Computational statistics

Daniel Mortlock (mortlock@ic.ac.uk)
Last modified: September 9, 2014

1 Random numbers

1.1 Definitions

The word "random" is used very often and with great confidence. Most people have an intuitive sense of what "random" means and hence what a "random number" is. If, however, random numbers are to be used in statistical algorithms then it is important to have a more precise definition of what they are, how randomness can be assessed, how random numbers can be generated using (fundamentally deterministic) computer-based algorithms.

A starting point is to look at some definitions of random:

- "made, done, happening or chosen without method of conscious decision"
  (source: Mac Dictionary app)
- "unsystematic, unmethodical, arbitrary, unplanned, undirected, casual, indiscriminate, nonspecific, haphazard, stray, erratic, chance, accidental"
  (source: Mac Dictionary thesaurus)
- "statistics governed by or involving equal chances for each item"
  (source: Oxford Dictionaries)
- "having no definite aim or purpose; not sent or guided in a particular direction; made, done, occurring, *etc.*, without method or conscious choice; haphazard"
  (source: Oxford English Dictionary)

Question: *How would you define randomness? Imagine that you had to define this concept for an intelligent person with no background in mathematics.*

A random number might then be expected to be a numerical quantity satisfying the above properites, but of course the nature of mathematics is such that a precise definitions might seem possible – and is certainly desireable. Some possible definitions are:

- "a sequence (*e.g.*, of bits) that is shorter than any computer program that can produce it"
  (Kolmogorov randomness)
- "a sequence in which the value of any one element (or sub-sequence of elements) provides no information about the value of any other element"
  (my attempted definition)
- "a sequence in which each element is generated by a independent stochastic process" (my alternative attempted definition)

Question: *Try and come up with an example that satisfies the above definitions. Try and come up with an example that contradicts the above definitions.*

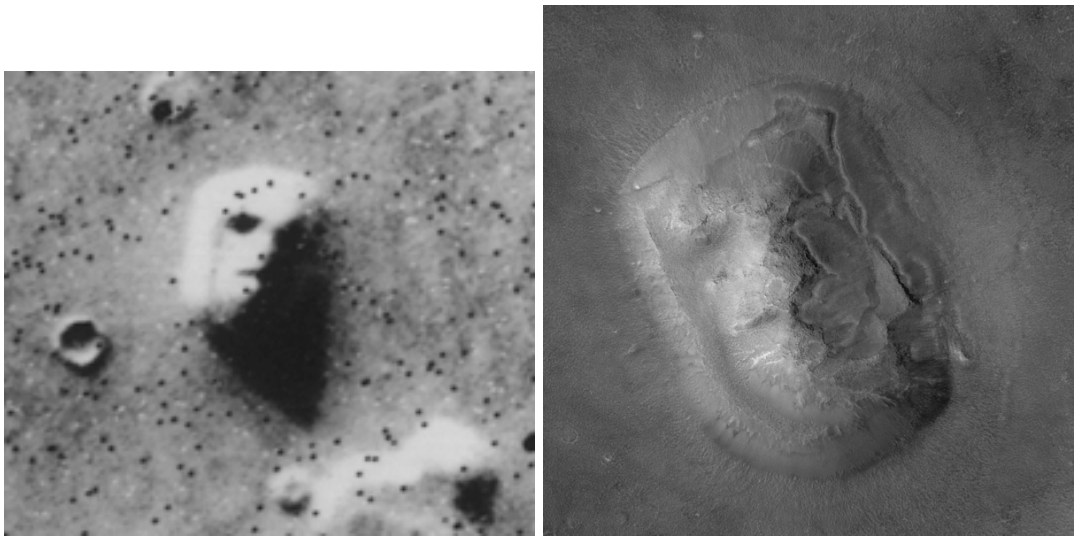

Figure 1: Images of the "face on Mars" produced by Viking 1 (left) and the Mars Global Surveyor (right).

Question: *Try and come up with a rigorous mathmatical definition of randomness. See if you – or a classmate – can come up with a counter-example that contradicts this definition.*

The above definitions, while precise, are also somewhat abstract, in the sense that they do not immediately define an answer as to whether a given number fits the definition of randomness. If a characteristic can be defined then it ought to be possible to determine whether something fits that definition. A good test, then, for whether randomness has been precisely defined, is whether it can be identified and/or tested for.

1.2 Tests for randomness

Developing and implementing methods to test for randomness is an entire sub-field of mathematics that brings in ideas from statistics, information theory and even psychology.

The reason for psychological considerations is that the human mind has evolved to be so good at identifying patterns (*i.e.*, the antithesis of randomess). Indeed, if anything, the human mind is *too good* at finding patterns, in the sense that it regularly ascribes significance to what are really mere coincidences:

- In 1975 the NASA Viking 1 probe took images of the surface of Mars, one of which is shown in left panel of Fig. 1. Most people instantly recognise this as a face, leading to a huge number of wild theories about Martian civilisations – and even a central plot point in the film *Mission To Mars*. However, subsequent missions had a much higher resolution cameras, demonstrating that the "face" was in fact just a fairly ordinarily mountain that had been lit from a particular angle when initially observed by Viking.
- Apple received complaints from early iPod users that the shuffle feature was not (completely) random. The main "problem" was that it too often played songs by the same artist – or even the same song – back-to-back. Of course this sort of coincidence should happen sometimes, and testing revealed that the RNG that Apple was using was not at fault. The

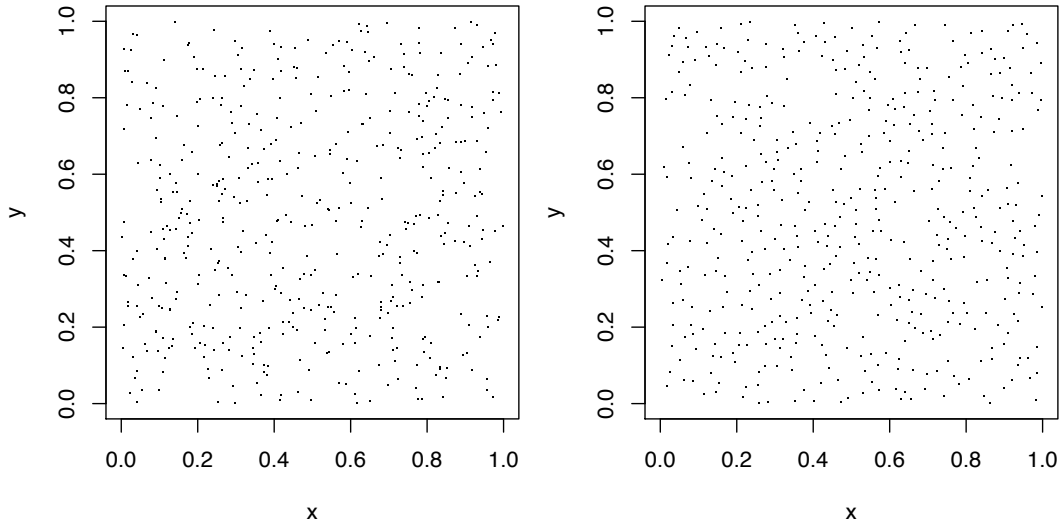

Figure 2: Two sets of points produced using different stochastic processes, only one of which is the canonical algorithm of drawing points independently from the uniform distribution over the unit square.

solution was to actually make the song sequence less random, adding anti-correlations to decrease the chance of playing songs by the same artist twice in a row (and precluding a repeat performance of any given song).

- The images in Fig. 2 show two distributions of points produced by different processes: one is a set of independent draws from the uniform distribution over the unit square; the other has correlations included. Which is which?

All of the above examples can be converted into numbers (or sequences of numbers), and so it is worth examining whether the human has the same fallibilities when faced with more abstract objects, that maybe have less of a link to evolutionary drivers. In other words, can the human mind accurately assess whether numbers random?

Question: *Which of these numbers is random?*

- *11111111 [. . .]?*
- *31415926 [. . .]?*
- *64338327 [. . .]?*
- *64914401 [. . .]?*
- *31173451 [. . .]?*

Anyone who answered the above questions presumably has some sort of vague algorithm for assessing randomness in their mind; but in the cases where a number was shown to have some simple provenance it is possible that your initial answer changed in light of this new information. For this reason it is debatable that a single number can even be thought of as random in principle – perhaps it is only meaningful to think of randomness as a (potential) property of a sequence of numbers, or of the process that can generate such a sequence (or both)?

Question: *Which of the following sequences are random?*

- *222, 222, 222, 222, 222, 222, 222, 222, [. . .]?*
- *979, 323, 846, 264, 338, 279, 502, 884, [. . .]?*
- *615, 371, 559, 155, 105, 314, 106, 024, [. . .]?*
- *726, 327, 487, 966, 329, 692, 014, 392, [. . .]?*

It was maybe possible to assess the randomness of these sequences more easily than to assess individual numbers . . . but the fact that these sequences could also simply be strung together as longer numbers – just remove the commas above! – implies that distinction between single numbers and sequences is rather arbitrary, as both can be arbitrarily long.

Reversing this argument then implies that decimal digit is not a fundamental quantity, as it could itself be represented as a sequence of (four) binary digits. A single binary digit – a bit that takes on one of just two values – is, however, an indivisible quantity. Given that a bit is the unit of information, this reinforces the notion that randomness is inextricably linked to information, something which is of particular relevance to Bayesian inference.

Considering bits – and, by extension, representing numbers in base 2 – makes at least some questions clearer. It is surely nonsensical – and certainly not very useful – to try and determine if 1 is random, any more than it is to ask if 0 is random. The process which led to the value of a certain bit being set might be random, but the value of the bit just is. Thinking in base 2 is also rather useful for side-stepping the biases and prejudices of the human mind, because most people seldom think of numbers in base 2 and so do not recognise them nearly so well.

Question: *Are these binary sequences random?*

- *111111111111111111111111 [. . .]?*
- *110010010000111111011010 [. . .]?*
- *011011100101110111100010 [. . .]?*
- *001111011100000010110011 [. . .]?*

Objective tests for randomness

The final step in eradicating subjectivity from random number testing is to make the test itself mathematical. This is a near textbook example of a classical statistical hypothesis test: there is a sharply defined null hypothesis (*i.e.*, that the sequence being investigated is random) that makes specific and testable predictions against which any sequence can be compared.

It is possible to show all but decisively that a finite sequence is non-random, although a formal proof is never possible. To see this, consider a truly random process being used to generate a sequence of N bits. Each of the 2^N sequences, including $0, 0, 0, \ldots, 0, 0, 0$, is not only possible, but equally likely. Hence no self-consistent test could give $\Pr(\text{random}) = 0$, although a good test would give $\Pr(\text{random}) \ll 1$, and $\Pr(\text{random}) \to 0$ as $N \to \infty$.

The majority of tests for randomness are based on summary statistics that i) can be computed easily from a sequence and ii) have a simply calculated sampling distribution. Examples include:

- the distribution of digits (which generally should be uniform from 1 to N, where N is the base)
- the distribution of length M sub-sequences (which reduces to the above if the base is changed to N^M)
- the correlation betwen elements of the sequence at different lags
- the distribution in higher dimensions produced when elements are combined as the components of vectors
- lengths of runs of consecutive digits
- just about any other statistic you can think of!

The sampling distributions of these statistics are known under the null hypothesis (*i.e.*, that the sequence is random), but there is no obvious alternative to compare against. Hence Bayesian model comparison techniques cannot be applied and instead classical hypothesis tests – based on the p-value of the test statistic of the sequence being investigated – are typically used.

Question: *Come up with a test for randomness yourself. It might help to consider definite examples of sequences that you would identify as non-random and which you want your test to successfully identify as such.*

Is π a random number?

Many of these points come together when considering π. It is, fundamentally, a single number, but can be treated as a sequence of digits, or a sequence of numbers (*i.e.*, sub-sequences of consecutive digits) as well. Indeed, π has probably been studied more than any other number in the context of randomness (and in other contexts, too). These studies revealed many interesting properties of of π, but they have also revealed much about the difficulties reasoning well about hypothesis testing.

For instance, Tu and Fischbach (2005) attained some prominence by subjecting the digits of π to a series of null tests, which they also applied to sequences produced by RNGs. Unsurprisingly, there were no decisive rejections of the null hypothesis, but the potential traps in such reasoning

are well illustrated by Ephraim Fischbach, one of the authors of the study, who was quoted[1] "Our work showed no correlations or patterns in π's number set – in short, π is indeed a good source of randomness. However, there were times when π's performance was outdone by the RNGs." The second statement seems to contradict the first – how can a sequence be more random than one in which no patterns were found?

