \mu = 0.7, \sigma^2 = 0.05^2) \times \prod_{i; z_i = 2} \pi(y_i \mid \mu = 0.8, \sigma^2 = 0.02^2),$$

where N_1 and N_2 are the number of letters for B1 and B2 respectively. This form makes it much easier to derive the posterior distributions and estimate the parameter values.

We place a uniform prior distribution on the parameter p, which gives the posterior distribution

$$p \mid y, z \sim \text{Beta}(N_1 + 1, N_2 + 1).$$

The distribution of z_i given the parameter p and the observation y_i can be derived using Bayes' theorem

$$p_i^* = \pi(z = 1 \mid p, y_1) = \frac{p\pi(y_i \mid \mu = 0.7, \sigma^2 = 0.05^2)}{p\pi(y_i \mid \mu = 0.7, \sigma^2 = 0.05^2) + (1-p)\pi(y_i \mid \mu = 0.8, \sigma^2 = 0.02^2)}.$$

The full conditional distribution is therefore $z_i \mid y, p \sim \text{Bernoulli}(p_i^*)$.

An MCMC algorithm for this would repeat the following two steps:

1. Sample $p \mid y, z \sim \text{Beta}(N_1 + 1, N_2 + 1)$.
2. Sample $z_i \mid y, p \sim \text{Bernoulli}(p_i^*)$ for each i.

6.3 Approximate Bayesian Computation

So far, we have always considered models where the likelihood function is easy to work with. By easy, we mean that we can evaluate the likelihood function for lots of different values, and we can evaluate it cheaply. In some cases, it might not be possible to write down the likelihood function, or it might not be possible to evaluate it. In these cases, we refer to methods call **likelihood free inference**.

Example 6.4. Models to predict the weather are notoriously complex. They contain a huge number of parameters, and sometimes it is not possible to write this model down exactly. In cases where the likelihood function for the weather model can be written down, we would have to start the MCMC algorithm from scratch every time we collected new data.

Approximate Bayesian Computation (ABC) is a likelihood free algorithm that relies on reject sampling from the prior distribution. When constructing an ABC algorithm, we only need to be able to generate data given a parameter value and not evaluate the likelihood of seeing specific data given a parameter value.

We are going to look at two types of ABC. The first is ABC with rejection

6.3.1 ABC with Rejection

Definition 6.13. To carry out inference for a parameter θ using an Approximate Bayesian Computation algorithm with rejection

1. Sample a value for the parameter θ^* from the prior distribution $\pi(\theta)$.
2. Generate some data $y*$ from the data generating process using the parameter value θ^*.

3. Accept θ^* as a value from the posterior distribution if $||y - y^*|| < \varepsilon$ for some $\varepsilon > 0$. Otherwise reject θ^*
4. Repeat steps 1 - 3.

Proposition 6.3. *The approximate posterior distribution using ABC with rejection is*

$$\pi_\varepsilon(\theta \mid y) \propto \int \pi(y^* \mid \theta^*)\pi(\theta^*)I_{A_\varepsilon(y^*)}dy^*,$$

where $A_\varepsilon(y^*) = \{y^* \mid ||y^* - y|| < \varepsilon\}$.

Example 6.5. This is a simple example, where we can derive the posterior distribution, but it allows us to see how this method works. Suppose we observe $Y_1, \dots, y_510 \sim Beta(3, \beta)$. We place a uniform prior on β such that $\beta \sim U[0, 5]$. The ABC algorithm with rejection works as follows:

1. Sample a value $\beta^* \sim U[0, 5]$.
2. Simulate $y_1^*, \dots, y_1^*0 \sim Beta(3, \beta^*)$
3. Compute $D = \sum_{i=1}^{10}(y_i - y_i^*)^2$. If $D < 0.75$, accept β^* as a sample from the posterior distribution. Otherwise, reject β^*.
4. Repeat steps 1, 2, and 3.

The code below carries out this algorithm.

