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 $X_1, X_2 \dots, X_n$, 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 $X_1, X_2, \dots, X_n$ is a "**random sample** from X" to mean that $X_1, X_2, \dots, X_n$ 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 $X_1, X_2, \dots, X_n$ 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.

```r
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

```r
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

```r
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

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

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

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

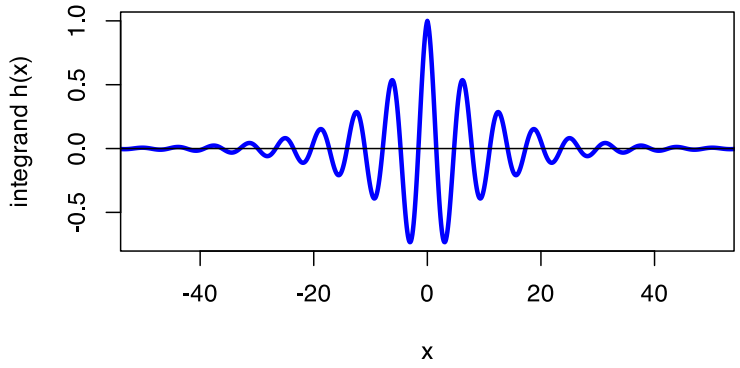
But which normal distribution should we take? Well, we're *allowed* to take any one – we will still get an accurate estimate in the limit as $n \to \infty$. But we'd like an estimator that gives accurate results at moderate-sized n, and picking a "good" distribution for X will help that.

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,

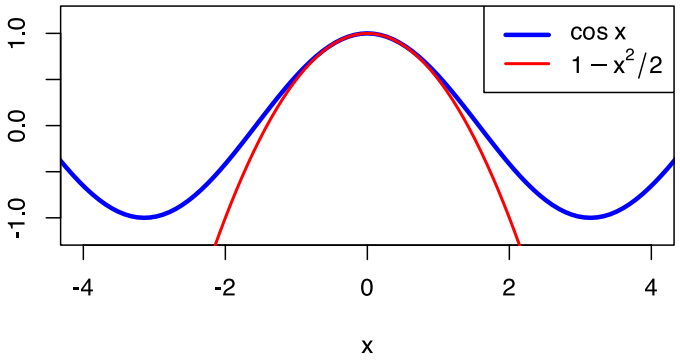
$$\cos x \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots.$$

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 $\mathbb{E}\,\phi(X)$.

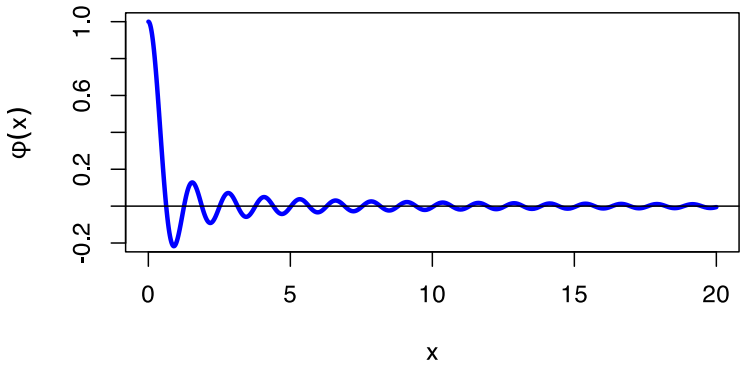
Why might we want to do this? There are two main reasons:

- 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

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

so the relevant weighting of a sample y is

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

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

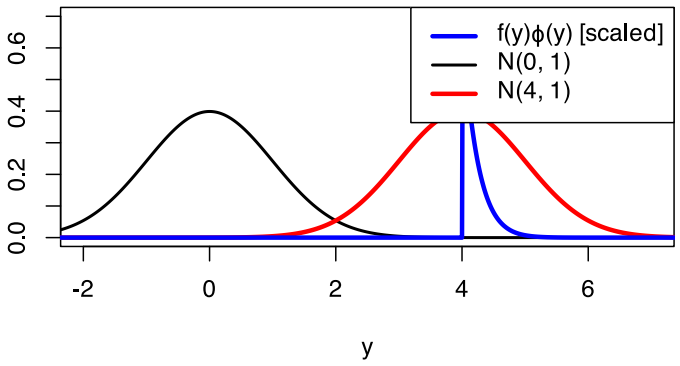
$$f(y)\,\phi(y) = f(y)\,\mathbb{I}_{(4,\infty)}(y) = \begin{cases} \dfrac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-y^2/2} & y > 4 \\ 0 & y \le 4 \end{cases}$$

(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 $\mathbb{E}\,\phi(X)$.

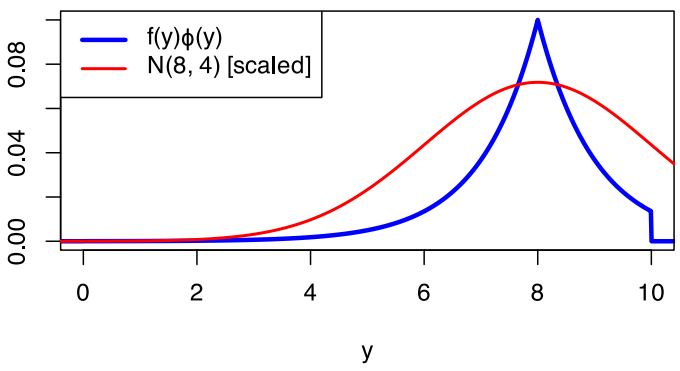
The standard Monte Carlo estimator and its RMSE are as follows

```
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).