Still, it is interesting that a completely determined mathematical constant can be a good source of random numbers – this all but contradicts some of the definitions of randomness discussed above (and that you might have had).

1.3 Random number generation

The above discussion implicitly makes some distinction between a random number and a process that might be used to generate a random number, with a particular focus on what these ideas mean. Critical as such questions are, it is far more important that it is possible to generate (sequences of) numbers that have at least some of the properties of randomness described above.

1.3.1 Deterministic physical systems

Most people used physical systems to generate random numbers with much simpler physical systems: decks of cards; dice; coins. The tossing of a coin is a particularly interesting example, as the dynamics of a falling, rotating coin are very simple, and not significantly chaotic at all. A machine can toss a coin so that it reliably lands the same way, and a person can learn to do this to some degree as well. Given this, it is perhaps surprising that coin tossing is used as a way to generate what is effectively a random bit.

Strange as it may seem, there is no requirement that the system used to generate random numbers even be dynamic. Reasonably random bits could be generated by asking, *e.g.*, whether a tall building has an even or odd number of floors, or whether a street name has an even or odd number of consonants. It is plausible that the these will not be 50/50 splits, but they are random in the sense of being unknown a priori.

1.3.2 Chaotic physical systems

The macroscopic world (*i.e.*, at human sizes and time scales) is very close to being deterministic – the sheer number of atoms ($\gtrsim 10^{20}$) in any object that a person can manipulate is sufficient that the law of large numbers means that the quantum mechanical stochasticity of individual atoms is almost completely averaged out. Newtonian mechanics is sufficient to explain the movement of an animal, trajectory of a ball, flow of electric current, *etc.*. Still, some systems, while being fundamentally deterministic, are so complex that arrangements with almost identical initial conditions can evolve into very different states; such systems are chaotic.

An example that has an impact on us every day is the weather and, more broadly, the Earth's climate. The difficulty in predicting the weather is generally seen as a difficulty, but every cloud has a silver lining – in this case it is that the weather can be used as a RNG. This is actually

[1] At `https://news.uns.purdue.edu/html4ever/2005/050426.Fischbach.pi.html` .

exploited by the web-site `www.random.org`, which claims[2] to be a "true random number service" that uses "atmospheric fluctuations" to generate random numbers.

Other, simpler systems can also be utilised in this way, as is often done in lotteries. Despite the fact that the motion of ~ 40 balls in a barrel is a tractable problem in mechanics, most people are happy to accept that balls drawn in this fashion are effectively random. The reason is that the system is, once again, chaotic – the large number of collisions between the balls means that the identity of the ball that rolls out of the barrel cannot be predicted without unreasonably precise knowledge of the balls starting positions.

Such systems can, however be predicted to some degree, a fact that has occasionally been exploited. In the 1970s a group of students in California realised that the physics of roulette was sufficiently predictable that some measurements of the ball's motion on the wheel (at which time bets can still be placed) was sufficient to predict the quadrant of the wheel in which the ball would land, greatly stacking the odds in the gambler's favour. They actually implemented their scheme with, amongst other things, early microcomputers hidden in their shoes. This particular story is told in full in *The Newtonian Casino*, but the broader point is that it can be quite a serious issue if the outcome of a random number generator can be predicted even partially.

Question: *How could you generate random numbers from the objects in this room?*

1.3.3 Quantum mechanical systems

Probably the most fundamental source of random numbers is the physical world. Quantum mechanics, the theory which describes the behaviour of the microscopic world (and in particular the fundamental particles that make up atoms) is inherently stochastic. Even perfect knowledge of a system is insufficient to predict the result of a measurement or observation with certainty. For example, The half-life of an unstable isotope might be known with absolute precision, but that cannot be converted into a definite prediction about when a given atom will decay; it is only possible to predict the probability that it will decay in a given time interval. That does not mean the theory is untestable – the predictions for the distribution of decay times of a collection of unstable atoms can be easily verified. But it does mean that the physical world has a "truly" random aspect to it, in the sense that $\Pr(t_{\text{decay}}|\text{information})$ is never a delta function for any possible conditioning information. (This lack of predictivity has nothing to do with measurement errors or imprecsive models; it is, as far as it has been possible to tell, fundamental to the workings of the Universe.)

1.3.4 Pure mathematics

As alluded to above, random numbers seem to appear regularly in pure mathematics. Constants such as π, e and $\sqrt{2}$ seem, when written as a sequence of digits (in any base), to be random. While it has not been proved that this is the case, none of the many attempts to show that it is *not* the case have been decisive (or even offered hints that, *e.g.*, π is not random).

This deeper philosophical aspects of this – whether a constant that is defined by a simple formula can be "random" – are discussed above; the more pragmatic statement is that there is no known

[2] It pays to be skeptical here – the first five random numbers between 1 and 100 it provided were 14, 21, 36, 51, 95. What is the probability that a truly random process would produce an ascending sequence like this?

reason to distrust any numerical result based on using the digits of, *e.g.*, π as a source of random numbers.

That is not to say, however, that π is a good *practical* source of random numbers. For that to be true it would have to be possible to calculate sequences of digits from π with minimal computations, and to be able to start anywhere in the sequence. (Always having to start with $3, 1, 4, 1, 5, \ldots$, would mean either repeating calculations precisely or wasting time getting to a new starting point millions of digits in; this issue is discussed further below.)

1.3.5 Numerical algorithms

The most useful processes for generating random numbers are, of course, numerical algorithms that can be implemented on a computer, but the fact that (current) computers are deterministic generally causes some uneasiness about whether it is even *possible* for a computer to generate a random number. One apparently compelling argument is that the output of any computer program is completely determined by the algorithm and the inputs (or, indeed, from the state of the algorithm at any intermediate point). However useful such algorithms might be, their results can never "truly" random in the way that quantum mechanical processes can be. Instead the requirement must be more pragmatic, for instance that any number (mean of a distribution, *etc.*) estimated using a RNG should have the same distribution as if it had been evaluated using truly random sampling.

Many RNGs are based on using modular arithmetic, exploiting the fact that the remainder obtained when dividing a large number by a smaller number is difficult to predict. A very simple example of a RNG, implemented in `R`, based on this idea is:

```
# Random draw from a uniform distribution (bad implementation)

runif.bad <- function(n, seed = 1) {

  # Ensure the sequence starts with the seed (on first call only)
  # or the previous value.

  val <- attr(runif.bad, "valprev")
  if (is.null(val)) {
    val <- seed
  }

  # Psuedo-random calculation using remainders.

  vals <- rep(0, times = n)
  for (i in 1:length(vals)) {
    val <- (val * 32719 + 3) \%\% 32749
    vals[[i]] <- val / 32749
  }

  # Store the last value calculated to continue the sequence.

  attr(runif.bad, "valprev") <<- val
```

```
  return (vals)
}
```

Samples from the unit square obtained using `runif.bad` and `runif` (the native `R` RNG, described below) are shown in Fig. 3.

Question: *Why is* `runif.bad` *bad? How could you demonstrate that it was bad? But why does* `runif.bad` *produce numbers that could even remotely be considered random?*

1.3.6 Random number generation in `R`

Many powerful algorithms have been developed and several are implemented in the standard distribution of `R`. These – as implemented in `runif` – should be always be used as the basis for your stochastic calculations. The calling syntax is:

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

where `n` is the number of values to be calculated and `min` and `max` are the optional minimum and maximum values.

Hence

```
> vals <- runif(10000, 43.6, 237.4)
> print(vals[37])
[1] 130.182971
```

should produce a vector of 10000 values uniformly distributed between 43.6 and 237.4. The period of the `R` RNG is very high ($2^{19937} - 1$ in default mode) although still, of couse, finite. (If it were possible to very quickly calculate sequential digits of π then it would be possible to generate a non-period sequence of random numbers; however the computational cost of doing so is too great.)

Run as shown above, `runif` is repeatable: if a new `R` session is started (on the same computer, and possibly on different computers) the same sequence will be repeated. This can be useful – if some problem occurs then it is possible to reproduce the results even if there is a nominally random component to the algorithm – but it could also be a problem, as multiple independent runs could instead be exact repeats. This issue is dealt with by seeding the RNG, essentially choosing where to start in the long sequence. (If π was being used as a source of random numbers then the equivalent would be choosing the starting digit.) All common RNGs have the option of setting the seed, although the implementation can be very different. In `R` the random number generator is seeed by using `set.seed`, the calling syntax of which is:

```
set.seed(seed, kind = NULL, normal.kind = NULL)
```

where the important argument is the integer `seed`. An example of its use is:

```
> set.seed(43)
> runif(5)
[1] 0.48503768 0.91076604 0.05767389 0.70539841 0.31349510
> runif(5)
[1] 0.54303263 0.67910090 0.50670364 0.18297172 0.87225859
> set.seed(43)
> runif(5)
[1] 0.48503768 0.91076604 0.05767389 0.70539841 0.31349510
```

It is good practice to record the value of the seed used (*e.g.*, including it in the header of an output file) and doing this manually is easier than using R's enquiry functions to do so. The reason is that the seed is stored (in `.Random.seed`) as a long vector of integers. Hence:

```
> seed <- 43
> set.seed(seed)
> print(seed)
[1] 43
```

is tidier and easier to understand than:

```
> set.seed(43)
> seedvals <- .Random.seed
> runif(5)
[1] 0.48503768 0.91076604 0.05767389 0.70539841 0.31349510
> runif(5)
[1] 0.54303263 0.67910090 0.50670364 0.18297172 0.87225859
> .Random.seed <- seedvals
> runif(5)
[1] 0.48503768 0.91076604 0.05767389 0.70539841 0.31349510
```

It would be convenient to have the seed set randomly every time a program is run, so that the results are independent each time. This requires some external randomizing factor that is inherently non-computational. One option is to use information from the computer itself, such as the current time or the process number assigned to the program that is running. An implementation that combines both these pieces of information is:

```
> set.seed(as.integer((as.double(Sys.time()) * 1000 + Sys.getpid()) %% 2^31))
> runif[5]
[1] 0.5853403 0.6229182 0.4019968 0.8551930 0.1681481
> set.seed(as.integer((as.double(Sys.time()) * 1000 + Sys.getpid()) %% 2^31))
> runif[5]
[1] 0.7100214 0.8607342 0.6563417 0.3824806 0.5721210
```

For most practical purposes, `runif` can efficiently and reliably produce long sequences of random numbers (which themselves are sequences of random bits). Making a computer draw from uniform$(0, 1)$ is *hard* – and hard to explain in detail. You will not be expected to know in detail how this is done – the aims of this course are practical. Transforming the output from uniform$(0, 1)$ is (relatively) easy. So you will be expected to know, understand and be able to implement the methods for doing so, even though many (*e.g.*, `rnorm`, `rlnorm`, `rpois`, *etc.*.) are standard functions in R (and other languages/libraries).

1.4 Uses of random numbers

The ability to do something does not mean that it should be done, and none of the above discussion of random numbers gives any hint as to *why* anyone would want to generate random numbers. It turns out that a good RNG is an incredibly powerful tool, with applications in a number of fields:

- mathematics (**evaluation of integrals**; optimization problems; *etc.*.)
- statistics (**simulation**; **parameter estimation in Bayesian inference**; *etc.*)
- physical sciences (modelling of complex systems; testing; *etc.*.)
- crytography (generation of keys; *etc.*)
- finance/taxation (tax evation; *etc.*)
- gambling (on-line casinos; security testing; *etc.*)

Most of these applications are beyond the scope of this course, but a few are key topics; these are listed in bold above.