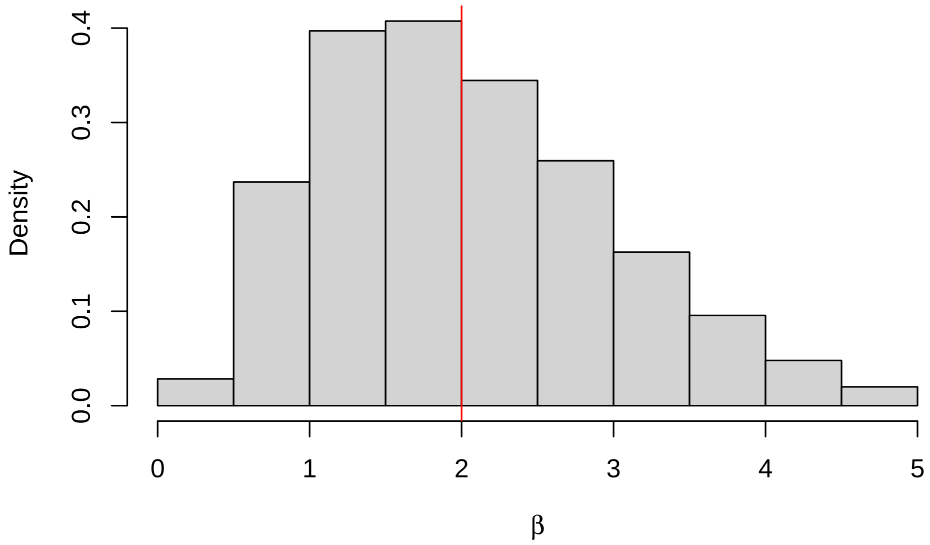
```
#Set Up Example
set.seed(1234)
n <- 10
y <- rbeta(n, 3, 2)
y
```

```
##  [1] 0.8519237 0.5286251 0.3126172 0.9691679 0.4883547 0.4677043 0.7339799
##  [8] 0.7279578 0.7317827 0.7971786
```

```
#Set Up ABC
n.iter <- 50000
b.store <- numeric(n.iter)
epsilon <- 0.75

#Run ABC
for(i in 1:n.iter){

  #Propose new beta
  b <- runif(1, 0, 5)

  #Simualate data
  y.star <- rbeta(n, 3, b)

  #Compute statistic
  d <- sum((y-y.star)^2)
  #Accept/Reject
  if(d < epsilon){
    b.store[i] <- b
  } else{
    b.store[i] <- NA
  }

}

#Get number of reject samples
sum(is.na(b.store))
```

```
## [1] 37886
```

```
#Plot Approximate Posterior
hist(b.store, freq = FALSE, xlab = expression(beta), main = "")
abline(v = 2, col = 'red')
```



```
mean(b.store, na.rm = TRUE)
```

```
## [1] 2.03304
```

```
quantile(b.store, c(0.025, 0.975), na.rm = TRUE)
```

```
##      2.5%     97.5%
## 0.5843792 4.1369339
```

One important question is how to choose the value for ε? It turns out this is an incredibly hard question that is specific to each application. Often the approximate posterior distribution $\pi_\varepsilon(\theta \mid y)$ is very sensitive to the choice of ε.

Example 6.6. Let's repeat the example, first with $\varepsilon = 0.12$. In this case, almost all of the proposals are rejected.

