MAS61006 Bayesian Statistics and Computational Methods

Semester 2: Computational Methods

Jeremy Oakley

2023-04-24

Contents

Acknowledgement

These lecture notes were mostly written by Dr Alison Poulston in 2020/21. I made a few small contributions to the original version, and have made a few changes each year since.

Literature

You are not expected to buy or read textbooks for this part of the module, but if you do wish to read around the subject, some suggestions are as follows.

- A Gelman, J B Carlin, H S Stern, D B Dunson, A Vehtari and D B Rubin (2013) *Bayesian Data Analysis (3rd Edition)*, Chapman and Hall/CRC.
- B Lambert (2018) *A Student's Guide to Bayesian Analysis*, SAGE Publications.
- M Betancourt (2018) *A Conceptual Introduction to Hamiltonian Monte Carlo*, ArXiv.
- S Brooks, A Gelman, G L Jones and X Meng (2011) *Handbook of Markov Chain Monte Carlo*, Chapman and Hall/CRC.
- J J Faraway (2016) *Extending the Linear Model with R: Generalised Linear, Mixed Effects and Nonparametric Regression Models (2nd Edition)*, Chapman and Hall/CRC.
- R J A Little and D B Rubin (2020) *Statistical Analysis with Missing Data (3rd Edition)*, Wiley.
- Zhou and Reiter (2010) *A Note on Bayesian Inference After Multiple Imputation*, The American Statistician.

Part I

Inference using advanced samplers

1 MCMC Sampling Recap

Aims of this chapter
1. Revision of Markov Chain Monte Carlo for practical Bayesian inference.
2. In particular, revise the random walk Metropolis-Hastings.
3. Appreciate the limitations of random walk Metropolis-Hastings as an inference tool.

1.1 Bayesian inference continued

We will continue the Bayesian approach to statistical inference that we studied in Semester 1. In particular, given data x and an unknown parameters of interest θ, we derive the posterior distribution $f(\theta|x)$ so that, for example, if we want to consider our probability of θ lying in some set R, we would compute

$$P(\theta \in R|x) = \int_R f(\theta|x)d\theta.$$

Given our prior distribution $f(\theta)$ and likelihood $f(x|\theta)$, the integral above can be computed via

$$\int_R f(\theta|x)d\theta = \frac{\int_R f(\theta)f(x|\theta)d\theta}{\int_\Theta f(\theta)f(x|\theta)d\theta},$$

where Θ is the sample space of θ. Often, however, we cannot evaluate these integrals analytically (we usually won't have conjugate priors we can use). The parameter θ may be high-dimensional, and so to use Bayesian inference in practice, we need a reliable way to compute (estimate) high-dimensional integrals.

1.2 Monte Carlo estimation

Suppose we are interested in an integral

$$I = \mathbb{E}(g(X)) = \int g(x)f(x)\mathrm{d}x.$$

Let $X_1, X_2, \ldots, X_n$ be independent random variables with pdf $f(x)$. Then a Monte Carlo approximation to I is

$$\hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} g(X_i). \tag{1}$$

The main idea in Monte Carlo integration is to approximate I by $\hat{I}_n$.

For example, suppose we wish to find the theoretical mean

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x f_X(x) \mathrm{d}x,$$

then we can approximate the theoretical mean by the sample mean

$$\mathbb{E}(X) \approx \frac{1}{n} \sum_{i=1}^{n} X_i,$$

and this is the same as the Monte Carlo estimate.

1.2.1 Properties of the Monte Carlo estimate

- $\hat{I}_n$ is an unbiased estimator of I.
- $\hat{I}_n$ converges to I as $n \to \infty$ according to the strong law of large numbers (SLLN).
- Although we know that $\hat{I}_n$ converges, we do not know how fast. We need to know how large n must be to achieve a certain error. The variance of the Monte Carlo estimate is given by

  $$\mathbb{E}[(\hat{I}_n - I)^2] = \frac{\sigma^2}{n},$$

  where $\sigma^2 = \mathrm{Var}(g(X))$. Thus the 'root mean square error' (RMSE) of $\hat{I}_n$ is

  $$\mathrm{RMSE}(\hat{I}_n) = \frac{\sigma}{\sqrt{n}} = O(n^{-1/2}).$$

  Thus, our estimate is more accurate as $n \to \infty$, and is less accurate when σ^2 is large. σ^2 will usually be unknown, but we can estimate it:

  $$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (g(X_i) - \hat{I}_n)^2.$$

  We call $\frac{\hat{\sigma}}{\sqrt{n}}$ the Monte Carlo standard error.

  We write

  $$\mathrm{RMSE}(\hat{I}_n) = O(n^{-1/2}),$$

  to emphasise the rate of convergence of the error with n. To get 1 digit more accuracy requires a 100-fold increase in n. A 3-digit improvement would require us to multiply n by 10^6. Consequently Monte Carlo is not usually suited for problems where we need a very high accuracy. Although the error rate is low (the RMSE decreases slowly with n), it has the nice properties that the RMSE:

  - Does not depend on $d = \dim(x)$,
  - Does not depend on the smoothness of f.
    Consequently Monte Carlo is very competitive in high dimensional problems that are not smooth.

- In addition to the rate of convergence, the central limit theorem tells us the asymptotic distribution of $\hat{I}_n$:

$$\frac{\sqrt{n}(\hat{I}_n - I)}{\sigma} \to N(0,1) \text{ in distribution as } n \to \infty.$$

  Informally, $\hat{I}_n$ is approximately $N(I, \frac{\sigma^2}{n})$ for large n. This allows us to calculate confidence intervals for I.

1.2.2 Expressing quantities as expectations

In many statistical inference problems, we are interested in estimating some descriptive variable, such as a probability. It is useful to us that we can often write these quantities in terms of expectations, and apply our Monte Carlo estimate appropriately.

- For example, a variance can be expressed as two expectations:

$$\text{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2. \tag{2}$$

- A probability $\mathbb{P}(X < a)$ can be expressed as an expectation: the expectation of an indicator function of X. An indicator function $\mathbb{I}(E)$ of an event E is defined as

$$\mathbb{I}(E) = \begin{cases} 1 & E \text{ is true} \\ 0 & E \text{ is false} \end{cases} \tag{3}$$

  Then we have

$$\begin{aligned} \mathbb{E}\{\mathbb{I}(X < a)\} &= 1 \times \mathbb{P}(X < a) + 0 \times \mathbb{P}(X \geq a) && (4) \\ &= \mathbb{P}(X < a). && (5) \end{aligned}$$

- A percentile of the distribution of a random variable X can be estimated by taking the sample percentile from the generated sample of values $X_1, \ldots, X_n$. Informally, we would expect the estimate to be more accurate as n increases. Determining a percentile is equivalent to inverting the distribution function; if for example we wish to know the 95th percentile, we must find ν such that

$$\mathbb{P}(X \leq \nu) = 0.95, \tag{6}$$

  so the more accurately we estimate the distribution function $F(X)$, the more accurate we would expect the estimate of any particular percentile to be.

1.2.3 Estimation of general integrals

In general, let the integral of interest be

$$R = \int f(x) \mathrm{d}x. \tag{7}$$

Suppose that $g(x)$ is some density function that we can easily produce a sample of values $X_1, \ldots, X_n$ from. Then

$$R = \int \frac{f(x)}{g(x)} g(x) \mathrm{d}x \tag{8}$$

$$= \int h(x) g(x) \mathrm{d}x, \tag{9}$$

with $h(x) = \frac{f(x)}{g(x)}$. So we now have

$$R = \mathbb{E}\{h(X)\}, \tag{10}$$

where X has the density function $g(x)$. If we now sample $X_1, \ldots, X_n$ from $g(x)$, then evaluate $h(X_1), \ldots, h(X_n)$, then

$$\hat{R} = \frac{1}{n} \sum_{i=1}^{n} h(X_i), \tag{11}$$

is an unbiased estimator of R.

1.3 Markov chain Monte Carlo (MCMC)

In Bayesian statistics, we are faced with the problem that we cannot often derive the posterior distribution analytically. We can, however, write down the posterior distribution up to proportionality, simply by multiplying the prior and likelihood

$$f(\theta|x) \propto f(\theta) f(x|\theta).$$

If we can somehow produce a sample $\theta_1, \ldots, \theta_n$ from the posterior distribution then, using the Monte Carlo ideas described in the previous section, we can obtain any posterior summary of θ that we want, and to arbitrary accuracy given a sufficiently large sample. One way to obtain this sample is through Markov chain Monte Carlo methods.

Recall that a sequence of random variables $X_1, \ldots, X_n$ is an instance of a Markov chain if the transition kernel $\mathbb{P}(X_{t+1}|X_t)$ satisfies the Markov property, i.e. that

$$\mathbb{P}(X_{t+1}|X_1, \ldots, X_t) = \mathbb{P}(X_{t+1}|X_t),$$

for all $t > 0$. If a Markov chain is aperiodic and irreducible, then it has a unique stationary distribution, π, such that $\lim_{t\to\infty} \mathbb{P}(X_t = i) = \pi_i$.

Our aim for Bayesian inference therefore is to specify a transition kernel for a Markov chain that allows us to sample $\theta_1, \ldots, \theta_n$ such that the stationary distribution is our posterior. The important algorithm that you were introduced to in Semester 1 to achieve this task is the Metropolis-Hastings algorithm, which we recap next.

1.3.1 The Metroplis-Hastings (MH) algorithm

We will revisit the MH algorithm for generating a sample $\theta_1, \ldots, \theta_n$ from the posterior distribution $f(\theta|x)$.

Given a sample θ_i, we:

1. Simulate the proposal $\theta^* \sim q(\cdot|\theta_i)$.
2. Evaluate
$$m = \min\left\{1, \frac{f(\theta^*)f(x|\theta^*)q(\theta_i|\theta^*)}{f(\theta_i)f(x|\theta_i)q(\theta^*|\theta_i)}\right\}, \tag{12}$$
3. Accept $\theta_{i+1} = \theta^*$ with probability m, and set $\theta_{i+1} = \theta_i$ otherwise.

The *MH acceptance probability*, given by m above, comprises of two parts:

- The ratio of the posterior density between the proposed and current parameter values:
$$\frac{f(\theta^*)f(x|\theta^*)}{f(\theta_i)f(x|\theta_i)},$$
and
- The ratio of the proposal density between the forwards and backwards proposal steps:
$$\frac{q(\theta_i|\theta^*)}{q(\theta^*|\theta_i)}.$$
We'll particularly focus on the concept of the *random walk MH* algorithm, where the proposal distribution is symmetric, i.e. $q(i|j) = q(j|i)$ for all i, j. It is common in practice for this to be the case, with the proposal distribution being Gaussian centred on θ_i. The MH acceptance ratio therefore simplifies to $m = \min\left\{1, \frac{f(\theta^*)f(x|\theta^*)}{f(\theta_i)f(x|\theta_i)}\right\}$, and is dependent only on the ratio of the posterior density.

1.3.2 The problem with random walk proposals

Under the random walk MH scenario, the algorithm amounts to choosing at random a new direction and distance to 'step' away from the current location (given by θ_i) within the parameter space. The shape or magnitude of the posterior density has no basis for how we make this step proposal, only whether or not we accept the proposal once it has been made. If our proposed value, θ^*, has a higher posterior density than the current value θ_i, then we automatically accept the proposal. If not, we have the MH ratio of densities to probabilistically decide on acceptance of the proposal.

The theory of Markov chains tells us that a well designed MCMC sampler, such as the MH algorithm, will eventually converge to its stationary distribution (the posterior). But this theory does not tell us that this convergence will happen in a finite—or more importantly computationally feasible—amount of time/iterations.

Convergence becomes particularly problematic for random walk MH in parameter spaces of high dimension. As dimension increases, the region of the parameter space that contributes significantly towards the posterior distribution function reduces, and it can be hard to design a 'good' proposal distribution for the MH sampler. This is because:

- If the proposed steps are too large, they are almost always rejected. This means the MCMC chains will get 'stuck' in an area of the parameter space.

- If the proposed steps are made very small so that acceptances occur, then the chain move around the parameter space prohibitively slowly. In either of these cases, this can result in an MCMC sample having an effective sample sizes of almost zero and not having explored the full parameter space. In high dimensional parameter spaces, multi-modality and complex correlations between dimensions further exacerbate this problem.

The problems discussed here form the motivation for our next chapter—we aim to define an MCMC sampler that produces samples from our target posterior distribution, but which moves around the parameter space in a more efficient manner than the random walk.

1.3.3 The Gibbs algorithm

An improvement to MCMC algorithms that rely on random walks that you met in Semester 1 was the Gibbs algorithm, which is a special case of the MH sampler. Rather than proposing a step in the parameter space at random, without consideration to the shape of the posterior distribution, the Gibbs algorithm does make proposals based the shape of the posterior.

We first recap the Gibbs algorithm for sampling the posterior of the parameter vector $(\theta_1, \dots, \theta_n)$.

Given $(\theta_{1,t}, \dots, \theta_{n,t})$:

- Choose a random update ordering $\{d_1, \dots, d_n\}$ of the n dimensions, i.e. a permutation of $\{1, \dots, n\}$.

- In the order determined, sample from the conditional posterior for each parameter, using the most up-to-date parameter values. For example:
  - set $\theta_{d_1,t+1}$ equal to a sample from $f(\theta_{d_1}|x, \theta_{d_2,t}, \dots, \theta_{d_n,t})$,
  - set $\theta_{d_2,t+1}$ equal to a sample from $f(\theta_{d_2}|x, \theta_{d_1,t+1}, \theta_{d_3,t}, \dots, \theta_{d_n,t})$,
  - set $\theta_{d_3,t+1}$ equal to a sample from $f(\theta_{d_3}|x, \theta_{d_1,t+1}, \theta_{d_2,t+1}, \theta_{d_4,t}, \dots, \theta_{d_n,t})$,
  - and so forth.

Intuitively, in two dimensions, the Gibbs sampler amounts to exploring the parameter space by taking steps alternately along the horizontal (x) and vertical (y) axes. If the current location is at (x, y) and the next update is in the y direction, then conditional posterior distribution is considered at the fixed point x. Because we sample y from this conditional distribution, we are taking into account the shape of the conditional posterior, and are most likely to move towards a region of high posterior conditional density.

The Gibbs sampler is a special case of the MH sampler, but we do not have to evaluate the acceptance probability given in Equation (12), because the proposal distribution is equal to the conditional posterior we need to consider, and therefore this term simplifies to 1.

1.3.4 The problem with Gibbs

The Gibbs sampler appears to solve our problem of proposing random directions to move around the parameter space. However, this also has a number of disadvantages.

- We need to be able to evaluate the conditional posterior density of every dimension of our parameter vector and be able to sample from it. In practice, this is often not possible, particularly at high dimensions.
- We can only move in a subset of dimensions at a time. This can be slow in high dimensions, and is exacerbated when correlations are high or ridges exist in the posterior. Intuitively, the stepping direction of Gibbs in two dimensions is alternately the horizontal and vertical, so if there is a high correlation between the two dimensions the region of high posterior density sits as a diagonal. Therefore the geometry of the stepping directions is not aligned with the geometry of the posterior.

1.3.5 A solution?

Our ideal situation is to define an MCMC sampler that goes beyond both the random stepping of Metropolis-Hastings and the straight-line movement of Gibbs. Head to the next chapter to see an approach for this.

2 Hamiltonian Monte Carlo (HMC)

Aims of this chapter
1. Introduce the Hamiltonian approach for Metropolis-Hastings proposals.
2. Have an appreciation for *why* Hamiltonian Monte Carlo can be more efficient than Random-walk Metropolis-Hastings.
3. Implement simple examples of Hamiltonian Monte Carlo in R.

Implementing MCMC can be difficult if we can't construct a suitable proposal distribution. In this Chapter, we introduce Hamiltonian Monte Carlo (HMC), which can be thought of as a particular way to generate proposals in each iteration of the Metropolis-Hastings algorithm. The aim is to generate proposals θ^* which

1. can be far from the current values θ^t in the Markov chain;
2. have high acceptance probability.

These two properties should lead to quick and efficient exploration of the posterior distribution: we can produce representative samples from the posterior distribution quickly.

The disadvantage of this method is that much more computational work is needed to obtain the proposed values at each iteration. However, HMC can be implemented using the language `Stan`, and so much of this work will be done for us. `Stan` can be run within R using the package `rstan` (`Stan` interfaces with many other languages such as Python and MATLAB.) `Stan` has a large user community and is very well supported and documented.

2.1 Generating proposals: intuition

As an example, suppose we are trying to sample from a posterior distribution that is a mixture of two normal distributions:

$$\theta|x \sim 0.9 \times N(2, 0.25^2) + 0.1 \times N(6, 0.5^2).$$

This would be easy to sample from directly, but let's think about how to sample from this using a Metropolis-Hastings sampler. In HMC, we use a physical analogy of a ball rolling over a frictionless surface.

Consider the following plot of $-\log f(\theta|x)$.

Imagine that $-\log f(\theta|x)$ represents our frictionless surface. The value of θ describes the **position** of a ball on this surface; given θ the coordinates of the ball are known to be $(\theta, -\log f(\theta|x))$. We flick the ball with random force and direction: we give the ball **momentum**, and wait to see where it is after some duration T.

The ball could end up far from where it started, but gravity will pull the ball towards lower values of $-\log f(\theta|x)$: higher values of posterior density. An occasional flick of sufficient force in the right direction will help the ball move from one 'valley' (mode of the posterior) to another. If the ball

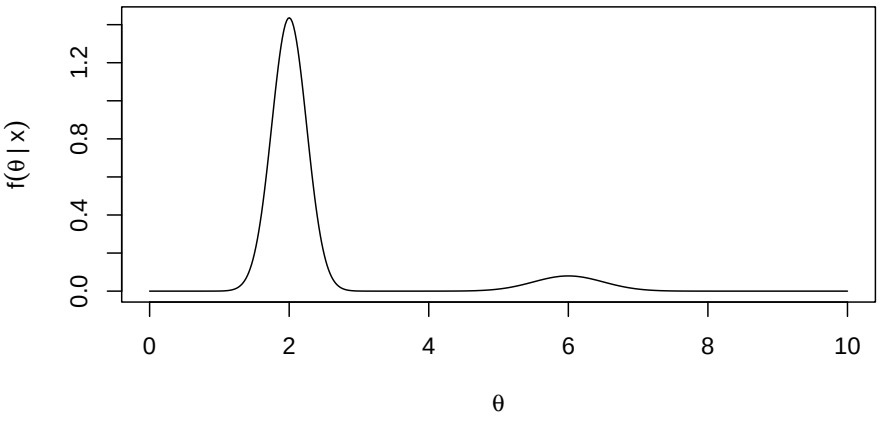


Figure 1: Example: we wish to obtain samples from this posterior distribution.

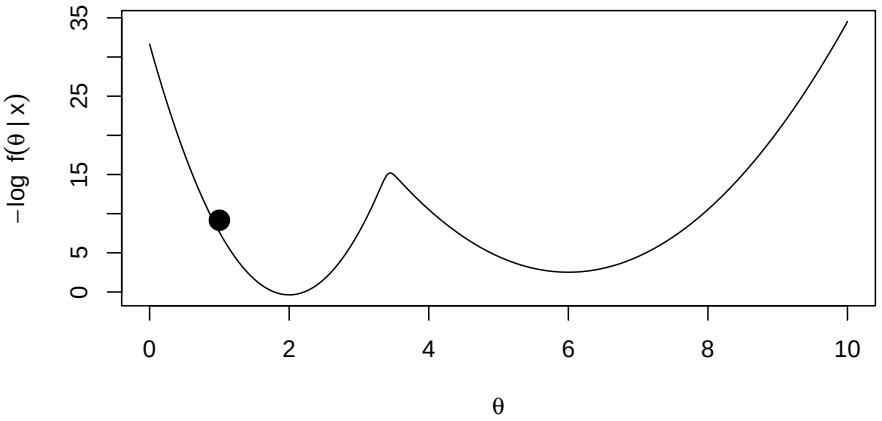


Figure 2: Negative log of our target posterior density. We imagine this function as a surface, with a ball rolling around on top of this surface.

is travelling upwards at either edge (i.e. it is one of the tails of the distribution), it will eventually stop moving, and accelerate back towards the nearest posterior mode.

This is the intuition behind how we will generate proposals, emphasising that we will keep changing the trajectory of the ball by giving it a random flick every T units of time. (It is not yet apparent how this might give proposals with high acceptance probabilities; this will become clear later.)

2.2 Hamiltonian dynamics

In general, for a ball moving over a frictionless surface, given the ball's starting position θ and momentum m, **Hamilton's equations of motion** to tell us how the ball's position and momentum change over time:

$$\frac{d\theta}{dt} = \frac{\partial H(\theta, m)}{\partial m}, \tag{13}$$

$$\frac{dm}{dt} = -\frac{\partial H(\theta, m)}{\partial \theta}, \tag{14}$$

where $H(\theta, m)$ is the **Hamiltonian** and gives the ball's total energy as a function of its position and momentum. You do not need to know how these equations are derived. We will need to solve these equations numerically; we will discuss this shortly.

We will consider Hamiltonians of the form

$$H(\theta, m) = U(\theta) + K(m),$$

where $U(\theta)$ is the **potential energy** of a ball at location θ and encodes information about the surface the ball is travelling over: relatively large $U(\theta)$ means that at position θ, the ball is relatively high above the ground, and so has more potential energy. The term $K(m)$ is the **kinetic energy** of the ball given a momentum of m.

2.2.1 Conservation of energy

Given an initial position θ and momentum m, the Hamiltonian $H(\theta, m)$ is constant over time:

$$\frac{dH(\theta, m)}{dt} = 0,$$

i.e. total energy is conserved. This is the property we are going to exploit to get proposals with high acceptance probability.

2.3 Using Hamilton's equations to generate proposals

An outline of what we will do is as follows.

1. Introduce a dummy variable m, and jointly sample θ and m. (We will discard our samples of m at the end.)

2. Construct a joint probability distribution for θ and m based on the Hamiltonian $H(\theta, m)$. Do this in such a way that the marginal distribution of θ will be the posterior that we wish to sample from.
3. Within a Metropolis-Hastings sampler, make use of Hamilton's equations to obtain a proposed value θ^*, m^* given the current state (θ^t, m^t) in the Markov chain.

2.4 The joint distribution for (θ, m)

We set

$$p(\theta, m) \propto \exp(-H(\theta, m)),$$

where $H(\theta, m)$ is a Hamiltonian given by

$$H(\theta, m) = U(\theta) + K(m),$$

and where we set

$$U(\theta) = -\log f(\theta|x).$$

Note that when evaluating $U(\theta)$ HMC algorithm, it will be sufficient to evaluate $-\log(f(\theta)f(x|\theta))$ only: we only need to know the prior and likelihood.

We also set

$$K(m) = \frac{m^2}{2\sigma^2}$$

It then follows that

$$p(\theta, m) \propto f(\theta|x) \exp\left(-\frac{m^2}{2\sigma^2}\right),$$

from which we see that

- θ and m are independent;
- the marginal distribution of θ is the posterior distribution we want to sample from;
- m has the $N(0, \sigma^2)$ distribution.

2.5 The HMC algorithm

Suppose the current state of the Markov chain is (θ_t, m_t)

1. Discard the current value of the momentum and set m_t to be a new randomly sampled value, sampled directly from its marginal distribution: $N(0, \sigma^2)$.
2. Given the pair (θ_t, m_t), solve (numerically) Hamilton's equations, to obtain a new position and momentum (θ^*, m^{**}) after some fixed time T.
3. Multiply the new momentum by -1: define $m^* = -m^{**}$
4. Accept (θ^*, m^*) as the new location and momentum (θ_{t+1}, m_{t+1}), with probability

$$\min\left\{1, \exp\left\{H(\theta_t, m_t) - H(\theta^*, m^*)\right\}\right\}, \tag{15}$$

$$= \min\left\{1, \exp\left\{U(\theta_t) - U(\theta^*) + K(m_t) - K(m^*)\right\}\right\} \tag{16}$$

$$= \min\left\{1, \frac{f(\theta^*)f(x|\theta^*)p(m^*)}{f(\theta_t)f(x|\theta_t)p(m_t)}\right\}, \tag{17}$$

otherwise set $(\theta_{t+1}, m_{t+1}) = (\theta_t, m_t)$.

Some comments:

- In theory, the acceptance probability will be 1, because of conservation of energy: when we move from (θ_t, m_t) to (θ^*, m^*), the Hamiltonian is unchanged. In practice, the acceptance probability may be slightly less than 1, because there may be numerical errors when we solve Hamilton's equations - we may not find the correct position and momentum after time T.
- We need step 1 to randomly change the momentum and consequently the trajectory of the ball at each iteration, thereby enabling random exploration of the parameter space. Without this step, we would just follow a deterministic trajectory specified by Hamilton's equations.
- Multiplying the momentum by -1 gives proposals that are reversible: applying Hamilton's equations to a starting point $(\theta^*, m^*) = (\theta^*, -m^{**})$ for duration T would get us back to (θ_t, m_t). (This is why there is no ratio of proposal densities in the formula for the acceptance probability: they cancel out.) We illustrate this in the following plots.

2.5.1 Approximate solution of Hamilton's equations

We have a pair of simultaneous equations in Equation (13) describing the location and momentum over time. Step (2) of the HMC algorithm says that given a current location and momentum, we simulate forwards according to our definition for total energy for a set time T. To implement this simulation, we must discretise time, and perform a number of small steps of size ϵ.

You may have previously seen the most common (and simple) approach for approximating the solution to a system of differential equations. This is **Euler's method**, which says $x(t+\epsilon) = x(t) + \epsilon\frac{dx(t)}{t}$. Therefore for our one dimensional scenario in HMC, we would have:

$$m_{t+\epsilon} = m_t - \epsilon\frac{dU(\theta_t)}{d\theta}, \tag{18}$$

$$\theta_{t+\epsilon} = \theta_t + \epsilon\frac{m_t}{\sigma^2}. \tag{19}$$

The error associated with the Euler approach is of the order of ϵ globally, and so in practice for HMC, we use an approximation with error of order ϵ^2 for more stable results. This is called the

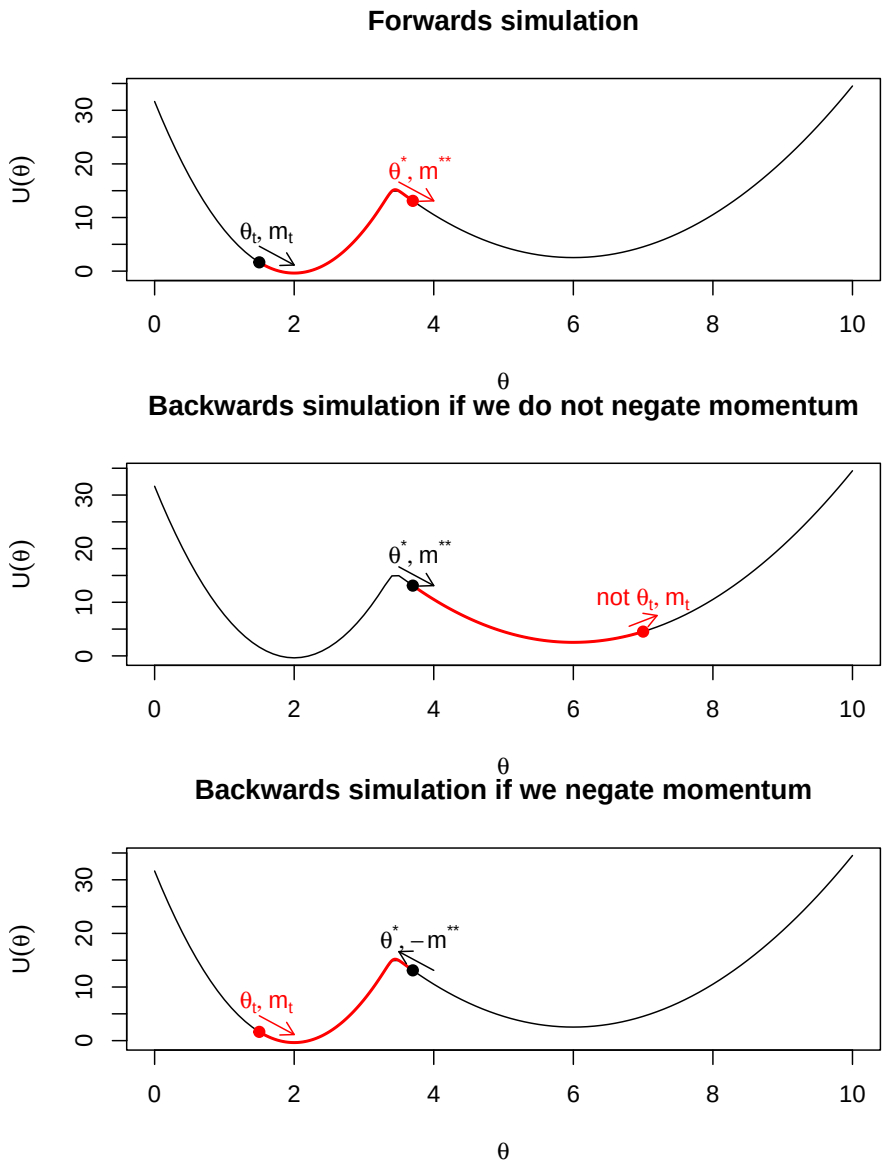


Figure 3: Intuition on why the final momentum is reversed to create a symmetric proposal distribution.

Leapfrog method. We give the details of this for completeness, but will not discuss it in detail. Notice, however, its similarity to the Euler method—we take steps of half the size, but rather than updating both momentum and location simultaneously, we *leapfrog* over the previous update each time. The leapfrog method is:

$$m_{t+\frac{\epsilon}{2}} = m_t - \frac{\epsilon}{2}\frac{dU(\theta_t)}{d\theta}, \tag{20}$$

$$\theta_{t+\epsilon} = \theta_t + \epsilon\frac{m_{t+\frac{\epsilon}{2}}}{\sigma^2}, \tag{21}$$

$$m_{t+\epsilon} = m_{t+\frac{\epsilon}{2}} - \frac{\epsilon}{2}\frac{dU(\theta_{t+\epsilon})}{d\theta}. \tag{22}$$

2.6 Multivariate θ

The plots and description so far have all corresponded to a posterior distribution for a single parameter θ. The extension to multivariate θ is straightforward. For d-dimensional $\theta = (\theta_1, \ldots, \theta_d)$, we introduce a d-dimensional momentum vector $m = (m_1, \ldots, m_d)$. The elements of m are typically chosen to be independent and normally distributed: $m_i \sim N(0, \sigma_i^2)$.

In step 1 of the HMC algorithm, we sample a new momentum vector m^t, with element i sampled from the $N(0, \sigma_i^2)$ distribution. In the Leapfrog method, the momentum and position vectors are updated component-wise:

$$m_{i,t+\frac{\epsilon}{2}} = m_{i,t} - \frac{\epsilon}{2}\frac{\partial U(\theta_t)}{\partial \theta_i}, \tag{23}$$

$$\theta_{i,t+\epsilon} = \theta_{i,t} + \epsilon\frac{m_{i,t+\frac{\epsilon}{2}}}{\sigma_i^2}, \tag{24}$$

$$m_{i,t+\epsilon} = m_{i,t+\frac{\epsilon}{2}} - \frac{\epsilon}{2}\frac{\partial U(\theta_{t+\epsilon})}{\partial \theta_i}. \tag{25}$$

2.7 Tuning parameters

In the HMC algorithm, we have to choose a variance parameter σ^2 for each momentum variable, the time duration T over which we update the position and momentum using Hamilton's equations, and the step size ϵ used in the Leapfrog method.

2.7.1 The tuning parameter σ^2

Recall that the definition of the kinetic energy is $K(m) = \frac{m^2}{2\sigma^2}$, and that the marginal distribution of the kinetic energy is $m \sim N(0, \sigma^2)$.

The size of σ therefore affects how variable the momentum will be, and because the mean is 0, we can think of this as how *large* a momentum we will allow. Returning to our intuitive introduction, if the ball is currently in a trough of the mountainous region that it is exploring, it will need a large momentum to get "up and over" the potential energy gain required to surpass a summit and be able to explore other parts of the parameter space.

2.7.2 The Hamiltonian movement tuning parameters T and ϵ

These two parameters are closely linked. A large ϵ will result in increased errors in the approximation to the Hamiltonian system. Smaller values are therefore ideal, but come at a computational cost.

The choice of T is a more difficult decision that is not necessarily equal for every scenario, in a similar way to the step length variation parameter in the classic random-walk MH algorithm. Large T can lead to us overshooting peaks in the posterior density and missing them entirely, whereas small T may not allow the efficient exploration that we're hoping for. See more on this below when we implement HMC in an example.

Highly complex posteriors can cause problems for the approximation methods, and lead to *divergent iterations*. This is when sharp curvatures lead to poor approximations and can lead to biases in the posterior estimate because the parameter space was not explored in full. It's important to remember that HMC is not a complete solution to difficulties posed by MCMC methods, and the benefits it provides in efficiency come at a computational cost. Further, there are situations where the system cannot be approximated at all, and so HMC cannot be applied.

2.8 Implementing HMC 'by hand'

In this section we will implement a two-dimensional parameter space example, implemented directly in `R` to explore the differences between the three MCMC samplers that you have seen: random-walk MH, Gibbs and HMC. In the next section, we will see how to implement HMC in more complex scenarios using pre-built software—this has a number of benefits such as helping with the issue of tuning parameters.

We'll attempt to sample from a standard distribution, so we can check the results. Suppose our posterior is a bivariate Gaussian:

$$\begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}\Bigg| x \sim N_2\left(\begin{pmatrix}0\\ 0\end{pmatrix}, \begin{pmatrix}1 & 0.95\\ 0.95 & 1\end{pmatrix}\right).$$

2.8.1 Random-walk MH

We will implement a random-walk MH approach to this sampling problem by using a proposal distribution

$$\begin{pmatrix}\theta_1^*\\ \theta_2^*\end{pmatrix}\Bigg| \begin{pmatrix}\theta_{1,t}\\ \theta_{2,t}\end{pmatrix} \sim N_2\left(\begin{pmatrix}\theta_{1,t}\\ \theta_{2,t}\end{pmatrix}, \begin{pmatrix}\sigma_1^2 & 0\\ 0 & \sigma_2^2\end{pmatrix}\right). \tag{26}$$

A function to carry out random-walk MH for this scenario is given here:

```
gaussian_2d_RW <- function(n_iter, sigma, theta) { # sigma and theta 2d vectors

  samples <- matrix(NA, nrow = n_iter, ncol = 6)

  for (i in 1:n_iter) {
```

```
      prop_theta <- rnorm(2, mean = theta, sd = sigma)
      m <- mvtnorm::dmvnorm(prop_theta,
                            mean = c(0, 0),
                            sigma = matrix(c(1,0.95,0.95,1),
                                           nrow = 2),
                            log = T) -
            mvtnorm::dmvnorm(theta,
                             mean = c(0, 0),
                             sigma = matrix(c(1,0.95,0.95,1),
                                            nrow = 2),
                             log = T)

      if (runif(1) < exp(m)) {
         theta <- prop_theta
         samples[i, ] <- c(theta, prop_theta, min(exp(m),1), 1)
      } else {
         samples[i, ] <- c(theta, prop_theta, min(exp(m),1), 0)
      }
   }

   colnames(samples) <- c("theta1","theta2","ptheta1",
                          "ptheta2","m","accept")
   return(tibble::as_tibble(samples))
}
```

Here, we will use a random walk perturbation variance of $\sigma^2 = 0.5^2$, begin our chain at $(\theta_{1,0}, \theta_{2,0}) = (-1, 1)$ and run the chain for 1,000 iterations. This gives an acceptance rate of 0.457 in the instance implemented here.

The results of this sampler are shown over Figures 4-6. We see from the bivariate scatter/density plot in Figure 4 that the target distribution is being captured reasonably well.

We gain insight into the process of the MH algorithm with Figure 5, which shows both the accepted and rejected proposed values. For each proposal, the colour quantifies the acceptance probability, and the shape of the point indicates whether it was accepted in reality. From this plot we can see the guaranteed proposals which lie along the ridge of high density according to the target distribution, but we also see how occasionally we get a proposal accepted that had low probability (e.g. a dark blue circle). We also get an impression of the efficiency of our sampler from such a plot. The high volume of proposed parameter pairs that are shown as having been rejected highlights the wasted computational efforts here.

Finally, in Figure 6 we see the trace plot of θ_1. We see the autocorrelation that exists from an MCMC sampler, and that this particular example is moving around the parameter space slowly. This figure also shows the proposed parameter value at each iteration, which gives an indication of the size of steps that can be taken in this particular random walk.

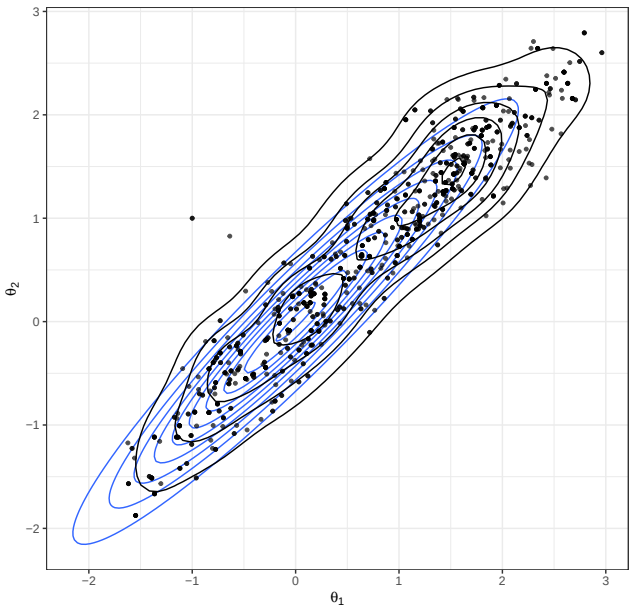


Figure 4: Bivariate Gaussian sampled using a random-walk Metropolis-Hastings algorithm. The true Gaussian density is shown in blue, and overlaid in black with both the samples and estimated density.

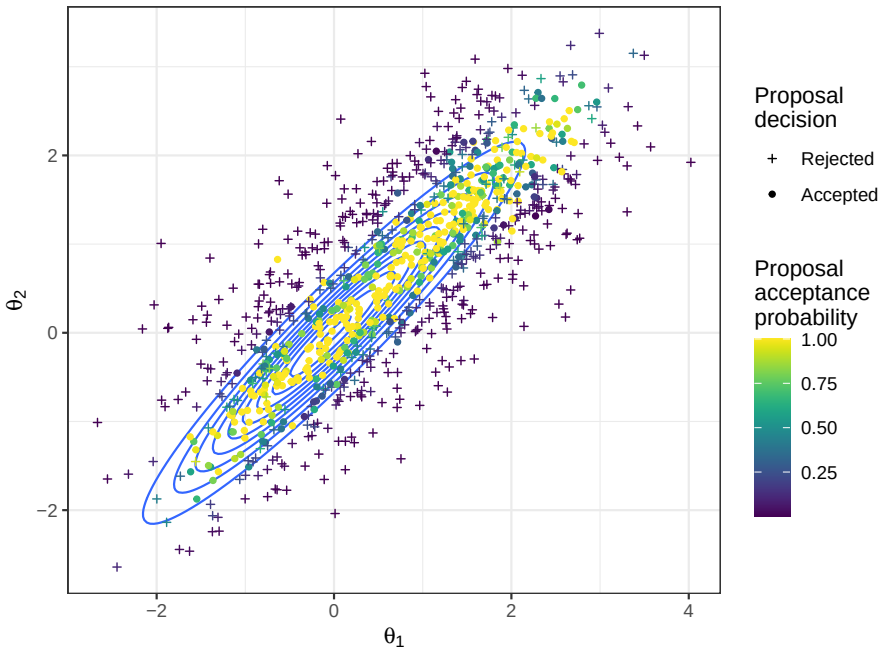


Figure 5: The proposed parameter values during a random-walk Metropolis-Hastings algorithm to sample from a bivariate Gaussian. The true density is shown in blue. The proposed values are coloured according to their acceptance probability when proposed, and the point character gives the resulting acceptance decision.

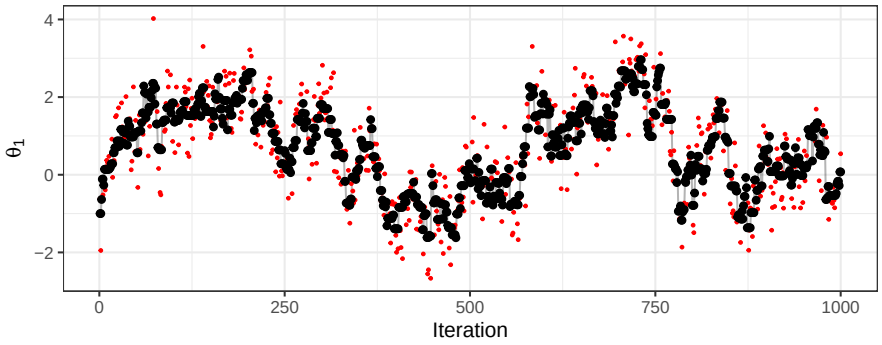


Figure 6: The trace plot for samples of the first dimension of a bivariate Gaussian. Obtained using a random-walk Metropolis-Hastings algorithm. Black points give the accepted sample trace, with the proposed parameter value at each iteration also highlighted in red.