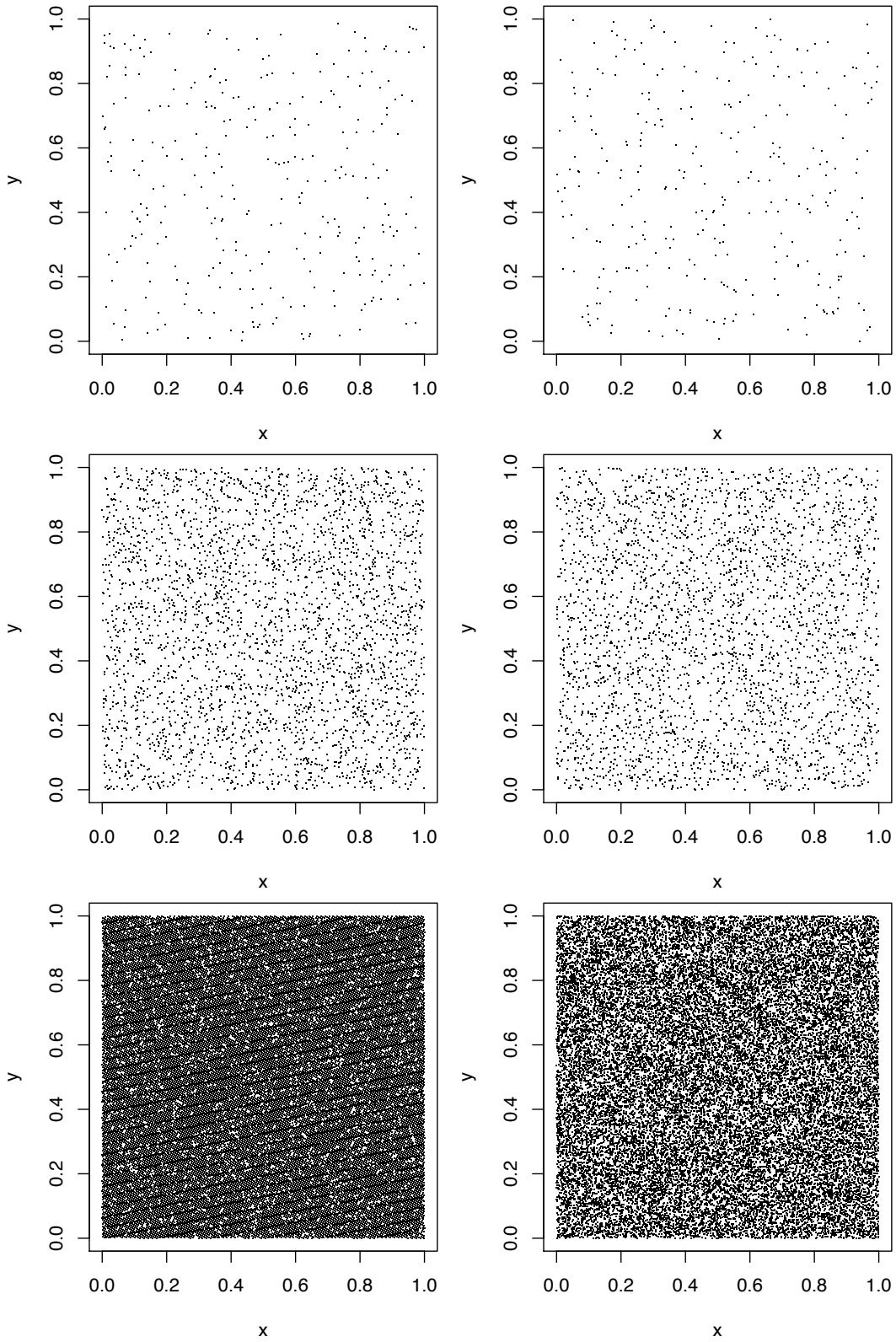

Figure 3: Points uniformly sampled from the unit square produced using `runif.bad` (left) and `runif` (right). The different rows show sample sizes of $N = 300$ (top row), $N = 3000$ (middle row) and $N = 30000$ (bottom row).

2 Simulation

2.1 Introduction

Simulation is the very general term for generating a realisation – or many realisations – of the output from a stochastic process. This can range from simply generating samples from some specified target density to modelling the formation of structure in the entire Universe from simple initial conditions. In the latter case the target distribution cannot be written down, and really cannot even be expressed in any useful form at all – except as simulated draws from a huge ensemble of possibilities.

In many ways a large set of (preferably uncorrelated) samples from a distribution is operationally equivalent to the distribution itself – anything that can be calculated from the former can be estimated from the latter. There is, of course, some error in any estimated quantity, but this can usually be reduced to the level of practical insignificance by obtaining a sufficiently high number of samples. In most cases arbitrary precision can be obtained in the limit of infinite sample size.

Moreover, it is only in a few simple cases (uniform distribution, normal distribution, *etc.*) that it is possible to obtain any useful analytic results, and these most often find their use in the toy problems used to illustrate basic principles in textbooks. In some cases it is reasonable to approximate real world processes with these standard distributions – analytic results are always to be preferred if available, not least because they can be verified – but it is not always sufficient to fall back on this option. Hence there is a great utility in being able to generate samples from a broad range of distributions, and there is a great need for generic algorithms not only to generate samples, but also to post-process the samples to calculate quantities of interest.

2.1.1 Post-processing samples

The basic output of all sampling algorithm is the same: a potentially correlated set of N_s samples $\{\boldsymbol{x}_s\} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_\mathrm{s}}\}$ with associated weights $\{W_s\} = \{W_1, W_2, \ldots, W_{N_\mathrm{s}}\}$, drawn from the (potentially unnormalised) target density $p(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_{N_\mathrm{p}})^\mathrm{T}$ is a point in the N_p-dimensional space over which the density is defined. Given these samples and weights, the distribution is effectively approximated in terms of Dirac delta functions as

$$\hat{p}(\boldsymbol{x}) = \frac{\sum_{s=1}^{N_\mathrm{s}} W_s \, \delta_\mathrm{D}(\boldsymbol{x} - \boldsymbol{x}_s)}{\sum_{s=1}^{N_\mathrm{s}} W_s}, \tag{1}$$

where the denominator can be ignored if the weights have been scaled to add to unity. While most distributions of interest are much smoother than the above sum of delta functions, any derived quantities of interest are inevitably obtained by some sort of averaging, either over a small region (in the case of binning) or globally (in the case of an integral quantity). It is really the values of these derived quantities that can be estimated to high accuracy by sampling the posterior distribution. The key mathematical point is that if $\boldsymbol{x}_i$ is drawn from the density $p(\boldsymbol{x})$, then

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{x}_i) = \int p(\boldsymbol{x}) \, f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \tag{2}$$

A list of samples $\{\boldsymbol{x}_s\}$ is almost never interpreted or presented directly – the human mind cannot make much sense of what is essentially just a huge lists of numbers. The closest thing to any direct presentation of the samples is a scatter plot, although even this is problematic if the samples have non-uniform weights, as it is difficult to reflect this fact graphically (and correlations are even more difficult to represent). Rather, the uitlity of $\{\boldsymbol{x}_s\}$ is that it can be used to estimate various derived quantities, such as means, covariances, credible intervals, *etc.*

Mean and (co)variance of a distribution

The natural estimate of the posterior mean of the p'th parameter of a distribution is

$$\hat{x}_{p,\mathrm{mean}} = \frac{\sum_{s=1}^{N_\mathrm{s}} W_s \, x_{p,s}}{\sum_{s=1}^{N_\mathrm{s}} W_s}, \tag{3}$$

which could also considered as an estimator of the expectation value, $\mathrm{Exp}(x_p)$. The corresponding covariance matrix has elements which can be estimated as

$$\hat{C}_{p,p'} = \frac{\sum_{s=1}^{N_\mathrm{s}} W_s \, (x_{p,s} - \hat{x}_{p,\mathrm{mean}})(x_{p',s} - \hat{x}_{p',\mathrm{mean}})}{\sum_{s=1}^{N_\mathrm{s}} W_s}. \tag{4}$$

Highest density credible regions

As the boundary of the highest density credible region depends in part on the probability density, it is necessary to bin the samples from the posterior in some way, which in turn implies that a range for each of the parameters must be chosen. A simple, if not completely robust, option is to use the posterior mean and variance calculated as described above to determine a (hopefully) inclusive range in each parameter. For the p'th parameter a reasonable range might be given by $x_{p,\mathrm{min}} = \hat{x}_{p,\mathrm{mean}} - 5\hat{C}_{p,p}^{1/2}$ and $x_{p,\mathrm{max}} = \hat{x}_{p,\mathrm{mean}} + 5\hat{C}_{p,p}^{1/2}$, where $\hat{x}_{p,\mathrm{mean}}$ and $\hat{C}_{p,p}^{1/2}$ are estimated as described above.

Ideally, binning would be avoided, and methods such as Delaunay triangulation have been used to convert a set of samples into a continuous density. The computational cost of such approaches are, however, considerable, and binning onto a simple one- or two-dimensional grid is the most common option.

Marginalisation

The marginal distribution of some subset of parameters can be estimated very simply from the samples $\{\boldsymbol{x}_s\}$ by simply truncating the components of each sample that are not of interest. It is

interesting that such a potentially difficult analytical problem of calculating a potentially multi-dimensional integral over a subspace of some distribution, can be achieved almost trivially with access to a large set of samples. The immediate implication is that sampling is the equivalent of integration, an idea that will be returned to several times subsequently.

The distribution of derived quantities

If some quantity, y, is a function $y(\boldsymbol{x})$, of the distribution's parameters, then its distribution can be approximated by

$$q(y) = \frac{\sum_{s=1}^{N_\mathrm{s}} W_s \, \delta_\mathrm{D}[y - y(\boldsymbol{x}_s)]}{\sum_{s=1}^{N_\mathrm{s}} W_s}. \tag{5}$$

Operationally, this means simply calculating $y_s = y(\boldsymbol{x}_s)$ for each of the N_s samples; these can then be used to calculate expectation values, *etc.*, using the techniques described above.

Density plot

Simply making a histogram of a set of samples gives a clear picture of the both the underlying density and the degree to which the sampling process makes this estimate imprecise. Other, more sophisticated options are possible (*e.g.*, using kernel density estimation), but they inevitably induce correlations and have an artificially smooth output, which can be misleading.

Correlations

Some simulation methods produce correlated samples from the target density. This is not necessarily a problem, as the chains can be post-processed correctly if the correlation structure is known; but the inconvenience and chance of misinterpretation can be considerable.

The standard tool for assessing the degree to which the elements of a chain are non-independent is the chain's auto-correlation function. For a one-dimensional chain (or a single parameter of a multi-dimensional chain) the autocorrelation is defined for lags of $\Delta = 0, 1, 2, \ldots N-1$ as

$$\hat{C}_\Delta = \frac{1}{N-\Delta} \sum_{i=1}^{N-\Delta} \frac{(x_i - \hat{x}_\mathrm{mean})(x_{i+\Delta} - \hat{x}_\mathrm{mean})}{\hat{C}^2}, \tag{6}$$

where $\hat{\mu}$ is the empirical sample mean, evaluated as in Eq. (**??**) and $\hat{C}$ is the empirical sample variance, evaluated as in Eq. (4) and here it is assumed that the samples are equally weighted. Clearly $\hat{C}_0 = 1$ by construction, regardless of the correlation structure of the chain; an uncorrelated chain would have $\hat{C}_\Delta = 0$ on average for all $\Delta \geq 1$. For most non-trivial sampling algorithms (most obviously MCMC), $\hat{C}_\Delta$ will drop from unity to zero for increasing Δ. The worst case scenario is that $\hat{C}_\Delta$ is significantly non-zero even as $C \to N-1$, implying that all the samples are highly correlated.

It is possible to obtain (approximately) independent samples by finding a threshold Δ_min above which $\hat{C}_\Delta \simeq 0$ and producing a new, shorter chain in which only every Δ_min'th element is retained. This process is known as thinning.

Thinned chains are particularly useful for producing diagnostic plots, such as two-parameter scatter plots of the chain, as any such plots with inter-sample correlations are liable to misinterpretation.

Effective sample size

A correlated set of samples clearly contains less useful information about the generating distribution than an uncorrelated sample of the same size. In the extreme case that all the samples are identical even an infinite sample contains no more information than just its first value. It is clearly important to be able to quantify this idea to assess the utility of a correlated sample.

One approach is to characterise a sample of N values by its "effective sample size", N_{eff}, defined to be the size of an uncorrelated sample that encodes the same information. Several definitions of N_{eff} are in use, but a common and useful choice is to define

$$N_{\text{eff}} = \frac{N}{1 + 2\sum_{\Delta=1}^{\Delta_{\max}-1} C_\Delta} \tag{7}$$

where C_Δ is the empirical auto-correlation defined in Eq. (6) above and $\Delta_{\max}$ is the lowest value of Δ for which $C_\Delta < 0$. This is motivated by the fact that, as shown by **?**, the variance of any derived estimator scales with the denominator of Eq. (7). The reason for the truncation of the sum is to avoid the noisy high-lag contributions that yield spurious negative values of C_Δ, although this is only a heuristic recipe.

Thinning

Correlated chains can be used "as is", but it is often desireable to post-process a set of samples to produce an (almost) uncorrelated chain by discarding some elements, a process known as "thinning". Even though this necessarily involves a loss of information, it is often worthwhile because the resultant chains have such simple statistical properties.