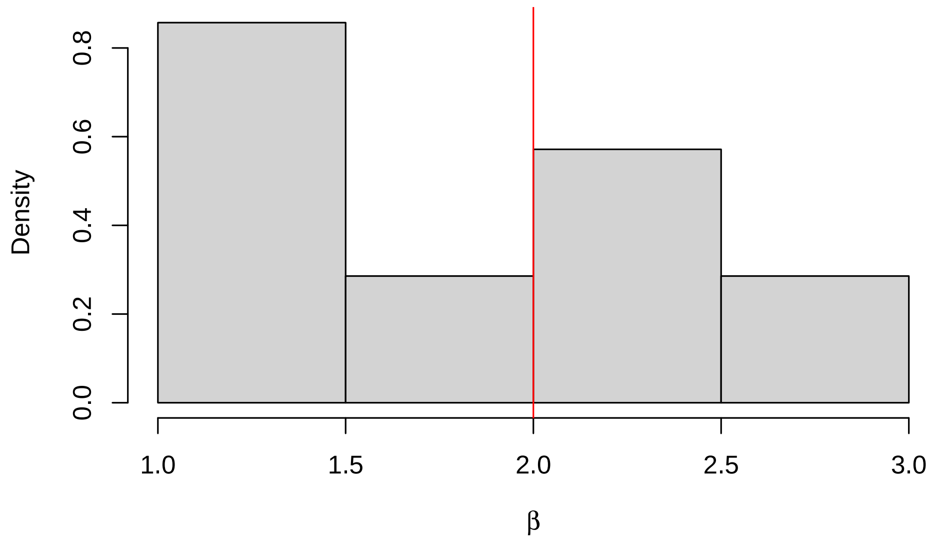
```
#Set Up Example
set.seed(1234)
n <- 10
y <- rbeta(n, 3, 2)
y
```

```
##  [1] 0.8519237 0.5286251 0.3126172 0.9691679 0.4883547 0.4677043 0.7339799
##  [8] 0.7279578 0.7317827 0.7971786
```

```
#Set Up ABC
n.iter <- 50000
b.store <- numeric(n.iter)
epsilon <- 0.12

#Run ABC
for(i in 1:n.iter){

  #Propose new beta
  b <- runif(1, 0, 5)

  #Simualate data
  y.star <- rbeta(n, 3, b)

  #Compute statistic
  d <- sum((y-y.star)^2)

  #Accept/Reject
  if(d < epsilon){
    b.store[i] <- b
  } else{
    b.store[i] <- NA
  }

}

#Get number of reject samples
sum(is.na(b.store))
```

```
## [1] 49993
```

```
#Plot Approximate Posterior
hist(b.store, freq = FALSE, xlab = expression(beta), main = "")
abline(v = 2, col = 'red')
```



```r
mean(b.store, na.rm = TRUE)
```

```
## [1] 1.831118
```

```r
quantile(b.store, c(0.025, 0.975), na.rm = TRUE)
```

```
##     2.5%    97.5%
## 1.181349 2.587056
```

Example 6.7. And now again with $\varepsilon = 2$. In this case, almost all of the proposals are accepted.

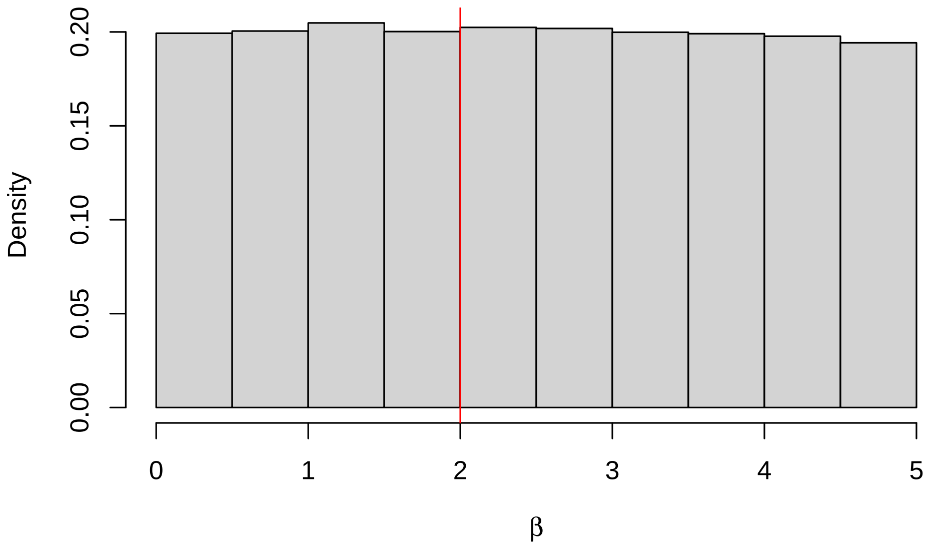
```r
#Set Up Example
set.seed(1234)
n <- 10
y <- rbeta(n, 3, 2)
y
```

```
##  [1] 0.8519237 0.5286251 0.3126172 0.9691679 0.4883547 0.4677043 0.7339799
##  [8] 0.7279578 0.7317827 0.7971786
```

```r
#Set Up ABC
n.iter <- 50000
b.store <- numeric(n.iter)
epsilon <- 2

#Run ABC
for(i in 1:n.iter){

  #Propose new beta
  b <- runif(1, 0, 5)
  #Simualate data
  y.star <- rbeta(n, 3, b)

  #Compute statistic
  d <- sum((y-y.star)^2)

  #Accept/Reject
  if(d < epsilon){
    b.store[i] <- b
  } else{
    b.store[i] <- NA
  }

}

#Get number of reject samples
sum(is.na(b.store))
```

```
## [1] 471
```

```
#Plot Approximate Posterior
hist(b.store, freq = FALSE, xlab = expression(beta), main = "")
abline(v = 2, col = 'red')
```



```
mean(b.store, na.rm = TRUE)
```

```
## [1] 2.488073
```

```r
quantile(b.store, c(0.025, 0.975), na.rm = TRUE)
```

```
##      2.5%     97.5% 
## 0.1281937 4.8666030
```

