MATH5835M Statistical Computing

Matthew Aldridge

2025-09-30

Table of contents

About MATH5835

Organisation of MATH5835

This module is **MATH5835M Statistical Computing**.

This module lasts for 11 weeks from 29 September to 12 December 2025. The exam will take place sometime between 12 and 23 January 2026.

The module leader, the lecturer, and the main author of these notes is Dr Matthew Aldridge. (You can call me "Matt", "Matthew", or "Dr Aldridge", pronounced "*old*-ridge".) My email address is m.aldridge@leeds.ac.uk, although I much prefer questions in person at office hours (see below) rather than by email.

The HTML webpage is the best way to view the course material. There is also a **PDF version**, although I have been much less careful about the presentation of this material, and it does not include the problem sheets.

Lectures

The main way you will learn new material for this module is by attending lectures. There are three lectures per week:

- Mondays at 1400
- Thursdays at 1200
- Fridays at 1000

all in in Roger Stevens LT 14.

I recommend taking your own notes during the lecture. I will put brief summary notes from the lectures on this website, but they will not reflect all the details I say out loud and write on the whiteboard. Lectures will go through material quite quickly and the material may be quite difficult, so it's likely you'll want to spend time reading through your notes after the lecture. Lectures should be recorded on the lecture capture system; I find it very difficult to read the whiteboard in these videos, but if you unavoidably miss a lecture, for example due to illness, you may find they are better than nothing.

In Weeks 3, 5, 7, 9 and 11, the Thursday lecture will operate as a "problems class" – see more on this below.

Attendance at lectures in compulsory. You should record your attendance using the UniLeeds app and the QR code on the wall in the 15 minutes before the lecture or the 15 minutes after the lecture (but not during the lecture).

Problem sheets and problem classes

Mathematics and statistics are "doing" subjects! To help you learn material for the module and to help you prepare for the exam, I will provide 5 unassessed problem sheets. These are for you to work through in your own time to help you learn; they are *not* formally assessed. You are welcome to discuss work on the problem sheets with colleagues and friends, although my recommendation would be to write-up your "last, best" attempt neatly by yourself.

There will be an optional opportunity to submit one or two questions from the problem sheet to me in advance of the problems class for some brief informal feedback on your work. See the problem sheets for details.

You should work through each problem sheet in preparation for the problems class in the Thursday lecture of Week 3, 5, 7, 9 and 11. In the problems class, you should be ready to explain your answers to questions you managed to solve, discuss your progress on questions you partially solved, and ask for help on questions you got stuck on.

You can also ask for extra help or feedback at office hours (see below).

Coursework

There will be one piece of assessed coursework, which will make up 20% of your module mark. You can read more about the coursework here.

The coursework will be in the form of a worksheet. The worksheet will have some questions, mostly computational but also mathematical, and you will have to write a report containing your answers and computations.

The assessed coursework will be introduced in the **computer practical** sessions in Week 9.

The deadline for the coursework will be the penultimate day of the Autumn term, **Thursday 12 December** at 1400. Feedback and marks will be returned on Monday 13 January, the first day of the Spring term.

Office hours

I will run a n optional "office hours" drop-in session each week for feedback and consultation. You can come along if you want to talk to me about anything on the module, including if you'd like more feedback on your attempts at problem sheet questions. (For extremely short queries, you can approach me before or after lectures, but my response will often be: "Come to my office hours, and we can discuss it there!")

Office hours will happen on **Thursdays from 1300 to 1400** – so directly after the Thursday lecture / problems class – in my office, which is **EC Stoner 9.10n** in "Maths Research Deck" area on the 9th floor of the EC Stoner building. (One way to the Maths Research Deck is via the doors directly opposite the main entrance to the School of Mathematics; you can also get there from Staircase 1 on the Level 10 "red route" through EC Stoner, next to the Maths Satellite.) If you cannot make this time, contact me for an alternative arrangement.

Exam

There will be one exam, which will make up 80% of your module mark.

The exam will be in the January 2026 exam period (12–23 January); the date and time will be announced in December. The exam will be in person and on campus.

The exam will last 2 hours and 30 minutes. The exam will consist of 4 questions, all compulsory. You will be allowed to use a permitted calculator in the exam.

Content of MATH5835

Necessary background

I recommend that students should have completed at least two undergraduate level courses in probability or statistics – although confidence and proficiency in basic material is more important than very deep knowledge of more complicated topics.

For Leeds undergraduates, MATH2715 Statistical Methods is an official prerequisite (please get in touch with me if you are/were a Leeds undergraduate and have not taken MATH2715), although confidence and proficiency in the more basic material of MATH1710 & MATH1712 Probability and Statistics 1 & 2 is probably more important.

Some knowledge I will assume:

- **Probability:** Basic rules of probability; random variables, both continuous and discrete; "famous" distributions (especially the normal distribution and the continuous uniform distribution); expectation, variance, covariance, correlation; law of large numbers and central limit theorem.
- **Statistics:** Estimation of parameters; bias and error; sample mean and sample variance

This module will also include an material on Markov chains. I won't assume any pre-existing knowledge of this, and I will introduce all new material we need, but students who have studied Markov chains before (for example in the Leeds module MATH2750 Introduction to Markov Processes) may find a couple of lectures here are merely a reminder of things they already know.

The lectures will include examples using the **R** program language. The coursework and problem sheets will require use of R. The exam, while just a "pencil and paper" exam, will require understanding and writing short portions of R code. We will assume basic R capability – that you can enter R commands, store R objects using the `<-` assignment, and perform basic arithmetic with numbers and vectors. Other concepts will be introduced as necessary. If you want to use R on your own device, I recommend downloading (if you have not already) the R programming language and the program RStudio. (These lecture notes were written in R using RStudio.)

Syllabus

We plan to cover the following topics in the module:

- **Monte Carlo estimation:** definition and examples; bias and error; variance reduction techniques: control variates, antithetic variables, importance sampling. [9 lectures]
- **Random number generation:** pseudo-random number generation using linear congruential generators; inverse transform method; rejection sampling [7 lectures]
- **Markov chain Monte Carlo** (MCMC)**:** [7 lectures]
  - Introduction to Markov chains in discrete and continuous space
  - Metropolis–Hastings algorithm: definition; examples; MCMC in practice; MCMC for Bayesian statistics
- **Resampling methods:** Empirical distribution; plug-in estimation; bootstrap statistics; bootstrap estimation [4 lectures]
- Frequently-asked questions [1 lecture]

Together with the 5 problems classes, this makes 33 lectures.

Book

The following book is strongly recommended for the module:

- J Voss, *An Introduction to Statistical Computing: A simulation-based approach*, Wiley Series in Computational Statistics, Wiley, 2014

The library has electronic access to this book (and two paper copies).

Dr Voss is a lecturer in the School of Mathematics and the University of Leeds, and has taught MATH5835 many times. *An Introduction to Statistical Computing* grew out of his lecture notes for this module, so the book is ideally suited for this module. My lectures will follow this book closely – specifically:

- Monte Carlo estimation: Sections 3.1–3.3
- Random number generation: Sections 1.1–1.4
- Markov chain Monte Carlo: Section 2.3 and Sections 4.1–4.3
- Bootstrap: Section 5.2

For a second look at material, for preparatory reading, for optional extended reading, or for extra exercises, this book comes with my highest recommendation!

Part I

Monte Carlo estimation

1 Introduction to Monte Carlo

Today, we'll start the first main topic of the module, which is called "Monte Carlo estimation". But first, a bit about the subject as a whole.

1.1 What is statistical computing?

"Statistical computing" – or "computational statistics" – refers to the branch of statistics that involves not attacking statistical problems merely with a pencil and paper, but rather by combining human ingenuity with the immense calculating powers of computers.

One of the big ideas here is **simulation**. Simulation is the idea that we can understand the properties of a random model not by cleverly working out the properties using theory – this is usually impossible for anything but the simplest "toy models" – but rather by running the model many times on a computer. From these many simulations, we can observe and measure things like the typical (or "expected") behaviour, the spread (or "variance") of the behaviour, and other things. This concept of simulation is at the heart of the module MATH5835M Statistical Computing.

In particular, we will look at **Monte Carlo** estimation. Monte Carlo is about estimating a parameter, expectation or probability related to a random variable by taking many samples of that random variable, then computing a relevant sample mean from those samples. We will study Monte Carlo in its standard "basic" form, then look at ways we can make Monte Carlo estimation more accurate (Lectures 1–9).

To run a simulation – for example, when performing Monte Carlo estimation – one needs random numbers with the correct distribution. **Random number generation** (Lectures 10–16) will be an important part of this module. We will look first at how to generate randomness of any sort, and then how to manipulate that randomness into the shape of the distributions we want.

Sometimes, it's not possible to generate perfectly independent samples from exactly the distribution you want. But we can use the output of a process called a "Markov chain" to get "fairly independent" samples from nearly the distribution we want. When we perform Monte Carlo estimation with the output of a Markov chain, this is called **Markov chain Monte Carlo**

(MCMC) (Lectures 17–23). MCMC has become a vital part of modern Bayesian statistical analysis.

The final section of the module is about dealing with data. Choosing a random piece of data from a given dataset is a lot like generating a random number from a given distribution, and similar Monte Carlo estimation ideas can be used to find out about that data. We think of a dataset as being a sample from a population, and sampling again from that dataset is known as **resampling** (Lecture 24–27). The most important method of finding out about a population by using resampling from a dataset is called the "bootstrap", and we will study the bootstrap in detail.

MATH5835M Statistical Computing is a *mathematics* module that will concentrate on the *mathematical* ideas that underpin statistical computing. It is not a programming module that will go deeply into the practical issues of the most efficient possible coding of the algorithms we study. But we will want to investigate the behaviour of the methods we learn about and to explore their properties, so will be computer programming to help us do that. We will be using the statistical programming language R, (although one could just as easily have used Python or other similar languages). As my PhD supervisor often told me: "You don't really understand a mathematical algorithm until you've coded it up yourself."

1.2 What is Monte Carlo estimation?

Let X be a random variable. We recall the **expectation** $\mathbb{E}X$ of X. If X is discrete with probability mass function (PMF) p, then the expectation of X is

$$\mathbb{E}X = \sum_x x\, p(x);$$

while if X is continuous with probability density function (PDF) f, then the expectation is

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

More generally, the expectation of a function ϕ of X is

$$\mathbb{E}\,\phi(X) = \begin{cases} \displaystyle\sum_x \phi(x)\, p(x) & \text{for } X \text{ discrete} \\ \displaystyle\int_{-\infty}^{+\infty} \phi(x)\, f(x)\, \mathrm{d}x & \text{for } X \text{ continuous.} \end{cases}$$

(This matches with the "plain" expectation when $\phi(x) = x$.)

But how do we actually *calculate* an expectation like one of these? If X is discrete and can only take a small, finite number of values, then we can simply add up the sum $\sum_x \phi(x)\, p(x)$. But otherwise, we just have to hope that ϕ and p or f are sufficiently "nice" that we can

manage to work out the sum/integral using a pencil and paper (and our brain). But while this is often possible in the sort of "toy example" one comes across in maths or statistics lectures, this is very rare in "real life" problems.

Monte Carlo estimation is the idea that we can get an approximate answer for $\mathbb{E}X$ or $\mathbb{E}\,\phi(X)$ if we have access to lots of samples from X. If we have access to $X_1, X_2 \ldots, X_n$, independent and identically distributed (IID) samples with the same distribution as X, then we already know that the mean

$$\overline{X} = \frac{1}{n}(X_1 + X_2 + \cdots + X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i$$

can be used to estimate the expectation $\mathbb{E}X$. We know that $\overline{X}$ is usually close to the expectation $\mathbb{E}X$, at least if if the number of samples n is large; this is justified by the "law of large numbers", which says that $\overline{X} \to \mathbb{E}X$ as $n \to \infty$.

Similarly, we can use

$$\frac{1}{n}(\phi(X_1) + \phi(X_2) + \cdots + \phi(X_n)) = \frac{1}{n}\sum_{i=1}^{n} \phi(X_i)$$

to estimate $\mathbb{E}\,\phi(X)$. The law of large numbers again says that this estimate tends to the correct value $\mathbb{E}\,\phi(X)$ as $n \to \infty$.

In this module we will write that $X_1, X_2, \ldots, X_n$ is a "**random sample** from X" to mean that $X_1, X_2, \ldots, X_n$ are IID with the same distribution as X.

Definition 1.1. Let X be a random variable, ϕ a function, and write $\theta = \mathbb{E}\,\phi(X)$. Then the **Monte Carlo estimator** $\hat{\theta}_n^{\mathrm{MC}}$ of θ is

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} \phi(X_i),$$

where $X_1, X_2, \ldots, X_n$ are a random sample from X.

While general ideas for estimating using simulation go back a long time, the modern theory of Monte Carlo estimation was developed by the physicists Stanislaw Ulam and John von Neumann. Ulam (who was Polish) and von Neumann (who was Hungarian) moved to the US in the early 1940s to work on the Manhattan project to build the atomic bomb (as made famous by the film *Oppenheimer*). Later in the 1940s, they worked together in the Los Alamos National Laboratory continuing their research on nuclear physics generally and nuclear weapons more specifically, where they used simulations on early computers to help them numerically solve difficult mathematical and physical problems.

The name "Monte Carlo" was chosen because the use of randomness to solve such problems reminded them of gamblers in the casinos of Monte Carlo, Monaco. Ulam and von Neumann also worked closely with another colleague Nicholas Metropolis, whose work we will study later in this module.

1.3 Examples

Let's see some simple examples of Monte Carlo estimation using R.

Example 1.1. Let's suppose we've forgotten the expectation of the exponential distribution $X \sim \text{Exp}(2)$ with rate 2. In this simple case, we could work out the answer using the PDF $f(x) = 2\text{e}^{-2x}$ as

$$\mathbb{E}X = \int_0^\infty x\, 2\text{e}^{-2x}\, \text{d}x$$

and, without too much difficulty, get the answer $\frac{1}{2}$. But instead, let's do this the Monte Carlo way.

In R, we can use the `rexp()` function to get IID samples from the exponential distribution: the full syntax is `rexp(n, rate)`, which gives `n` samples from an exponential distribution with rate `rate`. The following code takes the mean of $n = 100$ samples from the exponential distribution.

```
n <- 100
samples <- rexp(n, 2)
MCest <- (1 / n) * sum(samples)
MCest
```

```
[1] 0.4594331
```

So our Monte Carlo estimate is 0.4594, to 4 decimal places.

That's fairly close to the correct answer of $\frac{1}{2}$. But we should (hopefully) be able to get a more accurate estimation if we use more samples. We could also simplify the third line of our code by using the `mean()` function.

```
n <- 1e6
samples <- rexp(n, 2)
MCest <- mean(samples)
MCest
```

```
[1] 0.4996347
```

In the second line, `1e6` is R code for the scientific notation 1×10^6, or a million. I just picked this as "a big number, but where my code still only took a few seconds to run."

Our new Monte Carlo estimate is 0.4996, which is (probably) much closer to the true value of $\frac{1}{2}$.

By the way: all R code "chunks" displayed in the notes should work perfectly if you copy-and-paste them into RStudio. (Indeed, when I compile these lecture notes in RStudio, all the R code gets run on my computer – so I'm certain in must work correctly!) If you hover over a code chunk, a little "clipboard" icon should appear in the top-right, and clicking on that will copy it so you can paste it into RStudio. I strongly encourage playing about with the code as a good way to learn this material and explore further!

Example 1.2. Let's try another example. Let $X \sim \mathrm{N}(1, 2^2)$ be a normal distribution with mean 1 and standard deviation 2. Suppose we want to find out $\mathbb{E}(\sin X)$ (for some reason). While it *might* be possible to somehow calculate the integral

$$\mathbb{E}(\sin X) = \int_{-\infty}^{+\infty} (\sin x) \, \frac{1}{\sqrt{2\pi \times 2^2}} \exp\left(-\frac{(x-1)^2}{2 \times 2^2}\right) \, \mathrm{d}x,$$

that looks extremely difficult to me.

Instead, a Monte Carlo estimation of $\mathbb{E}(\sin X)$ is very straightforward: we just take the mean of the sine of a bunch of normally distributed random numbers. That is we get a random samples $X_1, X_2, \ldots, X_n$ from X; then take the mean of the values

$$\sin(X_1), \sin(X_2), \ldots, \sin(X_n).$$

(We must remember, though, when using the `rnorm()` function to generate normally distributed random variates, that the third argument is the *standard deviation*, here 2, *not* the variance, here $2^2 = 4$.)

```
n <- 1e6
samples <- rnorm(n, 1, 2)
MCest <- mean(sin(samples))
MCest
```

```
[1] 0.1131249
```

Our Monte Carlo estimate is 0.11312.

Next time: *We look at more examples of things we can estimate using the Monte Carlo method.*

Summary:

- Statistical computing is about solving statistical problems by combining human ingenuity with computing power.

- The Monte Carlo estimate of $\mathbb{E}\,\phi(X)$ is

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i),$$

  where $X_1, \ldots, X_n$ are IID random samples from X.

- Monte Carlo estimation typically gets more accurate as the number of samples n gets bigger.

Read more: Voss, *An Introduction to Statistical Computing*, Section 3.1 and Subsection 3.2.1.

2 Uses of Monte Carlo

Quick recap: Last time we defined the Monte Carlo estimator for an expectation of a function of a random variable $\theta = \mathbb{E}\,\phi(X)$ to be

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}(\phi(X_1) + \phi(X_2) + \cdots + \phi(X_n)) = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i),$$

where $X_1, X_2, \ldots, X_n$ are independent random samples from X.

Today we look at two other things we can estimate using Monte Carlo simulation: probabilities, and integrals.

2.1 Monte Carlo for probabilities

What if we want to find a *probability*, rather than an expectation? What if we want $\mathbb{P}(X = x)$ for some x, or $\mathbb{P}(X \geq a)$ for some a, or, more generally, $\mathbb{P}(X \in A)$ for some set A?

The key thing that will help us here is the *indicator function*. The indicator function simply tells us whether an outcome x is in a set A or not.

Definition 2.1. Let A be a set. Then the **indicator function** $\mathbb{I}_A$ is defined by

$$\mathbb{I}_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A. \end{cases}$$

The set A could just be a single element $A = \{y\}$. In that case $\mathbb{I}_A(x)$ is 1 if $x = y$ and 0 if $x \neq y$. Or A could be a semi-infinite interval, like $A = [a, \infty)$. In that case $\mathbb{I}_A(x)$ is 1 if $x \geq a$ and 0 if $x < a$.

Why is this helpful? Well $\mathbb{I}_A$ is a function, so let's think about what the expectation $\mathbb{E}\,\mathbb{I}_A(X)$ would be for some random variable X. Since $\mathbb{I}_A$ can only take two values, 0 and 1, we have

$$\begin{aligned}\mathbb{E}\,\mathbb{I}_A(X) &= \sum_{y\in\{0,1\}} y\,\mathbb{P}\big(\mathbb{I}_A(X)=y\big)\\ &= 0\times\mathbb{P}\big(\mathbb{I}_A(X)=0\big)+1\times\mathbb{P}\big(\mathbb{I}_A(X)=1\big)\\ &= 0\times\mathbb{P}(X\notin A)+1\times\mathbb{P}(X\in A)\\ &= \mathbb{P}(X\in A).\end{aligned}$$

P(X A) . \end{align*} In line three, we used that $\mathbb{I}_A(X)=0$ if and only if $X\notin A$, and that $\mathbb{I}_A(X)=1$ if and only if $X\in A$.

So the expectation of an indicator function a set is the probability that X is in that set. This idea connects "expectations of functions" back to probabilities: if we want to find $\mathbb{P}(X\in A)$ we can find the expectation of $\mathbb{I}_A(X)$.

With this idea in hand, how do we estimate $\theta=\mathbb{P}(X\in A)$ using the Monte Carlo method? We write $\theta=\mathbb{E}\,\mathbb{I}_A(X)$. Then our Monte Carlo estimator is

$$\widehat{\theta}_n^{\mathrm{MC}}=\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}_A(X_i).$$

We remember that $\mathbb{I}_A(X_i)$ is 1 if $X_i\in A$ and 0 otherwise. So if we add up n of these, we count an extra +1 each time we have an $X_i\in A$. So $\sum_{i=1}^n\mathbb{I}_A(X_i)$ counts the total number of the X_i that are in A. So the Monte Carlo estimator can be written as

$$\widehat{\theta}_n^{\mathrm{MC}}=\frac{\#\text{ of }X_i\text{ that are in }A}{n}.$$

(I'm using # as shorthand for "the number of".)

Although we've had to do a bit of work to get here, this a totally logical outcome! The right-hand side here is the proportion of the samples for which $X_i\in A$. And if we want to estimate the probability something happens, looking at the proportion of times it happens in a random sample is very much the "intuitive" estimate to take. And that intuitive estimate is indeed the Monte Carlo estimate!

Example 2.1. *Let $Z\sim\mathrm{N}(0,1)$ be a standard normal distribution. Estimate $\mathbb{P}(Z>2)$.*

This is a question that it is impossible to answer exactly using a pencil and paper: there's no closed form for

$$\mathbb{P}(Z>2)=\int_2^\infty\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-z^2/2}\,\mathrm{d}z,$$

so we'll have to use an estimation method.

The Monte Carlo estimate means taking a random sample $Z_1,Z_2,\ldots,Z_n$ of standard normals, and calculating what proportion of them are greater than 2. In R, we can do this as follows.

```
n <- 1e6
samples <- rnorm(n)
MCest <- mean(samples > 2)
MCest
```

```
[1] 0.022716
```

In the second line, we could have written `rnorm(n, 0, 1)`. But, if you don't give the parameters `mean` and `sd` to the function `rnorm()`, R just assumes you want the standard normal with `mean = 0` and `sd = 1`.

We can check our answer: R's inbuilt `pnorm()` function estimates probabilities for the normal distribution (using a method that, in this specific case, is much quicker and more accurate than Monte Carlo estimation). The true answer is very close to

```
pnorm(2, lower.tail = FALSE)
```

```
[1] 0.02275013
```

so our estimate was pretty good.

We should explain the third line in the code we used for the Monte Carlo estimation `mean(samples > 2)`. In R, some statements can be answered "true" or "false": these are often statements involving equality `==` (that's a *double* equals sign) or inequalities like `<`, `<=`, `>=`, `>`, for example. So `5 > 2` is `TRUE` but `3 == 7` is `FALSE`. These can be applied "component by component" to vectors. So, for example, testing which numbers from 1 to 10 are greater than or equal to 7, we get

```
1:10 >= 7
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

six `FALSE`s (for 1 to 6) followed by four `TRUE`s (for 7 to 10).

We can also use `&` ("and") and `|` ("or") in true/false statements like these.

But R also knows to treat `TRUE` like the number 1 and `FALSE` like the number 0. (This is just like the concept of the indicator function we've been discussing.) So if we add up some `TRUE`s and `FALSE`s, R simply counts how many `TRUE`s there are

```
sum(1:10 >= 7)
```

```
[1] 4
```

So in our Monte Carlo estimation code, `samples > 2` was a vector of `TRUE`s and `FALSE`s, depending on whether each sample was greater than 2 or not, then `mean(samples > 2)` took the *proportion* of the samples that were greater than 2.

2.2 Monte Carlo for integrals

There's another thing – a non-statistics thing – that Monte Carlo estimation is useful for. We can use Monte Carlo estimation to approximate integrals that are too hard to do by hand.

This might seem surprising. Estimating the expectation of (a function of) a random variable seems a naturally statistical thing to do. But an integral is just a straight maths problem – there's not any randomness at all. But actually, integrals and expectations are very similar things.

Let's think of an integral: say,

$$\int_a^b h(x)\,\mathrm{d}x,$$

the integral of some function h (the "integrand") between the limits a and b. Now let's compare that to the integral $\mathbb{E}\,\phi(X)$ of a continuous random variable that we can estimate using Monte Carlo estimation,

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

Matching things up, we can see that we if we were to a function ϕ and a PDF f such that

$$\phi(x)\,f(x) = \begin{cases} 0 & x < a \\ h(x) & a \leq x \leq b \\ 0 & x > b, \end{cases} \tag{2.1}$$

then we would have

$$\mathbb{E}\,\phi(X) = \int_{-\infty}^{\infty} \phi(x)\,f(x)\,\mathrm{d}x = \int_a^b h(x)\,\mathrm{d}x,$$

so the value of the expectation would be precisely the value of the integral we're after. Then we could use Monte Carlo to estimate that expectation/integral.

There are lots of choices of ϕ and f that would satisfy this the condition in Equation 2.1. But a "common-sense" choice that often works is to pick f to be the PDF of X, a continuous

uniform distribution on the interval $[a, b]$. (This certainly works when a and b are finite, anyway.) Recall that the continuous uniform distribution means that X has PDF

$$f(x) = \begin{cases} 0 & x < a \\ \dfrac{1}{b-a} & a \leq x \leq b \\ 0 & x > b. \end{cases}$$

Comparing this equation with Equation 2.1, we then have to choose

$$\phi(x) = \frac{h(x)}{f(x)} = (b-a)h(x).$$

Putting this all together, we have

$$\mathbb{E}\,\phi(X) = \int_{-\infty}^{+\infty} \phi(x)\, f(x)\,\mathrm{d}x = \int_a^b (b-a)h(x)\,\frac{1}{b-a}\,\mathrm{d}x = \int_a^b h(x)\,\mathrm{d}x,$$

as required. This can then be estimated using the Monte Carlo method.

Definition 2.2. Consider an integral $\theta = \int_a^b h(x)\,\mathrm{d}x$. Let f be the probability density function of a random variable X and let ϕ be function such that Equation 2.1 holds. Then the **Monte Carlo estimator** $\widehat{\theta}_n^{\mathrm{MC}}$ of the integral θ is

$$\widehat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} \phi(X_i),$$

where $X_1, X_2, \ldots, X_n$ are a random sample from X.

Example 2.2. Suppose we want to approximate the integral

$$\int_0^2 x^{1.6}(2-x)^{0.7}\,\mathrm{d}x.$$

Since this is an integral on the finite interval $[0, 2]$, it would seem to make sense to pick X to be uniform on $[0, 2]$. This means we should take

$$\phi(x) = \frac{h(x)}{f(x)} = (2-0)h(x) = 2\,x^{1.6}(2-x)^{0.7}.$$

We can then approximate this integral in R using the Monte Carlo estimator

$$\int_0^2 x^{1.6}(2-x)^{0.7}\,\mathrm{d}x = \mathbb{E}\,\phi(X) \approx \frac{1}{n}\sum_{i=1}^{n} 2\,X_i^{1.6}(2-X_i)^{0.7}.$$

```r
n <- 1e6
integrand <- function(x) x^1.6 * (2 - x)^0.7
a <- 0
b <- 2
samples <- runif(n, a, b)
mean((b - a) * integrand(samples))
```

```
[1] 1.444437
```

You have perhaps noticed that, here and elsewhere, I tend to split my R code up into lots of small bits, perhaps slightly unnecessarily. After all, those 6 lines of code could simply have been written as just 2 lines

```r
samples <- runif(1e6, 0, 2)
mean(2 * samples^1.6 * (2 - samples)^0.7)
```

There's nothing *wrong* with that. However, I find that code is easier to read if divided into small pieces. It also makes it easier to tinker with, if I want to use it to solve some similar but slightly different problem.

Example 2.3. Suppose we want to approximate the integral

$$\int_{-\infty}^{+\infty} \mathrm{e}^{-0.1|x|} \cos x \, \mathrm{d}x.$$

This one is an integral on the whole real line, so we can't take a uniform distribution. Maybe we should take $f(x)$ to be the PDF of a normal distribution, and then put

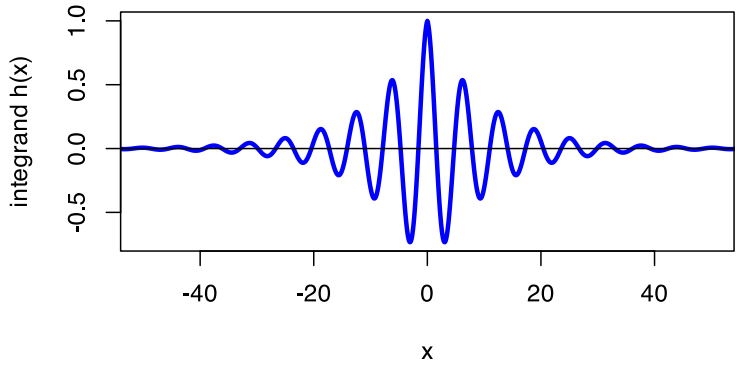
$$\phi(x) = \frac{h(x)}{f(x)} = \frac{\mathrm{e}^{-0.1|x|} \cos x}{f(x)}.$$

But which normal distribution should we take? Well, we're *allowed* to take any one – we will still get an accurate estimate in the limit as $n \to \infty$. But we'd like an estimator that gives accurate results at moderate-sized n, and picking a "good" distribution for X will help that.

We'll probably get the best results if we pick a distribution that is likely to mostly take values where $h(x)$ is big – or, rather, where the absolute value $|h(x)|$ is big, to be precise. That is because we don't want to "waste" too many samples where $h(x)$ is very small, because they don't contribute much to the integral. But we don't want to "miss" – or only sample very rarely – places where $h(x)$ is big, which contribute a lot to the integral.

Let's have a look at the graph of $h(x) = \mathrm{e}^{-0.1|x|} \cos x$.

```r
integrand <- function(x) exp(-0.1 * abs(x)) * cos(x)

curve(
  integrand, n = 1001, from = -55, to = 55,
  col = "blue", lwd = 3,
  xlab = "x", ylab = "integrand h(x)", xlim = c(-50,50)
)
abline(h = 0)
```



This suggests to me that a mean of 0 and a standard deviation of 20 might work quite well, since this will tend to take values in $[-40, 40]$ or so.

We will use R's function `dnorm()` for the probability density function of the normal distribution (which saves us from having to remember what that is).

```r
n <- 1e6
integrand <- function(x) exp(-0.1 * abs(x)) * cos(x)
pdf       <- function(x) dnorm(x, 0, 20)
phi       <- function(x) integrand(x) / pdf(x)

samples <- rnorm(n, 0, 20)
mean(phi(samples))
```

```
[1] 0.2189452
```

Next time: *We will analyse the accuracy of these Monte Carlo estimates.*

Summary:

- The indicator $\mathbb{I}_A(x)$ function of a set A is 1 if $x \in A$ or 0 if $x \notin A$.
- We can estimate a probability $\mathbb{P}(X \in A)$ by using the Monte Carlo estimate for $\mathbb{E}\,\mathbb{I}_A(X)$.
- We can estimate an integral $\int h(x)\,\mathrm{d}x$ by using a Monte Carlo estimate with $\phi(x)\,f(x) = h(x)$.

Read more: Voss, *An Introduction to Statistical Computing*, Section 3.1 and Subsection 3.2.1.

3 Monte Carlo error I: theory

3.1 Estimation error

Today we are going to analysing the accuracy of Monte Carlo estimation. But before talking about Monte Carlo estimation specifically, let's first remind ourselves of some concepts about error in statistical estimation more generally. We will use the following definitions.

Definition 3.1. Let $\hat{\theta}$ be an estimator of a parameter θ. Then we have the following definitions of the estimator $\hat{\theta}$:

- The **bias** is $\text{bias}\,(\hat{\theta}) = \mathbb{E}(\hat{\theta} - \theta) = \mathbb{E}\hat{\theta} - \theta$.
- The **mean-square error** is $\text{MSE}\,(\hat{\theta}) = \mathbb{E}(\hat{\theta} - \theta)^2$.
- The **root-mean-square error** is the square-root of the mean-square error,

$$\text{RMSE}\,(\hat{\theta}) = \sqrt{\text{MSE}(\hat{\theta})} = \sqrt{\mathbb{E}(\hat{\theta} - \theta)^2}.$$

Usually, the main goal of estimation is to get the mean-square error of an estimate as small as possible. This is because the MSE measures by what distance we are missing on average. It can be easier to interpret what the root-mean-square error means, as the RMSE has the same units as the parameter being measured: if θ and $\hat{\theta}$ are in metres, say, then the MSE is in metres-squared, whereas the RMSE error is in metres again. If you minimise the MSE you also minimise the RMSE and vice versa.

It's nice to have an "unbiased" estimator – that is, one with bias 0. This is because bias measures any systematic error in a particular direction. However, unbiasedness by itself is not enough for an estimate to be good – we need low variance too. (Remember the old joke about the statistician who misses his first shot ten yards to the left, misses his second shot ten yards to the right, then claims to have "hit the target on average.")

(Remember also that "bias" is simply the word statisticians use for $\mathbb{E}(\hat{\theta} - \theta)$; we don't mean "bias" in the derogatory way it is sometimes used in political arguments, for example.)

You probably also remember the relationship between the mean-square error, the bias, and the variance:

Theorem 3.1. $\mathrm{MSE}\left(\hat{\theta}\right) = \mathrm{bias}\left(\hat{\theta}\right)^2 + \mathrm{Var}\left(\hat{\theta}\right)$.

Proof. The MSE is

$$\mathrm{MSE}\left(\hat{\theta}\right) = \mathbb{E}\left(\hat{\theta} - \theta\right)^2 = \mathbb{E}\left(\hat{\theta}^2 - 2\theta\hat{\theta} + \theta\right)^2 \tag{3.1}$$

$$= \mathbb{E}\,\hat{\theta}^2 - 2\theta\,\mathbb{E}\,\hat{\theta} + \theta^2, \tag{3.2}$$

where we have expanded the brackets and bought the expectation inside (remembering that θ is a constant). Since the variance can be written as $\mathrm{Var}(\hat{\theta}) = \mathbb{E}\,\hat{\theta}^2 - (\mathbb{E}\,\hat{\theta})^2$, we can use a cunning trick of both subtracting and adding $(\mathbb{E}\,\hat{\theta})^2$. This gives

$$\mathrm{MSE}\left(\hat{\theta}\right) = \mathbb{E}\,\hat{\theta}^2 - \left(\mathbb{E}\,\hat{\theta}\right)^2 + \left(\mathbb{E}\,\hat{\theta}\right)^2 - 2\theta\,\mathbb{E}\,\hat{\theta} + \theta^2 \tag{3.3}$$

$$= \mathrm{Var}\left(\hat{\theta}\right) + \left((\mathbb{E}\,\hat{\theta})^2 - 2\theta\,\mathbb{E}\,\hat{\theta} + \theta^2\right) \tag{3.4}$$

$$= \mathrm{Var}\left(\hat{\theta}\right) + \left(\mathbb{E}\,\hat{\theta} - \theta\right)^2 \tag{3.5}$$

$$= \mathrm{Var}\left(\hat{\theta}\right) + \mathrm{bias}(\hat{\theta})^2. \tag{3.6}$$

This proves the result. □

Since the bias contributes to the mean-square error, that's another reason to like estimator with low – or preferably zero – bias. But again, unbiasedness isn't enough by itself; we want low variance too. (There are some situations where there's a "bias–variance tradeoff", where allowing some bias reduces the variance and so can reduce the MSE. It turns out that Monte Carlo is not one of these cases, however.)

3.2 Error of Monte Carlo estimator: theory

In this lecture, we're going to be looking more carefully at the size of the errors made by the Monte Carlo estimator

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}(\phi(X_1) + \phi(X_2) + \cdots + \phi(X_n)) = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i).$$

Our main result is the following.

Theorem 3.2. *Let X be a random variable, ϕ a function, and $\theta = \mathbb{E}\,\phi(X)$. Let*

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i)$$

be the Monte Carlo estimator of θ. Then:

1. *$\hat{\theta}_n^{\mathrm{MC}}$ is unbiased, in that $\mathrm{bias}\,(\hat{\theta}_n^{\mathrm{MC}}) = 0$.*
2. *The variance of of $\hat{\theta}_n^{\mathrm{MC}}$ is $\mathrm{Var}\,(\hat{\theta}_n^{\mathrm{MC}}) = \frac{1}{n}\mathrm{Var}\,(\phi(X))$.*
3. *The mean-square error of $\hat{\theta}_n^{\mathrm{MC}}$ is $\mathrm{MSE}\,(\hat{\theta}_n^{\mathrm{MC}}) = \frac{1}{n}\mathrm{Var}\,(\phi(X))$.*
4. *The root-mean-square error of $\hat{\theta}_n^{\mathrm{MC}}$ is*

$$\mathrm{RMSE}\,(\hat{\theta}_n^{\mathrm{MC}}) = \sqrt{\frac{1}{n}\mathrm{Var}\,(\phi(X))} = \frac{1}{\sqrt{n}}\,\mathrm{sd}\,(\phi(X)).$$

Before we get to the proof, let's recap some relevant probability.

Let $Y_1, Y_2, \ldots$ be IID random variables with common expectation $\mathbb{E}Y_1 = \mu$ and common variance $\mathrm{Var}(Y_1) = \sigma^2$. Consider the mean of the first n random variables,

$$\overline{Y}_n = \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

Then the expectation of $\overline{Y}_n$ is

$$\mathbb{E}\overline{Y}_n = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}Y_i = \frac{1}{n}\,n\,\mu = \mu.$$

The variance of $\overline{Y}_n$ is

$$\mathrm{Var}\,(\overline{Y}_n) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \left(\frac{1}{n}\right)^2\sum_{i=1}^{n}\mathrm{Var}(Y_i) = \frac{1}{n^2}\,n\,\sigma^2 = \frac{\sigma^2}{n},$$

where, for this one, we used the independence of the random variables.

Proof. Apply the probability facts from above with $Y = \phi(X)$. This gives:

1. $\mathbb{E}\hat{\theta}_n^{\mathrm{MC}} = \mathbb{E}\overline{Y}_n = \mathbb{E}Y = \mathbb{E}\,\phi(X)$, so $\mathrm{bias}(\hat{\theta}_n^{\mathrm{MC}}) = \mathbb{E}\,\phi(X) - \mathbb{E}\,\phi(X) = 0$.
2. $\mathrm{Var}\,(\hat{\theta}_n^{\mathrm{MC}}) = \mathrm{Var}\,(\overline{Y}_n) = \frac{1}{n}\mathrm{Var}(Y) = \frac{1}{n}\mathrm{Var}\,(\phi(X))$.

3. Using Theorem 3.1,

$$\mathrm{MSE}(\widehat{\theta}_n^{\mathrm{MC}}) = \mathrm{bias}(\widehat{\theta}_n^{\mathrm{MC}})^2 + \mathrm{Var}(\widehat{\theta}_n^{\mathrm{MC}}) = 0^2 + \frac{1}{n}\,\mathrm{Var}\left(\phi(X)\right) = \frac{1}{n}\,\mathrm{Var}\left(\phi(X)\right).$$

4. Take the square root of part 3.

□

Let's think about MSE $\frac{1}{n}\,\mathrm{Var}(\phi(X))$. The variance terms is some fixed fact about the random variable X and the function ϕ. So as n gets bigger, $\frac{1}{n}$ gets smaller, so the MSE gets smaller, and the estimator gets more accurate. This goes back to what we said when we introduced the Monte Carlo estimator: we get a more accurate estimate by increasing n. More specifically, the MSE scales like $1/n$, or – perhaps a more useful result – the RMSE scales like $1/\sqrt{n}$. We'll come back to this in the next lecture.

3.3 Error of Monte Carlo estimator: practice

So when we form a Monte Carlo estimate $\widehat{\theta}_n^{\mathrm{MC}}$, we now know it will be unbiased. We'd also like to know it's mean-square and/or root-mean-square error too.

There's a problem here, though. The reason we are doing Monte Carlo estimation in the first place is that we *couldn't* calculate $\mathbb{E}\,\phi(X)$. So it seems very unlikely we'll be able to calculate the variance $\mathrm{Var}(\phi(X))$ either. So how will be able to assess the mean-square (or root-mean-square) error of our Monte Carlo estimator?

Well, we can't know it exactly. But we *can* estimate the variance from the samples we are already using: by taking the sample variance of the samples $\phi(x_i)$. That is, we can estimate the variance of the Monte Carlo estimator by the sample variance

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\phi(X_i) - \widehat{\theta}_n^{\mathrm{MC}}\right)^2.$$

Then we can similarly estimate the mean-square and root-mean-square errors by

$$\mathrm{MSE} \approx \frac{1}{n}S^2 \qquad \text{and} \qquad \mathrm{RMSE} \approx \sqrt{\frac{1}{n}S^2} = \frac{1}{\sqrt{n}}\,S$$

respectively.

Example 3.1. Let's go back to the very first example in the module, Example 1.1, where we were trying to find the expectation of an Exp(2) random variable. We used this R code:

```
n <- 1e6
samples <- rexp(n, 2)
MCest <- mean(samples)
MCest
```

```
[1] 0.5006352
```

(Because Monte Carlo estimation is random, this won't be the *exact* same estimate we had before, of course.)

So if we want to investigate the error, we can use the sample variance of these samples. We will use the sample variance function `var()` to calculate the sample variance. In this simple case, the function is $\phi(x) = x$, so we need only use the variance of the samples themselves.