The degree of thinning required to obtain a reasonably uncorrelated sample can be judged from auto-correlation structure of a chain as desribed above. A heuristic approach is to identify a lag scale, $\Delta_{\max}$, above which the correlations are sufficiently small to be acceptable, and then to keep one element in $\Delta_{\max}$. The result is a considerably reduced chain of length $\sim N/\Delta_{\max}$ but that is largely uncorrelated, and so has an effective sample size of $\sim N/\Delta_{\max}$ as well.

Thinned chains are particularly useful for producing diagnostic plots, such as two-parameter scatter plots of the chain, as any such plots with inter-sample correlations are liable to misinterpretation.

2.2 Simulation methods

In the context of this course, simulation is mainly concerned with taking the output of a RNG (that produces random bits or, more commonly, large random integrers) and producing samples drawn from a non-uniform target distribution given only its density. In some cases (*e.g.*, the inverse transform method discussed in Section 2.2.2) this is just a simple transformation that can be applied directly to the outputs of the RNG; in more complex cases (Section 2.2.3 and Section 2.2.4) some more exploratory techniques (*i.e.*, involving some "wasted" random numbers) are needed because the target distribution does not have convenient analytical properities. In the most extreme cases, in which the broad properties of the target distribution (*e.g.*, its mean or variance) are not known a priori then the distribution must be explored and characterised as well as sampled from; this is a sufficiently different process that it is treated separately.

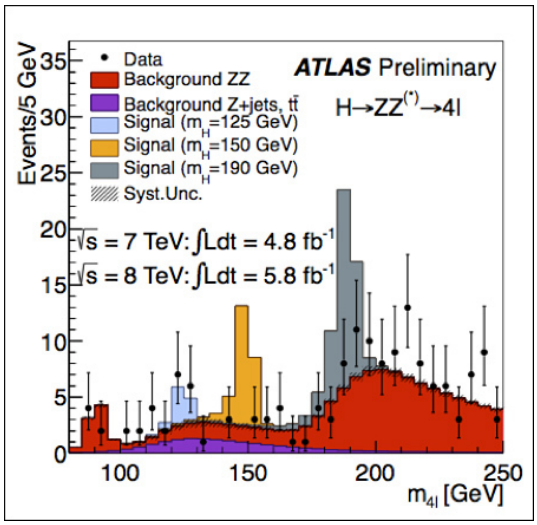

Figure 4: The distribution of collision energies measured by the ATLAS detector at the LHC compared to predictions from highly realistic simulations. Predictions are shown for several different Higgs boson masses, one of which corresponds well with the data.

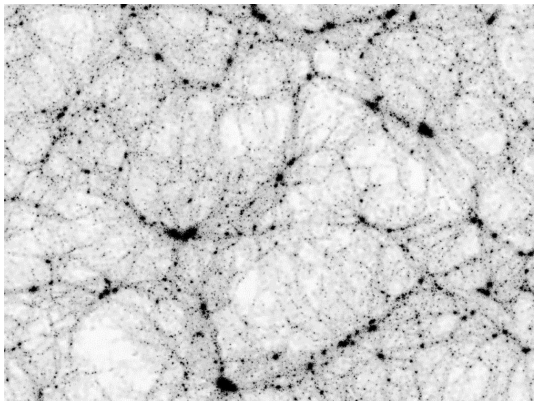

Figure 5: The distribution of dark matter in simulated clusters of galaxies. The plot is close to one billion light years across (or, at least, the region simulated is).

Another distinct aspect of simulation is that it is possible to generate samples from distributions even without access to the density. These are often cases in which the notion of "simulation" is very clearly linked to the more colloquial use of the word outside statistics. Examples include:

- Fig. 4 compares the results from the ATLAS detector at CERN's Large Hadron Collider (LHC) to simulations of the expected signal for different putative Higgs boson masses. These simulations, which were vital to the recent announcement that the Higgs boson had been discovered, were incredibly complicated, including models of sub-atomic particle interactions, detector efficiencies, background signals, *etc.*, but technically the output was "just" a set of samples from the fairly benign looking distribution shown by the histogram in the figure.

- Fig. 5 shows the distribution of dark matter obtained by integrating the coupled differential equations that describe the motion of billions of particles subject to each others' gravitational pull. Even though this might seem unrelated to statistics, the initial con-

ditions for these simulations are very simple, and the final result is, effectively, a single sample from a highly complex billion-dimensional distribution.

Question: *What is the distribution,* $\Pr(s|N)$ *of nearest neighbour distances,* s*, between* N *points uniformly distributed on the unit square? What about if there is an additional requirement that no two points can be within a separation* $s_{\min}$ *of each other?*

Question: *What is the distribution,* $\Pr(x_{\text{mean}}|2N+1)$*, of the mean,* x_{mean}*, of* $2N+1$ *points* $x_i \sim \text{normal}(0, 1^2)$*? What is the distribution,* $\Pr(x_{\text{median}}|2N+1)$ *of the median,* x_{median}*, of* $2N+1$ *points* $x_i \sim \text{normal}(0, 1^2)$*? Assuming these values are noisy measurements of some quantity, which is the better estimator for true value,* x*? What if the measurements had instead been drawn from a student t-distribution of* $N_{\text{dof}} = 5$*?*

2.2.1 Transformation methods

Given a set of samples $\{x_s\}$ drawn from a density $p(x)$, what distribution will be sampled by applying the strictly monotonic (and hence bijective) transformation $y_s = y(x_s)$? A sample between x and $x + \mathrm{d}x$ would be mapped into the range between $y(x)$ and $y(x + \mathrm{d}x) = y(x) + \mathrm{d}y/\mathrm{d}x\,\mathrm{d}x$, so the transformed density is given by the usual change of variables,

$$q(y) = \left|\frac{\mathrm{d}x}{\mathrm{d}y}\right| p[x(y)]. \tag{8}$$

This result is particularly useful if the starting point is the uniform distribution, which can be written as

$$p(u) = \text{uniform}(u; 0, 1) = \Pr(u|\text{uniform}, 0, 1) = \Theta(u)\,\Theta(1-u), \tag{9}$$

where $\Theta(u)$ is the Heaviside step function. Applying the transformation $x(u)$ then gives

$$q(x) = \left|\frac{\mathrm{d}u}{\mathrm{d}x}\right| \Theta[u(x)]\,\Theta[1-u(x)]. \tag{10}$$

Question: *Given a RNG that can produce samples from* $\text{uniform}(0, 1)$*, how would you generate a sample from* $\text{uniform}(a, b)$*? How does your answer relate to the above transformation?*

2.2.2 The inverse transform method

The inverse transform method is the most efficient and precise option for sampling from a one-dimensional distribution; however it requires the target distribution be i) analytically (or easily numerically) integrable and ii) that the cumulative distribution can be analytically (or numerically) inverted. If both of these criteria are met then it is possible to make draw from the target density by simply transforming a single output from `runif`.

Given a one-dimensional target density $p(x)$, with associated cumulative distribution $P(x) = \int_{-\infty}^{x} p(x')\,\mathrm{d}x'$, which can be inverted to give $x(P)$, a random sample from $p(x)$ is given by:

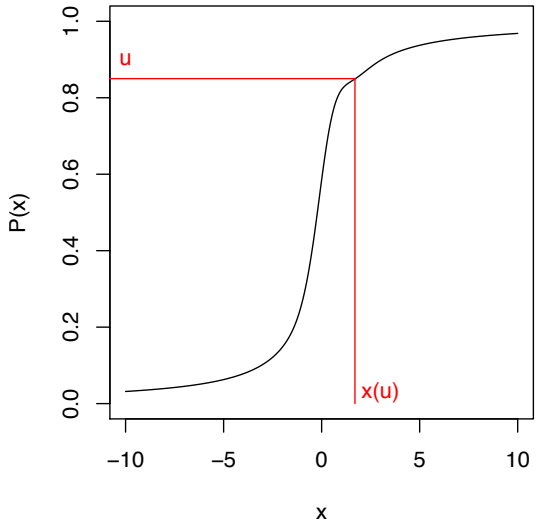

Figure 6: Illustration of the inverse transform method.

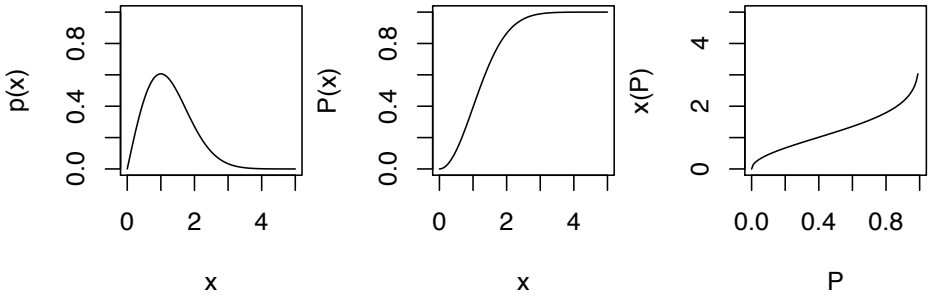

Figure 7: Illustration of the inverse transform method for a Rayleigh distribution.

1. Draw $u \sim \text{uniform}(0, 1)$.
2. Calculate $x(P)$ for $P = u$.

This algorithm can seem somewhat counter-inituitive, but this method is justified by the transformation argument above. Given that transforming u to $x(u)$ gives draws from a distribution that is proportional to $|\mathrm{d}u/\mathrm{d}x|$, this implies identifying $p(x)$ with $|\mathrm{d}u/\mathrm{d}x|$ and hence $P(x)$ with $u(x)$.

The inverse transform algorithm is very simple; the difficulty is in calculating $x(P)$. Still, this can be done for a number of common distributions, and so the inverse transform method is an important lower rung in the hierarchy of sampling methods that can be deployed on problems of practical interest.

Question: *Use the inverse transform method to generate samples from the exponential distribution with density* $p(x) = \Theta(x)/\beta \exp(-x/\beta)$*?*

Question: *Use the inverse transform method (as illustrated in Fig. 7 to generate samples from the Rayleigh distribution with density* $p(r) = \text{Rayleigh}(\sigma^2) = \Theta(r)\, r/\sigma^2 \exp[-r^2/(2\sigma^2)]$*. This is*

the distribution of the magnitude of a two-dimensional vector comprised of components x and y which are drawn from normal$(0, \sigma^2)$.

2.2.3 The grid method

The grid method is useful if and only if the target density is expensive to evaluate – if not then other methods should be used, as it is neither particular accurate nor particularly fast. Its only virtue is that only a small number of evaluations of the target density are needed.

The algorithm for the grid method to obtain a sample from (an approximation to) the one-dimensional density $p(x)$ is:

1. Identify the range from $x_{\min}$ to $x_{\max}$ over which most of the mass of $p(x)$ is distributed.
2. Divide the range into N bins of width $\Delta x = (x_{\max} - x_{\min})/N$ and centred at $x_i = x_{\min} + (i + 1/2)\,\Delta x$.
3. Draw i from $\{1, 2, \ldots, N\}$ with weights $W_i = p(x_i)$ using the multinomial method (see below).
4. Draw x from uniform$(x_i - \Delta x/2, x_i + \Delta x/2)$.

Question: *How should the initial sampling range be chosen? What fraction of the probability should lie in this range?*

Question: *Why are values generated in this way* not *drawn from $p(x)$? What density are they drawn from?*

Multinomial sampling

A basic workhorse routine that underlies a number of aspects of simulation is to draw an integer, i, from a set of N weights $\{W_i\}$ (all of which must be non-negative, and at least one of which must be positive). A basic algorithm for doing so is straightforward, although it can easily be rendered slow if N is large.