2.8.2 Gibbs

This particular example is easy to implement as a Gibbs sampler, due to conditional distributions of the multivariate Gaussian being Gaussian themselves. Here we show the approach for sampling from the bivariate Gaussian using Gibbs, with 1,000 iterations.

```
gaussian_2d_Gibbs <- function(n_iter, theta) { # theta 2d vectors

   samples <- matrix(NA, nrow = n_iter, ncol = 2)

   for (i in 1:n_iter) {
      theta[1] <- rnorm(1, mean = 0.95 * theta[2], sd = sqrt((1-0.95^2)))
      theta[2] <- rnorm(1, mean = 0.95 * theta[1], sd = sqrt((1-0.95^2)))
      samples[i, ] <- theta
   }

   colnames(samples) <- c("theta1","theta2")
   return(as_tibble(samples))
}
```

Figures 7 and 8 show the results of applying Gibbs. We can see this provides a good approximation to the target distribution. In Gibbs sampling, all proposals are accepted, and so we do not waste computational time. However, note that the ease of sampling from the conditional distribution is fairly rare in practice.

2.8.3 HMC

We now show an implementation to this sampling exercise using HMC. The following code implements HMC for our simple, bivariate Gaussian example. Note that we make use of the function `Deriv::Deriv()` for differentiating the log posterior: this makes a new function which will return the derivatives.

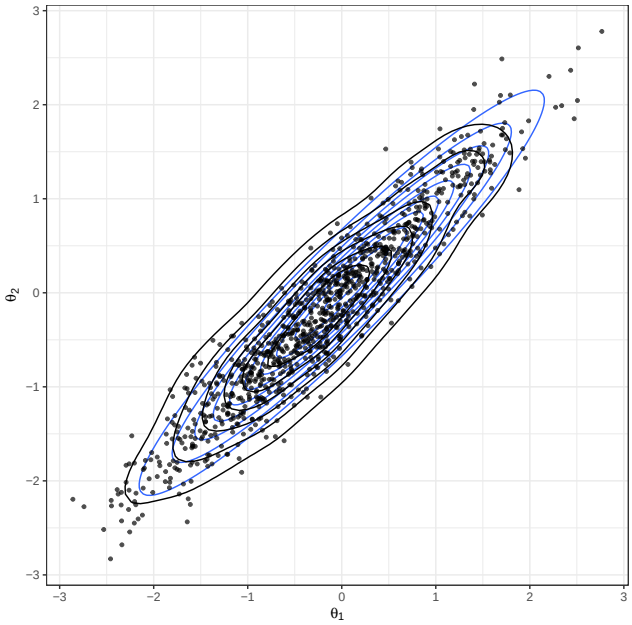


Figure 7: Bivariate Gaussian sampled using a Gibbs algorithm. The true Gaussian density is shown in blue, and overlaid in black with both the samples and estimated density.

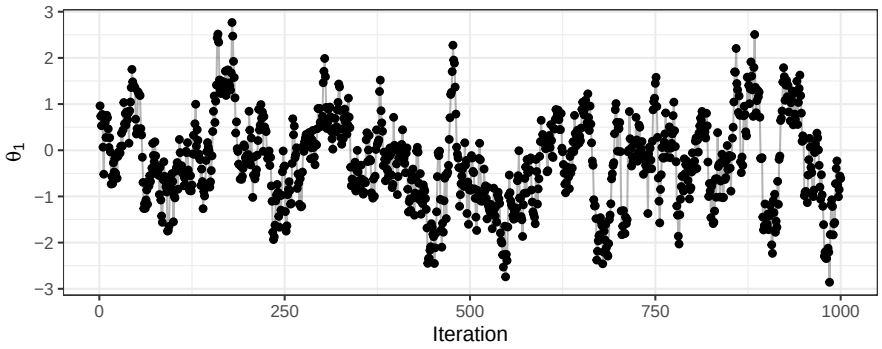


Figure 8: The trace plot for samples of the first dimension of a bivariate Gaussian. Obtained using a Gibbs algorithm.

```r
sigma_inv <- solve(matrix(c(1, 0.95, 0.95, 1), nrow = 2))
U <- function(x,y) (matrix(c(x,y), nrow = 1) %*%
                      sigma_inv %*%
                      matrix(c(x,y), nrow = 2)) / 2
U_grad <- Deriv::Deriv(U)

gaussian_2d_HMC <- function(n_iter, epsilon, L, current_theta) {

   samples <- matrix(NA, nrow = n_iter, ncol = 6)

   for (j in 1:n_iter) {

      theta <- current_theta
      m <- rnorm(2, 0, 1)
      current_m <- m

      # Make a half step for momentum at the beginning
      m <- m - epsilon * U_grad(theta[1], theta[2]) / 2
      for (i in 1:floor(L/epsilon)) {
         # Make a full step for the position
         theta <- theta + epsilon * m
         # Make a full step for the momentum, except at end of trajectory
         if (i != floor(L/epsilon)) m <- m - epsilon * U_grad(theta[1], theta[2])
      }
      # Make a half step for momentum at the end
      m <- m - epsilon * U_grad(theta[1], theta[2])  / 2

      # Negate momentum at end of trajectory to make the proposal symmetric
      m <- -m

      # Evaluate potential and kinetic energies at start and end of trajectory
      current_U <- U(current_theta[1], current_theta[2])
      current_K <- sum(current_m^2) / 2
      proposed_U <- U(theta[1], theta[2])
      proposed_K <- sum(m^2) / 2

      # Accept or reject the state at end of trajectory
      accept_prob <- exp(current_U - proposed_U + current_K - proposed_K)
      if (runif(1) < accept_prob) {
         current_theta <- theta
         samples[j, ] <- c(current_theta, current_theta, min(accept_prob,1), 1)
      } else {
         samples[j, ] <- c(current_theta, theta, min(accept_prob,1), 0)
      }
   }
```

```
    colnames(samples) <- c("theta1","theta2","ptheta1","ptheta2","acceptprob","accept")
    return(as_tibble(samples))
}
```

Here we have implemented HMC with a momentum proposal variance of $\sigma^2 = 1$, a leapfrog step duration of $L = 5$, and a leapfrog step size of $\epsilon = 0.25$. Note that this means our leapfrog method will involve simulating 20 locations to reach the proposal location in each step of the HMC algorithm. The sampler was run for 1,000 iterations, and had an acceptance rate of 0.892. Note that, although not 1 as in Gibbs sampling, this is a very high acceptance rate and about double that of the random-walk MH algorithm we have implemented here.

The HMC sample is shown in Figure 9, and in Figure 10 all proposals are shown, with both their acceptance probability and final outcome. Because of the high acceptance rate, there are very few rejected parameter values. However, we do see that the algorithm is able to explore extreme, low density regions in the parameter space. If this were a multi-modal example, we can imagine this would be useful for moving between modes.

Finally, Figure 11, the trace plot for θ_1 is shown, with sampled values in black, and any rejected proposals shown in red.

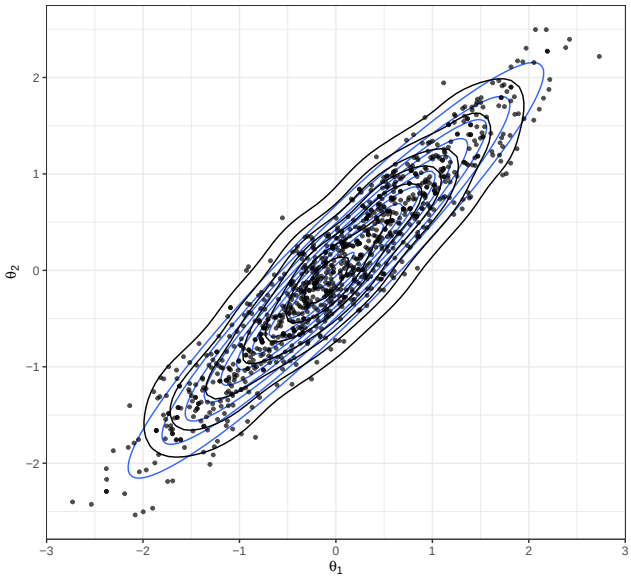


Figure 9: Bivariate Gaussian sampled using a HMC algorithm. The true Gaussian density is shown in blue, and overlaid in black with both the samples and estimated density.

The autocorrelation here is much lower than for the random-walk MH approach. Given that the marginal distribution of θ_1 is a standard normal, this trace plot looks similar to a random sample from this marginal (which would be our gold-standard sampling approach).

We will confirm this with some autocorrelation plots (for θ_1 only)

```
acf(MH$theta1)
acf(Gibbs$theta1)
```

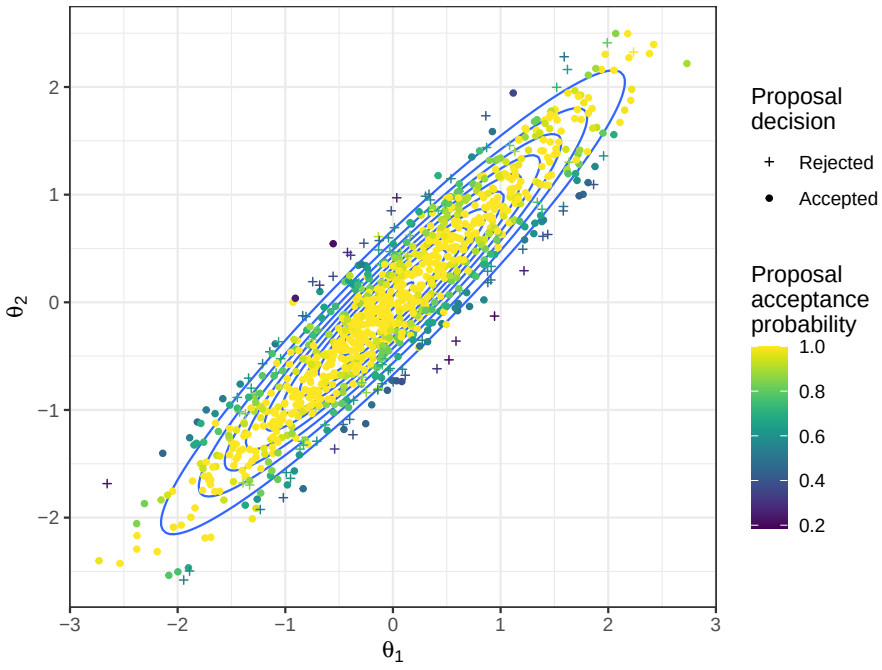


Figure 10: The proposed parameter values during a HMC algorithm to sample from a bivariate Gaussian. The true density is shown in blue. The proposed values are coloured according to their acceptance probability when proposed, and the point character gives the resulting acceptance decision.

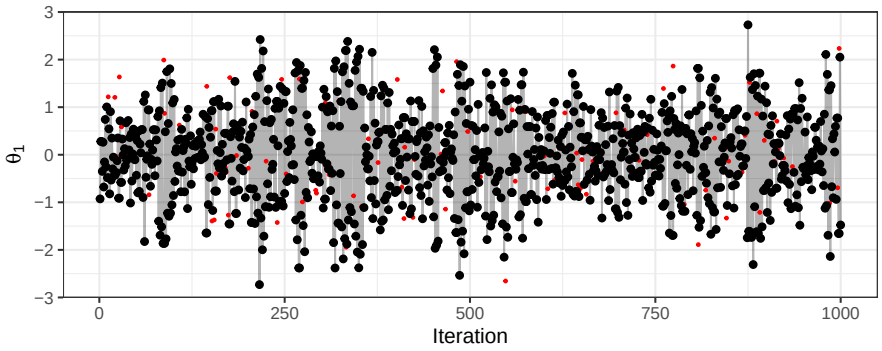


Figure 11: The trace plot for samples of the first dimension of a bivariate Gaussian. Obtained using a HMC algorithm. Black points give the accepted sample trace, with the proposed parameter value at each iteration also highlighted in red.

```
acf(HMC$theta1)
```

2.8.3.1 The leapfrog proposals To gain a deeper understanding of the HMC proposal steps, we can implement the leapfrog movement algorithm to view proposals.

For example, Figure 13 shows the path of the Hamiltonian movement, given an initial location of $(-1.5, -1.55)$, shown in green. The bivariate Gaussian density of the target distribution is shown in blue, and we can see how the concept of an item under gravity applies here. Remember that this is not the path of the HMC algorithm itself, but is the path of the leapfrog algorithm used to generate the next proposal in the HMC algorithm. So here, the current sampled value is that in green, and the value in red will be the proposed parameter value for this iteration.

Note that the resulting path is dependent upon the initial momentum (which we re-sample at the start of each HMC iteration). Here, the initial momentum was (-1,1), which relates to moving left and up in the location space. We see that this is the initial course that the location trajectory takes.

The value of the Hamiltonian is shown, along with its breakdown into the kinetic and potential energy components (remember that $H = U + K$). The constant trade-off between potential and kinetic energy is obvious here. We also see the error we are introducing through implementing the leapfrog algorithm. If our simulation of Hamiltonian movement was exact, the Hamiltonian would be constant. This leapfrog error is why an MH acceptance step must take place.

The bottom panel shows what this MH acceptance probability would be, were we to stop the leapfrog movement at any of these intermediate points and consider them as a proposed location in the HMC algorithm. In this example, the leapfrog path covers a wide range of possible parameter values, yet we can see that the acceptance probability remains above 0.7 throughout. It's important to remember that the acceptance of a proposed location in HMC is not related to its distance from the current location, but is only related to the discrepancy in energy brought about by the leapfrog approximation.

If we repeat this simulation with a smaller ϵ, we get a better approximation to the true Hamiltonian movement. This is shown in Figure 14, where it is clear that given the initial location and momentum, the path is deterministic. At this better approximation, the HMC acceptance probability is much higher (over 0.97) at any location along this trajectory. Remember that HMC is still a Metropolis-Hastings sampler, it just has a different way (the leapfrog trajectory such as in Figure 13) of proposing locations to the random-walk approach. Although this has a higher acceptance probability than we would likely expect from a random-walk proposal, this is clearly much more computationally expensive than a single draw from a bivariate normal proposal distribution.

Note that we can also relate the energy to the leapfrog location path. About halfway along the path is at the mode of the bivariate normal (square, magenta coloured point), and if we consider the potential energy at this time, it is at a minimum (as potential energy is the negative log density).

We can also compare the importance of the initial, re-sampled momentum on the resulting path using Figure 15. This shows the leapfrog path from the same initial location, and for the same length of time, but with an initial starting momentum of (2,2). This equates to moving up and to the right at a higher speed than the previous example. Rather than 'bouncing' back and forth across the space, the imaginary ball has lots of energy to shoot across the space.

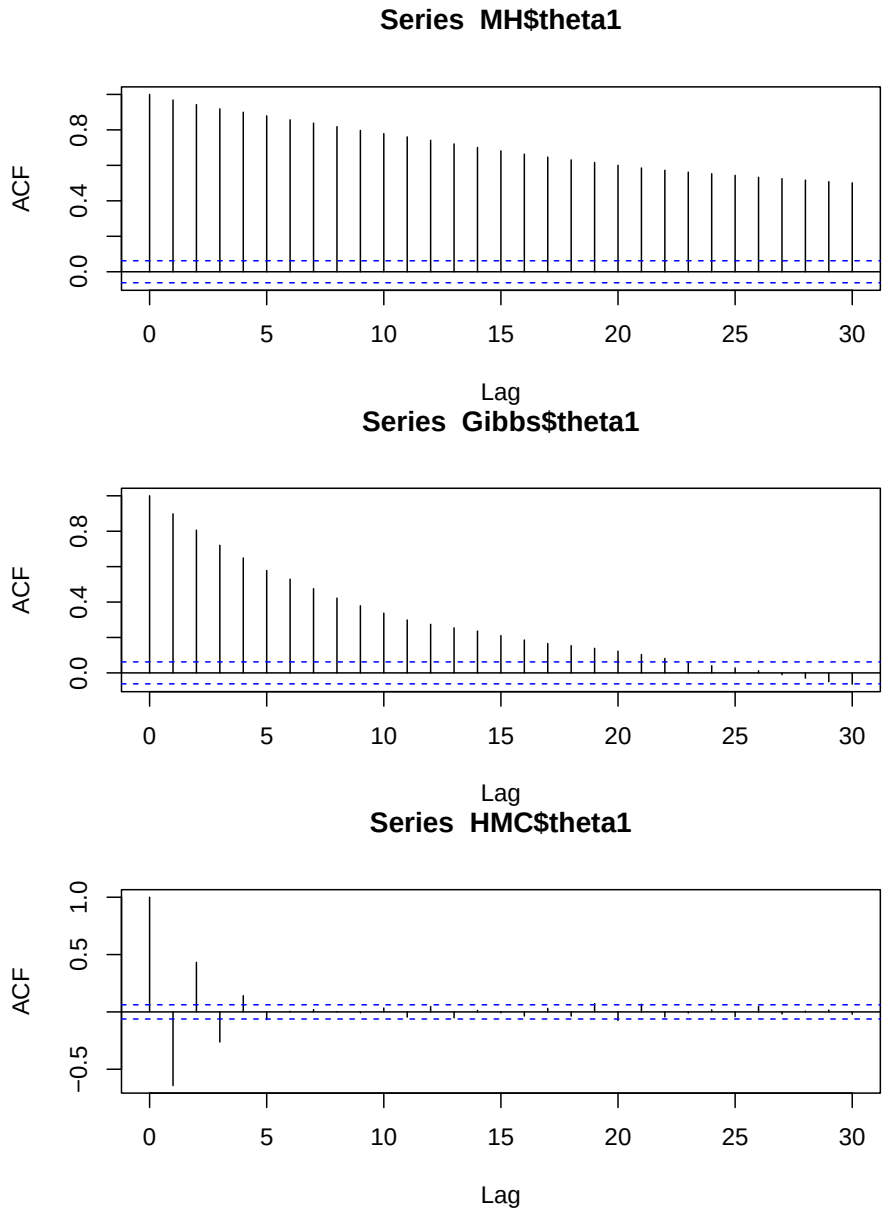


Figure 12: Comparing autocorrelations in the Markov chains produced by random walk MH, Gibbs, and HMC

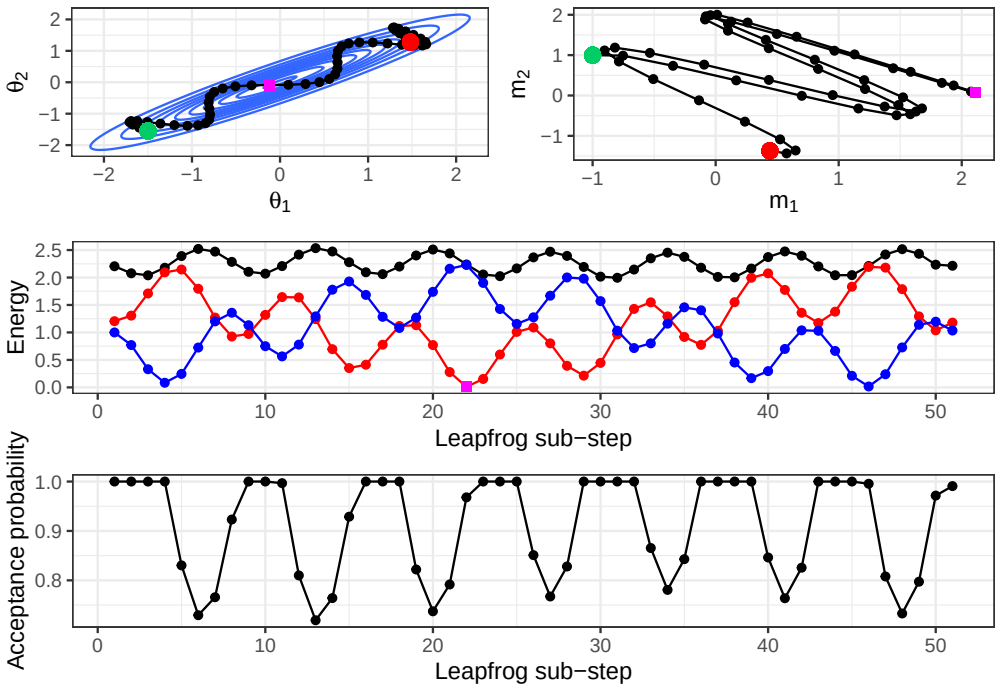


Figure 13: Given a starting location of (-1.5,-1.55) and an initial momentum of (-1,1), the leapfrog algorithm is implemented for 5 'time-units'. This is achieved by using an L of 5 and an epsilon of 0.1, which results in 50 intermediate simulated locations. Note that this is the same 'length' of time as our HMC implementation used, but a smaller step size in the leapfrog algorith, as we used an epsilon of 0.25 there. The resulting path is shown in the parameter space, the momentum space, and the Hamiltonian is also given. The initial location is shown in green and the final location in red in the top set of panels. In the middle panel, the Hamiltonian is shown (black), along with the potential (red) and kinetic (blue) energies. The point in magenta is highlighted, showing its location near the mode of the bivariate normal, and that the corresponding potential energy is low. The bottom panel shows what the acceptance probability would be for each point along the leapfrog path, were that to be the stopping point chosen for the HMC proposal.

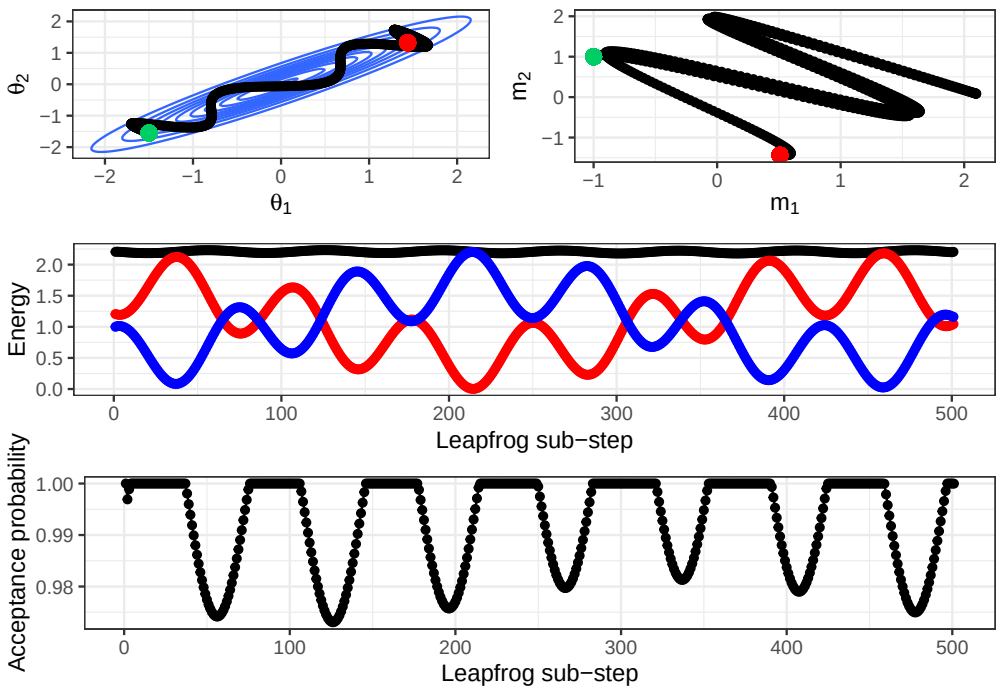


Figure 14: The previous figure re-produced but with an epsilon of 0.01 rather than 0.1, i.e. the Hamiltonian movement is simulated at a closer approximation to the exact process.

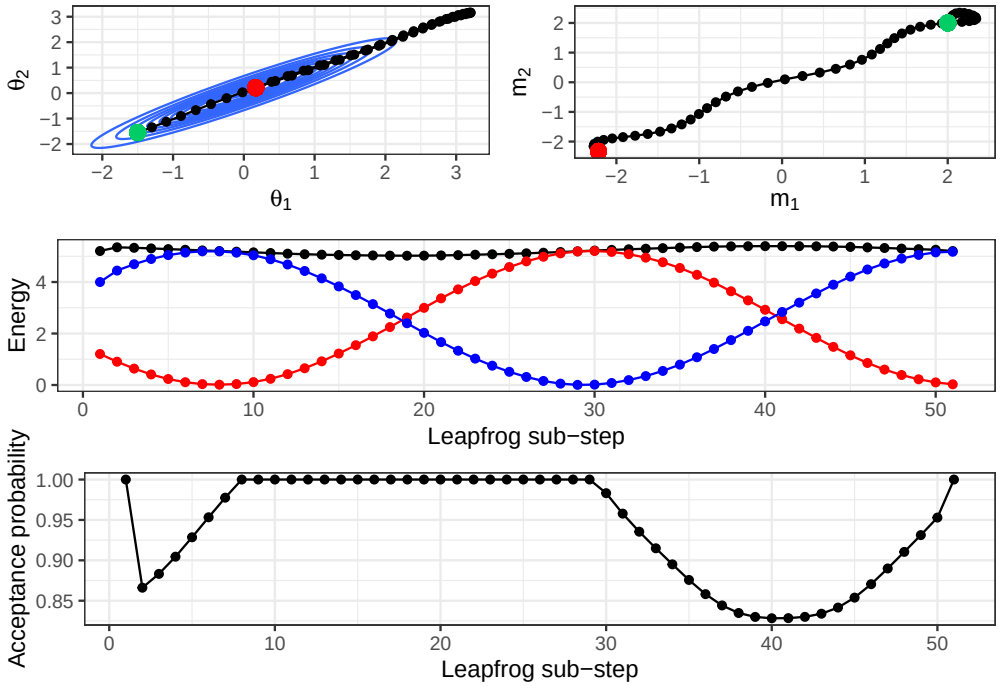


Figure 15: A repeat of the above simulation, but with an initial momentum of (2,2).

2.8.4 Comparing random-walk and HMC samplers

In Figure 16 we compare the movement around the sample space of the random-walk MH and HMC algorithms. Here we have thinned the random-walk samples, only showing every 20 iterations of the random-walk sampler. Recall that the HMC implementation here involved the tuning parameters $L = 5, \epsilon = 0.25$ and so there are 20 intermediate steps taken as part of the leapfrog location proposal. This is still not a direct comparison between the likely change in locations at each step for the two approaches, but is a fair comparison.

We see the key differences in how the proposal processes of the two algorithms produce different sampling schemes. The HMC approach can traverse back and forth across the entire sample space in fewer iterations than the random walk and displays lower autocorrelation in the sampled locations.

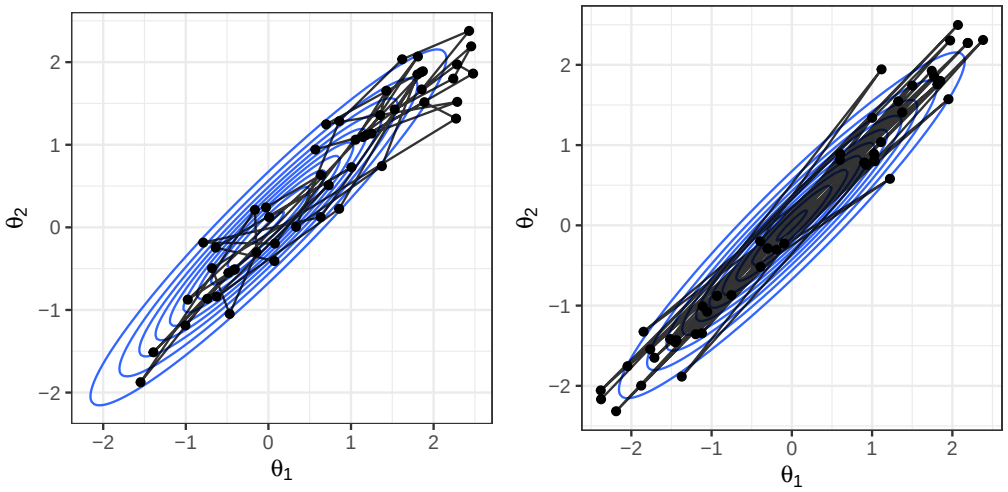


Figure 16: Left: A sample from the random walk MH algorithm, thinned to every 20 samples to give 50 values. Right: A sample of 50 values (not thinned) from the HMC algorithm.

2.9 Summary

In this chapter we've introduced the HMC algorithm as an improvement to the difficulties that random walk MCMC algorithms can pose. In HMC, we use the curvature of the posterior density at our current location to inform on a sensible direction to move in, rather than randomly stepping like in MH. This additional information can often lead to more efficient samplers, but this comes at a computational cost—we must evaluate the derivative of the log posterior density. **There are situations where this will be computationally burdensome, or even impossible.**

3 Implementing HMC in Stan

Aims of this chapter
1. Implement Hamiltonian Monte Carlo using Stan and gain confidence interpreting analyses.

In this Chapter, we will take a first look at using Stan (implemented via the `rstan` package (Stan Development Team, 2021)) to implement HMC. We will also look at some additional R packages which are useful for working with output from `rstan`.

This Chapter is only intended to introduce you to using Stan; it will not cover everything you might need to know. Further reading is discussed at the end of this Chapter.

3.1 Getting set up with Stan

In this course we will use RStudio to write, edit and run Stan code. Please ensure that you have an up-to-date version of Rstudio and R itself on your machine.

You can follow the installation steps at the RStan Getting Started page, but we will also summarise these:

- You will need R version 3.4.0 or later (version 4.0.0 or later is strongly recommended)
- You will need RStudio version 1.2.x or later (version 1.4.x or later is strongly recommended. Note that version numbering system has changed! Version numbers now start with the year: anything starting with 2022 is newer than 1.4.x).
- You need to be able to compile C++ code in R. Windows users will need to install RTools. Mac users may be prompted to install/update Xcode. The above page links to the appropriate option for Windows, Mac and Linux users.
- You need to install the `RStan` package in RStudio.

3.2 `rstan` options

When you run the command

```
library(rstan)
```

you will get some text recommending you set two options:

```
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

We suggest you try both. The first will enable parallel processing on your computer. The second will save your Stan models to a temporary folder on your hard drive, which may save a little time if you find yourself recompiling models in the same session.

3.3 An example model

For our first implementation in Stan, we will introduce a simple model. We have n random variables $Y_1, \ldots, Y_n$ that are assumed to be independent and identically distributed with

$$Y_i|\mu, \sigma^2 \sim N(\mu, \sigma^2). \tag{27}$$

Prior distributions are specified by

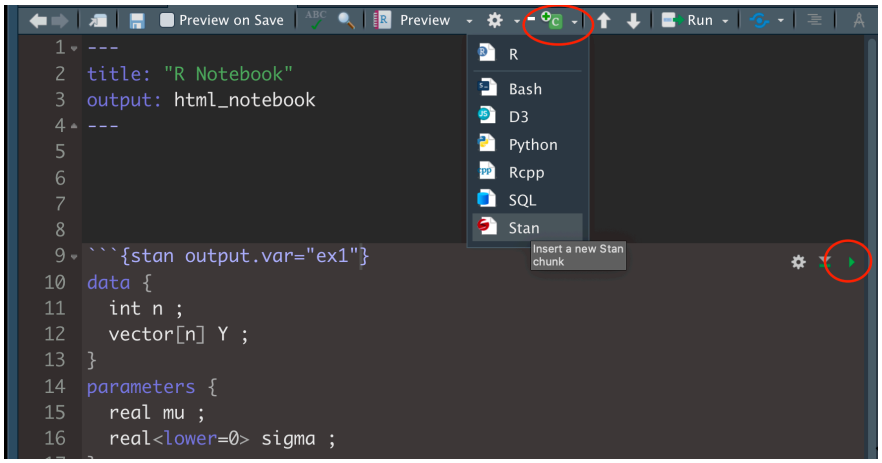
$$\mu \sim N(1.5, 0.1^2), \tag{28}$$
$$\sigma \sim \Gamma(1, 1). \tag{29}$$

Denoting the observations by $y_1, \ldots, y_n$, we wish to sample from two distributions:

- the posterior distribution of $\mu, \sigma^2|y_1, \ldots, y_n$;
- the predictive distribution of $Y_{n+1}, Y_{n+2}|y_1, \ldots, y_n$.

3.4 Specifying a model in RStudio

In RStudio, we make a `stan` code block. The code block has an argument `output.var`, defined as a string: we will use the string `"ex1"`. When we click on the green arrow at the top right of the code chunk, the model will be compiled as a "stanmodel" object called `ex1`.



The model code (that goes inside the code block) is as follows. Note that each commands needs to end in a semicolon.

```
data {
    int n ;
    vector[n] Y ;
}
parameters {
    real mu ;
    real<lower=0> sigma ;
}
model {
```

```
    mu ~ normal(1.5, 0.1) ;
    sigma ~ gamma(1, 1) ;
    Y ~ normal(mu, sigma) ;
}
generated quantities {
  vector[2] Ynew ;
  for (i in 1:2)
    Ynew[i] = normal_rng(mu, sigma) ;
}
```

3.5 Stan code blocks

The code is divided into Stan 'code blocks'. We've used four different types (there are a few others) which we will discuss in turn.

3.5.1 `data` block

```
data {
    int n ;
    vector[n] Y ;
}
```

This is where we declare the data we will pass to Stan for use in our statistical inference. We must declare all our data, and its type.

- Here, we have said that the variable Y is vector, with each element an unbounded continuous variable. For flexibility here, we've also included the number of observations n as an integer variable to let Stan know how large our vector of observations is. If we knew that we had, say 10, observations, we could have defined `vector[10] Y ;` directly.
- Common examples of data types include `real`, `int`, `vector`, and `matrix`. We can also attach modifiers, such as truncated variables as `real<lower=0,upper=1>`.

3.5.2 `parameters` block

```
parameters {
    real mu ;
    real<lower=0> sigma ;
}
```

This is where we declare all the parameters of our model that we aim to infer. Here we have the two parameters:

- μ, which is an unbounded continuous random variable, and
- σ, which is a positive continuous random variable.

In addition to the above data types, there are some additional useful parameter data types. This includes `simplex`, which is a vector of non-negative continuous variables whose sum is 1 and is useful for multinomial parameter sets or other probabilities. Also `corr_matrix` behaves as expected, and `ordered` allows the common constraint of $Z[1] > Z[2] > ... > Z[k]$ to be described conveniently.

3.5.3 `model` block

```
model {
    mu ~ normal(1.5, 0.1) ;
    sigma ~ gamma(1, 1) ;
    Y ~ normal(mu, sigma) ;
}
```

This is where we specify our likelihood and priors. This block is essentially telling Stan how to calculate the negative log posterior probability density (up to proportionality) which would be used to calculate the energy.

- In the above you will see we have used what is called *sampling statements*, which are denoted with the $\sim$ symbol. These describe the sampling distribution of the variables, and is not an instruction to draw a sample from such a distribution.
- Though `mu` is a scalar and `Y` is a vector, we have used the same type of sampling statement: `mu ~ normal()` and `Y ~ normal()`. Stan will understand the difference and interpret the elements of Y to be i.i.d. with the specified distribution.
- Stan understands a wide range of standard families of distributions, including all those from your distribution table. Note that these begin with a capital letter only where appropriate (such as `Student-t` but not `gamma`).

3.5.3.1 Improper priors

We could leave out specification of prior distributions for μ and σ:

```
model {
    Y ~ normal(mu, sigma) ;
}
```

This will assume improper uniform priors (with the constraint that `sigma` is positive). As you have seen in semester 1, the data will dominate the prior as the sample size increases, so this specification may be convenient in such cases, but robustness to the choice of prior (whether informative or uninformative) should always be investigated.

3.5.4 `generated quantities` block

```
generated quantities {
  vector[2] Ynew ;
  for (i in 1:2)
    Ynew[i] = normal_rng(mu, sigma) ;
}
```

Here we define any quantities that we want Stan to simulate.

- We need to declare that `Ynew` is a `vector` with 2 elements, before we can assign randomly generated values to `Ynew`.
- We've used the function `normal_rng()` (rather than the function `normal()`) to generate a normally distributed random variable with mean `mu` and standard deviation `sigma`.
- The `normal_rng()` function generates a scalar random variable, so we've had to use a `for` loop to generate i.i.d. variables: the elements of the vector `Ynew`.

3.6 Running the HMC algorithm

We'll first make up some data for the example:

```
exampleY <- rnorm(10, 1.6, 0.2)
```

We then make a `list` with data specified in the Stan `data` code block:

```
exampleData <- list(n = length(exampleY), Y = exampleY)
```

We are going to run a single chain sampler of 10000 iterations, but discard the first half as *burn-in*. We use the `sampling` function to carry out HMC sampling, using the definition of our Stan model that we assigned to the name `ex1` in the `stan` code chunk from above, and store this as `fit`.

```
sampling_iterations <- 1e4
fit <- sampling(ex1,
                data = exampleData,
                chains = 1,
                iter = sampling_iterations,
                warmup = sampling_iterations/2)
```

Now we will explore the output samples that Stan has provided from a HMC sampler. There are many ways that we can view results, starting with a simple print-out of a summary, shown below.

```
fit
```

```
## Inference for Stan model: 72072a3a3945e2d4332cba14d76c3dd9.
## 1 chains, each with iter=10000; warmup=5000; thin=1;
## post-warmup draws per chain=5000, total post-warmup draws=5000.
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## mu       1.62    0.00 0.06 1.50 1.59 1.63 1.66  1.73  2675    1
## sigma    0.20    0.00 0.06 0.12 0.16 0.19 0.23  0.36  2252    1
## Ynew[1]  1.62    0.00 0.22 1.17 1.49 1.63 1.75  2.03  4670    1
## Ynew[2]  1.62    0.00 0.22 1.15 1.50 1.62 1.76  2.05  5063    1
## lp__     9.03    0.03 1.08 6.12 8.63 9.37 9.81 10.09  1768    1
##
## Samples were drawn using NUTS(diag_e) at Thu Feb 10 20:19:35 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

This includes:

- Information about our implementation, including the number of chains and their length.
- A table of the variables we have sampled (μ and σ), and the log probability of the model (`lp`):
  - The posterior mean and standard deviation.
  - An estimate of effective sample size for each parameter, which can give us some indication about the autocorrelation in our chain.
- Information about the sampling algorithm, which includes an indication of the efficiency of the sampler.

3.7 Extracting and analysing the samples

We extract the raw samples as follows

```
chains <- rstan::extract(fit)
names(chains)
```

```
## [1] "mu"    "sigma" "Ynew"  "lp__"
```

Then we can do things such as:

```
summary(chains$mu)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.382   1.588   1.625   1.622   1.661   1.804
```

3.7.1 R packages for plotting outputs

3.7.1.1 ggmcmc An excellent package for exploring the output of an MCMC sample is `ggmcmc` (i Marín, 2016), which you can also use for the results of samplers that you have written yourself rather than merely Stan samplers. In the following we highlight some popular plots from this package that can be used to summaries our sampler.

In Figure 17, we plot the posterior marginals as separate one-dimensional density estimates, alongside the two-dimensional samples themselves.

```
# extract the posterior groups in a format that ggmcmc likes
samples <- ggmcmc::ggs(fit)
ggmcmc::ggs_pairs(samples)
```

In Figure 18, we plot a *caterpillar plot*, which gives displays credible intervals for each parameter. Compare this with the information table provided in the summary table, above. This is extremely useful for cases with high numbers of estimated parameters (see the random effects regression model later in this course!).

```
ggmcmc::ggs_caterpillar(samples, thick_ci = c(0.25,0.75))
```

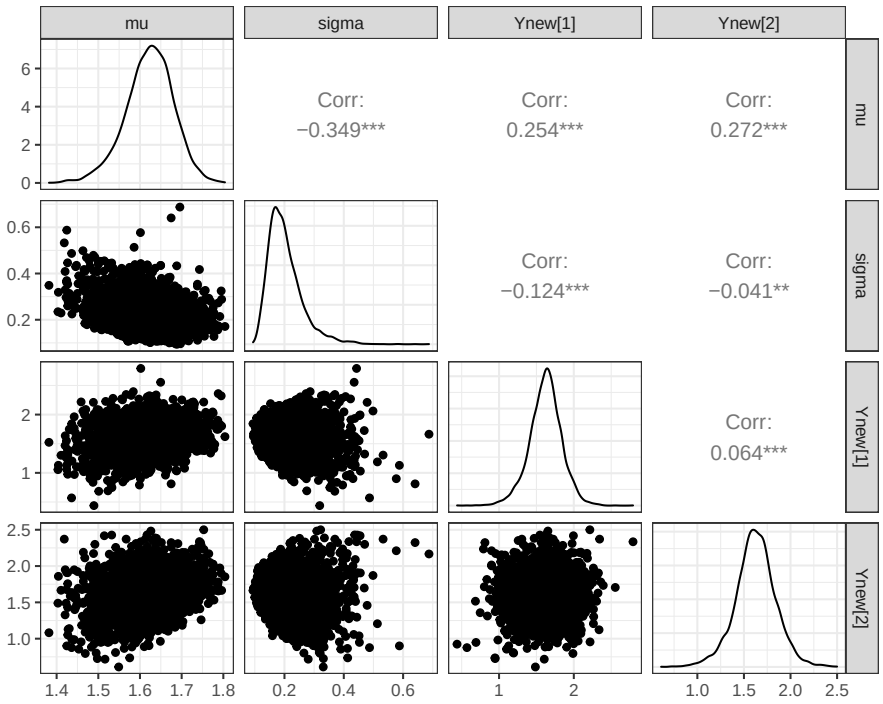


Figure 17: Posterior density estimate of the two parameters in the Gaussian example implemented in Stan.