When $\varepsilon = 0.12$, almost all the proposals are rejected. Although the approximate posterior mean is close to the true value, given we only have 7 samples we cannot say much about the posterior distribution. When $\varepsilon = 2$, almost all the proposals are accepted. This histogram shows that we are really just sampling from the prior distribution, i.e. $\pi_2(\beta \mid y) \approx \pi(\beta)$.

Proposition 6.4. *Using an ABC rejection algorithm*

$$\lim_{\varepsilon \to \infty} \pi_\varepsilon(\theta \mid y) \overset{D}{=} \pi(\theta),$$

and

$$\lim_{\varepsilon \to 0} \pi_\varepsilon(\theta \mid y) \overset{D}{=} \pi(\theta \mid y).$$

This example and the proposition show that if we set ε too large, we don't learn anything about θ, we just recover the data. The smaller the value of ε, the better. But very small values may require very long run times, or have such few samples that the noise from the sampling generator is larger than the signal in the accepted samples. The only diagnostic tools we have are the proportion of samples accepted and the histograms of the approximate posterior and prior distributions.

Example 6.8. An example of where this is useful is epidemic modelling. Suppose we have a population of 100 individuals and at each time point an individual is Susceptible to a disease, Infected with the disease, or Recovered and therefore immune. Once infected with the disease, an individual infects people according to a Poisson process with rate β. Each infected person is infected from a time period drawn from an Exponential distribution with rate 1. We observe the total number of people infected with the disease at the end of the outbreak. To carry out inference for the infection rate β, we need to use the augmented likelihood function The augmented likelihood function for this model is given by

$$\pi(\mathbf{i}, \mathbf{r} | \beta) \propto \underbrace{\exp\left(-\sum_{j=1}^{n}\sum_{k=1}^{N} \beta((r_j \wedge i_k) - (i_j \wedge i_k))\right)}_{\text{Avoiding infection}} \times \underbrace{\prod_{\substack{j=1 \\ j \neq \kappa}}^{n} \left(\sum_{k \in \mathcal{Y}_j} \beta\right)}_{\text{Becoming infectious}} \times \underbrace{\prod_{j=1}^{n} \pi(r_j - i_j | \gamma = 1)}_{\text{Remaining infected}}.$$