```
var_est <- var(samples)
MSEest  <- var_est / n
RMSEest <- sqrt(MSEest)
c(var_est, MSEest, RMSEest)
```

```
[1] 2.503570e-01 2.503570e-07 5.003569e-04
```

The first number is `var_est` $= 0.2504$, the sample variance of our $\phi(x_i)$s:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\phi(x_i) - \hat{\theta}_n^{\mathrm{MC}}\right)^2.$$

This should be a good estimate of the true variance $\mathrm{Var}(\phi(X))$. (In fact, in this simple case, we know that $\mathrm{Var}(X) = \frac{1}{2^2} = 0.25$, so we know that the estimate is good.) In calculating this in the code, we used R's `var()` function, which calculates the sample variance of some values.

The second number is `MSEest` $= 2.504 \times 10^{-7}$, our estimate of the mean-square error. Since $\mathrm{MSE}(\hat{\theta}_n^{\mathrm{MC}}) = \frac{1}{n}\mathrm{Var}(\phi(X))$, then $\frac{1}{n}S^2$ should be a good estimate of the MSE.

The third number is `RMSEest` $= 5 \times 10^{-4}$ our estimate of the root-mean square error, which is simply the square-root of our estimate of the mean-square error.

Example 3.2. In Example 2.1, we were estimating $\mathbb{P}(Z > 2)$, where Z is a standard normal.

Our code was

```
n <- 1e6
samples <- rnorm(n)
MCest <- mean(samples > 2)
MCest
```

```
[1] 0.022458
```

So our root-mean-square error can be approximated as

```
MSEest <- var(samples > 2) / n
sqrt(MSEest)
```

```
[1] 0.0001481677
```

since `samples > 2` is the indicator function of whether $X_i > 2$ or not.

Next time: *We'll continue analysing Monte Carlo error, looking at confidence intervals and assessing how many samples to take..*

Summary:

- The Monte Carlo estimator is unbiased.
- The Monte Carlo estimator has mean-square error $\mathrm{Var}(\phi(X))/n$, so the root-mean-square error scales like $1/\sqrt{n}$.
- The mean-square error can be estimated by S^2/n, where S^2 is the sample variance of the $\phi(X_i)$.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.2.2.

4 Monte Carlo error II: practice

4.1 Recap

Let's recap where we've got to. We know that the Monte Carlo estimator for $\theta = \mathbb{E}\,\phi(X)$ is

$$\widehat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i).$$

Last time, we saw that the Monte Carlo estimator is unbiased, and that its mean-square and root-mean-square errors are

$$\mathrm{MSE}\left(\widehat{\theta}_n^{\mathrm{MC}}\right) = \frac{1}{n}\,\mathrm{Var}\left(\phi(X)\right) \qquad \mathrm{RMSE}\left(\widehat{\theta}_n^{\mathrm{MC}}\right) = \sqrt{\frac{1}{n}\,\mathrm{Var}\left(\phi(X)\right)}.$$

We saw that these themselves can be estimated as S^2/n and $S/\sqrt{n}$ respectively, where S^2 is the sample variance of the $\phi(X_i)$s.

4.2 Confidence intervals

So far, we have described our error tolerance in terms of the MSE or RMSE. But we could have talked about "confidence intervals" or "margins of error" instead. This might be easier to understand for non-mathematicians, for whom "root-mean-square error" doesn't really mean anything.

Here, we will want to appeal to the central limit theorem approximation. A bit more probability revision: Let $Y_1, Y_2, ...$ be IID again, with expectation μ and variance σ^2. Write $\overline{Y}_n$ for the mean. We've already reminded ourselves of the law of large numbers, which says that $\overline{Y}_n \to \mu$ as $n \to infty$. Then in the last lecture we saw that $\mathbb{E}\overline{Y}_n = \mu$ and $\mathrm{Var}(\overline{Y}_n) = \sigma^2/n$. The **central limit theorem** says that the distribution of $\overline{Y}_n$ is approximately normally distributed with those parameters, so $\overline{Y}_n \approx \mathrm{N}(\mu, \sigma^2/n)$ when n is large. (This is an informal statement of the central limit theorem: you probably know some more formal ways to more precisely state it, but this will do for us.)

Recall that, in the normal distribution $\mathrm{N}(\mu, \sigma^2)$, we expect to be within 1.96 standard deviations of the mean with 95% probability. More generally, the interval $[\mu - q_{1-\alpha/2}\sigma, \mu + q_{1-\alpha/2}\sigma]$, where $q_{1-\alpha/2}$ is the $(1 - \frac{\alpha}{2})$-quantile of the normal distribution, contains the true value with probability approximately $1 - \alpha$.

We can form an approximate confidence interval for a Monte Carlo estimate using this idea. We have our Monte Carlo estimator $\hat{\theta}_n^{\mathrm{MC}}$ as our estimator of the μ parameter, and our estimator of the root-mean-square error $S/\sqrt{n}$ as our estimator of the σ parameter. So our confidence interval is estimated as

$$\left[\hat{\theta}_n^{\mathrm{MC}} - q_{1-\alpha/2}\frac{S}{\sqrt{n}},\ \hat{\theta}_n^{\mathrm{MC}} + q_{1-\alpha/2}\frac{S}{\sqrt{n}}\right].$$

Example 4.1. We continue the example of Example 2.1 and Example 3.2, where we were estimating $\mathbb{P}(Z > 2)$ for Z a standard normal.

```
n <- 1e6
samples <- rnorm(n)
MCest   <- mean(samples > 2)
RMSEest <- sqrt(var(samples > 2) / n)
MCest
```

```
[1] 0.023177
```

Our confidence interval is estimates as follows

```
alpha <- 0.05
quant <- qnorm(1 - alpha / 2)
c(MCest - quant * RMSEest, MCest + quant * RMSEest)
```

```
[1] 0.02288209 0.02347191
```

4.3 How many samples do I need?

In our examples we've picked the number of samples n for our estimator, then approximated the error based on that. But we could do things the other way around – fix an error tolerance that we're willing to deal with, then work out what sample size we need to achieve it.

We know that the root-mean-square error is

$$\mathrm{RMSE}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \sqrt{\frac{1}{n}\mathrm{Var}\left(\phi(X)\right)}$$

So if we want to get the RMSE down to ϵ, say, then this shows that we need

$$\epsilon = \sqrt{\frac{1}{n} \operatorname{Var}(\phi(X))}.$$

Squaring both sides, multiplying both sides by n, and dividing both sides by ϵ^2 gives

$$n = \frac{1}{\epsilon^2} \operatorname{Var}(\phi(X)).$$

So this tells us how many samples n we need. Except we still have a problem here, though. We (usually) don't know $\operatorname{Var}(\phi(X))$. But we can't even *estimate* $\operatorname{Var}(\phi(X))$ until we've already taken the samples. So it seems we're stuck.

But we can use this idea with a three-step process:

1. Run an initial "pilot" Monte Carlo algorithm with a small number of samples n. Use the results of the "pilot" to estimate the variance $S^2 \approx \operatorname{Var}(\phi(X))$. We want n small enough that this runs very quickly, but big enough that we get a reasonably OK estimate of the variance.
2. Pick a desired RMSE accuracy ϵ. We now know that we require roughly $N = S^2/\epsilon^2$ samples to get our desired accuracy.
3. Run the "real" Monte Carlo algorithm with this big number of samples N. We will put up with this being quite slow, because we know we're definitely going to get the error tolerance we need.

(We could potentially use further steps, where we now check the variance with the "real" big-N samples, and, if we learn we had underestimated in Step 1, take even more samples to correct for this.)

Example 4.2. Let's try this with Example 1.2 from before. We were trying to estimate $\mathbb{E}(\sin X)$, where $X \sim \mathrm{N}(1, 2^2)$.

We'll start with just $n = 1000$ samples, for our pilot study.

```
n_pilot <- 1000
samples <- rnorm(n_pilot, 1, 2)
var_est <- var(sin(samples))
var_est
```

```
[1] 0.4905153
```

This was very quick! We won't have got a super-accurate estimate of $\mathbb{E}\phi(X)$, but we have a reasonable idea of roughly what $\mathrm{Var}(\phi(X))$ is. This will allow us to pick out "real" sample size in order to get a root-mean-square error of 10^{-4}.

```
epsilon <- 1e-4
n_real  <- round(var_est / epsilon^2)
n_real
```

```
[1] 49051535
```

This tells us that we will need about 50 million samples! This is a lot, but now we know we're going to get the accuracy we want, so it's worth it. (In this particular case, 50 million samples will only take a few second on a modern computer. But generally, once we know our code works and we know how many samples we will need for the desired accuracy, this is the sort of thing that we could leave running overnight or whatever.)

```
samples <- rnorm(n_real, 1, 2)
MCest <- mean(sin(samples))
MCest
```

```
[1] 0.1139149
```

```
RMSEest <- sqrt(var(sin(samples)) / n_real)
RMSEest
```

```
[1] 9.964886e-05
```

This second step was quite slow (depending on the speed of the computer being used – it was only about 5 seconds on my pretty-new laptop, but slower on my ancient work desktop). But we see that we have indeed got our Monte Carlo estimate to (near enough) the desired accuracy.

Generally, if we want a more accurate Monte Carlo estimator, we can just take more samples. But the equation

$$n = \frac{1}{\epsilon^2}\,\mathrm{Var}\left(\phi(X)\right)$$

is actually quite bad news. To get an RMSE of ϵ we need order $1/\epsilon^2$ samples. That's not good. Think of it like this: to *double* the accuracy we need to *quadruple* the number of samples. Even worse: to get "one more decimal place of accuracy" means dividing ϵ by ten; but that means multiplying the number of samples by one hundred!

More samples take more time, and cost more energy and money. Wouldn't it be nice to have some better ways of increasing the accuracy of a Monte Carlo estimate besides just taking more and more samples?

Next time: *We begin our study of clever "variance reduction" methods for Monte Carlo estimation.*

Summary:

- We can approximate confidence intervals for a Monte Carlo estimate by using a normal approximation.
- To get the root-mean-square error below ϵ we need $n = \mathrm{Var}(\phi(X))/\epsilon^2$ samples.
- We can use a two-step process, where a small "pilot" Monte Carlo estimation allows us to work out how many samples we will need for the big "real" estimation.

Read more: Voss, *An Introduction to Statistical Computing*, Subsections 3.2.2–3.2.4.

5 Control variate

5.1 Variance reduction

Let's recap where we've got to. The Monte Carlo estimator of $\theta = \mathbb{E}\,\phi(X)$ is

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i),$$

where $X_1, X_2, \ldots, X_n$ are IID random samples from X. The mean-square error of this estimator is

$$\mathrm{MSE}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \frac{1}{n}\,\mathrm{Var}\left(\phi(X)\right).$$

If we want a more accurate estimate, we can just take more samples n. But the problem is that the root-mean-square error scales like $1/\sqrt{n}$. To double the accuracy, we need four times as many samples; for one more decimal place of accuracy, we need one hundred times as many samples.

Are there other ways we could reduce the error of Monte Carlo estimation, so we need fewer samples? That is, can we use some mathematical ingenuity to adapt the Monte Carlo estimate to one with a smaller error?

Well, the mean-square error is the variance divided by n. So if we can't (or don't want to) increase n, perhaps we can *decrease* the *variance* instead? Strategies to do this are called **variance reduction strategies**. In this module, we will look at three variance reduction strategies:

- **Control variate:** We can "anchor" our estimate of $\mathbb{E}\,\phi(X)$ to a similar but easier-to-calculate value $\mathbb{E}\,\psi(X)$. (This lecture)
- **Antithetic variables:** Instead of using independent samples, we could use correlated samples. If the correlation is negative this can improve our estimate. (Lectures 6 and 7)
- **Importance sampling:** Instead of sampling from X, sample from some other more suitable distribution instead, then adjust the answer we get. (Lectures 8 and 9)

5.2 Control variate estimation

In Monday's lecture, I polled the class on this question: *Estimate the average time it takes to fly from London to New York.*

- The actual answer is: 8 hours.
- The mean guess was: 9 hours and 38 minutes (98 minutes too much)
- The root-mean-square error for the guesses was: 158 minutes

After you'd guessed, I gave the following hint: *Hint: The average time it takes to fly from London to Washington D.C. is 8 hours and 15 minutes.* After the hint:

- The mean guess was: 8 hours and 50 minutes (50 minutes too much)
- The root-mean-square error for the guesses was: 72 minutes

So after the hint, the error of the class was reduced by 55%.

(Incidentally, you were about 30% at guessing this than last year's students…)

Why did the hint help? We were trying to estimate θ^{NY}, the distance to New York. But that's a big number, and the first estimates had a big error (over an hour, on average). After the hint, I expect most people thought something like this: "The answer θ^{NY} is going to be similar to the $\theta^{\mathrm{DC}} = 8{:}15$ to Washington D.C., but New York isn't quite as far, so I should decrease the number a bit, but not too much."

To be more mathematical, we could write

$$\theta^{\mathrm{NY}} = \theta^{\mathrm{NY}} + (\theta^{\mathrm{DC}} - \theta^{\mathrm{DC}}) = \underbrace{\theta^{\mathrm{DC}}}_{\text{known}} + \underbrace{(\theta^{\mathrm{NY}} - \theta^{\mathrm{DC}})}_{\text{small}}.$$

In that equation, the first term, $\theta^{\mathrm{DC}} = 8{:}15$ was completely known, so had error 0, while the second term $\theta^{\mathrm{NY}} - \theta^{\mathrm{DC}}$ (actually minus 15 minutes) was a small number, so only had a small error.

This idea of improving an estimate by "anchoring" it to some known value is called **controlled estimation**. It is a very useful idea in statistics (and in life!).

We can apply this idea to Monte Carlo estimation too. Suppose we are trying to estimate $\theta = \mathbb{E}\,\phi(X)$. We could look for a function ψ that is similar to ϕ (at least for the values of x that have high probability for the random variable X), but where we know for certain what $\mathbb{E}\,\psi(X)$ is. Then we can write

$$\theta = \mathbb{E}\,\phi(X) = \mathbb{E}\,(\phi(X) - \psi(X) + \psi(X)) = \underbrace{\mathbb{E}\,(\phi(X) - \psi(X))}_{\text{estimate this with Monte Carlo}} + \underbrace{\mathbb{E}\,\psi(X)}_{\text{known}}.$$

Here, $\psi(X)$ is known as the **control variate**.

Definition 5.1. Let X be a random variable, ϕ a function, and write $\theta = \mathbb{E}\,\phi(X)$. Let ψ be a function such that $\eta = \mathbb{E}\,\psi(X)$ is known. Suppose that $X_1, X_2, \ldots, X_n$ are a random sample from X. Then the **control variate Monte Carlo estimate** $\hat{\theta}_n^{\text{CV}}$ of θ is

$$\hat{\theta}_n^{\text{CV}} = \frac{1}{n}\sum_{i=1}^{n} (\phi(X_i) - \psi(X_i)) + \eta.$$

Example 5.1. Let's try to estimate $\mathbb{E}\cos(X)$, where $X \sim \text{N}(0,1)$ is a standard normal distribution.

We could do this the "usual" Monte Carlo way.

```
n <- 1e6
phi <- function(x) cos(x)
samples <- rnorm(n)
MCest <- mean(phi(samples))
MCest
```

```
[1] 0.6067224
```

But we could see if we can do better with a control variate. But what should we pick for the control function ψ? We want something that's similar to $\phi(x) = \cos(x)$, but where we can actually calculate the expectation.

Here's a suggestion. If we remember our Taylor series, we know that, for x near 0,

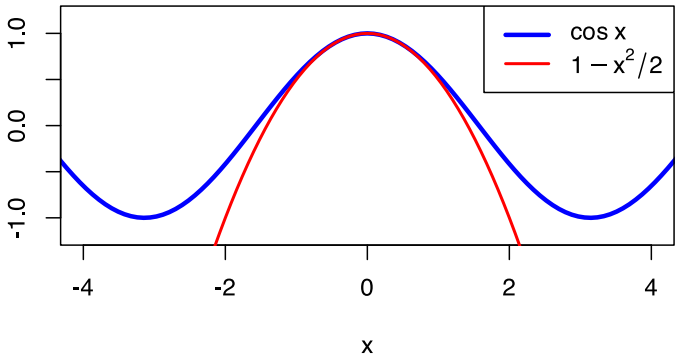
$$\cos x \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots.$$

So how about taking the first two nonzero terms in the Taylor series

$$\psi(x) = 1 - \frac{x^2}{2}.$$

That is quite close to $\cos x$, at least for the values of x near 0 that $X \sim \text{N}(0,1)$ is most likely to take.

```
curve(
  cos(x), from = -4.5, to = 4.5,
  col = "blue", lwd = 3,
  xlab = "x", ylab = "", xlim = c(-4,4), ylim = c(-1.2,1.2)
)
curve(1 - x^2 / 2, add = TRUE, col = "red", lwd = 2)
legend(
  "topright", c("cos x", expression(1 - x^2 / 2)),
  lwd = c(3, 2), col = c("blue", "red")
)
```



Not only that, but we know that for $Y \sim \mathrm{N}(\mu, \sigma^2)$ we have $\mathbb{E}Y^2 = \mu^2 + \sigma^2$. So

$$\mathbb{E}\,\psi(X) = \mathbb{E}\left(1 - \frac{X^2}{2}\right) = 1 - \frac{\mathbb{E}X^2}{2} = 1 - \frac{0^2 + 1}{2} = \frac{1}{2}.$$

So our control variate estimate is:

```
psi <- function(x) 1 - x^2 / 2
CVest <- mean(phi(samples) - psi(samples)) + 1/2
CVest
```

```
[1] 0.6060243
```

5.3 Error of control variate estimate

What is the error in a control variate estimate?

Theorem 5.1. *Let X be a random variable, ϕ a function, and $\theta = \mathbb{E}\,\phi(X)$. Let ψ be a function such that $\eta\,\mathbb{E}\,\psi(X)$ is known. Let*

$$\hat{\theta}_n^{\mathrm{CV}} = \frac{1}{n}\sum_{i=1}^{n}\left(\phi(X_i) - \psi(X_i)\right) + \eta$$

be the control variate Monte Carlo estimator of θ. Then:

1. *$\hat{\theta}_n^{\mathrm{CV}}$ is unbiased, in that* $\mathrm{bias}\left(\hat{\theta}_n^{\mathrm{CV}}\right) = 0$.

2. *The variance of of* $\hat{\theta}_n^{\mathrm{CV}}$ *is* $\operatorname{Var}(\hat{\theta}_n^{\mathrm{CV}}) = \frac{1}{n}\operatorname{Var}(\phi(X) - \psi(X))$.

3. *The mean-square error of* $\hat{\theta}_n^{\mathrm{CV}}$ *is* $\operatorname{MSE}(\hat{\theta}_n^{\mathrm{CV}}) = \frac{1}{n}\operatorname{Var}(\phi(X) - \psi(X))$.

4. *The root-mean-square error of* $\hat{\theta}_n^{\mathrm{CV}}$ *is* $\operatorname{RMSE}(\hat{\theta}_n^{\mathrm{CV}}) = \frac{1}{\sqrt{n}}\sqrt{\operatorname{Var}(\phi(X) - \psi(X))}$.

Proof. This is very similar to Theorem 3.2, so we'll just sketch the important differences.

In part 1, we have

$$\begin{aligned}
\mathbb{E}\,\hat{\theta}_n^{\mathrm{CV}} &= \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - \psi(X_i))\right) + \eta \\
&= \frac{1}{n}\,\mathbb{E}\left(\sum_{i=1}^{n}(\phi(X_i) - \psi(X_i))\right) + \eta \\
&= \frac{n}{n}\,\mathbb{E}\,(\phi(X) - \psi(X)) + \eta \\
&= \mathbb{E}\,\phi(X) - \mathbb{E}\,\psi(X) + \eta \\
&= \mathbb{E}\,\phi(X),
\end{aligned}$$

since $\eta = \mathbb{E}\,\psi(X)$. So the estimator is unbiased.

For part 2, remembering that $\eta = \mathbb{E}\,\psi(X)$ is a constant, so doesn't affect the variance, we have

$$\begin{aligned}
\operatorname{Var}(\hat{\theta}_n^{\mathrm{CV}}) &= \operatorname{Var}\left(\frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - \psi(X_i)) + \eta\right) \\
&= \left(\frac{1}{n}\right)^2 \operatorname{Var}\left(\sum_{i=1}^{n}(\phi(X_i) - \psi(X_i))\right) \\
&= \frac{n}{n^2}\operatorname{Var}(\phi(X) - \psi(X)) \\
&= \frac{1}{n}\operatorname{Var}(\phi(X) - \psi(X)).
\end{aligned}$$

Parts 3 and 4 follow in the usual way. □

This tells us that a control variate Monte Carlo estimate is good when the variance of $\phi(X) - \psi(X)$. This variance is likely to be small if $\phi(X) - \psi(X)$ is usually small – although, to be more precise, it's more important for $\phi(X) - \psi(X)$ to be *consistent*, rather than small per se.

As before, we can't usually calculate the variance $\mathrm{Var}(\phi(X) - \psi(X))$ exactly, but we can estimate it from the samples. Again, we use the sample variance of $\phi(X_i) - \psi(X_i)$,

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} \Big((\phi(X_i) - \psi(X_i)) - (\hat{\theta}_n^{\mathrm{CV}} - \eta) \Big)^2,$$

and estimate the MSE and RMSE by S^2/n and $S/\sqrt{n}$ respectively.

Example 5.2. We return to Example 5.1, where we were estimating $\mathbb{E}\cos(X)$ for $X \sim \mathrm{N}(0, 1)$.

The naive Monte Carlo estimate had mean-square and root-mean-square error

```
n <- 1e6
phi <- function(x) cos(x)
samples <- rnorm(n)
MC_MSE <- var(phi(samples)) / n
c(MC_MSE, sqrt(MC_MSE))
```

```
[1] 2.002522e-07 4.474955e-04
```

The variance and root-mean-square error of our control variate estimate, on the other hand, are

```
psi <- function(x) 1 - x^2 / 2
CV_MSE <- var(phi(samples) - psi(samples)) / n
c(CV_MSE, sqrt(CV_MSE))
```

```
[1] 9.326675e-08 3.053960e-04
```

This was a success! The mean-square error roughly halved, from 2×10^{-7} to 9.3×10^{-8}. This meant the root-mean-square went down by about a third, from 4.5×10^{-4} to 3.1×10^{-4}.

Halving the mean-square error would normally have required doubling the number of samples n, so we have effectively doubled the sample size by using the control variate.

Next time: *We look at our second variance reduction technique: antithetic variables.*

Summary:

- Variance reduction techniques attempt to improve on Monte Carlo estimation making the variance smaller.

- If we know $\eta = \mathbb{E}\,\psi(X)$, then the control variate Monte Carlo estimate is

$$\hat{\theta}_n^{\mathrm{CV}} = \frac{1}{n}\sum_{i=1}^{n}\left(\phi(X_i) - \psi(X_i)\right) + \eta.$$

- The mean-square error of the control variate Monte Carlo estimate is

$$\mathrm{MSE}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \frac{1}{n}\,\mathrm{Var}\left(\phi(X) - \psi(X)\right).$$

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.3.3.

6 Antithetic variables I

6.1 Estimation with correlation

This lecture and the next, we will be looking at our second variance reduction method for Monte Carlo estimation: the use of antithetic variables." The word "antithetic" refers to using negative correlation to reduce the variance an estimator.

Let's start with the simple example of estimating an expectation from $n = 2$ samples. Suppose Y has expectation $\mu = \mathbb{E}Y$ and variance $\mathrm{Var}(Y) = \sigma^2$. Suppose Y_1 and Y_2 are independent samples from Y. Then the Monte Carlo estimator is

$$\overline{Y} = \tfrac{1}{2}(Y_1 + Y_2).$$

This estimator is unbiased, since

$$\mathbb{E}\overline{Y} = \mathbb{E}(\tfrac{1}{2}(Y_1 + Y_2)) = \tfrac{1}{2}(\mathbb{E}Y_1 + \mathbb{E}Y_2) = \tfrac{1}{2}(\mu + \mu) = \mu.$$

Thus the mean-square error equals the variance, which is

$$\mathrm{Var}\,(\overline{Y}) = \mathrm{Var}\,(\tfrac{1}{2}(Y_1 + Y_2)) = \tfrac{1}{4}(\,\mathrm{Var}(Y_1) + \mathrm{Var}(Y_2)) = \tfrac{1}{4}(\sigma^2 + \sigma^2) = \tfrac{1}{2}\sigma^2.$$

But what if Y_1 and Y_2 still have the same distribution as Y but now are *not* independent? The expectation is still the same, so the estimator is still unbiased. But the variance (and hence mean-square error) is now

$$\mathrm{Var}\,(\overline{Y}) = \mathrm{Var}\,(\tfrac{1}{2}(Y_1 + Y_2)) = \tfrac{1}{4}(\,\mathrm{Var}(Y_1) + \mathrm{Var}(Y_2) + 2\,\mathrm{Cov}(Y_1, Y_2)).$$

Write ρ for the correlation

$$\rho = \mathrm{Corr}(Y_1, Y_2) = \frac{\mathrm{Cov}(Y_1, Y_2)}{\sqrt{\mathrm{Var}(Y_1)\,\mathrm{Var}(Y_2)}} = \frac{\mathrm{Cov}(Y_1, Y_2)}{\sqrt{\sigma^2 \times \sigma^2}} = \frac{\mathrm{Cov}(Y_1, Y_2)}{\sigma^2}.$$

(Remember that $-1 \le \rho \le +1$.) Then the variance is

$$\mathrm{Var}\,(\overline{Y}) = \tfrac{1}{4}(\sigma^2 + \sigma^2 + 2\rho\sigma^2) = \frac{1+\rho}{2}\,\sigma^2.$$

We can compare this with the variance $\frac{1}{2}\sigma^2$ from the independent-sample case:

- If Y_1 and Y_2 are **positively correlated**, in that $\rho > 0$, then the variance, and hence the mean-square error, has got bigger. This means the estimator is worse. This is because, with positive correlation, errors compound each other – if one sample is bigger than average, then the other one is likely to be bigger than average too; while if one sample is smaller than average, then the other one is likely to be smaller than average too.
- If Y_1 and Y_2 are **negatively correlated**, in that $\rho < 0$, then the variance, and hence the mean-square error, has got smaller. This means the estimator is better. This is because, with negative correlation, errors compensate for each other – if one sample is bigger than average, then the other one is likely to be smaller than average, which will help "cancel out" the error.

6.2 Monte Carlo with antithetic variables

We have seen that negative correlation helps improve estimation from $n = 2$ samples. How can we make this work in our favour for Monte Carlo simulation with many more samples?

We will look at the idea of **antithetic pairs**. So instead of taking n samples

$$X_1, X_2, \dots, X_n$$

that are all independent of each other, we will take $n/2$ pairs of samples

$$(X_1, X_1'), (X_2, X_2'), \dots, (X_{n/2}, X_{n/2}').$$

(Here, $n/2$ pairs means n samples over all.) *Within* each pair, X_i and X_i' will *not* be independent, but *between* different pairs $i \neq j$, (X_i, X_i') and (X_j, X_j') *will* be independent.

Definition 6.1. Let X be a random variable, ϕ a function, and write $\theta = \mathbb{E}\,\phi(X)$. Let X' have the same distribution as X (but not necessarily be independent of it). Suppose that (X_1, X_1'), (X_2, X_2'), ..., $(X_{n/2}, X_{n/2}')$ are pairs of random samples from (X, X'). Then the **antithetic variables Monte Carlo estimator** $\hat{\theta}_n^{\mathrm{AV}}$ of θ is

$$\hat{\theta}_n^{\mathrm{AV}} = \frac{1}{n} \sum_{i=1}^{n/2} (\phi(X_i) + \phi(X_i')).$$

The expression above for $\hat{\theta}_n^{\mathrm{AV}}$ makes it clear that that this is a mean of the sum from each pair. Alternatively, we can rewrite the estimator as

$$\hat{\theta}_n^{\mathrm{AV}} = \frac{1}{2} \left(\frac{1}{n/2} \sum_{i=1}^{n/2} \phi(X_i) + \frac{1}{n/2} \sum_{i=1}^{n/2} \phi(X_i') \right),$$

which highlights that it is the mean of the estimator from the X_is and the the estimator from the X_i's.

6.3 Examples

Example 6.1. Recall Example 2.1 (continued in Example 3.2 and Example 4.1). Here, we were estimating $\mathbb{P}(Z > 2)$ for Z a standard normal.

The basic Monte Carlo estimate was

```
n <- 1e6
samples <- rnorm(n)
MCest   <- mean(samples > 2)
MCest
```

```
[1] 0.022723
```

Can we improve this estimate with an antithetic variable? Well, if Z is a standard normal, then $Z' = -Z$ is also standard normal and is not independent of Z. So maybe that could work as an antithetic variable. Let's try

```
n <- 1e6
samples1 <- rnorm(n / 2)
samples2 <- -samples1
AVest <- (1 / n) * sum((samples1 > 2) + (samples2 > 2))
AVest
```

```
[1] 0.022723
```

Example 6.2. Let's consider estimating $\mathbb{E} \sin U$, where U is continuous uniform on $[0, 1]$.

The basic Monte Carlo estimate is

```
n <- 1e6
samples <- runif(n)
MCest <- mean(sin(samples))
MCest
```

```
[1] 0.4598663
```

We used `runif(n, min, max)` to generate n samples on the interval [`min`, `max`]. However, if you omit the `min` and `max` arguments, then R assumes the default values `min = 0`, `max = 1`, which is what we want here.

If U is uniform on $[0, 1]$, then $1 - U$ is also uniform on $[0, 1]$. We could try using that as an antithetic variable.

```r
n <- 1e6
samples1 <- runif(n / 2)
samples2 <- 1 - samples1
AVest <- (1 / n) * sum(sin(samples1) + sin(samples2))
AVest
```

```
[1] 0.4596694
```

Are these antithetic variables estimates an improvement on the basic Monte Carlo estimate? We'll find out next time.

Next time: *We continue our study of the antithetic variables method with more examples and analysis of the error.*

Summary:

- Estimation is helped by combining individual estimates that are negatively correlated.
- For antithetic variables Monte Carlo estimation, we take pairs of non-independent variables (X, X'), to get the estimator

$$\hat{\theta}_n^{\mathrm{AV}} = \frac{1}{n} \sum_{i=1}^{n/2} (\phi(X_i) + \phi(X_i')).$$

On **Problem Sheet 1**, you should now be able to answer all questions. You should work through this problem sheet in advance of the problems class on *Thursday 17 October*.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.3.2.

Problem Sheet 1

This is Problem Sheet 1, which covers material from Lectures 1 to 6. You should work through all the questions on this problem sheet in advance of the problems class, which takes place in the lecture of **Thursday 16 October**.

This problem sheet is to help you practice material from the module and to help you check your learning. It is *not* for formal assessment and does not count towards your module mark.

However, if, optionally, you would like some brief informal feedback on **Questions 4, 6 and 8** (marked), I am happy to provide some. If you want some brief feedback, you should submit your work electronically through Gradescope via the module's Minerva page by **1400 on Tuesday 14 October**. (If you hand-write solutions on paper, the easiest way to scan-and-submit that work is using the Gradescope app on your phone.) I will return some brief comments on your those two questions by the problems class on Thursday 16 October. Because this informal feedback, and not part of the official assessment, I cannot accept late work for any reason – but I am always happy to discuss any of your work on any question in my office hours.

Many of these questions will require use of the R programming language (for example, by using the program RStudio).

Full solutions will be released on Friday 17 October.

7 Antithetic variables II

7.1 Error with antithetic variables

Recall from last time the antithetic variables Monte Carlo estimator. We take sample pairs

$$(X_1, X_1'), (X_2, X_2'), \dots, (X_{n/2}, X_{n/2}'),$$

where samples are independent between different pairs but *not* independent within the same pair. The estimator of $\theta = \mathbb{E}\,\phi(X)$ is

$$\hat{\theta}_n^{\mathrm{AV}} = \frac{1}{n} \sum_{i=1}^{n/2} (\phi(X_i) + \phi(X_i')).$$

We hope this is better than the standard Monte Carlo estimator if $\phi(X)$ and $\phi(X')$ are negatively correlated.

Theorem 7.1. *Let X be a random variable, ϕ a function, and $\theta = \mathbb{E}\,\phi(X)$. Let X' have the same distribution as X, and write $\rho = \mathrm{Corr}(\phi(X_i), \phi(X_i'))$. Let*

$$\hat{\theta}_n^{\mathrm{AV}} = \frac{1}{n} \sum_{i=1}^{n/2} (\phi(X_i) + \phi(X_i'))$$

be the antithetic variables Monte Carlo estimator of θ. Then:

1. *$\hat{\theta}_n^{\mathrm{AV}}$ is unbiased, in that $\mathrm{bias}\,(\hat{\theta}_n^{\mathrm{AV}}) = 0$.*
2. *The variance of of $\hat{\theta}_n^{\mathrm{AV}}$ is*

$$\mathrm{Var}\,(\hat{\theta}_n^{\mathrm{AV}}) = \frac{1}{2n} \mathrm{Var}\,(\phi(X) + \phi(X')) = \frac{1+\rho}{n} \mathrm{Var}\,(\phi(X)).$$

3. *The mean-square error of $\hat{\theta}_n^{\mathrm{AV}}$ is*

$$\mathrm{MSE}\,(\hat{\theta}_n^{\mathrm{AV}}) = \frac{1}{2n} \mathrm{Var}\,(\phi(X) + \phi(X')) = \frac{1+\rho}{n} \mathrm{Var}\,(\phi(X)).$$

4. *The root-mean-square error of* $\hat{\theta}_n^{\mathrm{AV}}$ *is*

$$\mathrm{RMSE}\left(\hat{\theta}_n^{\mathrm{AV}}\right) = \frac{1}{\sqrt{2n}}\sqrt{\mathrm{Var}\left(\phi(X)+\phi(X')\right)} = \frac{\sqrt{1+\rho}}{\sqrt{n}}\sqrt{\mathrm{Var}\left(\phi(X)\right)}.$$

In points 2, 3 and 4, generally the first expression, involving the variance $\mathrm{Var}(\phi(X)+\phi(X'))$, is the most convenient for computation. We can estimate this easily from data using the sample variance in the usual way (as we will in the examples below).

The second expression, involving the correlation ρ, is usually clearer for understanding. Comparing these to the same results for the standard Monte Carlo estimator (Theorem 3.2), we see that the antithetic variables method is an improvement (that is, has a smaller mean-square error) when $\rho < 0$, but is worse when $\rho > 0$. This proves that negative correlation improves our estimator.

Proof. For unbiasedness, we have

$$\mathbb{E}\hat{\theta}_n^{\mathrm{AV}} = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n/2}\left(\phi(X_i)+\phi(X_i')\right)\right) = \frac{1}{n}\,\frac{n}{2}\left(\mathbb{E}\,\phi(X)+\mathbb{E}\,\phi(X')\right) = \frac{1}{2}(\theta+\theta) = \theta,$$

since X' has the same distribution as X.

For the other three points, each of the first expressions follows straightforwardly in essentially the same way. (You can fill in the details yourself, if you need to.) For the second expressions, we have

$$\begin{aligned}
\mathrm{Var}\left(\phi(X)+\phi(X')\right) &= \mathrm{Var}\left(\phi(X)\right) + \mathrm{Var}\left(\phi(X')\right) + 2\,\mathrm{Cov}\left(\phi(X),\phi(X')\right) \\
&= \mathrm{Var}\left(\phi(X)\right) + \mathrm{Var}\left(\phi(X')\right) + 2\rho\sqrt{\mathrm{Var}\left(\phi(X)\right)\mathrm{Var}\left(\phi(X')\right)} \\
&= \mathrm{Var}\left(\phi(X)\right) + \mathrm{Var}\left(\phi(X)\right) + 2\rho\sqrt{\mathrm{Var}\left(\phi(X)\right)\mathrm{Var}\left(\phi(X)\right)} \\
&= 2(1+\rho)\,\mathrm{Var}\left(\phi(X)\right).
\end{aligned}$$

The results then follow. □

Let's return to the two examples we tried last time.

Example 7.1. In Example 6.1, we were estimating $\mathbb{P}(Z > 2)$ for Z a standard normal.

The basic Monte Carlo estimate and its root-mean-square error are

```
n <- 1e6
samples <- rnorm(n)
MCest   <- mean(samples > 2)
MC_MSE <- var(samples > 2) / n
c(MCest, sqrt(MC_MSE))
```

```
[1] 0.0229120000 0.0001496231
```

We then used $Z' = -Z$ as an antithetic variable. its root-mean-square error are

```
n <- 1e6
samples1 <- rnorm(n / 2)
samples2 <- -samples1
AVest <- (1 / n) * sum((samples1 > 2) + (samples2 > 2))
AV_MSE <- var((samples1 > 2) + (samples2 > 2)) / (2 * n)
c(AVest, sqrt(AV_MSE))
```

```
[1] 0.0226630000 0.0001470912
```

This looked like it made very little difference – perhaps a small improvement. This can be confirmed by looking at the sample correlation with R's `cor()` function.

```
cor(samples1 > 2, samples2 > 2)
```

```
[1] -0.02329094
```

We see there was a very small but negative correlation: the variance, and hence the mean-square error, was reduced by about 2%.

Example 7.2. In Example 6.2, we were estimating $\mathbb{E} \sin U$, where U is continuous uniform on $[0, 1]$.

The basic Monte Carlo estimate and its root-mean square error is

```
n <- 1e6
samples <- runif(n)
MCest <- mean(sin(samples))
MC_MSE <- var(sin(samples)) / n
c(MCest, sqrt(MC_MSE))
```

```
[1] 0.4596343810 0.0002477225
```

We then used $U' = 1 - U$ as an antithetic variable

```r
n <- 1e6
samples1 <- runif(n / 2)
samples2 <- 1 - samples1
AVest <- (1 / n) * sum(sin(samples1) + sin(samples2))
AV_MSE <- var(sin(samples1) + sin(samples2)) / (2 * n)
c(AVest, sqrt(AV_MSE))
```

```
[1] 4.596807e-01 2.483595e-05
```

This time, we see a big improvement: the root-mean-square error has gone down by a whole order of magnitude, from 2×10^{-4} to 2×10^{-5}. It would normally take 100 times as many samples to reduce the RMSE by a factor of 10, but we've got the extra 99 million samples for free by using antithetic variables!

The benefit here can be confirmed by looking at the sample correlation.