There is an efficient implementation of multinomial sampling in `R`, available through the `sample` function:

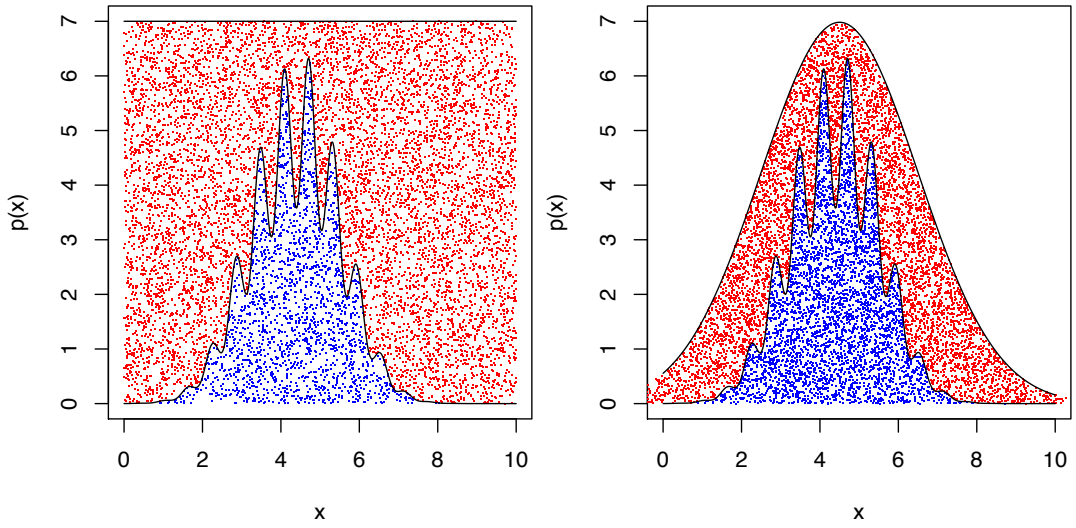

Figure 8: Illustration of the rejection method using a uniform envelope function (left) and a more efficient normal envelope function (right).

```
> sample.int(n, size = n, replace = FALSE, prob = NULL)
```

As the syntax – particularly the two instances of `n` – is a little counter-intuitive, its use is best illustrated by example:

```
> weights <- c(0.1, 0.9, 0.4, 0.1)
> nsmp <- 30
> sample.int(length(weights), size = nsmp, replace = TRUE, prob = weights)
[1] 2 3 1 3 3 2 2 3 2 2 2 1 4 2 2 2 2 2 2 2 2 2 1 2 3 4 2 3 1 2
```

Multinomial sampling is particularly useful in sampling from mixture models.

2.2.4 Rejection sampling

Rejection sampling is, in its simplest form, a basic method of generating independent samples from distributions of compact support and low dimensionality, but it can be extended to broader classes of distributions if sufficient information about them is available.

Basic rejection sampling

The basic algorithm for generating a single sample from the target density $p(x)$ which has support only in the range $a \leq x \leq b$ and has a peak density of $p_{\max}$ is:

1. Draw x_{trial} from uniform(a, b).
2. Draw p_{trial} from uniform$(0, p_{\max})$.
3. If $p_{\text{trial}} \leq p(x_{\text{trial}})$ then set $x = x_{\text{trial}}$; otherwise return to Step 1.

This algorithm is illustrated in Fig. 8. This method can only be used (at least in this form) if both a and b are finite and if $p_{\max}$ is known.

Question: *Name some common distributions which do not satisfy all of the above requirements.*

Even if the above criteria are satisfied, rejection sampling is efficient only for a small class of distributions which have reasonably uniform support over the relevant range.

Rejection sampling with an envelope function

Rejection sampling would not be of much practical use if subject to the limitations described above; fortunately, the basic algorithm can be extended to be used to efficiently generate samples from densities which are non-zero everywhere and/or are sharply-peaked. The key change is to adopt an envelope function which is more closely matched to the target density than the uniform distribution implicitly used above. The envelope function must be related to a normalised density $p_{\text{envelope}}(x)$, but multiplied by some constant $C > 1$ so that $C\, p_{\text{envelope}}(x) \geq p(x)$ for all x.

The algorithm is very similar to the basic rejection sampling algorithm above:

1. Draw x_{trial} from $p_{\text{envelope}}(x)$.
2. Draw p_{trial} from uniform$[0, Cp_{\text{envelope}}(x_{\text{trial}})]$.
3. If $p_{\text{trial}} \leq p(x_{\text{trial}})$ then set $x = x_{\text{trial}}$; otherwise return to Step 1.

Question: *How can basic rejection sampling be recast in terms of an envelope function?*

The "art" here is finding an envelope function that can easily be sampled from and, when suitably scaled, closely follows the target density.

2.3 Common distributions

A clear clear heirarchy of distributions has developed within computational statistics, with the uniform distribution at the bottom and essentially arbitrary, problem-specific distributions at the top. In between are the "common" distributions, such as normal, exponential, Poisson, *etc.*, although their defining characteristic is not so much that they are genuinely common in the real world, but that they are mathematically tractable, with analytic densities, cumulative distributions, quantiles, *etc.*. Even the most complicated of simulations (*e.g.*, those used at the Large Hadron Collider to generate mock data-sets under an assumed particle physics model) are based on the ability to sample from these common distributions. Sampling algorithms for some (*e.g.*, the normal distribution) have been described above; some other cases are included here; and algorithms for sampling from many more are decribed elsewhere (and implemented wll in `R`).

2.3.1 The normal distribution

The single most important distribution in statistics is the normal distribution. (Why?) An immediate corrollary is that it vital to be able to draw independent samples from the normal

distribution efficiently. Ideally, it would be possible to use the inverse transform method, but the density

$$p(x) = \text{normal}(x; \mu, \sigma^2) = \frac{1}{(2\pi)^{1/2}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{11}$$

is only integrable in terms of the error function as

$$P(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{2^{1/2}\sigma}\right)\right]. \tag{12}$$

This is not expressable in terms of elementary functions and is not analytically invertible.

Fortunately, a "trick" makes it possible to sample effectively from $p(x) = \text{normal}(x; \mu, \sigma^2)$. The starting point is to consider two quantities, x and y, both of which are normally distributed. Treating these as Cartesian coordinates, the quantity $r = (x^2 + y^2)^{1/2}$ is then distributed according to the Rayleigh distribution considered above and $\phi = \arctan(y, x)$ (defined to behave like `atan2` in `R`) is uniformly distributed between 0 and 2π. Hence an algorithm for efficiently drawing a sample from $\text{normal}(0, 1^2)$ is:

1. Draw r from the unit Rayleigh distribution, $\text{Rayleigh}(1^2)$.
2. Draw ϕ from $\text{uniform}(0, 2\pi)$.
3. Calculate $x = r\cos(\phi)$ and $y = r\sin(\phi)$, both of which are *independent* random draws from $\text{normal}(0, 1^2)$.

Condensing the above leads to the form known as the Box Muller transform, which is more compact, if less clear. Drawing u_1 and u_2 independently from $\text{uniform}(0, 1)$, the quantity

$$x = \mu + \sigma[-2\ln(u_1)]^{1/2}\cos(2\pi u_2) \tag{13}$$

is drawn from $\text{normal}(\mu, \sigma^2)$.

Multivariate normal distribution

To generate a sample $\boldsymbol{x}$ from a multivariate normal distribution with density $p(\boldsymbol{x}) = \text{normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is the $N \times N$ covariance matrix:

1. Generate a vector $\boldsymbol{y}$ by drawing N components $y_i \sim \text{normal}(0, 1)$.
2. Apply the linear transform $\boldsymbol{x} = \boldsymbol{\mu} + \mathsf{L}\boldsymbol{y}$, where L is any matrix such that $\mathsf{L}\mathsf{L}^{\mathrm{T}} = \boldsymbol{\Sigma}$. (A simple option is the Cholesky decomposition, so that L is lower triangular.)

Question: *Why is the distribution the same even for different choices of* L*?*

Question: *What are the expectation values* $\langle \boldsymbol{x} \rangle$ *and* $\langle (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}} \rangle$*?*

2.3.2 The Poisson distribution

The Poisson distribution,

$$\Pr(n|\lambda) = \text{Poisson}(n;\lambda) = \Theta(n)\,\frac{\lambda^n\,\exp(-\lambda)}{n!}, \tag{14}$$

gives the probability of n events occurring in an interval, given a rate λ. The Poisson distribution has wide applicability because so many real world processes can be reasonably well modelled as independent events that occur with a known (or unknown) frequency.

Question: *What real world processes can be reasonably well described by the Poisson distribution? In each case try to come up with reasons why it is not perfectly/absolutely Poisson.*

In practice, the Poisson distribution can be sampled using the `rpois` function in `R`:

```
> rpois(n, lambda)
```

But the Poisson distribution is a particularly useful case to illustrate how the different sampling techniques described above can be deployed for different λ ranges and combined into an accurate and efficient algorithm for sampling from a Poisson distribution.

Question: *Use the fact that the periods, T, between successive Poisson events are distributed as* $\Pr(T|\lambda) = \Theta(T)\,\lambda\exp(-\lambda T)$ *to write a routine to sample from* $\text{Poisson}(n;\lambda)$.

Question: *In the high-λ limit the Poisson probability becomes* $\Pr(n|\lambda) \propto \exp[-1/2(n-\lambda)^2/\lambda]$. *Use rejection sampling to sample from the Poisson distribution in this limit.*

3 Iterative sampling methods

3.1 Introduction

While the simulation methods described in Section 2 were quite different from each other – the most efficient (*e.g.*, the inverse CDF method) used extensive information about the target distribution to provide a highly-efficient simulation algorithm, whereas others (*e.g.*, basic rejection sampling) assumed very little about the target distribution but were correspondingly slow – the fact that all these sampling approaches generated independent samples from the target distribution immediately implies that the first samples drawn (or, equivalently, the first density evaluations) were not used to refine the algorithm in any way. Independent samples are, of course, desireable from a statistical point of view, but in general it is impossible to sample an arbitrary distribution (or, equivalently, a multi-dimensional distribution with unknown mean, covariance and correlation structure) without using some sort of sampling technique that "learns" about the distribution and adjusts its sampling scheme accordingly. Such iterative techniques are the subject of this section.

All the algorithms described below produce samples from an arbitrary multi-dimensional target density $p(\boldsymbol{x})$ given only the ability to i) evaluate $p(\boldsymbol{x})$ and ii) the ability to generate random numbers. The common feature of these algorithms is that generate a sequence of samples in which, at each stage, some of the information obtained in generating the previous samples is used to determine what region of the parameter space to explore subsequently. The result is that these algorithms, while very powerful and general, produce correlated samples of the target density that are correlated, non-uniformly weighted or both.

Question: *How would you explore a density given only the ability to evaluate the density at any position? (A useful conceit is to imagine that you are trying to explore a mountain range in the fog – you cannot see the overall landscape, but can take altitude readings.)*

3.2 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) algorithms are a particular popular and commonly-used family of sampling methods that are almost ubiquitous in, particularly, Bayesian parameter estimation. The most basic MCMC algorithms are incredibly simple; and, in principle, even a basic MCMC algorithm can generate samples from an arbitrary distribution, albeit only in the limit of an infinite number of density evaluations. In practice, MCMC methods are not, of course, a panacea: great care must be taken to check any results obtained from a finite set of samples generated using the most sophisticated of MCMC methods.

3.2.1 Markov chains

A Markov chain is an *ordered* sequence of points $\{\boldsymbol{x}_s\} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_\mathrm{s}}\}$, in which the position of the s'th point, $\boldsymbol{x}_s$, depends explicitly only on $\boldsymbol{x}_{s-1}$. The s'th sample hence is drawn from some distribution of the from $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$, which is independent of $\boldsymbol{x}_{s-2}, \boldsymbol{x}_{s-3}, \ldots$ (and, in principle, also independent of $\boldsymbol{x}_{s-1}$, although in this case the ordering is arbitrary and so the sample is not really a chain; none of the MCMC processes considered here have this independence). As such, Markov chains cannot be the optimal way to explore a distribution, as almost all of the information obtained from the previous evaluations is disregarded – a hypothetical non-Markovian algorithm which used the full set of previous evaluations could be considerably more

efficient. However, the fact that most elements of a Markov chain are not directly dependent means that it is easy to understand the stochastic properties of the process and, in particular, to choose $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$ so that the limiting distribution of the chain is the target density.

Algorithm

There is no general prescription for how to generate or choose the first sample, $\boldsymbol{x}_1$, but if an MCMC algorithm is designed to explore a target distribution then any results calculated from a sufficiently large set of samples should be independent of the starting point. In practice the starting point is often chosen at random, with multiple, independent chains run to check that the final results are indeed robust.

Once the Markov process has been started, the next element of the chain, $\boldsymbol{x}_s$, is generated from the previous point, $\boldsymbol{x}_{s-1}$, by the following procedure:

1. Draw a trial point from the (as yet unspecified) proposal distribution $\Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1}, {\rm trial})$, the form of which can be chosen to increase the efficiency of the algorithm.
2. Accept the trial point with the (as yet unspecified) probability $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1})$; for some MCMC algorithms this is always unity (*i.e.*, the trial point is always accepted).
3. If the trial point is accepted then set $\boldsymbol{x}_s = \boldsymbol{x}_{\rm trial}$; otherwise set $\boldsymbol{x}_s = \boldsymbol{x}_{s-1}$.

This process is repeated (at least $\sim 10^3$ times, and as many as $\sim 10^8$ times in some applications), building up a large set of samples. Both $\Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1}, {\rm trial})$ and $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1})$ remain general at this stage; the inter-related choice of these two functions defines the particular MCMC algorithm.