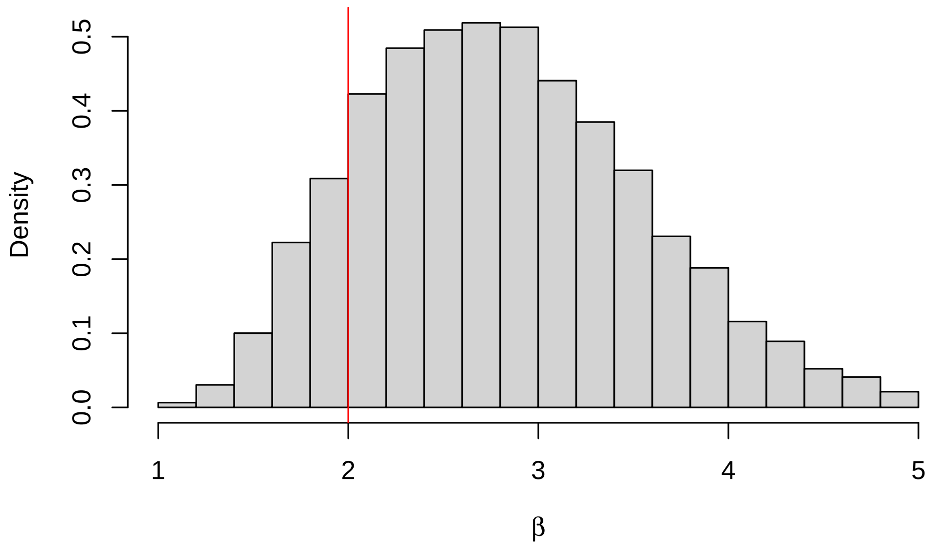
This likelihood function cannot be evaluated as we do not observe the infection time i or recovery time r. Instead we can use ABC sampling with rejection. Each iteration, sample a value for β, simulate an outbreak and compare the observed and simulated number of people infected. If they are ‘close’ we accept the value for β as a sample from our posterior distribution. This means all we have to do is simulate outbreaks.

```
#This function simualtes an outbreak of a disease in a population of size N, with infe
simSIR.Markov <- function(N, beta, gamma) {

  # initial number of infectives and susceptibles;
  I <- 1
  S <- N-1;

  # recording time;
  t <- 0;
  times <- c(t);

  # a vector which records the type of event (1=infection, 2=removal)
  type <- c(1);

  while (I > 0) {

    # time to next event;
    t <- t + rexp(1, (beta/N)*I*S + gamma*I);
    times <- append(times, t);

    if (runif(1) < beta*S/(beta*S + N*gamma)) {
      # infection
      I <- I+1;
      S <- S-1;
      type <- append(type, 1);
    }
    else {
      #removal
      I <- I-1
      type <- append(type, 2);
    }
  }


  # record the times of events (infections/removals), the type of the event, the final
  # the initial infective) and the duration.

  res <- list("t" = times, "type" = type, "final.size" = sum(type==1), "duration" = t,
  return(res)
}

#Set Up Example
```

```
set.seed(1234)
n <- 200
y <- simSIR.Markov(100, 2, 1)$final.size


#Set Up ABC
n.iter <- 50000
b.store <- numeric(n.iter)
epsilon <- 250

#Run ABC
for(i in 1:n.iter){

  #Propose an infection rate
  b <- runif(1, 0, 5)

  #Simualte an outbreak
  y.star <- simSIR.Markov(100, b, 1)$final.size

  #Compute difference
  d <- sum((y-y.star)^2)

  #Accept/Reject
  if(d < epsilon){
    b.store[i] <- b
  } else{
    b.store[i] <- NA
  }

}

#Get number of reject samples
sum(is.na(b.store))
```

```
## [1] 39166
```

```
#Plot Approximate Posterior
hist(b.store, freq = FALSE, xlab = expression(beta), main = "")
abline(v = 2, col = 'red')
```



```r
mean(b.store, na.rm = TRUE)
```

```
## [1] 2.811462
```

```r
quantile(b.store, c(0.025, 0.975), na.rm = TRUE)
```

```
##     2.5%    97.5%
## 1.589679 4.367686
```

6.3.2 Summary ABC with Rejection

ABC with rejection suffers from the curse of dimensionality (see Chapter 5). As the number of data points increases, the probability we get a ‘close’ match decreases. This means we have to increase ε and degrade the quality of our approximation.

Example 6.9. Let’s repeat the Beta example with $n = 200$ observed data points. We need $\varepsilon > 15$ for any proposals to be accepted.

We can avoid the curse of dimensionality by comparing summary statistics instead. This leads us to the Summary ABC algorithm.

Definition 6.14. To carry out inference for a parameter θ using an Summary Approximate Bayesian Computation algorithm with rejection

1. Sample a value for the parameter θ^* from the prior distribution $\pi(\theta)$.
2. Generate some data $y*$ from the data generating process using the parameter value θ^*.
3. Accept θ^* as a value from the posterior distribution if $||S(y) - S(y^*)|| < \varepsilon$ for some $\varepsilon > 0$ and summary summary statistic S. Otherwise reject θ^*
4. Repeat steps 1 - 3.

Similar to the ABC algorithm with rejection, we also have the following proposition.

Proposition 6.5. *The approximate posterior distribution using ABC with rejection is*

$$\pi_\varepsilon(\theta \mid S(y)) \propto \int \pi(y^* \mid \theta^*)\pi(\theta^*) I_{A_\varepsilon(y^*)} dy^*,$$

where $A_\varepsilon(y^*) = \{y^* \mid ||S(y^*) - (y)|| < \varepsilon\}$.

Using summary statistics only increases the approximation however, as we are approximating the data using a summary of it. The only case when we are not approximating further is when the statistic contains all the information about the underlying sample it is summarising. This is known as a sufficient statistic.