```r
cor(sin(samples1), sin(samples2))
```

```
[1] -0.9899549
```

That's a very large negative correlation, which shows why the antithetic variables made such a huge improvement.

7.2 Finding antithetic variables

Antithetic variables can provide a huge advantage compared to standard Monte Carlo, as we saw in the second example above. The downside is that it can often be difficult to *find* an appropriate antithetic variable.

To even be able to *try* the antithetic variables method, we need to find a random variable X' with the same distribution as X that isn't merely an independent copy. Both the examples we have seen of this use a symmetric distribution; that is, a distribution X such that $X' = a - X$ has the same distribution as X, for some a.

- We saw that if $X \sim \mathrm{N}(0, 1)$ is a standard normal distribution, then $X' = -X \sim \mathrm{N}(0, 1)$ too. More generally, if $X \sim \mathrm{N}(\mu, \sigma^2)$, then $X' = 2\mu - X \sim \mathrm{N}(\mu, \sigma^2)$ can be tried as an antithetic variable.
- We saw that if $U \sim \mathrm{U}[0, 1]$ is a continuous uniform distribution on $[0, 1]$, then $U' = 1 - U \sim \mathrm{U}[0, 1]$ too. More generally, if $X \sim \mathrm{U}[a, b]$, then $X' = (a + b) - X \sim \mathrm{U}[a, b]$ can be tried as an antithetic variable.

Later, when we study the inverse transform method (in Lecture 13) we will see another, more general, way to generate antithetic variables.

But to be a *good* antithetic variable, we need $\phi(X)$ and $\phi(X')$ to be negatively correlated too – preferably strongly so. Often, this is a matter of trial-and-error – it's difficult to set out hard principles. But there are some results that try to formalise the idea that "nice functions of negatively correlated random variables are themselves negatively correlated", which can be useful. We give one example of such a result here.

Theorem 7.2. *Let $U \sim \mathrm{U}[0,1]$ and $V = 1-U$. Let ϕ be a monotonically increasing function. Then $\phi(U)$ and $\phi(V)$ are negatively correlated, in that* $\mathrm{Cov}\big(\phi(U), \phi(V)\big) \leq 0$.

I didn't get to this proof in the lecture and it's a bit tricky (although not very technically deep), so let's say it's non-examinable.

Proof. [Non-examinable] You probably already know two different expressions for the covariance:

$$\mathrm{Cov}(Y,Z) = \mathbb{E}(Y-\mu_Y)(Z-\mu_Z) = \mathbb{E}\,YZ - \mu_Y\mu_Z.$$

But for this proof it will be helpful to use a third, less well-known equation:

$$\mathrm{Cov}(Y,Z) = \tfrac{1}{2}\,\mathbb{E}(Y-Y')(Z-Z'),$$

where (Y',Z') is an IID copy of (Y,Z).

To see that this third expression is true, we can start by expanding out the brackets. We get

$$\tfrac{1}{2}\,\mathbb{E}(Y-Y')(Z-Z') = \tfrac{1}{2}(\,\mathbb{E}\,YZ - \mathbb{E}\,YZ' - \mathbb{E}\,Y'Z + \mathbb{E}\,Y'Z').$$

We have four terms to deal with. The first has no dashed variables, so can stay as it is. The second and third terms have one dashed and one non-dashed variable, so these are independent, and we can write $\mathbb{E}\,YZ' = \mathbb{E}\,Y'Z = \mu_Y\mu_Z$. The fourth term has both terms dashed, but these have the same distribution as if they were non-dashed, so $\mathbb{E}\,Y'Z' = \mathbb{E}\,YZ$. All together, we have

$$\tfrac{1}{2}\,\mathbb{E}(Y-Y')(Z-Z') = \tfrac{1}{2}(2\,\mathbb{E}\,YZ - 2\mu_Y\mu_Z) = \mathbb{E}\,YZ - \mu_Y\mu_Z,$$

which is indeed the second expression for the covariance.

We can now apply this to the theorem in question. Put $Y = \phi(U)$ and $Z = \phi(1-U)$, and introduce an IID copy V of U, so $Y' = \phi(V)$ and $Z' = \phi(1-V)$. Then we have

$$\mathrm{Cov}\,\big(\phi(U),\phi(V)\big) = \tfrac{1}{2}\,\mathbb{E}\,\big(\phi(U)-\phi(V)\big)\big(\phi(1-U)-\phi(1-V)\big).$$

We now claim that this expectation is negative. In fact, we have a stronger result:

$$\big(\phi(U)-\phi(V)\big)\big(\phi(1-U)-\phi(1-V)\big) \tag{7.1}$$

is *always* negative, so its expectation certainly is. To see this, think separately of the two cases $U \leq V$ and $V \leq U$.

- If $U \leq V$, then $\phi(U) \leq \phi(V)$ too, since ϕ is increasing. But, also this means that $1-U \geq 1-V$, so $\phi(1-U) \geq \phi(1-V)$. This means that, in Equation 7.1, the first term is negative and the second term is positive, so the product is negative.
- If $V \leq U$, then $\phi(V) \leq \phi(U)$ too, since ϕ is increasing. But, also this means that $1-V \geq 1-U$, so $\phi(1-V) \geq \phi(1-U)$. This means that, in Equation 7.1, the first term is positive and the second term is negative, so the product is negative.

This completes the proof. □

7.3 A note on sample size comparisons

Throughout these two lectures, when using antithetic pairs, we have taken $n/2$ pairs of samples. This is because that means we have $n/2 \times 2 = n$ samples all together, which seems like a fair comparison to usual Monte Carlo with n samples. This is certainly the case if generating the sample and generating its antithetic pair cost roughly the same in terms of time (or energy, or money). This is always how we will compare methods in this module.

However, if generating the first variate of each pair is slow, but then generating the second antithetic variate is much quicker, it might be a fairer comparison to take a full n pairs. This could happen if we use a complicated method (like we will discover later in the module) for generating the X, but then the X' is something similar like $X' = -X$. You could even consider more complicated ways of assessing the "cost" of Monte Carlo estimation, by assigning different costs to generating the original sample and to the antithetic pair, and also a cost to applying the function ϕ; but we won't get into that here.

Next time: *We come to the third, and most important, variance reduction scheme: importance sampling.*

Summary:

- The antithetic variables estimator is unbiased and has mean-square error

$$\mathrm{MSE}\left(\hat{\theta}_n^{\mathrm{AV}}\right) = \frac{1}{2n}\,\mathrm{Var}\left(\phi(X) + \phi(X')\right) = \frac{1+\rho}{n}\,\mathrm{Var}\left(\phi(X)\right).$$

- If $U \sim \mathrm{U}[0,1]$ and ϕ is monotonically increasing, then $\phi(U)$ and $\phi(1-U)$ are negatively correlated.

On Thursday's lecture, we will be discussing your answers to **Problem Sheet 1**.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.3.2.

8 Importance sampling I

8.1 Sampling from other distributions

So far, we have looked at estimating $\mathbb{E}\,\phi(X)$ using samples $X_1, X_2, \ldots, X_n$ that are from the same distribution as X. **Importance sampling** is based on the idea of taking samples $Y_1, Y_2, \ldots, Y_n$ from some *different* distribution Y, but then making an appropriate adjustment, so that we're still estimating $\mathbb{E}\,\phi(X)$.

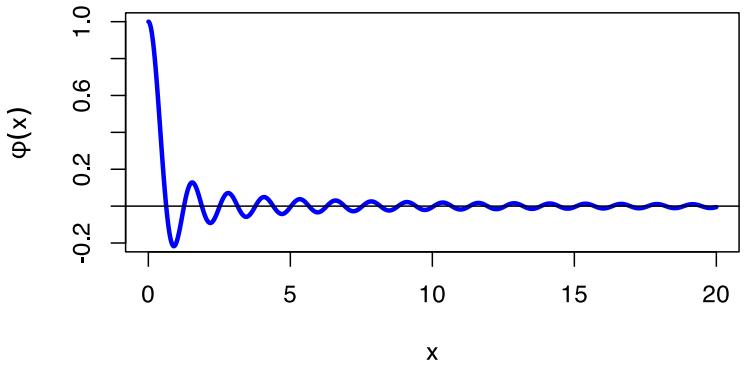
Why might we want to do this? There are two main reasons:

- First, we might not be able to sample from X, so we might be forced into sampling from some other distribution Y instead. So far, X has always been a nice pleasant distribution, like a normal, exponential or continuous uniform distribution, for which we can use R's built-in sampling function. But what if X were instead a very unusual or awkward distribution? In that case, we might not be able to sample directly from X, so would be forced into sampling from a different distribution.
- Second, we might *prefer* to sample from a distribution other than Y. This might be the case if $\phi(x)$ varies a lot over different values of x. There might be some areas of x where it's very important to get an accurate estimation, because they contribute a lot to $\mathbb{E}\,\phi(X)$, so we'd like to "oversample" (take lots of samples) there; meanwhile, other areas of x where it is not very important to get an accurate estimation, because they contribute very little to $\mathbb{E}\,\phi(X)$, so we don't mind "undersampling" (taking relatively few samples) there. Then we could sample instead from a distribution Y that concentrates on the most important areas for ϕ; although we'll need to make sure to adjust our estimator by "down-weighting" the places that we have oversampled.

Consider, for example, trying to estimate $\mathbb{E}\,\phi(X)$ where X is uniform on $[0, 20]$ and ϕ is the function shown below.

```
phi <- function(x) sin(5 * x) / (5 * x)
curve(
  phi, n = 10001, from = 0, to = 20,
  lwd = 3, col = "blue",
```

```
  xlab = "x", ylab = expression(phi(x)), ylim = c(-0.2, 1)
)
abline(h = 0)
```



We can see that what happens for small x – say, for x between 0 and 2, or so – will have an important effect on the value of $\mathbb{E}\,\phi(X)$, because that where ϕ has the biggest (absolute) values. But what happens for large x – say for $x \geq 10$ or so – will be much less important for estimating $\mathbb{E}\,\phi(X)$. So it seems wasteful to have all values in $[0, 20]$ to be sampled equally, and it would seem to make sense to take more samples from small values of x.

This is all very well in practice, but how exactly should we down-weight those over-sampled areas?

Think about estimating $\mathbb{E}\,\phi(X)$. Let's assume that X is continuous with probability density function f. (Throughout this lecture and the next, we will assume all our random variables are continuous. The arguments for discrete random variables are very similar – just swap probability density functions with probability mass functions and integrals with sums. You can fill in the details yourself, if you like.) Then we are trying to estimate

$$\mathbb{E}\,\phi(X) = \int_{-\infty}^{+\infty} \phi(x)\, f(x)\, \mathrm{d}x = \int_{-\infty}^{+\infty} \phi(y)\, f(y)\, \mathrm{d}y.$$

(In the second equality, we merely changed the "dummy variable" from x to y, as we are at liberty to do.)

Now suppose we sample from some other continuous distribution Y, with PDF g. If we estimate $\mathbb{E}\,\psi(Y)$, say, for some function ψ, then we are estimating

$$\mathbb{E}\,\psi(Y) = \int_{-\infty}^{+\infty} \psi(y)\, g(y)\, \mathrm{d}y = \int_{-\infty}^{+\infty} \psi(x)\, g(x)\, \mathrm{d}x.$$

But we want to be estimating $\mathbb{E}\,\phi(X)$, not $\mathbb{E}\,\psi(Y)$. So we will need to pick ψ such that

$$\mathbb{E}\,\phi(X) = \int_{-\infty}^{+\infty} \phi(y)\, f(y)\,\mathrm{d}y = \int_{-\infty}^{+\infty} \psi(y)\, g(y)\,\mathrm{d}y = \mathbb{E}\,\psi(Y).$$

So we need to pick ψ such that $\phi(y)\, f(y) = \psi(y)\, g(y)$. That means that we should take

$$\psi(y) = \frac{\phi(y) f(y)}{g(y)} = \frac{f(y)}{g(y)}\,\phi(y).$$

So we could build a Monte Carlo estimate for $\mathbb{E}\,\phi(X)$ instead as a Monte Carlo estimate for

$$\mathbb{E}\,\psi(Y) = \mathbb{E}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

There is one other thing: we need to be careful of division by 0 errors. So we should make sure that g is only 0 when f is 0. In other words, if it's possible for X to take some value, then it must be possible for Y to take that value too.

We are finally ready to define our estimator.

Definition 8.1. Let X be a continuous random variable with probability density function f, let ϕ be a function, and write $\theta = \mathbb{E}\,\phi(X)$. Let Y be a continuous random variable with probability desnity function g, where $g(y) > 0$ for all y where $f(y) > 0$. Then the **importance sampling Monte Carlo estimator** $\widehat{\theta}_n^{\mathrm{IS}}$ of θ is

$$\widehat{\theta}_n^{\mathrm{IS}} = \frac{1}{n}\sum_{i=1}^{n} \frac{f(Y_i)}{g(Y_i)}\,\phi(Y_i),$$

where $Y_1, Y_2, \ldots, Y_n$ are independent random samples from Y.

We can think of this as taking a weighted mean of the $\phi(Y_i)$s, where the weights are $f(Y_i)/g(Y_i)$. So if a value y is more likely under Y than under X, then $g(y)$ is big compared to $f(y)$, so $f(y)/g(y)$ is small, and y gets a low weight. If a value y is less likely under Y than under X, then $g(y)$ is small compared to $f(y)$, so $f(y)/g(y)$ is big, and it gets a high weight. Thus we see that the weighting compensates for values that are likely to be over- or under-sampled.

8.2 Example

Example 8.1. Let $X \sim \mathrm{N}(0,1)$ be a standard normal. Suppose we want to estimate $\mathbb{P}(X > 4)$. We could do this the standard Monte Carlo way by sampling from X itself.

$$\widehat{\theta}_n^{\mathrm{MS}} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{[4,\infty)}(X_i).$$

However, this will not be a good estimator. To see the problem, lets run this with $n = 100\,000 = 10^5$ samples, but do it 10 times, and see what all the estimates are.

```
n <- 1e5
MCest <- rep(0, 10)
for (i in 1:10) MCest[i] <- mean(rnorm(n) > 4)
MCest
```

```
 [1] 4e-05 3e-05 3e-05 7e-05 1e-05 5e-05 6e-05 1e-05 5e-05 1e-05
```

We see a big range of values. I get different results each time I run it, but anything between 1×10^{-5} and 8×10^{-5}, and even 0, comes out fairly regularly as the estimate. The problem is that $X > 4$ is a very rare event – it only comes out a handful of times (perhaps 0 to 8) out of the 100,000 samples. This means our estimate is (on average) quite inaccurate.

It would be better not to sample from X, but rather to sample from a distribution that is greater than 4 a better proportion of the time. We could try anything for this distribution Y, but to keep things simple, I'm going to stick with a normal distribution with standard deviation 1. I'll want to increase the mean, though, so that we sample values bigger than 4 more often. Let's try importance sampling with $Y \sim \mathrm{N}(4, 1)$.

The PDFs of $X \sim \mathrm{N}(0, 1)$ and $Y \sim \mathrm{N}(4, 1)$ are

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\tfrac{1}{2}x^2\right) \qquad g(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\tfrac{1}{2}(y-4)^2\right),$$

so the relevant weighting of a sample y is

$$\frac{f(y)}{g(y)} = \frac{\exp\left(-\tfrac{1}{2}y^2\right)}{\exp\left(-\tfrac{1}{2}(y-4)^2\right)} = \exp\left(\tfrac{1}{2}(-y^2 + (y-4)^2)\right) = \exp(-4y + 8).$$

So our importance sampling estimate will be

$$\hat{\theta}_n^{\mathrm{IS}} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{e}^{-4Y_i + 8} \mathbb{I}_{[4,\infty)}(Y_i).$$

Let's try this in R. Although we could use the function e^{-4y+8} for the weights, I'll do this by using the ratios of the PDFs directly in R (just in case I made a mistake…).

```
n <- 1e5
pdf_x <- function(y) dnorm(y, 0, 1)
pdf_y <- function(y) dnorm(y, 4, 1)
samples_y <- rnorm(n, 4, 1)
ISest <- mean((pdf_x(samples_y) / pdf_y(samples_y)) * (samples_y > 4))
ISest
```

```
[1] 3.163129e-05
```

8.3 Errors in importance sampling

The following theorem should not by now be a surprise.

Theorem 8.1. *Let X be a continuous random variable with probability density function f, let ϕ be a function, and write $\theta = \mathbb{E}\,\phi(X)$. Let Y another continuous random variable with probability density function with probability density function g, such that $g(y) = 0$ only when $f(y) = 0$. Let*

$$\widehat{\theta}_n^{\mathrm{IS}} = \frac{1}{n}\sum_{i=1}^{n} \frac{f(Y_i)}{g(Y_i)}\,\phi(Y_i)$$

be the importance sampling Monte Carlo estimator of θ. Then:

1. *$\widehat{\theta}_n^{\mathrm{IS}}$ is unbiased, in that $\mathrm{bias}\,(\widehat{\theta}_n^{\mathrm{IS}}) = 0$.*
2. *The variance of of $\widehat{\theta}_n^{\mathrm{IS}}$ is*

$$\mathrm{Var}\,(\widehat{\theta}_n^{\mathrm{IS}}) = \frac{1}{n}\,\mathrm{Var}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

3. *The mean-square error of $\widehat{\theta}_n^{\mathrm{IS}}$ is*

$$\mathrm{MSE}\,(\widehat{\theta}_n^{\mathrm{IS}}) = \frac{1}{n}\,\mathrm{Var}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

4. *The root-mean-square error of $\widehat{\theta}_n^{\mathrm{IS}}$ is*

$$\mathrm{RMSE}\,(\widehat{\theta}_n^{\mathrm{IS}}) = \frac{1}{\sqrt{n}}\sqrt{\mathrm{Var}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right)}.$$

Proof. Part 1 follows essentially the same argument as our discussion at the beginning of this lecture. We have

$$\mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n}\frac{f(Y_i)}{g(Y_i)}\,\phi(Y_i)\right) = \frac{1}{n}\,n\,\mathbb{E}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right) = \mathbb{E}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

But

$$\mathbb{E}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right) = \int_{-\infty}^{+\infty}\frac{f(y)}{g(y)}\,\phi(y)\,g(y)\,\mathrm{d}y = \int_{-\infty}^{+\infty}\phi(y)\,f(y)\,\mathrm{d}y = \mathbb{E}\,\phi(X).$$

This last step is because f is the PDF of X; it doesn't matter whether the dummy variable in the integration is x or y. Hence the estimator is unbiased.

Parts 2 to 4 follow in the usual way. □

As we are now used to, we can estimate the variance using the sample variance.

Example 8.2. We continue Example 8.1, where we are estimating $\mathbb{P}(X > 4)$ for $X \sim \mathrm{N}(0, 1)$.

For the standard Monte Carlo method, we estimate the root-mean-square error as

```
n <- 1e5
MC_MSE <- var(rnorm(n) > 4) / n
sqrt(MC_MSE)
```

```
[1] 1.732033e-05
```

As before, this still varies a lot, but it seems to usually be about 2×10^{-5}.

For the importance sampling method, we estimate the mean-square error as

```
n <- 1e5
pdf_x <- function(x) dnorm(x, 0, 1)
pdf_y <- function(y) dnorm(y, 4, 1)
samples_y <- rnorm(n, 4, 1)
IS_MSE <- var((pdf_x(samples_y) / pdf_y(samples_y)) * (samples_y > 4)) / n
sqrt(IS_MSE)
```

```
[1] 2.129446e-07
```

This is about 2×10^{-7}. This is about 100 times smaller than for the standard method: equivalent to taking about 10,000 times as many samples! That's a huge improvement, which demonstrates the power of importance sampling.

Next time: *We continue our study of importance sampling – and complete our study of Monte Carlo estimation, for now – by considering how to pick a good distribution Y.*

Summary:

- Importance sampling estimates $\mathbb{E}\,\phi(X)$ by sampling from a different distribution Y.
- The importance sampling estimator is $\widehat{\theta}_n^{\mathrm{IS}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} \frac{f(Y_i)}{g(Y_i)}\,\phi(Y_i)$.
- The importance sampling estimator is unbiased with mean-square error

$$\mathrm{MSE}\left(\widehat{\theta}_n^{\mathrm{IS}}\right) = \frac{1}{n}\,\mathrm{Var}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

Solutions are now available for Problem Sheet 1.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.3.1.

9 Importance sampling II

9.1 Picking a good distribution

Let's remind ourselves where we've got to on importance sampling.

- We want to estimate $\mathbb{E}\,\phi(X)$.
- Rather than sampling from X, with PDF f, we instead sample from a different distribution Y, with PDF g.
- The estimator is $\widehat{\theta}_n^{\mathrm{IS}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} \frac{f(Y_i)}{g(Y_i)}\,\phi(Y_i)$.

We've seen that importance sampling can be a very powerful tool, when used well. But how should pick a good distribution Y to sample from?

Let's examine the mean-square error more carefully:

$$\mathrm{MSE}\,(\widehat{\theta}_n^{\mathrm{IS}}) = \frac{1}{n}\,\mathrm{Var}\left(\frac{f(Y)}{g(Y)}\,\phi(Y)\right).$$

So our goal is to try and pick Y such that $\frac{f(Y)}{g(Y)}\phi(Y)$ has low variance. We also, of course, want to be able to sample from Y.

The best possible choice, then, would be to pick Y such that $\frac{f(Y)}{g(Y)}\phi(Y)$ is constant – and therefore has zero variance! If ϕ is non-negative, then it seems like we should pick Y such that its probability density function is $g(y) \propto f(y)\phi(y)$. (Here, $\propto$ is the "proportional to" symbol.) That is, to have

$$g(y) = \frac{1}{Z} f(y)\phi(y),$$

for some constant Z. Then $\frac{f(Y)}{g(Y)}\phi(Y) = Z$ is a constant, has zero variance, and we have a perfect estimator!

What is this constant Z? Well, g is a PDF, so it has to integrate to 1. So we will need to have

$$1 = \int_{-\infty}^{+\infty} g(y)\,\mathrm{d}y = \int_{-\infty}^{+\infty} \frac{1}{Z} f(y)\phi(y)\,\mathrm{d}y = \frac{1}{Z}\int_{-\infty}^{+\infty} f(x)\phi(x)\,\mathrm{d}x = \frac{1}{Z}\,\mathbb{E}\,\phi(X).$$

(We did the "switching the dummy variable from y to x" thing again.) So $Z = \mathbb{E}\,\phi(X)$. But that's no good: $\theta = \mathbb{E}\,\phi(X)$ was the thing we were trying to estimate in the first place. If we knew that, we wouldn't have to do Monte Carlo estimation to start with!

So, as much as we would like to, we can't use this "perfect" ideal distribution Y. More generally, if ϕ is not always non-negative, it can be shown that $g(y) \propto f(y)\,|\phi(y)| = |f(x)\,\phi(x)|$ would be the best possible distribution, but this has the same problems.

However, we can still be guided by this idea – we would like $g(y)$ to be as close to proportional to $f(y)\phi(y)$ (or $|f(y)\phi(y)|$) as we can manage, so that $\frac{f(y)}{g(y)}\phi(y)$ is close to being constant, so hopefully has low variance. This tells us that Y should be likely – that is, $g(y)$ should be big – where both f and $|\phi|$ are both big – that is, where X is likely and also ϕ is big in absolute value. While Y should be unlikely where both X is unlikely and ϕ is small in absolute value.

Example 9.1. Let's look again at Example 8.1 (continued in Example 8.2), where we wanted to estimate $\mathbb{P}(X > 4) = \mathbb{E}\,\mathbb{I}_{(4,\infty)}(X)$ for $X \sim \mathrm{N}(0, 1)$. We found are estimator was enormously improved when we used instead $Y \sim \mathrm{N}(4, 1)$.

In the figure below, the blue line is

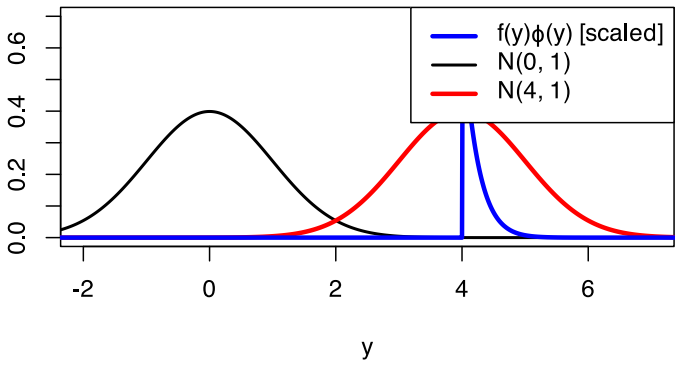
$$f(y)\,\phi(y) = f(y)\,\mathbb{I}_{(4,\infty)}(y) = \begin{cases} \dfrac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-y^2/2} & y > 4 \\ 0 & y \le 4 \end{cases}$$

(scaled up, otherwise it would be so close to the axis line you wouldn't see it).

The black line is the PDF $f(y)$ of the original distribution $X \sim \mathrm{N}(0, 1)$, while the red line is the PDF $g(y)$ of our importance distribution $Y \sim \mathrm{N}(4, 1)$.

```
curve(
  dnorm(x, 0, 1), n = 1001, from = -2.5, to = 7.5,
  col = "black", lwd = 2,
  xlim = c(-2, 7), xlab = "y", ylim = c(0, 0.7), ylab = ""
)
curve(
  dnorm(x, 4, 1), n = 1001, from = -2.5, to = 7.5,
  add = TRUE, col = "red", lwd = 3,
)
curve(
```

```
  dnorm(x, 0, 1) * (x > 4) * 5000, n = 1001, from = -2.5, to = 7.5,
  add = TRUE, col = "blue", lwd = 3
)
legend(
  "topright",
  c(expression(paste("f(y)", varphi, "(y) [scaled]")), "N(0, 1)", "N(4, 1)"),
  lwd = c(3, 2, 3), col = c("blue", "black", "red")
)
```



We have noted that a good distribution will have a PDF that is big when $f(x)\phi(x)$ (the blue line) is big. Clearly the red line is much better at this then the black line, which is why the importance sampling method was so much better here.

There's scope to do better here, though. Perhaps an asymmetric distribution with a much more quickly-decaying left-tail might be good – for example, a shifted exponential $4 + \text{Exp}(\lambda)$ might be worth investigating. Or a thinner, spikier distribution, such as a normal with smaller standard deviation. In both cases, though, we have to be careful – because it's the ratio $f(y)/g(y)$, we still have to be a bit careful about what happens when both $f(y)$ and $g(y)$ are small *absolutely*, in case one is *proportionally* much bigger than the other.

Aside from the exact theory, in the absence of any better idea, choosing Y to be "in the same distribution family as X but with different parameters" is often a reasonable thing to try. For example:

- If $X \sim \text{N}(\mu, \sigma^2)$, then try $Y \sim \text{N}(\nu, \sigma^2)$ for some other value ν.
- If $X \sim \text{Exp}(\lambda)$, then try $Y \sim \text{Exp}(\mu)$ for some other value μ.

9.2 Bonus example

Example 9.2. *Let* $X \sim \mathrm{U}[0, 10]$ be an uniform distribution, so $f(x) = \frac{1}{10}$ for $0 \le x \le 10$, and let $\phi(x) = \mathrm{e}^{-|x-8|}$. Estimate $\mathbb{E}\,\phi(X)$.

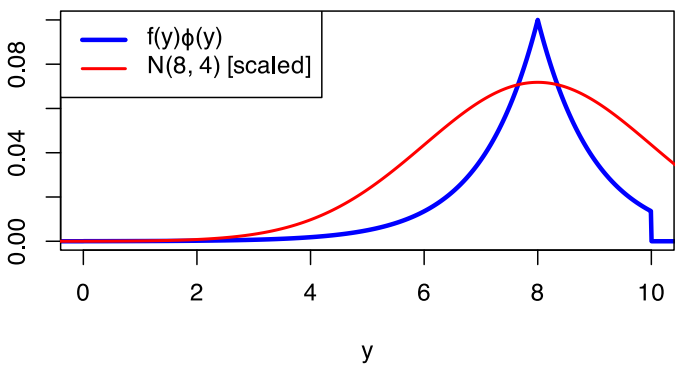
The standard Monte Carlo estimator and its RMSE are as follows

```
phi <- function(x) exp(-abs(x - 8))
n <- 1e6
samples <- runif(n, 0, 10)
MCest <- mean(phi(samples))
MC_MSE <- var(phi(samples)) / n
c(MCest, sqrt(MC_MSE))
```

```
[1] 0.1865579341 0.0002537701
```

Maybe we can improve on this using importance sampling. Let's have a look at a graph of $f(y)\,\phi(y)$ (blue line).

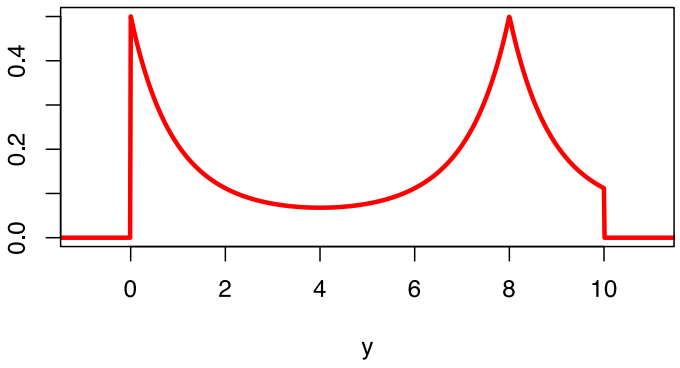
```
curve(
  dunif(x, 0, 10) * exp(-abs(x - 8)), n = 1001, from = -1, to = 11,
  col = "blue", lwd = 3,
  xlim = c(0, 10), xlab = "y", ylab = ""
)
curve(
  dnorm(x, 8, 2)*0.36, n = 1001, from = -1, to = 11,
  add = TRUE, col = "red", lwd = 2,
)
legend(
  "topleft",
  c(expression(paste("f(y)", varphi, "(y)")), "N(8, 4) [scaled]"),
  lwd = c(3, 2), col = c("blue", "red")
)
```



After some experimentation, I decided that $Y \sim \mathrm{N}(8, 2^2)$ (red line; not to same scale) seemed to work quite well, in $g(y)$ being roughly proportional to $f(y)\,\phi(y)$. (Maybe reducing the standard deviation a bit more, to around 1.6, to get the red curve a bit tighter, might have done a little bit better still.)

The following graph shows $\frac{f(y)}{g(y)}\phi(y)$, and shows that the graph is not *too* pointy, so should have reasonably small variance.

```
curve(
  dunif(x, 0, 10) * exp(-abs(x - 8)) / dnorm(x, 8, 2), n = 1001, from = -2, to = 12,
  col = "red", lwd = 3,
  xlim = c(-1, 11), xlab = "y", ylab = ""
)
```



So our importance sampling estimate is as follows.

```
phi <- function(x) exp(-abs(x - 8))
pdf_x <- function(x) dunif(x, 0, 10)
pdf_y <- function(y) dnorm(y, 8, 2)

n <- 1e6
samples <- rnorm(n, 8, 2)
ISest <- mean((pdf_x(samples) / pdf_y(samples)) * phi(samples))
IS_MSE <- var((pdf_x(samples) / pdf_y(samples)) * phi(samples)) / n
c(ISest, sqrt(IS_MSE))
```

```
[1] 0.1863512119 0.0001351446
```

We see that the RMSE has roughly halved, which is the equivalent of taking four times as many samples.

9.3 Summary of Part I

This is our last lecture on Monte Carlo estimation – at least for now, and at least in its standard form. So let's end this section of the module by summarising the estimators we have learned about. We have been learning how to estimate $\theta = \mathbb{E}\,\phi(X)$

- The **standard Monte Carlo estimator** simple takes a sample mean of $\phi(X_i)$, where X_i are independent random samples from X.
- The **control variate** Monte Carlo estimator "anchors" the estimator to some known value $\eta = \mathbb{E}\,\psi(X)$, for a function ψ that is "similar to ϕ, but easier to calculate the expectation exactly".
- The **antithetic variables** Monte Carlo estimator uses pairs of samples (X_i, X_i') that both have the same distribution as X, but where $\phi(X)$ and $\phi(X')$ have negative correlation $\rho < 0$.
- The **importance sampling** Monte Carlo estimator samples not from X, with PDF f, but from a different distribution Y, with PDF Y. The distribution Y is chosen to oversample from the most important values, but then gives lower weight to those samples.

	Estimator	**MSE**
Standard Monte Carlo	$\frac{1}{n}\sum_{i=1}^{n}\phi(X_i)$	$\frac{1}{n}\operatorname{Var}(\phi(X))$

	Estimator	MSE
Control variate	$\frac{1}{n}\sum_{i=1}^{n}(\phi(X_i)-\psi(X_i))+\eta$	$\frac{1}{n}\operatorname{Var}(\phi(X_i)-\psi(X_i))$
Antithetic variables	$\frac{1}{n}\sum_{i=1}^{n/2}(\phi(X_i)+\phi(X_i'))$	$\frac{1}{2n}\operatorname{Var}(\phi(X_i)+\phi(X_i')) = \frac{1+\rho}{n}\operatorname{Var}(\phi(X))$
Importance sampling	$\frac{1}{n}\sum_{i=1}^{n}\frac{f(Y_i)}{g(Y_i)}\phi(X_i)$	$\frac{1}{n}\operatorname{Var}\left(\frac{f(Y_i)}{g(Y_i)}\phi(X_i)\right)$

Next time: *We begin the second section of the module, on random number generation.*

Summary:

- A good importance sampling distribution Y is one whose PDF $g(y)$ is roughly proportional to $|f(y)\,\phi(y)|$. Equivalently, $\frac{f(y)}{g(y)}|\phi(y)|$ is approximately constant.

You should now be able to answer Questions 1–3 on **Problem Sheet 2**.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 3.3.1.

Part II

Random number generation

10 Generating random numbers

Please complete the **mid-semester survey**.

10.1 Why generate random numbers?

So far in this module, we have made a lot of use of generating random numbers or random samples – for us, this has been when performing Monte Carlo estimation. We will also need random numbers for other purposes later in the module. There are lots of other situations in statistics, mathematics, data science, physics, computer science, cryptography, … where we want to use random numbers to help us solve problems. But where do we get these random numbers from?

When performing Monte Carlo estimation, we have used lots of samples from the distribution of some random variable X. We did this using R's built-in functions for random number generation, like `runif()`, `rnorm()`, `rexp()`, and so on.

In this part of the module, we are interested in these questions:

- How do these random number generation functions in R actually *work*?
- What if we want to sample from a distribution for which R doesn't have a built-in function – how can we do that?

It turns out that this will break down into two questions that we can treat largely separately:

1. How do we generate some randomness – any randomness – in the first place? We usually think of this as generating $U \sim \mathrm{U}[0, 1]$, a uniform random number between 0 and 1. (We will look at this today and in the next lecture.)
2. How do we transform that uniform randomness U to get it to behave like the particular distribution X that we want to sample from? (We will look at this in Lectures 12–16.)

10.2 Random numbers on computers

We start by considering the question of how to generate a uniform random number between 0 and 1.

The first thing to know is that computers do not perfectly store exact real numbers in decimals of unending length – that's impossible! Instead, it stores a number to a certain accuracy, in terms of the number of decimal places. To be more precise, computers store numbers in *binary*; that is, written as a sequence of 0s and 1s. These 0s and 1s are called "binary digits", or **bits**, for short. (In the presentation here, we will somewhat simplify matters – computer science experts will be able to spot where I'm lying, or "gently smoothing out the truth".)

A number between 0 and 1 could be (approximately) stored as a 32-bit binary number, for example. A 32-bit number is a number like

```
0. 00110100 11110100 10001111 10011001
```

that is "0." followed by a string of 32 binary digits. A string $0.x_1x_2\cdots x_{31}x_{32}$ represents the number

$$x = \sum_{i=1}^{32} x_i 2^{-i}.$$

So the number above represents

$$\begin{aligned} &0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + \cdots + 0 \times 2^{-31} + 1 \times 2^{-32} && (10.1)\\ &= 2^{-3} + 2^{-4} + 2^{-6} + \cdots + 2^{-29} + 2^{-32} && (10.2)\\ &= 0.20685670361854135990142822265625. && (10.3) \end{aligned}$$

We could generate such a 32-bit (or, more generally, b-bit) number in two different ways:

- A sequence of 32 0s and 1s (each being 50:50 likely to be 0 or 1 and each being independent of the others). This then represents a number in its binary expansion, as above.
  - Or, more generally, we want a string of b 0s and 1s.
- A integer at random between 0 and $2^{32} - 1$, which we can then divide by 2^{32} to get a number between 0 and 1.
  - Or, more generally, we want a random integer between 0 and $m - 1$ for some m, which we can then divide by m. Usually $m = 2^b$ for some b, to give a b-bit number.

There are two ways we can do this. First, we can use **true physical randomness**. Second, we can use **computer-generated "pseudorandomness"**.

True physical randomness means randomness from some real-life random process. For example, you could toss a coin 32 times, and write down "1" each time it lands heads and "0" each time it lands tails: this would give a random 32-bit number. While this will be genuinely random, it is, of course, very slow if you need a large amount of randomness. For example, when we have done Monte Carlo estimation, we have often used one million random numbers, which took R about 1 second – it's obviously completely infeasible to toss 32 million coins that quickly!

For a greater amount of physical randomness more quickly, one could look at times between the decay of radioactive particles, thermal noise in electrical circuits, the behaviour of photons in a laser beam, and so on. But these are quite expensive, and even these may not be quick enough for some applications.

The bad news about true physical randomness is that it is typically slow and expensive. But the good news is that we know it will definitely be perfectly random with no hidden patterns. (For Monte Carlo, we probably are happy simply to avoid any "obvious" patterns; but for uses in cryptography, for example, true perfect randomness ensuring no patterns that an enemy could exploit can be very important.)

Here's a cool video by the YouTuber Tom Scott about an internet company that uses a wall of lava lamps for their true physical randomness:

https://www.youtube.com/embed/1cUUfMeOijg

10.3 PRNGs

A **pseudorandom number generator** (**PRNG**) is a computer program that outputs a sequence of numbers that appear to be random. Of course, the numbers are not *actually* random – a computer program always performs exactly what you tell it to, exactly the same every time. But for a PRNG – or at least for a good PRNG – there should be no obvious patterns in the sequence of numbers produced, so they should act for all practical purposes *as if* they were random numbers. ("Pseudo" is a prefix that means something like "appears to be, even though it's not".) The best thing about PRNGs is because they are simple computer programs they run very very quickly and cheaply.

Many pseudorandom number generators work by applying a **recurrence**; that is, applying a function again and again. Suppose we want (pseudo)random integers between 0 and $m-1$. Then we have a **seed** x_1, which behaves as a starting point for the sequence, and a function f from $\{0, 1, \ldots, m-1\}$ to $\{0, 1, \ldots, m-1\}$. Then starting from x_1, we apply f to get the next

number in the sequence $x_2 = f(x_1)$. Then we apply f to *that*, to get the next point $x_3 = f(x_2)$. Then apply f to *that* to get the next number, and so on. So the sequence would be

$$x_1 \qquad x_2 = f(x_1) \qquad x_3 = f(x_2) = f(f(x_1)) \tag{10.4}$$

$$x_4 = f(x_3) = f(f(f(x_1))) \qquad \cdots \qquad x_{i+1} = f(x_i) = f(f(\cdots(f(x_1))\cdots)) \tag{10.5}$$

and so on.

Some functions f would be not produce a very random-looking sequence of numbers: think of a silly example like $f(x) = (x + 1) \bmod m$, so $x_{i+1} = (x_i + 1) \bmod m$ just increases by 1 each time, before "wrapping around" back to 0 when it gets to m. (The "mod m" means to wrap around when you get to m, or to take the remainder when you divide by m.) But mathematicians have come up with lots of examples of functions f which, for all possible practical purposes, seem to have outputs that appear just as random as an actual random sequence.

One example of such a function f would be $f(x) = (ax+c) \bmod m$, so $x_{i+1} = (ax_i+c) \bmod m$, which can be a very good PRNG for some values of a and c. In this case, the PRNG is known as a **linear congruential generator** (or **LCG**). We'll talk more about LCGs in the next lecture.

R's default method is of this form – well, it's *almost* a recurrence of the form $x_{i+1} = f(x_i)$. It actually uses a method called the Mersenne Twister, which uses a recurrence of the form $x_{i+1} = (g(x_i) + i) \bmod m$ for a complicated function g; here the "plus i", where i is the step number, means we have a slightly different update rule each timestep.

But how do we pick the seed – the starting point x_1? Normally, we use some true physical randomness to pick the seed. The benefit here is that just a small amount of true physical randomness can "start you off" by choosing the seed, and then the PRNG can produces huge amounts of pseudorandomness incredibly quickly. Indeed, that's what the wall of lava lamps in the video did – the lava lamps were just for producing lots of seeds for the PRNGs.

However, it is possible, and sometimes desirable, to set the seed "by hand". This is useful if you want to produce the same "random-looking" numbers as someone else (or yourself, previously). This is because if two people set the same seed x_1, then the numbers $x_2, x_3, ...$ produced after that will still appear to be random, but because the process is completely deterministic after the seed is chosen, both people will actually produce exactly the same sequence of numbers. This can be useful for checking accuracy of code, for example, or ensuring "reproducible analysis".

In R, by default, the seed is set based the current time on your computer, measured down to about 10 milliseconds. However, you can set the seed yourself, using `set.seed()`. For example, the following code sets the seed to be 123456, then generates 10 uniform pseudorandom numbers.