This two-step process can also be recast in terms of a single distribution, $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$, which is given in terms of $\Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1}, {\rm trial})$ and $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1})$ by marginalising over $\boldsymbol{x}_{\rm trial}$. From the above algorithm the conditional distribution of $\boldsymbol{x}_s$ is

$$\begin{aligned} \Pr(\boldsymbol{x}_s|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1}) &= \Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1})\, \delta_{\rm D}(\boldsymbol{x}_s - \boldsymbol{x}_{\rm trial}) \\ &\quad + [1 - \Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1})]\, \delta_{\rm D}(\boldsymbol{x}_s - \boldsymbol{x}_{s-1}). \end{aligned} \tag{15}$$

Marginalising over the trial position then gives

$$\begin{aligned} \Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1}) &= \int \Pr(\boldsymbol{x}'_{\rm trial}|\boldsymbol{x}_{s-1}, {\rm trial})\, \Pr(\boldsymbol{x}_s|\boldsymbol{x}'_{\rm trial}, \boldsymbol{x}_{s-1})\, {\rm d}\boldsymbol{x}'_{\rm trial} \\ &= \Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1}, {\rm trial})\, \Pr({\rm accept}|\boldsymbol{x}_s, \boldsymbol{x}_{s-1}) \\ &\quad + \delta_{\rm D}(\boldsymbol{x}_s - \boldsymbol{x}_{s-1}) \left[1 - \int \Pr(\boldsymbol{x}'|\boldsymbol{x}_{s-1}, {\rm trial})\, \Pr({\rm accept}|\boldsymbol{x}', \boldsymbol{x}_{s-1})\, {\rm d}\boldsymbol{x}' \right], \end{aligned} \tag{16}$$

where in the second line the integration parameter has been changed from $\boldsymbol{x}'_{\rm trial}$ to $\boldsymbol{x}'$ to emphasize the shift away from the idea of a trial point. If the trial point is always accepted then $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}_{s-1}) = 1$ and so Eq. (16) simplifies to

$$\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1}) = \Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1}, {\rm trial}). \tag{17}$$

As defined to this point, $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$ has no explicit relationship with the target density, $p(\boldsymbol{x})$, and a general Markov process would *not*, in general, produce samples from $p(\boldsymbol{x})$. Several requirements must be met to ensure that this is the case.

Stationary distribution

The main requirement for MCMC to be useful is that the samples are, at least in the limit of high s, draws from $p(\boldsymbol{x})$. If the $(s-1)$'th element was drawn from some density $p(\boldsymbol{x})$ then the s'th element will, from Eq. (16), be drawn from the distribution

$$\begin{aligned}
\Pr(\boldsymbol{x}_s) &= \int p(\boldsymbol{x}'_{s-1}) \Pr(\boldsymbol{x}_s|\boldsymbol{x}'_{s-1}) \, \mathrm{d}\boldsymbol{x}'_{s-1} \\
&= \\
&= \\
&= \\
&= p(\boldsymbol{x}_s) + \int \left[p(\boldsymbol{x}') \Pr(\boldsymbol{x}_s|\boldsymbol{x}', \text{trial}) \Pr(\text{accept}|\boldsymbol{x}_s, \boldsymbol{x}') \right. \\
&\qquad \left. - p(\boldsymbol{x}_s) \Pr(\boldsymbol{x}'|\boldsymbol{x}_s, \text{trial}) \Pr(\text{accept}|\boldsymbol{x}', \boldsymbol{x}_s) \right] \, \mathrm{d}\boldsymbol{x}'.
\end{aligned} \tag{18}$$

Hence the s'th point will be drawn from the target density (and from the same density as the s'th point) if the above integrand is zero. This criterion will be satisfied if, for any pair of points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$,

$$p(\boldsymbol{x}_1) \Pr(\boldsymbol{x}_2|\boldsymbol{x}_1, \text{trial}) \Pr(\text{accept}|\boldsymbol{x}_1, \boldsymbol{x}_2) = p(\boldsymbol{x}_2) \Pr(\boldsymbol{x}_1|\boldsymbol{x}_2, \text{trial}) \Pr(\text{accept}|\boldsymbol{x}_2, \boldsymbol{x}_1). \tag{19}$$

Rearranging Eq. (19) implies that the proposal and acceptance probabilities must satisfy

$$\frac{\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1, \text{trial}) \Pr(\text{accept}|\boldsymbol{x}_1, \boldsymbol{x}_2)}{\Pr(\boldsymbol{x}_1|\boldsymbol{x}_2, \text{trial}) \Pr(\text{accept}|\boldsymbol{x}_2, \boldsymbol{x}_1)} = \frac{p(\boldsymbol{x}_2)}{p(\boldsymbol{x}_1)}, \tag{20}$$

which is known as "detailed balance". Note that, as this relationship is defined in terms of the density ratio between different points, there is no need for the target density to be normalised, which is often a critical practical consideration.

There are several distinct ways in which detailed balance can be satisfied, although not all options are equally useful:

- If the proposal distribution is taken to be independent of the target density, so that $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1, \text{trial}) = \Pr(\boldsymbol{x}_1|\boldsymbol{x}_2, \text{trial})$, then the acceptance probability must include the necessary dependence on $p(\boldsymbol{x})$; this approach is used in the Metropolis algorithm (Section 3.2.2).
- For an MCMC algorithm in which moves are always accepted, $\Pr(\text{accept}|\boldsymbol{x}_1, \boldsymbol{x}_2) = 1$ and the detailed balance criterion reduces to

$$\frac{\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1, \text{trial})}{\Pr(\boldsymbol{x}_1|\boldsymbol{x}_2, \text{trial})} = \frac{p(\boldsymbol{x}_2)}{p(\boldsymbol{x}_1)}.$$

Figure 9: Example of a combination of target and proposal distribution that would not result in the former being correctly sampled.

Figure 10: Trace plot showing how a one-parameter chain moves towards the peak of the target density.

One solution to this is to have $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1, \text{trial}) = \Pr(\boldsymbol{x}_2|\text{trial}) \propto p(\boldsymbol{x}_2)$, which is true of Gibbs sampling (Section 3.2.3).

Exhaustiveness

Detailed balance ensures that pairs of points are sampled with the correct relative frequency, but there is also the global desideratum that the chain must be capable of exploring the entire parameter space.

This requirement is certainly satisfied if $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$ is non-zero for all $\boldsymbol{x}_s$, whereas if, *e.g.*, $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1}) \propto \Theta(r - |\boldsymbol{x}_s - \boldsymbol{x}_{s-1}|)$ then there would be no way for a chain to move between different high-density regions separated by a distance of more than $2r$ and the chain would not be exhaustive, as illustrated in Fig. 9. Another problem can arise if $\Pr(\boldsymbol{x}_s|\boldsymbol{x}_{s-1})$ is periodic in structure – while there might be no bound to the region that could be sampled, certain regions could never be explored.

These considerations are critical mathematically, and must be satisfied for the process to be capable in principle of exploring the full distribution. However, they are rarely important in practice.

Burn-in

If the target distribution is known to be unimodal and the approximate location of the peak is also known then it is perfectly legitimate – and, indeed, preferable – to start sampling from that location in parameter space [*i.e.*, if $p(\boldsymbol{x})$ is known to be unimodal with its peak close to $\hat{\boldsymbol{x}}$ then choosing $\boldsymbol{x}_1 \simeq \hat{\boldsymbol{x}}$ guarantees that this first sample is already a plausible draw from the target distribution]. If the target density is known (how?) to be unimodal then repeat chains can be started from the same position; if it is multi-modal then the problem is considerably more difficult.

Unfortunately, it is in general not known where the target density is appreciable, but one of the key attributes of most MCMC algorithms is that they can usually be run with an arbitrary starting point. The resultant chain will then tend to propagate towards the peak of the density, a process known as "burn-in". The only problem is that these first points can be in regions of parameter space with arbitrarily small densities that would most likely not have been sampled in a chain of plausible length.

The crude, if effective, solution to this problem is simply to remove these first elements from the chain, which can be done after sampling is completed. No general rigorous algorithms for doing this exist, but heuristic options abound. A simpe guide is that any sample which has an appreciable probability, relative to the peak probability sampled, would have been a plausible first sample were it possible to draw directly from the target distribution. Under the assumption that the chain has eventually reached the region(s) of high probability the samples that are generated after it has first reached this high density region are acceptable.

One useful visual diagnostic is a trace plot, in which the value of one parameter is plotted against sample number, as shown in Fig. 10.

Convergence tests

MCMC algorithms could be run indefinitely – there is no universal stopping criterion. In most cases the errors on any estimates (*e.g.*, of means, covariances, *etc.*) vary with the number of samples as ${N_\mathrm{s}}^{-1/2}$, although the scaling of this convergence depends on the degree of correlation in the chain (and hence on both the precise sampling algorithm and the nature of the target distribution).

One key idea is that multiple independent chains should converge on the same stationary distribution, and so any quantities derived from different chains should be consistent with being drawn from the same distribution. If multiple chains exhibit significant differences then it is likely that the target distribution has not been sampled well; but passing such a test is *not* a guarantee that the sampling has been effective.

Gelman, A. and Rubin, D. B. (1992) devised a heuristic convergence test based on this principle that just uses the sample means and variances of the marginal distributions of independent chains (that have been pruned of pre-burn-in elements). The test can be applied to each parameter independently, so without loss of generality a single-parameter model is considered here for simplicity. Assuming that N_c chains of (equal) length N_s, with elements $\{x_{c,s}\}$ (where $c \in \{1, 2, \ldots, N_\mathrm{c}\}$ and $s \in \{1, 2, \ldots, N_\mathrm{s}\}$), have been generated, the test is peformed by:

1. Calculate the mean of each chain:

$$\bar{x}_c = \frac{1}{N_\mathrm{s}} \sum_{s=1}^{N_\mathrm{s}} x_{c,s}.$$

2. Calculate the variance of each chain:

$$\sigma_c^2 = \frac{1}{N_\mathrm{s}-1}\sum_{s=1}^{N_\mathrm{s}}(x_{c,s}-\bar{x}_c)^2.$$

3. Calculate the mean of all the chains (*i.e.*, the best combined estimate for the mean of the distribution):

$$\bar{x} = \frac{1}{N_\mathrm{c}}\sum_{c=1}^{N_\mathrm{c}}\frac{1}{N_\mathrm{s}}\sum_{s=1}^{N_\mathrm{s}}x_{c,s} = \frac{1}{N_\mathrm{c}}\sum_{c=1}^{N_\mathrm{c}}\bar{x}_c.$$

4. Calculate the average of the individual chains' variances:

$$\sigma^2_\mathrm{chain} = \frac{1}{N_\mathrm{c}}\sum_{i=1}^{N_\mathrm{c}}\sigma_c^2.$$

5. Calculate the (empirical) variance of the chains' means:

$$\sigma^2_\mathrm{mean} = \frac{1}{N_\mathrm{c}}\sum_{c=1}^{N_\mathrm{c}}(\bar{x}_c-\bar{x})^2.$$

6. Calculate the ratio

$$\hat{R} = \frac{\frac{N_\mathrm{c}-1}{N_\mathrm{c}}\sigma^2_\mathrm{chain} + \frac{1}{N_\mathrm{c}}\sigma^2_\mathrm{mean}}{\sigma^2_\mathrm{chain}}$$

and use this to assess convergence. If the chains are well mixed and have all sampled the target distribution then $\sigma^2_\mathrm{chain} \simeq \sigma^2_\mathrm{mean}$ and $\hat{R} \simeq 1$, whereas if the chains have sampled different parts of the target distribution then their individual variances will be less than the variance between the estimates of the chains and $\hat{R} > 1$. The common heuristic approach is to regard the chains as converged if $\hat{R} \lesssim 1.2$.

Question: *Generate multiple realistiations of* $N_\mathrm{c} = 4$ *chains of* $N_\mathrm{s} = 10^4$ *samples from* $\mathrm{normal}(0, 1^2)$ *using* `rnorm` *and calculate sampling distribution of the Gelman-Rubin convergence statistic. Does* $\hat{R} = 1.2$ *seem like an appropriate rejection criterion? What is the chance that it would falsely reject a fully converged set of 4 chains of* 10^4 *elements?*

3.2.2 The Metropolis(-Hastings) algorithm

The Metropolis(-Hastings) (MH) algorithm is probably the most commonly used variety of MCMC, as it is simple, intuitive and (with some refinements) effective on a wide variety of problems. MH sampling algorithms are effectively a guided random walk that preferentially samples from the high-probability regions while exploring the full range of possibilities.