Definition 6.15. A statistic S is a sufficient statistic for the parameter θ if the conditional distribution $\pi(y|S(y))$ does not depend on θ.

Proposition 6.6. *Using a Summary ABC rejection algorithm with a sufficient statistic S*

$$\lim_{\varepsilon \to 0} \pi_\varepsilon(\theta \mid S(y)) \overset{D}{=} \pi(\theta \mid y).$$

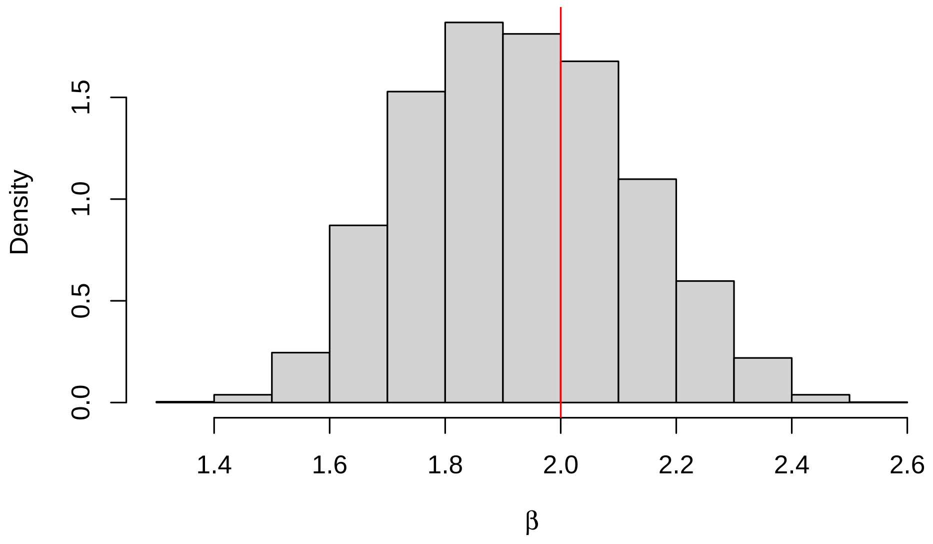
The difficulty with sufficient statistics is that they only exist for 'nice' distributions, like the Gamma, Beta and Poisson distributions. In these cases, we can work with the posterior distribution directly or use and MCMC algorithm.

Example 6.10. Let's repeat the beta distribution example using the mean as the summary statistic. We can set $\varepsilon = 0.001$.

```r
set.seed(1234)
n <- 200
y <- rbeta(n, 3, 2)


n.iter <- 50000
b.store <- numeric(n.iter)
epsilon <- 0.001
for(i in 1:n.iter){

  b <- runif(1, 0, 5)

  y.star <- rbeta(n, 3, b)

  d <- sum((mean(y)-mean(y.star))^2)

  if(d < epsilon){
    b.store[i] <- b
  } else{
    b.store[i] <- NA
  }

}

#Get number of reject samples
sum(is.na(b.store))
```

```
## [1] 45028
```

```
#Plot Approximate Posterior
hist(b.store, freq = FALSE, xlab = expression(beta), main = "")
abline(v = 2, col = 'red')
```



```
mean(b.store, na.rm = TRUE)
```

```
## [1] 1.930908
```

```
quantile(b.store, c(0.025, 0.975), na.rm = TRUE)
```

```
##     2.5%    97.5%
## 1.593348 2.302256
```

6.4 Lab

6.4.1 Gaussian Processes

Exercise 6.1. Code up example 6.1. How does your choice of length scale affect the posterior distribution. You can use

```
x <- -5:5
y <-  c(3.0942822, 3.0727920, 2.6137341, 1.8818820, 1.2746738, 1.2532116, 1.4620830, 1.4194647, 1
```

with $\sigma^2 = 0.2$.To draw samples from the multivariate normal distribution with mean vector μ and covariance matrix Σ

```
mvnorm.chol <- function(mu, Sigma){
  #Multivariate Normal Sampler with Cholesky Input
  #Inputs: mu -- mean, chol -- covariance matrix
  Sigma.chol <- chol(Sigma + 0.0001*dim(Sigma)[1])
  return(mu + t(Sigma.chol)%*%rnorm(length(mu)))
}
```