```
set.seed(123456)
runif(10)
```

```
 [1] 0.79778432 0.75356509 0.39125568 0.34155670 0.36129411 0.19834473
 [7] 0.53485796 0.09652624 0.98784694 0.16756948
```

Those numbers certainly appear to be random. But if you run those two lines of code on your computer, you should get exactly the same 10 numbers that I got. That's because you will also start with the same seed 123456, and then R will run the same pseudorandom – that is, completely deterministic – function to generate the next 10 numbers.

So PRNGs are quick and cheap. If they are designed well and started from a truly random seed, then we hope there will be know hidden patterns in the numbers, although it is difficult to be sure.

Next time: *We'll take a closer look at pseudorandom number generation using linear congruential generators.*

Summary:

- Random number generation has two problems: how to generate uniform random numbers between 0 and 1, and how to convert these to your desired distribution.
- Uniform random numbers can be generated from true physical randomness (which will definitely be totally random, but will be slow and expensive) or using a pseudorandom number generator on a computer (which is very fast, but needs to be designed carefully to ensure a random-looking output).
- LCGs are a type of pseudorandom number generator that are started from a "seed", which can be chosen using physical randomness or set by the user.

Read more: Voss, *An Introduction to Statistical Computing*, introduction to Chapter 1, introduction to Section 1.1, and Subsection 1.1.3.

11 LCGs

11.1 Definition and examples

Last lecture we introduced the idea of pseudorandom number generators (PRNGs), which are deterministic functions that produce a sequence of numbers that look for all practical purposes as if they are random.

We introduced the idea of a recurrence, where we have a function f and start with a seed x_1. We then produce a sequence through the recurrence $x_{i+1} = f(x_i)$. So $x_2 = f(x_1)$, $x_3 = f(x_2) = f(f(x_1))$, and so on.

We briefly mentioned a class of such PRNGs called **linear congruential generators**, or **LCGs**. An LCG generates integers between 0 and $m-1$ using a recurrence function of the form

$$f(x) = (ax + c) \bmod m,$$

so

$$x_{i+1} = (ax_i + c) \bmod m.$$

Here, "mod m" means "modulo m"; that is, we are using modular arithmetic, where when we get to $m-1$ we wrap back to 0 and start again. (Modular arithmetic is sometimes called "clock arithmetic", because hours of the day work modulo 12: for example, 3 hours after 11 o'clock is 2 o'clock, because $11 + 3 = 14 \equiv 2 \bmod m$.)

In the LCG $x_{i+1} = (ax_i + c) \bmod m$:

- m is called the **modulus**,
- a is called the **multiplier**,
- c is called the **increment**,
- x_1, the starting point, is called the **seed**.

Example 11.1. Let us look at two simple LCGs with modulus $m = 2^4 = 16$.

First, let $a = 5$ be the multiplier, $c = 3$ be the increment, and $x_1 = 1$ be the seed. Then we have

$$x_2 = (5x_1 + 3) \bmod 16 = (5 \times 1 + 3) \bmod 16 = 8 \bmod 16 = 8 \tag{11.1}$$
$$x_3 = (5x_2 + 3) \bmod 16 = (5 \times 8 + 3) \bmod 16 = 43 \bmod 16 = 11 \tag{11.2}$$
$$x_4 = (5x_3 + 3) \bmod 16 = (5 \times 1 + 3) \bmod 16 = 43 \bmod 58 = 10, \tag{11.3}$$

and so on. The sequence continues $(1, 8, 11, 10, 5, 12, 15, 14, 9, 0, \dots)$. This looks pretty much like a random sequence of numbers between 0 and 15 to me – I certainly don't see any obvious pattern.

Second, let $a = 3$ be the multiplier, $c = 6$ be the increment, and $x_1 = 1$ be the seed. Then we have

$$x_2 = (3x_1 + 2) \bmod 16 = (3 \times 1 + 6) \bmod 16 = 9 \bmod 16 = 9 \tag{11.4}$$
$$x_3 = (3x_2 + 4) \bmod 16 = (3 \times 9 + 6) \bmod 16 = 33 \bmod 16 = 1. \tag{11.5}$$

But now, using $x_3 = 1$ will give $x_4 = 9$ again. And $x_4 = 9$ will give $x_5 = 1$ again. So the sequence will be $(1, 9, 1, 9, 1, 9, 1, \dots)$, with just 1 and 9 repeating for ever. This definitely doesn't looks random!

The example illustrates that, while an LCG *can* provide a good sequence of pseudorandom numbers, we need to be careful with the parameters we choose.

Example 11.2. Of course, it doesn't make much sense to run LCGs by hand – the whole purpose of LCGs is that they can produce lots of (pseudo)random numbers very fast. So we should run them on computers.

The following R code sets up a function for sampling n numbers from an LCG.

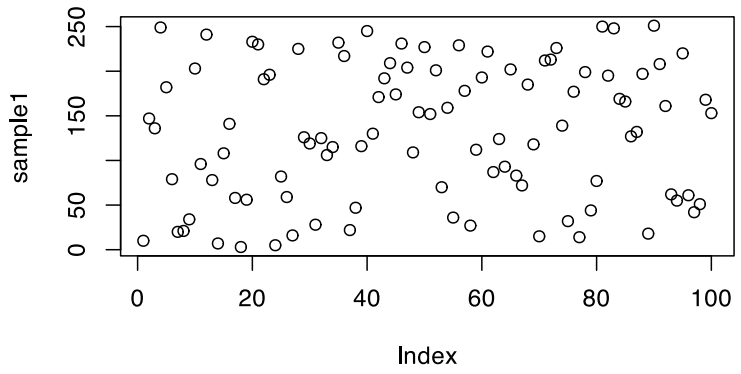
```
lcg <- function(n, modulus, mult, incr, seed) {
  samples <- rep(0, n)
  samples[1] <- seed
  for (i in 1:(n - 1)) {
    samples[i + 1] <- (mult * samples[i] + incr) %% modulus
  }
  return(samples)
}
```

In the fourth line, `%%` is R's "mod" operator.

Let's look at two examples with modulus $m = 2^8 = 256$.

First, let $a = 13$ be the multiplier, $c = 17$ be the increment, and $x_1 = 10$ be the seed.

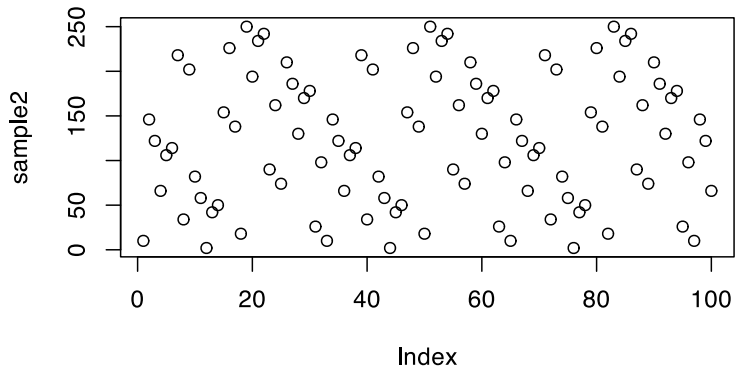
```
m <- 2^8
sample1 <- lcg(100, m, 13, 17, 10)
plot(sample1)
```



That looks like a pretty random collection of numbers to me.

Second, we stick with $a = 13$ be the multiplier and $x_1 = 10$ as the seed, be decrease the increment by 1 to $c = 16$.

```
sample2 <- lcg(100, m, 13, 16, 10)
plot(sample2)
```



This doesn't seem as good. We can see there's some pattern where there are parallel downward sloping lines. And also there seems to be some sort of pattern *within* these downward sloping lines, sometimes with quite regularly-spaced points on those lines. And looking more closely, we can see that actually the pattern of numbers repeats exactly every 32 steps

```
which(sample2 == 10)
```

```
[1]  1 33 65 97
```

so we only ever see 32 of the possible 256 values. This doesn't seem to look like a sequence of independent uniformly random points.

So again, it seems like LCG *can* provide a good sequence of pseudorandom numbers, but it seems quite sensitive to a good choice of the parameters.

Is these examples, we've usually taken m to be a power of 2. There are a few reasons for this:

- We will want to divide each term in the sequence x_i by m to get a number between $[0, 1]$. If our number will be stored as a b-bit number, then it is makes sense to have created b bits (for an integer between 0 and $m = 2^b - 1$) in the first place. This means every integer in $\{0, 1, \dots, m-1\}$ corresponds to exactly one b-bit number. Further this makes the division by $m = 2^b$ extremely simple: you simple add "0." (zero point…) at the front of the number!
- Modular arithmetic modulo a power of 2 is very simple for a computer. For a number in binary, the value modulo 2^b is simply the last b bits of the number. So `10111001` modulo 2^4 is simply `1001`, the last four bits.
- Having $m = 2^b$ (or, more generally, having m being the product of lots of small prime factors) makes it easier to choose parameters such that the LCG is a good pseudorandom number generator … as we shall see in the next section.

11.2 Periods of LCGs

In an LCG, each number in the sequence depends only on the one before, since $x_{i+1} = (ax_i + c) \bmod m$. This means if we ever get a single "repeat" in our sequence – that is, if we see a number we have seen at some point before – then the whole sequence from that point on will copy what came before.

For example, in the second part of Example 11.1, the sequence started $(1, 9, 1, \dots)$. As soon as we hit that repeat 1, we know we're going to see that pattern $1, 9$ repeated forever. We say

that this LCG has "period" 2. More generally, the **period** of an LCG is the smallest $k \geq 1$ such that $x_{i+k} = x_i$ for some i.

We would like our LCGs to have a big period. If an LCG has a small period, it will not look random, as we will just repeat the same small number of values over and over again.

The smallest possible period is 1. That would be an extraordinarily bad LCG, as it would just spit out the same number forever. The biggest possible period is m. That is because there are only m possible values in $\{0, 1, \ldots, m-1\}$; so after $m+1$ steps we must have had a repeat, by the pigeonhole principle. Having the maximum period m is also called having "full period".

Generally, the only way to find the period of an LCG is to run it for a long time, and see how long it takes to start repeating. But, conveniently, there *is* a very easy way to tell if an LCG has the maximum possible period m, thanks to a result of TE Hull and AR Dobell.

Theorem 11.1 (Hull–Dobell theorem)**.** *Consider the linear congruential generator* $x_{i+1} = (ax_i + c) \bmod m$*. This LCG has period* m *if and only if the following three conditions hold:*

1. m *and* c *are coprime;*
2. $a - 1$ *is divisible by all prime factors of* m*;*
3. *if* m *is divisible by 4, then* $a - 1$ *is divisible by 4.*

If $m = 2^b$ is a power of 2 (with $b \geq 2$), then the three conditions simplify to a particularly pleasant form:

1. c is odd;
2. $a - 1$ is even;
3. $a - 1$ is divisible by 4.

Of course, the third point on the list implies the second. So we only actually need to check *two* things:

1. c is odd
2. a is 1 mod 4 (that is, a is one more than a multiple of 4).

(We won't prove the Hull–Dobell theorem here – it's some pretty tricky number theory. But see Knuth, *The Art of Computer Programming*, *Volume 2: Seminumerical algorithms*, Subsubsection 3.2.1.2 if you really want a proof and have sufficient number theory background.)

Example 11.3. Let's go back to the earlier examples, and see if they have full periods or not.

In the first LCG of Example 11.1, we had $m = 2^4$, $a = 5$ and $c = 3$. Here, c is odd, and $a = 4+1$ is 1 mod 4. This LCG fulfils both conditions, so it has the maximum possible period of 16.

In the second LCG of Example 11.1, we had $m = 2^4$, $a = 3$ and $c = 6$. Here, c is even, and a is 2 mod 4. This LCG does not fulfil both the conditions – in fact, it fails them both – so it does not have the maximum possible period of 16. (We already saw that it in fact has period 2.)

In the first LCG of Example 11.2, we had $m = 2^8$, $a = 13$ and $c = 17$. Here, c is odd, and $a = 12+1$ is 1 mod 4. This LCG fulfils both conditions, so it has the maximum possible period of 256.

In the second LCG of Example 11.2, we had $m = 2^8$, $a = 13$ and $c = 16$. Here, c is even, and $a = 12+1$ is 1 mod 4. So although this LCG does fulfil the second condition, it does not fulfil the first, so it does not have the maximum possible period of 256. (We already saw that it in fact has period 32.)

It normally a good idea to make sure your LCG has full period – if it's so easy to ensure, then why not? (That said, a very large but not-quite-maximum period may be good enough. For example, if an LCG with modulus 2^{64} has a period of "only" 2^{60}, that might well be enough: one million samples a second for one thousand years is only 2^{55} samples, so you'd never actually *see* a repeat.)

But merely having full period (and a large modulus) isn't enough by itself to guarantee an LCG will make a good pseudorandom number generator. After all, the silly LCG $x_{i+1} = x_i + 1$ has full period, but the sequence

$$(0, 1, 2, 3, \dots, m-2, m-1, 0, 1, 2, \dots)$$

will not look random.

11.3 Statistical testing

Before being used properly, any pseudorandom number generator is subjected to a barrage of statistical tests, to check if its output seems to "look random". Alongside checking it has a very large period, the tester will want to check other statistical properties of randomness. Even the best PRNGs might not pass every single such test. See Voss, *An Introduction to Statistical Computing*, Subsection 1.1.2 for more (non-examinable) material on statistical tests for randomness.

LCGs were considered state-of-the-art until the late-90s or so. However, it was discovered that m needs be very big and the number of samples used fairly small in order to pass some of the

more stringent statistical tests of randomness. For example, it's suggested that $n = 1\,000\,000$ (one million) samples from a full-period LCG with modulus $m = 2^{64}$ might be the limit before it starts "not looking random enough". In particular, an LCG with a large period may not actually see *enough* repeats – after all, random numbers will have the occasional one-off repeat, just by chance. Compared to other more modern methods (like R's default, the Mersenne Twister, mentioned in the last lecture, discovered in 1997), an LCG requires quite a lot of computation for only a modest number of samples. So while LCGs are still admired for their simplicity and elegance, they have fallen out of favour for cutting-edge computational work.

Next time: *We'll use our pseudorandom uniform* $[0, 1]$ *random numbers to make random numbers with other discrete or uniform distributions.*

Summary:

- Linear congruential generators are pseudorandom number generators based on the recurrence $x_{n+1} = (ax_n + c) \bmod m$.
- Any LCG will eventually repeat with periodic behaviour.
- Suppose m is a power of 2. Then an LCG has full period m if and only if c is odd and $a = 1 \bmod 4$.

You should now be able to answer all questions on **Problem Sheet 2**. Your answers will be discussed in the problems class on **Thursday 31 October**.

Read more: Voss, *An Introduction to Statistical Computing*, Subsections 1.1.1 and 1.1.2.

Problem Sheet 2

Full solutions are now available.

This is Problem Sheet 2, which covers material from Lectures 7 to 11. You should work through all the questions on this problem sheet in advance of the problems class, which takes place in the lecture of **Thursday 30 October**.

This problem sheet is to help you practice material from the module and to help you check your learning. It is *not* for formal assessment and does not count towards your module mark.

If you want some brief informal feedback on **Question 1** (marked), you should submit your work electronically through Gradescope via the module's Minerva page by **1400 on Tuesday 28 October**. (If you hand-write solutions on paper, the easiest way to scan-and-submit that work is using the Gradescope app on your phone.) I will return some brief comments on your those two questions by the problems class on Thursday 30 October. Because this informal feedback, and not part of the official assessment, I cannot accept late work for any reason – but I am always happy to discuss any of your work on any question in my office hours.

Full solutions will be released on Friday 31 October.

12 Uniform and discrete

We've seen how to generate uniform random variates $U \sim \mathrm{U}[0,1]$, either using true physical randomness or the output of a pseudorandom number generator. But in statistics – whether performing Monte Carlo estimation or anything else – we typically want to sample from some other distribution X.

In the next five lectures, we will look at ways we can transform the uniform random variable U to take on different distributions instead. We start today by looking at some important special cases.

12.1 Uniform random variables

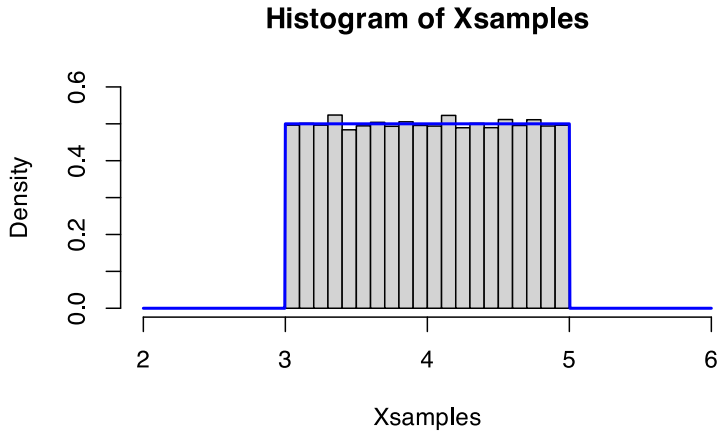
We know how to generate $U \sim \mathrm{U}[0,1]$. But suppose we want to generate $X \sim \mathrm{U}[a,b]$ instead, for some $a < b$; how can we do that.

Well, the original U has "width" 1, and we want X to have width $b-a$, so the first thing we should do is multiply by $(b-a)$. This gives us $(b-a)U$, which we expect should be uniform on $[0, b-a]$. Then we need to shift it, so it starts not a 0 but at a; we can do this by adding a. This gives us $(b-a)U + a$, which should be uniform on $[0+a, (b-a)+a] = [a,b]$. So $X = (b-a)U + a$ would seem to give us the desired $X \sim \mathrm{U}[a,b]$ random variable.

We can check this seems to have worked by using a histogram, for example.

Example 12.1. Let $U \sim \mathrm{U}[0,1]$. We can generate $X \sim \mathrm{U}[3,5]$ by $X = 2U + 3$.

```
n <- 1e5
Usamples <- runif(n)
Xsamples <- 2 * Usamples + 3
hist(Xsamples, probability = TRUE, xlim = c(2, 6), ylim = c(0, 0.6))
curve(dunif(x, 3, 5), add = TRUE, n = 1001, lwd = 2, col = "blue")
```



Here, we used the `probability = TRUE` argument to the histogram function `hist()` to plot the density on the y-axis, rather than the raw number of samples. The density should match the probability density function of the random variable X. We drew the PDF of X over the histogram in blue, and can see it is a superb match.

But what if we very formally wanted to *prove* that $X = (b-a)U + a$ definitely has the distribution $X \sim \mathrm{U}[a, b]$; how could we do that?

The best way to give a formal proof of something like this is to use the **cumulative distribution function** (CDF). Recall that the CDF F_Y of a distribution Y is the function $F_Y(y) = \mathbb{P}(Y \leq y)$. One benefit of the CDF is it works equally well for both discrete and continuous random variables, so we don't need to give separate arguments for discrete and continuous cases.

The CDF of the standard uniform distribution $U \sim \mathrm{U}[0, 1]$ is

$$F_U(u) = \begin{cases} 0 & u < 0 \\ u & 0 \leq u \leq 1 \\ 1 & u > 1, \end{cases} \tag{12.1}$$

and the CDF of any uniform distribution $X \sim \mathrm{U}[a, b]$ is

$$F_X(x) = \begin{cases} 0 & x < a \\ \dfrac{x-a}{b-a} & a \leq x \leq b \\ 1 & x > b. \end{cases} \tag{12.2}$$

So to show that $X = (b-a)U + a \sim \mathrm{U}[a, b]$, we take the fact that U has the CDF in Equation 12.1 and try to use it to show that X has the CDF in Equation 12.2.

Indeed, we have

$$F_X(x) = \mathbb{P}(X \leq x) = \mathbb{P}((b-a)U + a \leq x) = \mathbb{P}\left(U \leq \frac{x-a}{b-a}\right).$$

But putting $u = (x-a)/(b-a)$ in Equation 12.1, does indeed give the CDF in Equation 12.2. The lower boundary $u < 0$ becomes $x < 0 \times (b-a) + a = a$; the upper boundary $u > b$ becomes $x > 1 \times (b-a) + a = b$; and, in between, the CDF u becomes $(x-a)/(b-a)$. Thus we have proven that X has the CDF of the U$[a, b]$ distribution, as required.

12.2 Discrete random variables

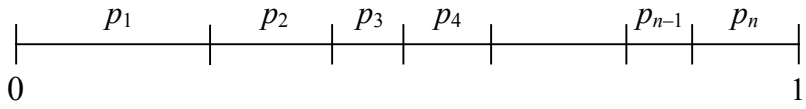
Suppose we want to simulate a Bernoulli trial; that is, a random variable X that is 1 with probability p and 0 with probability $1-p$, for some p with $0 < p < 1$. As ever, we only have a standard uniform $U \sim$ U$[0, 1]$ to work with. How can we form our Bernoulli trial?

Here are two possible ways:

- If $U < p$, then take $X = 1$; while if $U \geq p$, take $X = 0$. Note that $\mathbb{P}(X = 1) = \mathbb{P}(U < p) = p$ and $\mathbb{P}(X = 0) = \mathbb{P}(U \geq p) = 1 - \mathbb{P}(U < p) = 1 - p$, as required.
- Alternatively: if $U \leq 1 - p$, take $X' = 0$; while if $U > p$, take $X' = 1$.

The first method is just the second method with U replaced by $1-U$. Interestingly, that means we can generate two different Bernoulli trials X and X' from the same U, which will therefore be negatively correlated. Although there's unlikely to be any situation where a Bernoulli trial would be used in Monte Carlo estimation, in theory, the two versions (X, X') generated from the same U could potentially be used as antithetic variables.

What about more general discrete random variables? Suppose a random variable takes values $x_1, x_2, ...$ with probabilities $p_1, p_2, ...$ We can think of these probabilities as splitting up the interval $[0, 1]$ into subintervals of lengths $p_1, p_2, ...$, since these probabilities add up to 1.



So the first interval is $I_1 = (0, p_1]$; the second interval is $(p_1, p_1 + p_2]$; the third intervals is $(p_1 + p_2, p_1 + p_2 + p_3]$, and so on. We then pick a point U from $[0, 1]$ uniformly at random, and whichever interval I_i it is in, take the corresponding value x_i.

In terms of the CDF, we have $F_X(x_i) = p_1 + p_2 + \cdots + p_i$, so the intervals are of the form $(F_X(x_{i-1}), F_X(x_i)]$. So we can think of this as rounding $F_X(U)$ up to the next $F_X(x_i)$, then taking that value x_i.

Example 12.2. Consider generating a binomial distribution $X \sim \text{Bin}(5, \frac{1}{2})$. The PMF and CDF are as shown below

value x	0	1	2	3	4
PMF $p(x)$	$\frac{1}{32}$	$\frac{5}{32}$	$\frac{10}{32}$	$\frac{10}{32}$	$\frac{5}{32}$
CDF $F_X(x)$ (fraction)	$\frac{1}{32}$	$\frac{6}{32}$	$\frac{16}{32}$	$\frac{26}{32}$	$\frac{31}{32}$
CDF $F_X(x)$ (decimal)	0.03125	0.1875	0.5	0.8125	0.96875

Suppose I want a sample from this, based on the uniform variate $u_1 = 0.7980$. We see that u_1 is bigger that $F_X(2) = 0.5$ (the upper end of the "2" interval) but less than $F_X(3) = 0.8125$ (the upper end of the "3" interval), so falls into the "3" interval. So our first binomial variate is $x_1 = 3$.

If we then got the next uniform variate $u_2 = 0.4353$, we would see that u_2 is between $F_X(1) = 0.1875$ and $F_X(2) = 0.5$, so would take $x_2 = 2$ for our next binomial variate.

Next time: *Sampling from continuous distribution using the CDF.*

Summary:

- If $U \sim \text{U}[0, 1]$, then $X = (b - a)U + a \sim \text{U}[a, b]$.
- A discrete random variable can be generated by splitting $[0, 1]$ into subintervals with lengths according to the probabilities, then picking a point from the interval at random.

Remember that your answers to **Problem Sheet 2** will be discussed in the problems class on **Thursday 31 October**.

Read more: Voss, *An Introduction to Statistical Computing*, Sections 1.2 and 1.3.

13 Inverse transform method

13.1 Inverse CDF

We have started looking at how to transform a standard uniform random variable $U \sim \mathrm{U}[0,1]$ into any other distribution X.

Last lecture, we saw how to do this for other uniform distributions and for discrete random variables. Booth involved the cumulative distribution function $F(x) = \mathbb{P}(X \leq x)$. For generating a wider class of random variables (including continuous random variables) the CDF will continue to be important. In fact, it is the *inverse* of the CDF that will play a crucial role.

It seems natural to define the inverse F^{-1} of the CDF in just the same way we would define the inverse of any other function: that $F^{-1}(u)$ is the unique value x such that $F(x) = u$. This is illustrated in the figure below.

[picture]

This definition works fine for a purely continuous distribution that doesn't have any "gaps" in the set of values it can take. But for a general random variable, there are two problems with this definition.

1. If X has any point masses – points x with strictly positive probability $\mathbb{P}(X = x) > 0$ of hitting that point exactly – then F may "jump past" the value u, so there is *no* value x such that $F(u)$. (See the blue line on the graph below.)
2. If there is an interval on the line where X has probability zero, then all the xs in that interval have the same value of $F(x) = u$, so there is no *unique* value x such that $F(x) = u$. (See the red line on the graph below.)

[picture]

It turns out that the best way to solve this is the following. For the first obstacle, if there are many points x with $F(x) = u$, we take the first one – that is, the smallest x with $F(x) = u$. For the second obstacle, we take the value of the smallest x where $F(x)$ is above u, which comes right at the jump. These two cases are illustrated below.

[picture]

These two awkward cases can be encapsulated in the following definition.

Definition 13.1. Let X be a random variable with cumulative distribution function $F_X(x) = \mathbb{P}(X \leq x)$. Then the **inverse cumulative distribution function** (inverse CDF) F_X^{-1} is defined for $u \in (0,1)$ by

$$F_X^{-1}(u) = \min\{x : F_X(x) \geq u\}.$$

You should check that you agree this definition matches the discussion above.

13.2 Inverse transform

The inverse transform method works like this: To generate X, simply apply the inverse CDF F^{-1} to a standard uniform random variable U.

Theorem 13.1. *Let F be a cumulative distribution function, and let F^{-1} be its inverse. Let $U \sim \mathrm{U}[0,1]$. Then $X = F^{-1}(U)$ has cumulative distribution function F.*

Proof. We need to show that $\mathbb{P}(X \leq x) = F(x)$ when this is in $(0,1)$. The proof is very easy if F has no "gaps" or "jumps" as described above. Then, we simply have

$$\mathbb{P}(X \leq x) = \mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x),$$

where we have simply "undone" the function F^{-1} and used that $\mathbb{P}(U \leq u) = u$ for $u \in (0,1)$.

For the general case, we need to be just a little bit more careful. We have

$$\mathbb{P}(X \leq x) = \mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(\min\{y : F(y) \geq U\} \leq x).$$

If x is bigger than the minimum y with $F(y) \geq u$, then certainly $F(x) \geq u$ as well; while if $F(x) \geq u$, then x must be at least as big as the minimum y for which $F(y) \geq u$. Hence $\min\{y : F(y) \geq U\} \leq x$ if and only if $F(x) \geq U$. So

$$\mathbb{P}(X \leq x) = \mathbb{P}(\min\{y : F(y) \geq U\} \leq x) = \mathbb{P}(F(x) \geq U) = \mathbb{P}(U \leq F(x)) = F(x),$$

as before. □

The method – known as the **inverse transform method** gives a simple way to generate any random variable X for which the inverse CDF F_X^{-1} can be computed easily.

This is not all random variables, however. In particular, the inverse CDF of the normal distribution does not have a closed form, so this does not give a way of sampling normal distributions. Later we'll see other methods that allow us to sample from normal random variables.

13.3 Examples

Let's see some examples. The idea for all these problems is "Write $U = F(X)$, then invert, to get $X = F^{-1}(U)$."

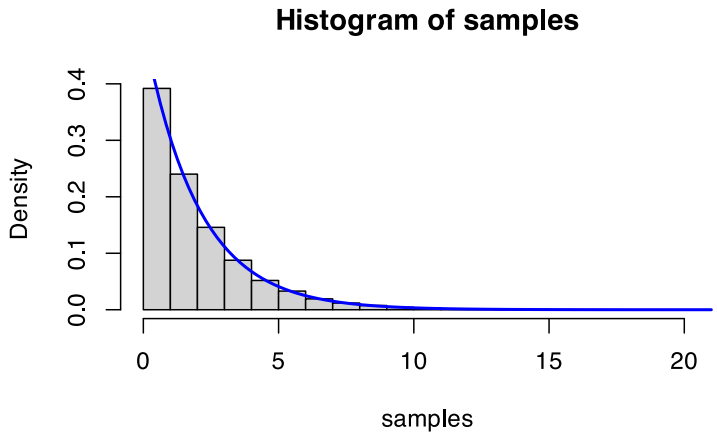
Example 13.1. Let $X \sim \text{Exp}(\lambda)$ be an exponential distribution with rate λ. This has PDF $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$ and CDF $F(x) = 1 - e^{-\lambda x}$.

We write $U = F(X)$ and invert it to make X the subject. So $U = 1 - e^{-\lambda X}$, and therefore

$$X = -\frac{1}{\lambda} \log(1 - U).$$

We should check that this really does have an exponential distribution.

```
rate <- 0.5
n <- 1e5
unif <- runif(n)
samples <- -(1 / rate) * log(1 - unif)

hist(samples, probability = TRUE)
curve(dexp(x, rate), add = TRUE, col = "blue", lwd = 2)
```



Looks like an accurate sample!

Since $1 - U$ has the same distribution as U, it can be more convenient to simply write

$$X' = -\frac{1}{\lambda} \log U$$

instead.

Alternatively, since X and X' are not independent and both have the same exponential distribution, they are a candidate to use as an antithetic pair in Monte Carlo estimation.

Example 13.2. Consider X with PDF

$$f(x) = \frac{x}{\gamma} \exp\left(-\frac{x^2}{2\gamma}\right)$$

for $x \geq 0$, and CDF

$$F(x) = 1 - \exp\left(-\frac{x^2}{2\gamma}\right)$$

This is known as the **Rayleigh distribution** with scale parameter γ.

Again, we write $U = F(X)$ and invert. so

$$U = 1 - \exp\left(-\frac{X^2}{2\gamma}\right)$$

so

$$X = \sqrt{-2\gamma \log(1-U)}.$$

Again, $X' = \sqrt{-2\gamma \log U}$ is a slightly simpler expression, or could be used in an antithetic variables Monte Carlo approach.

Example 13.3. Suppose $X \sim \mathrm{U}[a, b]$, then

$$F(x) = \frac{x-a}{b-a}$$

(except for when $F(x) = 0$ or 1).

Write $U = F(X)$ and invert. We get $X = (b-a)U + a$. This is precisely the method for generating general uniform distributions that we saw in the last lecture.

Example 13.4. Let X be discrete on the values $x_1, x_2, ...$ with probabilities $p_1, p_2, ...$ The CDF is

$$F(x) = \begin{cases} 0 & x < x_1 \\ p_1 & x_1 \leq x < x_2 \\ p_1 + p_2 & x_2 \leq x < x_3 \\ p_1 + p_2 + p_3 & x_3 \leq x < x_4 \\ \cdots & \cdots. \end{cases}$$

Remembering the rule for "jumps" in the CDF, we see that the inverse CDF is

$$F^{-1}(u) = \begin{cases} x_1 & u < p_1 \\ x_2 & p_1 \le x < p_1 + p_2 \\ x_3 & p_1 + p_2 \le x < p_1 + p_2 + p_3 \\ \cdots & \cdots . \end{cases}$$

Taking $X = F^{-1}(U)$ gives the same method for generating discrete random variables as we discussed in the last lecture.

Example 13.5. *Consider a distribution with PDF*

$$f(x) = \begin{cases} x^2 & 0 \le x \le 1 \\ \frac{2}{3} & 1 < x \le 2 \\ 0 & \text{otherwise.} \end{cases}$$

Show how to sample X using a standard uniform random variable U.

This is a standard sort of question. First we have to find the CDF, then we have to invert it.

We find the CDF from the PDF by integrating. For $0 \le x \le 1$, we have

$$F(x) = \int_0^x f(y)\,\mathrm{d}y = \int_0^x y^2\,\mathrm{d}y = \tfrac{1}{3}x^3.$$

Then for $1 < x \le 2$, we have

$$F(x) = F(1) + \int_1^x f(y)\,\mathrm{d}y = \tfrac{1}{3} + \int_1^x \tfrac{2}{3}\,\mathrm{d}y = \tfrac{1}{3} + \tfrac{2}{3}x - \tfrac{2}{3} = \tfrac{2}{3}x - \tfrac{1}{3}.$$

For $0 \le U <\le F(1) = \frac{1}{3}$, we have $U = \frac{1}{3}X^3$, so $X = \sqrt[3]{3U}$. For $F(1) = \frac{1}{3} < U \le 1$, we have $U = \frac{2}{3}X - \frac{1}{3}$, so $X = \frac{3}{2}U + \frac{1}{2}$. So the inverse transform is

$$X = \begin{cases} \sqrt[3]{3U} & U \le \frac{1}{3} \\ \frac{3}{2}U + \frac{1}{2} & U > \frac{1}{3}. \end{cases}$$

13.4 Box–Muller transform

[This was actually lectured at the beginning of Lecture 14, but belongs in this section.]

While the inverse transform method is very powerful, we have mentioned that it doesn't work directly for the normal distribution, which does not have an exact closed form for the inverse

CDF. (Although R does actually use the inverse transform method in its `rnorm()` function by using an *approximation* to the inverse CDF.)

Instead, the **Box–Muller transform** (discovered by the statistician GEP Box and the computer scientist Marvin E Muller in 1958) is a clever way to easily transform a standard uniform into a normal distribution.

Actually, that's not quite true. Rather than transforming one standard uniform U into one normal distribution X, it instead transforms *two* independent standard uniforms U, V into *two* normal distributions X, Y. (There is some profound mathematical sense in which the two-dimensional bivariate normal is somehow a more "deeply" important mathematical object than the one-dimensional univariate normal we are used to, but we don't have time to get into that here.)

First, let's note it suffices to produce a standard normal $X \sim \mathrm{N}(0,1)$. Any other normal distribution $W \sim \mathrm{N}(\mu, \sigma^2)$ can then be formed as $W = \sigma X + \mu$.

The idea of the Box–Muller transform is based on converting the two standard normal distributions (X, Y) from cartesian coordinates into polar coordinates (R, Θ). (Those of you who have know how to calculate the Gaussian integral will have seen this idea before.)

[picture]

Theorem 13.2. *Let $X, Y \sim \mathrm{N}(0,1)$ be independent standard normal distributions. Write*

$$R = \sqrt{X^2 + Y^2} \qquad \Theta = \tan \frac{Y}{X}$$

for the radius and the angle of (X, Y) in polar coordinates. Then the radius R has a Rayleigh distribution with scalar parameter $\gamma = 1$, the angle Θ has a uniform distribution on $[0, 2\pi]$, and R and Θ are independent.

Proof. The joint PDF of (X, Y) is

$$\begin{aligned} f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y &= f_X(x)\, f_Y(y)\,\mathrm{d}x\,\mathrm{d}y && (13.1)\\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\tfrac{1}{2}x^2\right) \frac{1}{\sqrt{2\pi}} \exp\left(-\tfrac{1}{2}y^2\right) \mathrm{d}x\,\mathrm{d}y && (13.2)\\ &= \frac{1}{2\pi} \exp\left(-\tfrac{1}{2}(x^2+y^2)\right) \mathrm{d}x\,\mathrm{d}y. && (13.3) \end{aligned}$$

{#eq-box-1} The joint PDF of (R, Θ) is

$$\begin{aligned} f_{R,\Theta}(r,\theta)\,\mathrm{d}r\,\mathrm{d}\theta &= f_R(r)\, f_\Theta(\theta)\,\mathrm{d}r\,\mathrm{d}\theta && (13.4)\\ &= r \exp\left(-\tfrac{1}{2}r^2\right) \frac{1}{2\pi}\,\mathrm{d}r\,\mathrm{d}\theta, && (13.5) \end{aligned}$$

{#eq-box-2}

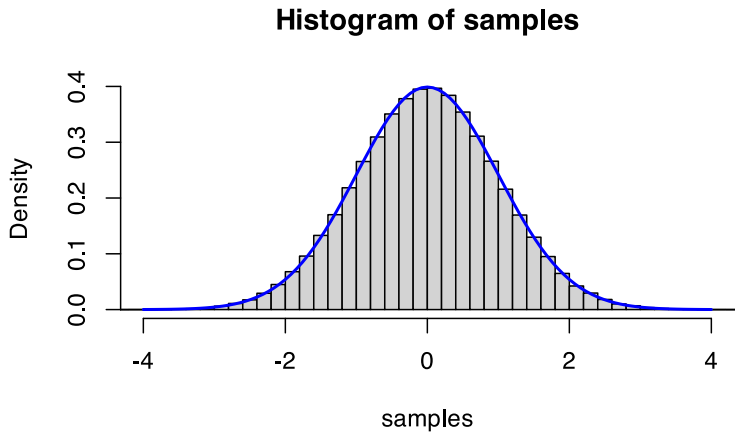
But in **?@eq-box-1**, we can substitute $r = x^2 + y^2$ and $\mathrm{d}x\,\mathrm{d}y = r\,\mathrm{d}r\,\mathrm{d}\theta$, and get **?@eq-box-2**. □

So we've reduced the problem of sample two normal distributions to sampling a Rayleigh (with scale parameter 1) and a uniform (on $[0, 2\pi]$). But we know how to do this. We saw in Example 13.2 that we can take $R = \sqrt{-2\log U}$ (or $\sqrt{-2\log(1-U)}$), and we saw in the last lecture that we can take $\Theta = 2\pi V$. We can then transform back into cartesian coordinates with

$$X = R\cos\Theta = \sqrt{-2\log U}\cos(2\pi V) \tag{13.6}$$

$$Y = R\sin\Theta = \sqrt{-2\log U}\sin(2\pi V). \tag{13.7}$$

```
n <- 1e5
unif1 <- runif(n)
unif2 <- runif(n)
rad <- sqrt(-2 * log(unif1))
ang <- 2 * pi * unif2
samples <- c(rad * cos(ang), rad * sin(ang))

hist(samples, probability = TRUE, breaks = 50, xlim = c(-4, 4))
curve(dnorm(x), add = TRUE, col = "blue", lwd = 2)
```



This confirms that we get an excellent match to the normal distribution.

Next time: *Sampling using rejection.*

Summary:

- The inverse F^{-1} of a CDF F is defined by $F_X^{-1}(u) = \min\{x : F_X(x) \geq u\}$.

- The inverse transform method converts $U \sim \mathrm{U}[0,1]$ to a random variable with CDF by setting $X = F^{-1}(U)$. That is: Set $U = F(X)$, and rearrange to make X the subject.
- The Box–Muller transform is a way to generate two independent standard normal distributions. Set R to Rayleigh with scale parameter 1, set $\Theta \sim \mathrm{U}[0, 2\pi]$, then take $X = R\cos\Theta$ and $Y = R\sin\Theta$.

Read more: Voss, *An Introduction to Statistical Computing*, Section 1.3.

14 Rejection sampling

14.1 Rejection

In the last two lectures, we have taken standard uniform $U \sim \mathrm{U}[0,1]$ random variables, and have applied a function to them to transform into some other distribution X. One U gets turned into one X. (Or, for the Box–Muller transform, two Us become two Xs.)

But so far, we have taken each sample we are given. But another way to get a different distribution is to throw out samples we don't like and wait until we get a sample we do like. This is called **rejection sampling**.

Suppose we want not a $\mathrm{U}[0,1]$ random variable but instead a $\mathrm{U}[0,\frac{1}{2}]$ random variable. One way we've already seen to do this is by the inverse transform method: simply multiply U by $\frac{1}{2}$. But we could also do this by rejection. We start with a proposed sample $U \sim \mathrm{U}[0,1]$. If $U \leq \frac{1}{2}$, we "accept" the sample, and keep it. But if $U > \frac{1}{2}$, we "reject" the samples – we throw it away and ask for a new one. We keep proposing samples until we accept one that's less than $\frac{1}{2}$. It should be easy to convince yourself that we get a $\mathrm{U}[0,\frac{1}{2}]$ random variable this way. (But we'll prove it later, if not.)

The advantage of rejection sampling is that it can help us get samples from some distributions that we couldn't access with the inverse transform method. The disadvantage is that it can be costly or slow, because we may have to reject lots of samples before finding enough that we can accept. The more often we reject samples, the slower the procedure will be.

Rejection sampling is particularly useful for sampling from a conditional distribution, such as the conditional distribution of Y given that $Y \in A$: we simply accept a sample y if $y \in A$ and reject it if not.

Example 14.1. Let $Y \sim \mathrm{N}(0,1)$. Suppose we wish to use Monte Carlo estimation to estimate $\mathbb{E}(Y \mid Y \geq 1)$.

To do this, we will need samples from the conditional distribution $Y \mid Y \geq 1$. So we accept proposed standard normal samples that are at least 1, and reject proposed samples that are less than 1.

There are two ways we could run this in practice. First, we could decide to take n proposal samples from Y and just see how many get accepted.