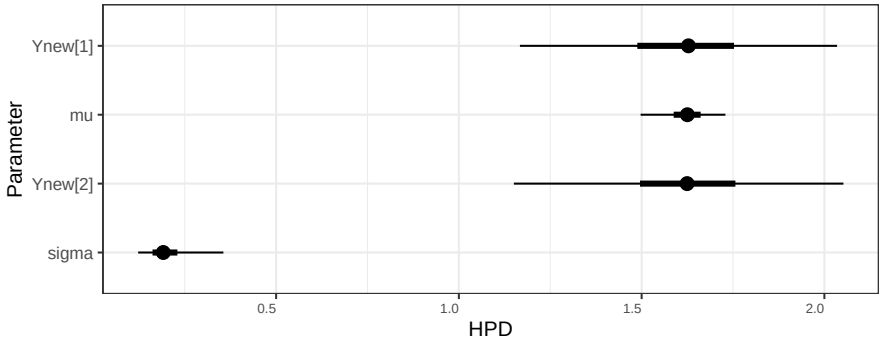


Figure 18: Caterpillar plot of the two parameters in the Gaussian example implemented in Stan. The thick bars indicate a posterior quantiles at 0.25 and 0.75, and the thin bards indicate quantiles at 0.025 and 0.975.

Figure 19 gives the trace plot of the two parameters, highlighting the autocorrelation in the sampled chain.

```
ggmcmc::ggs_traceplot(samples)
```

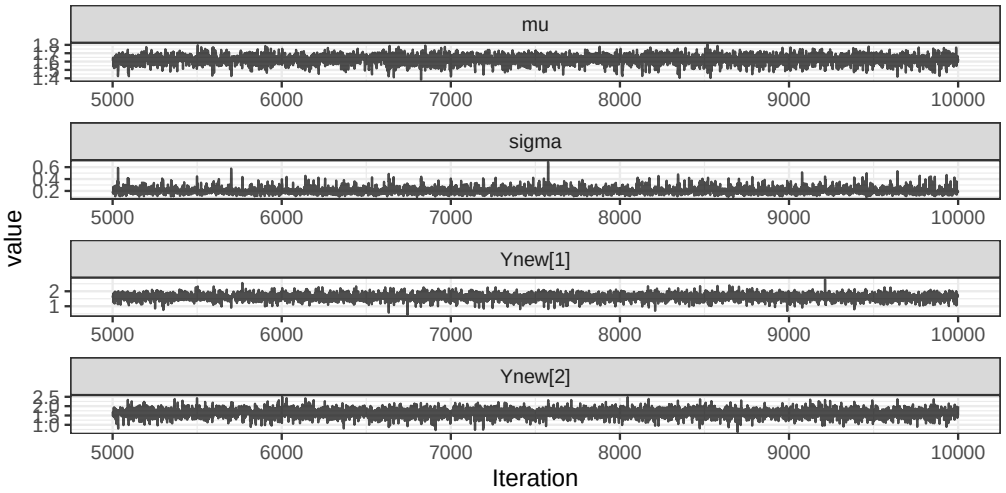


Figure 19: Trace plot of the two parameters in the Gaussian example implemented in Stan.

3.7.1.2 `bayesplot` The package `bayesplot` (Gabry and Mahr, 2021) interfaces nicely with `stan`. Trace plots can be produced as follows. Note that vectors are specified within the `contains()` function.

```
bayesplot::mcmc_trace(fit,
                      pars = vars(contains("Ynew"),
                                  "mu",
                                  "sigma"))
```

The following is similar to the caterpillar plot, but shows density functions.

```
bayesplot::mcmc_areas(fit, pars = vars(contains("Ynew"),"mu","sigma"),
                      prob_outer = 0.95)
```

3.7.1.3 `shinyStan` An interactive approach for exploring the results of the sampler is using the `shinystan` package (Gabry, 2018), which launches a shiny application in your browser. To launch this, you would use the command:

```
ssfit <- shinystan::as.shinystan(fit)
shinyStan::launch_shinystan(ssfit)
```

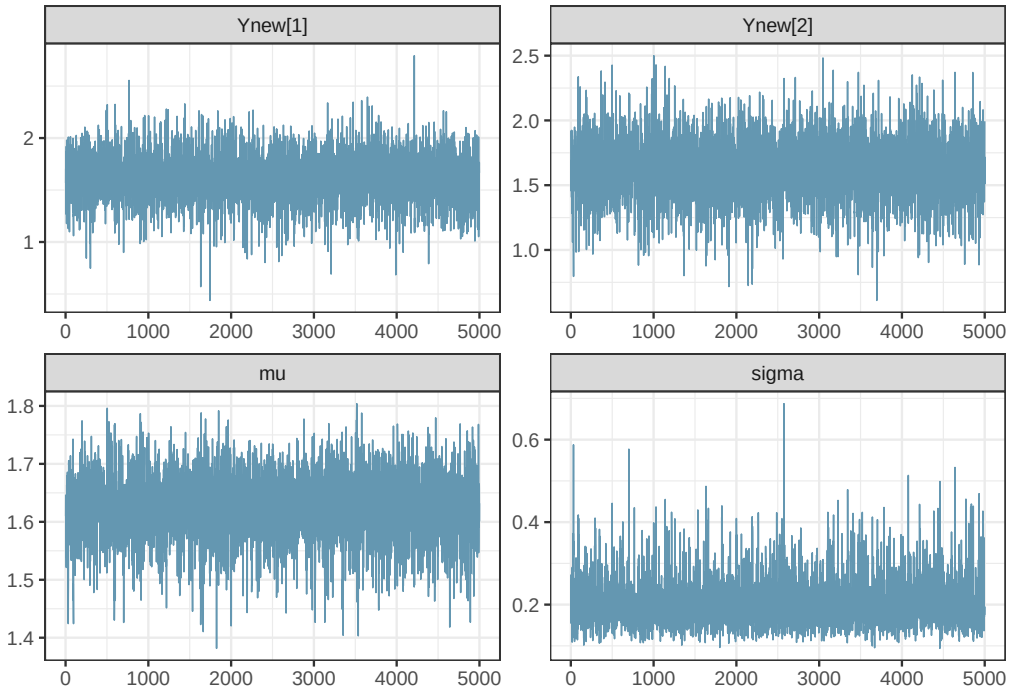


Figure 20: (ref:hier)

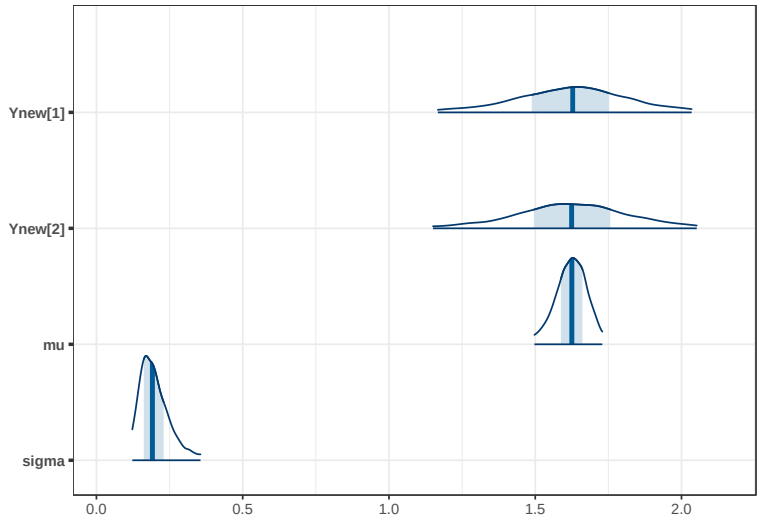


Figure 21: (ref:hier)

3.8 No U-Turn Sampler (NUTS)

You might have noticed that the information about the Stan sampler we implemented in this section said
Samples were drawn using NUTS(diag_e). This is because Stan does not actually use the base HMC algorithm that we have introduced, but an improvement to this known as the No U-Turn Sampler.

The base HMC algorithm requires the specification of a tuning parameter, T, which controls how much 'time' the fictitious ball rolls around the parameter space before being stopped in each iteration. The choice of T is not an easy one. In Figure 22 we show the proposed path from our earlier bivariate Gaussian example, but the difference here is that in the right panel the ball was left for twice the amount of time as the first. Although more time was allowed for the ball to move around, it has actually ended up closer to its original location than the left example. This is because it did a *U-turn*.

The ideal implementation of HMC would be to choose a value for T that allows large moves to be made, such as the left example in Figure 22. We could attempt to identify the ideal T for a given analysis through pilot runs, similar to practices carried out for choosing the step variance in a random-walk MH. However, in a multi-modal parameter space, the ideal T that avoids U-turns would differ depending upon the current location in that space.

The NUTS algorithm is a modification of the original HMC algorithm that dynamically changes the tuning parameter T in each iteration to produce efficient results. We will not go into the details of this, but intuitively, the path is being simulated until the particle is no longer moving further away from the starting point, at which point it is stopped. The specifics of the implementation of this in practice require additional considerations for the sampler to be a valid MCMC sampler.

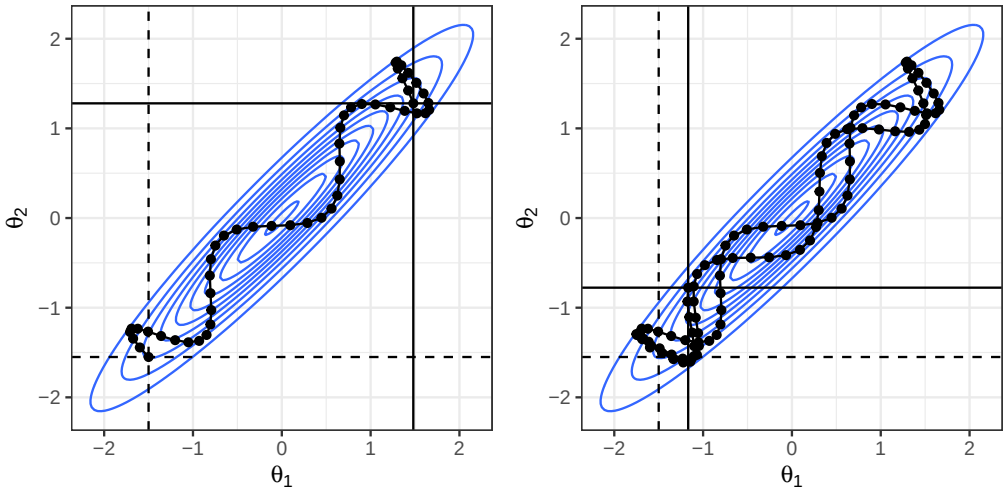


Figure 22: Given a starting location of (-1.5,-1.55) and an initial momentum of (-1,1), the leapfrog algorithm is implemented. Left: for 5 time-units. Right: for 10 time-units. The resulting path is shown with initial position highlight by the dashed line, and the final position highlighted by the solid line.

3.9 Further reading

Several online guides are available from the Stan home page:

- The User's Guide has lots of examples for particular models. Try here if you know what model you want to fit.
- Stan Functions Reference includes sections describing probability distributions you might want to use and how to implement them in Stan.
- Stan Reference Manual includes information about writing Stan programs, in particular (Section 8) on the different types of code blocks.

4 Bayesian regression in Stan

Aims of this chapter
1. Implement simple linear regression as a Bayesian using Stan.
2. Explore mixed effects and generalised linear regression using Stan.
3. Identify appropriate modelling assumptions for applied examples based on regression.
4. Interpret model output and compare with maximum likelihood approaches.

In this chapter we're going to explore how the Bayesian approach to regression inference can be conveniently implemented using Stan. We will start with a simple example, before moving on to demonstrate the potential with a more complex model. The focus here is on gaining familiarity of using Stan and gaining confidence identifying appropriate modelling assumptions for applied scenarios. The balance experiment example that we cover here in these lecture notes is *not* a 'blueprint' for implementing regression in any applied scenario, but rather aims to highlight the types issues we would investigate in a dataset and how we might justify the particular modelling assumptions we implement.

4.1 Simple linear regression

This example is a quick application of applying the simple linear regression model to see how we would implement this in Stan, and the Bayesian approach to regression fitting, in a real data setting.

4.1.1 Holiday hangover cures

The `hangover` dataset (available on the course page, and also see Ben Lambert's book *A Student's Guide to Bayesian Statistics*) contains information from Google Trends with estimates of the search traffic volume for the term *hangover cure* in the UK each week between February 2012 and January 2016. An extract is shown in Table 5 and displayed in Figure 23.

Figure 23 shows that search traffic varies over the course of a year, fluctuating reasonably evenly around some average. There is, however, a clear increase annually over the Christmas holiday period. Further, this increase occurs gradually as the holiday period is entered and gradually decreases as it is completed.

In relation to this data, we might be interested in estimating what the average annual uplift in searches is during the holiday period (which we will class as 10 December to 7 January). Regression could be a sensible approach to this problem because we have recognised:

- that there is a 'base-line' average level of search traffic (*an intercept*),
- an average increase over the holiday period (*an explanatory gradient*), and
- that there is random variation around this expected level (*error/noise*).

Introducing the holiday period as a variable could be implemented in a number of ways. Here, we encode information about the holiday season as the variable `holiday`. If the given week is not in the *holiday season*, then this variable is 0, if part of the week is included then this is given by the

Table 5: Extract of the hangover dataset.

date	volume	holiday
2012-02-05	18	0
2012-02-12	27	0
2012-02-19	22	0
2012-02-26	27	0
2012-03-04	35	0

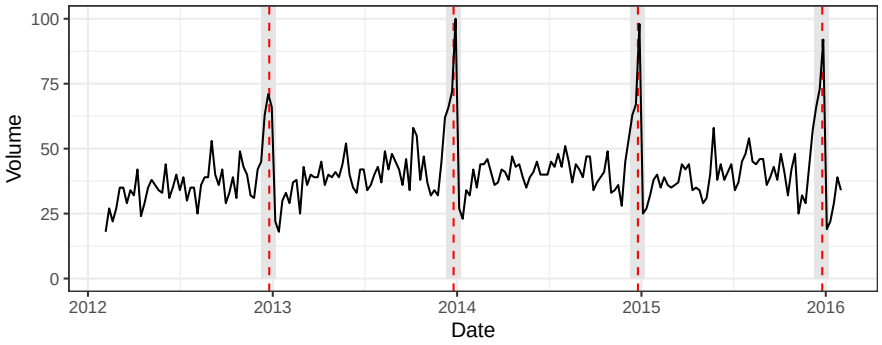


Figure 23: The hangover dataset. Google search traffic volume for the term hangover cure in the UK between Feb 2012 and Jan 2016. Red dashed lines highlight Christmas day each year and grey bars represent the region between 10 Dec and 7 Jan, classed as the holiday season each year.

appropriate proportion, and if the whole week if included in the holiday then the variable takes the value 1.

A simple linear regression to explore our aim is given by

$$V_i = \beta_0 + \beta_1 h_i + \epsilon_i, \tag{30}$$

where V_i is the search traffic volume of week i, h_i is the holiday season variable describing the proportion of week i that falls within the holiday period, and $\epsilon_i \sim N(0, \sigma^2)$. The parameters of this model are interpreted as β_0 being the average hangover search volume for weeks that are not during the holiday season and β_1 is the average additional search traffic for a wholly holiday week. The proportional increase, or *uplift* as we will refer to it, is given by the ratio $\frac{\beta_1}{\beta_0}$. The parameter σ^2 describes the weekly variation in traffic volume, after the holiday season expected uplift has been accounted for.

4.1.2 Least squares fit

We'll first fit the model using least squares. Even if the intention is to use a Bayesian approach, it can help to have some results obtained via another method for comparison.

```
ls_hang <- lm(volume ~ holiday, data = hangover)
summary(ls_hang)
```

```
##
## Call:
## lm(formula = volume ~ holiday, data = hangover)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -38.326  -3.988   0.806   5.012  31.070
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   37.988      0.624   60.88   <2e-16 ***
## holiday       30.942      2.344   13.20   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.616 on 207 degrees of freedom
## Multiple R-squared:  0.4571, Adjusted R-squared:  0.4544
## F-statistic: 174.3 on 1 and 207 DF,  p-value: < 2.2e-16
```

The least squares estimate would therefore lead to the estimate of the uplift being an 81.5% increase in search traffic during a week in the holiday season.

4.1.3 Bayesian approach in Stan

The most naive implementation of the hangover model in a Bayesian setting is to assume improper, flat priors for the three parameters, $\{\beta_0, \beta_1, \sigma^2\}$. Under such an assumption, the `.stan` program code is given by:

```
data {
  int T ;
  vector[T] V ; // volume
  vector[T] h ; // holiday
}

parameters {
  real beta0 ;
  real beta1 ;
  real<lower = 0> sigma ;
}

model {
  V ~ normal(beta0 + beta1*h, sigma) ;
}

generated quantities {
  real uplift ;
```

```
  uplift = beta1 / beta0 ;
}
```

Note that we've used a `generated quantities` code block to monitor a quantify of interest that is a function of some parameters in the model:

$$uplift = \frac{\beta_1}{\beta_0}.$$

For each iteration of the sampler, the current samples of β_0 and β_1 are used to calculate a sample of the uplift. Generated quantities will appear in the parameter summary table. The ability to specify new variables as combinations of our parameters is extremely useful. In the likelihood approach above we can obtain a point estimate of the uplift, based on our maximum likelihood estimates, but quantifying the uncertainty of this is not so straightforward. But by specifying the uplift as a generated quantity, because it is calculated for each sampled pair of the two parameters, we can easily quantify uncertainty.

As in our previous examples, we use the following `sample` command to implement our HMC sampler. Here we have run two chains to demonstrate the output this produces (an overall summary and separate chain summaries). Each of the two chains ran for 1,000 iterations, with the first half removed as 'burn-in'.

```
# set data in a way Stan understands
data <- list(T = nrow(hangover), V = hangover$volume, h = hangover$holiday)

sampling_iterations <- 1e4
fit_improper <- sampling(hangimproper,
                 data = data,
                 chains = 2,
                 iter = sampling_iterations,
                 warmup = sampling_iterations/2)
```

A summary of the results is given:

```
## $summary
##                mean      se_mean         sd         2.5%          50%
## beta0    37.9906300 0.0071924263 0.63568553   36.7381845   37.9871900
## beta1    30.9596242 0.0261558570 2.36568298   26.3229674   30.9711890
## sigma     8.6744835 0.0049039809 0.43725740    7.8678092    8.6550348
## uplift    0.8154774 0.0007847387 0.06775083    0.6836838    0.8156254
## lp__   -552.9759663 0.0176707715 1.25326271 -556.2108417 -552.6493043
##                97.5%    n_eff      Rhat
## beta0     39.223919 7811.489 0.9998941
## beta1     35.592301 8180.412 0.9998168
## sigma      9.586811 7950.177 0.9999100
## uplift     0.949653 7453.825 0.9998113
## lp__    -551.556051 5030.061 1.0001219
##
## $c_summary
```

```
## , , chains = chain:1
##
##          stats
## parameter         mean         sd         2.5%          50%        97.5%
##    beta0    37.9910379 0.63401834   36.7517638   37.9813528   39.2262303
##    beta1    30.9513961 2.32965854   26.3433769   30.9348518   35.5377681
##    sigma     8.6772193 0.43215957    7.8834217    8.6585727    9.5637623
##    uplift    0.8152383 0.06671415    0.6862336    0.8147219    0.9491596
##    lp__   -552.9514129 1.21760081 -556.0581414 -552.6439069 -551.5577478
##
## , , chains = chain:2
##
##          stats
## parameter         mean         sd         2.5%          50%       97.5%
##    beta0    37.9902221 0.63741151   36.7256590   37.9919937   39.221869
##    beta1    30.9678523 2.40137192   26.2721119   31.0061655   35.647604
##    sigma     8.6717477 0.44232280    7.8596864    8.6514699    9.611406
##    uplift    0.8157165 0.06877774    0.6811207    0.8169106    0.949670
##    lp__   -553.0005197 1.28759129 -556.3285699 -552.6606701 -551.554346
```

Note that the posterior means of β_0 and β_1 are similar to the least squares estimates.

The results of this implementation give a posterior mean for the uplift of 81.5% and a 95% credible interval of (68.4%, 95%).

4.2 Recap of logistic regression

You may have previously encountered generalised linear models (GLMs) in other courses and, in particular, that of logistic regression. Here we will give an overview of logistic regression as we are going to apply this to a more substantial modelling example in this chapter.

The simple linear regression model that you have encountered assumes that

$$Y_i \sim N(\mu_i, \sigma^2),$$

where

$$\mu_i = \sum_{j=1}^{m} \beta_j X_{ij}.$$

Logistic regression is used when the dependent variable is not continuous, but is instead a binary variable (such as success and failure encoded as 1 and 0, respectively). The motivation behind this modelling approach is shown in Figure 24.

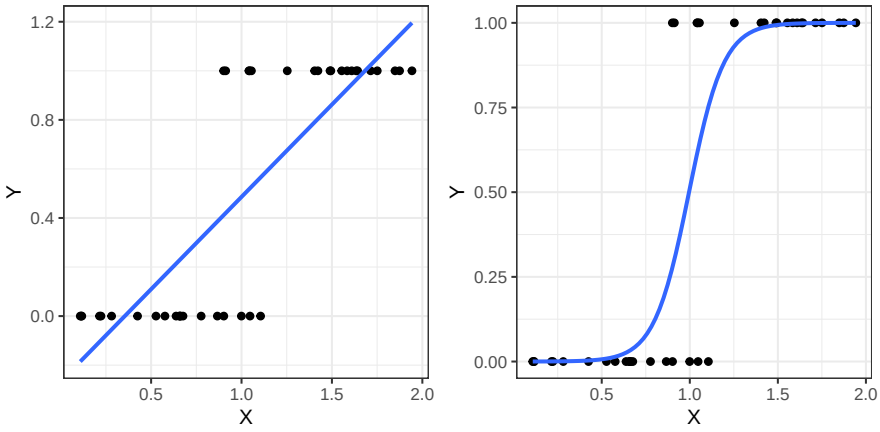


Figure 24: Motivation for using logistic regression. If our response variable is binary, simple linear regression (left) is problematic because predictions can exceed the (0,1) range. Alternatively, logistic regression (right) only produces predictions within the valid range.

To implement logistic regression we model the binary outcome as a Bernoulli response,

$$Y_i \sim Bernoulli(\mu_i),$$

where the probability of success is defined in terms of the explanatory variables as

$$\mu_i = \mathbb{P}(Y_i = 1) = \frac{e^{\sum_{j=1}^{m} \beta_j X_{ij}}}{1 + e^{\sum_{j=1}^{m} \beta_j X_{ij}}}.$$

You can see that although we assume a different distribution for the response variable Y in logistic regression, we still retain a similar process of expressing the parameter of the distribution as a linear combination of the variables $\{X_j\}$. The reason that the expression above differs to the expression for μ in simple linear regression is because this now represents a probability and so is constrained between 0 and 1. Explicitly, we have applied a **log-odds/logit** transformation to μ, given by

$$\text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right),$$

and defined the regression model as

$$\text{logit}(\mu_i) = \sum_{j=1}^{m} \beta_j X_{ij}.$$

The most common method of estimating the β coefficients in a logistic regression model is based on maximum likelihood, using iteratively reweighted least squares. We will not cover the details of this during this course, but you may encounter this in your other courses. To implement model fitting in `R`, we use the `glm` package, where the `glm` function uses the syntax you are familiar with from implementing simple linear regression with `lm`.

4.3 Overview of mixed effects

A random effects model (or multilevel, hierarchical, mixed effects) is an extension to the regression theory you have previously met. In the simple linear regression model, a key assumption is that the observations Y_i, for $i = 1, \dots, n$ are independent from one another, with the joint distribution being

$$\mathbf{Y} \sim N_n(\mathbf{X}\beta, \sigma^2 I_n),$$

where I_n is the identity matrix. The vector of coefficients, β, is referred to as the coefficients of the *fixed effects*. The variation that we observe between the observations Y_i arises from the fixed structure defined by $\mathbf{X}\beta$ and the error associated with the shared variance σ^2.

A common situation, however, is that observations arise in groups that are correlated, such as blocked group experiments or repeated measurements. The main idea of random effects models is recognising that the variables defining the grouping structure should be treated differently to how we would usually treat explanatory variables. We do this by introducing an additional source of variation that is group-specific. We can think of this as missing information on the groupings that, if it were available, would be included in the statistical model.

We can express a general mixed effects model as

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{Z}\gamma + \epsilon,$$

where:

- $\mathbf{X}\beta$ is defined as previously, with a design matrix $\mathbf{X}$ based on independent variables, and a vector of coefficients that are to be estimated β,
- $\mathbf{Z}\gamma$ consists of another design matrix $\mathbf{Z}$ that describes the grouping structure, and a vector of random effects $\gamma \sim N(\mathbf{0}, G)$, and
- $\epsilon \sim N(\mathbf{0}, \sigma^2 I)$ is the error, as defined previously.

4.3.1 Simple example of a mixed effects model

Let's consider a simple example. A set of observations correspond to repeated measures on the same group of subjects, so that we have observations Y_{ij} representing the j^{th} response from the i^{th} subject, for $j = 1, \dots, k$ and $i = 1, \dots, n$. Additionally we have observations of a single explanatory variable X_{ij} corresponding to the response. We could propose the model

$$Y_{ij} = \beta_0 + \beta_1 X_{ij} + \gamma_i + \epsilon_{ij},$$

so that there is a subject-specific random effect $\gamma_i \sim N(0, \sigma^2_\gamma)$ that represents the assumption that observations from the same individual share some commonality, but that individuals differ from one another, having arisen from some common population that has a variation of σ^2_γ. The within-group variation is then given by the residuals $\epsilon_{ij} \sim N(0, \sigma^2)$. We note that, as we do not observe the random effects γ_i, this random effects model has the following features:

- a general observation has the distribution $Y_{ij} \sim N(\beta_0 + \beta_1 X_{ij}, \sigma^2_\gamma + \sigma^2)$,
- observations from the same individual, say Y_{i1} and Y_{i2}, have covariance σ^2_γ, and

- observations from different individuals, say Y_{11} and Y_{32}, are independent.

The above model differs from including a fixed effect that is a factor variable for the individual, with coefficient γ_i. In that scenario the observations from the same individual would be uncorrelated. The results of model fitting would produce estimates of the effect of each subject in the experiment, but we would not have any information about the general population to predict anything about a *new* individual we have not seen.

4.3.2 Fixed or random effects?

This will depend on the aims of your analysis. If you're interested in learning about the specific subjects/groups that you observed in your data, you want the fixed effects estimates of the groups. If you're interested in learning about the general population that your subjects were sampled from, you want the random effect variation estimate. Statistical efficiency may also play a part in your decision making: if you have a large number of groups, then a fixed effects treatment involves many more parameters than the random effects counterpart.

4.4 Mixed effect logistic regression

In this section we will be using some experimental data to explore implementing mixed effect logistic regression. We are primarily interested in implementing this approach in a Bayesian framework, using Stan, but as a comparison we will also be implementing more traditional maximum likelihood approaches. As with any regression modeling application, we will be considering how the various independent variables that we have information on affect the dependent variable, which includes asking whether these variables have *any* significant effect upon the observed balance variable.

4.4.1 Balance experiment

The `balance` dataset, presented in Faraway (2016) contains the results of an experiments on the effects of surface and vision on balance. More details about the sources and an exploratory data analysis are available in these MAS61004 notes, but we will also perform an EDA here.

Balance was assessed qualitatively based on observation by the experimenter. This dependent variable, `CTSIB`, was on a four point ordinal scale, with 1 meaning 'stable' and 4 meaning 'unstable'. Faraway converted this to a binary variable called `Stable`, where:

- 1 represents the original level 1 (stable), and
- 0 represents the original levels 2-4 (varying degrees of instability).

The balance of subjects were observed for:

- `Surface`: two different surfaces (foam or normal), and
- `Vision`: restricted and unrestricted vision (eyes closed, eyes open or with a dome placed over the head).

Forty subjects were studied (`Subject`), twenty males and twenty females (`Sex`), ranging in age from 18 to 38 (`Age`), with heights given in *cm* (`Height`) and weights in *kg* (`Weight`). Each subject was

Table 6: Extract of the balance dataset showing the experimental design of subject, suface and vision. This includes all observations for subject 1.

Surface	Vision	CTSIB	Stable
norm	open	1	1
norm	open	1	1
norm	closed	2	0
norm	closed	2	0
norm	dome	1	1
norm	dome	2	0
foam	open	2	0
foam	open	2	0
foam	closed	2	0
foam	closed	2	0
foam	dome	2	0
foam	dome	2	0

tested twice in each of the surface (two) and eye combinations (three), which results in a total of 12 measures per subject.

We will centre the `Age`, `Height` and `Weight` variables by subtracting the mean and dividing by the range. This can help with numerical computation, by weakening dependence between model parameters.

This study design is known as a **full factorial design** between subject, surface and vision with two replicates.

In addition to the observed dependent variables listed above, we also add a final variable (`Dummy`) which is an independent variable with no effect on the response. These are independent draws from a standard normal. We therefore have a variable that we *know* is not linked to the balance of individuals. This will be useful for considering whether other dependent variables display useful predictive information.

The data are constructed with the following code.

```
set.seed(123)
balance <- faraway::ctsib %>%
  mutate(Stable = ifelse(CTSIB == 1, 1, 0),
         Subject = factor(Subject),
         Age = (Age - mean(Age)) / diff(range(Age)),
         Weight = (Weight - mean(Weight)) / diff(range(Weight)),
         Height = (Height - mean(Height)) / diff(range(Height)),
         Dummy = round(rnorm(480), 1))
```

The full factorial experimental design/set-up is displayed in Table 6, which includes all observations for `subject 1`. The full data frame also contains the information on sex, age, height and weight for subject 1, along with that of the 39 other subjects. An extract of the data frame is shown in Table 7.

Table 7: Extract of the balance dataset showing all variables that will be considered in our regression analysis.

Subject	Surface	Vision	Sex	Age	Height	Weight	Dummy	Stable
1	norm	open	male	-0.24	0.0822917	-0.0506014	-0.6	1
1	norm	open	male	-0.24	0.0822917	-0.0506014	-0.2	1
1	norm	closed	male	-0.24	0.0822917	-0.0506014	1.6	0
1	norm	closed	male	-0.24	0.0822917	-0.0506014	0.1	0
1	norm	dome	male	-0.24	0.0822917	-0.0506014	0.1	1
1	norm	dome	male	-0.24	0.0822917	-0.0506014	1.7	0

The response variable for this experiment is binary, as we have *stable* or *unstable* observations. It is therefore sensible to model this using logistic regression. This is our first modelling decision.

There are multiple explanatory variables that could be incorporated in such a model, and these could feature as fixed or random effects. Determining the appropriateness of variables as fixed or random effects is an important skill. In the following we carry out some exploratory data analysis to explore the appropriateness of some modelling decisions we could implement. For clarity, we will refer to the variables as:

- `Surface` and `Vision` as *design variables*. Note that these are both factor variables.
- `Sex`, `Age`, `Height`, `Weight` as *covariates*. Note that `Sex` is a factor variable.
- `Subject` as the *grouping variable*. Note that this is a factor variable.

4.4.2 Exploratory data analysis

4.4.2.1 Considering the grouping variable as a fixed or random effect Let's consider the variable `Subject`, which is a factor variable of which we have observed 40 categories of. By applying logistic regression we are modeling the probability of stability, given some combination of observed variables. If a given subject was only ever observed to give the same outcome (say all unstable) then this would cause problems if `Subject` was included as a fixed effect term, because the probability of stability would be 0 for that subject. For any data application you are working on, this is an important aspect of your data that you should check in the exploratory analysis stage.

Considering the `balance` data, we can compute the proportion of stable outcomes for the 12 observations of each subject, and tabulate how many subjects have each possible proportion. This is given in Table 8. From this we see that five subjects were unstable for all observations. We therefore cannot include a subject-specific fixed effect in our model for this experiment. If we would like to incorporate the concept of correlation between repeated measurements on the same subject, we can make a decision to include subject as a random effect.

4.4.2.2 Considering the design variables effect Another set of factor variables we would be interested in including in our balance model are the `Surface` and `Vision` variables, with two and three categories, respectively. These were the experimental design variables and so are of particular importance in determining their effect on balance.

Table 8: The number of subjects with the range of possible mean stability scores. The mean stability for a subject is calculated as the proportion of their 12 binary stability observations that were classed as stable.

propStable	n
0.0000000	5
0.0833333	4
0.1666667	15
0.2500000	3
0.3333333	4
0.4166667	3
0.5000000	4
0.6666667	2

We produce contingency tables for number of observations that were stable/unstable at the different levels of the design variables:

```
##       Surface
## Stable foam norm
##      0  230  136
##      1   10  104
```

```
##       Vision
## Stable closed dome open
##      0    143  138   85
##      1     17   22   75
```

This shows that there is a mixture of stability observations for the two categories of surface, and similarly for the three categories of vision. These contingency tables also highlight that there does appear to be an effect on balance between the categories. We could consider including both `Surface` and `Vision` are fixed effects in our regression.

Note, however, if we look at the combinations of these variables together we see in Table 9 the proportions of stability. Here we see that two combinations of surface and vision were only ever observed to be unstable. We therefore would find complications if we attempt to include a fixed effects **interaction term** between `Surface` and `Vision`. Our initial model will therefore feature both the design variables as fixed effects, but will not feature the possible interaction term.

4.4.2.3 Considering the covariates The remaining variables in our data are the covariates `Sex`, `Age`, `Height`, `Weight`, and our dummy variable `Dummy`. Before deciding on our proposed regression model, we can produce exploratory plots of whether the probability of stability is affected by each variable. This exploratory analysis is important, because it could highlight important issues with our observations that were otherwise would not be aware of. It may also be the case that we have a very large number of covariates in an application, and including all of them in a model may not be appropriate.

In Figure 25, we have calculated the proportion of stable observations for each subject, and plotted

Table 9: The proportion of stable observations observed, given the combinations of surface and vision variables.

Surface	Vision	propStable
foam	closed	0.0000
foam	dome	0.0000
foam	open	0.1250
norm	closed	0.2125
norm	dome	0.2750
norm	open	0.8125

this against `Sex`. This suggests that this variable has some affect on stability and would be an appropriate fixed effect choice for inclusion in our model. Remember that our fitted model may still indicate that this effect is not significant.

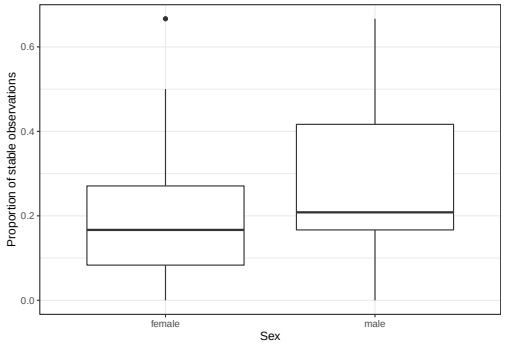


Figure 25: The proportion of stable observations out of the 12 carried out for each subject, split by the sex variable.

In Figure 26, we have plotted the age, weight, height and dummy variables, split by sex and stability. These plots are useful for considering the effects of these four covariates on stability, but also their interaction with the sex covariate. We can see that there is a fairly clear separation between the two sexes in terms of the height and weight of individuals (males are taller and heavier). Let's explore this a little more with pairs plots in Figure 27. The separation of the two sexes is obvious, as is the high correlation between height and weight. This structure in the data is useful to be aware of, and suggests that it may not be necessary (or even useful) to include both height and weight as variables in our model.

When considering the possible effects of the covariates on stability in Figure 26 it is useful to have our dummy variable. We know that this variable is independent of stability, and so when comparing the stable/unstable box plot distributions we see patterns that can only have arisen by chance. This provides some perspective for considering whether the remaining three covariates in this plot also display differences in stability through random chance or an actual effect. These plots suggest that age, weight and height may not be at all important in determining stability.

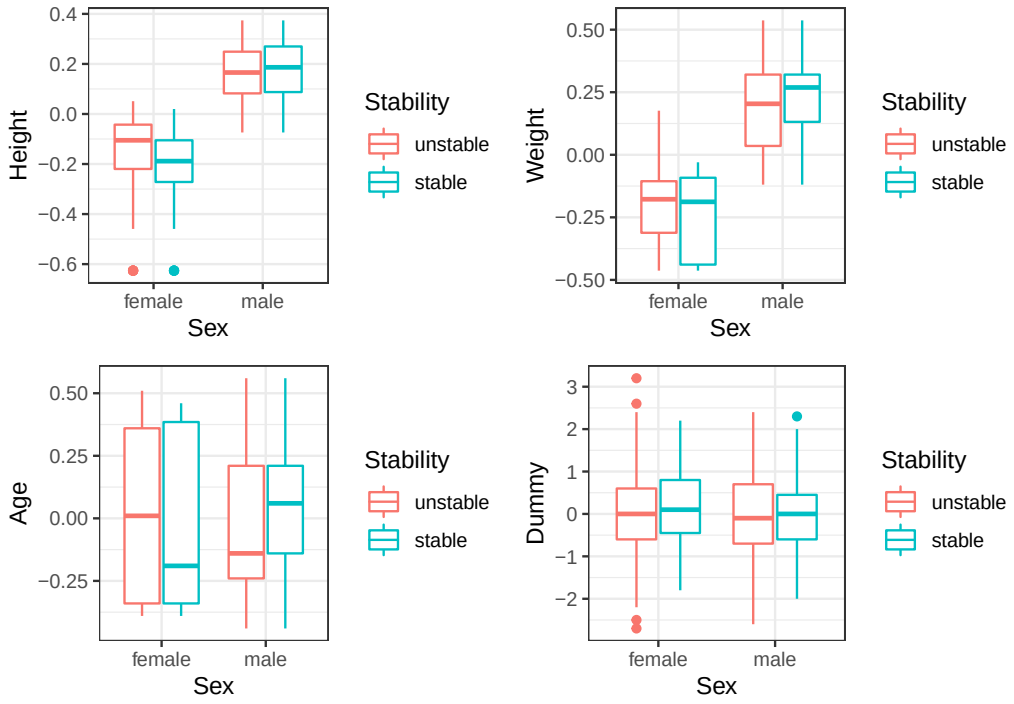


Figure 26: The height/weight/age/dummy variable of individuals split by sex and stability.

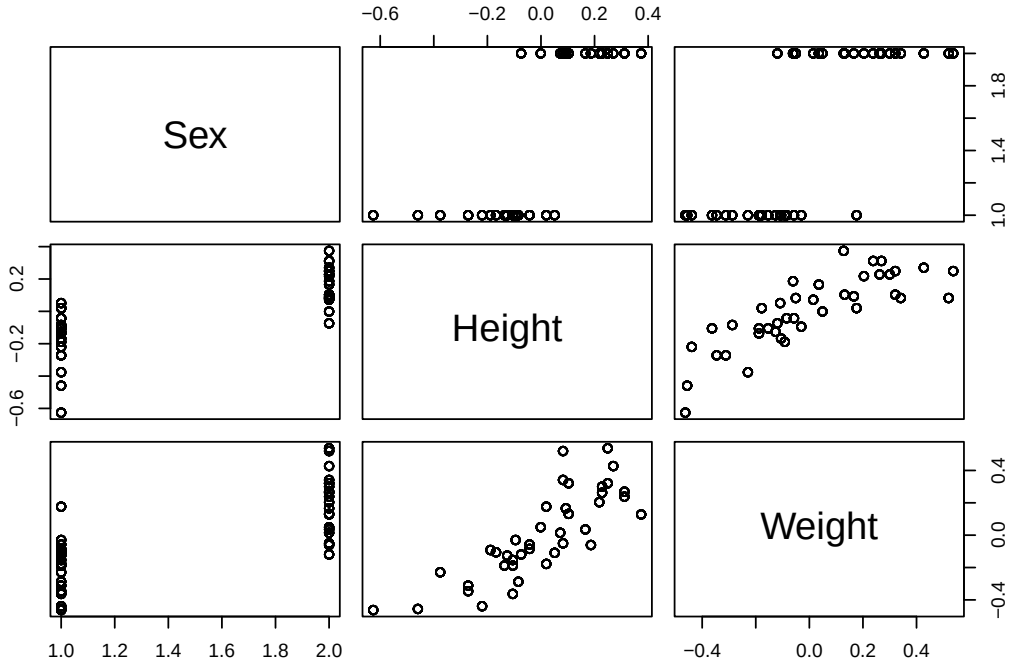


Figure 27: The combinations of sex, height and weight variables for the balance experiment.

4.4.3 Maximum likelihood approach

4.4.3.1 Ordinary logistic regression (no grouping variable) In the above we found that we cannot include `Subject` as a fixed effect for this experiment. We would like to include `Subject` as a random effect to take into account the variability between individuals. However, first for simplicity, we will implement the model without this feature. We are therefore modeling stability as a logistic regression, with fixed effects for the design variables and the covariates. Let's set this up with formal notation.