Algorithm

The MH algorithm is a two-step process for generating each link in the chain. Given a previous point $\boldsymbol{x}_{s-1}$, the next point is obtained by:

1. Draw a trial point from the proposal distribution $\Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1})$, the form of which can be chosen to increase the efficiency of the algorithm.

2. Accept the trial point with probability

$$\Pr({\rm accept}|\boldsymbol{x}_{\rm trial},\boldsymbol{x}_{s-1}) = \min\left[\frac{p(\boldsymbol{x}_{\rm trial})}{p(\boldsymbol{x}_{s-1})},1\right] = \min\left\{e^{\ln[p(\boldsymbol{x}_{\rm trial})]-\ln[p(\boldsymbol{x}_{s-1})]},1\right\},$$

   which is unity if the trial point is more probable than the previous point, but is only zero if $p(\boldsymbol{x}_{\rm trial}) = 0$. The second expression, which depends only on the difference of the logarithms of the density at the two points, is useful in cases where the density itself is difficult to represent using whatever precision is available computationally. It is also implicit here that $\ln(0)$ is, while technically undefined, less than $\ln(x)$ if $x > 0$.

3. If the trial point is accepted then set $\boldsymbol{x}_s = \boldsymbol{x}_{\rm trial}$; otherwise set $\boldsymbol{x}_s = \boldsymbol{x}_{s-1}$.

This last step is critical to the algorithm: a MH chain must inevitably include sequences in which the trail point is rejected, resulting in $\boldsymbol{x}_s = \boldsymbol{x}_{s+1} = \boldsymbol{x}_{s+2} = \ldots$; if the chain does not include such sequences it is almost certainly not sampling the target distribution effectively.

Stationary distribution

Independent of the form of $\Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1})$, the MH algorithm satisfies Eq. (20) and so its stationary distribution will be $p(\boldsymbol{x})$. The fact that this can be demonstrated so simply is a key to the utility of the MH algorithm.

Question: *A particularly intuitive way of understanding the detailed balance of the MH algorithm is to consider the case of the simple target density*

$$p(\boldsymbol{x}) = f_1\delta_{\rm D}(\boldsymbol{x}_A) + (1-f_1)\delta_{\rm D}(\boldsymbol{x}_B),$$

with $0 < f_A < 1$. *Provided the trial distribution is sufficiently broad that transitions between the two points can occur [i.e.,* $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1,{\rm trial}) > 0$*], the geometrical aspect of the situation is irrelevant; the key point is simply that the chain can exist in one of two states, A and B, with ssociated occupation probabilities* f_A *and* $1-f_A$. *The sampling algorithm is then defined in terms of the transition probabilities,* $\Pr(A|A) = 1-\Pr(B|A)$, $\Pr(A|B) = 1-\Pr(B|B)$. *If the chain is in state A at one step, what is the probability that it will be in state A on the next step? If the probability that the chain is in state A on the previous step is* f_A *(as desired), what is the probability that it is in state A on the next step? How does this determine the values of the transition probabilities?*

Proposal distribution

The one aspect of the MH algorithm that can be tuned is the proposal distribution. While most combinations of proposal and target distributions will result in full sampling in the limit of infinite samples, there is a clear motivation for the Markov chain to explore the target distribution

as efficiently as possible. That the proposal distribution can be tuned to do this can be seen easily by considering two extreme cases.

If the proposal distribution is very concentrated relative to the scales on which the target density varies then $p(\boldsymbol{x}_{\rm trial}) \simeq p(\boldsymbol{x})$ for all proposed points $\boldsymbol{x}_{\rm trial}$. The acceptance probability is then $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}) \simeq 1$, and almost all jumps are accepted; but the result is that the distribution is explored very slowly. (An analogy would be an adventurer trying to explore a mountain range by taking steps of a few millimetres.)

The other extreme is that the proposal distribution is much broader than region of parameter space for which the density is appreciable. If $\boldsymbol{x}_{s-1}$ is in a region of high density (as it will tend to be) then the odds are that $\boldsymbol{x}_{\rm trial}$ will be outside the high density region and so $\Pr({\rm accept}|\boldsymbol{x}_{\rm trial}, \boldsymbol{x}) \simeq 0$, and almost all jumps will be rejected. The resultant chain would just contain many copies of the current position until a jump was eventually accepted. The final chain would then contain very few independent points and the exploration of the target distribution would be very inefficient. (An analogy would be an adventurer trying to explore a mountain range by taking steps of millions of kilometres.)

Given that proposal distributions can be too narrow or too broad, the implication is that there is a range of intermediate values that are well suited to the target distribution. The most efficient sampling occurs if the acceptance ratio is in the range from $\sim 20\%$ to $\sim 40\%$, although these figures are distribution-dependent. More to the point, these values are only concerned with the efficiency of the sampling scheme – even an acceptance ratio of a $\sim 5\%$ or $\sim 90\%$ will produce a useful set of samples run for long enough. (And a large set of highly-correlated samples can be made less unwieldy by thinning, as described above.)

The most commonly used proposal distribution is a multi-variate normal, so that

$$\begin{aligned} \Pr(\boldsymbol{x}_{\rm trial}|\boldsymbol{x}_{s-1}) &= {\rm normal}(\boldsymbol{x}_{\rm trial}; \boldsymbol{x}_{s-1}, \boldsymbol{\Sigma}) \\ &= \frac{1}{[(2\pi)^N |\boldsymbol{\Sigma}|]^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}_{\rm trial} - \boldsymbol{x}_{s-1})^{\rm T} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_{\rm trial} - \boldsymbol{x}_{s-1})\right], \end{aligned} \tag{21}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix. Samples from this distribution can be drawn using the algorithm given in Section 2.3.1.

In the absence of any information about the structure of $p(\boldsymbol{x})$, there is no reason to prefer correlations between parameters, in which case the covariance matrix is diagonal, so that $C_{p,p'} = \delta_{p,p'} \Sigma_{p,p})$ and the trial point can be generated component by component by using $x_{{\rm trial},p} \sim {\rm normal}(x_{{\rm trial},p}; x_{s-1,p}, \Sigma_{p,p})$, for $p = 1, 2, \ldots, N_{\rm p}$.

One way of learning about the distribution is by from the initial samples drawn; if the "guessed" proposal distribution is inefficient then the sampling will not be very effective, but will still provide some information about the range of the high-probability region in each parameter (and about the correlations). A reasonable choice is then to use the covariance matrix of the chain, calculated using Eq. (4) for the proposal distribution. Note, however, that if the proposal distribution is updated in this manner then the chain must be started from scratch – combining samples produced using different proposal distributions does not necessarily satisfy detailed balance.

Even if the parameters are not strongly correlated, some information is required to set sensible scales for the "width" of the proposal in each of the coordinates. A proposal of ${\rm normal}(0, 1^2)$ could be quite useless even if just the units of the parameter in question are extreme. As with

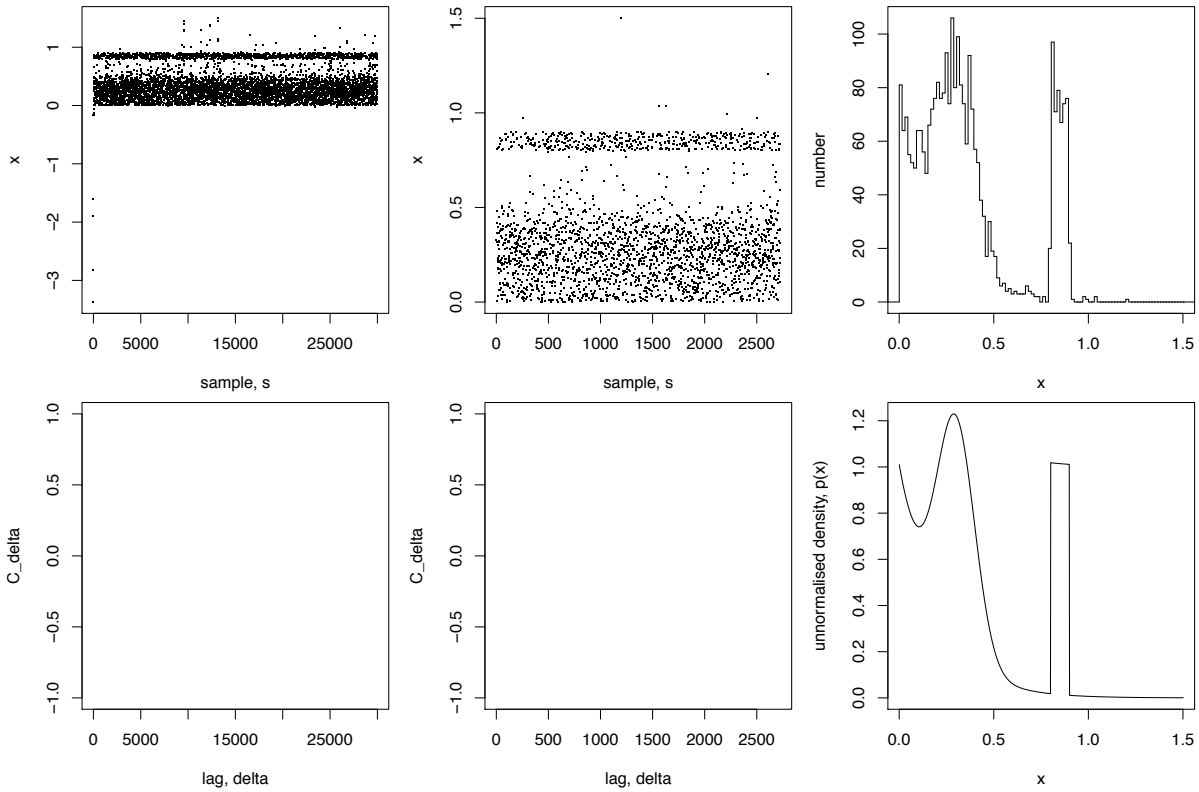

Figure 11: Worked example of the MH algorithm. The trace plot and auto-correlation function are shown in (a) and (b) before any post-processing and in (c) and (d) after the burn-in points have been removed and the chain has been thinned. The resultant histogram is shown in (e), which can be compared to the (normalised) target density in (f).

most problems in computational statistics, some "external" information is needed in order to produce a practical algorithm.

Worked example

Consider the task of generating samples from the (unnormalised) univariate density

$$p(x) \propto \exp\left[-\frac{1}{2}\left(\frac{x-0.3}{0.1}\right)^2\right] + \Theta(x)\,\exp\left(-\frac{x}{0.2}\right) + \Theta(x-0.8)\,\Theta(0.9-x), \tag{22}$$

where $\Theta(x)$ is the Heaviside step function.

Several options are available:

- The three components of this density are all analytically integrable and could be sampled from separately. This would be very quick computationally, but would require some algebra and coding. It would also not be easily extendable to even very similar distributions if, *e.g.*, one of the components was replace by a non-analytic funtion.
- Rejection sampling would be plausible, although some work would be required to ensure that a suitable envelope function was found.

- Running MCMC using the MH algorithm is plausible. If a normal proposal distribution was used then there would be only one parameter (*i.e.*, the width of the proposal distribution) to tune, although a number of convergence checks would be needed.

What if the actual form of $p(x)$ was not known? Imagine instead that the density is a "black box" routine that returns the density for any supplied value of x. In this case no analytic techniques could be used to generate samples from $p(x)$; it would also be impossible to use rejection sampling as there would be no way to come up with an envelope function. MCMC – and MH in particular – can be used, however, as these methods require only minimal "prior" information about the density from which samples are required.

Some inputs are required, though: a starting value; and a proposal distribution. Here a starting value is arbitrarily chosen to be in the interval $-100 \leq x \leq 100$ and the proposal distribution is taken to be the unit normal distribution centred on the previous point, *i.e.*, $\Pr(x_s|x_{s-1}, \text{trial}) = \text{normal}(x_s; x_{s-1}, 1^2)$. While these are arbitrary choices, they will eventually result in samples from $p(x)$; the only question is whether they will do so in a useful number of iterations.

The MH algorithm was run for $N_\text{s} = 30,000$ steps, yielding a chain with $6,063$ unique values, implying an acceptance ratio of ~ 0.2. These will not, in general, be independent samples from the target density: there will be a burn-in phase that means the first samples are not representative; and the samples will be correlated, so that the effective sample size is less than N_s. These issues can be at least partially assessed graphically.