```
n_prop <- 1e6
props   <- rnorm(n_prop)
accepts <- props[props >= 1]
length(accepts)
```

```
[1] 158933
```

```
MCest1 <- mean(accepts)
MCest1
```

```
[1] 1.526205
```

We end up accepting around 160,000 samples out of the 1,000,000 proposals we had to start with.

Second, we could keep proposing as many samples as needed until we reach some desired number of acceptances.

```
n_acc <- 1e5

samples <- rep(0, n_acc)
count <- 0
for (i in 1:n_acc) {
  newsample <- 0
  while (newsample < 1) {
    newsample <- rnorm(1)
    count <- count + 1
  }
  samples[i] <- newsample
}
count
```

```
[1] 632946
```

```
MCest2 <- mean(samples)
MCest2
```

```
[1] 1.526
```

This required taking about 630,000 proposals to get 100,000 acceptances.

Here we used a "`while`" loop to keep taking samples *until* we go one that was not less than 1. The lines involving `count` were just so I could see how many proposals ended up being needed – these aren't an integral part of the code.

14.2 Acceptance probability

So far, we have looked at always accepting or always rejecting a proposed sample, depending on its value. But we could "perhaps" accept some proposals too. Suppose we are already sampling from some distribution Y (perhaps generated via the inverse transform method, for example). If we see the proposed sample $Y = x$, we could accept it with some **acceptance probability** $\alpha(x) \in [0, 1]$. We can control the accepted samples more delicately by adjusting this acceptance function α to values that aren't just 0 or 1.

What is the distribution of an accepted sample X?

Well, using Bayes' theorem, we have in the discrete case

$$\mathbb{P}(X = x) = \mathbb{P}(Y = x \mid \text{accept}) = \frac{\mathbb{P}(Y = x)\,\mathbb{P}(\text{accept} \mid Y = x)}{\mathbb{P}(\text{accept})} = \frac{1}{Z}\alpha(x)\,\mathbb{P}(Y = x).$$

where $Z = \mathbb{P}(\text{accept})$ is the normalising constant. In the continuous case, with g the PDF of the original Y and f the PDF of the accepted X, we have

$$f(x) = g(x \mid \text{accept}) = \frac{g(x)\,\mathbb{P}(\text{accept} \mid X = x)}{\mathbb{P}(\text{accept})} = \frac{1}{Z}\,\alpha(x)\,g(x),$$

where $Z = \mathbb{P}(\text{accept})$ again.

Example 14.2. Suppose we wish to sample from the distribution

$$f(x) \propto \exp\left(-\tfrac{1}{2}x^2\right)(\sin^2 x). \tag{14.1}$$

How can we do this?

Well, we can note that the PDF of the standard normal is

$$g(x) = \frac{1}{2\pi}\,\exp\left(-\tfrac{1}{2}x^2\right) \propto \exp\left(-\tfrac{1}{2}x^2\right)$$

and that

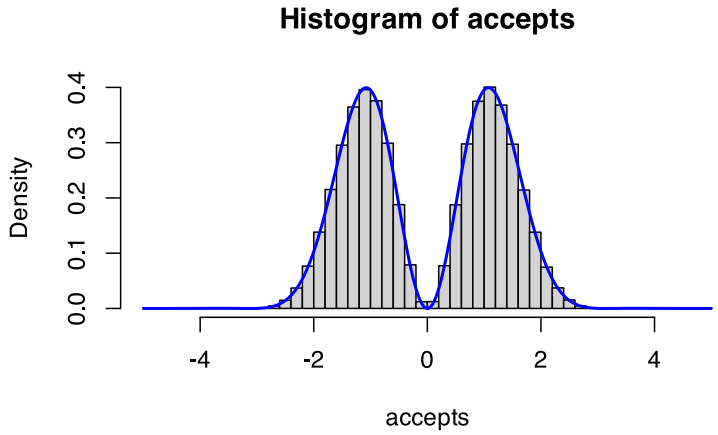
$$0 \le \sin^2 x \le 1.$$

(Here, $\sin^2 x$ means $(\sin x)^2$, by the way.) This means that, if we take proposals $Y \sim \mathrm{N}(0, 1)$, and then accept an proposed sample with probability $\alpha(x) = \sin^2 x$, that will give us the distribution Equation 14.1.

```
n_prop <- 1e6
props <- rnorm(n_prop)
accepts <- props[runif(n_prop) <= sin(props)^2]
length(accepts)
```

```
[1] 432649
```

```
hist(accepts, probability = TRUE, breaks = 50)
curve(
  0.92 * exp(-x^2 / 2) * sin(x)^2, add = TRUE, n = 1001,
  lwd = 2, col = "blue"
)
```



By rejecting lots of proposals with values near 0, we turned the unimodal ("one hump")proposal distribution $Y \sim \mathrm{N}(0, 1)$ into this interesting bimodal ("two hump") distribution.

Let's explain line 3 more carefully. We want to accept a proposal x with probability $\sin^2 x$. We saw in Lecture 12 that we can simulate a Bernoulli(p) distribution by taking the value 1 is $U \leq p$ and taking 0 if $U > p$. So in line 3, we are accepting each proposed sample x_i if a standard uniform variate u_i satisfies $u_i \leq \sin^2 x_i$.

In this example, we found we accepted about 430,000 samples.

In this example, we managed to sample from the PDF in Equation 14.1,

$$f(x) = \frac{1}{Z} \exp\left(-\tfrac{1}{2}x^2\right)(\sin^2 x),$$

even though we never found out what the normalising constant Z was. This idea – that we can sample from a distribution even if we only know it up to a multiplicative constant – is a very important one that will come up a lot later in this module.

We won't go into that idea deeply now, but we briefly mention that it is very important in Bayesian statistics. In Bayesian statistics, the posterior distribution is often known only up to proportionality. That's because we have

$$\text{posterior} \propto \text{prior} \times \text{likelihood} \tag{14.2}$$

$$\pi(\theta \mid x) \propto \pi(x) \times p(x \mid \theta) \tag{14.3}$$

It's often very difficult to find the normalising constant in this expression – indeed, it can be impossible in practice. So being able to sample from such a posterior distribution without finding the constant is very important for Bayesian statisticians.

14.3 How many samples?

We have mentioned that the downside of rejection sampling is that we may have to take lots of proposed samples to get enough accepted ones. Or, conversely, we may not get enough accepted samples from a fixed number of proposed samples. Remember that the accuracy of Monte Carlo estimation, for example, depends on how many (accepted) samples we get – the mean-square error scales like $1/n$ and the root-mean-square error like $1/\sqrt{n}$. So it's important to be able to get a lot of accepted samples n.

Let's examine these questions a bit closer. Write $a = \mathbb{P}(\text{accept})$ for the unconditional probability a proposal gets accepted (that is, the a priori acceptance probability, before we have seen the value $Y = x$ of the proposal). In the discrete case, this is

$$a = \mathbb{P}(\text{accept}) = \sum_x \mathbb{P}(Y = x)\, \alpha(x),$$

and in the continuous case this is

$$a = \mathbb{P}(\text{accept}) = \int_{-\infty}^{+\infty} g(x)\, \alpha(x)\, \mathrm{d}x.$$

In both cases, this can be written more succinctly as $a = \mathbb{E}\, \alpha(Y)$.

- If we take N proposals, then the expected number of accepted samples is $n = aN$.
- If we keep taking proposals until getting n acceptances, then the expected number of proposals is $N = n/a$.

If the number of proposals N is large and the unconditional acceptance probability a is not very close to 0, then the actual number of acceptances will likely be very close to aN. This is justified by the law of large numbers (and the central limit theorem). Similarly, if the desired number of acceptances n is large and a is not very close to 0, then the number of required proposals will likely be very close to n/a. Thus it is in our interest to make a as large as we can: this gives us the most acceptances, or requires the fewest proposals.

We have to be careful when the unconditional acceptance probability is very close to 0. In that case, there can be a lot of variability required in either the (very small) number of accepted samples or the (very large) number of required proposals. This unpredictability is another reason to avoid rejection sampling where the unconditional acceptance probability a is very small.

Next time. *We look closer at rejection sampling, and in particular how we can target rejection sampling at a given distribution using the "envelope" method.*

Summary:

- In rejection sampling, we accept a proposed sample $Y = x$ with probability $\alpha(x)$.
- If the PDF of a proposed sample is g, then the PDF of an accepted sample is proportional to $\alpha(x)\, g(x)$.
- When the acceptance probability is low, rejection sampling can require a lot of proposed samples to get enough accepted samples.

Read more: Voss, *An Introduction to Statistical Computing*, Subsections 1.4.1 and 1.4.3.

15 Envelope rejection sampling I

Last time, we started looking at rejection sampling. We found that if you start with a proposal distribution Y with PDF g, then accept a proposal $Y = x$ with probability $\alpha(x)$, then the PDF f of a sample conditional on acceptance satisfies $f(x) \propto \alpha(x)g(x)$.

The problem is that this sort of gets the problem backwards. If we pick a proposal distribution g and acceptance function α, this tells us what the sample distribution f will turn out to be. But we are interested in the reverse question: If we want to target a distribution f, what proposal distribution g and acceptance function α do we need to get it?

Today we look at how to do that with the "envelope" method.

15.1 Sampling under the curve

We will start with a slightly different discussion, to try to motivate our approach. (If at any point you find this section confuses more than it helps, you can skip straight to the definition in the next section.)

Consider a PDF f, and draw f as a curve. We know that $f(x)$ is always positive, so this curve is always above the x-axis. We also know that $\int_{-\infty}^{+\infty} f(x)\,\mathrm{d}x = 1$, so the total area under the curve is 1.

Suppose we pick a point under the curve, uniformly at random, and then look at its x-coordinate X. What is the PDF of X?

[picture]

Well, the probability that X lies in the interval $[a, b]$ is the probability the point we pick lies in the part of the curve between a and b, which is the area of that area divided by the total area under the curve 1. So

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f(x)\,\mathrm{d}x.$$

But that's just the probability associated with the PDF f, so X has PDF f.

[picture]

This gives us a way – in theory, at least – to sample from a PDF f. Simply pick a point at random under the curve, and take its x-coordinate. Unfortunately, there is no known way to do this directly in general, so it doesn't really help.

However, suppose we could find a function h that (a) is positive everywhere, (b) has finite area under the curve, (c is above f everywhere, (d) and that we *could* sample a point under. We could call such a curve an **envelope** for f. We could then propose a point picked at random under the envelope h, reject it if it were above the curve f, but accept it if it were under the curve f. An accepted point would be uniform under the curve f, so its x-coordinate would have PDF f, as required.

[picture]

But how can we find such an envelope h we can use?

If the area under the envelope h is c, with $1 \leq c < \infty$ (the area must be at least 1, if the curve is above f), then h must be over the form $h(x) = cg(x)$ for a PDF g. But picking the x-coordinate point at random under g is just sampling from the random variable that has PDF g. And the probability that a point with x-coordinate x from under the envelope $cg(x)$ also lies under f is $f(x)/cg(x)$, so that would be our acceptance probability.

This tells us how to perform envelope rejection sampling.

15.2 The envelope rejection sampling algorithm

If I lost you during my motivational discussion, now is the time to switch back on! We are going to set out the steps of the **envelope rejection sampling** algorithm.

- **Goal:** To sample from a random variable X with PDF f.
- **We will need:** A random variable Y with PDF g that we can sample from, and a finite constant c such that $f(x) \leq cg(x)$ for all x.
- **Step 1:** Sample a proposal Y from the PDF g.
- **Step 2:** Accept the proposal Y with probability $f(Y)/cg(Y)$; otherwise reject Y. If accepted, *end*. If rejected, *go back to Step 1*.

Note that for given f and g, there's no guarantee a finite constant c exists such that $f(x) \leq cg(x)$. Informally, we need g to have tails that are "at least as heavy" as the tails of f.

We should check this really does sample from f. We saw last time that the PDF of a sample had PDF $f(x) \propto g(x)\,\alpha(x)$. Here, that is

$$f(x) \propto g(x)\,\frac{f(x)}{cg(x)} = \frac{1}{c}\,f(x),$$

as required.

In envelope rejection sampling, we reject proposals x with probability $f(x)/cg(x)$. But we saw last time that rejecting samples is bad – higher rejection probabilities leave us with fewer accepted samples (or requiring more proposals to get the same number of accepted samples). So, for given f and g, it would be good to pick c as small as possible. We must have $f(x) \leq cg(x)$ for all x, so we must have $c \geq f(x)/g(x)$ for all x, and therefore $c \geq \sup_x f(x)/g(x)$. (Here, "sup" means "supremum" – it's like the maximum, except allows for the fact the maximum might only be achieved in a limit, such as $x \to -\infty$ or $x \to +\infty$.) The best possible c, therefore is to take c to *equal* this maximum, $c = \sup_x f(x)/g(x)$, if it's possible to calculate it.

15.3 Examples

Example 15.1. Consider the Wigner semi-circle distribution with PDF

$$f(x) = \frac{2}{\pi}\sqrt{1-x^2} \qquad -1 \leq x \leq 1.$$

We need to choose the envelope that surrounds the PDF f. Here's one choice. Let $Y \sim \mathrm{U}[-1, 1]$, which has PDF

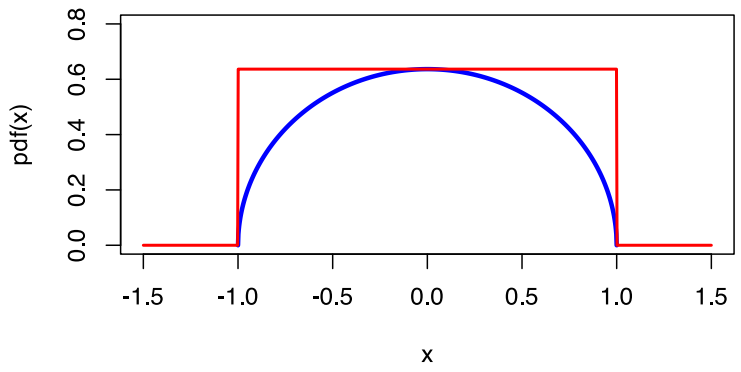
$$g(x) = \tfrac{1}{2} \qquad -1 \leq x \leq 1,$$

which well give a simple "box" around f for the envelope. We can also generate these samples easily, for example by using the inverse transform $Y = 2U - 1$ for $U \sim \mathrm{U}[0, 1]$.

The maximum point of f is at $x = 0$, where $f(0) = \frac{2}{\pi}$ and $g(0) = \frac{1}{2}$. So we see that $c \geq \frac{4}{\pi}$ will be sufficient to ensure $f(x) \leq cg(x)$ for all x, with the box surrounding the semi-circle. We want to take the smallest possible c, so will take $c = \frac{4}{\pi}$, with the box *just* touching the semi-circle at its top.

```
pdf <- function(x) (2 / pi) * sqrt(1 - x^2)
env <- function(x) (4 / pi) * (1 / 2) * (-1 <= x & x <= 1)
```

```
curve(
  pdf, n = 1001, from = -1, to = 1,
  xlim = c(-1.5, 1.5), ylim = c(0, 0.8), lwd = 3, col = "blue"
)
curve(env, n = 1001, add = TRUE, lwd = 2, col = "red")
```

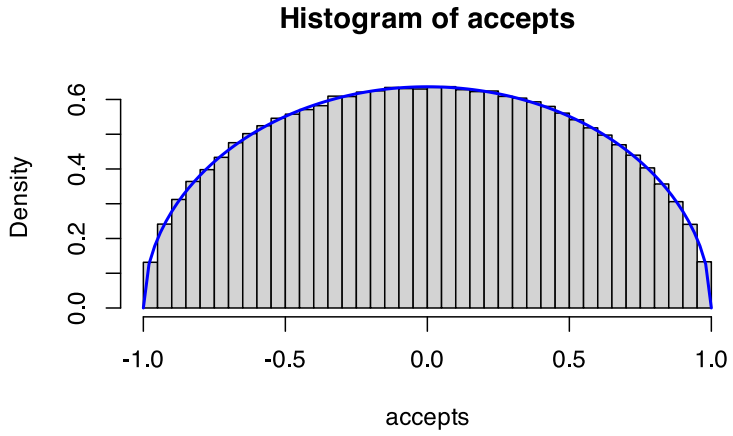


In my code, I've set it up so that `env()` (the envelope function) is c times g.

```
n_prop <- 1e6
props <- 2 * runif(n_prop) - 1
accepts <- props[runif(n_prop) <= pdf(props) / env(props)]

length(accepts)
```

```
[1] 785263
```

```
hist(accepts, probability = TRUE, breaks = 40)
curve(pdf, add = TRUE, lwd = 2, col = "blue")
```



We see that we accepted about 78% or 79% of proposals, and the histogram is an excellent fit to the semicircle distribution.

Example 15.2. Suppose we want to sample from the "half-normal" distribution

$$f(x) = \sqrt{\frac{2}{\pi}} \exp\left(\tfrac{1}{2}x^2\right) \qquad x \geq 0.$$

Let's take $Y \sim \text{Exp}(1)$ with $g(x) = \mathrm{e}^{-x}$. We know this can be easily sampled from, using the inverse transform $Y = -\log U$. The exponential also has fatter tails than the normal, so this should work.

To find an appropriate c, we consider

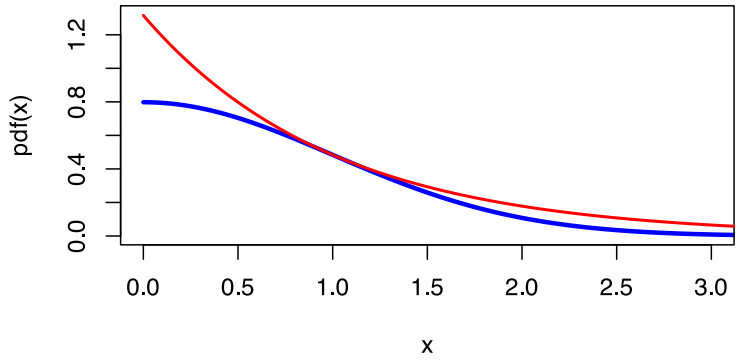
$$\frac{f(x)}{g(x)} = \frac{\sqrt{\frac{2}{\pi}}\exp(\frac{1}{2}x^2)}{\exp(-x)} = \sqrt{\frac{2}{\pi}} \exp\left(-\tfrac{1}{2}x^2 + x\right).$$

This is maximised where $-\frac{1}{2}x^2 + x$ is maximised which (by differentiating and setting equal to 0) is at $x = 1$. So we take the best possible c, which is

$$c = \frac{f(1)}{g(1)} = \sqrt{\frac{2}{\pi}} \exp\left(-\tfrac{1}{2} + 1\right) = \sqrt{\frac{2\mathrm{e}}{\pi}}.$$

```
pdf <- function(x) sqrt(2 / pi) * exp(-x^2 / 2)
env <- function(x) sqrt(2 * exp(1) / pi) * exp(-x)
```

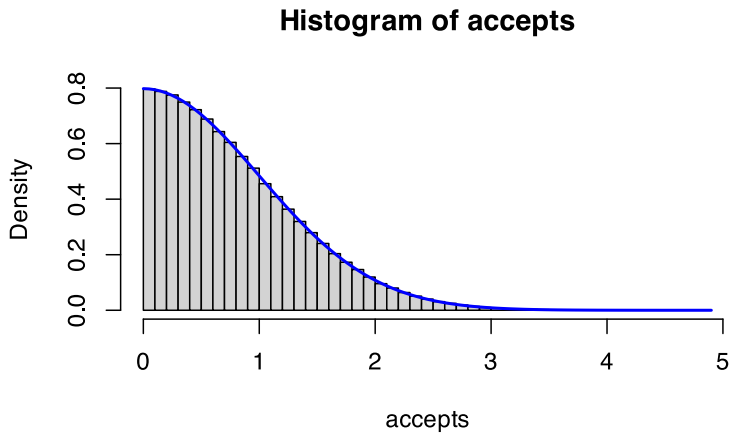
```
curve(
  pdf, n = 1001, from = 0, to = 4,
  xlim = c(0, 3), ylim = c(0, 1.32), lwd = 3, col = "blue"
)
curve(env, n = 1001, from = 0, to = 4, add = TRUE, lwd = 2, col = "red")
```



```r
n_prop <- 1e6
props <- -log(runif(n_prop))
accepts <- props[runif(n_prop) <= pdf(props) / env(props)]

length(accepts)
```

```
[1] 760751
```

```r
hist(accepts, probability = TRUE, breaks = 40)
curve(pdf, add = TRUE, lwd = 2, col = "blue")
```



We accept roughly 760,000 proposals, and get an excellent fit to the half-normal distribution.

Next time. *We study envelope rejection sampling more closely, and complete this part of the module on random number generation*

Summary:

- For envelope rejection sampling from a PDF f, we need a PDF g from which we can sample and a constant c such that $f(x) \leq cg(x)$ for all x.
- We propose samples from g, and accept a proposal x with probability $f(x)/cg(x)$.
- To keep the acceptance probability high, we want to pick c as small as possible.

Read more: Voss, *An Introduction to Statistical Computing*, Subsections 1.4.2.

16 Envelope rejection sampling II

Last time we discussed envelope rejection sampling. The process was the following:

- **Goal:** To sample from a random variable X with PDF f.
- **We will need:** A random variable Y with PDF g that we can sample from, and a finite constant c such that $f(x) \leq cg(x)$ for all x.
- **Step 1:** Sample a proposal Y from the PDF g.
- **Step 2:** Accept the proposal Y with probability $f(Y)/cg(Y)$.

While we covered the most important facts about envelope rejection sampling last time, there are a few quick matters to mop up today.

16.1 Acceptance probability

Conditional on seeing a proposal $Y = x$, we saw that the acceptance probability is

$$\alpha(x) = \frac{f(x)}{cg(x)}.$$

For any particular x, this increases as c decreases, which is why we wanted c as small as possible, subject to $f(x) \leq cg(x)$ for all x.

But what is the overall acceptance probability – unconditionally, before we've seen the value of the proposal?

As we saw before, the overall acceptance probability is

$$\mathbb{E}\,\alpha(Y) = \int_{-\infty}^{+\infty} \alpha(x)\, g(x)\, \mathrm{d}x.$$

Substituting in our value of α, we have

$$\mathbb{E}\,\alpha(Y) = \int_{-\infty}^{+\infty} \frac{f(x)}{cg(x)}\, g(x)\, \mathrm{d}x = \frac{1}{c} \int_{-\infty}^{+\infty} f(x)\, \mathrm{d}x = \frac{1}{c},$$

since the integral of PDF f is 1. Thus the unconditional acceptance probability is therefore $1/c$. To put it another way, the expected number of proposals per acceptance is c.

This is another even more direct reason why we should want to take c as small as possible.

Example 16.1. In Example 15.1, we took $c = \frac{4}{\pi}$, so $1/c = \frac{\pi}{4} = 0.785$. We saw that we accepted 78% or 79% of the proposals.

In Example 15.2, we took

$$c = \sqrt{\frac{2\mathrm{e}}{\pi}} \qquad \frac{1}{c} = \sqrt{\frac{\pi}{2\mathrm{e}}} = 0.760.$$

We saw that we accepted about 76% of proposals.

16.2 Unnormalised measures

We briefly mentioned in Lecture 14 that it can be useful to be able to be able to sample from PDFs even when we only know the PDF up to proportionality, and that this is particularly useful in Bayesian statistics.

Suppose we want to sample from a PDF

$$f(x) = \frac{1}{Z}\,\mu(x),$$

where the **measure** μ is known, but the normalising constant

$$Z = \int_{-\infty}^{+\infty} \mu(x)\,\mathrm{d}x$$

is unknown (but finite). Can we do this with envelope rejection sampling?

The answer is yes – and algorithm is basically exactly the same, just with f replaced by μ.

- **Goal:** To sample from a random variable X with PDF proportional to the measure μ.
- **We will need:** A random variable Y with PDF g that we can sample from, and a finite constant c such that $\mu(x) \leq cg(x)$ for all x.
- **Step 1:** Sample a proposal Y from the PDF g.
- **Step 2:** Accept the proposal Y with probability $\mu(Y)/cg(Y)$; otherwise reject Y. If accepted, *end*. If rejected, *go back to Step 1*.

To check that this still works, we remember that the accepted PDF is proportional to

$$\alpha(x)\,g(x) = \frac{\mu(x)}{cg(x)}\,g(x) = \frac{1}{c}\mu(x) \propto \mu(x) \propto f(x).$$

In the unnormalised case, $1/c$ is only proportional to, not directly equal to, the unconditional acceptance probability. So we still want c as small as possible, but can't give it a direct interpretation as an acceptance probability.

Example 16.2. The **von Mises distribution** is a distribution on $[0, 2\pi)$ with PDF

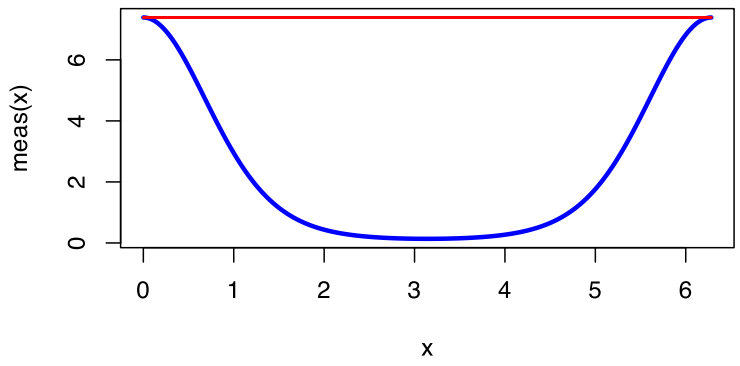
$$f(x) \propto \mu(x) = \exp\left(\kappa \cos(x - m)\right),$$

for some parameters $m \in [0, 2\pi)$ and $\kappa \geq 0$. The von Mises distribution is used to model data on a circle, with x being the angle around the circle. Circular data might be a compass direction (north/south/east/west), or a time of day (with the circle representing the hours from midnight to midnight). The von Mises distribution is sort of a "circular equivalent" of the normal distribution on the line, with m being the mean value and κ playing a similar role $1/\sigma^2$. There is no closed form for the constant of proportionality.

We wish to sample from the von Mises distribution with $m = 0$ and $\kappa = 2$, where the unnormalised measure is $\mu(x) = \exp(2 \cos x)$. We choose the uniform distribution $g(x) = \frac{1}{2\pi}$ on $[0, 2\pi)$ for our envelope, which we can easily sample from as $Y = 2\pi U$. To choose c, we note that the maximum of $\mu(x)$ is at $x = 0$, where it takes the value $\exp(2)$. So we take $c = \mu(0)/g(0) = 2\pi e^2$.

```
meas <- function(x) exp(2 * cos(x))
env  <- function(x) 2 * pi * exp(2) * (1 / (2 * pi)) * (x <= 2 * pi)
```

```
curve(
  meas, n = 1001, from = 0, to = 2 * pi,
  lwd = 3, col = "blue"
)
curve(env, n = 1001, add = TRUE, lwd = 2, col = "red")
```

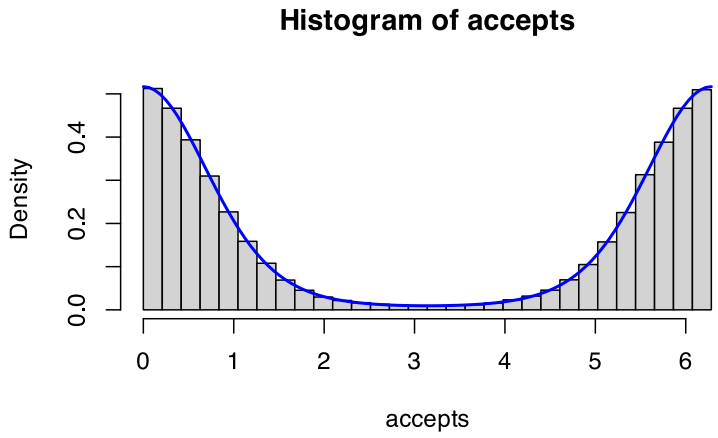


(This graph might look a little odd, but remember that 0 and 2π are the same part of the circle, so this measure takes its largest values around that "join".)

```
n_prop <- 1e6
props <- 2 * pi * (runif(n_prop))
accepts <- props[runif(n_prop) <= meas(props) / env(props)]

length(accepts)
```

```
[1] 307849
```

```
hist(accepts, probability = TRUE, breaks = 2 * pi * (0:30) / 30)
curve(meas(x) / 14.3, add = TRUE, lwd = 2, col = "blue")
```



We accepted about 31% of proposals.

16.3 Summary of Part II

This completes our study of random number generation. This would be a good time to summarise what we have learned.

First we discussed generating randomness.

- We can generate random numbers uniform on $[0, 1]$ through true physical randomness or by a pseudorandom number generator.
- LCGs are one type of pseudorandom number generator. An LCG is a recurrence $x_{n+1} = (ax_n + c) \bmod m$.

- Conditions for an LCG to have full period of m are given by the Hull–Dobell theorem: if m is a power of 2, then we need c to be odd and a to be 1 mod 4. [**Note:** *In the lecture, I wrongly said we need c to be even; in fact, we need c to be odd.*]

Then we discussed manipulating standard uniform samples into other distributions.

- The inverse transform method uses the cumulative distribution function: set $U = F(X)$ and invert to make X the subject.
- Discrete random variables can be simulated by splitting up the intervals $[0, 1]$ into segments of length $p(x_i)$, then seeing which segment U falls into.
- Two normal distributions can be sampled using the Box–Muller theorem and polar coordinates. Let R be Rayleigh distributed, Θ be uniform on $[0, 2\pi)$, then set $X = R\cos\Theta$ and $Y = R\sin\Theta$.
- We can also get different distributions by accepting a proposed sample $Y = x$ from g with probability $\alpha(x)$. The PDF of an accepted sample is $f(x) \propto \alpha(x)\, g(x)$.
- Envelope rejection sampling is a way to target rejection sampling a PDF f, by choosing an "envelope" $cg(x)$, and accepting a sample from g with probability $f(x)/cg(x)$.
- Rejection sampling works best when the acceptance probability is made as large as possible.

Next time. *We begin our study on MCMC: Markov chain Monte Carlo.*

Summary:

- Envelope rejection sampling with envelope $cg(x)$ has unconditional acceptance probability $1/c$.
- Envelope rejection sampling works even if the desired distribution is only know up to a proportionality constant.

Read more: Voss, *An Introduction to Statistical Computing*, Subsections 1.4.2.

Problem Sheet 3

Full solutions are now available.

This is Problem Sheet 3, which covers material from Lectures 12 to 16. You should work through all the questions on this problem sheet in advance of the problems class, which takes place in the lecture of **Thursday 13 November**.

This problem sheet is to help you practice material from the module and to help you check your learning. It is *not* for formal assessment and does not count towards your module mark.

If you want some brief informal feedback on **Question 3 parts (a) and (b) and Question 6** (marked), you should submit your work electronically through Gradescope via the module's Minerva page by **1400 on Tuesday 11 November**. (If you hand-write solutions on paper, the easiest way to scan-and-submit that work is using the Gradescope app on your phone.) I will return some brief comments on your those two questions by the problems class on Thursday 13 November. Because this informal feedback, and not part of the official assessment, I cannot accept late work for any reason – but I am always happy to discuss any of your work on any question in my office hours.

Full solutions will be released on Friday 14 November.

Part III

MCMC

17 Markov chains in discrete space

17.1 Markov chains and MCMC

In the first part of this module, we looked at Monte Carlo methods, where we estimated $\theta = \mathbb{E}\,\phi(X)$ by

$$\hat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} \phi(X_i)$$

. To do this, we need samples $X_1, X_2, \ldots, X_n$ that (a) come from the exactly the same distribution as X, and (b) are independent of each other.

We can often manage to generate random samples like these if X has a relatively simple distributions X: for example, we can use the inverse transform method or envelope rejection sampling (or a built-in R function). These simple distributions tend to be one-dimensional distributions, sometimes with a simple dependence on one parameter.

However, these random number generation methods are often not available when dealing with very complex distributions X. For example, these might not be one-dimensional but rather living in some very high-dimensional space. Or they might depend on many parameters – and those parameters might themselves be drawn from random distributions (a so-called "hierarchical model").

For more complicated distributions, we can make progress by loosening the assumption that $X_1, X_2, \ldots, X_n$ are perfect IID copies of X.

- Rather than the X_i having *exactly* the same distribution as X, we might be willing for them to have approximately the same distribution as X, or to converge to that distribution as i gets large.
- Rather than having the X_i be completely independent, we could let them have some dependence; but we will want to set things up so that we limit the dependence so there is only "light" dependence and so that we understand the dependence structure well.

The way will do this is to allow $X_1, X_2, ...$ to be a random process (or "stochastic" process) that has a particular dependence structure known as the **Markov property**. Such a process is known as a **Markov chain**. Random variables in Markov chains can be shown (under certain conditions) to tend to a certain distribution – we will want to set things up so that that "limiting distribution" is the distribution X we want to sample from.

Using a Monte Carlo estimator where the samples $X_1, X_2, ...$ are not IID but rather form a Markov chain is known as **Markov chain Monte Carlo** – almost always referred to by the abbreviation **MCMC**. And "MCMC" has become to be used more widely for generating samples according to a Markov chain even if those samples aren't necessarily used for Monte Carlo estimation. MCMC is one of the most important ideas in statistics in the second-half of the 20th and in the 21st centuries, and is especially important in Bayesian statistics.

In this part of the of the module, we will study MCMC in depth. We will take a brief tour through the theory of Markov chains, then talk about how to use the output of a Markov chain for Monte Carlo estimation. We will look specifically at the **Metropolis–Hastings algorithm**, which is one way of setting up a Markov chain to have a specific distribution as its limiting distribution, and has some properties in common with rejection sampling ideas we have already seen.

The schedule will be:

- Today and Lecture 18: Theory of Markov chains in discrete space
- Lecture 19: Metropolis–Hastings algorithm in discrete space
- Lecture 20: Theory of Markov chains in continuous space
- Lecture 21: Metropolis–Hastings algorithm in continuous space
- Lectures 22 and 23: MCMC in practice (including for Bayesian statistics).

Some of you may have studied Markov chains before – for example, in the Leeds second-year module MATH2750 Introduction to Markov Processes. If so, you should find that today and the next lecture are just a brief reminder of things you already know, but the rest of the material is likely to be new. However, I will not assume any pre-existing knowledge of Markov chains, and will teach you everything you need to know for this module.

17.2 Introduction to Markov chains

A **Markov chain** in discrete time $i = 1, 2, ...$ and discrete space $\mathcal{S}$ is a sequence of random variables $(X_1, X_2, X_3, ...)$ taking values in $\mathcal{S}$. The random variables are not independent, but the dependence of any of the random variables is limited to just the random variable before it in the list. That is, the next state X_{i+1} can depend on on the current state X_i; but, given the

current state X_i, it has no further dependence on the past states $X_{i-1}, X_{i-2}, \dots, X_2, X_1$. This is known as the "Markov property".

Think of playing a simple board game where you roll a dice and move that many squares along the board. Let X_i be the current square you are on. The the next square you land on, X_{i+1}:

- is random – because it depends on the roll of the dice;
- depends on which square X_i you are on now – because the value of dice roll will be added to your current square;
- *given* the square X_i you are on now, it doesn't depend which sequence of squares $X_1, X_2, \dots, X_{i-1}$ you previously landed on to get there.

Definition 17.1. A sequence of random variables $(X_i) = (X_1, X_2, \dots)$ taking values in a countable state space $\mathcal{S}$ is said to be a **Markov chain** or to have the **Markov property** if

$$\mathbb{P}(X_{i+1} = x_{i+1} \mid X_i = x_i, X_{i-1} = x_{i=1}, \dots, X_1 = x_1) = \mathbb{P}(X_{i+1} = x_{i+1} \mid X_i = x_i)$$

for all $i = 1, 2, \dots$ and for all $x_1, \dots, x_{i-1}, x_i, x_{i+1} \in \mathcal{S}$ such that the conditional probability is defined.

Example 17.1. Consider a simple model of an unreliable printer:

- On day 1, the printer is working.
- If the printer is working, then the next day there is a 90% chance it is still working, but a 10% chance it has broken.
- If the printer is broken, then the next day there is a 50% chance it has been mended, but a 50% chance it is still broken.

We can model this as a Markov chain on the state space $\mathcal{S} = \{1, 2\}$, where state 1 denotes that the printer is working and state 2 denotes that the printer is working. We have

$$\mathbb{P}(X_{i+1} = 1 \mid X_i = 1) = 0.9 \qquad \mathbb{P}(X_{i+1} = 2 \mid X_i = 1) = 0.1 \tag{17.1}$$
$$\mathbb{P}(X_{i+1} = 1 \mid X_i = 2) = 0.5 \qquad \mathbb{P}(X_{i+1} = 2 \mid X_i = 2) = 0.5. \tag{17.2}$$

Example 17.2. Consider the **simple random walk** on $\mathcal{S} = \mathbb{Z}$. We start at $X_1 = 0$. At each time step, we move up 1 with probability p and down one with probability $q = 1 - p$; so

$$\mathbb{P}(X_{i+1} = y \mid X_i = x) = \begin{cases} p & \text{if } y = x + 1 \\ q & \text{if } y = x - 1 \\ 0 & \text{otherwise.} \end{cases}$$

If $p = q = \frac{1}{2}$, this is called the **simple symmetric random walk**.

We can also write this as

$$X_{i+1} = X_i + Z_i, \tag{17.3}$$

where the Z_i are IID with distribution

$$Z_i = \begin{cases} +1 & \text{with probability } p \\ -1 & \text{with probability } q. \end{cases}$$

Any Markov chain with the structure Equation 17.3 for an IID sequence (Z_i) is called a **random walk**. If the Z_i are symmetric, in that $\mathbb{P}(Z_i = +z) = \mathbb{P}(Z_i = -z)$ for all z, then it is a **symmetric random walk**.

In both the Markov chains we have looked at – and, indeed, all the Markov chains we will ever look at – the **transition probability** $p(x, y) = \mathbb{P}(X_{i+1} = y \mid X_i = x)$ was the same for all i. That is, the probability $p(x, y)$ of moving from x to y does not depend on which timestep i we are at. This is called being **time homogeneous**.

Once we have the notation $p(x, y)$ for the transition probability, it will in fact be useful to write them in a matrix $\mathsf{P} = (p(x, y))$, called the **transition matrix**.

For the two-state "unreliable printer" Markov chain, the transition matrix is

$$\mathsf{P} = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}.$$

For the simple random walk, the transition matrix is the "infinite matrix"

$$\mathsf{P} = \begin{pmatrix} \ddots & \ddots & & & & \\ \ddots & 0 & p & & & \\ & q & 0 & p & & \\ & & q & 0 & p & \\ & & & q & 0 & \ddots \\ & & & & \ddots & \ddots \end{pmatrix}$$

This has 0s down the diagonal (representing the probability 0 of staying still), p one place to the right of the diagonal (representing the probability of moving up 1), and q one place to the left of the diagonal (representing the probability of moving down 1). Blank spaces in this matrix denotes 0s.

The xth row of a transition matrix represents the probabilities of moving from x to each of the other states. Thus each row must consist of non-negative numbers that add up to 1, as is the case in both of our examples.

17.3 Simulation of Markov chains

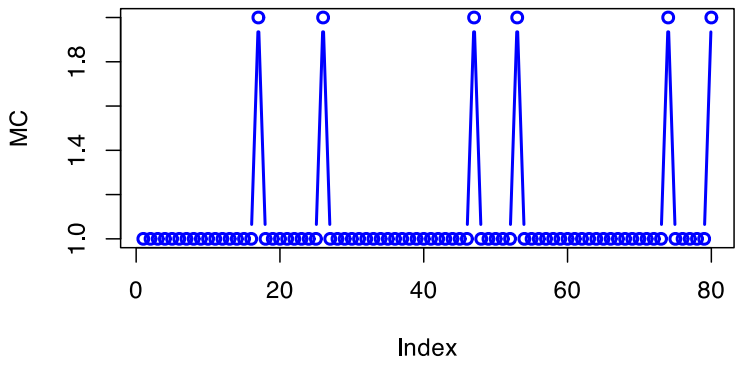
We can take n samples from a finite-state Markov chain in R with the following function. In the function `rmarkov()`, `n` is the number of samples (or steps) to take, `trans` is the transition matrix P, and `initial` is the initial state X_1.