Define Y_{ijkl} to be the stability response for subject $i = 1, \ldots, 12$, surface $j = 1, 2$, vision setting $k = 1, 2, 3$, and replicate $l = 1, 2$. We have

$$Y_{ijkl} \sim Bernoulli(\mu_{ijkl}),$$

and we suppose

$$\text{logit}(\mu_{ijkl}) = \beta_0 + \alpha_j + \gamma_k + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \beta_3 x_{3,i} + \beta_4 x_{4,i} + \beta_5 x_{5,ijkl},$$

where

- α_j is the additive contribution of the surface factor, with $\alpha_1 = 0$,
- γ_k is the additive contribution of the vision factor, with $\gamma_1 = 0$,
- β_1 is the additive contribution of the sex factor where $x_{1,i}$ is the sex of subject i, with $x_{1,i} = 1$ for a male and 0 for a female,
- $x_{2,i}, x_{3,i}, x_{4,i}$ are the corresponding age, height, weight covariates of subject i, and
- $x_{5,ijkl}$ is the dummy variable.

Note that we have not included the subject as an effect here, and that the subscript i is needed only to identify the subject's covariate values.

Exercise:
Spend some time making sure you understand the interpretation of the coefficients in this model definition. For example, what is the significance of β_0? Or what combination of coefficients produces the expected probability of stability for a 20 year old female with restricted vision?

Our `glm` model definition and summary is given here:

```
glmNoSubject <-
  glm(Stable ~ Surface + Vision +
        Sex +
        Age + Height + Weight + Dummy,
      family = binomial,
      data = balance)

summary(glmNoSubject)
```

```
##
## Call:
## glm(formula = Stable ~ Surface + Vision + Sex + Age + Height +
##     Weight + Dummy, family = binomial, data = balance)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.41266  -0.51936  -0.13654  -0.04721   2.62161
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.29989    0.65959  -9.551  < 2e-16 ***
## Surfacenorm  4.05639    0.45701   8.876  < 2e-16 ***
## Visiondome   0.38653    0.38537   1.003 0.315851
## Visionopen   3.26062    0.42425   7.686 1.52e-14 ***
## Sexmale      1.46294    0.52152   2.805 0.005029 **
## Age          0.07167    0.48681   0.147 0.882950
## Height      -4.86326    1.31178  -3.707 0.000209 ***
## Weight       2.71654    1.05972   2.563 0.010364 *
## Dummy        0.28974    0.16266   1.781 0.074881 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 526.25  on 479  degrees of freedom
## Residual deviance: 291.96  on 471  degrees of freedom
## AIC: 309.96
##
## Number of Fisher Scoring iterations: 6
```

Exercise:
Spend some time interpreting this model output. What do these results indicate about the three levels of the `Vision` variable? Do these results agree with your understanding of the data given our exploratory data analysis? What conclusions would you report in a summary to a client if you were analysing this data for them?

4.4.3.2 Random effects (including the grouping variable)

Will will extend the previous model to include a random effect for each of the 12 different subjects.

The full model is now

$$\text{logit}(\mu_{ijkl}) = \beta_0 + \alpha_j + \gamma_k + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \beta_3 x_{3,i} + \beta_4 x_{4,i} + \beta_5 x_{5,ijkl} + b_i,$$

with the extra random effects term $b_i \sim N(0, \sigma_b^2)$.

Let's fit this random effects GLM with the `lme4` package:

```
glmSubjectRandom <-
  lme4::glmer(Stable ~ Surface + Vision +
                Sex +
                Age + Height + Weight + Dummy +
                (1 | Subject),
              family = binomial,
              data = balance,
              nAGQ = 25)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.320305 (tol = 0.002, component 1)
```

We get a message that the method for model fitting failed to converge! It's important to remember that there is not a closed form expression for the maximum likelihood estimates in mixed effects generalised linear models, and numerical methods must be used. This means that a *solution* is not always viable.

Let's return to our exploratory data analysis, and recall how we suspected little information was being gained by our collection of covariates, and in particular, that height and weight were highly correlated. So we're going to proceed by removing the `Height` variable.

```
glmSubjectRandom <-
  lme4::glmer(Stable ~ Surface + Vision +
                Sex +
                Age + Weight + Dummy +
                (1 | Subject),
              family = binomial,
              data = balance,
              nAGQ = 25)

summary(glmSubjectRandom)
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 25) [glmerMod]
##  Family: binomial  ( logit )
## Formula: Stable ~ Surface + Vision + Sex + Age + Weight + Dummy + (1 |
##     Subject)
##    Data: balance
##
##      AIC      BIC   logLik deviance df.resid
```

```
##    251.2    288.8   -116.6    233.2      471
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.9493 -0.1411 -0.0196 -0.0007  5.6453
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  Subject (Intercept) 8.491    2.914
## Number of obs: 480, groups:  Subject, 40
##
## Fixed effects:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -11.20462    1.87176  -5.986 2.15e-09 ***
## Surfacenorm   7.32839    1.07053   6.846 7.62e-12 ***
## Visiondome    0.67763    0.52966   1.279    0.201
## Visionopen    6.14100    0.99263   6.187 6.15e-10 ***
## Sexmale       1.89216    1.64924   1.147    0.251
## Age           0.05018    1.63334   0.031    0.975
## Weight       -0.05908    3.03973  -0.019    0.984
## Dummy         0.23971    0.23094   1.038    0.299
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Srfcnr Visndm Visnpn Sexmal Age    Weight
## Surfacenorm -0.821
## Visiondome  -0.246  0.109
## Visionopen  -0.809  0.831  0.377
## Sexmale     -0.591  0.174  0.023  0.184
## Age         -0.058  0.002  0.003  0.006  0.131
## Weight       0.366 -0.028 -0.003 -0.039 -0.765 -0.172
## Dummy       -0.014 -0.012  0.006  0.007  0.027  0.014  0.007
```

This has helped and convergence has been reached.

> **Exercise:** How do the fitted estimates compare to the fitted model that did not consider a subject random effect?

The summary of this fitted model suggests that the covariates are not significantly useful in explaining the variation in our data. So we can explore the possibility of removing these (the sex, age, height, weight, and of course the dummy variable we know for certain is uninformative). This reduced model is fit as in the below, and the nested model comparison shows that indeed, we could reasonably drop the subject covariate information.

```
glmSubjectRandomReduced <-
  lme4::glmer(Stable ~ Surface + Vision +
                (1 | Subject),
              family = binomial,
              data = balance)

anova(glmSubjectRandomReduced,
      glmSubjectRandom)
```

```
## Data: balance
## Models:
## glmSubjectRandomReduced: Stable ~ Surface + Vision + (1 | Subject)
## glmSubjectRandom: Stable ~ Surface + Vision + Sex + Age + Weight + Dummy + (1 | Subject)
##                          npar    AIC    BIC  logLik deviance  Chisq Df
## glmSubjectRandomReduced     5 246.40 267.26 -118.20   236.40
## glmSubjectRandom            9 251.23 288.79 -116.61   233.23 3.1682  4
##                          Pr(>Chisq)
## glmSubjectRandomReduced
## glmSubjectRandom            0.5301
```

Let's summarise this reduced model:

```
summary(glmSubjectRandomReduced)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: Stable ~ Surface + Vision + (1 | Subject)
##    Data: balance
##
##      AIC      BIC   logLik deviance df.resid
##    246.4    267.3   -118.2    236.4      475
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.2407 -0.1570 -0.0160 -0.0004  6.3710
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  Subject (Intercept) 11.2     3.346
## Number of obs: 480, groups:  Subject, 40
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.9807     1.9061  -5.761 8.37e-09 ***
## Surfacenorm   7.8425     1.3011   6.027 1.67e-09 ***
## Visiondome    0.6904     0.5344   1.292    0.196
```

```
## Visionopen    6.5930     1.2099   5.449 5.05e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Srfcnr Visndm
## Surfacenorm -0.930
## Visiondome  -0.259  0.116
## Visionopen  -0.918  0.882  0.336
```

4.4.4 Bayesian approach using Stan

We're going to define our mixed effects GLM in Stan using the building blocks that we have previously used. The following `.stan` program defines our model, and we will explain each of the blocks contained here in the following.

```
data {
  // number of observations
  int<lower = 0> Nobs ;
  // number of random effects groups (40 subjects)
  int<lower = 0> Nsubs ;
  // number of independent variables
  int<lower = 0> Npreds ;
  // the observations, note that they are a vector of binary outcomes
  int<lower = 0, upper = 1> y[Nobs] ;
  // vector associating each observation with its corresponding subject
  int<lower = 1, upper = Nsubs> subject[Nobs] ;
  // the independent variables
  matrix[Nobs, Npreds] X ;
}

parameters {
  vector[Nsubs] b ;
  real<lower = 0> sigmab ;
  vector[Npreds] beta ;
}

model {
  b ~ normal(0, sigmab) ;
  sigmab ~ cauchy(0, 1) ;
  for(n in 1:Nobs) {
    y[n] ~ bernoulli_logit(X[n] * beta + b[subject[n]]) ;
  }
}
```

```
generated quantities {
  real bTilde = normal_rng(0, sigmab) ;
  int<lower = 0, upper = 1> yTilde = bernoulli_logit_rng(X[1] * beta + bTilde) ;

}
```

4.4.4.1 The data block Remember that the `data` block specifies the information that we observe. Previously in our `hangover` example, this involved the dependent variable and the single independent variable. Here we have the same process, but because we have multiple independent variables, we must combine these into a *design matrix*. Let's discuss the structure of this matrix in more detail.

If the observations are collected in a vector $\mathbf{Y}$, with mean vector μ, we have

$$\mu = X\beta + Z\mathbf{b},$$

with $\mathbf{b}$ the vector of random effects. When we use functions such as `glmer`, the function will implicitly have been creating our design matrix for us from the model definition we provide. This is actually stored as part of the results of the fitted model, so we can use the command `model.matrix` to extract the design matrix X from the models we have already implemented. Here we'll implement the model equivalent to `glmSubjectRandom` from earlier, and this shows an extract of the design matrix:

```
##   (Intercept) Surfacenorm Visiondome Visionopen Sexmale   Age      Weight Dummy
## 1           1           1          0          1       1 -0.24 -0.05060137  -0.6
## 2           1           1          0          1       1 -0.24 -0.05060137  -0.2
## 3           1           1          0          0       1 -0.24 -0.05060137   1.6
## 4           1           1          0          0       1 -0.24 -0.05060137   0.1
## 5           1           1          1          0       1 -0.24 -0.05060137   0.1
## 6           1           1          1          0       1 -0.24 -0.05060137   1.7
```

4.4.4.2 The parameter block In our simple `hangover` regression, we had three parameters: the intercept, slope and error variance. We specified these as three separate continuous variables. In this more complex case we do not need to create a long list of parameters, but can instead use vectors. We declare as parameters:

- `sigmab`: the standard deviation σ_b in the random effects distribution $b_i \sim N(0, \sigma_b^2)$,
- `beta`: the coefficients β for all the fixed effects. Note in our notation above this is the collection $\{\alpha_2, \gamma_2, \gamma_3, \beta_0, \beta_1, \beta_2, \beta_4, \beta_5\}$ (remember, we have removed β_3 as this is the height coefficient), and
- `b`: the random effects themselves.

4.4.4.3 The model block Our random effects GLM definition consists of:

- The random effects that are normally distributed with zero mean, i.e. $b_i \sim N(0, \sigma_b^2)$ for $i = 1, \ldots, 40$.

- A prior distribution for the random effect variation, $\sigma_b \sim Cauchy(0, 1)$.
- The distribution of the response variable, which is Bernoulli, but with the logit link equation $\log(\frac{\mu}{1-\mu})$, i.e. $Y_j \sim Bernoulli(\mu_j)$ where $\text{logit}(\mu_j) = X_j\beta + b_i$ for $j = 1, \ldots, 480$ and i is the subject corresponding to observation j.
- A prior distribution for the vector coefficients, β. By leaving out this specification, Stan will assume an improper uniform prior, i.e. $f(\beta_l) \propto 1$ for the $l = 1, \ldots, 8$ as we have a set of 8 independent variables in our chosen model.

Some detail on the choice of the prior for σ_b. A suggestion (in Faraway) is to use a 'half-Cauchy', truncated at 0. This half-Cauchy distribution is given by

$$f(\sigma_b) = 2\frac{1}{\pi(1 + \sigma_b^2)},$$

for $\sigma_b > 0$, and $f(\sigma_b) = 0$ otherwise. To set this up in Stan, we specify `sigmab` as a real-valued parameter with lower bound 0, and then separately, give it a Cauchy distribution in the model block.

It's important to question our prior choices we make and whether the impact they have on our posterior is appropriate. The definition of the prior we have used involves $v = 1$, and has prior median for σ_b equal to 1. The 99th percentile of the prior is 64, which is implausibly large. We will return to this when we analyse our posterior results and consider whether this was appropriate.

4.4.4.4 The generated quantities block As we are building up our experience with using Stan, we will now introduce the `generated quantities` block. We can get Stan to sample directly from the posterior predictive distribution, and this is the most common use of the `generated_quantities` block. We'll consider a new subject, with design and covariate values $\tilde{x}$ which are equal to the first observation (i.e. the first row of the design matrix):

```
## (Intercept) Surfacenorm  Visiondome  Visionopen     Sexmale         Age
##        1.00        1.00        0.00        1.00        1.00       -0.24
##      Weight       Dummy
##       -0.05       -0.60
```

Let $\tilde{Y}$ be the observed response for this new subject. We want to sample from the distribution $\mathbb{P}(\tilde{Y}|Y, X, \tilde{x})$: we do so via

$$\mathbb{P}(\tilde{Y}|Y, X, \tilde{x}) = \int \mathbb{P}(\tilde{Y}|\tilde{b}, \tilde{x}, \beta) f(\tilde{b}|\sigma_b) f(\sigma_b, \beta|Y, X) d\tilde{b} d\sigma_b d\beta,$$

where $\tilde{b}$ is the random effect for the new subject. Hence we need to get Stan to sample $\tilde{b}$, and then sample $\tilde{Y}$ given the current sampled value of the parameters β, σ_b. Notice that we therefore do not predict the response of a new individual only as the contribution from the fixed effects (i.e. the expected value $\mathbf{X}\beta$), but we sample a subject random effect to add to this fixed effects value *and* then sample from the binomial distribution with such an expected value.

4.4.4.5 Sampling and output As with previous examples, we group our data into the list form that Stan requires, and use the `sampling` function to implement the HMC algorithm:

```
glmData <- list(Nobs = nrow(balance),
                Nsubs = length(unique(balance$Subject)),
                Npreds = ncol(designMatrix),
                y = balance$Stable,
                subject = as.numeric(balance$Subject),
                X = designMatrix)

myfit <- sampling(glmRandomEffects,
                  data = glmData,
                  cores = 7,
                  refresh = 0)
```

Let's explore the output from our fitted model in Stan. We can print out a summary of our results, as previously, and explore the trace-plots for convergence. Note that the summary print-out includes:

- the subject random effects variance, σ_b^2,
- the fixed effects coefficients β,
- all 40 of the subject-specific random effects themselves, $b_1, \ldots, b_{40}$,
- the samples from the posterior predictive distribution, $\tilde{b}, \tilde{Y}$, and
- the Hamiltonian energy itself.

In reality, we are probably not interested in the actual random effects, so we will omit these from the results plots in the following.

```
## Inference for Stan model: c808de51faf75c5c13f75aad1c4debdc.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean   sd    2.5%     50%   97.5% n_eff Rhat
## b[1]       0.02    0.04 1.69   -3.32    0.02    3.32  1550    1
## b[2]      -1.76    0.05 1.91   -5.70   -1.72    1.78  1793    1
## b[3]      -6.40    0.07 2.44  -11.90   -6.10   -2.37  1079    1
## b[4]      -4.12    0.05 2.01   -8.56   -3.99   -0.47  1349    1
## b[5]      -2.16    0.04 1.91   -6.07   -2.13    1.44  2044    1
## b[6]       2.36    0.04 1.56   -0.59    2.34    5.51  1330    1
## b[7]      -1.60    0.04 1.78   -5.18   -1.50    1.72  2338    1
## b[8]      -0.03    0.04 1.91   -3.72   -0.01    3.62  2300    1
## b[9]      -0.01    0.03 1.82   -3.70    0.02    3.55  3642    1
## b[10]      1.41    0.03 1.40   -1.32    1.42    4.11  1714    1
## b[11]     -1.56    0.04 1.79   -5.19   -1.51    1.77  2237    1
## b[12]     -1.59    0.04 1.84   -5.51   -1.53    1.86  2280    1
## b[13]     -1.53    0.04 1.88   -5.33   -1.49    1.99  1882    1
## b[14]     -5.11    0.06 2.56  -11.14   -4.78   -1.06  1587    1
## b[15]      0.05    0.03 1.77   -3.38    0.06    3.49  3619    1
## b[16]      4.24    0.05 1.72    1.01    4.18    7.83  1344    1
## b[17]      3.22    0.04 1.59    0.23    3.15    6.48  1390    1
## b[18]      0.21    0.04 1.88   -3.45    0.20    3.89  2188    1
```

```
## b[19]      0.33    0.04 1.69   -3.15    0.37    3.59  1715    1
## b[20]      1.20    0.05 1.75   -2.19    1.19    4.69  1282    1
## b[21]      3.50    0.06 1.86    0.13    3.38    7.42   957    1
## b[22]     -2.24    0.04 1.92   -6.21   -2.20    1.50  2021    1
## b[23]     -1.61    0.05 2.17   -5.95   -1.55    2.61  2037    1
## b[24]     -1.63    0.04 1.80   -5.38   -1.56    1.71  2299    1
## b[25]      5.54    0.05 1.82    2.35    5.42    9.44  1406    1
## b[26]      3.45    0.05 1.75    0.24    3.39    7.20  1267    1
## b[27]      7.50    0.07 2.05    4.07    7.31   12.02   980    1
## b[28]     -0.01    0.04 1.93   -3.76   -0.01    3.81  2355    1
## b[29]      4.83    0.05 1.91    1.39    4.73    8.96  1292    1
## b[30]     -4.84    0.06 2.41  -10.30   -4.57   -0.89  1595    1
## b[31]      2.29    0.05 1.81   -1.15    2.22    6.09  1305    1
## b[32]      2.29    0.04 1.62   -0.88    2.28    5.42  1550    1
## b[33]      3.37    0.05 1.70    0.25    3.29    6.81  1329    1
## b[34]     -1.63    0.04 1.84   -5.47   -1.56    1.74  2301    1
## b[35]      3.31    0.04 1.44    0.63    3.29    6.31  1441    1
## b[36]      0.10    0.04 1.95   -3.81    0.11    3.91  2600    1
## b[37]     -4.93    0.07 2.53  -10.93   -4.65   -0.77  1479    1
## b[38]     -4.73    0.07 2.57  -10.68   -4.54   -0.20  1541    1
## b[39]      0.22    0.04 1.91   -3.42    0.21    3.99  2251    1
## b[40]     -2.22    0.04 1.87   -6.08   -2.17    1.39  2230    1
## sigmab     3.63    0.03 0.79    2.39    3.54    5.44   660    1
## beta[1]  -12.98    0.09 2.34  -18.40  -12.72   -9.09   621    1
## beta[2]    8.38    0.05 1.30    6.18    8.26   11.36   678    1
## beta[3]    0.75    0.01 0.55   -0.28    0.75    1.84  3526    1
## beta[4]    7.08    0.05 1.24    5.03    6.96   10.00   651    1
## beta[5]    2.43    0.07 2.08   -1.30    2.30    6.81   886    1
## beta[6]    0.12    0.06 2.12   -4.14    0.13    4.16  1234    1
## beta[7]   -0.62    0.12 3.76   -8.15   -0.60    6.39  1047    1
## beta[8]    0.24    0.00 0.25   -0.25    0.24    0.72  4292    1
## bTilde    -0.04    0.06 3.79   -7.44   -0.05    7.65  3693    1
## yTilde     0.86    0.01 0.35    0.00    1.00    1.00  3667    1
## lp__    -154.44    0.28 7.12 -169.55 -154.05 -141.87   654    1
##
## Samples were drawn using NUTS(diag_e) at Thu Feb 10 20:21:59 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The trace plots of the four chains (only showing the samples after the burn-in period) are given in Figure 28. This shows the σ_b^2 parameter and the fixed effects coefficients β, but re-labelled to add some descriptive context to the interpretation of each.

Posterior credible intervals are shown for these parameters in Figure 29. Again, the parameters have been relabelled to add some descriptive context to each.

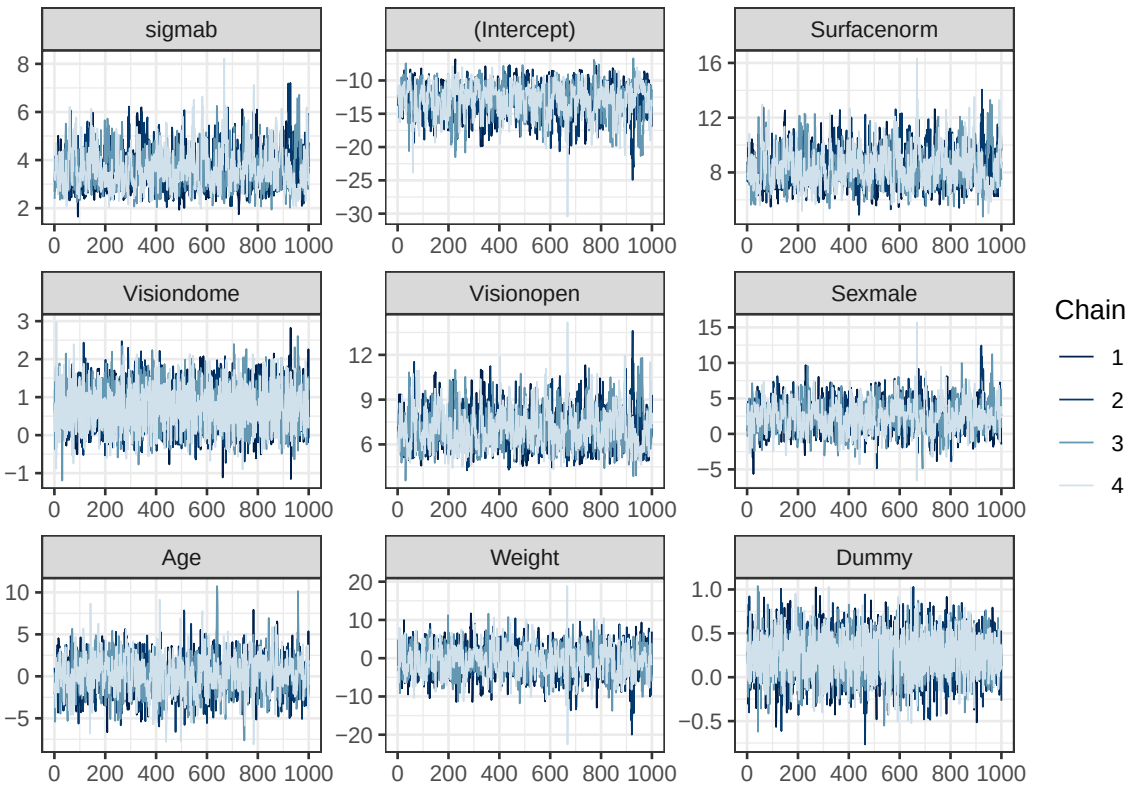


Figure 28: Trace plots of the sampled parameters for the balance experiment, fitting a random effects GLM in Stan. Note that the beta parameters have been relabelled here descriptively to add context.

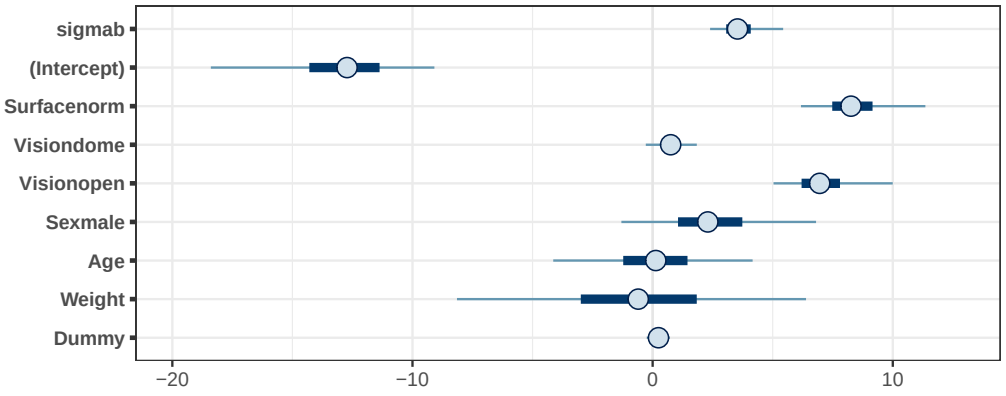


Figure 29: Posterior credible interval summaries for the balance experiment, fitting a random effects GLM in Stan. Note that the beta parameters have been relabelled here descriptively to add context. Point gives the posterior median, bar gives the interval with quantiles at 0.25 and 0.75, and the line gives the quantiles at 0.025 and 0.975.

Table 10: Prior and Posterior percentiles of the subject random effects parameter.

	Prior	Posterior
1%	0.016	2.2
5%	0.079	2.5
50%	1.000	3.5
95%	13.000	5.1
99%	64.000	6.0

Exercise
What information can we determine from these plots?
Are you satisfied about the convergence of the HMC sampler? What other diagnostics could you explore?
What do the posterior credible intervals suggest about our full proposed model?

4.4.4.6 Examining the effect of our choice of prior We will look more carefully at σ_b, the subject random effect. Recall that this was the only parameter we set a 'proper' prior for, using a half-Cauchy, and it featured a heavy right tail. It's important to check that our posterior is not just a reflection of our prior, and that there truly is information from that data here.

We compare the prior and posterior tails for σ_b in Table 10. We see that the data does seem to have pulled in the posterior; both away from the heavy tail, and away from the prior median. To explore this visually, Figure 30 shows the prior density for σ_b (black line), along with a histogram of the posterior samples (blue) and the estimate of σ_b from the likelihood-based approach using `lme4` that we implemented earlier. Note that `lme4` does not give error estimates of random effect parameters, so we would have nothing more than this point estimate were we using the maximum likelihood approach.

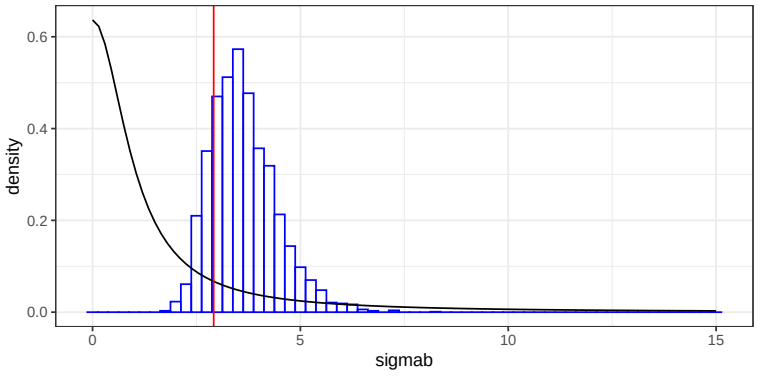


Figure 30: Prior and posterior comparison of the subject random effects parameter. The prior density is given (black) along with posterior histogram (blue), and the likelihood-based estimate from lme4 (red).

Table 11: Comparison of the fixed effects parameter estimates between Stan (HMC) and lme4 (likelihood-based).

	Stan mean	Stan sd	lme4 mean	lme4 sd
(Intercept)	-13.000	2.340	-11.2000	1.870
Surfacenorm	8.380	1.300	7.3300	1.070
Visiondome	0.752	0.550	0.6780	0.530
Visionopen	7.080	1.240	6.1400	0.993
Sexmale	2.430	2.080	1.8900	1.650
Age	0.118	2.120	0.0502	1.630
Weight	-0.619	3.760	-0.0591	3.040
Dummy	0.242	0.247	0.2400	0.231

4.4.4.7 Examining the new subject prediction Recall that we included a command in our Stan program to sample from the posterior predictive distribution of a new subject who has the same covariate values as subject 1. Note that we are therefore assuming that this individual has the same age, height, weight, etc. as 'subject 1', but is not a new observation on that particular subject, but merely a different person with those covariate measurements. We can extract the results of these two variables to obtain a predictive credible interval. Remember that $\tilde{b}$ is the posterior predicted random effect of the new subject, and $\tilde{Y}$ is the posterior predicted balance outcome of that subject.

```
## Inference for Stan model: c808de51faf75c5c13f75aad1c4debdc.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##         mean se_mean   sd  2.5%   25%   50% 75% 97.5% n_eff Rhat
## bTilde -0.04    0.06 3.79 -7.44 -2.48 -0.05 2.3  7.65  3693    1
## yTilde  0.86    0.01 0.35  0.00  1.00  1.00 1.0  1.00  3667    1
##
## Samples were drawn using NUTS(diag_e) at Thu Feb 10 20:21:59 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

4.4.5 Comparing Stan and lme4

Let's compare the results that are obtained from fitting this random effect GLM using a likelihood-based approach (`lme4`) and a Bayesian approach (`rstan`). The estimated mean and standard deviation for the fixed effects parameters are given in Table 11. We also show these results in Figure 31. In this figure, the Bayesian approach from `rstan` is displayed as posterior credible intervals using the samples from the posterior, whereas the likelihood approach from `lme4` is displayed as confidence intervals obtained by applying a normal approximation (i.e. $2 \times s.e.$).

Additionally, we can explore the subject-specific random effects, including the predictive distri-

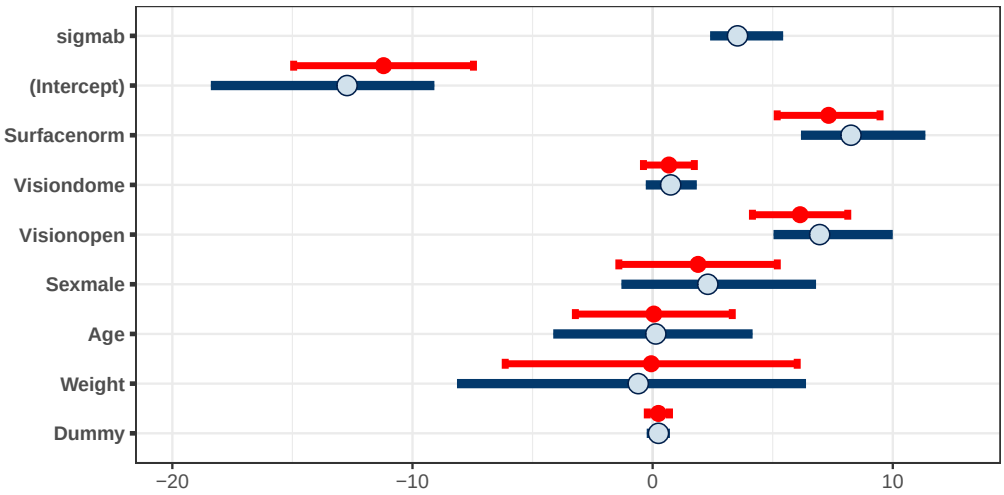


Figure 31: Estimates of the parameters in the balance model. The blue circles/bars give the posterior meadian/(0.025,0.975) credible interval from the Bayesian approach, and the red circles/bars give the estimate/(0.025,0.975) approximate confidence interval. Note that lme4 does not report a standard error for random effects parameters.

bution. Figure 32 shows the estimates of the random effects for each of the 40 subjects in the experiment. It also shows the posterior predictive interval for the subject-specific effect of a new, unknown individual. The Bayesian results are shown in blue. For this new individual, we compare with $\pm 2\hat{\sigma}_b$, where $\hat{\sigma}_b$ is the estimate obtained via `lme4` (red).

4.4.6 'Easy' Bayesian approach with `brms`

We have implemented a Bayesian approach to regression analysis using Stan, and this took a fair amount of set-up, creating the `.stan` file with the appropriate design matrix construction. The flexibility of creating the `.stan` file ourselves is that we can define our model however we like, along with the priors and any generated quantities, such as the predictive distribution we sampled from in our example. However, if we do not need to include these flexible choices, there is a rather quick way we can implement this model, using the package `brms`, that retains the same syntax that we are familiar with from `lm` and `lme4`.

```
glmBrm <- brm(Stable ~ Surface + Vision +
                Sex +
                Age + Weight + Dummy +
                (1 | Subject),
              family = bernoulli,
              data = balance)
```

We can summarise and compare the output here with what we obtained ourselves in Stan:

```
summary(glmBrm)
```

```
##  Family: bernoulli
```

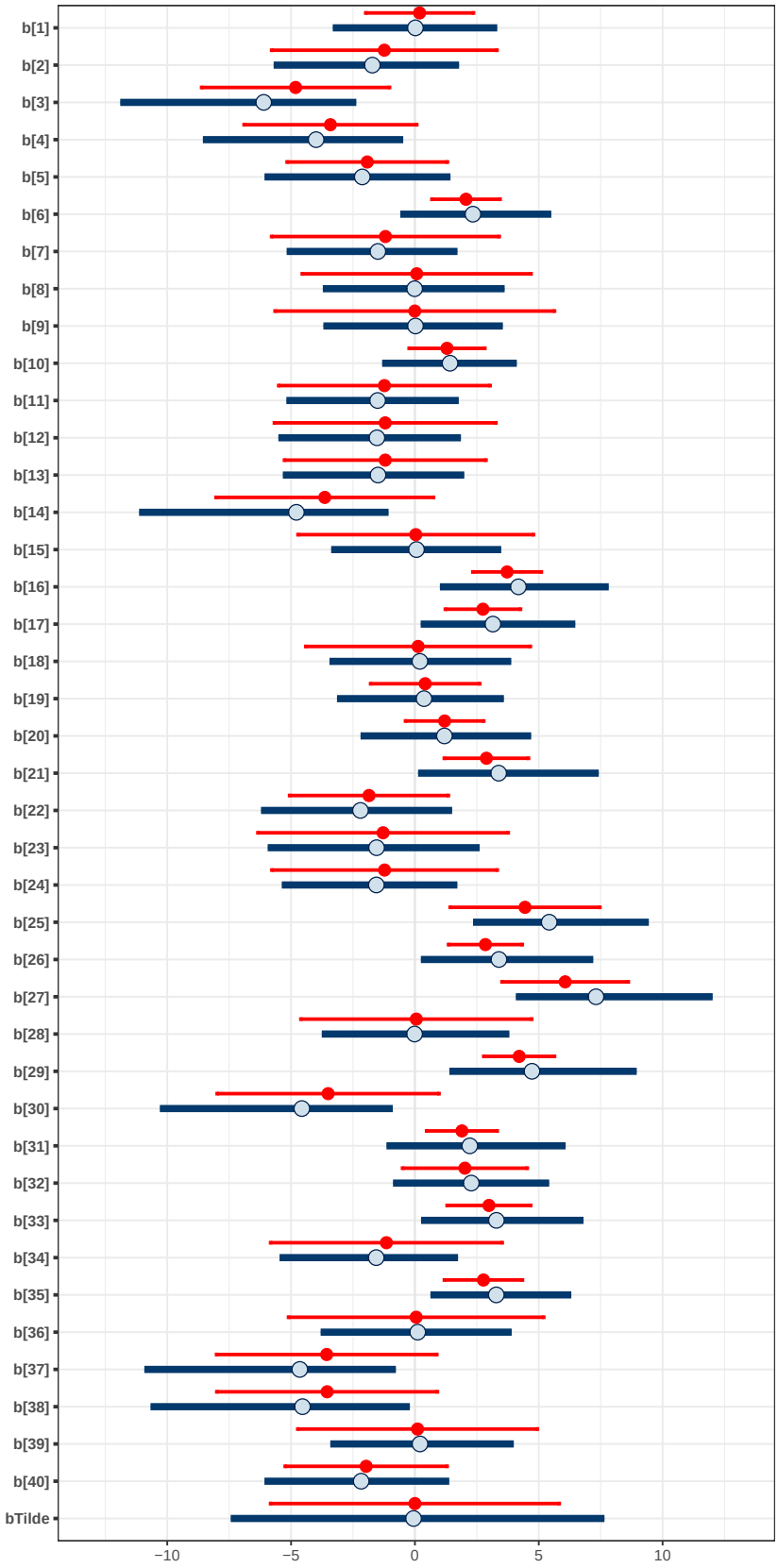


Figure 32: Comparison of the subject-specific random effects. The (0.025,0.975) posterior credible intervals of each subject's effect obtained via a Bayesian approach is shown in blue, and the approximate confidence interval obtained via a likelihood approach is shown in red. In additon to the estimates of the 40 subjects from the experiment, we also include the prediction for a new unknown individual, using bTilde. In blue we show the posterior predictive median (point) and (0.025,0.975) credible predictive interval obtained via the Bayesian approach. In red we show the interval given by plus and minus 2 times the estimate of the random effect deviation.

```
##   Links: mu = logit
## Formula: Stable ~ Surface + Vision + Sex + Age + Weight + Dummy + (1 | Subject)
##    Data: balance (Number of observations: 480)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Group-Level Effects:
## ~Subject (Number of levels: 40)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     3.47      0.73     2.32     5.09 1.00     1197     1933
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     -12.18      2.10   -16.71    -8.65 1.00     1558     1987
## Surfacenorm     7.99      1.18     6.00    10.61 1.00     2150     2040
## Visiondome      0.72      0.56    -0.38     1.82 1.00     5816     2562
## Visionopen      6.73      1.10     4.84     9.11 1.00     2089     2034
## Sexmale         2.21      1.98    -1.55     6.37 1.00     1386     1832
## Age            -0.04      1.99    -3.91     4.02 1.00     1381     2119
## Weight         -0.29      3.60    -7.19     6.91 1.00     1388     1909
## Dummy           0.24      0.25    -0.24     0.71 1.00     5251     2641
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

As you can see, this approach retains interpretable variable labelling without us needing to, which is a bonus! Comparable results to our analysis in Stan can be seen by the trace plot in Figure 33 and the caterpillar plot in Figure 34. As we can see, these results are very similar to the Stan implementation. Note that both are stochastic processes so we cannot expect identical results. However, there is an implementation difference here because we used a specified prior in our custom Stan implementation.