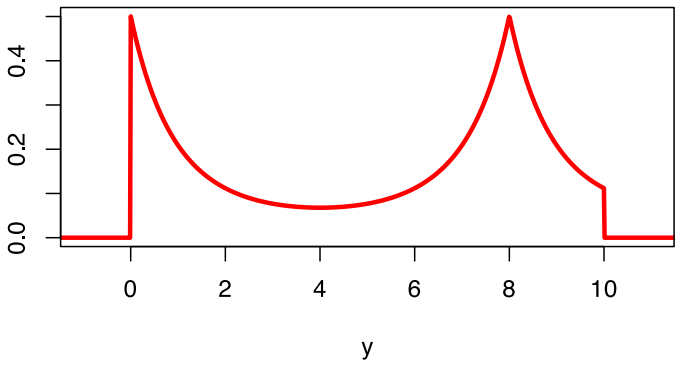
```
curve(
  dunif(x, 0, 10) * exp(-abs(x - 8)), n = 1001, from = -1, to = 11,
  col = "blue", lwd = 3,
  xlim = c(0, 10), xlab = "y", ylab = ""
)
curve(
  dnorm(x, 8, 2)*0.36, n = 1001, from = -1, to = 11,
  add = TRUE, col = "red", lwd = 2,
)
legend(
  "topleft",
  c(expression(paste("f(y)", varphi,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

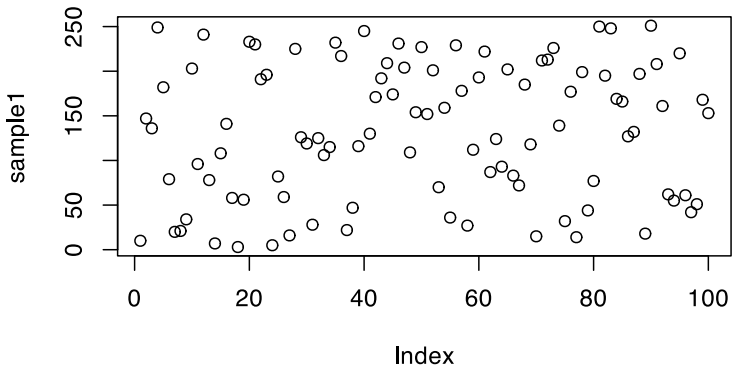
```
lcg <- function(n, modulus, mult, incr, seed) {
  samples <- rep(0, n)
  samples[1] <- seed
  for (i in 1:(n - 1)) {
    samples[i + 1] <- (mult * samples[i] + incr) %% modulus
  }
  return(samples)
}
```

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.

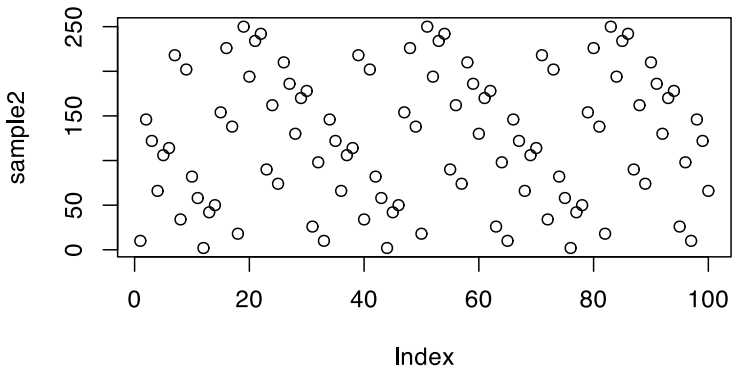
```
m <- 2^8
sample1 <- lcg(100, m, 13, 17, 10)
plot(sample1)
```



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

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

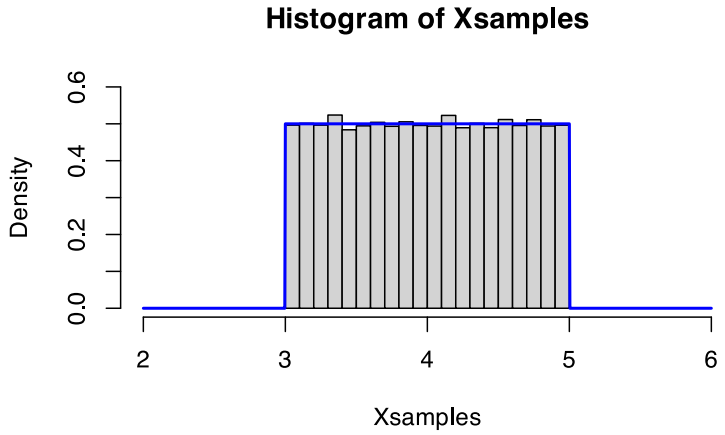
We know how to generate $U \sim \mathrm{U}[0,1]$. But suppose we want to generate $X \sim \mathrm{U}[a,b]$ instead, for some $a < b$; how can we do that.

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

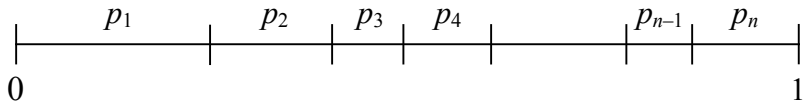
Suppose we want to simulate a Bernoulli trial; that is, a random variable X that is 1 with probability p and 0 with probability $1-p$, for some p with $0 < p < 1$. As ever, we only have a standard uniform $U \sim$ U$[0, 1]$ to work with. How can we form our Bernoulli trial?

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

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.

Solutions

Problem Sheet 1

Problem Sheet 2