```
rmarkov <- function(n, trans, initial) {
  states <- nrow(trans)
  MC <- rep(0, n)

  MC[1] <- initial
  for (i in 1:(n - 1)) MC[i + 1] <- sample(states, 1, prob = trans[MC[i], ])

  return(MC)
}
```

The key line is the `for` loop in the penultimate line. Here, the next state X_{i+1} is chosen by sampling one state from the state space with probabilities according to the current state's row of the transition matrix.

Example 17.3. Let's simulate the two-state broken printer Markov chain from Example 17.1.

```
trans <- matrix(c(0.9, 0.1, 0.5, 0.5), 2, 2, byrow = TRUE)
initial <- 1
MC <- rmarkov(80, trans, initial)
plot(MC, col = "blue", lwd = 2, type = "b")
```



In the first line, we entered a 2×2 matrix using the code `matrix(P, 2, 2)`, where `P` was a vector of length $2 \times 2 = 4$. R default is to fill up the matrix column at a time; I personally find it more logical (at least when working with Markov chains) to fill up a matrix row at a time, so I used `byrow = TRUE` to ensure that.

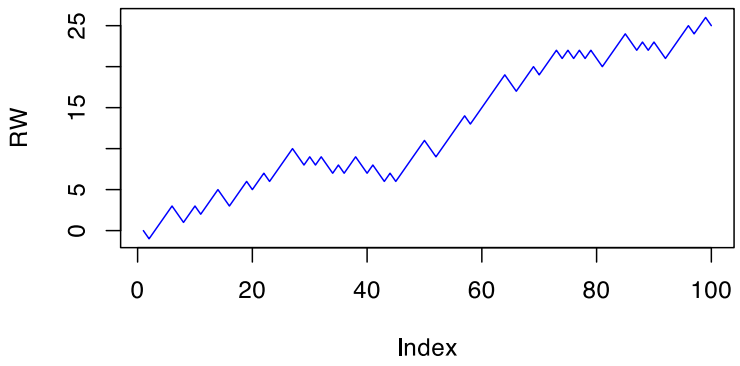
Our sample shows that printer spent most of the time working (state 1), but when it did break (state 2) it usually got mended pretty quickly.

Example 17.4. We can simulate the simple random walk from Example 17.4 with the following code.

```
rrw <- function(n, up) {
  RW <- rep(0, n)
  down <- 1 - up

  RW[1] <- 0
  for (i in 1:(n - 1)) {
    RW[i + 1] <- RW[i] + sample(c(1, -1), 1, prob = c(up, down))
  }

  return(RW)
}
```

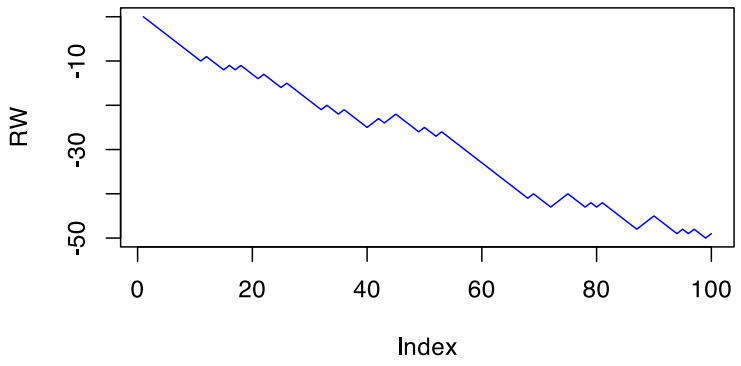
So with $p = 0.6$, we have

```
RW <- rrw(100, 0.6)
plot(RW, col = "blue", type = "l")
```



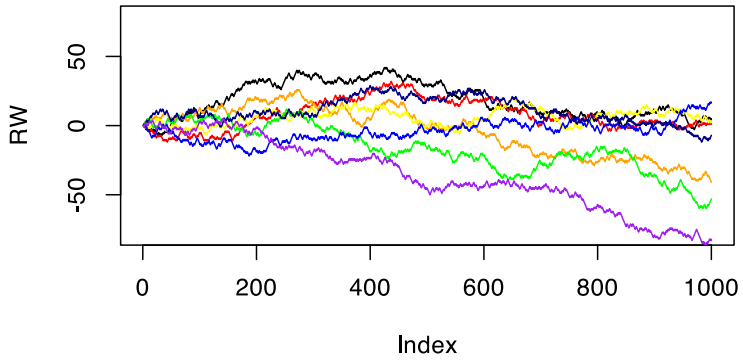
which goes up on average. With $p = 0.3$, we have

```r
RW <- rrw(100, 0.3)
plot(RW, col = "blue", type = "l")
```



which goes down on average. The simple symmetric random walk, with $p = 0.5$ can be much more unpredictable.

```r
RW <- rrw(1000, 0.5)
plot(RW, col = "black", type = "l", ylim = c(-80, 80))

cols <- c("red", "orange", "yellow", "green", "blue", "darkblue", "purple")
for (i in 1:7) {
  RW <- rrw(1000, 0.5)
  points(RW, col = cols[i], type = "l")
}
```



Next time. *We continue our whistle-stop tour of discrete-space Markov chains.*

Summary:

- A Markov chain is a stochastic process where the next step X_{i+1} depends on the current step X_i, but, given current step X_i, does not depend on the past $X_1, \ldots, X_{i-1}$.
- A Markov chain is governed by its transition probabilities $p(x, y) = \mathbb{P}(X_{i+1} = y \mid X_i = x)$. These are written in the transition matrix P, whose rows add up to 1.
- The simple random walk on the integers at each step goes up 1 with probability p and down 1 with probability q. If $p = q = \frac{1}{2}$, it is a simple symmetric random walk.

Read more:

- Voss, *An Introduction to Statistical Computing*, Subsections 2.3.1
- my notes for MATH2750 Introduction to Markov Processes, Lectures 1, 2 and 5.

18 Markov chains in the long run

18.1 *n*-step transition probabilities

Last time we saw that the probability of a "1-step transition" from x to y is

$$p(x,y) = \mathbb{P}(X_{i+1} = y \mid X_i = x).$$

But what is the probability of a "2-step transition"

$$p^{(2)}(x,y) = \mathbb{P}(X_{i+2} = y \mid X_i = x)?$$

Well, the first step will be from x to some other state z; then the second step will have to go from that z to y. Thus we have

$$\begin{aligned} p^{(2)}(x,y) &= \mathbb{P}(X_{i+2} = y \mid X_i = x) && (18.1)\\ &= \sum_{z\in\mathcal{S}} \mathbb{P}(X_{i+1} = z \mid X_i = x)\, \mathbb{P}(X_{i+2} = y \mid X_{i+1} = z, X_i = x) && (18.2)\\ &= \sum_{z\in\mathcal{S}} \mathbb{P}(X_{i+1} = z \mid X_i = x)\, \mathbb{P}(X_{i+2} = y \mid X_{i+1} = z) && (18.3)\\ &= \sum_{z\in\mathcal{S}} p(x,z)\, p(z,y). && (18.4) \end{aligned}$$

Here, in the third line we used the Markov property to delete the unnecessary conditioning on X_i.

What we have here, though, is the (x,y)th entry of the matrix square $\mathsf{P}^2 = \mathsf{P}\,\mathsf{P}$. That is, to get the matrix of 2-step transitions, we simply take the second matrix power of the matrix of 1-step transitions.

In the same way we can calculate an n-step transition probability $p^{(n)}(x,y) = \mathbb{P}(X_{i+n} = y \mid X_i = x)$ by summing over all the potential paths $x \to z_1 \to \cdots \to z_{n-1} \to y$ of length n from x to y. This gives

$$p^{(n)}(x,y) = \sum_{z_1,\ldots,z_{n-1}\in\mathcal{S}} p(x,z_1)\, p(z_1,z_2) \cdots p(z_{n-2},z_{n-1})\, p(z_{n-1},y).$$

This is the expression for the (x,y)th entry of the nth matrix power $\mathsf{P}^n = \mathsf{P}\,\mathsf{P}^{n-1} = \mathsf{P}^{n-1}\,\mathsf{P}$, so we can find all the n-step transition probabilities from P^n.

(Remember that the matrix power P^n is what we get by multiplying the whole matrix P by itself n times, using the rules for multiplying matrices. It's not just what we get from taking the nth power of the number in each entry. In R, proper matrix multiplication is `P %*% P`, while `P * P` is simply entry-wise multiplication.)

Example 18.1. Let's go back to our two-state "unreliable printer" Markov chain. Here, we had

$$\mathsf{P} = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}.$$

The 2-step transition probabilities are given by

$$\mathsf{P}^2 = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix} = \begin{pmatrix} 0.86 & 0.14 \\ 0.7 & 0.3 \end{pmatrix}.$$

So if the printer is working today, there's an 86% probability it's working in two days' time, for example.

For bigger matrix powers, it's best to use a computer. In R, the following "quick and dirty" function works well for small powers. (For larger powers, I recommend finding a package with an appropriate built-in matrix power function.)

```
matrixpow <- function(M, n) {
  if (n == 1) return(M)
  else return(M %*% matrixpow(M, n - 1))
}
```

From this we find the 10-step transition probability

$$\mathsf{P}^{10} = \begin{pmatrix} 0.8334 & 0.1667 \\ 0.8332 & 0.1668 \end{pmatrix}.$$

The first row of this matrix denotes the probabilities of where we end up after 10 steps if we start in state 1, and the second row for if we start in state 2. These are very nearly the same – we have a probability ≈ 0.883 if being in state 1 and ≈ 0.167 of being in state 2, *regardless of which state we started in*. It's as if the Markov chain has forgotten where we started.

Similarly, if we look at the 11-step transition probability, that comes out as

$$\mathsf{P}^{11} = \begin{pmatrix} 0.8333 & 0.1667 \\ 0.8333 & 0.1667 \end{pmatrix},$$

which is virtually the same distribution as after 10 steps. It seems that, after a large number of steps i, that we are "settling down" to a "long run distribution" where $\mathbb{P}(X_i = 1) = 0.8333$ and $\mathbb{P}(X_i = 2) = 0.1667$, not only regardless of which state we started in but also for all large i.

We will investigate these phenomena in the next section.

18.2 Stationary distributions

Suppose our Markov chain is currently in the distribution π at step i. That is, for each $x \in \mathcal{S}$, we have $\mathbb{P}(X_i = x) = \pi(x)$. What is the probability we are in state y at the next step $i+1$? Well, conditioning on the current step, we have

$$\mathbb{P}(X_{i+1} = y) = \sum_{x \in \mathcal{S}} \mathbb{P}(X_i = x)\, \mathbb{P}(X_{i+1} = y \mid X_i = x) = \sum_{x \in \mathcal{S}} \pi(x)\, p(x, y).$$

Now if, X_{i+1} *also* has the distribution π – that is, if $\mathbb{P}(X_{i+1} = y) = \pi(y)$ – then we would remain in the distribution π a time step $i+1$. And, by the same logic, time steps $i+2, i+3, \ldots$ and forever. That seems a bit like what we saw happening in Example 18.1. We call this a **stationary distribution**.

A stationary distribution means that *the probability of being in state x* is staying the same, as $\pi(x)$. Any particular realisation of the Markov chain will, of course, continue moving between states.

Definition 18.1. Let (X_i) be a Markov chain on a discrete state space $\mathcal{S}$ with transition matrix $\mathsf{P} = (p(x, y))$. Let π be a distribution on $\mathcal{S}$, in that $\pi(x) \geq 0$ for all x and $\sum_{x \in \mathcal{S}} \pi(x) = 1$. If, for all $y \in \mathcal{S}$ we have

$$\pi(y) = \sum_{x \in \mathcal{S}} \pi(x)\, p(x, y), \tag{18.5}$$

then we say that π is a **stationary distribution**.

In matrix form, we can write Equation 18.5 as $\pi = \pi\mathsf{P}$, where π is a *row* vector (not the more common column vector holding the values of $\pi(x)$).

Solving Equation 18.5 or $\pi = \pi\mathsf{P}$ can be a bit fiddly. It's often easier to check something called the **detailed balance equations**.

Theorem 18.1. *Let (X_i) be a Markov chain on a discrete state space $\mathcal{S}$ with transition matrix $\mathsf{P} = (p(x, y))$. Let π be a distribution on $\mathcal{S}$ that solved the* ***detailed balance equations***

$$\pi(y)\, p(y, x) = \pi(x)\, p(x, y) \qquad \text{for all } x, y \in \mathcal{S}.$$

Then π is a stationary distribution.

Proof. Sum both sides over x. The left-hand side becomes

$$\sum_{x \in \mathcal{S}} \pi(y)\, p(y, x) = \pi(y) \sum_{x \in \mathcal{S}} p(y, x) = \pi(y),$$

since rows of a transition matrix sum up to 1. The right hand side becomes

$$\sum_{x \in \mathcal{S}} \pi(x)\, p(x, y).$$

Hence, we have

$$\pi(y) = \sum_{x \in \mathcal{S}} \pi(x)\, p(x, y),$$

which is the definition of a stationary distribution. □

Example 18.2. We return to Example 17.1 and Example 18.1. There's no need to check the detailed balance equations when $x = y$, so we just need

$$\pi(2)\, p(2, 1) = \pi(1)\, p(1, 2) \qquad \Longrightarrow \qquad 0.5\pi(2) = 0.1\pi(1).$$

Remembering that π must sum to 1, we get $\pi(1) = \frac{1}{6} = 0.1667$ and $\pi(2) = \frac{5}{6} = 0.8333$.

Look how that compares with our results for P^{10} and P^{11} – this π was precisely the values we saw in every row of P^n for large n.

18.3 Limit theorems

The big central theorem of Markov chains in discrete space is the following. We will highlight some technical conditions in red that we will return to later.

Theorem 18.2. *Let (X_i) be a Markov chain on a discrete state space $\mathcal{S}$ with transition matrix P. Suppose that (X_i) is irreducible and positive recurrent.*

1. *There exists a stationary distribution π, which is unique.*
2. ***(Limit theorem)** If (X_i) is also aperiodic, then $\mathbb{P}(X_n = y \mid X_1 = x) \to \pi(y)$ as $n \to \infty$ for all $y \in \mathcal{S}$, regardless of the starting state $X_1 = x$, where π is the unique stationary distribution.*
3. ***(Ergodic theorem, 1)** Write*

$$V_n(x) = \frac{1}{n} \big| \{ i = 1, 2, \dots, n : X_i = x \} \big|$$

for the proportion of the first n steps spent in state x. Then $V_n(x) \to \pi(x)$ as $n \to \infty$ for all $x \in \mathcal{S}$, regardless of the starting state X_1, where π is the unique stationary distribution.

4. ***(Ergodic theorem, 2)*** *Let ϕ be a function on the state space $\mathcal{S}$. Let X have probability mass function π, where π is the unique stationary distribution. Then*

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

as $n \to \infty$, regardless of the starting state X_1.

("Ergodic" is a word mathematicians use when talking about long-run average behaviour.)

The precise mathematical statements of this theorem are not important for this module. However, it is important to have a rough idea what the statements mean – especially part 4, which is central to the idea of Markov chain Monte Carlo we will discuss over the next lectures.

The first part tells us that (provided the technical conditions are fulfilled) we always have a stationary distribution and there's always exactly one of them. This allows us to use phrases like "where π is the unique stationary distribution" in the other parts of the theorem.

The second part tells us that any n-step transition probability $p^{(n)}(x, y)$ tends to $\pi(y)$, no matter what the value of x. In terms of the n-step transition matrix P^n, this means that every row of P^n should end up looking like an identical copy of the row vector π. That is exactly what we found in Example 18.1.

The third part tells us that, in the long run, π describes the proportion of time we spend in each state. In the "unreliable printer" example of Example 18.2, this means that the printer spends 83% of the time working and 17% of the time broken in the long run, regardless of whether it was working or broken on day 1.

The fourth part is by far the most important result for us, as it relates Markov chains back to the idea of Monte Carlo estimation. Let's look at the equation in the fourth part,

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

If the X_i were independent and identically distributed, this would just be the ordinary law of large numbers, which tells us that the Monte Carlo estimator (the left-hand side) is an accurate estimator of $\mathbb{E}\,\phi(X)$, when the number of samples is large. This result tells us that we still get a good estimator when the X_i are not IID, but rather come from a Markov chain whose stationary distribution is the PMF of X.

This fourth part is what allows us to do **Markov chain Monte Carlo** (**MCMC**): Monte Carlo estimation when the X_i are the outputs from a Markov chain. If we want to estimate $\mathbb{E}\,\phi(X)$, we just need to find a Markov chain who stationary distribution is the PMF of X, and then form the Monte Carlo estimate in the usual way. In the next lecture, the **Metropolis–Hastings algorithm** will show us a way to do find a Markov chain with a given stationary distribution.

A quick word before we end about the technical conditions in Theorem 18.2. The precise definitions are not important here, but let us say the following:

- **Irreducible** roughly means that a Markov chain is "connected up", and isn't just two Markov separate Markov chains (for example). Specifically, it must be at least *possible* to get from any state x to any other state y – maybe not in a single step, but in some finite number of steps, with probability greater than 0.
- **Aperiodic** is another technical condition, but if an irreducible Markov chain ever has a non-zero probability of staying in the same state, then this is fulfilled. The Markov chains we look at for MCMC will all have a strictly positive probability of staying put, so will be aperiodic.
- **Positive recurrence** is a highly technical condition we won't get into here.

If you do want to read more about the theory of Markov chains, and these technical conditions in particular, I recommend my lecture notes for my notes for MATH2750 Introduction to Markov Processes. This is entirely optional, though, and this knowledge is *not* required for this module and its exam.

Next time. *We will look at Markov chain Monte Carlo; specifically, how to set up a Markov chain that has a given probability mass function as its stationary distribution.*

Summary:

- The n-step transition probabilities $p^{(n)}(x, y) = \mathbb{P}(X_{i+n} = y \mid X_i = x)$ can be found from the nth matrix power P^n.
- A stationary distribution π for a Markov chain satisfies the detailed balance equations $\pi(y)\, p(y, x) = \pi(x)\, p(x, y)$.
- The ergodic theorem says that (under certain technical conditions) $\frac{1}{n} \sum_{i=1}^{n} \phi(X_i)$, where X_i is Markov chain, tends to $\mathbb{E}\, \phi(X)$, where the PMF of X is the unique stationary distribution of the Markov chain.

Read more:

- Voss, *An Introduction to Statistical Computing*, Subsections 2.3.1 and 4.1.2
- my notes for MATH2750 Introduction to Markov Processes, Lectures 7 and 9–11

19 Metropolis–Hastings in discrete space

19.1 The Metropolis–Hastings algorithm

Last time, we looked at the long-run behaviour of a Markov chain (X_i). We saw that (under certain technical conditions) the Markov chain has a unique stationary distribution π, which we can find by solving the detailed balance equations $\pi(y)\,p(y,x) = \pi(x)\,p(x,y)$.

We then saw the ergodic theorem: If ϕ be a function on the state space $\mathcal{S}$ and X has probability mass function π, then

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

as $n \to \infty$. This means we can do Monte Carlo estimation where the samples $X_1, \ldots, X_n$ are not IID, but are rather the output to a Markov chain with stationary distribution π.

So, suppose we want to estimate $\mathbb{E}\,\phi(X)$, where X has PDF π. Then all we need to find a Markov chain that has π as its stationary distribution. We could try to do that be being clever – just by thinking hard and trying to come up with one. However, that's rather difficult. Instead, the **Metropolis–Hastings algorithm** is a method to create such a Markov chain.

The Metropolis–Hastings algorithm is based on a similar idea to rejection sampling. From state $X_i = x$, we *propose* moving to some other state y, and then we accept the proposal with some acceptance probability. If we accept the proposal, we move to $X_{i+1} = y$; if we reject the proposal, we stay where we are $X_{i+1} = x$.

Nicholas Metropolis first came up with this idea when he worked with Ulam and von Neumann (of Monte Carlo fame) at the Los Alamos National Laboratories. His original idea was generalised by the Canadian statistician WK Hastings.

The Metropolis–Hastings algorithm works like this:

- We want to define a Markov chain on a state space $\mathcal{S}$, which contains the range of the probability mass function π we want to sample from.
- We have an initial state $X_1 = x_1$ and we choose a transition matrix $\mathsf{R} = (r(x,y))$ representing the proposal moves.

1. From a current state $X_i = x$ we propose moving to a new state y, where y is chosen with probability $r(x,y)$.

2. With probability
$$\alpha(x, y) = \min\left\{\frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)}, 1\right\},$$
we accept the proposal, and set $X_i = y$; otherwise we stay put, and set $X_{i+1} = x$.

3. We repeat steps 1. and 2. n times to get n samples.

So a generic Metropolis–Hastings algorithm on a finite state space in "sort-of-R-code" would look something like this:

```
# INPUTS:
# trans:   proposal transition matrix
# target:  target stationary distribution
# initial: initial state
# n:       number of samples

states <- nrow(trans)
MC <- rep(0, n)
accept <- function(x, y) {
  ratio <- (target[y] * trans[y, x]) / (target[x] * trans[x, y])
  min(ratio, 1)
}

MC[1] <- initial
for (i in 1:(n - 1)) {
  prop <- sample(1:states, 1, prob = trans[MC[i], ])
  if (runif(1) <= accept(MC[i], prop)) MC[i + 1] <- prop
  else                                 MC[i + 1] <- MC[i]
}
```

It's the last three lines that are important here. In this code `MC` records the states of our Markov chain. First we propose a move to state `prop`, according to the row of the transition matrix corresponding to the current state. Second, we accept that proposal move with probability `accept()`, where the arguments of `accept()` are the current state and the proposed state; we do this by checking whether a standard uniform `runif(1)` is less than this acceptance probability or not. If it is, we move to `prop` (last-but-one line); and if not, we stay where we are (last line).

We will show later that this algorithm really does have π as its stationary distribution.

19.2 Random walk Metropolis

There are quite a lot of cases where the proposals are **symmetric**, meaning that $r(x, y) = r(y, x)$. This is the case if, for example, the proposal probabilities are those of the simple symmetric random walk: $r(x, x + 1) = \frac{1}{2}$ and $r(x, x - 1) = \frac{1}{2}$. In the symmetric case, the acceptance probability simplifies to

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\}.$$

The symmetric case was the version originally considered by Metropolis, before Hastings generalised it to non-symmetric proposals. For this reasons, when R has this symmetry property, we often refer to the resulting algorithm just as the **Metropolis algorithm**. When the proposal probabilities are those of the simple symmetric random walk, we call it the **random walk Metropolis algorithm**.

Example 19.1. Let's do an example of the random walk Metropolis algorithm where we aim to sample from the geometric distribution with parameter $\frac{1}{3}$,

$$\pi(x) = \left(\frac{2}{3}\right)^{x-1} \times \frac{1}{3} \qquad x = 1, 2, \ldots.$$

We will start from $X_1 = 1$.

So at each step we propose moving up one with probability $\frac{1}{2}$ and moving down one with probability $\frac{1}{2}$. Since the proposals are symmetric, the acceptance probabilities are

$$\alpha(x, x+1) = \min\left\{\frac{\pi(x+1)}{\pi(x)}, 1\right\} = \min\left\{\frac{\left(\frac{2}{3}\right)^{x} \times \frac{1}{3}}{\left(\frac{2}{3}\right)^{x-1} \times \frac{1}{3}}, 1\right\} = \min\left\{\frac{2}{3}, 1\right\} = \frac{2}{3} \tag{19.1}$$

$$\alpha(x, x-1) = \min\left\{\frac{\pi(x-1)}{\pi(x)}, 1\right\} = \min\left\{\frac{\left(\frac{2}{3}\right)^{x-1} \times \frac{1}{3}}{\left(\frac{2}{3}\right)^{x} \times \frac{1}{3}}, 1\right\} = \min\left\{\frac{3}{2}, 1\right\} = 1, \tag{19.2}$$

except for

$$\alpha(1, 0) = \min\left\{\frac{\pi(0)}{\pi(1)}, 1\right\} = \min\left\{\frac{0}{\frac{1}{3}}, 1\right\} = \min\{0, 1\} = 0.$$

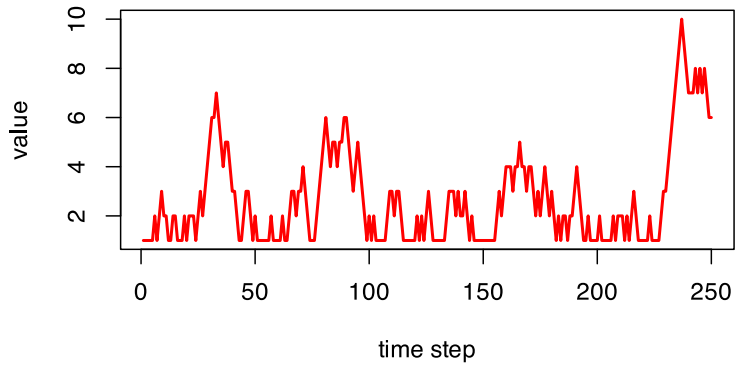
So if the proposal is up one, we accept it with probability $\frac{2}{3}$ and otherwise stay where we are. If the proposal is down one, we always accept – except going down from 1 to 0, which we always reject.

Let's try it.

```r
n <- 1e6
MC <- rep(0, n)

MC[1] <- 1
for (i in 1:(n - 1)) {
  prop <- MC[i] + sample(c(+1, -1), 1, prob = c(1/2, 1/2))
  if      (prop == 0)          MC[i + 1] <- MC[i]
  else if (prop == MC[i] - 1) MC[i + 1] <- MC[i] - 1
  else if (prop == MC[i] + 1) MC[i + 1] <- MC[i] + (runif(1) <= 2/3)
}
```

If we look at a graph of the first 250 steps of this Markov chain, we see that these aren't at all random samples from the geometric distribution – each step is either the same as the one before, one bigger, or one smaller.

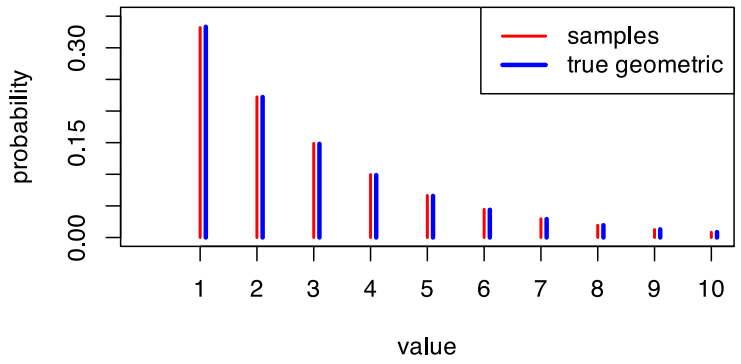
```r
plot(
  MC[1:250],
  type = "l", col = "red", lwd = 2,
  xlab = "time step", ylab = "value"
)
```



But if we look at a graph of which samples came up overall, we see that their proportions (red) are extremely close to what we would expect from the true geometric distribution (blue).

```r
plot(
  table(MC)/n,
  xlim = c(0, 10), col = "red", ylim = c(0, 0.35), lwd = 2,
```

```
  xlab = "value", ylab = "probability"
)

points(1:10 + 0.1, (2/3)^(1:10 - 1) * (1/3), col = "blue", type = "h", lwd = 3)

legend("topright", c("samples", "true geometric"),
  col = c("red", "blue"), lwd = c(2, 3)
)
```



Suppose we wanted to estimate $\mathbb{E}X^2$, where $X \sim \text{Geom}(\frac{1}{3})$. We already know how to do this the "basic" Monte Carlo way. But we can now do it the Markov chain Monte Carlo (MCMC) way, by using the output to this Markov chain.

Our estimate is the following.

```
mean(MC^2)
```

```
[1] 14.79205
```

The true answer is 15, so we are in the right area, but probably not as accurate as the basic Monte Carlo estimator with the same sample size would have been. This suggests that the dependence structure in a Markov chain might be a slight disadvantage, and may make the variance of our estimator bigger. The real strength of MCMC is when a basic Monte Carlo estimate is impossible to get – when basic MCMC is possible (such as for simple distributions like this geometric) we should probably stick with it.

19.3 Proof of stationary distribution

We have defined the Metropolis–Hastings Markov chain in terms of the proposal transition probabilities $r(x, y)$ and the acceptance probability $\alpha(x, y)$. But what are the actual transition probability $p(x, y)$ of this Markov chain?

Well, to move from x to $y \neq x$, we first have to propose that move, then we have to accept it. So we have

$$p(x, y) = r(x, y)\, \alpha(x, y) = r(x, y)\, \min\left\{ \frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)}, 1 \right\}. \tag{19.3}$$

(We can find $p(x, x)$, if we need it, by using the fact that $\sum_y p(x, y) = 0$.)

We should check that the Metropolis–Hastings algorithm really does give a Markov chain with stationary distribution π.

Theorem 19.1. *Let π be a probability mass function on a discrete state space $\mathcal{S}$, and let $\mathsf{R} = (r(x, y))$ be a transition matrix on $\mathcal{S}$. Let (X_i) be the Metropolis–Hastings Markov chain with proposal transition matrix R and acceptance probability*

$$\alpha(x, y) = \min\left\{ \frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)}, 1 \right\}.$$

Then π is a stationary distribution for (X_i).

We say "a" stationary distribution. But provided the Markov chain fulfils the technical conditions in Theorem 18.2, we know that this will be the unique stationary distribution, and that the ergodic theorem will hold.

Proof. We need to check the detailed balance equations

$$\pi(y)\, p(y, x) = \pi(x)\, p(x, y)$$

for $y \neq x$. By Equation 19.3, the detailed balance equations are

$$\pi(y)\, r(y, x)\, \min\left\{ \frac{\pi(x)\, r(x, y)}{\pi(y)\, r(y, x)}, 1 \right\} = \pi(x)\, r(x, y)\, \min\left\{ \frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)}, 1 \right\}. \tag{19.4}$$

Note that the two fractions in the first terms on the minimums are reciprocals of each other. So one of these will be greater than equal to 1, and the minimum will be 1; and one of the will be less than or equal to 1, and the minimum will be that fraction.

Suppose first that

$$\frac{\pi(x)\, r(x, y)}{\pi(y)\, r(y, x)} \geq 1 \qquad \text{and} \qquad \frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)} \leq 1.$$

Then Equation 19.4 becomes

$$\pi(y)\, r(y,x) \times 1 = \pi(x)\, r(x,y) \times \frac{\pi(y)\, r(y,x)}{\pi(x)\, r(x,y)}.$$

On the right-hand side, the two $\pi(x)\, r(x,y)$ terms cancel, so we have equality, and the detailed balance equations hold.

If, on the other hand

$$\frac{\pi(x)\, r(x,y)}{\pi(y)\, r(y,x)} \leq 1 \qquad \text{and} \qquad \frac{\pi(y)\, r(y,x)}{\pi(x)\, r(x,y)} \geq 1,$$

then the same argument works the other way around. □

Next time. *We will start our study of MCMC in continuous space by giving an overview of the theory of Markov chains in continuous space.*

Summary:

- The Metropolis–Hastings algorithm gives a way of designing a Markov chain whose stationary distribution is a given PMF π.
- We make proposal moves according to a transition matrix $\mathsf{R} = (r(x,y))$. A proposal move from x to y is accepted with probability

$$\alpha(x,y) = \min\left\{\frac{\pi(y)\, r(y,x)}{\pi(x)\, r(x,y)},\, 1\right\}.$$

- The most important example is the random walk Metropolis algorithm, where $`mathsf{R}$ is the transition matrix of the simple symmetric random walk and

$$\alpha(x,y) = \min\left\{\frac{\pi(y)}{\pi(x)},\, 1\right\}.$$

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 4.1.2.

20 Markov chains in continuous space

20.1 Markov chains with densities

In the last three lectures, we have looked at Markov chains and the Metropolis–Hastings algorithm on a discrete state space $\mathcal{S}$. This allowed us to form Markov chain Monte Carlo (MCMC) estimators for discrete random variables. But very often we want to sample from continuous random variables. So in the next two lectures we will look at Markov chains and Metropolis–Hastings in continuous space. (We are still in discrete time, though.)

The general theory of Markov chains on continuous state spaces can get very complicated – there are lots of technical conditions, and you have to deal with a "measure theoretic" approach that looks at least as much like analysis from pure mathematics as it does probability and statistics. However, this technical material is not necessary to me able to understand and use the basic ideas of MCMC in continuous space. For this reason, we will often make broad simplifying assumptions, not get into the precise definitions of technical terms, and occasionally just outright lie. (If you want to get into more of the theoretical details, I recommend Subsection 2.3.2 and Sections 4.1 and 4.2 of Voss, *An Introduction to Statistical Computing* as a good place to begin your studies.)

The concept of a Markov chain and the Markov property remain the same: the next step X_{i+1} may depend on the current step X_i, but, given the current step X_i, may not depend any further on on the past steps $X_{i-1}, X_{i-2}, \ldots, X_2, X_1$. That is,

$$\mathbb{P}(X_{i+1} \in A \mid X_i = x_i, X_{i-1} = x_{i=1}, \ldots, X_1 = x_1) = \mathbb{P}(X_{i+1} \in A \mid X_i = x_i)$$

for all $i = 1, 2, \ldots$ and for all $x_1, \ldots, x_{i-1}, x_i \in \mathcal{S}$ and $A \subset \mathcal{S}$ such that the conditional probability is defined.

Therefore, the steps of such a Markov chain will depend only on the initial state and the transition rule from moving from $X_i = x$ to $X_{i+1} \in A$. In general, we will need what is called a "transition kernel" $p(x, A)$. However, we will make the simplification that the transition rule always has a density; that is, that there exists a conditional probability density function p where $p(x, y)$ is the probability density of X_{i+1} around y given $X_i = x$. Formally, we have

$$\mathbb{P}(X_{i+1} \in A \mid X_i = x) = \int_A p(x, y) \, \mathrm{d}y.$$

This transition density $p = p(x, y)$ behaves a lot like the transition probabilities $\mathsf{P} = (p(x, y))$ in discrete space.

In the discrete case, we had

$$\sum_{y \in \mathcal{S}} p(x, y) = 1 \qquad \text{for all } x \in \mathcal{S}$$

("rows of the transition matrix sum to 1"). Similarly, in the discrete case we have

$$\int_{\mathcal{S}} p(x, y) \, \mathrm{d}y = 1 \qquad \text{for all } x \in \mathcal{S},$$

by the same argument that we have to go *somewhere* from x.

20.2 Gaussian random walk

The most important Markov chain in continuous space is the following.

Example 20.1. Consider the **Gaussian random walk** with drift μ and volatility σ. This is a random walk with transitions given by the rule $X_{i+1} = X_i + Z_i$, where the $Z_i \sim \mathrm{N}(\mu, \sigma^2)$ are IID. So at each time step, the position of the random walk is shifted by a normally distributed amount. When $\mu = 0$, we call this the **symmetric Gaussian walk**, since it moves up and down symmetrically. (The name "Gaussian random walk" is because the "Gaussian distribution" is alternative name for the normal distribution.) This can be used as the model of the price of a stock each day or (in higher dimensions) the position of a gas particle in a room.

Another way to write this is that $X_{i+1} \sim \mathrm{N}(X_i + \mu, \sigma^2)$. So the transition density is

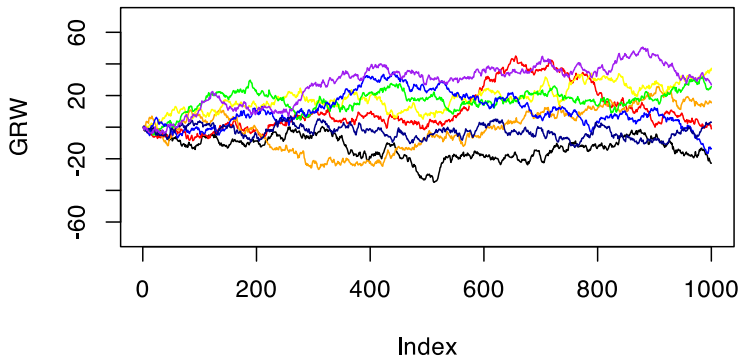
$$p(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y - (x + \mu))^2}{2\sigma^2} \right).$$

We can simulate this in R in a similar way to the simple random walk on the integers.

```
rgrw <- function(n, mean, sd) {
  GRW <- rep(0, n)
  GRW[1] <- 0
  for (i in 1:(n - 1)) GRW[i + 1] <- GRW[i] + rnorm(1, mean, sd)
  return(GRW)
}
```

You can play around with this function yourself, but generally we see similar behaviour to the simple random walk: for $\mu > 0$ (as with $p > \frac{1}{2}$) it trends fairly predictably upwards; for $\mu < 0$ (as with $p < \frac{1}{2}$) it trends fairly predictably downwards; and for $\mu = 0$ (as with with $p = \frac{1}{2}$) it is more unpredictable.

```
GRW <- rgrw(1000, 0, 1)
plot(GRW, col = "black", type = "l", ylim = c(-70, 70))

cols <- c("red", "orange", "yellow", "green", "blue", "darkblue", "purple")
for (i in 1:7) {
  GRW <- rgrw(1000, 0, 1)
  points(GRW, col = cols[i], type = "l")
}
```



20.3 Long-run behaviour

Most of the properties about long-run behaviour of Markov chains in discrete space continue to hold in continuous space: we just replace the transition probability by the transition density and sums by integrals.

We can find the two-step transition density $p^{(2)}(x, y)$ be integrating over all possible intermediate steps z

$$p^{(2)}(x, y) = \int_{\mathcal{S}} p(x, z)\, p(z, y)\, \mathrm{d}z.$$

Similarly, we get an n-step transition density from

$$p^{(n)}(x, y) = \int_{\mathcal{S}^{n-1}} p(x, z_1)\, p(z_1, z_2) \cdots p(z_{n-2}, z_{n-1})\, p(z_{n-1}, y)\, \mathrm{d}z_1\, \mathrm{d}z_2 \cdots \mathrm{d}z_{n-1}.$$

A stationary density π is a PDF on $\mathcal{S}$ (so $\pi(x) \geq 0$ and $\int_{\mathcal{S}} \pi(x)\,\mathrm{d}x = 0$) such that

$$\pi(y) = \int_{\mathcal{S}} \pi(x)\, p(x, y)\,\mathrm{d}x \qquad \text{for all } y \in \mathcal{S}.$$

This is often easier to find by solving the detailed balance equations

$$\pi(y)\, p(y, x) = \pi(x)\, p(x, y) \qquad \text{for all } x, y \in \mathcal{S}.$$

[Limit theorem]

Next time. *We will study the Metropolis–Hastings algorithm in continuous space.*

Summary:

- Markov chains also exist for continuous state space, where they are defined by the transition density $p(x, y)$.
- Properties are similar to Markov chains in discrete space: a stationary distribution can be found by solving the discrete balance equations $\pi(x)p(x, y) = \pi(y)p(y, x)$; and the ergodic theorem says that $\sum 1n \sum_{i=1}^{n} \phi(X_i) = \mathbb{E}\,\phi(X)$ where X has PDF π.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection 2.3.1.

21 Metropolis–Hastings in continuous space

21.1 The Metropolis–Hastings algorithm again

Last time, we looked at Markov chains (X_i) in continuous space, as defined by a transition density $p(x, y)$. We saw (under certain technical conditions we didn't get into) that we have convergence to a unique stationary probability density π. We further saw that we have an ergodic theorem

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

where X has PDF π.

As before, this opens up to us the possibility of MCMC: find a Markov chain with stationary distribution equal to the PDF you want to sample from, then use the output of the Markov chain as the samples in a Monte Carlo estimator. And, yet again, the Metropolis–Hastings algorithm gives us a way to find such a Markov chain. The continuous Metropolis–Hastings algorithm is essentially the same as the discrete time one, but with densities instead of probabilities.

- We want to define a Markov chain on a continuous state space $\mathcal{S}$, which contains the range of the probability density function π we want to sample from.
- We have an initial state $X_1 = x_1$ and we choose a transition density $r = r(x, y)$ representing the proposal moves.

1. From a current state $X_i = x$ we propose moving to a new state y, where y is chosen according to the probability density $r(x, y)$.
2. With probability
$$\alpha(x, y) = \min\left\{\frac{\pi(y)\,r(y, x)}{\pi(x)\,r(x, y)},\, 1\right\},$$
we accept the proposal, and set $X_i = y$; otherwise we stay put, and set $X_{i+1} = x$.
3. We repeat steps 1. and 2. n times to get n samples.

(Note that the acceptance probability $\alpha(x, y)$ really is a probability, not a density.)

This can be proved to have π as a stationary density by checking the detailed balance equations. The proof is identical to the discrete case, so we won't write it out again.

21.2 Random walk Metropolis again

As with the discrete case, when the transition density is symmetric, in that $r(y, x) = r(x, y)$ for all $x, y \in \mathcal{S}$, then the acceptance probability simplifies to

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\},$$

and we call it just the Metropolis algorithm.

When the proposal density is that of a Gaussian random walk with drift $\mu = 0$, we call this Random walk Metropolis.

Example 21.1. Suppose we wish to sample from an exponential distribution $X \sim \text{Exp}(\lambda)$, which has PDF $\pi(x) = \lambda e^{-\lambda x}$ for $x \geq 0$.

We can sample from this using the random walk Metropolis algorithm. From $X_i = x$, we propose a move to $y = x + \text{N}(0, \sigma^2) = \text{N}(x, \sigma^2)$. If $y \geq 0$, we accept the proposed move with probability

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\} = \min\left\{\frac{\lambda e^{-\lambda y}}{\lambda e^{-\lambda x}}, 1\right\} = \min\left\{e^{\lambda(x-y)}, 1\right\}.$$

So if $0 \leq y \leq x$, then we always accept the move with probability 1, while if $y > x$ then we accept with probability $\alpha(x, y) = e^{-\lambda(y-x)}$. If $y < 0$, then $\pi(y) = 0$, so $\alpha(x, y) = 0$, and we always reject.

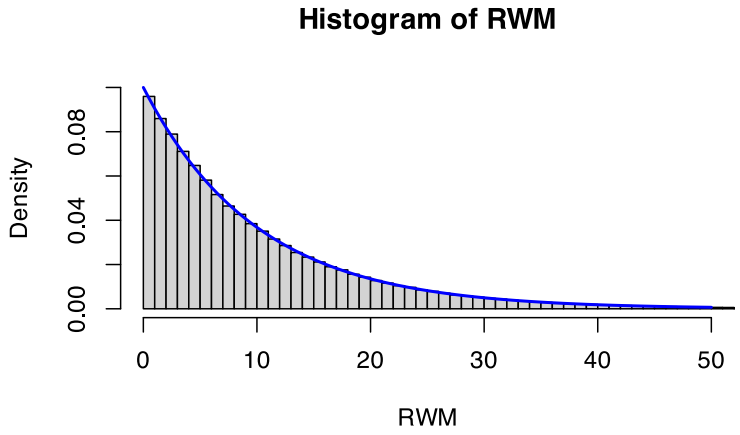
The following R function carries this out.