```
## Warning: 'parnames' is deprecated. Please use 'variables' instead.
```

4.4.7 What have we found?

The main aim of this example was to explore a sensible process for implementing a regression model using Stan for a non-simplified, real data example. However, it's useful to summarise our analysis findings nonetheless.

We've found that there is significant evidence to suggest that there is a difference in balance for the two surfaces explored, and also between restricted and unrestricted vision. There's evidence to suggest that there is a small difference between the two ways of restricting vision. There is also evidence to suggest a small difference in balance between the genders, but that other covariates are unlikely to provide useful predictive information. Somewhat comfortingly, we've shown (with little

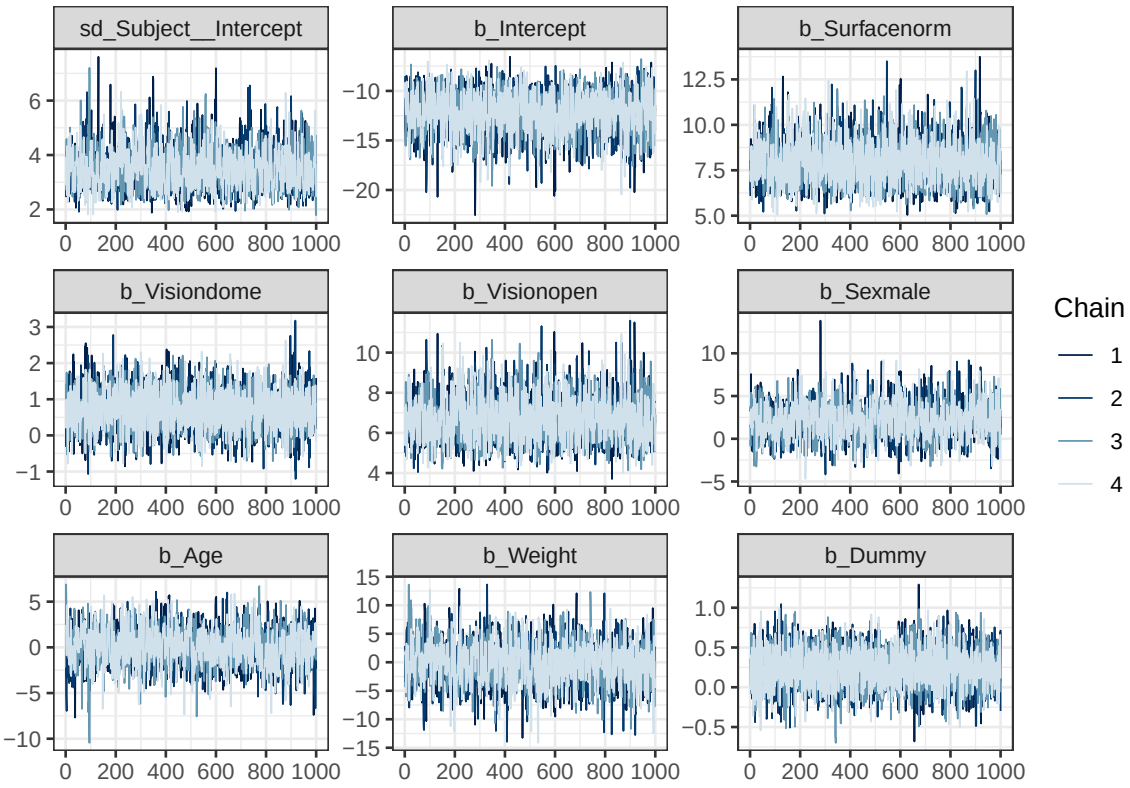


Figure 33: Trace plots of the sampled parameters for the balance experiment, fitting a random effects GLM in Stan using the package brms.

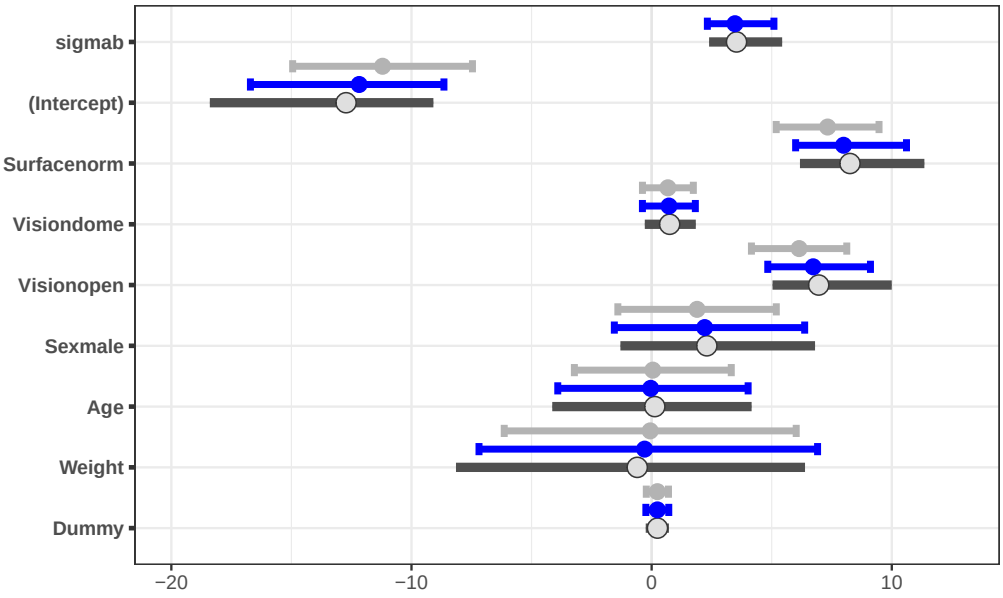


Figure 34: Estimates of the parameters in the balance model. The dark grey is the credible interval from the Bayesian approach in Stan, the light grey is the approximate confidence interval from lme4. The new edition here is the blue credible intervals from the Bayesian approach with brms.

uncertainty!) that the dummy variable is uninformative for explaining balance. Finally, we've seen evidence of a variation between individuals.

Note that we have not investigated model fit or validity of assumptions here, though this is an important aspect of model fitting analysis.

Part II

Inference with missing data

5 Multiple imputation for missing data

Aims of this chapter

1. Introduce and discuss statistical issues with missing data.
2. Understand the multiple imputation approach.
3. Implement and interpret the results of multiple imputation in practice using the R package `mice`.

5.1 Introduction

It is common to encounter data sets where some of the values are missing, for example, in a sample survey if a respondent does not answer all the questions, or if a process for recording data wasn't working on a particular day. Some scenarios are as follows.

1. Missing data we may choose to ignore.

Consider, for example, the `airquality` data frame in R:

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Each `NA` indicates a missing value. If we fit a linear model in R using these data, R will ignore any row with a missing value. This is sometimes referred to as a **complete case** analysis.

2. Censored data

In a clinical trial, we may observe 'survival data': the time T taken for some event to occur for a patient. Suppose, for example, a trial lasts for two years, and that for a particular patient, the event hasn't occurred by the end of the trial. For this patient, we have a censored observation: we only know $T > 2$; the actual value of T is not observed. Here, we can incorporate this observation into our analysis through appropriate specification of the likelihood function. The contribution to the likelihood from this patient's observation would be of the form $P(T > 2|\theta)$, rather than a density $f_T(t|\theta)$, had we observed $T = t$.

3. Bias

Table 13: Extract of the nhanes dataset.

age	bmi	hyp	chl
1	NA	NA	NA
2	22.7	1	187
1	NA	1	187
3	NA	NA	NA
1	20.4	1	113
3	NA	NA	184
1	22.5	1	118
1	30.1	1	187
2	22.0	1	238
2	NA	NA	NA

In a sample survey, the probability of a participant answering a question may depend on the answer the participant would give. For example, in a political opinion poll, participants with particular views may be less likely to answer certain questions. This is perhaps the hardest type of missing data to deal with; ignoring it could lead to biased results. We may have to make strong modelling assumptions, or seek additional data that would help us understand potential biases.

In this chapter, we will study one particular technique for dealing with missing data: multiple imputation. We'll consider this from a Bayesian perspective, so the aim will be to derive posterior distributions in situations with missing data.

5.2 Example: `nhanes` data

We will be using an example with a dataset `nhanes` from the `mice` package (van Buuren and Groothuis-Oudshoorn, 2011), which comprises 25 observations of 4 variables. An extract of this data is shown in Table 13, which includes a number of missing cells.

Using the `mice` package, we can investigate the missing data pattern of our dataset. Our classic tools for exploratory data analysis do still stand and are somewhat informative, such as `summary` will tell us the number of missing observations per variable. However, the pattern between variables does not come across here:

```
summary(mice::nhanes)
```

```
##       age            bmi             hyp             chl       
##  Min.   :1.00   Min.   :20.40   Min.   :1.000   Min.   :113.0  
##  1st Qu.:1.00   1st Qu.:22.65   1st Qu.:1.000   1st Qu.:185.0  
##  Median :2.00   Median :26.75   Median :1.000   Median :187.0  
##  Mean   :1.76   Mean   :26.56   Mean   :1.235   Mean   :191.4  
##  3rd Qu.:2.00   3rd Qu.:28.93   3rd Qu.:1.000   3rd Qu.:212.0  
##  Max.   :3.00   Max.   :35.30   Max.   :2.000   Max.   :284.0  
##                 NA's   :9       NA's   :8       NA's   :10
```

In the below, we use `md.pattern()` to obtain this information, which can both print out this missing pattern, and display it with an image, as given in Figure 35.

Each column represents one variable in the data, but each row is not a subject. Each row is a missingness pattern across the set of variables, that is present in the data. The left column of figures gives the number of subjects with that missing pattern, and the right column of figures gives the number of missing data that pattern involves (i.e. how many red squares are in that row). The bottom row of numbers therefore counts how many subjects have missing data for each variable.

```
mice::md.pattern(mice::nhanes)
```

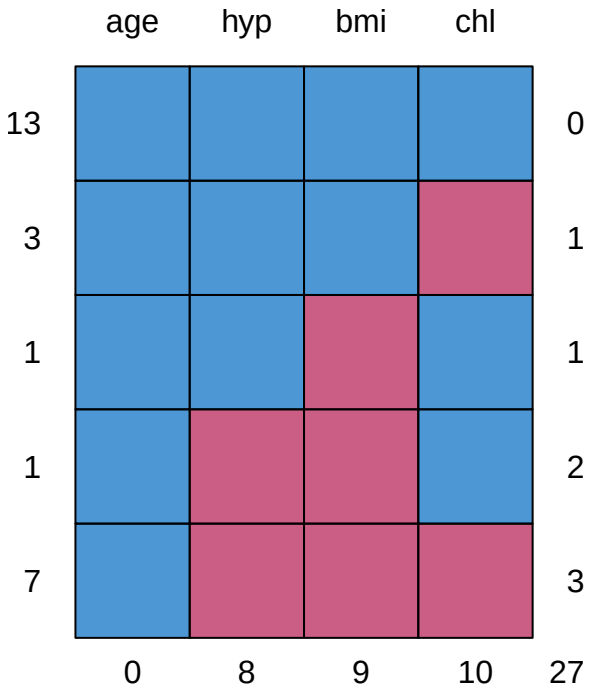


Figure 35: Missing data pattern of the nhanes dataset.

```
##    age hyp bmi chl
## 13   1   1   1   1  0
## 3    1   1   1   0  1
## 1    1   1   0   1  1
## 1    1   0   0   1  2
## 7    1   0   0   0  3
##      0   8   9  10 27
```

For the `nhanes` data, we have 13 subjects with full information, 4 subjects who have one variable out of the four missing, 1 subject with two variables missing, and finally 7 subjects who have the same set of the three variables missing. Note that all 25 subjects have observations of the variable `age`.

5.3 Mechanisms of missingness

Suppose we have data $Y = (y_{ij})$ given by the $n \times k$ matrix, which will typically contain information about k variables from n subjects. We distinguish the *pattern* of the missing data by introducing the **missing-data indicator matrix**, $R = (r_{ij})$, such that $r_{ij} = 1$ if y_{ij} is missing and $r_{ij} = 0$ if y_{ij} is present.

The pattern of missing data arises due to the *mechanism* of missingness. This is characterised by the conditional distribution of R given Y and two sets of parameters:

- θ: the parameters of interest, i.e. parameters in a likelihood $f(Y|\theta)$;
- ϕ: nuisances parameters that are related to the missing data mechanism only, e.g. a probability of non-response in a sample survey.

In the following discussion, we will assume θ and ϕ are independent, and that conditional on Y, we have independence between R and θ: given Y, missingness is not dependent on the parameters in the distribution of Y. We will therefore consider missingness mechanisms in terms of the distribution $f(R|Y, \phi)$.

Three possible mechanisms are as follows.

5.3.1 Missing completely at random (MCAR)

Missingness does not depend on the value of the data, so

$$f(R|Y, \phi) = f(R|\phi),$$

for all Y, ϕ. This is the ideal situation. Note that this name is slightly confusing—the pattern of missing data given by R need not be random, but merely that it is independent of the data values themselves.

5.3.2 Missing at random (MAR)

Missingness depends only on the components of Y that are observed, denoted Y_{obs}, and not on the components that are unobserved, denoted Y_{mis}. So

$$f(R|Y, \phi) = f(R|Y_{\text{obs}}, \phi),$$

for all Y_{mis}, ϕ. Here, the probability of a data point being missing can depend on the value of other variables that are not missing. For example, in a questionnaire there may be a question on salary that older age groups are more likely to not answer in comparison with younger age groups.

Note that MCAR implies MAR.

5.3.3 Not missing at random (NMAR)

Missingness can depend both on the missing and observed variables, so that $f(R|Y, \phi)$ cannot be simplified. For example, respondents to a questionnaire may be more likely to leave a question on salary unanswered dependent upon their actual salary. This is the hardest mechanism of missing data to handle and can lead to serious biases.

5.4 Ignoring information about missingness

When we have missing data, the observed data should be thought of as both Y_{obs} *and* R: the target posterior is $f(\theta|Y_{\text{obs}}, R)$. We can ignore R if we assume that the data are missing at random (and hence also if the data are MCAR). For the joint posterior distribution of θ, ϕ we have

$$f(\theta, \phi|Y_{\text{obs}}, R) \propto \pi(\theta)\pi(\phi)f(Y_{\text{obs}}, R|\theta, \phi), \tag{31}$$

$$= \pi(\theta)\pi(\phi)f(Y_{\text{obs}}|\theta)f(R|Y_{\text{obs}}, \phi), \tag{32}$$

assuming the data are MAR. Hence for the marginal posterior of θ we have

$$f(\theta|Y_{\text{obs}}, R) \propto \pi(\theta)f(Y_{\text{obs}}|\theta),$$

i.e. the posterior obtained if we ignore R.

5.5 Inference via imputation

From now on, we will assume our data are MAR, so that R can be ignored.

Returning to the `nhanes` data, suppose we want to fit a regression model with `bmi` as the dependent variable, and the other three variables as the independent variables. Amongst the missing data, we note one observation where `bmi` is observed, but `chl` is not:

```
mice::nhanes[24, ]
```

```
##    age  bmi hyp chl
## 24   3 24.9   1  NA
```

Should observations like this be ignored? As `age` and `hyp` were still observed, wouldn't there still be *some* information about the relationship between dependent and independent variables from this observation? In general, we may have a dataset where the proportion of *complete cases* (observations with no missing variables) is relatively small; if we only conduct a "complete case analysis" (where any incomplete rows are discarded), we may be ignoring a lot of data.

Here, we define

- Y_{obs}: all the observed individual observations, including any observed variables within rows containing missing data.
- Y_{mis}: all the individual missing observations. (Together, $(Y_{\text{obs}}, Y_{\text{mis}})$ would give a complete data frame in R).
- θ: the model parameters of interest.

If we are to make use of all the data we have, we want to derive $f(\theta|Y_{\text{obs}})$. But this will involve a likelihood $f(Y_{\text{obs}}|\theta)$ that we can't easily write down (how would we fit a linear model where some of the independent variables are missing?) The main idea of multiple imputation is to derive the posterior via

$$f(\theta|Y_{\text{obs}}) = \int f(\theta|Y_{\text{obs}}, Y_{\text{mis}}) f(Y_{\text{mis}}|Y_{\text{obs}}) dY_{\text{mis}}$$

The posterior $f(\theta|Y_{\text{obs}}, Y_{\text{mis}})$ is the posterior of θ given a complete data set, and so will be easier to deal with. We do, however, have a new problem, which is to consider a distribution of the missing data conditional on the observed data; we discuss this later.

Rather than trying to evaluate the integral analytically, we will use a Monte Carlo approach. In outline:

1. we sample a set of missing data $Y_{\text{mis}}^{(i)}$;
2. we derive a posterior $f(\theta|Y_{\text{obs}}, Y_{\text{mis}}^{(i)})$;
3. we repeat steps (1) and (2) m times, and then average over our distributions $f(\theta|Y_{\text{obs}}, Y_{\text{mis}}^{(1)}), ... f(\theta|Y_{\text{obs}}, Y_{\text{mis}}^{(m)})$..

In effect, we are approximating the posterior of interest with

$$f(\theta \mid Y_{\text{obs}}) \approx \frac{1}{m} \sum_{i=1}^{m} f(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}}),$$

where $Y_{\text{mis}}^{(i)} \sim f(Y_{\text{mis}} \mid Y_{\text{obs}})$.

The process of filling in the missing values is known as imputation, and because we do this multiple times, we refer to it as multiple imputation.

5.5.1 Further simplifications

Although deriving $f(\theta|Y_{\text{obs}}, Y_{\text{mis}}^{(i)})$ is simpler than deriving $f(\theta|Y_{\text{obs}})$, it still may be difficult, or involve substantial computational effort. We may choose to make two further simplifications:

1. Instead of obtaining the full posterior distribution $f(\theta \mid Y_{\text{obs}})$, we may just obtain the posterior mean and variance of θ, using only $E(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$ and $Var(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$ for $i = 1, ..., m$
2. We may choose to obtain quick approximations of $E(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$ and $Var(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$, for example (in a linear models setting) using a least squares estimate to approximate $E(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$ and the associated standard error (squared) to approximate $Var(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$.

5.6 Pooling multiple imputations

Suppose we have already we have already obtained m imputed datasets $(Y_{\text{mis}}^{(1)}, Y_{\text{obs}}), ..., (Y_{\text{mis}}^{(m)}, Y_{\text{obs}})$. We will use the simplification of obtaining approximate expressions for the posterior mean and variance $\theta|Y_{\text{obs}}$; we will not attempt to derive the full posterior $f(\theta|Y_{\text{obs}})$

We estimate $E(\theta|Y_{\text{obs}})$ by noting that

$$\mathbb{E}(\theta \mid Y_{\text{obs}}) = \mathbb{E}(\mathbb{E}(\theta \mid Y_{\text{mis}}, Y_{\text{obs}}) \mid Y_{\text{obs}})$$

and then using the estimate

$$\bar{\theta} := \frac{1}{m}\sum_{i=1}^{m} \hat{\theta}^{(i)}, \tag{33}$$

where

$$\hat{\theta}^{(i)} = \mathbb{E}(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}})$$

In the above, the first line uses the tower property of expectation. This equality says that the posterior mean is the average of the posterior means over repeatedly imputed data. The approach for combining the results of multiple imputation therefore arises by approximating the average by a sample mean of the estimator from our m imputations.

We estimate $\text{Var}(\theta|Y_{\text{obs}})$ by first noting that

$$\text{Var}(\theta \mid Y_{\text{obs}}) = \mathbb{E}(\text{Var}(\theta \mid Y_{\text{mis}}, Y_{\text{obs}}) \mid Y_{\text{obs}}) + \text{Var}(\mathbb{E}(\theta \mid Y_{\text{mis}}, Y_{\text{obs}}) \mid Y_{\text{obs}}),$$

and then using the estimate

$$T := \bar{U} + \left(1 + \frac{1}{m}\right) B, \tag{34}$$

where $\bar{U}$ is an estimate of $\mathbb{E}(\text{Var}(\theta \mid Y_{\text{mis}}, Y_{\text{obs}}) \mid Y_{\text{obs}})$, given by

$$\bar{U} := \frac{1}{m}\sum_{i=1}^{m} U^{(i)}, \quad U^{(i)} := \text{Var}(\theta \mid Y_{\text{mis}}^{(i)}, Y_{\text{obs}}),$$

and B is an estimate of $\text{Var}(\mathbb{E}(\theta|Y_{\text{mis}}, Y_{\text{obs}})|Y_{\text{obs}})$, given by

$$B := \frac{1}{m-1}\sum_{i=1}^{m} \left(\hat{\theta}^{(i)} - \bar{\theta}\right)^2,$$

and the factor $(1+1/m)$ is a correction for bias, the details of which we won't consider here.

Writing

$$T = \bar{U} + B + \frac{B}{m}, \tag{35}$$

we can summarise this as the posterior uncertainty of θ comprising of three sources:

1. The uncertainty caused by the fact that we are taking a sample of observations Y and estimating θ from this. This is the uncertainty that we would always have surrounding an estimate from complete data, as Y is still only a sample and not the entire population of interest. This contribution is $\bar{U}$.

Table 14: Simple dataset with one missing value.

Y	X1	X2
2.86	1.78	0.02
5.77	2.39	2.52
0.24	-2.68	1.06
2.71	-0.11	0.61
-3.67	NA	-1.67
-2.99	1.59	-2.26

2. The uncertainty caused by the missing data. This contribution to the uncertainty is given by B. Each imputed dataset represents a sample from the possible range of Y_{mis}, so B is the variability in the parameter estimates resulting from the range of possible missing data values.
3. The uncertainty caused by the fact that we have only implemented m imputed datasets and not the actual full set of possible Y_{mis}. The contribution to the uncertainty from this aspect is $\frac{B}{m}$.

Because any analysis will contain uncertainty of the type described in point (1), it is useful for us to evaluate the uncertainty that is arising because of the missing data and our imputation efforts. The proportion of the variance attributable to missing data and imputation out of the total variance can be expressed as

$$\lambda_m := \frac{(1+\frac{1}{m})\,B}{\bar{U}+(1+\frac{1}{m})\,B}.$$

5.7 Simple example

We have the data in Table 14, which contains the three variables Y, X_1, X_2, and there is one missing observation of X_1.

The aim of this analysis is to build a regression model to predict Y from the two remaining variables

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i,$$

where $\epsilon \sim N(0, \sigma^2)$.

5.7.1 Multiple imputation by hand

The multiple imputation work-flow consists of three steps:

1. Imputing $Y^{(1)}, \ldots, Y^{(m)}$ from $f(Y_{\text{mis}}|Y_{\text{obs}})$;
2. Calculating the statistic of interest from the imputations, $\hat{\theta}_1, \ldots, \hat{\theta}_m$; and
3. Combining the imputations as $\bar{\theta}$.

We have not yet discussed sophisticated methods for imputing datasets. For now, we will use a simple imputation method that imputes a missing value as equal to a randomly selected observation from that variable. In this example, this corresponds to a simple choice for the distribution

$f(Y_{\text{mis}}|Y_{\text{obs}})$, in which the missing data (a single missing value of `X1`) has a discrete uniform distribution, constructed from the observed values of `X1`.

Our statistic of interest here will be the estimates of the parameters $\beta_0, \beta_1, \beta_2$. We will therefore store the estimate of these coefficients by fitting the proposed linear model to each imputed dataset. This forms our $\hat{\theta}_1, \ldots, \hat{\theta}_m$.

For our estimates of uncertainty—see Equation (34)—we will also need the variance of our parameter estimates. For a linear model we access the covariance matrix with `vcov()`. For simplicity here, we'll just store the diagonal elements of these and not consider the covariances between the parameters.

Our imputation 'by hand' is given by the below (note that this has not been optimised in any way, but rather written for clarity of the process):

```
m <- 30
imputations <- rep(NA, m)
# store the imputed coefficients beta0, beta1, beta2
theta <- matrix(NA, nrow = m, ncol = 3)
# store the variance of each coefficient
V <- matrix(NA, nrow = m, ncol = 3)

for (i in 1:m) {

  imp <- sample(simple_data$X1[!is.na(simple_data$X1)], 1)
  complete_data <- simple_data
  complete_data$X1[is.na(complete_data$X1)] <- imp
  imputations[i] <- imp

  fit <- lm(Y ~ X1 + X2, data = complete_data)
  theta[i, ] <- coef(fit)
  V[i, ] <- diag(vcov(fit))
}
```

In the above we have implemented 30 imputations. An extract of the imputed parameter estimates $\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(30)}$ is given here, with the three columns being for $\beta_0, \beta_1, \beta_2$, respectively. Note that some of these rows are the same—this is because our simple imputation method meant that our single missing value was randomly chosen from the five observations of that variable, so we know there will be repetitions in our imputations.

```
head(theta)
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.2980052 0.4800095 1.996850
## [2,] 0.2911767 0.5745578 1.974844
## [3,] 0.4016995 0.6943554 1.871238
## [4,] 0.2980052 0.4800095 1.996850
## [5,] 0.4016995 0.6943554 1.871238
## [6,] 0.2911767 0.5745578 1.974844
```

We pool our estimates to obtain $\bar{\theta}$ by taking the sample mean of the imputed parameter estimates:

```
(theta_bar <- colMeans(theta))
```

```
## [1] 0.3840886 0.5867359 1.9124154
```

We calculate the variance of the pooled estimate using Equation (34) as:

```
(theta_var <- colMeans(V) + (1 + 1/m) * diag(cov(theta)))
```

```
## [1] 0.4666031 0.1300045 0.1601651
```

We calculate the proportion of the variance due to the missing observation as:

```
(lambda <- ((1 + 1/m) * diag(cov(theta))) / theta_var)
```

```
## [1] 0.05311199 0.03803215 0.07565886
```

We don't need to implement imputation by hand, and will be using the package `mice`. The equivalent of our analysis here is implemented by specifying the `method` as `'sample'`:

```
imp_mice <- mice(simple_data, m = 30, method = 'sample', print = F)
fit_mice <- with(imp_mice, lm(Y ~ X1 + X2))
(pool_mice <- pool(fit_mice))
```

```
## Class: mipo    m = 30
##          term  m  estimate      ubar           b         t dfcom       df
## 1 (Intercept) 30 0.4085576 0.4339769 0.030391099 0.4653810     3 1.864493
## 2          X1 30 0.5818886 0.1209822 0.004851643 0.1259956     3 1.920219
## 3          X2 30 1.8982188 0.1460214 0.014711458 0.1612232     3 1.810413
##          riv     lambda       fmi
## 1 0.07236361 0.06748048 0.4508789
## 2 0.04143885 0.03979000 0.4301019
## 3 0.10410694 0.09429063 0.4708526
```

Remember that these methods are stochastic, and so we do not expect to achieve equal results between these two implements when the number of iterations is the same. Let's compare our output, however. The estimate $\bar{\theta}$ is given by `estimate`, The components of the uncertainty of this estimate are given as `ubar` and `b`, with the overall variance given by `t`. The proportion of the variance due to missingness is given by `lambda`.

5.8 Imputing missing data: chained equation multiple imputation

In theory, to draw the imputations, we should specify a model, potentially involving some unknown parameters ϕ:

$$f(Y_{\text{mis}}|Y_{\text{obs}}, \phi),$$

specify a prior distribution $\pi(\phi)$, and then sample ϕ from $\pi(\phi)$ and Y_{mis} from $f(Y_{\text{mis}}|Y_{\text{obs}}, \phi)$. This is likely to be difficult, particularly if we have multiple variables with missing information, such that we need to define a joint distribution of the missingness over all variables.

One again, we will make some simplifications.

1. We will propose a missingness process on a per-variable basis, conditional on the remaining variables. For example, if we have three variables X_1, X_2, X_3, we will specify a missing process model of
$$f(X_{\text{mis},1} \mid Y_{\text{obs}}, X_{\text{mis},2}, X_{\text{mis},3}, \phi),$$
and likewise for X_2 and X_3.
2. We will specify missing process models for each variable, with separate parameters *and without considering whether they form a coherent joint distribution for all the missing variables.* (We will need to rely on empirical evidence of whether doing this provides useful/appropriate imputations.)

A method we can use to implement this is called **multiple imputation with chained equations**, which is referred to as the MICE algorithm.

Suppose we have k variables, $Y_1, \dots, Y_k$. The observed data is given by $Y_1^{\text{obs}}, \dots, Y_k^{\text{obs}}$, the missing data is $Y_1^{\text{mis}}, \dots, Y_k^{\text{mis}}$ and 'complete' data will be referred to as $Y_1, \dots, Y_k$. For each variable $j \in \{1, \dots, k\}$ we specify the conditional imputation model

$$f(Y_j^{\text{mis}} \mid Y_j^{\text{obs}}, Y_1, \dots, Y_{j-1}, Y_{j+1}, \dots, Y_k, \phi_j).$$

Note that we have a separate set of parameters ϕ_j for each conditional imputation model. The MICE algorithm imputes a single set of complete data, $Y^{(i)}$. Starting from a random set of values for the missing data, the algorithm implements a number of Gibbs-style iterations to sample missing data from the conditional distributions that have been specified.

Recall that a Gibbs iteration implements the concept of sampling from a joint distribution of k variables by cycling through each of the dimensions in turn and updating them based on their full conditional distribution. Within the t^{th} iteration of Gibbs, if we are handling the j^{th} variable, then the conditional distribution that we would use to sample from would be $f(Z_j \mid Z_1^t, \dots, Z_{j-1}^t, Z_{j+1}^{t-1}, \dots, Z_k^{t-1})$. With MICE, we implement this same process. However, with multiple imputation, we often specify our conditional missing information distributions without considering whether these form a coherent joint distribution, which is why MICE is not necessarily a 'true' Gibbs sampler. Because of this, convergence is not guaranteed to a stationary distribution, but has been shown to produce useful imputations. The MICE algorithm is detailed in the following algorithm:

MICE Algorithm: To generate a single imputed dataset, $Y^{(i)}$.

---

a. Initialise $\dot{Y}_1^0, \dots, \dot{Y}_k^0$ by some simple approach, such as randomly sampling from the observed data $Y_1^{\text{obs}}, \dots, Y_k^{\text{obs}}$.

b. Cycle through $t = 1, \dots, M$ iterations of the following:

   1. Simulate $\phi_1^t \sim f(\phi_1^t \mid Y^{\text{obs}}, \dot{Y}_1^{t-1}, \dots, \dot{Y}_k^{t-1})$, and set
$$\dot{Y}_1^t \sim f(Y_1^{\text{mis}} \mid Y^{\text{obs}}, \dot{Y}_2^{t-1}, \dots, \dot{Y}_k^{t-1}, \phi_1^t).$$

2. Simulate $\phi_2^t \sim f(\phi_2^t \mid Y^{\text{obs}}, \dot{Y}_1^t, \dot{Y}_2^{t-1}, \ldots, \dot{Y}_k^{t-1})$, and set

$$\dot{Y}_2^t \sim f(Y_2^{\text{mis}} \mid Y^{\text{obs}}, \dot{Y}_1^t, \dot{Y}_3^{t-1}, \ldots, \dot{Y}_k^{t-1}, \phi_2^t).$$

3. Simulate $\phi_3^t \sim f(\phi_3^t \mid Y^{\text{obs}}, \dot{Y}_1^t, \ldots, \dot{Y}_3^t, \dot{Y}_4^{t-1}, \ldots, \dot{Y}_k^{t-1})$, and set

$$\dot{Y}_3^t \sim f(Y_3^{\text{mis}} \mid Y^{\text{obs}}, \dot{Y}_1^t, \dot{Y}_2^t, \dot{Y}_4^{t-1}, \ldots, \dot{Y}_k^{t-1}, \phi_3^t).$$

$\vdots$

k. Simulate $\phi_k^t \sim f(\phi_k^t \mid Y^{\text{obs}}, \dot{Y}_1^t, \ldots, \dot{Y}_k^t)$, and set

$$\dot{Y}_k^t \sim f(Y_k^{\text{mis}} \mid Y^{\text{obs}}, \dot{Y}_1^t, \ldots, \dot{Y}_{k-1}^t, \phi_k^t).$$

c. Set $Y_1^{(i)}, \ldots, Y_k^{(i)} = \dot{Y}_1^M, \ldots, \dot{Y}_k^M$.

Note that the MICE algorithm, although it involves M iterations, only produces a single imputed dataset. The intermediate simulated values, the $\dot{Y}$, are discarded. To implement a multiple imputation work-flow, we therefore need to implement the MICE algorithm m times, independently from one another.

In summary, key points to note are

- this process will achieve our objective of obtaining multiple imputed datasets…
- …and will account for uncertainty in any parameters used to specify the conditional missing data distributions…
- …but at a cost that we may not have coherently specified a *joint* distribution for all the missing data, conditional on all the observed data. We need to recognise that our inferences will be approximate.

5.8.1 How many iterations?

MICE seems like a computationally intense process, because we need to implement M iterations, where we cycle through our k variables in each, to obtain a single imputed dataset. We repeat this whole process m times to obtain our multiple imputation set. In practice, the value of M is usually low (somewhere between 5 and 20 iterations). A number of studies have shown that this generally provides informative imputation datasets, because convergence of Gibbs-style samplers is fast when there is noise in the system and correlations are low. Situations we should be aware of in practice are when we have

- Very high correlations between the variables Y_i;
- Missing data rates are high; and
- Constraints on parameters across different variables exist.

5.9 MICE example: the `nhanes` dataset

We previously introduced the `nhanes` dataset in this chapter and explored its missingness pattern. Recall the structure of the data in Table 13, and that we have variables `age` (Y_{age}), `bmi` (Y_{bmi}), `hyp`

(Y_{hyp}) and `chl` (Y_{chl}). In this section we will implement the MICE algorithm on this data. The goal of our analysis is going to be to predict `bmi` from `age` and `chl`, i.e. we want to fit the regression model

$$Y_{\text{bmi},i} \sim N(\beta_0 + \beta_{\text{age}}\, Y_{\text{age},i} + \beta_{\text{chl}} Y_{\text{chl},i} + \beta_{\text{age*chl}} Y_{\text{age},i} Y_{\text{chl},i}, \sigma^2), \tag{36}$$

for $i \in \{1, \dots, 25\}$. Our parameters of interest are therefore $\beta_0, \beta_{\text{age}}, \beta_{\text{chl}}, \beta_{\text{age*chl}}, \sigma^2$. Notice that we aren't interested in the variable `hyp` for our regression model, but we won't remove this variable from our dataset because we are interested in how it can be informative for imputing our missing variables that do feature in the regression.

5.9.1 Complete case analysis

For completeness, let us fit a complete case analysis to this data to compare with out multiple imputation results.

```
nhanes_cc <- lm(bmi ~ chl * age, data = nhanes)
```

This gives the following parameter estimates:

```
##
## Call:
## lm(formula = bmi ~ chl * age, data = nhanes)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8064 -0.9114  0.4048  2.4316  2.8864
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.95690    8.54679   1.633   0.1369
## chl          0.11031    0.04370   2.524   0.0325 *
## age         -1.45829    5.77496  -0.253   0.8063
## chl:age     -0.01783    0.02557  -0.698   0.5031
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.147 on 9 degrees of freedom
##   (12 observations deleted due to missingness)
## Multiple R-squared:  0.6482, Adjusted R-squared:  0.5309
## F-statistic: 5.528 on 3 and 9 DF,  p-value: 0.01982
```

5.9.2 Imputing the missing data

We use `mice` to impute 10 datasets, using regression models for the missing data conditional distributions:

```
nhanes_imp <- mice(nhanes, m = 10, method = 'norm', print = F)
```

The object created by using the `mice` function is a list containing a wealth of information, including:

- The original, incomplete dataset, `data`;
- The imputed datasets, `imp` (easily extracted for use with the function `complete()`);
- The number of imputations, `m`;
- A matrix of the relations used for the imputation model, `predictorMatrix`. If all variables are used to impute all other variables, this will be a matrix of 1's with 0's on the diagonal;
- Information on the mean of the generated missing data over the M iterations that are internal to the MICE algorithm, `chainMean`. This can be used to assess convergence of the Gibbs-style approach and whether M should be increased.

We can extract the imputed data and take a look at it, as shown in Figure 36. This shows the fully observed data pairs between `bmi` and `age`, `bmi` and `chl`, and `chl` and `age` with red squares. Each set of coloured points is then one of the 10 imputed datasets, and a regression line to highlight the general trend of these is given. Note that these regression lines plotted are not the regression lines used in the imputation algorithm, but are only the regression of the resulting imputed dataset. We have shown `bmi` against both `age` and `chl` as these are to be the elements of our analysis regression. This exploratory plot suggests a relationship between these variables, that remains similar throughout the range of imputed datasets.

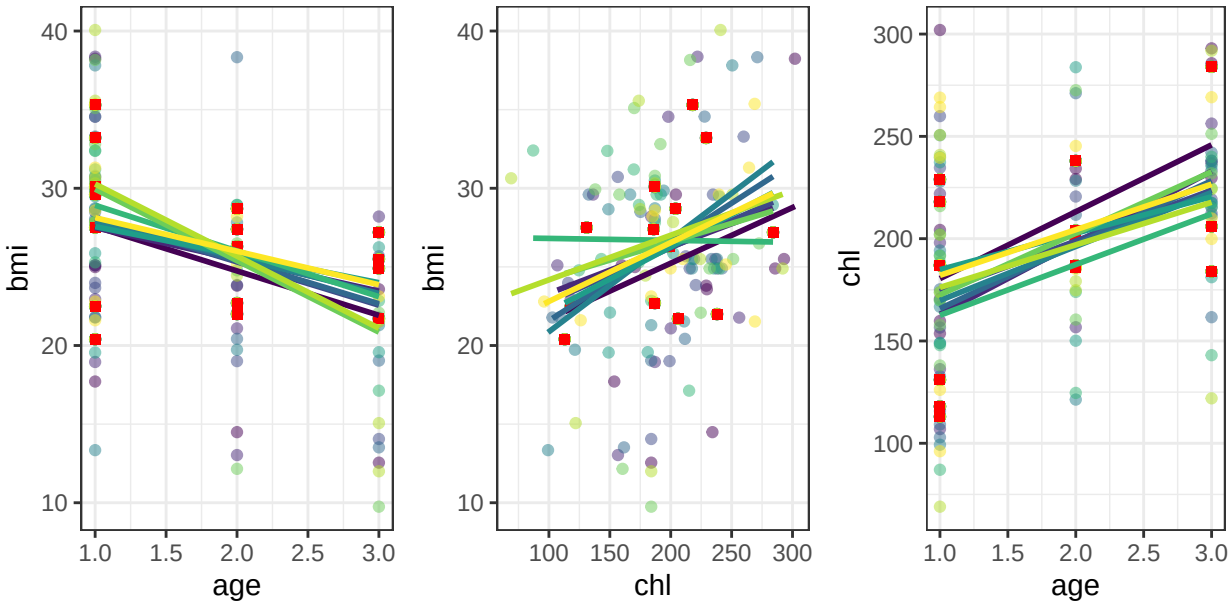


Figure 36: The imputed data using MICE for the nhanes dataset. For the three pairs of variables shown below, fully observed data are highlighted with red squares. Each of the 10 imputed datasets are shown by the varying colour scale of points. A regression line of each imputed set is shown—note that this is not related to the imputation regression model, but merely used to highlight the differences between the imputed datasets.

We can study the convergence of our MICE algorithm by using `plot()` on the list produced by the `mice()` function. This plot, in Figure 37 has a row for each imputed variable (recall that `age`

is fully observed in our dataset so we have 3 variables being imputed), and a column each for the mean and sd. Each plot contains m lines, one for each imputation.

The points on the x-axis are the iterations within the MICE algorithm; we referred to a total of M iterations earlier. The default value in MICE is $M = 5$, as shown in the top panel here. The points on the y-axis are the means or standard deviations of the imputed values of the missing variables. For example, for there were 9 instances of missing `bmi` values, so the plot is showing the mean and standard deviation of 9 imputed `bmi` values at each for the 5 iterations within the MICE algorithm for each of the 10 imputed data sets.

In general here we are looking for the imputation streams to intermingle and be free of any trends in the later iterations of the MICE algorithm to suggest convergence.

Let's increase the number of iterations of the MICE algorithm, to confirm that there is no trend and that convergence was acceptable. A great feature of `mice` is that we don't need to start from scratch, we can continue the number of iterations from our original 5. In the below, we add another 35 to obtain 40 iterations, shown in the bottom panel of Figure 37:

```
nhanes_imp_long <- mice.mids(nhanes_imp, maxit = 35, print = F)
```

We plotted our imputed datasets in Figure 36 to explore the relationships between the variables. A further common exploratory plot in multiple imputation would be to check that the imputed values are *plausible* and could reasonably be believed to have been values of possible observations. We can explore this with the `stripplot` function, as shown in Figure 38. In such a plot, we're hoping to see that the imputed data in red is not so extreme or unbelievable that we would not believe it could have been an observed data point, and also whether there are trends being followed.

In Figure 38 we see some general bunching of the imputations in areas where the observations have, such as around 190 for `chl`, which is promising. We also see that there are imputed values more extreme than we observed— we would want to be careful to question whether these extremes are plausible.

5.9.3 Imputation analysis

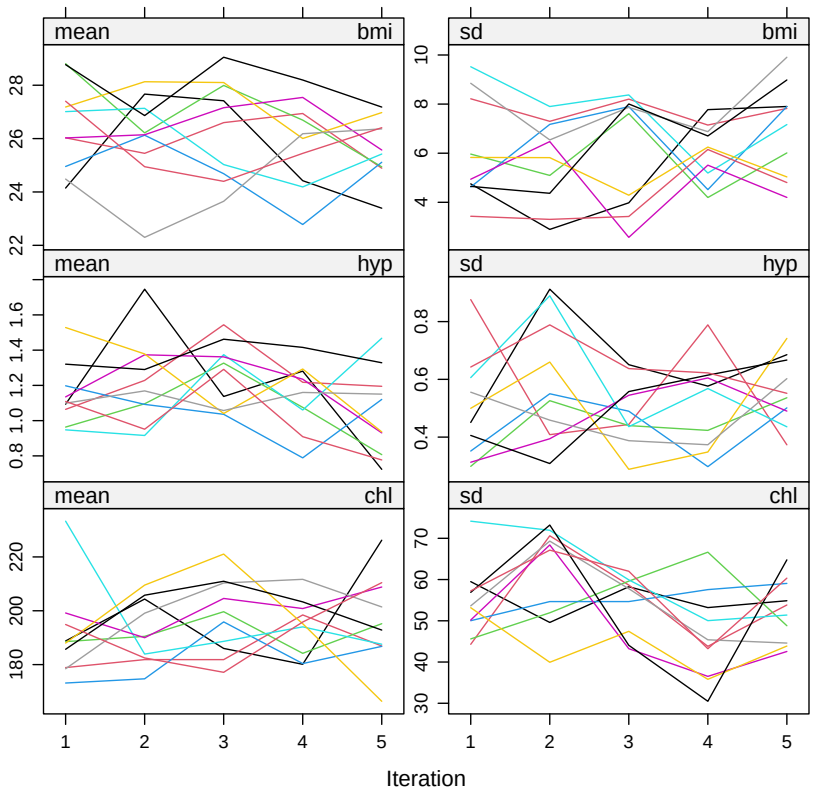
We now analyse each of each of our imputed datasets. Recall that our aim was to estimate the parameters $\beta_0, \beta_{\text{age}}, \beta_{\text{chl}}, \beta_{\text{age*chl}}, \sigma^2$ for the regression model in Equation (36). We use the function `with()` for this task:

```
nhanes_fit <- with(nhanes_imp, lm(bmi ~ chl * age))
```

We produce an object of type `mira`, *multiply imputed repeated analyses*, which contains the estimated regression parameters for each of the 10 imputed datasets we created earlier. For instance, the summary of the model fitted to the first imputed dataset is:

```
summary(nhanes_fit$analyses[[1]])
```

```
##
## Call:
## lm(formula = bmi ~ chl * age)
##
```



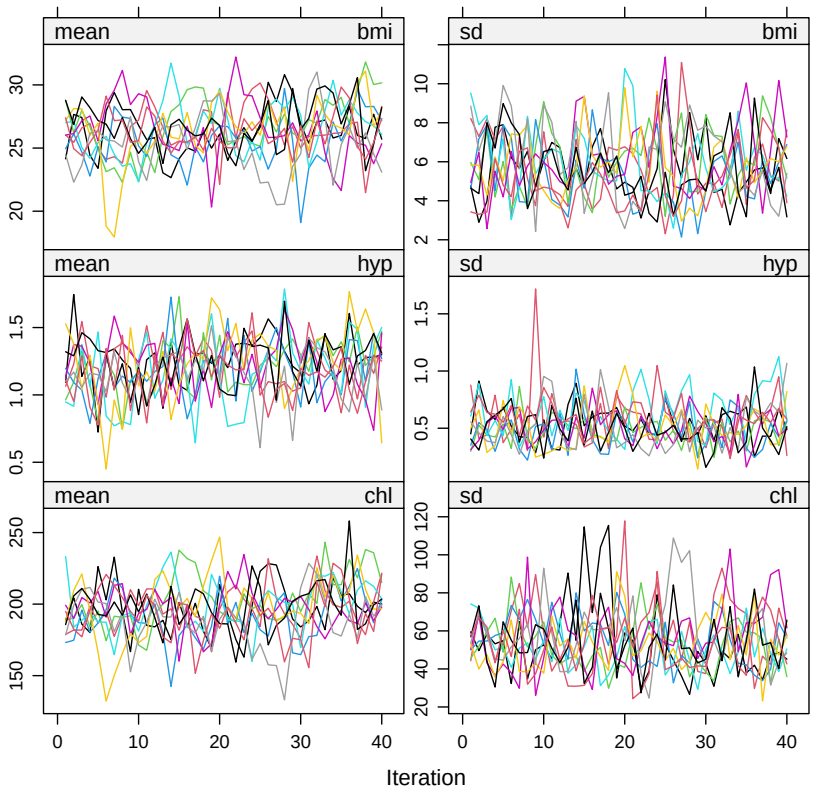


Figure 37: The mean and standard deviation of the three missing variables over the internal iterations of the MICE algorithm. The top panel shows the default 5 iterations for the 10 imputation datasets, and the bottom panel shows this after having been extended to 40 iterations.

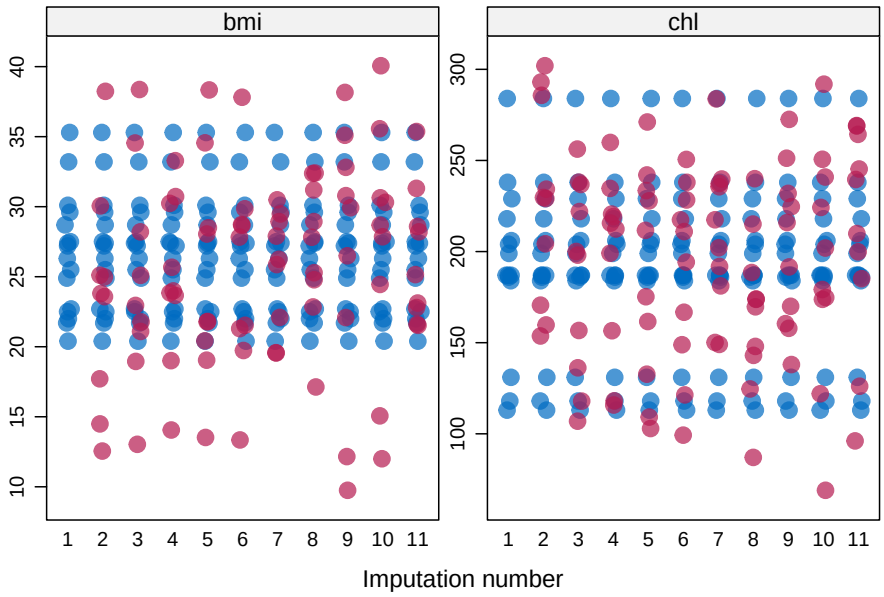


Figure 38: Strip plots of the imputed variables bmi and chl. The observed data is shown in blue, and imputed data in red. Imputation 1 is shown as the original, incomplete data, and the 10 imputations are shown proceeding this.

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.0483  -1.4156   0.2602   2.6360   4.7962
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.019780   8.040370   2.117   0.0464 *
## chl          0.088147   0.039157   2.251   0.0352 *
## age         -4.792027   5.011114  -0.956   0.3498
## chl:age     -0.003299   0.021579  -0.153   0.8799
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.251 on 21 degrees of freedom
## Multiple R-squared:  0.5393, Adjusted R-squared:  0.4735
## F-statistic: 8.195 on 3 and 21 DF,  p-value: 0.0008412
```

5.9.4 Analysis pooling

Finally, we pool our analyses together in step (3) of our multiple imputation work-flow with the function `pool()`:

```
(nhanes_pool <- pool(nhanes_fit))
```

```
## Class: mipo    m = 10
##          term  m     estimate         ubar            b            t dfcom
## 1 (Intercept) 10 18.644069364 5.256869e+01 5.114892e+01 1.088325e+02    21
## 2         chl 10  0.083297046 1.410073e-03 8.881682e-04 2.387058e-03    21
## 3         age 10 -3.631627438 2.066410e+01 2.845666e+01 5.196643e+01    21
## 4     chl:age 10 -0.006087487 4.483626e-04 3.977539e-04 8.858919e-04    21
##         df       riv    lambda       fmi
## 1 7.286312 1.0702913 0.5169762 0.6108920
## 2 9.384960 0.6928614 0.4092842 0.5046766
## 3 5.849503 1.5148169 0.6023567 0.6922247
## 4 7.707515 0.9758382 0.4938857 0.5884201
```

We produce an object of type `mipo`, *multiply imputed pooled object*. We can see the pooled estimates of our parameters of interest, which we have referred to as $\bar{\theta}$, via the `estimate` above. The components of the variance of this are given, with the total being `t`, and the proportion of the variance that is due to the missing data is given by `lambda`. Note that although `age` is completely observed in the data, this does not mean that this translates through into estimating the coefficient β_{age} without missingness error, because the `bmi` and `chl` were not fully observed.