The most basic graphical diagnostic is a trace plot, which simply shows x_s vs. sample number s. This is shown in Fig. 11 (a), and shows both an initial burn-in period, followed by apparently stationary sampling of the target density. The heuristic burn-in cut-off of defined by finding the first sample for which $p(x_s) \geq 0.1 p_\text{max}$ appears to provide a sensible cut-off; all the samples after this are regarded as being drawn from $p(x)$.

It is, however, clear from the trace plot that the samples are correlated. This can be seen more clearly in an auto-correlation plot, in which C_Δ vs. lag Δ is plotted, as in Fig. 11(b). The correlated chain could be used as is, but in a simple setting like this (*i.e.*, in which large samples can be generated with minimal computational effort) it is a useful simplifying step to thin the chain to produce a smaller set of (almost) independent samples. Here the auto-correlation function goes to zero for lags of $\Delta \gtrsim 10$, suggesting the chain be thinned by this amount.

The trace plot and auto-correlation function of the thinned post-burn-in chain is shown in Fig. 11 (c) and (d), respectively. These plots are now very uninteresting – but this is the desired state, as this is the way such plots would appear if it had been possible to draw independent samples from the target distribution in the first place.

These checks do not, however, demonstrate that the sampling procedure has explored the full density – if there were multiple, well-separated peaks then this might have only found one. Repeating this process for 10 separate chains and evaluating the Gelman-Rubin convergence statistic does confirm that this is the case.

Question: *How would the situation be changed if the density was* $q(x) = p(x) + \exp[-(x - 10^{100})^2/2]$ *instead?*

Having run these diagnostic checks, it is then reasonable to examine the chains to see what they imply about the distribution. For a univariate distribution like this, almost all the available information is captured in a simple histogram, as shown in Fig. 11 (e). In this case it is possible

to compare the results to a direct plot of the (normalised) density, something that is not possible in higher-dimensional settings. (If it is possible to make a plot of the distribution directly then the grid of values used to do so is probably more useful than a set of samples, and any subsequent calculations should make use of this.)

Question: *What would have happened if the proposal distribution was taken to be* $\Pr(x_s|x_{s-1}, \text{trial}) = \text{normal}(x_s; x_{s-1}, 100^2)$*? Or* $\Pr(x_s|x_{s-1}, \text{trial}) = \text{normal}(x_s; x_{s-1}, 0.01^2)$*?*

3.2.3 Gibbs sampling

A very different form of MCMC is provided by Gibbs sampling, in which only a single parameter is explored in any one step. This is useful if (all) the conditional distributions of the target distribution are known and can be sampled from easily. As such, Gibbs sampling is particularly useful for sampling distributions of high dimensionality in which most of the paramters are not directly linked, as is often the case for heirarchical models. It is hence useful to notate the target density $p(\boldsymbol{x})$ as a probability distribution $\Pr(\boldsymbol{x}) = \Pr(x_1, x_2, \ldots, x_{N_\text{p}})$, so that the conditional distribution of the i'th can be denoted as $\Pr(x_i|x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{N_\text{p}-1}, x_{N_\text{p}})$

Algorithm

Given an initial position $\boldsymbol{x}_1 = (x_{1,1}, x_{2,1}, \ldots, x_{N_\text{p},1})$, standard Gibbs sampling involves drawing from each conditional distribution successively, according to

$$\begin{aligned}
x_{1,2} &\sim \Pr(x_1|x_{2,1}, x_{3,1}, \ldots, x_{N_\text{p}-1,1}, x_{N_\text{p},1}),\\
x_{2,2} &\sim \Pr(x_2|x_{1,2}, x_{3,1}, \ldots, x_{N_\text{p}-1,1}, x_{N_\text{p},1}),\\
&\vdots\\
x_{N_\text{p}-1,2} &\sim \Pr(x_{N_\text{p}-1}|x_{1,2}, x_{2,2}, \ldots, x_{N_\text{p}-1,1}),\\
x_{N_\text{p},2} &\sim \Pr(x_{N_\text{p}}|x_{1,2}, x_{2,2}, \ldots, x_{N_\text{p}-1,2}, x_{N_\text{p}-1,2}),
\end{aligned}$$

where, in each case, the latest value for all the parameters (other than that being sampled) is used. The sequence then repeats, with a new value obtained that is conditional on the updated values of all the other parameters.

Hence this sequence of draws yields N_p new samples

$$\begin{aligned}
\boldsymbol{x}_1 &= (x_{1,2}, x_{2,1}, \ldots, x_{N_\text{p}-1,1}, x_{N_\text{p},1}),\\
\boldsymbol{x}_2 &= (x_{1,2}, x_{2,2}, \ldots, x_{N_\text{p}-1,1}, x_{N_\text{p},1}),\\
&\vdots\\
\boldsymbol{x}_{N_\text{p}-1} &= (x_{1,2}, x_{2,2}, \ldots, x_{N_\text{p}-1,2}, x_{N_\text{p},1}),\\
\boldsymbol{x}_{N_\text{p}} &= (x_{1,2}, x_{2,2}, \ldots, x_{N_\text{p}-1,2}, x_{N_\text{p},2}),
\end{aligned}$$

where some care must be taken with the indexing. Clearly, successive samples are correlated, and one option is to only retain the N_p'th sample (*i.e.*, after a full cycle); however $\boldsymbol{x}_{s+N_\text{p}}$ is *not* independent of $\boldsymbol{x}_s$, despite the fact that every parameter has been explored.

Exhaustiveness

There are certain classes of distribution that Gibbs sampling can never fully explore, most obviously those with separate peaks that are not aligned in any of the sampling parameters. Whereas the MH algorithm will, with a proposal distribution of broad support, eventually sample from an arbitrary distribution, Gibbs sampling has no equivalent of the proposal distribution that can be adjusted.

Stationarity

The stationary distribution of can be most easily seen in the simple case of a two-dimensional density for which $\boldsymbol{x} = (x, y)$. Assuming the s'th sample to be the result of a "move" in the x-direction, it is distributed according to

$$x_s \sim \Pr(x_s, |y_{s-1}) \tag{23}$$

and

$$y_s \sim \delta_{\mathrm{D}}(y_s - y_{s-1}). \tag{24}$$

But if (x_{s-1}, y_{s-1}) is itself already a draw from $p(x, y)$ then y_s is drawn from the target density by construction, and x_s is sampled from the correct conditional distribution. Hence Gibbs sampling will produce (correlated) draws from the target distribution, but whereas a single MH step locally explores all the parameters simultaneously, a Gibbs sampling step explores one parameter globally.

Question: *Consider the two-dimensional target density*

$$p(\boldsymbol{x}) = f_A\,\delta_{\mathrm{D}}(\boldsymbol{x} - \boldsymbol{x}_A) + f_B\,\delta_{\mathrm{D}}(\boldsymbol{x} - \boldsymbol{x}_B) + f_C\,\delta_{\mathrm{D}}(\boldsymbol{x} - \boldsymbol{x}_C) + f_D\,\delta_{\mathrm{D}}(\boldsymbol{x} - \boldsymbol{x}_D), \tag{25}$$

where $f_A + f_B + f_C + f_D = 1$, and the points are aligned according to: $x_A = x_C$ and $y_A = y_B$; $x_B = x_D$ and $y_B = y_A$; $x_C = x_A$ and $y_C = y_D$; and $x_D = x_B$ and $y_D = y_B$. What is the probability that $\boldsymbol{x}_s = \boldsymbol{x}_A$, given that $\boldsymbol{x}_{s-2}$ is drawn from the target density?

Utility

Figure 12: Illustration of the nested sampling scheme in two dimensions.

The most common use for Gibbs sampling is to explore a heirarchical model, which can have a large number of parameters (*e.g.*, hundreds or more, making MH sampling hopelessly ineffecient), most of which are not directly linked to each other. In that case it is usually much easier to make draws from the simple(r) conditional distributions than from the joint distribution.

Question: *Consider a survey of students' heights in which the data consist of the measuremed (i.e., noisy) heights of all* $N \simeq 30$ *students in the class,* $\{\hat{h}_i\}$*, and the aim is to determine the posterior distribution of the model parameters: the students'* N *true heights,* $\{h_i\}$*, and the population mean,* H*, and variance,* Σ^2*. Assuming that the measurement process gives sampling distributions* $\hat{h}_i \sim \text{normal}(h_i, \sigma_i^2)$ *and that the population itself is defined by* $h \sim \text{normal}(H, \Sigma^2)$*, what are the conditional distributions for the* h_i*,* H *and* Σ*? How could these be used to generate samples from the full posterior distribution* $\Pr(\{h_i\}, H, \Sigma|\{\hat{h}_i, \sigma_i\})$*? How could these samples then be used to estimate the height of the* i*'th student?*

3.2.4 Metropolis-within-Gibbs sampling

It is generally possible to combine MH and Gibbs sampling if neither algorithm is itself suitable for the entire sampling problem. If one (or more) of the conditionals is not a standard distribution then that parameter can be sampled using a univariate MH draw. The optimal approach is essentially completely problem-specific; the key point is to be aware that such options are available.

Figure 13: Weight vs. sample number for a typical nested sampling run.

3.3 Nested sampling

Nested sampling (Skilling, 2004) is a relatively new algorithm that increasingly is being used in astronomy and cosmology, in part due to the availability of the MultiNest code (Feroz *et al.*, 2009). Unlike MCMC methods, nested sampling produces independent samples from the target density. The samples are not uniformly weighted, however, which is a something of a disadvantage (or at least an inconvenience). That said, MCMC algorithms which have a accept/reject step (*e.g.*, MH) effectively produce non-uniformly weighted samples as well; the only difference is that the weights are integers. The most important distinction between the output of nested sampling and MCMC methods is that it also provides an estimate of the normalisation integral of the distribution.

Algorithm

Given a density $p(\boldsymbol{x})$ that is only non-zero over some region $R \subset R^n$ (and where $\boldsymbol{x} \in R^n$), nested sampling (which is illustrated schematically in Fig. 12) proceeds as follows:

1. Set the sample counter to $i = 1$.
2. Choose a number of live points, N, usually in the range 100-1000, which will determine the accuracy of the algorithm.
3. Distribute the live points uniformly within R and evaluate $p_j = p(\boldsymbol{x}_j)$ for $j \in \{1, 2, \ldots, N\}$.
4. Select the point with the lowest value of p_j and assign this as the i'th sample (*i.e.*, $\boldsymbol{x}_i = \boldsymbol{x}_j$ and $p_i = p_j$) with weight $W_i = e^{-i/N} p_i$.
5. Replace the above point with a new point $\boldsymbol{x}_{i,\mathrm{new}}$ such that $p(\boldsymbol{x}_{i,\mathrm{new}}) > p_i$ (*i.e.*, the new point must have a higher density than the old point which was just discarded).
6. Repeat steps 4 and 5 until the weights of the new points are much less (*e.g.*, $\lesssim 10^{-3}$) of the sum of the weights of the previous points that their contribution will be minimal.
7. Add the remaining all N "surviving" live points to the list of samples, assigning them equal weight $W_i = e^{-j/N} p_i$, leaving an eventual total of $j + N$ samples.

Nested sampling produces a set of differently-weighted but independent samples from the target density. As such it is quite different from MCMC algorithms, which all produce correlated

samples. Another important difference is that the need to create the initial sample of live points means that it cannot, at least in this simple form, sample a distribution with infinitely broad support. This limitation can be overcome by mapping R^n to some finite volume.

The most important distinctive feature of nested sampling is that the generation of a new point $\boldsymbol{x}$ with $p(\boldsymbol{x})$ becomes increasingly difficult/inefficieint as the algorithm explores ever smaller regions of ever greater densities. One option for doing this efficiently is to always sample from an ellipsoid that contains all the other suriving live points, but there is no formal guarantee that this ellipsoid will bound all the regions with densities above the current threshold. For the generally reasonably compact densities that arise in practical numerical problems numerical tests have shown that this approach is a good approximation, but no general proof is possible.

In terms of the output of the algorithm it is the weights of the samples that is critical to their interpretation. The first samples have low weights because the density is low (assuming that the initial sampling volume is large enough to include regions of low density) and the final samples have low weights because of the exponential factor; hence it is the intermediate samples which determine how effectively the density has been sampled. This weight profile is illustrated in Fig. 13.

References

Feroz, F., Hobson, M. P., and Bridges, M. (2009). MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics. *Monthly Notices of the Royal Astronomical Society*, **398**, 1601–1614.

Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, **7**, 457–511.

Skilling, J. (2004). Nested sampling. In *AIP Conference Proceedings of the 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, volume 735 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 395–405.

Tu, S.-J. and Fischbach, A. (2005). A study on the randomness of pi. *International Journal of Modern Physics*, **16**, 281–294.