```
metroexp <- function(n, rate, sigma, initial) {
  MC <- rep(0, n)
  accept <- function(x, y) exp(-rate * (y - x))

  MC[1] <- initial
  for (i in 1:(n - 1)) {
    prop <- MC[i] + rnorm(1, 0, sigma)
    if (prop < 0)  MC[i + 1] <- MC[i]
    else if (runif(1) <= accept(MC[i], prop)) MC[i + 1] <- prop
    else MC[i + 1] <- MC[i]
  }

  return(MC)
}
```

We used a cunning trick to slightly simplify the above code. If $\pi(y)/\pi(x) > 1$, then the acceptance probability is 1, from the "min" in the definition of $\alpha(x, y)$. But the implementation to accept if $U \leq \pi(y)/\pi(x)$ still works. This makes the `else if` line in the above code a bit simpler, as we didn't have to deal with this case separately.

Let's try it out for $\lambda = 0.1$ and $\sigma = 15$.

```
RWM <- metroexp(1e6, 0.1, 15, 0)

hist(RWM, probability = TRUE, xlim = c(0, 50), ylim = c(0, 0.1), breaks = 100)
curve(dexp(x, 0.1), add = TRUE, lwd = 2, col = "blue")
```

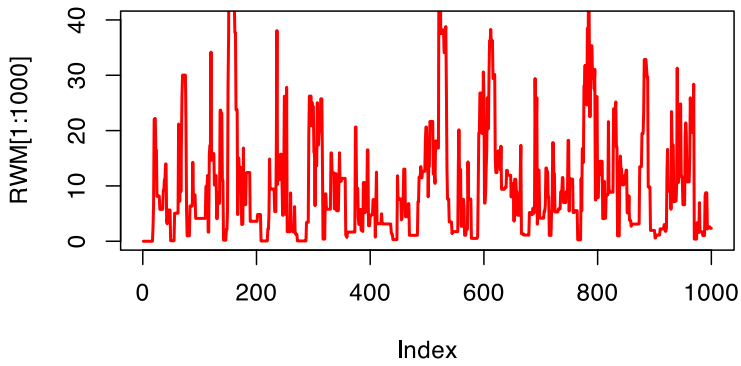


This looks like an excellent match to the Exp(0.1) distribution.

In the random walk Metropolis algorithm, we had to pick the value for the parameter σ. In this algorithm, σ can be interpreted as a "typical step size", in that the standard deviation of the step proposal $y - x$ is σ. The ergodic theorem will still hold for any value of σ in the limit as $n \to \infty$, but the practical performance at finite n may be different for different values.

Let's have a look at how the Markov chain moved with our step size of $\sigma = 15$

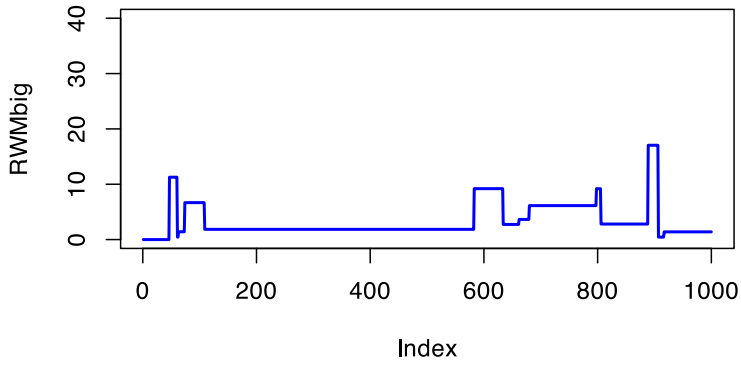
```
plot(RWM[1:1000], lwd = 2, ylim = c(0, 40), col = "red", type = "l")
```



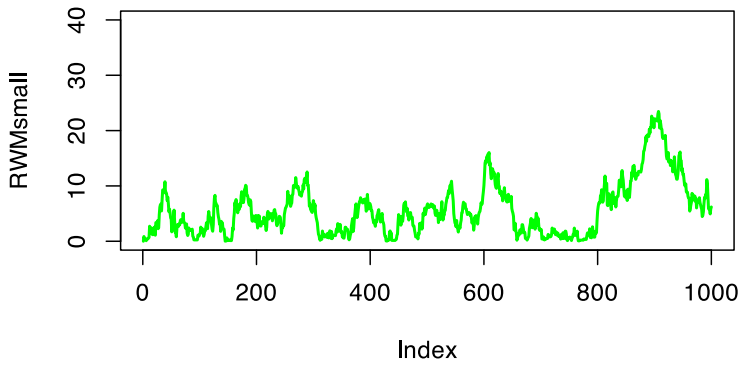
It's clear this isn't an independent sample, since this is not purely "exponentially distributed noise". But it seems to exploring the range of different values an Exp(0.1) distribution is likely to take very rapidly.

What if we had chosen a much larger step size, like $\sigma = 400$, or a much smaller one, like $\sigma = 1$?

```
RWMbig <- metroexp(1000, 0.1, 400, 0)
plot(RWMbig, lwd = 2, ylim = c(0, 40), col = "blue", type = "l")
```




```
RWMsmall <- metroexp(1000, 0.1, 1, 0)
plot(RWMsmall, lwd = 2, ylim = c(0, 40), col = "green", type = "l")
```



We can see that when $\sigma = 400$ (blue), there are lots of "flat parts" of the graph. This is where the Markov chain did not move, because it was rejecting lots of proposed moves. This would be because large steps would often produce either negative proposals, which are always rejected, or very large proposals, where the acceptance probability is very small. This Markov chain is rarely moving at all, so is not exploring the state space very well.

We can also see that when $\sigma = 1$ (green), the Markov chain was often accepting moves, but only making small steps. Compared to the $\sigma = 15$ case, this graph is much less "busy", and looks more like a gentle wander through the state space rather than a rapid exploration. So although this Markov chain is moving through the state space, it is crawling through space quite slowly.

We saw here a general pattern when choosing the step size in a Gaussian random walk Metropolis algorithm:

- If the step size is too large, then too many proposals are rejected. This means you often stay in the same state for a long time, and you only rarely move to explore a new state.
- If the step size is too small, then proposals are very close to the current state. This means you often stay in the same approximate area for a long time, and you crawl through the state space very slowly.

You want to try and pick the "Goldilocks" step size – not too small, and not too big! There is no perfect recipe you can follow to pick the ideal step size – MCMC is an art as well as a science. If you are able to, it can be helpful to think about what typical values you hope to be sampling. In our example, ranges of between 0 and 30 or so are typical for Exp(0.1), so you want a step size that will explore such a range well without too regularly stepping outside of it. Our choice of $\sigma = 15$, being half of that $[0, 30]$ range seemed to do quite well.

Some people consider a rule of thumb where the acceptance rate should be around 40–50% for one-dimensional (or low-dimensional) problems to be a good rule of thumb, decreasing towards 20–25% for higher dimensional problems; but this is only a guide, not a strict rule.
If one changes the code in the example above to keep track of the acceptance rate, one would see that with $\sigma = 1$, the acceptance rate is usually greater than 90%; with $\sigma = 400$. the acceptance rate is usually less than 5%; and with $\sigma = 1$, the acceptance rate is usually around 40 to 45%. This matches the rule of thumb.

It's also worth doing short "pilot" runs, where you try different values of σ and examine what seems to work best.

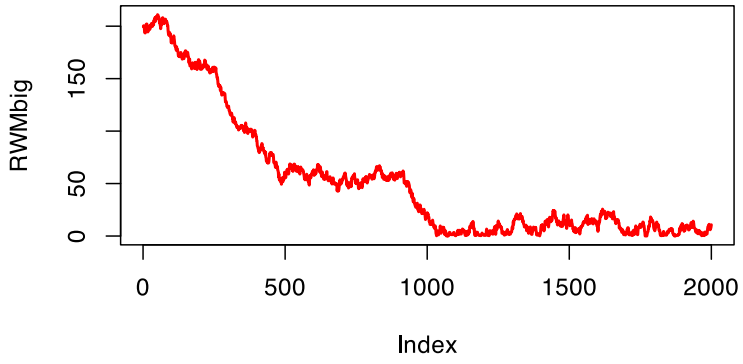
21.3 Burn-in period

Recall that the goal of MCMC is to sample from the stationary distribution π. It's therefore a good idea, if it's possible, to start from an initial value that's "typical" of π; that is, has a large value of π and is near other states with large values of π. That was definitely true for $X_1 = 0$ in our previous example.

But what happens if we don't pick such an initial state – either because we don't know what these typical states look like, or by mistake.

Example 21.2. We continue with the previous example, with step size $\sigma = 2$. What if we had started at $X_1 = 200$ instead – what then would the Markov chain look like.

```
set.seed(4)
RWMbig <- metroexp(2000, 0.1, 2, 200)
plot(RWMbig, lwd = 2, col = "red", type = "l")
```



You can see that, at the start, it takes us a while to move away from the "bad" initial state $X_1 = 200$ and to get to the "typical values" of $[0, 35]$ or so. In this example, it took around 1000 steps.

The ergodic theorem tells us this, eventually, for large enough n, these early unrepresentative samples will be drowned out of our large-n collection. But for modest finite values of n, these are likely to corrupt our Monte Carlo estimation procedure.

For that reason, it can often be useful to use a **burn-in period**. A burn-in period is when you run the Markov chain for a while without using the samples in estimation, and only start using samples once you have "reached the stationary distribution" – the phrase "reached equilibrium" is also sometimes used. Again, how long a burn-in period should be is art more than science – thinking about your specific problem and conducting experiments can help you decide if a burn-in period is necessary and how long it should be. If the state space and stationary distribution are easy to understand – say, $\mathcal{S}$ is one-dimensional and π has a single mode – you can even run the MCMC algorithm first, then afterwards decide which unrepresentative early samples to throw away.

Next time. *We analyses the error of Monte Carlo estimation with the output of a Markov chain.*

Summary:

- The Metropolis–Hastings algorithm in continuous space works essentially the same as in discrete space: propose a move according to the transition density $r(x, y)$, and accept it with probability
$$\alpha(x, y) = \min\left\{\frac{\pi(y)\, r(y, x)}{\pi(x)\, r(x, y)}, 1\right\}.$$
- When the proposal distribution is that of the symmetric Gaussian random walk, this is the random walk Metropolis algorithm.
- Choosing the step size σ in the Metropolis random walk is important: too small and the Markov chain takes only very small steps; too large and most proposals are rejected.
- A burn-in period can be used to find a good initial point for MCMC.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection ?.?.?.

Problem Sheet 4

Full solutions are now available.

This is Problem Sheet 4, which covers material from Lectures 17 to 21. You should work through all the questions on this problem sheet in advance of the problems class, which takes place in the lecture of **Thursday 27 November**.

This problem sheet is to help you practice material from the module and to help you check your learning. It is *not* for formal assessment and does not count towards your module mark.

If you want some brief informal feedback on **Questions 1 and 3(a) and (b)** (marked), you should submit your work electronically through Gradescope via the module's Minerva page by **1400 on Tuesday 25 November**. (If you hand-write solutions on paper, the easiest way to scan-and-submit that work is using the Gradescope app on your phone.) I will return some brief comments on your those two questions by the problems class on Thursday 27 November. Because this informal feedback, and not part of the official assessment, I cannot accept late work for any reason – but I am always happy to discuss any of your work on any question in my office hours.

Full solutions will be released on Friday 28 November.

22 MCMC error

22.1 Bias for MCMC

Back in Lectures 3 and 4 we looked at the bias, variance, mean-square error and root-mean-square error for the Monte Carlo estimator

$$\widehat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} \phi(X_i)$$

of $\theta = \mathbb{E}\,\phi(X)$ where the samples X_i are IID with the same distribution as X. We saw that the estimator is unbiased, and that the variance and mean-square error are

$$\mathrm{Var}\,(\widehat{\theta}_n^{\mathrm{MC}}) = \mathrm{MSE}\,(\widehat{\theta}_n^{\mathrm{MC}}) = \frac{1}{n}\,\mathrm{Var}\,(\phi(X)).$$

We now want to find these same values where the samples X_i are not IID but are the output of a Markov chain whose stationary distribution is that of X. This will be harder. We saw (under certain technical conditions that we will assume hold throughout) that the X_i tend to the distribution of X in the limit as $i \to \infty$. But this is not the same as saying that their distribution is exactly the same as X (let alone are independent). So here we can get as far as

$$\mathbb{E}\,\widehat{\theta}_n^{\mathrm{MC}} = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} \phi(X_i)\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\,\phi(X_i)$$

(remembering that linearity of expectation does not require independence), but then we're a bit stuck.

To make progress, we will make a simplifying assumption. Suppose we picked the initial state X_1 according to the distribution π. Then, since π is a stationary distribution, X_2 is distributed according to π too. And X_3, and X_4, and so on. And π itself is the distribution for X. So, if we started from the stationary distribution – or "in equilibrium" – then we have

$$\mathbb{E}\,\widehat{\theta}_n^{\mathrm{MC}} = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} \phi(X_i)\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\,\phi(X_i) = \frac{1}{n}\,n\,\mathbb{E}\,\phi(X) = \mathbb{E}\,\phi(X),$$

and our estimator is unbiased.

Of course, in real life, it is highly unlikely that we are able to sample the initial state from π. After all, if we could do that, we could presumably sample all the X_i from π independently, as just use the basic Monte Carlo estimator instead. However, if we have used a burn-in period of appropriate length, we hope that by the time we take the first sample "that counts", after the burn-in period, that will be very close to the stationary distribution, and hence we will have

$$\mathbb{E}\,\hat{\theta}_n^{\mathrm{MC}} \approx \mathbb{E}\,\phi(X),$$

and our estimator will be approximately unbiased.

22.2 Variance for MCMC

What about the variance of the MCMC estimator. Unlike the basic IID Monte Carlo case, we can no longer say

$$\mathrm{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n}\phi(X_i)\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}\left(\phi(X_i)\right),$$

because the samples from a Markov chain are not independent (although we have limited their dependence structure).

Instead, we have to include the "cross" covariance terms:

$$\mathrm{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n}\phi(X_i)\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\mathrm{Var}\left(\phi(X_i)\right) + 2\sum_{i<j}\mathrm{Cov}\left(\phi(X_i),\phi(X_j)\right)\right).$$

Again, we grind to a halt as far as exact results are concerned. But we can again invoke our simplifying assumption that X_1 was chosen from π, and that we are in equilibrium. Then the variance terms are all $\mathrm{Var}(\phi(X_i)) = \mathrm{Var}(\phi(X))$, which I shall call σ^2. What about the covariance terms? Well, if the Markov chain is stationary, then $\mathrm{Cov}(\phi(X_i),\phi(X_j))$ only depends on how many steps apart i and j are. In equilibrium, $\mathrm{Cov}(\phi(X_1),\phi(X_7))$ is the same as $\mathrm{Cov}(\phi(X_2),\phi(X_8))$ or $\mathrm{Cov}(\phi(X_{101}),\phi(X_{107}))$: these all represent the covariance between one state chosen according to π and the state $k = 7 - 1 = 6$ steps later.

So here we can write

$$\mathrm{Cov}\left(\phi(X_i),\phi(X_j)\right) = \gamma(j-i) = \gamma(k)$$

where $k = j - i$ is the number of steps between i and j. Students who have studied time series will know that $\gamma(k)$ is called the **autocovariance** at **lag** k. (The prefix "auto-" mean "self-", and "lag" means something like a delay.)

(The tempting hope that we might have $\gamma(k) = 0$ for $k \geq 2$ is not true. [EXPLAIN])

So now, in equilibrium, we have

$$\operatorname{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\operatorname{Var}\left(\phi(X_i)\right) + 2\sum_{i<j}\operatorname{Cov}\left(\phi(X_i),\phi(X_j)\right)\right) \tag{22.1}$$

$$= \frac{1}{n^2}\left(\sum_{i=1}^{n}\sigma^2 + 2\sum_{i<j}\gamma(j-i)\right) \tag{22.2}$$

$$= \frac{1}{n^2}\left(n\sigma^2 + 2\sum_{i<j}\gamma(j-i)\right) \tag{22.3}$$

When we studied antithetic variables, we found it more convenient to work with the correlation

$$\operatorname{Corr}\left(\phi(X_i),\phi(X_j)\right) = \frac{\operatorname{Cov}\left(\phi(X_i),\phi(X_j)\right)}{\sqrt{\operatorname{Var}\left(\phi(X_i)\right)\operatorname{Var}\left(\phi(X_i)\right)}}.$$

In equilibrium, this is

$$\rho(j-i) = \operatorname{Corr}\left(\phi(X_i),\phi(X_j)\right) = \frac{\gamma(j-i)}{\sqrt{\sigma^2\,\sigma^2}} = \frac{\gamma(j-i)}{\sigma^2},$$

where $\rho(k)$ is called the **autocorrelation** at lag k.

So now we have

$$\operatorname{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \frac{1}{n^2}\left(n\sigma^2 + 2\sum_{i<j}\gamma(j-i)\right) \tag{22.4}$$

$$= \frac{1}{n}\,\sigma^2 + \frac{2}{n^2}\sum_{i<j}\rho(j-i)\,\sigma^2 \tag{22.5}$$

$$= \frac{\sigma^2}{n}\left(1 + \frac{2}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\rho(j-i)\right) \tag{22.6}$$

$$= \frac{\sigma^2}{n}\left(1 + \frac{2}{n}\sum_{i=1}^{n}\sum_{k=1}^{n-i}\rho(k)\right). \tag{22.7}$$

We know from the limit theorem that, as $n \to \infty$, a Markov chain tends to its stationary distribution regardless of what state it started from – it's as if the Markov chain forgets where it starts from. Therefore we know that the autocorrelation $\rho(k)$ tends to 0 as $k \to \infty$. So, provided that the number of samples n is large, there usually very little loss from approximating $\sum_{k=1}^{n-i}\rho(k)$ by $\sum_{k=1}^{\infty}\rho(k)$, because the extra autocorrelations we've added in will all be very small.

Finally, we get the result

$$\operatorname{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) = \frac{\sigma^2}{n}\left(1 + \frac{2}{n}\sum_{i=1}^{n}\sum_{k=1}^{n-i}\rho(k)\right) \tag{22.8}$$

$$\approx \frac{\sigma^2}{n}\left(1 + \frac{2}{n}\sum_{i=1}^{n}\sum_{k=1}^{\infty}\rho(k)\right) \tag{22.9}$$

$$= \frac{\sigma^2}{n}\left(1 + \frac{2}{n}\,n\sum_{k=1}^{\infty}\rho(k)\right) \tag{22.10}$$

$$= \frac{\sigma^2}{n}\left(1 + 2\sum_{k=1}^{\infty}\rho(k)\right). \tag{22.11}$$

In conclusion, we have the following.

Theorem 22.1. *Let (X_i) be a Markov chain started in its stationary distribution π, and let X have distribution π also. Consider $\theta = \mathbb{E}\,\phi(X)$ and its MCMC estimator*

$$\hat{\theta}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n}\phi(X_i).$$

Then, writing $\sigma^2 = \operatorname{Var}(\phi(X))$, we have the following:

- *$\hat{\theta}_n^{\mathrm{MC}}$ is unbiased, in that $\mathbb{E}\,\hat{\theta}_n^{\mathrm{MC}} = \theta$.*
- *The variance of $\hat{\theta}_n^{\mathrm{MC}}$ is approximately*

$$\operatorname{Var}\left(\hat{\theta}_n^{\mathrm{MC}}\right) \approx \frac{\sigma^2}{n}\left(1 + 2\sum_{k=1}^{\infty}\rho(k)\right).$$

- *The mean-square error of $\hat{\theta}_n^{\mathrm{MC}}$ is approximately*

$$\operatorname{MSE}\left(\hat{\theta}_n^{\mathrm{MC}}\right) \approx \frac{\sigma^2}{n}\left(1 + 2\sum_{k=1}^{\infty}\rho(k)\right).$$

- *The root-mean-square error of $\hat{\theta}_n^{\mathrm{MC}}$ is approximately*

$$\operatorname{RMSE}\left(\hat{\theta}_n^{\mathrm{MC}}\right) \approx \frac{\sigma}{\sqrt{n}}\sqrt{1 + 2\sum_{k=1}^{\infty}\rho(k)}.$$

Compared to the standard Monte Carlo variance $\text{Var}(\hat{\theta}_n^{\text{MC}}) = \sigma^2/n$ we have the extra term $2\sum_{k=1}^{\infty} \rho(k)$. While it would be nice for the autocorrelation to be negative, to give us an improvement (like with antithetical variables), this almost never happens with Markov chains, which almost always have positive autocorrelation. Instead, we get the best results when the autocorrelation $\rho(k)$ dies away to 0 as quickly as possible, and get poor results when the autocorrelation only decays to 0 very slowly.

22.3 Example

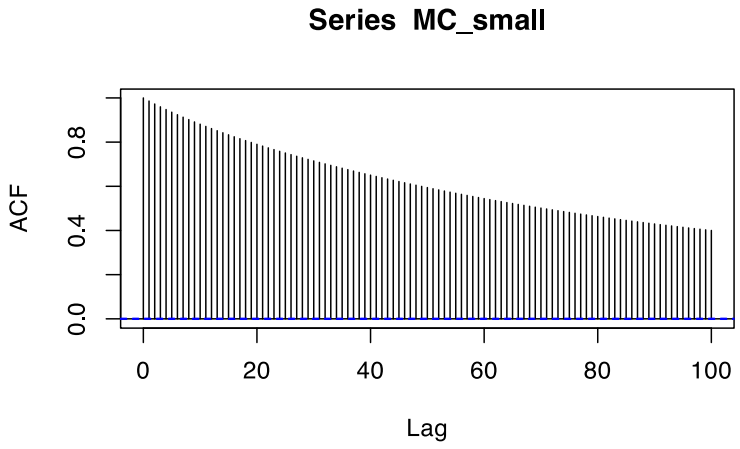
Last time, we used the random walk Metropolis in continuous space to sample from the Exp(0.1) distribution. Let's try some different typical step sizes $\sigma = 2, 15, 400$. Let's use these Markov chains to estimate $\mathbb{E}X$ (which we know is 10), and examine the error in these estimators.

The function we used last time was this.

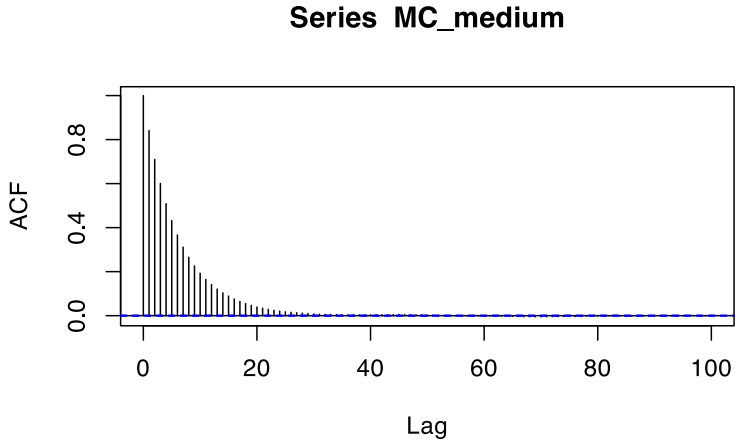
```
metroexp <- function(n, rate, sigma, initial) {
  MC <- rep(0, n)
  accept <- function(x, y) exp(-rate * (y - x))

  MC[1] <- initial
  for (i in 1:(n - 1)) {
    prop <- MC[i] + rnorm(1, 0, sigma)
    if (prop < 0)  MC[i + 1] <- MC[i]
    else if (runif(1) <= accept(MC[i], prop)) MC[i + 1] <- prop
    else MC[i + 1] <- MC[i]
  }

  return(MC)
}
```

We can plot the autocorrelation $\rho(k)$ against the lag k in R using the `acf()` function.

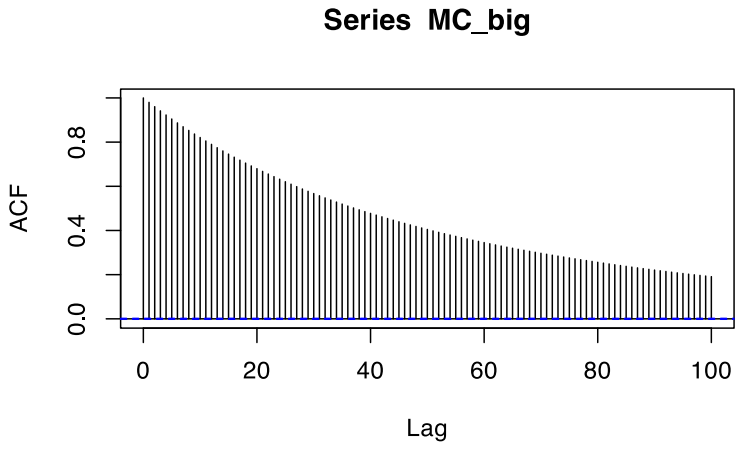
```
MC_small <- metroexp(1e6, 0.1, 2, 0)
acf(MC_small, lag.max = 100)
```




```
MC_medium <- metroexp(1e6, 0.1, 15, 0)
acf(MC_medium, lag.max = 100)
```




```
MC_big <- metroexp(1e6, 0.1, 400, 0)
acf(MC_big, lag.max = 100)
```



Remember that our goal is for the autocorrelation to die away as quickly as possible. We see that the large step size and small step size have the autocorrelation decaying slowly, while the medium step size has the autocorrelation decaying much quicker. This matched with our earlier discussion. When σ is too small, the Markov chain crawls around the state space too slowly; when σ is too large, large proposal moves are usually rejected, leading the Markov chain to stay in place. We need σ in the "Goldilocks" position where σ is small enough that the moves proposed are not too outrageous, but σ is big enough that we make good progress around the state space.

This is confirmed by looking at the relevant term $1 + 2\sum_{k=1}^{\infty} \rho(k)$ from the variance of the MCMC estimator. Well, let's just look at the sum of the first 1000 terms. We have

```
sum_small  <- 1 + 2 * sum(acf(MC_small,  lag.max = 1000, plot = FALSE)$acf)
sum_medium <- 1 + 2 * sum(acf(MC_medium, lag.max = 1000, plot = FALSE)$acf)
sum_big    <- 1 + 2 * sum(acf(MC_big,    lag.max = 1000, plot = FALSE)$acf)
round(c(sum_small, sum_medium, sum_big), 1)
```

```
[1] 269.0  14.2 112.3
```

With the small and big step sizes, we need to take over 100 samples from the Markov chain to get the equivalent of one independent sample. But with the medium step size, we are getting the equivalent of one independent sample from roughly every 15 samples of the Markov chain.

Summary:

- We can analyse the error in Monte Carlo estimation with the output of a Markov chain by assuming it begins "in equilibrium".

- MCMC is approximately unbiased and with MSE $\approx \frac{\sigma^2}{n}\left(1+2\sum_{k=1}^{\infty}\rho(k)\right)$.
- We want the autocorrelation $\rho(k)$ to decay as quickly as possible.

Read more: Voss, *An Introduction to Statistical Computing*, Subsection ?.?.?.

23 MCMC and Bayesian statistics

23.1 Bayesian set-up

Today, we will complete our study of Markov chain Monte Carlo by taking a look at how MCMC can be applied to Bayesian statistics.

We recall the Bayesian set-up. The data is modelled by the **likelihood** $f(\mathbf{x} \mid \theta)$, which is the distribution (probability mass or density function) of the data $\mathbf{x} = (x_1, x_2, \dots, x_n)$ given some unknown parameter θ. Typically, the datapoints x_i are assumed to be IID, so we can write

$$f(\mathbf{x} \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta).$$

Our goal in statistics is to make inferences about – that is, to learn about – the unknown parameter θ.

In the Bayesian paradigm, we start with a **prior** belief about what we think the likely values of θ are before we've collected any data. Our prior belief is represented by a distribution $\pi(\theta)$.

After collecting the data $\mathbf{x}$, we then seek the **posterior** distribution – that is, the distribution of θ *conditional on* the data we have seen. This is represented by a distribution $\pi(\theta \mid \mathbf{x})$. It is this posterior distribution that we want to learn about.

We can calculate the posterior from the prior and the likelihood using Bayes' theorem. We have

$$\pi(\theta \mid \mathbf{x}) \propto \pi(\theta)\, f(\mathbf{x} \mid \theta) = \pi(\theta) \prod_{i=1}^{n} f(x_i \mid \theta).$$

More informally, we can say

$$\text{posterior} \propto \text{prior} \times \text{likelihood}.$$

Let us note for future reference that this only gives the posterior

$$\pi(\theta \mid \mathbf{x}) = \frac{1}{Z}\, \pi(\theta)\, f(\mathbf{x} \mid \theta) = \frac{1}{Z}\, \pi(\theta) \prod_{i=1}^{n} f(x_i \mid \theta)$$

up to proportionality, where the normalising constant is

$$Z = \int \pi(\theta)\, f(\mathbf{x} \mid \theta)\, \mathrm{d}\theta.$$

If you have seen Bayesian statistics in previous modules, you may well have seen simple cases, where the posterior turns out to have a simple form – often in the same family as the prior but with a different parameter.

For example, take the case where the likelihood $X \sim \mathrm{Bin}(m, \theta)$ is a binomial distribution with a known number of trials m but an unknown success probability. If the prior for θ is a Beta distribution, then it's easy to check that the posterior for θ is another Beta distribution, just with different parameters.

But aside from these simple "toy examples", in real life the posterior distribution often has a complicated form. It is often not possible to calculate the normalising constant Z; nor, therefore, can we calculate statistics of the posterior, like its expectation or confidence intervals.

This is where statistical computing comes in. If we can manage to sample from the posterior distribution, then we can estimate statistics of the posterior using Monte Carlo estimation. But (again, aside from the easy toy examples), sampling from the posterior seems hard. The inverse transform method requires the normalisation constant Z, but that is typically to difficult to calculate. Occasionally, if we're very clever, we can think for a long time and manage to come up with an envelope rejection sampling method. But we'd like a method we know will always work.

MCMC is that method. If we use the random walk Metropolis algorithm, all we need to do is pick our step size σ and set it going.

A crucial point that we haven't mentioned yet, is that Metropolis–Hastings works perfectly fine when we don't know a normalising constant in the distribution we are sampling from. That's because the acceptance probability – let's just write it for the symmetric Metropolis case –

$$\alpha(\theta, \theta') = \min\left\{ \frac{\pi(\theta' \mid \mathbf{x})}{\pi(\theta \mid \mathbf{x})}, 1 \right\} \tag{23.1}$$

$$= \min\left\{ \frac{\frac{1}{Z}\, \pi(\theta')\, f(\mathbf{x} \mid \theta')}{\frac{1}{Z}\, \pi(\theta')\, f(\mathbf{x} \mid \theta)}, 1 \right\} \tag{23.2}$$

$$= \min\left\{ \frac{\pi(\theta')}{\pi(\theta)}\, \frac{f(\mathbf{x} \mid \theta')}{f(\mathbf{x} \mid \theta)}, 1 \right\}, \tag{23.3}$$

where the Zs in the fraction have cancelled out.

23.2 Example

Example 23.1. Consider a statistical model where X_i are IID and normally distributed with unknown mean μ and and standard deviation σ. Here our parameter $\theta = (\mu, \sigma)$ is two-dimensional. So the likelihood is

$$f(\mathbf{x} \mid \mu, \sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2\right).$$

There are some special priors for which the posterior here has a very simple distribution. For example, if the prior for μ is a normal distribution and the prior for σ is that $1/\sigma^2$ has a Gamma distribution and these are independent, then the posterior has the same form but with different parameters. But if your beliefs about μ and σ aren't well represented by a distribution of this form, then it will be very difficult or impossible to get a closed-form expression for the posterior.

Let's suppose our prior beliefs are represented something a bit weirder. Let's say that our prior is that μ is uniformly distributed on $[-10, 10]$ and that σ is exponentially distributed with rate 0.3 and that these are independent. So our prior is

$$\pi(\mu, \sigma) = \frac{1}{20} 0.3\mathrm{e}^{-0.3\sigma} \qquad -10 \le \mu \le 10, \sigma \ge 0.$$

Suppose now that we see that data

$$\mathbf{x} = (3.5, 13.7, 6.4, 0.7, 10.2, -2.3, 6.6, 5.1, 9.1, 10.7).$$

We want to update our beliefs about μ and σ and make inferences about the posterior. Finding the posterior here "by hand" seems a hopeless task. But we can certainly simulate from it with the random walk Metropolis algorithm.

```
prior <- function(mu, sigma) dunif(mu, -10, 10) * dexp(sigma, 0.3)
accept <- function(mu, sigma, propmu, propsigma, x) {
  if (propsigma < 0) return(0) else {
    priorratio <- prior(propmu, propsigma) / prior(mu, sigma)
    lhr <- dnorm(x, propmu, propsigma) / dnorm(x, mu, sigma)
    return(priorratio * prod(lhr))
  }
}

x <- c(3.5, 13.7, 6.4, 0.7, 10.2, -2.3, 6.6, 5.1, 9.1, 10.7)

n <- 1e6
```

```
stepmu    <- 4
stepsigma <- 2
initialmu    <- 0
initialsigma <- 3

MCmu    <- rep(0, n)
MCsigma <- rep(0, n)
MCmu[1]    <- initialmu
MCsigma[1] <- initialsigma

for (i in 1:(n - 1)) {
  propmu    <- MCmu[i]    + rnorm(1, 0, stepmu)
  propsigma <- MCsigma[i] + rnorm(1, 0, stepsigma)
  if (runif(1) < accept(MCmu[i], MCsigma[i], propmu, propsigma, x)) {
    MCmu[i + 1]    <- propmu
    MCsigma[i + 1] <- propsigma
  } else {
    MCmu[i + 1]    <- MCmu[i]
    MCsigma[i + 1] <- MCsigma[i]
  }
}
```

If we just want "point estimators" for μ and σ, we could just take the means of the posteriors distributions. We can calculate those using MCMC; they're simply

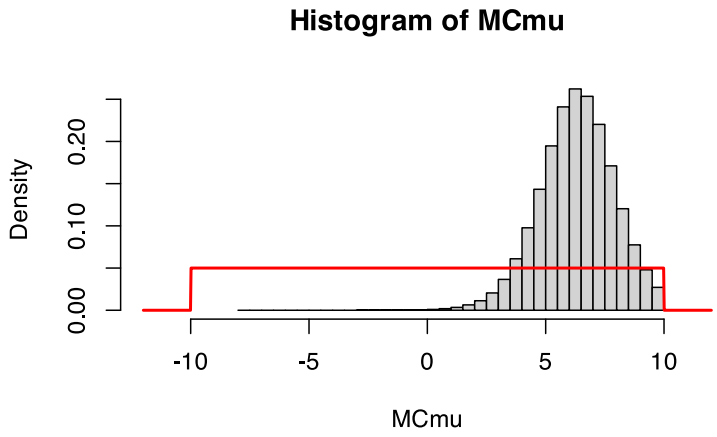
```
c(mean(MCmu), mean(MCsigma))
```

```
[1] 6.29991 5.06038
```

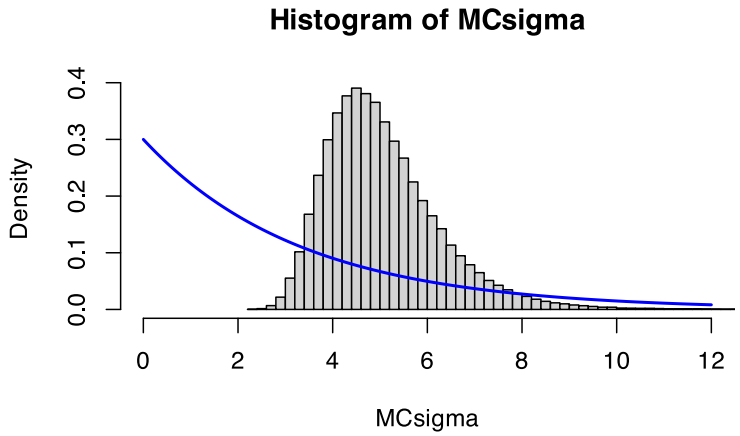
Compare these to the posterior means of 0 and $1/0.3 = 3.33$ respectively.

But the strength of Bayesian statistics is not just computing point estimates, but understanding the entire posterior distribution. We can compare our histograms of the marginal distributions of the posterior with the marginal distributions of the prior.

```
hist(MCmu, probability = TRUE, xlim = c(-12, 12), breaks = 50)
curve(dunif(x, -10, 10), add = TRUE, n = 1001, lwd = 2, col = "red")
```




```
hist(MCsigma, probability = TRUE, breaks = 80, xlim = c(0,12))
curve(dexp(x, 0.3), add = TRUE, n = 1001, lwd = 2, col = "blue")
```



Although in our prior the parameters μ and σ were independent, that might not be the case any longer in our posterior.