We can explore the pooled estimate also with `summary()`:

```
summary(nhanes_pool)
```

```
##          term     estimate   std.error  statistic       df   p.value
## 1 (Intercept) 18.644069364 10.43228159  1.7871517 7.286312 0.1153885
## 2         chl  0.083297046  0.04885753  1.7048969 9.384960 0.1210228
## 3         age -3.631627438  7.20877448 -0.5037788 5.849503 0.6328217
## 4     chl:age -0.006087487  0.02976394 -0.2045256 7.707515 0.8432399
```

This suggests from the pooled p-value estimates that the `age` and `chl*age` terms are not required in our model.

Part III

Resampling methods

6 Bootstrapping

Aims of this chapter
1. Introduce bootstrapping: a Monte Carlo method for obtaining standard errors and confidence intervals, and performing hypothesis tests. 2. Understand sampling from the empirical distribution function, or equivalently, resampling observations with replacement, as the key step in the bootstrapping method. 3. Explain the difference between nonparametric and parametric bootstrapping, and show how the latter can be used for hypothesis testing.

6.1 Bootstrap estimates of standard errors

We will first consider the problem of obtaining standard errors and confidence intervals for parameter estimates.

Suppose we want to estimate a parameter θ in some statistical model given a sample of data $\mathbf{x} = \{x_1, \dots, x_n\}$. We will use an estimate

$$\hat{\theta}(\mathbf{x}),$$

where $\hat{\theta}(.)$ is some function of the data $\mathbf{x}$. The standard error of the estimator, denoted by $s.e.(\hat{\theta})$ is defined as

$$s.e.(\hat{\theta}) = \sqrt{Var\{\hat{\theta}(\mathbf{X})\}}$$

for a random sample of data $\mathbf{X} = \{X_1, \dots, X_n\}$. In other words, before we get the data, we think of $\hat{\theta}(\mathbf{X})$ as a random variable, and the standard error is the standard deviation of this random variable. For simple problems, we can derive formulae for standard errors.

For example, if we have

$$X_1, \dots, X_n \overset{i.i.d}{\sim} N(\theta, \sigma^2),$$

and our estimator for θ is $\hat{\theta}(\mathbf{X}) = \bar{X}$, then we can show that

$$s.e.(\hat{\theta}) = \frac{\sigma}{\sqrt{n}}.$$

What can we do when we have more complicated models, and we can't derive formulae such as that above?

Suppose we knew the distribution F of the data. We could then use a Monte Carlo procedure to estimate a standard error:

1. Sample N independent data sets $\mathbf{x}_1^*, \dots, \mathbf{x}_N^* \sim F(\cdot)$, each of size n;
2. For each data set i, calculate the corresponding estimate $\hat{\theta}(\mathbf{x}_i^*)$ of the parameter θ;
3. Estimate the standard error $s.e.(\hat{\theta})$ by

$$\widehat{s.e.}(\hat{\theta}) = \left[\frac{1}{N-1} \sum_{i=1}^{N} \left\{ \hat{\theta}(\mathbf{x}_i^*) - \bar{\hat{\theta}} \right\}^2 \right]^{0.5},$$

with $\bar{\hat{\theta}} = \sum_{i=1}^{N} \hat{\theta}(\mathbf{x}_i^*)/N$.

This is the bootstrap estimate of the standard error. As long as we can sample datasets in step (1), and then actually calculate a parameter estimate given a data set, it doesn't matter how complicated the model is. This makes the bootstrapping method widely applicable.

We don't actually know what F is (if we did, we would already know all the corresponding parameters), but if we could estimate it, we could do an approximation of the Monte Carlo procedure above, where we instead sample random data sets from our estimate of F. This may seem like a somewhat circular argument: normally it's the parameter estimates that give us our estimate of the distribution F, but in bootstrapping, we can estimate F without reference to any particular model; we can estimate F *without* having to first estimate any parameters.

6.2 Estimating a distribution using the empirical cumulative distribution function

In (*nonparametric*) bootstrapping we estimate the distribution F given observations $\{x_1, \dots, x_n\}$ by the **empirical cumulative distribution function** (ECDF). We denote the ECDF by F_n and define it as

$$F_n(x) := \frac{1}{n} \sum_{i=1}^{n} I(x_i \leq x),$$

where $I(\cdot)$ is the indicator function.

We illustrate this for $X_1, \dots, X_5 \sim N(0,1)$. Let our observations be

$$(x_1, \dots, x_5) = (-0.897, 0.185, 1.59, -1.13, -0.0803),$$

then the ECDF is given by that in Figure 39 along with the true distribution function. We can see that this is not a particularly good estimate of the true cumulative distribution function, and so for small sample sizes, we may not expect bootstrapping to work well.

However, the estimate will be better for larger sample sizes, e.g. we repeat this process for observations $x_1, \dots, x_{50}$ of the i.i.d. random variables $X_i \sim N(0,1)$, for $i = 1, \dots, 50$ and show this in Figure 40.

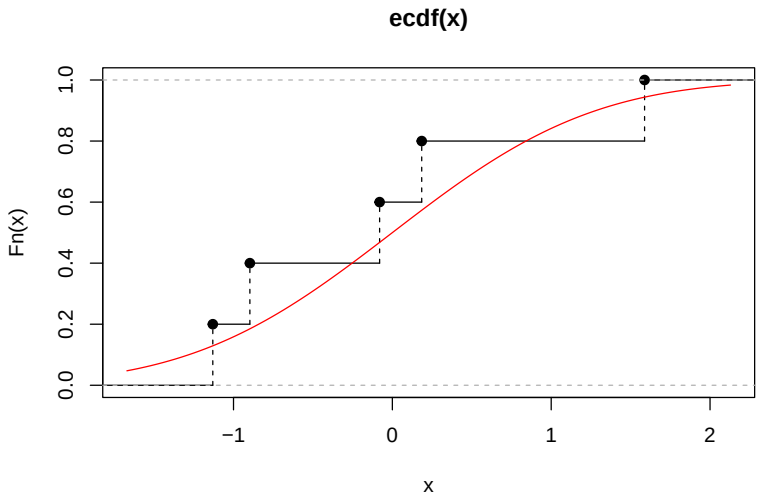


Figure 39: The ECDF for five observations drawn independently from a standard normal (black). The true distribution function for the standard normal is shown in red.

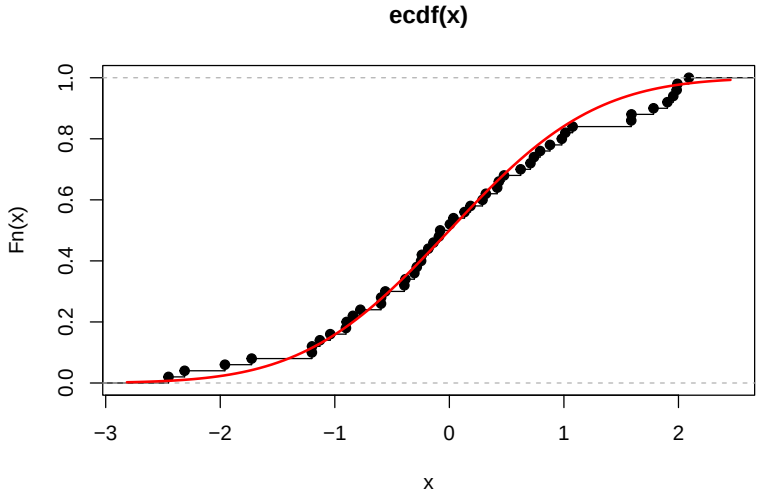


Figure 40: The ECDF for 50 observations drawn independently from a standard normal (black). The true distribution function for the standard normal is shown in red.

6.2.1 Sampling from an ECDF

To produce a bootstrap sample of data, we need to generate a sample of size n (the same size as the original data set) from the ECDF. Sampling from an ECDF is equivalent to sampling randomly from $\{x_1, \dots, x_n\}$ with replacement (i.e. where each observation is equally likely to be chosen each time). We therefore don't actually need to explicitly specify the ECDF! We will demonstrate this shortly, but there are two ways in which we can see that this is valid.

Firstly, if a random variable X has probability distribution specified by the empirical distribution function for the sample $\{x_1, \dots, x_n\}$, then for $i = 1, \dots, n$ we have

$$\begin{aligned} P(X = x) &= P(X \leq x) - P(X < x) && (37)\\ &= \frac{1}{n}\sum_{i=1}^{n} I(x_i \leq x) - \frac{1}{n}\sum_{i=1}^{n} I(x_i < x) && (38)\\ &= \frac{1}{n}\sum_{i=1}^{n} \left(I(x_i \leq x) - I(x_i < x)\right) && (39)\\ &= \begin{cases} \frac{1}{n} & \text{for } x \in \{x_1, \dots, x_n\};\\ 0 & \text{otherwise.} \end{cases} && (40) \end{aligned}$$

as $I(x_i \leq x) - I(x_i < x) = 1$ if $x = x_i$ and 0 otherwise.

Secondly, the process can be understood via the *inversion* method of sampling. Let X be any random variable with distribution function (CDF) $F_X(x)$. If we transform X using its own CDF, i.e. if we define a new random variable Y as $Y = F_X(X)$, then it can be shown that $Y \sim U[0, 1]$. Hence if we can sample Y from the $U[0, 1]$ distribution, we can obtain a random draw from the distribution of X by setting $X = F_X^{-1}(Y)$.

Now, if $F_X(x)$ is an ECDF, the function consists of n 'steps' of height all equal $1/n$, and so when we apply the inversion of a uniform random variable, any point in $\{x_1, \dots, x_n\}$ has the same probability of being selected. We see this process in Figure 41, where we show the ECDF of five standard uniform samples. If we draw the value u from a uniform distribution, we apply the inverse function $F_X^{-1}(u) = x_3$ and so we set x_3 as our random draw.

To give an indication that the process of sampling with replacement does indeed work, we illustrate this in Figure 42. We sample with replacement from a sample of standard normals 1000 times to obtain the histogram shown in the figure. We can see it is a fair approximation to the true density of the standard normal, which is shown in red.

6.3 Notation summary

We need to be careful with notation here, as there are various different types of random variables and samples to keep track of. In particular, watch for the 'big X random, little x observed' distinction, and the use of the $*$ superscript to indicate the use of bootstrapping. To summarise, we have:

- θ: a parameter in a model we wish to estimate;
- $\mathbf{X} = \{X_1, \dots, X_n\}$: independent and identically distributed random variables, with a probability distribution dependent on θ;

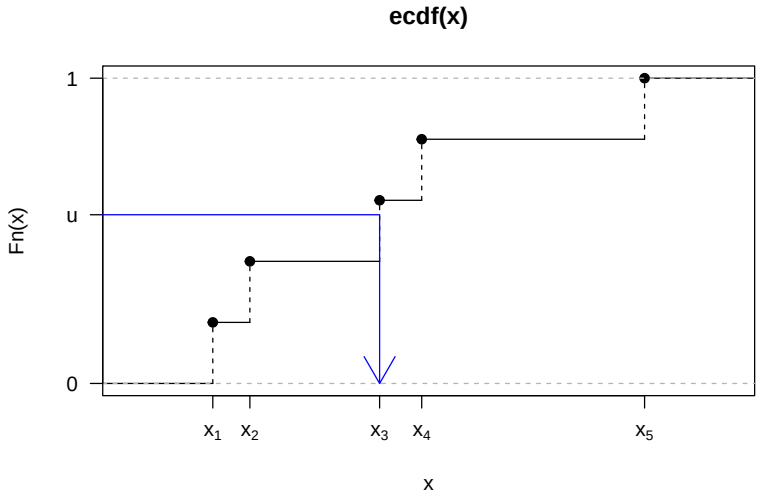


Figure 41: The ECDF for five observations drawn independently from a standard normal (black). We sample a value u from a uniform distribution and apply an inverse transformation $F_X^{-1}(u)$ to yield the corresponding value of x, which here is given by x_3.

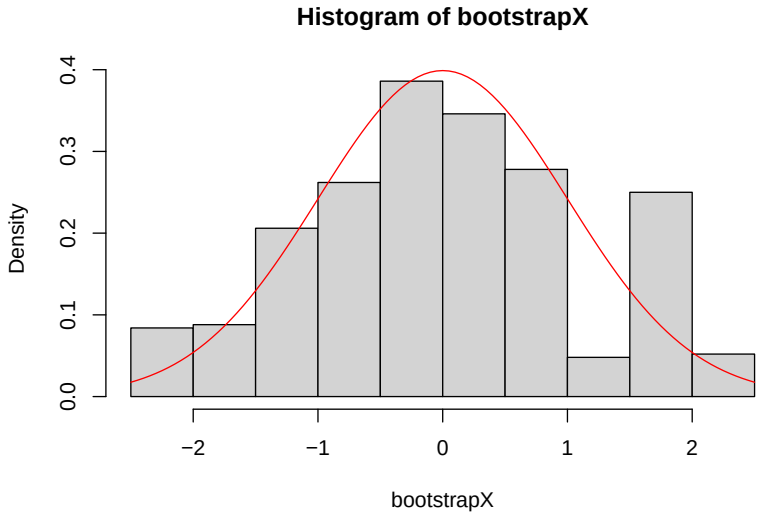


Figure 42: Given a sample of 50 independent draws from a standard normal, we sample with replacement 1000 times. This sample is represented by the histogram, and the true density function of the standard normal is shown in red.

- $\mathbf{x} = \{x_1, \ldots, x_n\}$: our observed sample of data;
- $\mathbf{X}^* = \{X_1^*, \ldots, X_n^*\}$: n 'bootstrapped' random variables, defined by resampling from $\{x_1, \ldots, x_n\}$ independently and with replacement;
- $\mathbf{x}^* = \{x_1^*, \ldots, x_n^*\}$: an observed bootstrapped sample
- $\hat{\theta}(.)$: a function of the data used to estimate θ;
- $\hat{\theta}(\mathbf{X})$: an **estimator**. A random variable, defined as a transformation of the random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$;
- $\hat{\theta}(\mathbf{x})$: the **estimate** we have obtained given the observed data; and
- $\hat{\theta}(\mathbf{x}^*)$: a bootstrapped estimate of θ obtained from a bootstrapped sample of data.

6.4 Example: Bootstrap standard errors of a sample mean and sample variance

We'll first try bootstrapping in a simple example where we actually know what the true standard errors are, so we can check the method works. We'll consider normally distributed data

$$X_1, \ldots, X_{30} \overset{i.i.d}{\sim} N(\mu, \sigma^2).$$

We can use the sample mean $\bar{X}$ and sample variance S^2 as estimators of μ and σ^2, and these have standard errors $\frac{\sigma}{\sqrt{30}}$ and $\sqrt{\frac{2\sigma^4}{(30-1)}}$ respectively. Let's see if we can recover these known standard errors using a bootstrapping estimate.

We'll first choose $\mu = 0$ and $\sigma = 1$, and generate a sample of data:

```
x <- rnorm(30)
```

Now, to obtain a bootstrap sample of data, we just sample the vector `x` 30 times with replacement, e.g.

```
bootSample <- sample(x, size = 30, replace = TRUE)
```

and then we would calculate the sample mean and sample variance of `bootSample`. We repeat this a large number of times to get our sample of estimators:

```
N <- 10000
bootMeans <- bootVars <- rep(NA, N)
for(i in 1:N){
  bootSample <- sample(x, size = 30, replace = TRUE)
  bootMeans[i] <- mean(bootSample)
  bootVars[i] <- var(bootSample)
}
```

Now let's compare the sample standard deviations of our bootstrapped sample of estimators (first value in the pairs calculated below) with the true standard errors that we know from distributional theory (second value in the pairs below). For the sample mean:

```
c(sd(bootMeans), 1/sqrt(30))
```

```
## [1] 0.1667101 0.1825742
```

```
c(sd(bootVars), sqrt(2/29))
```

```
## [1] 0.2298253 0.2626129
```

In both cases, the bootstrap standard errors are fairly close to the true standard errors.

6.5 Confidence intervals

6.5.1 Confidence intervals using the estimated standard error

If we assume the estimator $\hat{\theta}$ is normally distributed and unbiased, then once we have the bootstrap estimate of the standard error, $\widehat{s.e.}(\hat{\theta})$, we can report

$$\hat{\theta} \pm 2\widehat{s.e.}(\hat{\theta}),$$

as an approximate 95% confidence interval.

Continuing our example from the previous section, we can compare bootstrap confidence intervals with the exact confidence interval because we can derive that here. We know the distributions of the sample mean

$$\bar{X} \sim t_{n-1}\left(\mu, \frac{S}{\sqrt{n}}\right),$$

and variance

$$S^2 \sim \frac{\sigma^2}{n-1}\chi^2_{n-1}.$$

Starting with the mean, the 95% confidence interval for μ using the bootstrap approach would be:

```
c(mean(x) - 2*sd(bootMeans), mean(x) + 2*sd(bootMeans))
```

```
## [1] -0.2509621  0.4158785
```

and the exact interval would be calculated as $\bar{x} \pm t_{19;0.025}\sqrt{(s^2/20)}$:

```
mean(x) + c(-1,1) * qt(0.975,29) * sd(x) / sqrt(30)
```

```
## [1] -0.2626142  0.4275306
```

Actually, we can get this interval a little easier in R by fitting a linear model with an intercept only:

```
confint(lm(x~1))
```

```
##                  2.5 %    97.5 %
## (Intercept) -0.2626142 0.4275306
```

Similarly, for the variance parameter, the 95% confidence interval for σ^2 using the bootstrapping approach would be

```
c(var(x) - 2*sd(bootVars), var(x) + 2*sd(bootVars))
```

```
## [1] 0.3943486 1.3136500
```

which we can compare with the exact confidence interval $(29s^2/\chi^2_{29;0.025}, 29s^2/\chi^2_{29;0.975})$:

```
(29)*var(x) / qchisq(c(0.975, 0.025), 29)
```

```
## [1] 0.541661 1.543333
```

The bootstrap interval for the mean is fairly close to the exact interval. The bootstrap interval for the variance is slightly off (though in the right ballpark); we might expect this, as we know that the sample variance S^2 isn't normally distributed.

6.5.2 Confidence intervals using percentiles

If we think the distribution of the estimator deviates substantially from a normal distribution, we could derive an interval based on percentiles instead. Suppose we could find l and u such that

$$\mathbb{P}(\hat{\theta} > \theta + u) = \mathbb{P}(\hat{\theta} < \theta - l) = 0.025,$$

then we have

$$\mathbb{P}(\hat{\theta} - u < \theta < \hat{\theta} + l) = 0.95.$$

A 95% confidence interval for θ would be

$$(\hat{\theta} - u, \quad \hat{\theta} + l).$$

We can estimate l and u from our bootstrap sample: we estimate these as the 2.5th and 97.5th percentiles from our sample of $\hat{\theta}(\mathbf{x}^*_i) - \hat{\theta}(\mathbf{x})$ (for $i = 1, \ldots, n$).

The assumptions here are that:

1. The distribution of $\hat{\theta}(\mathbf{X}) - \theta$ would be the same for any value of θ, including $\theta = \hat{\theta}(\mathbf{x})$; and
2. The distribution of a randomly generated estimator $\hat{\theta}(\mathbf{X}^*)$ obtained via bootstrapping is (approximately) the same as the distribution of $\hat{\theta}(\mathbf{X})$.

In our example, we obtain the confidence interval for σ^2 using

```
var(x) - quantile(bootVars - var(x), c(0.975, 0.025))
```

```
##     97.5%      2.5%
## 0.4068962 1.2967503
```

In this case, this is not noticeably different from the standard error-based interval. If we plot a histogram of bootstrap estimates, as in Figure 43 although the distribution is skewed, the 2.5th and 97.5th percentiles are similar distances from $\hat{\theta}(\mathbf{x})$.

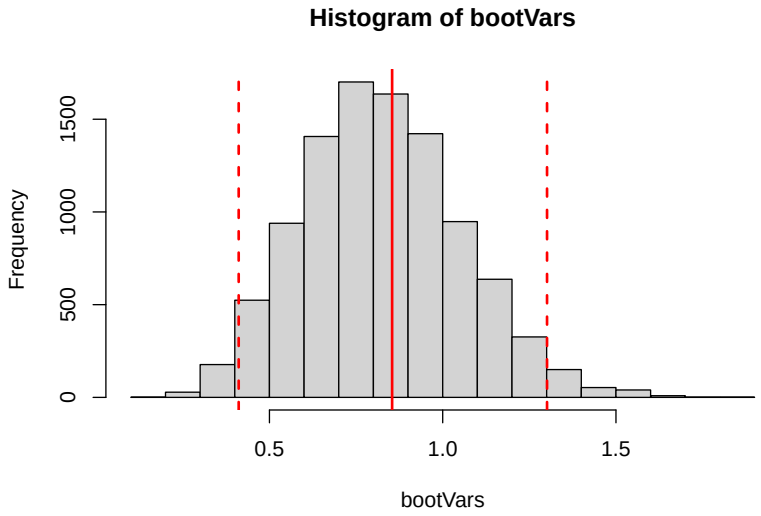


Figure 43: A histogram of bootstrapped estimates of the variance. The red solid line shows the sample variance of the observed data, and the dashed lines indicate 2.5th and 97.5th percentiles from the bootstrapped sample of estimates. Although the distribution of bootstrapped estimates is skewed, these percentiles are roughly the same distance from the sample variance.

6.6 Properties of samples from the empirical CDF

6.6.1 Expectation and variance

Following on from the discussion above, it is helpful to understand a little more about what the implications are when we sample from an ECDF. Denote the sample by $x_1, \ldots, x_n$, and let X be a random variable sampled from the corresponding ECDF. Then we have

$$E(X) = \sum_{i=1}^{n} x_i \mathbb{P}(X = x_i) \tag{41}$$

$$= \sum_{i=1}^{n} x_i \frac{1}{n} \tag{42}$$

$$= \bar{x}, \tag{43}$$

and similarly

$$E(X^2) = \sum_{i=1}^{n} x_i^2 \frac{1}{n},$$

so that

$$\mathrm{Var}(X) = \frac{n-1}{n} s^2,$$

where s^2 is the sample variance.

Hence for a bootstrapped sample mean, we would have

$$\mathrm{Var}(\bar{X}^*) = \frac{n-1}{n^2} s^2 \simeq \frac{s^2}{n}.$$

So, for a large enough N, the bootstrap estimate of the standard error of a sample mean would just be (approximately) the usual estimate s^2/n. Therefore, bootstrapping doesn't add anything to the story here! But, again, the motivation for bootstrapping comes from problems where we could not easily derive standard errors directly rather than the problem of estimating $\bar{X}$.

6.6.2 Sample percentiles

Suppose we wanted to estimate the standard error of a sample percentile, e.g., the 95th percentile. For a small n, we wouldn't expect the ECDF to approximate the tails of a distribution particularly well.

Consider, for example, $n = 21$, and denote the ordered sample by $x_{(1)}, \dots, x_{(21)}$, so that the sample estimate of the 95th percentile (obtained via default algorithm in the `quantile()` function in R) would be $x_{(20)}$. In any bootstrap sample, we would frequently obtain $x_{(20)}$ as the bootstrapped 95th percentile, and we couldn't get a value larger than $x_{(21)}$. Therefore, we would likely underestimate the standard error of the sampled 95th percentile (regardless of how many bootstrap samples we take).

We illustrate this as follows. Suppose we have

$$X_1, \dots, X_{21} \overset{i.i.d}{\sim} N(\mu, \sigma^2),$$

and given the observations $x_1, \dots, x_{21}$, we want to give an estimated standard error for the sample 95th percentile. We'll estimate this with bootstrapping, and in parallel, we'll also use Monte Carlo to estimate the standard error, using multiple samples of size 21 from the correct distribution.

Here is our approach to estimate the 95th percentile using Bootstrapping and Monte Carlo:

```
set.seed(1)
x <- sort(rnorm(21)) # obs
N <- 10000 # number of bootstrap samples
boot95th <- sampled95th <- rep(NA, N)

for(i in 1:N){

  # make a bootstrap sample of observations
  bootSample <- sample(x, 21, replace = TRUE)
  # calculate the bootstrap estimator
  boot95th[i] <- quantile(bootSample, 0.95)

  # sample from underlying distribution
  z <- rnorm(21)
  # calculate the estimator
  sampled95th[i] <- quantile(z, 0.95)
}
```

We compare the standard errors of our samples of the estimator:

```
sd(boot95th)
```

```
## [1] 0.2629693
```

```
sd(sampled95th)
```

```
## [1] 0.3981949
```

We see that the bootstrap estimate of the standard error is noticeably smaller than that obtained via Monte Carlo. We can see the problem clearer when we compare histograms of the percentile estimates, as in Figure 44. We see that there is a limited range of values that can be obtained for the percentile estimate in bootstrapping, and that the standard error of this estimate is an under-approximation.

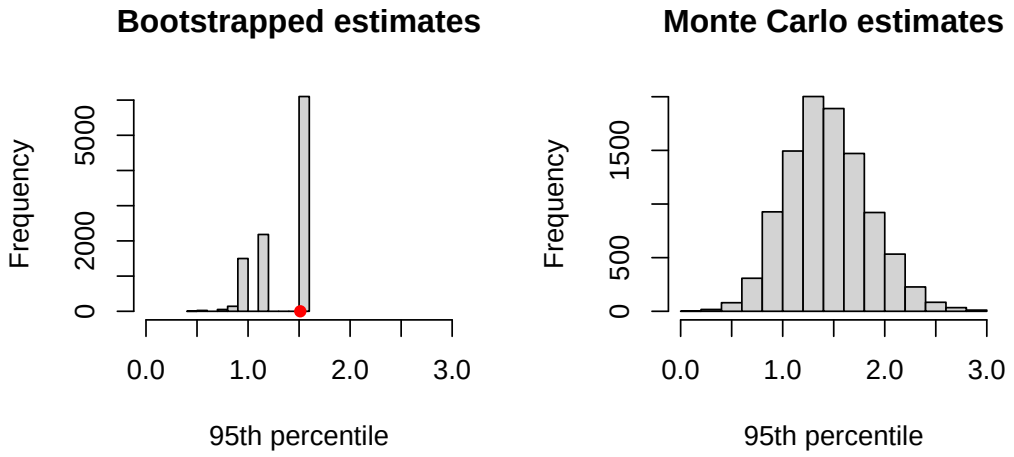


Figure 44: Comparison of bootstrapping and Monte Carlo when estimating the 95th percentile from a set of 21 observations. Both methods implemented 10000 samples.

6.6.3 Sources of error and sample sizes in bootstrapping

There are two sources of error to consider in bootstrapping, related to two sample sizes:

1. Error in approximating the distribution of the data by the empirical distribution function, based on n observations;
2. Monte Carlo error in estimating a standard error (or some other summary), based on a random sample of N bootstrapped data sets.

We can usually make N as large as we want, so that any Monte Carlo error will be negligibly small, but this won't have any effect on the first source of error. So if the original sample size n is too small, then by making N large, we may simply be obtaining a precise Monte Carlo estimate of a poor approximation.

Table 16: Extract of the first 6 rows of the diagnosis dataset, only considering the first two raters.

rater1	rater2
4. Neurosis	4. Neurosis
2. Personality Disorder	2. Personality Disorder
2. Personality Disorder	3. Schizophrenia
5. Other	5. Other
2. Personality Disorder	2. Personality Disorder
1. Depression	1. Depression

6.7 Example: Measuring observer agreement

So far, we've only illustrated bootstrapping in an example where we don't really need to use it; standard errors and confidence intervals could be obtained analytically. We'll now consider an example where it's not so straightforward to derive a confidence interval (although an asymptotic approximation is available).

6.7.1 The data

We consider the `diagnoses` data set in the `irr` package (Gamer et al., 2019). We have psychiatric diagnoses for 30 patients. Each patient has been diagnosed separately by 6 different 'raters', and the interest is in the extent to which the raters agree with each other. There are five possible diagnoses: depression; personality disorder, schizophrenia, neurosis, "other".

We'll just consider diagnoses made by the first two raters, an extract of this data is shown in Table 16. These two raters agreed on five out of their first six diagnoses.

6.7.2 The kappa statistic

One way to measure agreement between raters is to use Cohen's kappa statistic, defined as

$$\kappa := \frac{p_o - p_e}{1 - p_e},$$

where p_o is the observed proportion of times two raters agreed, and p_e is the estimated probability the raters would agree purely by chance (i.e. the diagnosis given by rater 1 is probabilistically independent of the diagnosis given by rater 2). In our example, we would obtain p_e as

$$p_e = \sum_{j=1}^{5} p_{1,j} p_{2,j},$$

where $p_{i,j}$ is the observed proportion of times rater i gave diagnosis j.

A kappa of 1 would correspond to perfect agreement, and a kappa of 0 would correspond to agreement no better than chance. Note that kappa can be negative if the agreement is worse than that expected by chance.

In R, we can compute kappa using the function `irr::kappa2()`:

```
##  Cohen's Kappa for 2 Raters (Weights: unweighted)
##
##  Subjects = 30
##    Raters = 2
##     Kappa = 0.651
##
##         z = 7
##   p-value = 2.63e-12
```

We obtained $\kappa = 0.651$ in this example: interpreting the actual value can be hard, but comparing relative values may be helpful for comparing how well different pairs of raters agree with each other. And since we calculated kappa using 30 observations only, it may be helpful to obtain a confidence interval, to assess uncertainty in this value.

6.7.3 Bootstrapping bivariate data

For the two raters, denote the data by

$$(r_{1,1}, r_{2,1}) \tag{44}$$
$$\vdots \tag{45}$$
$$(r_{1,30}, r_{2,30}), \tag{46}$$

where $r_{i,j}$ is the diagnosis given by rater i for observation j. To get a bootstrap sample, we resample *pairs* of diagnoses, with replacement, which we can do by sampling from the indices 1-30 of our data:

```
indices <- 1:30
(bootIndex <- sample(indices, size = 30, replace = TRUE))
```

```
##  [1] 25  4  7  1  2 29 23 11 14 18 27 19  1 21 21 10 22 14 10  7  9 15 21  5  9
## [26] 25 14  5  5  2
```

so the first element of our bootstrap sample is $(r_{1,25}, r_{2,25})$, the second element is $(r_{1,4}, r_{2,4})$ and so on. We then recompute kappa for each bootstrap sample. For example:

```
N <- 1000
bootKappa <- rep(0, N)
for(i in 1:N){
  bootIndex <- sample(indices,
                      size = 30,
                      replace = TRUE)
  bootData <- diag2[bootIndex, ]
  bootKappa[i] <- irr::kappa2(bootData)$value
}
```

We can now extract sample quantiles from our bootstrapped sample of kappa values, to get our approximate confidence interval:

Table 17: Extract of the milk dataset.

protein	Time	Cow	Diet
3.63	1	B01	barley
3.57	2	B01	barley
3.47	3	B01	barley
3.65	4	B01	barley
3.89	5	B01	barley
3.73	6	B01	barley

```
quantile(bootKappa, probs = c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.4451596 0.8516323
```

Although we have only considered the first two raters here, to give these numbers a little more context, we show the range of observed kappa statistics between rater 1 and each of the other four raters: 0.384, 0.258, 0.188, and 0.0809, so these range from 0.08-0.38. We may therefore conclude that there is evidence of agreement between the first two raters.

6.8 Parametric bootstrapping and hypothesis testing

So far in this chapter we have considered nonparametric bootstrapping.

> In parametric bootstrapping, we first fit a parametric distribution to our data, and then sample new data sets from this distribution, rather than the nonparametric approach of resampling our observed data. This can be useful in hypothesis testing problems in which it is difficult to work out the distribution of the test statistic under the null hypothesis; we can use Monte Carlo instead.

We illustrate this with an example in mixed effects modelling.

6.8.1 The `Milk` data set

Our example will use the `Milk` data from the `nlme` package (Pinheiro et al., 2021), an extract of which is given in Table 17.

This data features the following variables:

- `protein` is the dependent variable: the protein content of in a sample of cow milk;
- `Cow` labels the cow from which the sample was taken;
- `Time` is the number of weeks since calving for the corresponding `Cow`; and
- `Diet` is one of three possible diets the cow was on (`barley`, `lupins`, `barley+lupins`).

Multiple samples are taken from each cow, and we might expect correlation between observations taken on the same cow. The plot in Figure 45 confirms this.

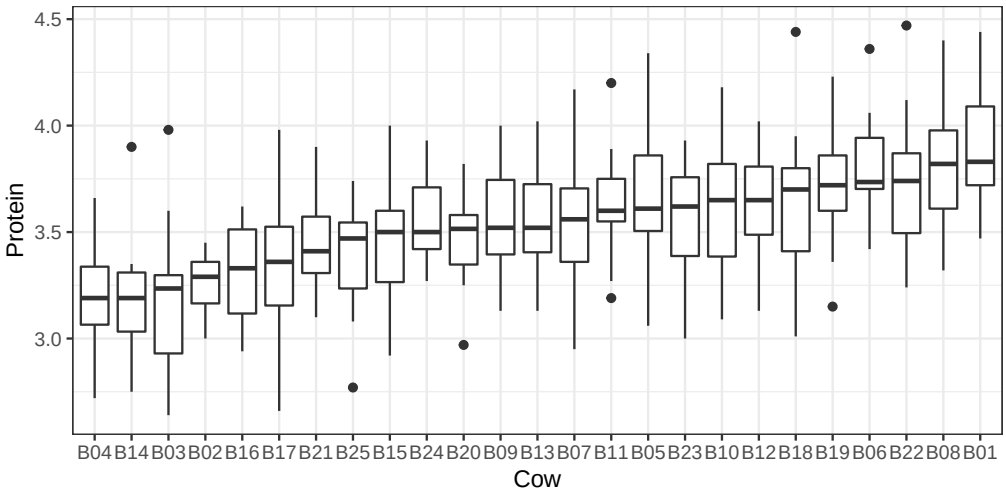


Figure 45: Protein content of milk from repeated measures across different cows from the milk dataset.

Each cow is assigned to one diet only, so we could not model the effects of both cow and diet as fixed effects. Assuming the interest is really in the 'population' of cows, rather than the cows in this particular data set, it makes sense to model the effects of the cows as random effects.

Plotting protein against time in Figure 46, we see a clear drop after week 1. We'll introduce an indicator variable for whether the sample was taken in week 1, but otherwise not model the effect of time.

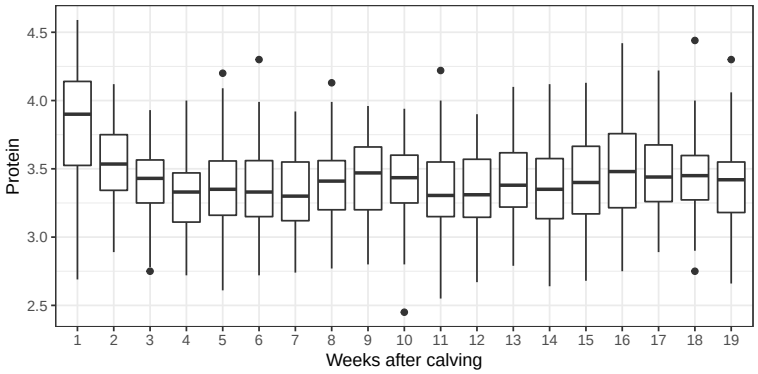


Figure 46: Milk yields observed at different weeks after calving. A simple approach to model the effect of time would be to assume yields change after week 1, but are constant, on average thereafter.