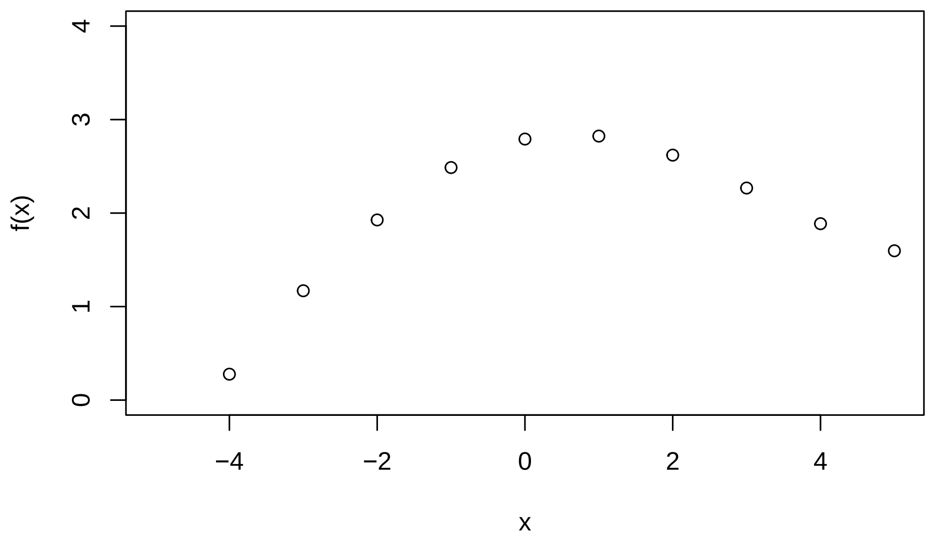
Exercise 6.2. Repeat Exercise 6.1, but this time set the fine grid to be $x^* = \{-5, -4.9, -4.8, \ldots, 9.8, 9.9, 10\}$. What happens to the posterior distribution after $x^* = 5$?

Exercise 6.3. You observe the following data

```
x <- -5:5
y <- cos(0.5*x) + log(x + 6)
y
```

```
##  [1] -0.8011436  0.2770003  1.1693495  1.9265967  2.4870205  2.7917595
##  [7]  2.8234927  2.6197438  2.2679618  1.8864383  1.5967517
```

```
plot(x, y, xlab = "x", ylab = "f(x)", ylim = c(0, 4))
```



Fit a function to the data using a GP prior distribution. Note that this time there is no noise.

6.4.2 Missing Data

Exercise 6.4. In Example 6.2, suppose the observed data is $\{y_1, y_2, y_3, y_4\} = \{4, 4, 5, 2\}$. Design and code an MCMC algorithm to generate samples from the posterior distribution for p and y_5.

Exercise 6.5. Suppose you manage a clinical trial. You administer a new drug to patients and record how many days until their symptoms are alleviated. You observe the times for the first 9 patients

```
x <- c(33,  17, 218,   3,  39,   3,  43,  14,  20)
```

Patient 10 drops out of the trial on day 50 and at this point, their symptoms have not changed. They send an email on day 200 to say they no longer have any symptoms (i.e. $x_i \in [50, \ldots, 200]$. Write down a model for this problem and derive the posterior distribution. Design and code an MCMC algorithm to generate samples from the posterior distribution for any model parameters and x_{10}.

6.4.3 Approximate Bayesian Computation

Exercise 6.6. You observe the following data from an $N(5, \sigma^2)$ distribution.

```
-5.93,  33.12, -21.41, -12.42, -17.64,  -5.47, -27.95, -22.25, -20.40, -26.28, -24.57,
3.06,  44.28, 6.02, -21.14,  14.79, -15.10, 53.18,  38.61,   5.71
```

Use an Exp(0.1) prior distribution on σ^2 and develop a summary statistic ABC algorithm to draw samples from the approximate posterior distribution.

Exercise 6.7. You observe the following data from an $Exp(\lambda)$ distribution.

```
2.6863422, 8.8468112, 8.8781831, 0.2712696, 1.8902442
```

Use an $Beta(1, 3)$ prior distribution on λ and develop an ABC algorithm to draw samples from the approximate posterior distribution. Write the ABC algorithm as a function so you can run it for different values of ε. Run your algorithm for $\varepsilon = \{20, \ldots, 100\}$ and record the approximate posterior median. Plot the relative error in your estimate against the true value of lambda, which is 0.2.