```
cor(MCmu, MCsigma)
```

```
[1] -0.06296037
```

It appears there's now a slight negative correlation (if μ is smaller than we expect, σ has to be a little bit bigger to allow the data to fit).

23.3 Numerical stability

In our example, we had only 10 data points. But when you have a large number of data points n (as is often the case in modern "big data" applications), then $f(\mathbf{x} \mid \theta) = \prod_{i=1}^{n} f(\mathbf{x} \mid \theta)$ can be extremely small. This can prove difficult when calculating the acceptance probability, because both the numerator and denominator in

$$\frac{\pi(\theta')\, f(\mathbf{x} \mid \theta')}{\pi(\theta)\, f(\mathbf{x} \mid \theta)} = \frac{\pi(\theta') \prod_{i=1}^{n} f(x_i \mid \theta')}{\pi(\theta) \prod_{i=1}^{n} f(x_i \mid \theta)}$$

can be extremely small. Dividing one very small number by another can be "numerically unstable", where small rounding errors can lead to big errors.

It's usually a good idea to split the fraction into multiple fractions each of which has a more sensible size; that is

$$\frac{\pi(\theta')}{\pi(\theta)} \times \frac{f(\mathbf{x} \mid \theta')}{f(\mathbf{x} \mid \theta)} = \frac{\pi(\theta')}{\pi(\theta)} \times \prod_{i=1}^{n} \frac{f(x_i \mid \theta')}{f(x_i \mid \theta)}.$$

Even more reliable can be to work on a log scale. We know that $a \times b$ can also be written as $\exp(\log a + \log b)$. So it can be even better to write the fraction above as

$$\begin{aligned} &\exp\big(\log \pi(\theta') - \log \pi(\theta) + \log f(\mathbf{x} \mid \theta') - \log f(\mathbf{x} \mid \theta)\big) \\ &\qquad = \exp\left(\log \pi(\theta') - \log \pi(\theta) + \sum_{i=1}^{n} \log f(x_i \mid \theta') - \sum_{i=1}^{n} \log f(x_i \mid \theta)\right). \end{aligned} \tag{23.4}$$

23.4 MCMC conclusions

In the second part of this module, we saw various ways to get exact IID samples from the exact distribution we want (in-built R functions, inverse transform method, Box–Muller transform, basic rejection sampling, envelope rejection sampling). In this third part of the module, we've only really seen one way in detail – the random walk Metropolis algorithm – and it produces non-independent samples from approximately the distribution we want. That description makes MCMC sound much worse!

But MCMC using the random walk Metropolis algorithm has big advantages too.

Summary:

-

Read more: Voss, *An Introduction to Statistical Computing*, Subsection ?.?.?.

Part IV

Resampling methods

24 Empirical distribution

24.1 Introduction

So far in this module, we have looked at sampling from distributions where we know the precise probability density function f or probability mass function p. We have then been able to discover things about that distribution by sampling from f or p: either exactly with independent samples (with the inverse transform method or envelope rejection sampling, for example), or approximately with samples of restricted dependence (with the Metropolis–Hastings algorithm).

But statisticians deal with data, not with probability distributions. A much more common situation is that we have some data. We believe that the data has come from a distribution, but we don't know what that distribution is. Nonetheless, we still want to find out facts about that unknown distribution.

One traditional way to do this would be to fit a distribution to the data. By using knowledge about the context of the data and by examining the data itself, an appropriate parametric model could be chosen, and then the parameters could be estimated from the data. Once the model has then been specified, we can find out about that model using the ideas we have looked at in this course.

But sometimes this is not possible or desirable. With insufficient contextual knowledge, we might not be able to choose an appropriate parametric model. Or the data might not seem to fit *any* of the famous parametric models. And even if we did choose and fit a model, there's no guarantee that out choice would be correct.

Instead, we could look for the model that makes the fewest possible assumptions about the data. This is called the **empirical distribution**, which we will look at today. This is the basis for a collection of statistical techniques called the **bootstrap** (or **bootstrapping**), which we will look at for the next lecture.

24.2 Definition and properties

Consider the following dataset $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ of $m = 5$ values:

$$2,\ 3,\ 3,\ 4,\ 6.$$

We could try to fit a distribution to this data. For example, if we knew it was recording the number of absences from a class of 20 students over five lectures, we might think a binomial $\text{Bin}(20, p)$ model was appropriate, and then try to estimate the value of p. If it was recording the number of emails received per hour, we might think a Poisson $\text{Po}(\lambda)$ model was appropriate, and try to estimate the value of λ.

But that involves making assumptions. What if we wanted to make no assumptions about the data – or, at least, as few assumptions as possible? Well, we can say that in our data set, one fifth of the data was the value 2, two fifths was 3 (because there were two 3s in the dataset), one fifth was 4, and one fifth was 6. So we choose the model that these come from a random variable X^* where

$$\mathbb{P}(X^* = 2) = \tfrac{1}{5} \qquad \mathbb{P}(X^* = 3) = \tfrac{2}{5} \tag{24.1}$$

$$\mathbb{P}(X^* = 4) = \tfrac{1}{5} \qquad \mathbb{P}(X^* = 6) = \tfrac{1}{5}. \tag{24.2}$$

This is called the empirical distribution. (The word "empirical" refers to what you actually observed, rather than assumed.)

Definition 24.1. Consider a dataset $\mathbf{x} = (x_1, x_2, \dots, x_m)$. The **empirical distribution** X^* of this data is a discrete random variable with probability mass function $p^*(x) = \mathbb{P}(X^* = x)$, where

$$p^*(x) = \frac{1}{m}\left|\{j : x_j = x\}\right| = \frac{1}{m}\sum_{j=1}^{m} \mathbb{I}_{\{x\}}(x_j).$$

In other words, $p^*(x)$ is simply the proportion of the dataset that took the value x.

We're not suggesting that the distribution X^* is necessarily the "true" distribution the data actually came from. Rather, we feel that by reducing any of our own assumptions that we make on the data, we are giving the data the best opportunity to "speak for itself", rather than imposing our own views and opinions on the data.

On that point, consider this dataset of $m = 6$ datapoints:

$$0.580,\ 3.219,\ 3.433,\ 4.913,\ 18.784,\ 28.133.$$

It certainly looks like these came from a continuous distribution. But the empirical distribution X^* is still discrete – it takes each of those values with probability $\frac{1}{6}$. The claim is not the the empirical distribution is likely to be "correct"; rather, the claim is that by minimising the assumptions we make, we aren't polluting the data any further.

Let's think further about this random variable X^*, the empirical distribution of a dataset $\mathbf{x}$.

First, taking a single sample from the random variable X^* is equivalent to picking one of the datapoints at random. The probability we pick a value x is simply the proportion of the datapoints that take the value x.

Taking multiple IID samples from X^* is the same as sampling from the dataset *with replacement* – since, to be *independent* samples, it has to be possible to pick the same datapoint twice.

Second, since X^* is a random variable, we can do calculations with it just as we would any other random variable.

For example, we can calculate its expectation. This is

$$\mathbb{E}_* X^* = \sum_x x\, p^*(x) \tag{24.3}$$

$$= \sum_x x\, \frac{1}{m} \sum_{j=1}^{m} \mathbb{I}_{\{x\}}(x_j) \tag{24.4}$$

$$= \frac{1}{m} \sum_{j=1}^{m} \sum_x x\, \mathbb{I}_{\{x\}}(x_j). \tag{24.5}$$

Now, if we think about the term $x\, \mathbb{I}_{\{x\}}(x_j)$ inside the sum over x here, the indicator will equal 0 for every term in the sum except the term $x = x_j$, when the term will equal $x \times 1 = x_j$. Hence, we have

$$\mathbb{E}_* X^* = \frac{1}{m} \sum_{j=1}^{m} x_j.$$

But this is just the sample mean of the data set. The empirical expectation is the sample mean, $\mathbb{E}_* X^* = \overline{x}$.

You probably noticed the notation $\mathbb{E}_*$ here. We use this notation when we want to emphasise we are taken the expectation over the empirical distribution, taking the data as fixed. This isn't really needed here – we know our data is the fixed observations $\mathbf{x}$. But later on, we will model the data itself as being samples from a random variable. There we will want to distinguish between taking an expectation $\mathbb{E}$ over the randomness in the samples themselves and taking an expectation $\mathbb{E}_*$ over the empirical distribution while treating the samples as fixed.

We can also calculate the variance similarly. Since $\mathbb{E}_* X^* = \overline{x}$, we have $\mathrm{Var}_*(X^*) = \mathbb{E}_*(X^* - \overline{x})^2$. By the same argument as for the expectation, we have

$$\mathrm{Var}_*(X^*) = \sum_x (x - \overline{x})^2\, p^*(x) \tag{24.6}$$

$$= \sum_x (x - \overline{x})^2\, \frac{1}{m} \sum_{j=1}^{m} \mathbb{I}_{\{x\}}(x_j) \tag{24.7}$$

$$= \frac{1}{m} \sum_{j=1}^{m} \sum_x (x - \overline{x})^2\, \mathbb{I}_{\{x\}}(x_j) \tag{24.8}$$

$$= \frac{1}{m} \sum_{j=1}^{m} (x_j - \overline{x})^2. \tag{24.9}$$

This is almost the sample variance – it just has $\frac{1}{m}$ in front instead of the usual $\frac{1}{m-1}$. This won't make much difference when m is large.

Third, its often more mathematically pleasant to work with is the **empirical cumulative distribution function** F^*. This is $F^*(x) = \mathbb{P}(X^* \leq x)$. We can calculate the empirical CDF as

$$F^*(x) = \sum_{y \leq x} p^*(y) = \sum_{y \leq x} \frac{1}{m} \left|\{j : x_j = y\}\right| \tag{24.10}$$

$$= \frac{1}{m} \sum_{y \leq x} \left|\{j : x_j = y\}\right| \tag{24.11}$$

$$= \frac{1}{m} \left|\{j : x_j \leq x\}\right|. \tag{24.12}$$

So this is just the proportion of the datapoints that are less than or equal to x_j. This tends to be more mathematically convenient, because the CDF works equally well for discrete and continuous random variables, so this is flexible to the fact that the empirical distribution is always discrete while the true distribution may be continuous.

24.3 Empirical distributions in R

Example 24.1. The heights (in cm) of 20 student's surveyed in one of Dr Voss's modules is as follows:

```
heights <- c(
  180, 182, 182, 181, 164, 180, 154, 153, 177, 190,
  182, 175, 167, 168, 185, 153, 172, 177, 176, 170
)
m <- length(heights)
```

In R, the `table()` function tells us how many outcomes we had of each value. We can then find the empirical PMF p^* by dividing this by m.

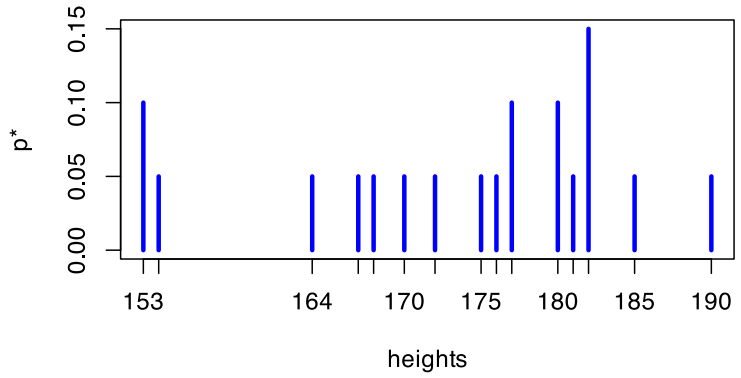
```
table(heights)
```

```
heights
153 154 164 167 168 170 172 175 176 177 180 181 182 185 190
  2   1   1   1   1   1   1   1   1   2   2   1   3   1   1
```

```
table(heights) / m
```

```
heights
  153   154   164   167   168   170   172   175   176   177   180   181   182   185   190
0.10 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.10 0.10 0.05 0.15 0.05 0.05
```

```
plot(table(heights) / m, lwd = 3, col = "blue", ylab = "p*")
```



We can sample from the empirical distribution X^*. Recall that we said taking IID samples from X^* is equivalent to sampling from $\mathbf{x}$ with replacement. Here is a sample of $n = 30$ heights. (Note that this is more than the $m = 20$ datapoints we had – but this is no problem if we are sampling with replacement.)

```
sample(heights, 30, replace = TRUE)
```

```
 [1] 182 176 175 170 175 168 177 180 153 177 182 182 180 182 180 190 154 175 153
[20] 176 182 180 181 177 182 180 172 168 180 180
```

Make sure you use `replace = TRUE` to ensure you are sampling *with* replacement.

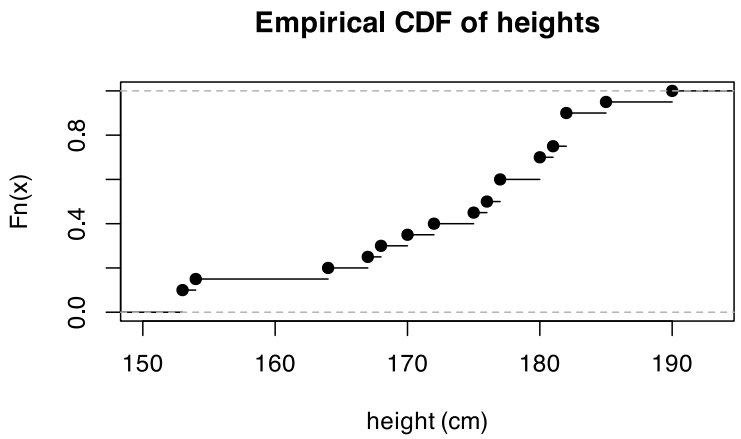
The expectation and variance of the empirical distribution are as follows.

```
heights_mean <- mean(heights)
heights_var <- sum((heights - heights_mean)^2) / m
c(heights_mean, heights_var)
```

```
[1] 173.40 109.64
```

We can form the empirical CDF in R with the `ecdf()` function.

```
Fstar <- ecdf(heights)
plot(Fstar, main = "Empirical CDF of heights", xlab = "height (cm)")
```



The object produced by `ecdf()` is a function. So we can find, for example, the empirical CDF at 172, $F^*(172)$, which is the proportion of a dataset with heights less than or equal to 172 cm.

```
Fstar(172)
```

```
[1] 0.4
```

One more thing on the empirical distribution X^*. If we pick an index J uniformly from $\{1, 2, \ldots, m\}$, then $x_J = X^*$. That is, we can think of X^* as picking one of the datapoints uniformly at random. (This way of looking at X^* is used a lot in the book of Voss.)

25 Plug-in estimation & Bootstrap I

Last time, we defined the empirical distribution of a dataset $\mathbf{x} = (x_1, x_2, \dots, x_m)$. We saw that the empirical distribution X^* has probability mass function

$$p^*(x) = \frac{1}{m} \sum_{j=1}^{m} \mathbb{I}_{\{x\}}(x_j),$$

which is simply the proportion of datapoints taking the value x_j.

25.1 The "plug-in" principle

Suppose now that a sample $\mathbf{X} = (X_1, X_2, \dots, X_m)$ is an IID sample from a populatoin distribution X that is either unknown or too difficult to work with directly. How can we find out things about this distribution X?

Well, we could form the empirical distribution X^* from these samples, and work with that instead.

We have to be a bit careful here, because there two levels of randomness here.

1. First, there is the fact that the samples $\mathbf{X} = (X_1, X_2, \dots, X_m)$ are random IID samples from X.
2. Once we have the samples $\mathbf{X}$, that fixes the empirical PMF p^*. Then X^* is itself a random variable, with PMF p^*.

We will write $\mathbb{E}$, $\mathbb{P}$, Var and so on for the first type of randomness – that is, randomness coming from the random variable X. We will write $\mathbb{E}_*$, $\mathbb{P}_*$, Var_* and so on for randomness coming from the empirical random variable X^* *treating the samples* $\mathbf{X}$ as fixed. So, for example, the expectation $\mathbb{E}_*(\phi(X^*))$ is really shorthand for the *conditional* expectation $\mathbb{E}_*(\phi(X^*) \mid \mathbf{X})$.

One way to estimate something about the random variable X is to take the formula involving X, then keep that same formula, but replace the true random variable X with the empirical random variable X^*. This is called the **plug-in principle**, and such an estimator is a **plug-in estimator** – the idea is that we simply "plug X^* in" to the existing formula.

This is easier to see if we take some examples.

Suppose we wanted to estimate the expectation $\mathbb{E}X$ of the true distribution X. To estimate this, we instead plug in the empirical distribution X^* in place of X and the empirical expectation $\mathbb{E}_*$ in place of the expectation over the random samples $\mathbb{E}$. So our estimator is instead $\mathbb{E}_* X^*$. We saw last time that $\mathbb{E}_* X^*$ is the sample mean

$$\overline{X} = \frac{1}{m} \sum_{j=1}^{m} X_j.$$

So the plug-in estimator for the expectation $\mathbb{E}X$ is the sample mean $\overline{X}$.

Suppose we wanted to estimate the variance $\mathrm{Var}(X)$ of the true distribution. Again, we plug in X^*, to instead find $\mathrm{Var}_*(X^*)$, which we saw last time is

$$\mathrm{Var}_*(X^*) = \frac{1}{m} \sum_{j=1}^{m} \left(X_j - \overline{X}\right)^2,$$

which is very similar to the sample variance of $\mathbf{X}$.

Suppose wanted to estimate $\mathbb{E}\,\phi(X)$ for some function ϕ. The plug-in estimator for this is

$$\begin{aligned}
\mathbb{E}_*\,\phi(X^*) &= \sum_x \phi(x)\, p^*(x) && (25.1)\\
&= \sum_x \phi(x)\, \frac{1}{m} \sum_{j=1}^{m} \mathbb{I}_{\{x\}}(X_j) && (25.2)\\
&= \frac{1}{m} \sum_{j=1}^{m} \sum_x \phi(x)\, \mathbb{I}_{\{x\}}(X_j) && (25.3)\\
&= \frac{1}{m} \sum_{j=1}^{m} \phi(X_j), && (25.4)
\end{aligned}$$

by the same logic we used for the expectation and variance last time. This is the Monte Carlo estimator from the beginning of this module – we have a sample $X_1, X_2, \ldots, X_m$ and we have form the Monte Carlo estimator $\mathbb{E}\,\phi(X)$. This shows there are deep connections between Monte Carlo estimation and the the empirical distribution and plug-in estimation.

25.2 The bootstrap set-up

OK, we're now moving on from the empirical distribution to a slightly different but related topic: the bootstrap.

Suppose a statistician is interested in a statistic $T = T(X_1, X_2, \ldots, X_n)$ of n IID samples from a random variable X. For example, this might be:

- Suppose I pick a basketball squad of 12 players at random; what is their average height? Here, X is the distribution of basketball players' heights, $n = 12$, and the statistic is

$$T = T(X_1, X_2, \dots, X_{12}) = \frac{1}{12} \sum_{i=1}^{12} X_i.$$

- Suppose I visit The Edit Room cafe 5 times; what's the longest queue I have to deal with. Here, X is the distribution of queue lengths at The Edit Room, $n = 5$, and the statistic is

$$T = T(X_1, X_2, X_3, X_4, X_5) = \max\{X_1, X_2, X_3, X_4, X_5\}.$$

- Suppose a supermarket distributor buys 1001 beef steaks; what is the median weight of the steaks? Here X is the distribution of weights of steaks, $n = 1001$, and the statistic is

$$T = T(X_1, X_2, \dots, X_{1000}) = \text{median}(X_1, X_2, \dots, X_{1001}).$$

The statistician is likely to be interested in properties of this statistic. For example, three of the most important things the statistician is likely to want to know are:

- The **expectation** $\mathbb{E}T = \mathbb{E}T(X_1, \dots, X_n)$ of the statistic.
- The **variance** $\text{Var}(T) = \text{Var}(T(X_1, \dots, X_n))$ of the statistic – or related concepts like the standard deviation.
- A **prediction interval** $[U, V]$ for the statistic, such that $\mathbb{P}(T \in [U, V]) = 1 - \alpha$.

Now, if the statistician knew the true distribution X, and if it were simple enough to work with, then she could calculate the true values of these. But suppose the distribution is unknown (or too complicated to work with). Instead, the statistician just has m samples $\mathbf{X} = (X_1, X_2, \dots, X_m)$. You could think of these as data measurements that are modelled as coming from the distribution X, or you could think of them as output from a computer program that can sample from X exactly.

Note that there's two numbers here: n is the number of samples required to calculate the statistic $T = T(X_1, X_2, \dots, X_n)$ once, and m is the total number of samples we have available. The most common situation is "m is somewhat bigger than n, although not vastly bigger", but the mathematical definitions are valid for any n and m.

The **bootstrap** method is the following idea:

1. Take n samples from the empirical distribution X^* of $\mathbf{X}$. This is equivalent to sampling n of the values $X_1, X_2, \dots, X_m$ with replacement. Let's call these samples $X_1^*, X_2^*, \dots, X_n^*$. Evaluate the statistic with these samples

$$T^* = T(X_1^*, X_2^*, \dots, X_n^*).$$

2. Repeat step 1 many times; let's say B times. Keep taking n of the samples with replacement and evaluating the statistic. We now have B versions of that statistic $T_1^*, T_2^*, \dots, T_B^*$.

3. Use these B versions of the statistic to get a **bootstrap estimate** of the expectation, variance, or prediction interval. To estimate the expectation of the statistic $\mathbb{E}T = \mathbb{E}T(X_1, \dots, X_n)$, use the sample mean of the evaluated statistics $\overline{T^*} = \frac{1}{B}\sum_{k=1}^{B} T_B^*$. To estimate the variance $\mathrm{Var}(T)$ use the sample variance

$$\sum 1B - 1 \sum_{k=1}^{B} (T_k^* - \overline{T^*}).$$

   We'll come back to the prediction interval next time.

The bootstrap concept was discovered by the American statistician Bradley Efron in a hugely influential paper "Bootstrap methods: another look at the jackknife" in 1979. The name "bootstrap" comes from the phrase "to pull yourself up by your bootstraps", which roughly means to make progress without any outside help, in a way that might initially seem impossible – similarly, the bootstrap manages to estimate properties of a statistic by just reusing the same set of samples over and over again. (The "jackknife" in the title of Efron's paper was and earlier, simpler, less powerful idea along similar lines, named after the multipurpose tool the jackknife.)

25.3 Bootstrap for expectation and variance

Example 25.1. Let's take the cafe example above. The statistic in question is

$$T = T(X_1, X_2, X_3, X_4, X_5) = \min\{X_1, X_2, X_3, X_4, X_5\}.$$

A researcher wants to estimate the expectation of this statistic.

The researcher visits The Edit Room at 30 random occasion and notes the following data.

Queue length	0	1	2	3	4	5	7	11	Total
Number of occasions	11	5	7	3	1	1	1	1	30

We start by taking 5 samples from the empirical distribution – that is, we choose 5 of the datapoints uniformly at random with replacement. Let's say these are $(0, 1, 4, 4, 5)$. (It turns out we sampled the value 4 twice, even though it only occured once – that does happen sometimes when we're sampling with replacement.) The value of the statistic for this sample is

$$T_1^* = T(0, 1, 4, 4, 5) = \max\{0, 1, 4, 4, 5\} = 5.$$

We keep doing this many times – we pick five samples with replacement, and calculate their maximum.

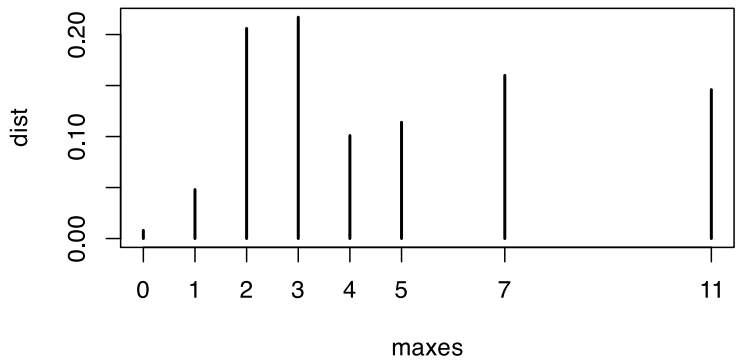
```
queues <- c(rep(0, 11), rep(1, 5), rep(2, 7), rep(3, 3), 4, 5, 7, 11)

boots <- 1000
maxes <- rep(0, boots)
for (k in 1:boots) {
  minisample <- sample(queues, 5, replace = TRUE)
  maxes[k] <- max(minisample)
}
```

This gives us 1000 realisations of the test statistic. We can use these to look at the distribution of the test statistic:

```
dist <- table(maxes) / boots
dist
```

```
maxes
    0     1     2     3     4     5     7    11
0.008 0.048 0.206 0.217 0.101 0.114 0.160 0.146
```

```
plot(dist)
```



We can also look at particular figures of interest. For example, the expectation of T should be close to the sample mean of our T^*s, and the variance of T should be close to the sample variance of our T^*s.

```r
c(mean(maxes), var(maxes))
```

```
[1] 4.81100 9.66094
```

Next time, we'll look into bootstrap methods in more detail.

26 Bootstrap II

26.1 Bootstrap with a prediction interval

Recall where we had got to last time. We are interested in a statistic $T = T(X_1, \dots, X_n)$, where $X_1, \dots, X_n$ are IID copies of a random variable X. We want to find out about the distribution of T. But all we have to work with is $X_1, \dots, X_m$, where are$m$ IID samples from X.

So the bootstrap procedure is to repeatedly choose n samples $X_1^*, \dots, X_n^*$ from the empirical distribution X^* – or, equivalently, choose n of the values $X_1, \dots, X_m$ with replacement, and calculate the statistic $T_k^* = T(X_1^*, \dots, X_n^*)$. The distribution of the T_k^* should be similar to the distribution of T.

In particular, to estimate $\mathbb{E}T$ we can use the sample mean of the $T_1^*, \dots, T_B^*$, and to estimate $\mathrm{Var}(T)$, we can use the sample variance S^2 of the T_k^*.

What if we want to estimate a prediction interval – that is, an interval $[U, V]$ such that $\mathbb{P}(U \leq T \leq V) \approx 1 - \alpha$?

There is a lazy way to do this, which is to hope that the T_k^* are approximately normally distributed. With $\overline{T^*}$ as the sample mean and S^2 as the sample variance of our observed statistics, we could take

$$\left[\overline{T^*} - z_{\alpha/2}\, S,\, \overline{T^*} + z_{\alpha/2}\, S\right].$$

But we can do better than this by taking the actual distribution of the T_k^*, which might not be normal, into account.

Instead, we can take the sample quantiles of the T_k^*. That is, put $T_1^*, \dots, T_B^*$ in increasing order. Go $\alpha/2$ of the way up the list to get the lower-$\alpha/2$ sample quantile, and $1 - \alpha/2$ of the way up the list to get the upper-$\alpha/2$ sample quantile. These two values can be our prediction interval.

Example 26.1. Let's go back to The Edit Room cafe queues from last time. The statistic in question was the maximum queue length from 5 visits. The data and our bootstrap samples were these:

```
queues <- c(rep(0, 11), rep(1, 5), rep(2, 7), rep(3, 3), 4, 5, 7, 11)

boots <- 1000
maxes <- rep(0, boots)
for (k in 1:boots) {
  minisample <- sample(queues, 5, replace = TRUE)
  maxes[k] <- max(minisample)
}
```

We saw that our estimates for the expectation and the variance of the statistic were the following:

```
max_mean <- mean(maxes)
max_var <- var(maxes)
c(max_mean, max_var)
```

```
[1] 4.738000 9.883239
```

A lazy 80% prediction interval under a normal assumption would be the following:

```
max_mean + qnorm(c(0.1, 0.9)) * sqrt(max_var)
```

```
[1] 0.7091069 8.7668931
```

But that seems a bit silly – our queue length isn't going to be a real number with seven decimal places. Better is to use the actual quantiles of the statistics we evaluated.

```
quantile(maxes, c(0.1, 0.9))
```

```
10% 90%
  2  11
```

I usually get the interval $[2, 11]$ from this data. this much better reflects the actual data. It also takes into account the "skew" of the data (lots of values of 0, where there was no queue, but no negative values, of course), to give a tighter lower boundary than the crude and inaccurate normal approximation.

26.2 Bootstrap for statistical inference

So we've seen two ideas here about what to do when we only have samples from a distribution (for example, some data measurements).

1. The first idea was that to estimate something about the distribution, use the **plug-in estimator**.
2. The second ideas was the to find out about properties of a statistic, use **bootstrap** sampling.

But an estimator $\widehat{\theta}$ is just a special type of statistic – one that is hoped to be close to some true parameter θ. So we can combine these two ideas together. First, we estimate the true parameter θ, use the plug-in estimator θ^*. Then we can find out things about that estimator θ^* – for example, its bias, variance, mean-square error, or a confidence interval – using boot resampling.

One way to think about this is the "two world" of bootstrap estimation:

- **The real-world:** We consider the data $X_1, X_2, \ldots, X_m$ as a random sample from the population; and due to that randomness, there is variability of the plug-in estimator θ^* around the true value θ.
- **The bootstrap world:** We treat the samples $X_1, X_2, \ldots, X_m$ as fixed; bu there is variability of the bootstrap estimates T_k^* around the plug-in estimate θ^*, due to the sampling-with-replacement within the bootstrap procedure.

The key idea of bootstrapping is this: that the variability of θ^* around θ in the real world can be approximated by the variability of the T_k^*s around θ^* in the bootstrap world.

26.3 Bootstrap estimation of bias

Suppose we have parameter θ we wish to estimate using a dataset $X_1, X_2, \ldots, X_m$. Our best "point estimate" for θ is the plug-in estimator θ^*.

How might we estimate the bias of θ^* as an estimate of θ?

Well, the bias in "the real world" is $\mathbb{E}\theta^* - \theta$, the expected value of the estimator θ^* minus the true value θ. In the bootstrap world, the role of θ is replaced θ^* and the role of θ^* is played by the T_k^*s; so the bias $\mathbb{E}\theta^* - \theta$ is estimated by $\overline{T^*} - \theta^*$

In other words, we repeatedly sample m points $X_1^*, \ldots, X_m^*$ with replacement from the dataset, and calculate the bootstrap statistic $T_k^* = \theta^*(X_1^*, \ldots, X_m^*)$ Then the estimate of the bias is

$$\overline{T^*} - \theta^* = \frac{1}{B}\sum_{k=1}^{B} T_k^* - \theta^*.$$

Example 26.2. Let's talk through an example. Let's suppose we want to estimate the average number of hours sleep people have per night. That is, we want to know $\theta = \mathbb{E}X$, where X is the random distribution of sleep times for the entire population.

We have data on $m = 1991$ people thanks to a survey by the US Centers for Disease Control and Prevention. (Credit: Scott, *Data Science in R*.) We can read this into R as follows.

```
sleep <- read.csv("https://bookdown.org/jgscott/DSGI/data/NHANES_sleep.csv")$SleepHrsNight
m <- length(sleep)
```

Our estimate for the median sleep time $\theta = \mathbb{E}X$ will be the plug-in estimator $\theta^* = \mathbb{E}_* X^*$, which, as we have discussed before, is the sample mean $\theta^* = \overline{X}$.

```
estimate <- mean(sleep)
estimate
```

```
[1] 6.878955
```

This is 6.8790 hours.

But we should check whether our estimator is likely to be biased or not. (Although, actually, in this case we do know the mean is an unbiased estimator of the expectation, so the answer is 0.)

We form our bootstrap statistics as follows:

```
set.seed(3)

boots <- 1e4
bootests <- rep(0, boots)
for (k in 1:boots) {
  resample <- sample(sleep, m, replace = TRUE)
  bootests[k] <- mean(resample)
}
est_exp <- mean(bootests)
est_exp
```

```
[1] 6.879461
```

This typically comes out as very close to the original estimate. With the seed set as 3 for reproducibility, we get 6.8795. So our estimate of the bias is

$$\widehat{\text{bias}}(\theta^*) = \widehat{\mathbb{E}(\theta^*)} - \theta^* = 6.8795 - 6.8790 = 0.0005.$$

This is very small (as we expected).

Had we found some bias, it is often preferable to improve our estimator by subtracting that bias. The refined, or "debiased", estimator would be

$$\theta^* - \widehat{\text{bias}}(\theta) = \theta^* - \left(\widehat{\mathbb{E}(\theta^*)} - \theta^*\right) = 2\theta^* - \widehat{\mathbb{E}(\theta^*)}.$$

26.4 Bootstrap estimation of MSE

What about the mean-square error of the plug-i estimator? Can we estimate the MSE? Well, the mean-square error, in the real world is

$$\text{MSE} = \mathbb{E}\left(\theta^* - \theta\right)^2.$$

In the bootstrap world, we estimate this by

$$\overline{(T^* - \theta^*)^2} = \frac{1}{B}\sum_{k=1}^{B}\left(T_k^* - \theta^*\right)^2,$$

where θ is replaced by θ^* and θ^* is replaced by the T_k^*s.

Example 26.3. We continue the sleep example.

```
MSEest <- (1 / boots) * sum((bootests - estimate)^2)
MSEest
```

```
[1] 0.0008791049
```

In the case of this simple parameter $\theta = \mathbb{E}X$, we could also have estimated the MSE in the tradition Monte Carlo-style way

```
var(sleep) / m
```

```
[1] 0.0008717188
```

Reassuringly, these come out as extremely close to each other.

Next time, we will look at the more complicated question of estimating a confidence interval for the plug-in estimator.

Problem Sheet 5

This is Problem Sheet 4, which covers material from Lectures 22 to 27. You should work through all the questions on this problem sheet in advance of the problems class, which takes place in the lecture of **Thursday 11 December**.

This problem sheet is to help you practice material from the module and to help you check your learning. It is *not* for formal assessment and does not count towards your module mark.

If you want some brief informal feedback on **Question 2** (marked), you should submit your work electronically through Gradescope via the module's Minerva page by **1400 on Tuesday 9 December**. (If you hand-write solutions on paper, the easiest way to scan-and-submit that work is using the Gradescope app on your phone.) I will return some brief comments on your those two questions by the problems class on Thursday 11 December. Because this informal feedback, and not part of the official assessment, I cannot accept late work for any reason – but I am always happy to discuss any of your work on any question in my office hours.

Full solutions will be released on Friday 12 December.

Remember that **Thursday 11 December at 1400** is also the deadline for the module course-work

Computational coursework

There is one piece of computational coursework for MATH5835M Statistical Computing. This coursework is worth 20% of the module mark.

The coursework sheet is available in **here in HTML format** or **here in R Markdown format**.

The deadline for the coursework will be the penultimate day of the Autumn term, **Thursday 11 December** at **1400**. (There is also an optional deadline if you wish to get feedback on a draft of your work: Friday 5 December at 1400.)

Feedback on the final report and marks will be returned on Monday 12 January, the first day of the Spring term.

About the coursework

There is one piece of computational coursework for MATH5835M Statistical Computing. This coursework is worth 20% of the module mark.

This page contains all the administrative and organisational information about the coursework.

A coursework sheet contains the tasks you must carry out. The coursework sheet is available in **here in HTML format** or **here in R Markdown format**. The HTML page can be opened in a standard web browser. The R Markdown version (which I personally prefer) should be downloaded then opened in RStudio; I recommend using the "Visual" editor – see the button in the top left of the editing window.

The are two deadlines for this work.

1. There is the **main submission deadline** which is the penultimate day of term, **Thursday 11 December** at **1400**. You must submit your work by this time. Work that is submitted after this deadline will receive a penalty of 5% for every day or part-day late. Work that is more than 14 days late will not be marked and will receive 0. Work will be submitted electronically through Gradescope via the Minerva page for the module.

2. *Optionally*, you *may* also wish to get some brief informal feedback on your draft work before the main deadline. *If* you wish to get feedback on your draft work, you should submit that draft by Friday 6 December at 1400. I will return a small amount of general feedback about your work by Monday 9 December. I will not give a mark to your draft, and the feedback will be a brief sentence or too that may be of a little help when completing the final version of your work. The quality of your draft submission will have no effect (either positive or negative) on the mark for your main submission, and there is no penalty if you choose not to take advantage of this offer. You still must submit your final work by the main submission deadline, even if you submit a draft for feedback. No extensions can be offered on the deadline for this informal feedback.

The Gradescope submission for the final report will open after 1400 on Friday 6 December (to avoid drafts for optional feedback getting mixed up with final reports). The Gradescope submission for optional feedback on draft work will be open from the computer practical sessions in week 9.

You must comply with the University's rules on academic integrity. See below for comments on use of AI.

If you encounter extraordinary unforeseeable personal circumstances that make it impossible for you to submit your coursework on time, you can apply for an deadline extension through the usual mitigating circumstances procedure. (Do not contact me about this – I cannot unilaterally offer deadline extensions.)

About your report

Your task is to write a short report in response to the tasks on the coursework sheet. Your report should include all the important R code you use and any important plots you draw. If I can't work out what R code you ran to get your results, I cannot award you marks for those results. Only include the code and figures relevant to your final and best solution – I don't need or want to see earlier attempts or errors along the way, unless they genuinely help explain your final choices.

All computational work must be done in R – no marks will be given for code written in Python or any other language. It is strongly recommended that you draw figures in R (with the possible exception of rough illustrative sketches, if you find them useful), but I do not insist on this.

You report must end with a declaration of AI use. See below for more on this.

I call your output a "report" because your work should be explained in detail, in full English-language sentences, with your solutions discussed and justified. The code and figures you include should be an integral part of your report, not just copy-pasted in at random, and

should be fully explained within the text. However, other parts of a more formal "report", such as an introduction, literature review, conclusion, references, etc, are not required here.

There is *no page limit* for the report, although I expect that good reports will typically be around 5–8 pages. If your report is 4 pages or less, make sure you have completed all the tasks and fully explained your work. If your report has 10 pages or more, make sure you are not wandering from the point or including too many unnecessary figures.

Your report should be prepared electronically, but can be submitted in any sensible format – I recommend PDF (whether produced by LaTeX, R Markdown, Microsoft Word, or any other way), but HTML or Word document are fine too.

Computer practical and other help

There will be a computer practical in Week 9 where I will introduce the practical coursework in more detail. This is a good place to ask any questions you have about the coursework. You may wish to bring your own laptop to the computer practical, if you prefer working on that to on University computers.

Places where you can get help with the coursework include:

- In the computer practical session.
- In my office hours: Thursdays 1300–1400 in my office, 9.10n in the "Maths Research Deck" area on the 9th floor of the EC Stoner building at staircase 1. (It is unlikely I will have time to discuss computational work at the beginning or end of a lecture, although if your question is *extremely* short you can try.)
- By submitting your draft work for optional feedback, as discussed above.

AI use

This coursework is rated AMBER for use of AI. This means that generative AI may be used in an "assistive role" only, provided you disclose (that is, tell me about) this use.

The following uses of generative AI go beyond merely "assistive" and are **not permitted**. Work that I suspect has used generative AI this way will be reported as an academic integrity case.

- Asking an LLM chatbot how to answer the questions on the problem sheet.
- Writing R code for you.
- Writing the text of your report for you.

The following uses of generative AI are only "assistive", and are permitted *provided you tell me that you used generative AI this way.* Work that uses AI this way, where the AI use *is* disclosed, will not be penalised in any way. Work that uses AI this way where the AI use is *not* disclosed will be treated as an academic integrity case.

- General revision of topics from the module, not particularly tailored to the coursework problems.
- Debugging almost-complete R code that you wrote yourself.
- Spell-checking and grammar-checking a report that you wrote yourself.
- Help with LaTeX code, if you choose to write your report in LaTeX.

All reports must end with a **declaration of generative AI use**.

- If you did not use generative AI at all, this can simply say: *"I did not use generative AI for the coursework project."*
- If you did use generative AI, this should say what system you used, what tasks it performed, and how you prompted it. Declarations like these should be one paragraph long, typically around 40–100 words.

For example, here are two hypothetical declarations:

> **Declaration of generative AI use (1).** The only place I used AI in this coursework was to spell- and grammar-check my report. I uploaded my Word document to Copilot, and prompted it: "Please check this report for spelling and grammar mistakes." It found 5 small errors, which I corrected.
>
> **Declaration of generative AI use (2).** I used AI to help when I had a bug in my R code that I couldn't fix myself. I copy-and-pasted my R script into ChatGPT, along with the text of the error from RStudio, and asked ChatGPT what the error was. It pointed out that, in drawing a graph, I had written `col = blue` without quotation-marks around `"blue"`. This was the only time I used AI for the coursework.

R and RStudio on University computers

A quick reminder, if you need it, on how to access R and RStudio on University computers.

1. **Open the AppsAnywhere portal** This should be a link on the desktop. If invited to run any software, accept.

2. **Load the language R** Search on AppsAnywhere for "R for Windows" (or similar) and launch it. This will (silently) load the language R. It will also open an inferior RStudio-like program called "RGui" – you can close it.

3.**Launch RStudio** Search on AppsAnywhere for "RStudio" (or similar) and launch it.

You can alternatively use the Posit Cloud to access R and RStudio online.

Solutions

Problem Sheet 1

Problem Sheet 2

Problem Sheet 3

Problem Sheet 4