We'll create a new data frame `milkDF`, with the indicator variable added:

```
milkDF <- nlme::Milk %>%
  mutate(weekOne = Time == 1)
```

6.8.2 The model and hypothesis

We consider the model

$$Y_{ijt} = \mu + \tau_i + \beta I(t = 1) + b_{ij} + \varepsilon_{ijt},$$

with

- Y_{ijt}: the protein content at time $t = 1, \ldots, 19$, for cow $j = 1, \ldots, n_i$ within group (diet) $i = 1, 2, 3$;
- μ: the expected protein content for a cow on diet 1 at time $t > 1$;
- τ_i: a fixed effect for diet i, with τ_1 constrained to be 0;
- β: the increase in expected protein content at time $t = 1$;
- $b_{ij} \sim N(0, \sigma_b^2)$, a random effect for each cow;
- $\varepsilon_{ijt} \sim N(0, \sigma^2)$ a random error term.

We suppose the null hypothesis of interest is

$$H_0 : \tau_2 = \tau_3 = 0,$$

so that diet has no effect on protein levels. The alternative hypothesis is that at least one τ_i is non-zero. One way to do this test would be to use the generalised likelihood ratio test, which we recap in the following section.

6.8.3 The generalized likelihood ratio test

We use the following notation. Suppose observations are independent and the i^{th} observation is drawn from the density (or probability function, if discrete) $f(y; x, \psi)$ where ψ is a vector of unknown parameters, and x is a vector of any corresponding independent variables Then the likelihood and log-likelihood of the observations are

$$L(\mathbf{y};\, \mathbf{x},\, \psi) = \prod_i f(y_i;\, x_i, \psi),$$

$$\ell(\mathbf{y};\, \mathbf{x},\, \psi) := \log L(\mathbf{y};\, \mathbf{x},\, \psi) = \sum_i \log f(y_i; x_i, \psi),$$

respectively.

Now suppose that a simplification is proposed that entails imposing r restrictions, $s_i(\psi) = 0$ for $i = 1, \ldots, r$, on the elements of ψ, where r is less than the number of components in ψ. These restrictions must be functionally independent, which is another way of saying all r of them are needed. In our example, we have two restrictions: $\tau_2 = 0$ and $\tau_3 = 0$. Suppose the maximum likelihood for the restricted situation is $\widehat{\psi}_s$, which is obtained by maximising ℓ subject to the constraints. When the simpler model is correct, approximately

$$-2\left(\ell(\mathbf{y};\, \mathbf{x},\, \widehat{\psi}_s) - \ell(\mathbf{y};\, \mathbf{x},\, \widehat{\psi})\right) \sim \chi_r^2$$

That is, twice the difference between the maximised value of the log-likelihood from the simpler model and that for the more complex one is approximately χ_r^2. On the other hand, if the simpler model is false the difference between these two is expected to be large; hence this theory can be used as the basis of a test, called the generalized likelihood ratio test (GLRT).

To summarise, to conduct a GLRT, we do the following:

1. Evaluate the maximised log-likelihood $\ell(\mathbf{y};\, \mathbf{x},\, \hat{\psi})$.
2. Evaluate the maximised log-likelihood $\ell(\mathbf{y};\, \mathbf{x},\, \hat{\psi}_s)$ with ψ constrained according to the null hypothesis.
3. Evaluate the test statistic

$$L = -2\left(\ell(\mathbf{y};\, \mathbf{x},\, \hat{\psi}_s) - \ell(\mathbf{y};\, \mathbf{x},\, \hat{\psi})\right).$$

   and compare with the χ^2_r distribution, where r is the number of constraints specified by the null hypothesis. So, if for example we find $L > \chi^2_{r;0.95}$, we would reject H_0 at the 5% level.

Various standard hypothesis tests can be constructed from the GLRT. In some cases we can derive the exact distribution of L, rather than using the χ^2 approximation, and in some other cases, the distribution of L is χ^2_r anyway.

However, for mixed effects models, and when testing fixed effects Pinheiro and Bates (2005, Section 2.4.2) say that this test can sometimes be badly anti-conservative, that is, it can produce p-values that are too small. Parametric bootstrapping gives us another way to do the test, without relying on the χ^2_r approximation.

6.8.4 The parametric bootstrap test

We need to simulate values of L *under the condition that* H_0 *is true*. One way to do this is to fit the reduced model to the data, then simulate new values of the dependent variables, keeping the independent variables the same, and assuming the parameters are equal to their estimated values. The algorithm is therefore:

1. Calculate the GLRT statistic for the observed data. Denote the value of the statistic by L_{obs}.
2. Fit the reduced model under H_0 to the data (i.e. leave out the `Diet` variable in our example).
3. Simulate new values of Y_{ijt} for $i = 1, 2, 3$, $j = 1, \ldots, n_i$ and $t = 1, \ldots, 19$ from the reduced model. This is our bootstrapped data set, but generated from a parametric model, rather than from an ECDF.
4. For the simulated data $\mathbf{y}^*$, calculate the GLRT statistic

$$L = -2\left(\ell(\mathbf{y}^*;\, \mathbf{x},\, \hat{\psi}_s) - \ell(\mathbf{y}^*;\, \mathbf{x},\, \hat{\psi})\right).$$

5. Repeat steps 3-4 N times, to get a sample of GLRT statistics $L_1, \ldots, L_N$.
6. Estimate the p-value by

$$\frac{1}{N}\sum_{i=1}^{N} I(L_i \geq L_{obs}).$$

6.8.5 Implementation with R

We first do the standard GLRT in R, using the `lme4` package (Bates et al., 2015) to fit the models. One technical detail that we won't worry too much about here is that we need to use 'ordinary' maximum likelihood rather than 'restricted' maximum likelihood (REML) for this test, even though the latter is preferred for model fitting.

We fit the full model and extract the maximised log likelihood:

```
fmFull <- lmer(protein ~ Diet  + weekOne + (1|Cow),
               REML = FALSE,
               data = milkDF)
logLik(fmFull)
```

```
## 'log Lik.' -147.7877 (df=6)
```

We then fit our reduced model under H_0, and extract the maximised log likelihood:

```
fmReduced <- lmer(protein ~ weekOne +(1|Cow),
                  REML = FALSE,
                  data = milkDF)
logLik(fmReduced)
```

```
## 'log Lik.' -155.5944 (df=4)
```

We can now compute an observed test statistic, and calculate a p-value, using the χ^2_r distribution, here with $r = 2$.

```
obsTestStat<- as.numeric(- 2*(logLik(fmReduced)
                              -logLik(fmFull)))
1 - pchisq(obsTestStat, 1)
```

```
## [1] 7.770794e-05
```

Here, `obsTestStat` is step (1) of our approach listed above.

Now we simulate test statistics using parametric bootstrapping; we simulate new data sets from the reduced model (step 3 in the above). We can use the `simulate()` function to do this:

```
set.seed(123)
N <- 1000
sampleTestStat<-rep(NA, N)

for(i in 1:N){

  # simulate new data (step 3)
  newY <- unlist(simulate(fmReduced))

  # calculate the test statistic under the simulated data (step 4)
  fmReducedNew<-lmer(newY ~ milkDF$weekOne  +
                       (1|milkDF$Cow),
                     REML = FALSE)
  fmFullNew <- lmer(newY ~ milkDF$Diet + milkDF$weekOne  +
                      (1|milkDF$Cow),
                    REML = FALSE)
  sampleTestStat[i] <- -2*(logLik(fmReducedNew) - logLik(fmFullNew))
}
```

Now we'll count the proportion of bootstrapped test statistics greater than or equal to our observed one (step 6):

```
mean(sampleTestStat >= obsTestStat)
```

```
## [1] 0
```

We didn't actually generate any test statistics as large as the observed one, so we'd need to generate more to get an estimated p-value. But if we compare the distribution of sampled test statistics with the χ^2_2 distribution, they are fairly similar. This is shown in Figure 47.

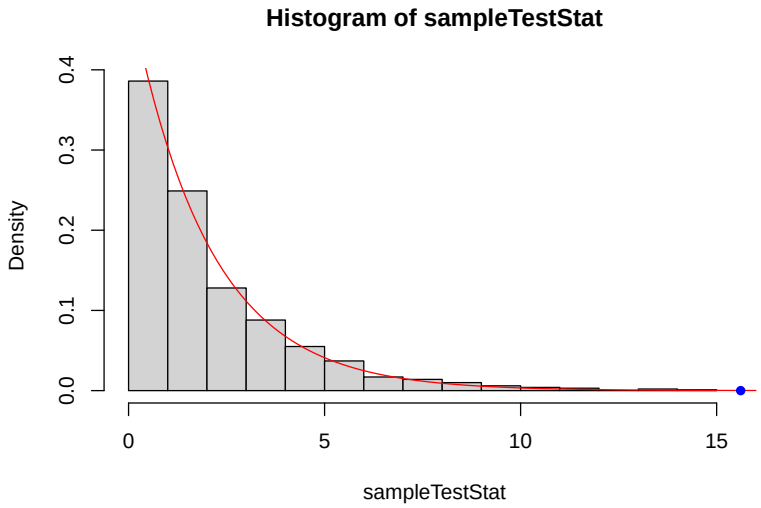


Figure 47: Comparing the distribution of the test statistic from bootstrapping (histogram) with the χ^2_2 approximation used in the generalised likelihood ratio test. The agreement looks fairly good here. The blue dot indicates the observed test statistic.

So the use of bootstrapping hasn't changed our conclusion here, but has given us a means of checking whether the distribution assumed in the GLRT is appropriate.

7 Cross-validation

Aims of this chapter
1. Understand cross-validation as a tool for both testing the predictive ability of a model, and for parameter estimation. 2. Learn about computational short-cuts for implementing cross-validation. 3. Explore the link between cross-validation and AIC for model selection.

We consider problems where the main interest is in *prediction* rather than inference: predicting the value of some dependent variable y given independent variables x. We suppose we have a data set

$$D = \{(x_i, y_i), i = 1, \dots, n\},$$

and we want to test the performance of some model or algorithm that, using these data, will predict y for any value of x.

The usual practice is to divide the data D into a training set and a test set. For each x in the test set, we would predict the corresponding y, using the training data only. For small data sets, this division may not be desirable; we may need to use all the data just to get good predictions and/or to reduce the uncertainty in our predictions as much as possible.

When we have a small data set, we can assess the performance of our model using cross-validation instead. The basic idea is as follows.

For $i = 1, \dots, n$ we

1. obtain a reduced data set D_{-i} by removing the observation (x_i, y_i) from the full data set D;
2. re-fit the model using the reduced data set D_{-i}, and predict the observation we removed in step 1: we obtain a prediction $\hat{y}_i$ of the dependent variable given x_i.

In removing any single observation, we're hoping that the fitted model using the data D_{-i} is similar to the fitted model based on all the data D; predictions wouldn't change much 'in general'. We then get a sample of n predictions errors $\hat{y}_1 - y_1, \dots, \hat{y}_n - y_n$ with which we can evaluate our model.

What we calculate precisely in step (2), and what we do with the results, will depend on the problem. We'll give two examples in this chapter. Note that this procedure is sometimes referred to as **leave-one-out** cross-validation, as we are leaving out a single observation in turn. **k-fold cross-validation** refers to leaving out k observations at a time. We will not consider this here, but the concept is the same.

7.1 Cross-validation in classification

7.1.1 The Palmer penguins data

We consider a data set from Gorman et al. (2014) and provided in an R package `palmerpenguins` (Horst et al., 2020). In this example, we suppose the aim is to predict ("classify") the species of

a penguin (one of Adelie, Chinstrap and Gentoo) based on the measurements of bill length, bill depth and flipper length. One can imagine that it may be difficult to obtain a large data set for this sort of problem.

Figure 48: Introducing the Palmer penguins. Artwork by Allison Horst.

The data look like this:

```
## # A tibble: 6 x 4
##   species bill_length_mm bill_depth_mm flipper_length_mm
##   <fct>            <dbl>         <dbl>             <int>
## 1 Adelie            39.1          18.7               181
## 2 Adelie            39.5          17.4               186
## 3 Adelie            40.3          18                 195
## 4 Adelie            NA            NA                  NA
## 5 Adelie            36.7          19.3               193
## 6 Adelie            39.3          20.6               190
```

There are 333 complete observations, so we have a data set $D = \{(y_i, x_i), i = 1, \ldots, 333\}$, where, for the i^{th} penguin:

- $y_i = 1$, 2 or 3 if the species is Adelie, Chinstrap or Gentoo, respectively, and
- $\mathbf{x}_i$ is a vector of the three measurements: bill length, bill depth and flipper length (all in mm).

For a new penguin with measurements $\mathbf{x}^*$, the aim would be to classify the species (y^*) given this measurement information alone.

Note that the penguin data from the `palmerpenguins` package contains 344 penguins, however only 333 of these have complete observations for the three measurement variables and the species type. We will not be considering this incomplete information in our analysis here, so have removed these individuals from the data we work with.

7.1.2 The K-nearest neighbours algorithm (KNN)

KNN is a simple algorithm for classification. There is no actual model, and just one parameter, K, to choose. Given a new penguin with measurements $\mathbf{x}$, the species is classified as follows:

1. For the penguins $i = 1, \ldots, 333$ in our data set D, calculate the Euclidean distance $||\mathbf{x}_i - \mathbf{x}||$.

Table 19: Summary of the scale of penguin measurements in the Palmer penguin data.

	Bill length	Bill depth	Flipper length
	Min. :32.10	Min. :13.10	Min. :172
	Median :44.50	Median :17.30	Median :197
	3rd Qu.:48.60	3rd Qu.:18.70	3rd Qu.:213

Table 20: Summary of the scaled penguin measurements in the Palmer penguin data.

	Bill length (scaled)	Bill depth (scaled)	Flipper length (scaled)
	Min. :-2.17471	Min. :-2.06418	Min. :-2.0667
	Median : 0.09275	Median : 0.06862	Median :-0.2830
	3rd Qu.: 0.84247	3rd Qu.: 0.77956	3rd Qu.: 0.8585

2. Find the K smallest Euclidean distances. This means that we have identified the K penguins in the data set D with measurements closest to the measurements $\mathbf{x}$ of the new penguin. These are the K *nearest neighbours*.
3. Classify the species of the new penguin as the species that occurred the most out of the K penguins found in step (2). If there is a tie, pick a species at random from those tied.

7.1.3 Implementing cross-validation in KNN

We can implement the KNN algorithm, using our training set of 333 penguins, for any measurement information that is obtained for a new penguin. However, we have no understanding of how well this algorithm is working, i.e. its *prediction ability*. We can asses how well this algorithm is working using cross-validation. We do the following, for $i = 1, \dots, 333$:

1. Remove penguin i from the data set D, to obtain a reduced data set D_{-i}.
2. Calculate the Euclidean distances $||\mathbf{x}_j - \mathbf{x}_i||$ for each $\mathbf{x}_j$ in the reduced data set D_{-i}. This would involve $j = 1, \dots, i-1, i+1, \dots, 333$.
3. Find the K penguins in D_{-i} with the smallest Euclidean distances from step (2), and obtain their species.
4. Obtain the predicted species $\hat{y}_i$ of the i-th penguin as the species that occurred the most in step (3).

We do this in `R` in the following. As we're using Euclidean distances, we'll convert the independent variables to be on (roughly) the same scale. If we did not do this, our algorithm could be placing more weight on classifying the penguins based on only one of the three measurements. To see this, we note the summary information of the three variables in Table 19. The scale of the flipper length variable is much larger than that of the bill depth, and therefore the distance measurement used as part of the KNN algorithm will be dominated by the differences in flipper length, causing this to be the determining variable for the classification process. When we scale the measurements we get comparable variables, as shown in Table 20.

We can implement the KNN algorithm using the `knn` function from the `class` package (Venables and Ripley, 2002). This function produces the predicted classification, based on the training variables

(`train`, given here by the $\mathbf{x}_j$ from the reduced data set D_{-i}), the training observations (`cl`, given here by the y_j from the reduced data set D_{-i}), and the test variables (`test`, given here by $\mathbf{x}_i$). Recall we apply this process for each of our 333 observations, so here we use a `for` loop:

```
yhat <- factor(rep(NA, length(y)), levels = levels(y))

for(i in 1:length(y)){
  yhat[i] <- class::knn(train = X[-i, ],
                        cl = y[-i],
                        test = X[i, ])
}
```

As an example of this in practice, let's look at penguin 150. We have the measurements:

```
X[150, ]
```

```
## [1]  1.0984771 -0.9977806  1.2152767
```

and the observation that it is a Gentoo penguin. When we remove this penguin from the datset and use the remaining 332 penguins to classify its species, we get the prediction ($\hat{y}_{155}$) of Gentoo. So the classification was correctly predicted when this penguin was removed from the data set. We can see how well all the observations were predicted using the `confusionMatrix` function from the `caret` package (Kuhn, 2021):

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie       142         3      0
##   Chinstrap      4        65      0
##   Gentoo         0         0    119
```

So we can see there were only 7 misclassified penguins out of 333 (4 Adelie penguins predicted to be Chinstrap, and 3 Chinstrap penguins predicted to be Adelie). The cross-validation approach here not only gives us an overall indication of predictive ability of a method, such as that 326 of the 333 penguins were correctly classified when removed from the training data, but also additional information about the classification problem. For instance, here we see that all penguins that were Gentoo were correctly classified, and the only misclassification is between Adelie and Chinstrap, so we might be interested in further investigating this overlap between these two species.

7.2 Cross-validation in regression

7.2.1 The flint tools data

We'll consider a simple example using a small data set of flint tools, where the breadth and length of each tool has been measured. (The context here is that we'd like to be able to estimate the original length of a damaged tool with its tip missing.) Again, we can imagine that such data would be scarce; we can't easily obtain a large data set.

```
## Warning: Missing column names filled in: 'X3' [3]
```

```
## Warning: 10 parsing failures.
```

```
## row col  expected    actual                  file
##   1  -- 3 columns 2 columns 'Data/flintsdata.txt'
##   2  -- 3 columns 2 columns 'Data/flintsdata.txt'
##   3  -- 3 columns 2 columns 'Data/flintsdata.txt'
##   4  -- 3 columns 2 columns 'Data/flintsdata.txt'
##   5  -- 3 columns 2 columns 'Data/flintsdata.txt'
## ... ... ......... ......... .....................
## See problems(...) for more details.
```

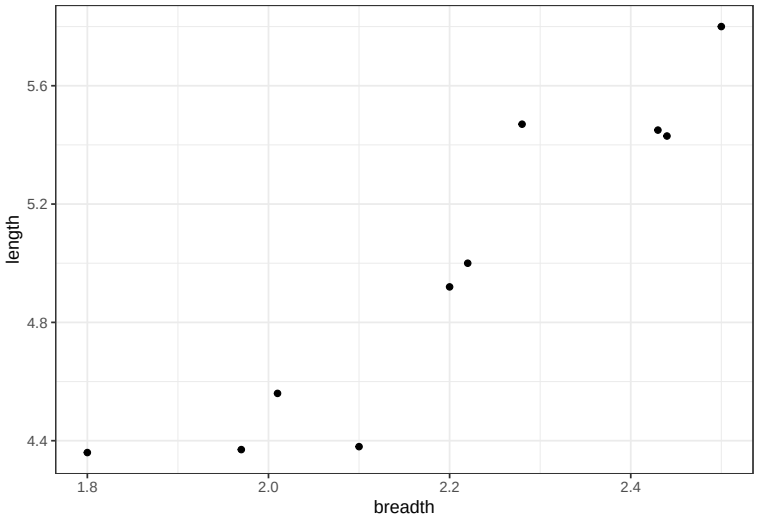


Figure 49: A small data set with the measurements of ten flint tools.

7.2.2 Implementing cross-validation in regression

Define $D = \{(x_i, y_i), i = 1, \ldots, 10\}$ to be the full data set, with x_i and y_i the breadth and length, respectively, in mm, for the i^{th} tool. We'll consider a simple linear regression model of the form

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

with $\varepsilon_i \sim N(0, \sigma^2)$.

Given parameter estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ and a tool breadth x, we would predict the corresponding tool length as $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$, and we can obtain a prediction interval using standard theory.

Linear models are well understood, and there are various diagnostics we can do to check goodness-of-fit, model assumptions and so on, so that we wouldn't normally consider cross-validation methods. However, as the sample size is so small, it's going to be harder to judge from the usual diagnostics whether we can really trust predictions from our model. Despite this, it would be interesting to get some idea how the model performs on 'new' observations. Further, we could apply the same ideas for more complex, 'black-box' regression models, where we don't have the same diagnostic tools that we have for simple linear regression.

We'll use cross-validation in a similar way as before, removing each observation in term and forming a prediction. To obtain this prediction, we will re-estimate the model parameters each time, and also compute a prediction interval in addition to an estimate $\hat{y}_i$. The procedure is as follows.

For $i = 1, \ldots, 10$:

1. Remove the observation (x_i, y_i) from the data set D to get a reduced data set D_{-i}.
2. Using D_{-i}, compute least squares estimates of β_0 and β_1, and estimate σ^2 with the corresponding residual sum of squares. Denote these estimates by $\hat{\beta}_{0,i}$ and $\hat{\beta}_{1,i}$, and $\hat{\sigma}_i^2$ respectively.
3. Estimate the i^{th} tool length using

$$\hat{y}_i = \hat{\beta}_{0,i} + \hat{\beta}_{1,i} x_i,$$

   and obtain the corresponding (95%) prediction interval.

These estimates and intervals can then be compared with the original observations $y_1, \ldots, y_{10}$.

In R, we'll first set up an empty matrix to take the results:

In `R`, we implement this as:

```
flint_yhat <- matrix(NA, nrow = 10, ncol = 3)

for(i in 1:10){

  lmReduced <- lm(length ~ breadth, subset = -i, data = flint)
  flint_yhat[i, ] <- predict(lmReduced, flint[i, ], interval = "prediction")

}
```

Figure 50 displays the results of this cross-validation experiment. For each observed breadth, the true length is shown in red. The fitted regression line from the complete set of 10 observations is shown in grey. The cross-validation results are shown in black, both as a point estimate and an interval. Note that the predictions do not sit on the regression line because they are each obtained by removing that observation from the training data. We can see, however, that the cross-validation predictions are in most cases close to the full data prediction which tells us that no single observation is having a great effect on the predicted model coefficients.

7.3 Parameter estimation with cross-validation

So far, we've used cross-validation to check how well a model (or algorithm) predicts observations. But we can also use it to estimate/choose parameters in our model/algorithm. We define a *loss function* $L(\hat{y}_i, y_i)$ which gives a penalty for predicting a value $\hat{y}_i$, using the reduced data set D_{-i}, when the true value is y_i. We could then choose our model parameters to minimise

$$\sum_{i=1}^{n} L(\hat{y}_i, y_i).$$

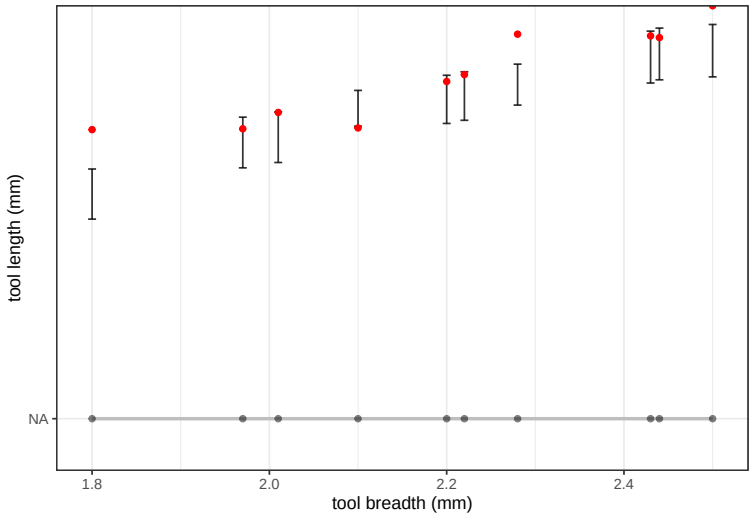


Figure 50: The results of cross-validation on the flint tools data. The observed data is shown in red. Predictions (black point) of the length, and intervals, are shown for each breadth measurement, obtained by removing that measurement from the training data. The regression line using all 10 observations is also included for comparison.

7.3.1 Example: choosing the value K in KNN

Recall that in the KNN algorithm we have to decide how many 'nearest neighbours' K to look for. Note that in the penguin example earlier, we implemented $K = 1$. This parameter could be selected using cross-validation. An appropriate loss function would be

$$L(\hat{y}_i, y_i) = \begin{cases} 0 & \text{for } \hat{y}_i = y_i; \\ 1 & \text{for } \hat{y}_i \neq y_i. \end{cases}$$

This loss function would lead to us searching for the K that minimises the number of incorrect predictions.

We'll try this for the Palmer penguins. We've already seen that the algorithm works about as well as we could hope for using $K = 1$ as there was only 7 incorrect predictions, so there's not really any scope for improvement, but just for illustration, we'll try a few different K.

Rather than using a for loop, we'll use the function `knn.cv()`, which implements cross-validation for us:

```
peng_yhat <- class::knn.cv(train = X, cl = y, k = 1)
peng_yhat10 <- class::knn.cv(train = X, cl = y, k = 10)
peng_yhat100 <- class::knn.cv(train = X, cl = y, k = 100)
```

The sum of the loss functions for these K are then 7, 6 and 20, respectively for $k = 1, 10, 100$. As we expected, there isn't an argument for increasing the complexity of our classification algorithm here (as for $K > 1$ we have to identify the most common classification from a number of neighbours) because there isn't much difference between $K = 1$ or 10, but the performance is markedly worse for $K = 100$.

7.3.2 Cross-validation as an alternative to maximum likelihood

For the KNN algorithm, there is no model; we couldn't write down a likelihood function. Cross-validation is a natural way to choose K: if we *didn't* remove y_i before predicting it, the nearest neighbour to the i-th observation would be itself, and we'd always get 100% prediction accuracy with $K = 1$ (assuming the independent variables are all distinct).

When we do have a model, cross-validation could be used as an alternative way to estimate parameters; there are advantages and disadvantages. If we only care about getting good point predictions for new observations y, cross-validation (with a suitable loss function) will attempt to get small prediction errors, *regardless of whether the model assumptions are valid.*

If we also care about quantifying uncertainty in our predictions, we could try to design a suitable loss function, but maximum likelihood may be preferable. With maximum likelihood, within our chosen model, we're trying to find the parameters that can 'best' describe the distribution of the data, and this will help us with our uncertainty quantification.

As we'll see later, there is a relationship between cross-validation and AIC, so cross-validation may be useful in situations where we want to penalise for model complexity.

7.4 Computational short-cuts

If we need to *repeatedly* do cross-validation, for example if we're using it to estimate parameters, then the method can get computationally expensive. For example, if we have n observations, and we want to try d different parameter values to find the 'best' choice, we would have to fit the model $n \times d$ times.

For linear models, it turns out that we don't actually need to refit the model each time with a reduced data set: we can compute everything we need from a single model fit to *all* the data. There are similar results for some other types of model, so we need to be alert to this possibility, as it may speed up the computation.

For a linear model, given the full data set D, define as follows:

- $\hat{\beta}$: the least squares estimator of β;
- X: the design matrix;
- H: the 'hat matrix' given by $X(X^TX)^{-1}X^T$
- $f(x_j)^T$: the j^{th} row of the design matrix X;
- r_j: the j^{th} residual, $r_j = y_j - f(x_j)^T\hat{\beta}$.

Now, when the i^{th} observation is removed from the data set D, denote the corresponding least squares estimate by $\hat{\beta}_{(i)}$ and define

$$r_{(i)} = y_i - f(x_i)^T\hat{\beta}_{(i)}, \tag{47}$$

which is the difference between the i^{th} observed dependent variable and its estimate obtained when this observation is removed from the data D. Note that $r_{(i)}$ is sometimes referred to as the i^{th} *deletion residual*, and is not equal to r_i because the least squares estimators are different (one is based on the full dataset and the other is from the reduced data).

Earlier we used a for loop to compute $\hat{\beta}_{(i)}$ and $f(x_i)^T\hat{\beta}_{(i)}$ for $i = 1, \ldots, n$, removing each point in turn, refitting the model, and calculating the least squares estimate. But it isn't necessary: it can be shown that

$$r_{(i)} = \frac{r_i}{1 - h_{ii}}, \tag{48}$$

and

$$\hat{\beta}_{(i)} = \hat{\beta} - (X^T X)^{-1} x_i r_{(i)}, \tag{49}$$

where h_{ii} is the i^{th} diagonal element of the hat matrix H. Proofs are given at the end of this chapter for reference, but you will not be assessed on these results or the proofs in this module.

In summary, we do not need to implement a for loop like earlier, refitting the model for each removed location. Instead, we only fit the model once to the full data-set, and then calculate each deletion residual based on Equation (48).

7.4.1 Example: Returning to the flint data

Let's check this short cut in R on a data-set. First, we'll fit the flint model using all the data, and obtain X and H:

```
lmFull <- lm(length ~ breadth, data = flint)
X <- model.matrix(lmFull)
H <- X %*% solve(t(X) %*% X, t(X))
```

Then we'll calculate the deletion residuals using the relationship in Equation (48):

```
(deletionResiduals <- lmFull$residuals/(1 - diag(H)))
```

```
##           1           2           3           4           5           6
## -0.13517895 -0.11047994 -0.03252041 -0.43621461 -0.05516638  0.21738803
##           7           8           9          10
##  0.45743603  0.34848697 -0.07227648 -0.00645657
```

We'll check these are the same as the deletion residuals that are calculated via Equation (47) using the results from our earlier approach of using a for loop and repeatedly fitting the linear model on the reduced data:

```
flint$length - flint_yhat[ ,1]
```

```
##  [1] -0.13517895 -0.11047994 -0.03252041 -0.43621461 -0.05516638  0.21738803
##  [7]  0.45743603  0.34848697 -0.07227648 -0.00645657
```

They are equal!

We'll also refit the model with the first observation removed to obtain $\hat{\beta}_{(1)}$, and check we get the same as via Equation (49):

```
lmReduced <- lm(length~breadth, subset = -1, data = flint)
lmReduced$coefficients
```

```
## (Intercept)     breadth
##   0.2820335   2.1437286
```

```
lmFull$coefficients -
  solve(t(X) %*%X, t(X[1, , drop = FALSE ])) * deletionResiduals[1]
```

```
##                      1
## (Intercept) 0.2820335
## breadth     2.1437286
```

Again, these are equal!

7.5 Relationship with AIC

Stone (1977) established a relationship between cross-validation and Akaike's Information Criterion (AIC). Recall that AIC is defined as

$$-2l(\hat{\theta}; y) + 2p,$$

where $l(\hat{\theta}; y)$ is the maximised log-likelihood function, and p is the number of parameters in the model. We can use AIC to compare models, with a lower AIC value indicating a better model fit, after penalising for the model complexity (number of parameters).

Now define a goodness-of-fit measure based on cross-validation as

$$CV := -2 \sum_{i=1}^{n} \log \pi(y_i | x_i, D_{-i}, \hat{\theta}_{(i)})$$

where $\hat{\theta}_{(i)}$ is the maximum likelihood estimate of the parameters given the reduced data set D_{-i}, and $\pi()$ is the predictive density function of the model. For example, in our simple linear regression model, we would have

$$y_i \mid x_i, D_{-i}, \hat{\theta}_{(i)} \sim N((1, x_i)\hat{\beta}_{(i)},\ \hat{\sigma}^2_{(i)}),$$

so that

$$\pi(y_i | x_i, D_{-i}, \hat{\theta}_{(i)}) = \frac{1}{\sqrt{2\pi\hat{\sigma}^2_{(i),MLE}}} \exp\left(-\frac{1}{2\pi\hat{\sigma}^2_{(i),MLE}} (y_i - (1, x_i)\hat{\beta}_{(i)})^2 \right),$$

where $\hat{\sigma}_{(i),MLE}$ is the maximum likelihood estimator of σ^2, given the reduced data set D_{-i}. Stone then shows that asymptotically, CV is equivalent to AIC. Intuitively, as the sample size tends to infinity, the maximised log-likelihood would dominate the penalty term $2p$, and we would expect $\hat{\theta}_{(i)}$ to be similar to $\hat{\theta}$, so that both AIC and CV are just reporting the maximised log-likelihood. But this does suggest that cross-validation does have the effect of penalising for model complexity, in a similar way to AIC.

7.5.1 Example: The cars data

We'll use the `cars` data, one of R's in-built datasets, and consider a simple linear regression model to predict stopping distance (`dist`) given the initial travelling speed (`speed`) for some (very old!) cars. There are 50 observations in this data set.

We'll first fit the model to all the data:

```
lmFull <- lm(dist ~ speed, data = cars)
```

This gives the AIC as 419.156863 and minus twice the maximised log likelihood as 413.156863.

Now we'll compute CV. We'll go back to using a for loop, to make the code a bit more readable:

```
X <- model.matrix(lmFull)
n <- nrow(X)
y <- cars$dist
logPredictiveDensity <- rep(NA, n )

for(i in 1:n) {

  lmCV <- lm(dist ~ speed, subset = -i, data = cars)
  # Need the MLE of sigma. We've got n-1 observations so this is RSS/(n-1)
  #sigmaMLE <- sqrt(sum(lmCV$residuals^2) / (n - 1))
  logPredictiveDensity[i] <- dnorm(y[i],
                                   sum(X[i, ]*lmCV$coefficients),
                                   sigma(lmCV),
                                   log = TRUE)
}

-2 * sum(logPredictiveDensity)
```

```
## [1] 421.028
```

This is close to the value obtained by AIC, and closer than when compared with minus twice the maximised log likelihood so it does appear that CV is penalising for the number of parameters.

7.6 (Non-examinable) Proof of the computational short cut

We outline how to derive the expressions for $r_{(i)}$ and $\hat{\beta}_{(i)}$ in Equations (48) and (49).

7.6.1 Helpful results we will use

There are two results that help us here (and then it's just fairly straightforward algebra - we won't give all the steps here). Recall that $f(x_i)^T$ is the i-th row of the design matrix X.

The first result is that

$$X^T X = \sum_{i=1}^{n} f(x_i) f(x_i)^T,$$

so we can write

$$X_{(i)}^T X_{(i)} = X^T X - f(x_i) f(x_i)^T, \tag{50}$$

where $X_{(i)}$ is the design matrix with the i-th row (observation) removed. We will use this result multiple times in the following to relate the full dataset to the reduced dataset.

The second result is the Woodbury identity for matrix inversion:

$$(A + UCV^T)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^T A^{-1}U)^{-1}VA^{-1},$$

for matrices of the appropriate sizes (with A and C invertible). To relate this to our scenario, we equate

- $X^T X$ with A,
- U with $-f(x_i)$,
- V with $f(x_i)$,

and set C to be the (1×1) identity matrix. After a little algebra we have

$$(X_{(i)}^T X_{(i)})^{-1} = (X^T X)^{-1} + \frac{(X^T X)^{-1} f(x_i) f(x_i)^T (X^T X)^{-1}}{1 - h_{ii}}, \tag{51}$$

where h_{ii} is the i-th diagonal element of $X(X^T X)^{-1} X^T$.

7.6.2 Relate the estimated coefficients of full and reduced models

Next, we relate $\hat{\beta}$ with $\hat{\beta}_{(i)}$.

Define $y_{(i)}$ to be the vector of all observations, with y_i removed, so that the estimated coefficients when the i^{th} observation is removed is

$$\hat{\beta}_{(i)} = (X_{(i)}^T X_{(i)})^{-1} X_{(i)}^T y_{(i)}. \tag{52}$$

Equation (50) also tells us that

$$X_{(i)}^T y_{(i)} = X^T y - y_i f(x_i). \tag{53}$$

Substituting Equations (51) and (53) into Equation (52) gives

$$\hat{\beta}_{(i)} = (X^T X)^{-1}(X^T y - y_i f(x_i)) + \frac{(X^T X)^{-1} f(x_i) f(x_i)^T (X^T X)^{-1}(X^T y - y_i f(x_i))}{1 - h_{ii}}.$$

which, after some algebra, becomes

$$\hat{\beta}_{(i)} = \hat{\beta} - (X^T X)^{-1} f(x_i) \frac{r_i}{1 - h_{ii}}. \tag{54}$$

7.6.3 Relating the residuals

We have that the deletion residual is

$$r_{(i)} = y_i - f(x_i)^T \hat{\beta}_{(i)},$$

and after substituting in Equation (54) for $\hat{\beta}_{(i)}$, we get

$$r_{(i)} = \frac{r_i}{1 - h_{ii}}.$$

Part IV

Approximate inference without sampling

8 Variational inference

Aims of this section
1. Appreciate the limitations of inference via sampling schemes, and that approximate inference can be a useful alternative in some situations. 2. Understand the need to assess 'agreement' between distributions as part of this approximation process, and the role that KL divergence and ELBO play in this. 3. Have an awareness of how approximate inference is practically implemented, including the choice of approximation using the mean field family and the optimisation process using coordinate ascent.

In this chapter we are going to introduce variational inference: an approach to approximating probability distributions that can be used in Bayesian inference, as an alternative to MCMC. Variational methods are popular for implementing Bayesian approaches in machine learning, and work well with large and complex data sets.

This and the following chapter are based on a review article (Blei et al., 2017).

8.1 Background theory

8.1.1 Jensen's inequality

Although not necessary for understanding the concepts in this chapter, they hinge on this important result. Jensen's inequality relates to the integrals of convex functions, and we are interested in this applied to expectations of logarithms (because log is a concave function):

$$\log(\mathbb{E}(f(x))) \geq \mathbb{E}(\log(f(x))),$$

for $f(x) > 0$.

8.1.2 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence is a measure of how one probability distribution is different from a second. Often we might compare distributions, such as two choices of prior distribution, by looking at summaries such as moments (mean/variance) or quantiles. However, these can only give snapshots of the similarity between two distributions. The KL divergence is a summary that takes into account the entire functional form of the distributions across all real values.

We will define KL divergence only in terms of continuous distributions, but note that this can similarly be defined for discrete distributions. The KL divergence between the distribution p and

q is given by

$$KL(p \mid\mid q) = \int_{\mathbb{R}} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{55}$$

$$= \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] \tag{56}$$

$$= \mathbb{E}_{p(x)}\left[\log p(x)\right] - \mathbb{E}_{p(x)}\left[\log q(x)\right]. \tag{57}$$

Note the the KL divergence is defined only if $q(x) = 0$ implies that $p(x) = 0$, for all x, and the contribution of such to the KL measure is interpreted as 0.

Important properties of the KL divergence include:

- Non-symmetry, i.e. $KL(p \mid\mid q) \neq KL(q \mid\mid p)$.
- Non-negative, i.e. $KL(p \mid\mid q) \geq 0$ for all p, q. Note that this result is proven based on Jensen's inequality.
- Equality identifying, i.e. $KL(p \mid\mid q) = 0$ if and only if $p(x) = q(x)$ for all x.

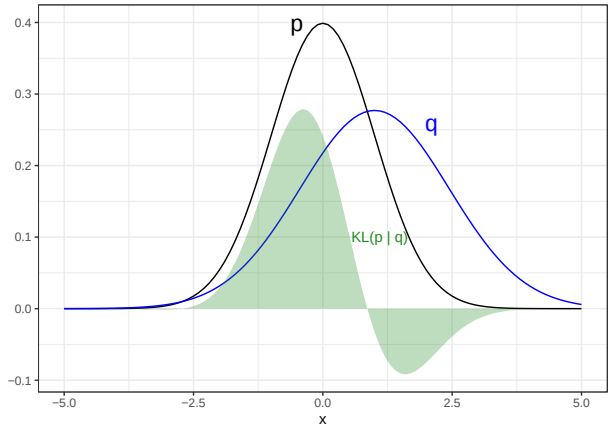


Figure 51: Example of the KL divergence between two Gaussian distributions. The contribution to the KL divergence is shown by the green region, with the KL figure being the area of this region.

The KL divergence is depicted in Figure 51, showing the densities of two Gaussian distributions, p (black) and q (blue). The contribution to the KL divergence, $KL(p \mid\mid q)$, is shown by the green region, and the actual value of the KL divergence is the area of this region. We see that the disagreement between distributions is not handled uniformly by the KL divergence. In this example the contribution to the KL divergence of p from q is not symmetric, it has higher penalty for p having density where q has little and less penalty for not having significant density where q has high density. We will return to this behaviour of the KL divergence later, as it plays an important role as to the use of $KL(p \mid\mid q)$ versus $KL(q \mid\mid p)$, which will produce different values. In this example, $KL(p \mid\mid q) = 0.42$ and $KL(q \mid\mid p) = 0.51$.

8.1.3 Optimisation with coordinate ascent

Coordinate ascent is an optimisation procedure to find

$$\arg\max_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}).$$

The main idea is to find the maximum of $f(\cdot)$ by maximising along one dimension at a time, holding all other dimensions constant—it being much easier to deal with one-dimensional problems. We therefore loop through each dimension $i \in 1, \ldots, n$ over a number of steps k, setting

$$x_i^{k+1} = \arg\max_{y\in\mathbb{R}} f(x_1^k, \ldots, x_{i-1}^k, y, x_{i+1}^k, \ldots, x_n^k).$$

An optimal update scheme involves using the gradient of the above function and setting

$$x_i^{k+1} = x_i^k + \epsilon \frac{\partial f(\mathbf{x})}{\partial x_i},$$

for $i = 1, \ldots, n$. Usually, a threshold will be set to monitor that convergence has been met, stopping the coordinate ascent algorithm once the difference between x_i^{k+1} and x_i^k is under such threshold.

We will not concern ourselves with the details of why this works, but it is enough to know the algorithm and that we end up moving in one dimension at a time, iteratively climbing the function to the maximum. We will be using this approach in the next chapter.

8.1.4 Stochastic optimisation

Another option for optimisation that we will look at in this part is stochastic optimisation. Here, the idea is to introduce $h^k(\mathbf{x}) \sim H(\mathbf{x})$, where $\mathbb{E}\left[h^k(\mathbf{x})\right] = \frac{df(\mathbf{x})}{d\mathbf{x}}$. Similarly to the above we update in each dimension with

$$x_i^{k+1} = x_i^k + \epsilon^k h^k(x_i^k),$$

where there are some conditions that we will not cover on the choice of ϵ. Again, we won't concern ourselves too much with the underlying theory here—but will show you this implementation in practice at the end of this part of the course.

8.2 Motivation for approximate inference approaches

8.2.1 Intractable integrals

When we implement Bayesian inference, our goal is evaluate the posterior distribution

$$p(\theta|x) = \frac{p(\theta)p(x|\theta)}{p(x)} = \frac{p(x,\theta)}{p(x)},$$

to learn about some parameters of interest θ, given observations x. For conciseness, we will write $p(x,\theta)$ instead of $p(\theta)p(x|\theta)$, but you should be thinking of multiplying a prior $p(\theta)$ and a likelihood $p(x|\theta)$ any time you see the joint density $p(x,\theta)$.

The numerator in the above is often tractable and easily evaluated. The difficulty in evaluating the posterior arises from the denominator, which we often refer to as the *normalising constant* and is also often called the *evidence*. This is given by

$$p(x) = \int_{\Theta} p(x, \theta) d\theta.$$

Previously, we used MCMC to get around this problem, as it was only necessary to be able to evaluate $p(x, \theta)$ and not $p(x)$. As we will see shortly, variational inference has the same feature.

8.2.2 Variational approach to intractable integrals

Variational inference is an **approximation** to the posterior. Unlike MCMC, the main computational effort is in constructing the approximation. We approximate $p(\theta|x)$ by an alternative distribution, $q^*(\theta)$. This distribution will be simple enough to be tractable, so that we can evaluate quantities of interest such as $\mathbb{E}_{q^*}(\theta)$ directly, or obtain i.i.d. samples very easily.

The aim of variational inference therefore is to find the particular form of $q^*(\cdot)$ so that it is both tractable, and *it is a good approximation to* $p(\theta|x)$, which is intractable. We will assume that we have a family of candidate distributions, which we refer to as $\mathcal{Q}$. We will return to exactly what form this family will take later, but for now you can imagine that θ is one-dimensional, and $\mathcal{Q}$ could be the family of all Gaussian distributions. The process of variational inference in such a scenario would be choosing the value of the mean and variance of the Gaussian distribution that is most similar to the posterior we are interested in.

The benefit of variational inference is that it tends to be very fast, and this is the reason for its choice over MCMC. Note that this is a trade-off situation: you are trading having a solution quickly in exchange for it no longer being the exact posterior. The speed makes it more appealing in problems involving large data sets. Note that it is generally the case that variational inference underestimates the variance of the posterior, and so it is better suited for problems where this uncertainty estimate is not the primary interest.

In the next sections we will cover the details of variational inference, which is ultimately the process of choosing which $q(\theta) \in \mathcal{Q}$ is the 'best' approximation to $p(\theta|x)$. Note here that although we write $q(\theta)$, this is a simplification. There will be some parameters involved in this distributional definition—the one-dimensional Gaussian example would involve there being a choice of the value of the mean and variance of such a Gaussian, and these would be the variational parameters. We do not include these in our notation to allow flexibility in the form of the distributions in the family $\mathcal{Q}$. Further, the values that these variational parameters will take will depend on the observed data x in some way. We could therefore write $q(\theta|\phi, x)$ to explicitly state this, where ϕ are the variational parameters. However, for readability throughout the following we will drop this and only use $q(\theta)$.

8.3 Approximate inference as an optimisation problem

We have stated that variational inference involves approximating the posterior of interest, $p(\theta|x)$, with a tractable distribution $q^*(\theta)$. This approximate distribution, $q^*(\theta)$, is chosen as the 'best' candidate from a family of distributions $q(\theta) \in \mathcal{Q}$. In this context, by 'best' candidate, we mean the

distribution in $\mathcal{Q}$ that is most similar in functional form to the posterior. We will use the Kullback-Leibler divergence to quantitatively measure the 'similarity' between the candidate distributions and the posterior of interest. To formalise the choice of the approximate distribution, this is

$$q^*(\theta) = \underset{q(\theta)\in\mathcal{Q}}{\arg\min}\ KL(q(\theta)\ ||\ p(\theta|x)). \tag{58}$$

|| p(|x)). \end{equation}

So, our inference approach would be to calculate the KL divergence from $p(\theta|x)$ for each $q(\theta) \in \mathcal{Q}$ and then choosing the distribution with the smallest divergence. We'll now examine how we deal with the unknown normalising constant $p(x)$. First note that

$$\begin{aligned} KL(q(\theta)\ ||\ p(\theta|x)) &= \mathbb{E}_q\left[\log q(\theta)\right] - \mathbb{E}_q\left[\log p(\theta|x)\right] && (59)\\ &= \mathbb{E}_q\left[\log q(\theta)\right] - \mathbb{E}_q\left[\log\left(\frac{p(x,\theta)}{p(x)}\right)\right] && (60)\\ &= \mathbb{E}_q\left[\log q(\theta)\right] - \mathbb{E}_q\left[\log p(x,\theta)\right] + \mathbb{E}_q\left[\log p(x)\right] && (61)\\ &= \mathbb{E}_q\left[\log q(\theta)\right] - \mathbb{E}_q\left[\log p(x,\theta)\right] + \log p(x). && (62) \end{aligned}$$

Where we obtain the final line in the above because $p(x)$ is a constant with respect to q, so $E_q(\log p(x)) = \log p(x)$. So our issue is that we cannot actually calculate the KL divergence for each $q(\theta) \in \mathcal{Q}$, because we cannot evaluate the final term in the above involving $p(x)$.

We define another measure, which is an element of the KL divergence calculation:

$$KL(q(\theta)\ ||\ p(\theta|x)) = \underbrace{\mathbb{E}_q\left[\log q(\theta)\right] - \mathbb{E}_q\left[\log p(x,\theta)\right]}_{-ELBO(q)} + \log p(x).$$

This is the **evidence lower bound**, or **variational lower bound**, given by

$$\begin{aligned} ELBO(q) &= \mathbb{E}_q\left[\log p(x,\theta)\right] - \mathbb{E}_q\left[\log q(\theta)\right] && (63)\\ &= \mathbb{E}_q\left[\log\left(\frac{p(x,\theta)}{q(\theta)}\right)\right]. && (64) \end{aligned}$$

Again, we are simplifying things somewhat by referring to ELBO as being a function of q alone, but we do this to highlight that $q(\theta) \in \mathcal{Q}$ will be the only varying input to the ELBO in practice (as our x have been observed). Therefore

$$KL(q(\theta)\ ||\ p(\theta|x)) = \log p(x) - ELBO(q).$$

The good news is that $ELBO(q)$ is something that we can calculate, as we are assuming that $p(x,\theta)$ is tractable. If the approach of variational inference was to choose the $q(\theta) \in \mathcal{Q}$ which minimises $KL(q(\theta)\ ||\ p(\theta|x))$, then because $\log p(x)$ is constant with respect to q, this is equivalent to maximising $ELBO(q)$.

Variational inference therefore approximates $p(\theta|x)$ with

$$q^*(\theta) = \underset{q(\theta)\in\mathcal{Q}}{\arg\max}\ ELBO(q(\theta)). \tag{65}$$

). \end{equation} The inference approach is an optimisation problem because once we have a chosen family of candidate distributions, $\mathcal{Q}$, we merely need to find the optimal candidate for the ELBO.

8.3.1 Exploring the ELBO

We have seen that variational inference is all about finding a distribution function that maximises the ELBO. Let us explore this further to gain an understanding of what this means for our chosen approximation to the posterior. We manipulate the above expression for the ELBO:

$$\begin{aligned} ELBO(q) &= \mathbb{E}_q\left[\log\left(\frac{p(x,\theta)}{q(\theta)}\right)\right] && (66)\\ &= \mathbb{E}_q\left[\log\left(\frac{p(\theta)p(x|\theta)}{q(\theta)}\right)\right] && (67)\\ &= \mathbb{E}_q\left[\log p(x|\theta)\right] + \mathbb{E}_q\left[\log\left(\frac{p(\theta)}{q(\theta)}\right)\right] && (68)\\ &= \mathbb{E}_q\left[\log p(x|\theta)\right] - \mathbb{E}_q\left[\log\left(\frac{q(\theta)}{p(\theta)}\right)\right] && (69)\\ &= \mathbb{E}_q\left[\log p(x|\theta)\right] - KL(q(\theta)\ ||\ p(\theta)). && (70) \end{aligned}$$

By choosing the q that maximising this expression:

- $\mathbb{E}_q\left[\log p(x|\theta)\right]$ is the expectation with respect to q of the likelihood of the data. Maximising this means finding the function q that explains the data.
- $KL(q(\theta)\ ||\ p(\theta))$ is the divergence from the prior. Minimising this (because of the minus sign in the ELBO expression above) means finding the function q that is similar to the prior. So intuitively, maximising the ELBO makes sense as we have the delicate balance that we would expect between remaining similar to our prior distribution, whilst matching the observed data.

We perform inference by finding the q that maximises the ELBO, but the value of the ELBO for this function can also be a useful measure in itself. Recall from above that

$$KL(q(\theta)\ ||\ p(\theta|x)) = \log p(x) - ELBO(q),$$

and also that $KL(\cdot||\cdot) \geq 0$. Therefore

$$ELBO(q) \leq \log p(x),$$

with equality if and only if $q(\theta) = p(\theta|x)$. This expression is where ELBO gets its name—it is the lower bound of the evidence (we said at the start of this chapter that a common name for the normalising constant is the evidence). Because of this relation, $ELBO(q^*)$ has been used as a model selection tool, under the assumption that equality has almost been reached. We won't discuss this further, as there is no justification from theory that this is the case.

8.3.2 Forward and reverse variational inference

We defined the variational approach as approximating the posterior with the distribution

$$q^*(\theta) = \underset{q(\theta)\in\mathcal{Q}}{\arg\min}\ KL(q(\theta)\ ||\ p(\theta|x)),$$

so that we choose the distribution from the candidate family with smallest KL divergence from the posterior. You might have questioned whether this is the only choice. Recall that the KL divergence is not symmetric so that

$$KL(q(\theta) \mid\mid p(\theta|x)) \neq KL(p(\theta|x) \mid\mid q(\theta)),$$

and so it's not immediately obvious why we have made the decision to use the KL divergence that we have done. Our decision of using $KL(q(\theta) \mid\mid p(\theta|x))$ is known as *reverse KL*, and the alternative option of using $KL(p(\theta|x) \mid\mid q(\theta))$ is known as *forward KL*. Here we will discuss briefly how inference is affected by the two approaches.

8.3.2.1 Forward KL In forward KL we find the q that minimises

$$KL(p(\theta|x) \mid\mid q(\theta)) = \int p(\theta|x) \log \left(\frac{p(\theta|x)}{q(\theta)} \right) d\theta,$$

and this ordering is referred to as weighting the difference between $p(\theta|x)$ and $q(\theta)$ by $p(\theta|x)$.

Consider the values of θ such that $p(\theta|x) = 0$. In the above, it does not matter what the value of $q(\theta)$ is in terms of the contribution to the KL divergence. So there is no consequence for there being large disagreement between the two functions at these values of θ. Minimising the forward KL divergence therefore optimises q to be non-zero wherever $p(\theta|x)$ is non-zero, and penalises most heavily at the values of θ with high density in $p(\theta|x)$. This leads to a good global fit, and this approach being known as **moment matching** (because the *mean* of the posterior is mimicked with highest importance) or **zero avoiding** (because we are avoiding $q(\theta) = 0$ when $p(\theta|x) > 0$).

To see the outcome of this approach visually, we show a one-dimensional posterior in Figure 52 that has two modes. If the candidate distribution family was Gaussian, then two possibilities for the approximate distribution are shown in the two panels, with the optimal choice under forward KL divergence being the right hand panel. You can see from the contribution to the KL divergence in green that not placing density where there is density in the posterior is highly penalised by the forward KL, which forces the optimal choice to be the right have option, even though the approximate distribution has its mode in a region of very low density in the posterior.

8.3.2.2 Reverse KL In reverse KL (the version that is most commonly used in practice and that we have introduced as our approach for variational inference) we find the q that minimises

$$KL(q(\theta) \mid\mid p(\theta|x)) = \int q(\theta) \log \left(\frac{q(\theta)}{p(\theta|x)} \right) d\theta,$$

and so now $q(\theta)$ is the 'weight'.

In this scenario, values of θ that lead to $q(\theta) = 0$ have no penalty. So there is no penalty for ignoring the region where the posterior has density. Instead, it is the region where $q(\theta) > 0$ that contributes large penalties to the KL divergence. Therefore, it does not matter if $q(\theta)$ fails to place density on areas of the posterior that have density, as long as the region where we do place density very closely mimics the posterior. The optimal approximation will avoid spreading density widely and will have good local fits to the posterior. This approach is known as **mode seeking** (because

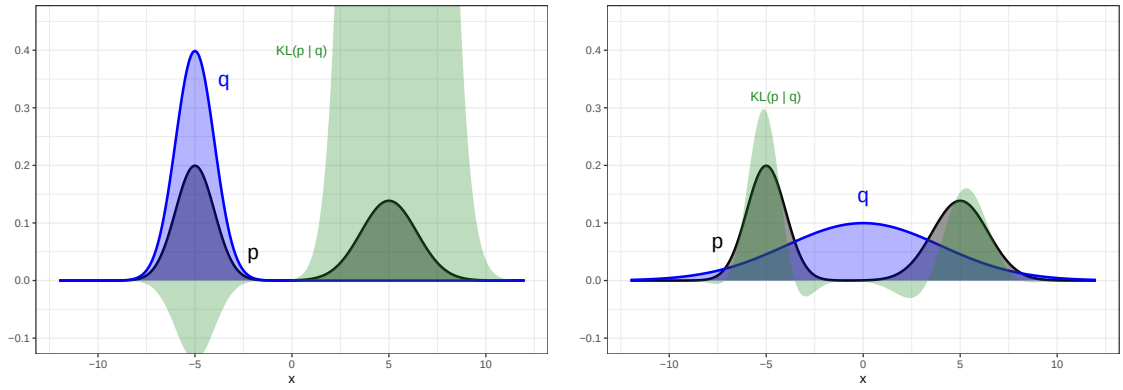


Figure 52: Example of the forward KL divergence when the target posterior is multi-modal. Target posterior is shown in black, and the two panels show two alternative candidate distributions from a Gaussian family (blue). The contribution to the KL divergence is shown by the green region, with the KL figure itself being the area of this region.

the *mode* of the posterior is mimicked with highest importance) or **zero forcing** (because we are often forced to allow $q(\theta) = 0$ in some areas even if the posterior is not).

To see the outcome of this approach visually, we show the same scenario as Figure 52 for the reverse KL in Figure 53. Now the optimal choice under has switched and is the distribution on the left. You can see from the contribution to the KL divergence in green that not placing density where there is density in the posterior is not penalised at all in reverse KL (the right mode is ignored by the approximate distribution in the left panel), but that matching is highly penalised where we do choose to put density.

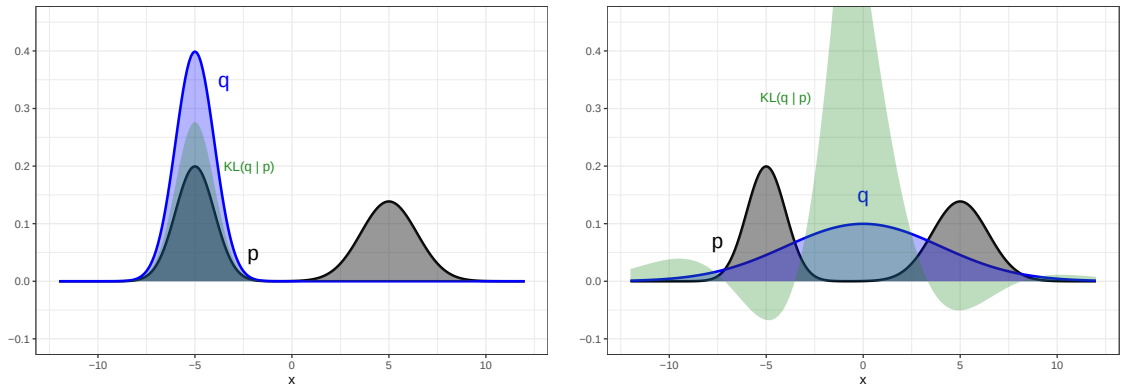


Figure 53: Example of the reverse KL divergence when the target posterior is multi-modal. Target posterior is shown in black, and the two panels show two alternative candidate distributions from a Gaussian family (blue). The contribution to the KL divergence is shown by the green region, with the KL figure itself being the area of this region.

8.4 The variational family of distributions

We have now seen the main concept of variational inference. This involves specifying a family of candidate distributions from which we will choose our approximate distribution from, referred to as $\mathcal{Q}$ in the above. In practice, a decision needs to be made about this. The family of distributions needs to be flexible enough that we believe it will contain distributions able to closely mimic our true posterior, whilst being simple enough that we can calculate the ELBO and maximise it.

8.4.1 Mean-field family

Here we will discuss the current most popular choice in practice, known as the **mean-field variational family**. For a general distribution $q(\theta) \in \mathcal{Q}$, we assume that the dimensions of θ are mutually independent and therefore the density of a general distribution can be simplified as

$$q(\theta) = \prod_{i=1}^{m} q_i(\theta_i),$$

where $\theta = (\theta_1, \dots, \theta_m)$. Each $q_i(\cdot)$ can be specified as generally as needed, dependent upon the parameter it is describing, i.e. we might have a two-dimensional parameter set and assume that q_1 comes from the family of Gaussians and q_2 comes from the family of gammas.

8.4.2 Correlation cannot be replicated

The mean-field family cannot capture correlations between parameters because our family of candidate distributions are defined to have independent components. As an example, see Figure 54, which shows a two-dimensional true posterior in blue with high correlation. The best approximation from a mean-field family will take the form of that in green. As the name of the family suggests, the approximation picks out moments of the underlying distribution, so that the estimate of the mean in both dimensions will be a good approximation to the true posterior.

The correlation cannot be replicated as the candidate family enforces independence. Further, note that the consequence of this is that the variance in the marginals is under-estimated significantly—this was noted at the start of this chapter that variational inference commonly suffers from this problem and this is main reason for this.

8.4.3 Why mean-field is useful

So if the mean-field family means we cannot reproduce correlations between parameters, why is it used? Variational inference involves us being able to maximise the ELBO and finding the $q \in \mathcal{Q}$ which does this. Recall that the ELBO is an expectation, so that

$$ELBO(q) = \mathbb{E}_q \left[\log \left(\frac{p(x, \theta)}{q(\theta)} \right) \right] = \int_{\Theta} q(\theta) \log \left(\frac{p(x, \theta)}{q(\theta)} \right) d\theta.$$

If θ is multi-dimensional, so that $\theta = (\theta_1, \dots, \theta_m)$ then this integral is explicitly

$$\int_{\Theta_1} \dots \int_{\Theta_m} q(\theta_1, \dots, \theta_m) \log \left(\frac{p(x, \theta_1, \dots, \theta_m)}{q(\theta_1, \dots, \theta_m)} \right) d\theta_1 \dots d\theta_m,$$

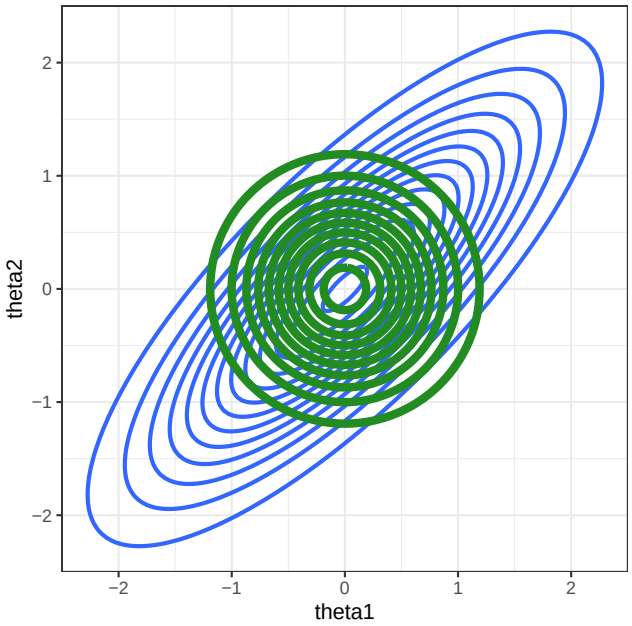


Figure 54: Example of how correlations between parameters are handled using a mean-field approximation in variational inference. Here we have a two-dimensional parameter set, and if the true distribution was highly correlated (blue), this cannot be captured by the optimal mean-field approximation (green).

which is going to very easily enter the realm of being intractable, which is what we need to avoid here! This is the reason that the mean-field approximation is commonly used in practice, the independence between dimensions means that this integral can be simplified significantly into a product of integrals.

Remember: variational inference is a trade-off where we are having to make simplifications (and thus loss of precision) in exchange for speed.

9 Coordinate ascent variational inference (CAVI)

Aims of this section
1. Present one method for implementing variational inference: the CAVI algorithm.
2. Demonstrate its application in a simple mixture modelling example.

9.1 The CAVI algorithm

Given a mean-field family of approximate distributions, we can use the CAVI algorithm to maximise the ELBO. For an m-dimensional parameter θ, the algorithm is as follows.

1. Choose some initial mean-field approximation
$$q(\theta) = \prod_{i=1}^{m} q_i(\theta_i).$$
2. Update each q_i in turn for $i = 1, \dots, m$. Holding $q_j(\theta_j)$ fixed for all $j \neq i$, replace the current q_i in the mean-field approximation with
$$q_i(\theta_i)^* \propto \exp\left\{\mathbb{E}_{q_{-i}}\left[\log p\left(\theta_i, \theta_{-i}, x\right)\right]\right\}. \tag{71}$$
3. Repeat step 2 until the ELBO has converged.

The update for q_i in equation (71) is optimal, in that it gives the greatest increase in the ELBO. To see this, we consider the ELBO as a function of q_i only and write

$$ELBO(q_i) = \mathbb{E}_{q_i}[\mathbb{E}_{q_{-i}}[\log p(\theta_i, \theta_{-i}, x)]] - \mathbb{E}_{q_i}[\log q_i(\theta_i)] + K,$$

for some constant K. This hold because of the independence in the mean-field approximation. Then, we note that

$$\begin{aligned} -KL(q_i||q_i^*) &= -\mathbb{E}_{q_i}[\log q_i(\theta_i) - \log q_i^*(\theta_i)] &(72)\\ &= -\mathbb{E}_{q_i}[\log q_i(\theta_i)] + \mathbb{E}_{q_i}[\mathbb{E}_{q_{-i}}[\log p(\theta_i, \theta_{-i}, x)]] + C, &(73)\end{aligned}$$

for some constant C, so that $ELBO(q_i)$ is equal to $-KL(q_i||q_i^*)$ plus some constant term. As $KL(q_i||q_i^*)$ is minimised by setting $q_i = q_i^*$, this must maximise the ELBO as a function of q_i.

Note there is some similarity with Gibbs sampling here. In Gibbs sampling, we iterate through the conditional distributions, drawing samples at each time. In CAVI, we again iterate through the dimensions, but we are now *updating the distribution* at each iteration, rather than sampling.

9.2 Example: Mixture of Gaussians

9.2.1 The observation model

Consider K mixtures of Gaussians, where each has a different mean μ_i but a common variance of 1. We have n observations $x_1, \dots, x_n$, where each observation is distributed according to one of the

K mixture components, so that

$$x_i \sim N(\mu_{c_i}, 1),$$

where c_i is the mixture assignment of observation i. Note that we do not observe the c_i, and $c_i \in \{1, \dots, K\}$. This model therefore has the set of unknown parameters $\theta = \{\mu, \mathbf{c}\}$: the K class means and the n class assignments. Note that we can also refer to the class assignments as $\mathbf{C}_i$, associated with observation x_i, which is a vector of length K, where all entries are 0 except the c_i^{th}, which is 1. This alternative treatment seems overly complex, but it will turn out to be useful for some algebra later.

Note that one might think of the class probabilities/proportions as being included in the uncertain quantities of interest: the uncertain population proportion belonging to class i, for $i = 1, \dots, K$. But this is *not* the focus here: our interest is in the uncertain class assignments c_i for each *observation*.

9.2.2 The prior

We assume that there is a global distribution that these component means independently arise from, so that the prior for the class means is

$$\mu_i \sim N(0, \sigma^2),$$

for $i = 1, \dots, K$. We will assume that we know the hyperparameter σ^2. In the following we will keep this hyperparameter general, and remember that it is a known constant.

The class assignment c_i of observation i is a categorical variable, taking one of the values $1, \dots, K$. We assume a prior for this which is a categorical distribution with equal weightings, i.e. $(1/K, \dots, 1/K)$.

9.2.3 The joint likelihood

This is a classic example for variational inference because the expression for $p(\mathbf{x})$ is computationally burdensome, being exponential in the number of mixture components, K. We can express the joint density as

$$\log p(\theta, \mathbf{x}) = \sum_{k=1}^{K} \log p(\mu_k) + \sum_{i=1}^{n} \left[\log p(c_i) + \log p(x_i | c_i, \mu_i)\right],$$

where:

- $p(\mu_k)$ is the Gaussian prior, where for each k they are identically distributed, having mean 0 and known variance σ^2. We can write $\log p(\mu_k) \propto -\frac{\mu_k^2}{2\sigma^2}$.
- $p(c_i)$ is a categorical distribution describing the weightings/probability of each mixture component arising (all equal at $1/K$). It is easiest here to work with the vector form of this variable, $\mathbf{C}_i$. Using our prior of equal weightings, this means $p(\mathbf{C}_i) = (\frac{1}{K}, \dots, \frac{1}{K})$ for all i and $\log p(C_{ik}) = -\log K$. This is a constant with respect to our parameters of interest and so, as we are only interested in likelihoods up to proportionality when considering gradients, we can drop this contribution to the joint likelihood.

- $p(x_i|c_i, \mu_i)$ is the Gaussian with mean μ_{c_i} and variance 1. We are again going to make use of our vector $\mathbf{C}_i$ here, as although it is intuitive to think of the mean as μ_{c_i}, it's tricky to work with this. We have

$$\log p(x_i|c_i, \mu_i) = \log \prod_{k=1}^{K} p(x_i|\mu_k)^{C_{ik}} = \sum_{k=1}^{K} C_{ik} \log p(x_i|\mu_k) \propto -\frac{1}{2} \sum_{k=1}^{K} C_{ik}(x_i - \mu_k)^2.$$

Combining all of the above, we can write

$$\log p(\theta, \mathbf{x}) \propto -\frac{1}{2\sigma^2} \sum_{k=1}^{K} \mu_k^2 - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{K} C_{ik}(x_i - \mu_k)^2. \tag{74}$$

9.2.4 The mean-field family approximation

If we are to assume a mean-field approximation to this system, we need a family of distributions

$$q(\theta) = \prod_{k=1}^{K} q(\mu_k|m_k, s_k^2) \prod_{i=1}^{n} q(\mathbf{C}_i|\phi_i), \tag{75}$$

as we assume that our parameters are independent from one another.

We will assume that the class means μ_k have Gaussian distributions, and so there are variational parameters m_k and s_k^2 that describe the range of Gaussians within the variational family. It is common practice to assume that the approximate distribution of a continuous variable is Gaussian.

We will then assume a categorical distribution for the i^{th} observation's class, which has a vector of K component weights/probabilities given by ϕ_i.

Again, note that we have one set of variational parameters (m_k, s_k^2) per mixture component, and one set of variational parameters ϕ_i *per observation*, because we're interested in the class membership of each observation here.

We can express our variational family as

$$\log q(\theta) \propto \sum_{i=1}^{n} \sum_{k=1}^{K} \log \phi_{ik} - \frac{1}{2} \sum_{k=1}^{K} \left(\log(2\pi s_k^2) + \frac{(\mu_k - m_k)^2}{s_k^2} \right). \tag{76}$$

You may notice that we have assumed the same functional form for our approximate distribution for $\theta|\mathbf{x}$ as we did for our prior θ. This isn't mandatory for variational inference, but is common. The variational parameter ϕ_i features in the prior as $(\frac{1}{K}, \dots, \frac{1}{K})$, and the parameters m_k and s_k^2 feature as 0 and σ^2, respectively, for all $k \in 1, \dots, K$. It's useful to compare the prior to posterior as part of our analysis.

9.2.5 CAVI

In this example, our $q(\cdot) \in \mathcal{Q}$ is defined by the values of $\mathbf{m}, \mathbf{s}^2, \phi_i$, so we are optimising the ELBO with respect to this set of *variational parameters*. An iteration of the optimisation process will involve updating each of $m_1, \dots, m_K, s_1^2, \dots, s_K^2, \phi_1, \dots, \phi_n$ in turn. This sounds like a lot, but we

will see the update process of that in Equation (71) for each of m_k, s_k^2, ϕ_i simplifies down to a set of neat, short equations.

We have

$$ELBO(q) = \mathbb{E}_q\left[\log p(\theta, \mathbf{x})\right] - \mathbb{E}_q\left[\log q(\theta)\right] \tag{77}$$

$$\propto -\frac{1}{2\sigma^2}\sum_{k=1}^{K}\mathbb{E}_q\left[\mu_k^2\right] - \frac{1}{2}\sum_{i=1}^{n}\mathbb{E}_q\left[\sum_{k=1}^{K}C_{ik}(x_i-\mu_k)^2\right] \tag{78}$$

$$-\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{E}_q\left[\log\phi_{ik}\right] + \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}_q\left[\log(2\pi s_k^2) + \frac{(\mu_k-m_k)^2}{s_k^2}\right], \tag{79}$$

We are going to derive in detail the update for ϕ_i. We will then only give results for the update for m_k, s_k^2; you can derive these results for yourself if you wish.

9.2.5.1 The CAVI update for the mixture class assignment The variational family for each class assignment $\mathbf{C}_i$ is a categorical distribution of the form

$$q(\mathbf{C}_i|\phi_i) = \prod_{j=1}^{K}\phi_{ij}^{C_{ij}},$$

so that

$$\log q(\mathbf{C}_i|\phi_i) = \sum_{i=1}^{K}C_{ij}\log\phi_{ij} \tag{80}$$

The CAVI update for ϕ_i means that we are updating just part of our variational family. Recall that we implement CAVI parameter by parameter because of our independence assumption. So our full variational distribution is that in Equation (75), and our update for ϕ_i means we are updating the part of the variational distribution $q(\mathbf{C}_i|\phi_i)$. From Equation (71) we need to set

$$q(\mathbf{C}_i|\phi_i)^* \propto \exp\left\{\mathbb{E}_{-\phi_i}\left[\log p\left(\phi_i, \theta_{-\phi_i}, \mathbf{x}\right)\right]\right\}.$$

As this is a function of ϕ_i, we only need to take all the elements in our expression for $p\left(\phi_i, \theta_{-\phi_i}, \mathbf{x}\right)$ in Equation (74) that involve ϕ_i (i.e. we only consider up to proportionality). This is therefore

$$q(\mathbf{C}_i|\phi_i)^* \propto \exp\left\{\mathbb{E}_{-\phi_i}\left[-\frac{1}{2}\sum_{k=1}^{K}C_{ik}(x_i-\mu_k)^2\right]\right\} \tag{81}$$

$$= \exp\left\{-\frac{1}{2}\sum_{k=1}^{K}C_{ik}\mathbb{E}_{-\phi_i}\left[x_i^2 - 2x_i\mu_k + \mu_k^2\right]\right\} \tag{82}$$

$$\propto \exp\left\{\sum_{k=1}^{K}C_{ik}\left(x_i\mathbb{E}_{-\phi_i}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-\phi_i}\left[\mu_k^2\right]\right)\right\} \tag{83}$$

$$= \exp\left\{\sum_{k=1}^{K}C_{ik}\left(x_i m_k - \frac{m_k^2+s_k^2}{2}\right)\right\}. \tag{84}$$

We have simplified this expression as much as possible, but where does that leave us? Now we compare the expression above, with Equation (80) , and we set

$$\phi_{ij} \propto \exp\left\{x_i m_k - \frac{m_k^2 + s_k^2}{2}\right\}.$$

9.2.5.2 The complete CAVI update In summary, in an iterated update of the CAVI approach, we replace the current set of $\phi_1, \ldots, \phi_n$ with the above expression, where m, s^2 are the current values of these parameters. A complete iteration of CAVI will then involve updating $m_1, \ldots, m_K$ given the current values of the remaining parameters, and then updating $s_1^2, \ldots, s_K^2$ given the current values of the remaining parameters. We will not show the derivation, but these updates are:

$$m_k = \frac{\sum_{i=1}^n \phi_{ik} x_i}{\frac{1}{\sigma^2} + \sum_{i=1}^n \phi_{ik}}, \tag{85}$$

$$s_k^2 = \frac{1}{\frac{1}{\sigma^2} + \sum_{i=1}^n \phi_{ik}}. \tag{86}$$

9.2.6 Implementing CAVI in R

We show CAVI in practice for an example of a mixture of 3 Gaussians. We generate some observations:

```
# create observations
true_mu <- c(-2,0,3)
sigma <- 1
n <- 1000
K <- 3
true_c <- sample(1:K, n, replace = T)
x <- rnorm(n, true_mu[true_c], 1)
```

Then run the CAVI algorithm:

```
n_iter <- 1000
stored_m <- matrix(NA, nrow = n_iter, ncol = K)
# initial values
m <- rnorm(K)
s <- rep(1, K)
phi <- matrix(NA, nrow = n_iter, ncol = K)

# CAVI
for (j in 1:n_iter) {
  for(i in 1:n) {
    phi[i, ] <- exp(m*x[i] - (m^2+s)/2)
    # normalise phi as they are supposed to sum to 1
    phi[i, ] <- phi[i, ] / sqrt(sum(phi[i, ])^2)
```

```
  }
  for(k in 1:K) {
    m[k] <- sum(phi[ , k] * x) / (1/sigma^2 + sum(phi[ , k]))
    s[k] <- 1 / (1/sigma^2 + sum(phi[ , k]))
  }
  # keep track of the m's each iteration to see convergence
  stored_m[j, ] <- m
}
```

Results of this analysis are plotted in Figures 55 and 56. Remember that the outcome of CAVI is a set of optimal values for m_k, s_k^2, ϕ_i, and these optimal values define the joint distribution of θ that is used as an approximation to the posterior.

We have observations x that arise from one of 3 Gaussians, and the true data is shown in the left panel of Figure 55, showing the general range of x values produced from each group classification. In the right panel of Figure 55 we show the most probable group classification, defined for each observation x_i as the most probable class from $\phi_i = (\phi_{i1}, \phi_{i2}, \phi_{i3})$. This is a fairly easy classification problem, as you can see that the observations from the three groups are fairly well separated, with only a small overlap. The VI classification has clear cut boundaries between the three groups when we look at the most probable classifications, but note that if we were to delve into the ϕ_i for an observation that is on the boundary between groupings (i.e. an outlier in the box plots in the left panel), there will likely be uncertainty in the predicted classification. For example, observation 13 had value $x_{13} = -0.848$ and was simulated from group 1. This value is in the tail of that Gaussian distribution, and the estimate of the class probabilities was $(0.457, 0.542, 6.1 \times 10^{-4})$, so although it was deemed most likely to have arisen from group 2 (incorrectly), we can see it was almost equally deemed to be from groups 1 or 2.

We can also produce a confusion matrix with the `caret` package. We have the true classes in `true_c`, and we extract the predictions as follows:

```
most_lik <- rep(NA, n)
for(i in 1:n) {
  most_lik[i] <- which(phi[i, ] == max(phi[i, ]))
}
```

Then we do

```
caret::confusionMatrix(as.factor(true_c),
                       as.factor(most_lik))$table
```

```
##           Reference
## Prediction   1   2   3
##          1 254   1  56
##          2  52  26 264
##          3   0 328  19
```

In Figure 56 we show the results of the group means, the μ_1, μ_2, μ_3. The distributions are the results from CAVI. Recall that CAVI gives the optimal m_k and s_k^2 that define the approximate distribution of independent Gaussians for the μ_k. The true mixture means are shown with dashed

lines, and we can see these have been replicated well by this analysis. Something to note it that this approach cannot include correlations between the parameters. If the distribution for μ_1 was to be lowered, then this would obviously affect the group classification probabilities ϕ_i, i.e. there is correlation in our model system that we expect to exist that we cannot estimate. This is important to always bear in mind with VI.

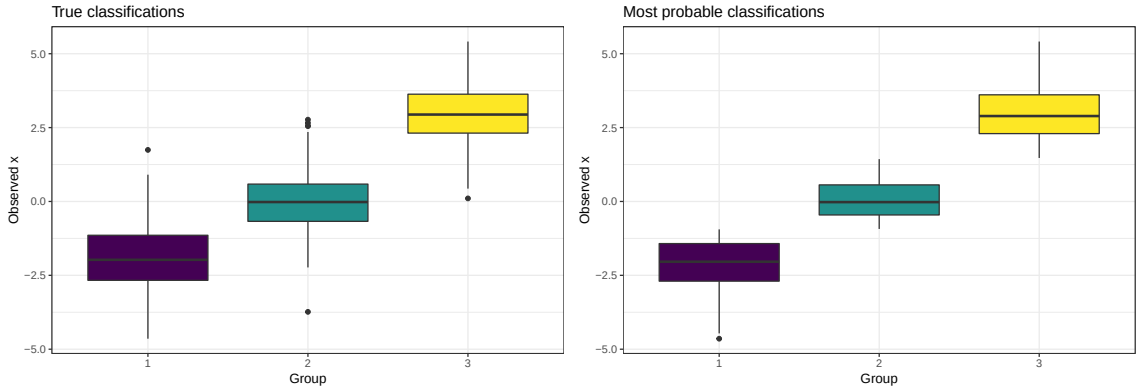


Figure 55: The distribution of observations, split by the true underlying mixture group classification (left panel) and the most probable classification from applying CAVI (right panel).

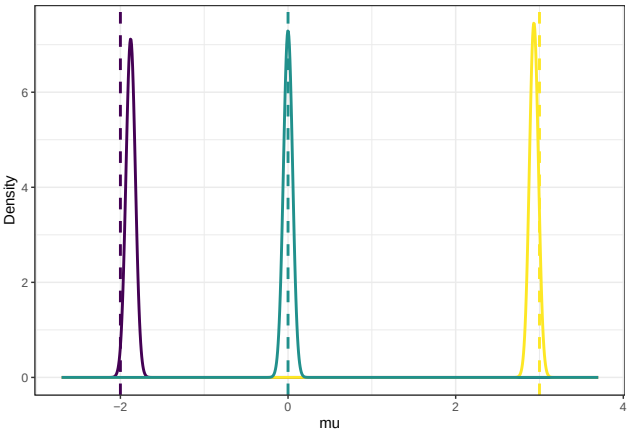


Figure 56: The approximate posterior distributions of the mixture means from CAVI (solid), along with the true means used for the simulation (dashed).

9.3 Comment

CAVI was useful for the example, because it was possible to derive closed-form expressions for the variational distribution update equations. Note the similarity with Gibbs sampling: this is useful when the full conditional distributions are relatively easy to sample from.

References

Bates, D., Mächler, M., Bolker, B., and Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Gabry, J. (2018). *shinystan: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models*. R package version 2.5.0.

Gabry, J. and Mahr, T. (2021). bayesplot: Plotting for bayesian models. R package version 1.8.1.

Gamer, M., Lemon, J., and <puspendra.pusp22@gmail.com>, I. F. P. S. (2019). *irr: Various Coefficients of Interrater Reliability and Agreement*. R package version 0.84.1.

Horst, A. M., Hill, A. P., and Gorman, K. B. (2020). *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*. R package version 0.1.0.

i Marín, X. F. (2016). ggmcmc: Analysis of MCMC samples and Bayesian inference. *Journal of Statistical Software*, 70(9):1–20.

Kuhn, M. (2021). *caret: Classification and Regression Training*. R package version 6.0-90.

Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., and R Core Team (2021). *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-153.

Stan Development Team (2021). RStan: the R interface to Stan. R package version 2.21.3.

van Buuren, S. and Groothuis-Oudshoorn, K. (2011). mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.