Computer-Intensive Statistics

1 What is 'Computer-Intensive Statistics'?

The problem with buzz phrases[1] is that they can be imprecise. Roughly, 'computer-intensive statistics' is statistics that could only be done with *modern* computing resources, typically either

- Statistical inference on small problems which needs a lot of computation to do at all, or to do well.
- Statistical inference on 'huge' problems.

All of these terms are relative,[2] and I was reminded of just how relative by Sir David Cox's comment in the verbal discussion of Ripley (2005) that when he was a PhD student inverting a 5×5 matrix was the work of hours.[3]

The aims of this preliminary material are

- To give you an overview of the subject of the module,
- whet your appetite and give you some resources at which to nibble, and
- to make you aware of what will be required to fully participate in the practical classes and the assessment including a quick look at the software we will be using.

One very important idea for doing statistical inference 'well' on intractable statistical models (that is, most real-world ones) is to make use of *simulation*. So about 75% of this module could be subtitled *simulation-based inference*. It is something I stumbled into by necessity when I was a research student, and that example is used in this preliminary material.

[1] This one was used by Adrian Smith and myself to get EPSRC's predecessor SERC to make a large injection of cash into computers for statistics in the 1980's.

[2] comments on the `R-help` list show that people's 'huge' problems differ by at least three orders of magnitude.

[3] and 'computers' were human beings, often statistics research students.

2 Prerequisites

If you completed the *Statistical Computing* and *Statistical Modelling* APTS modules, congratulations, you will just need to do any revision you feel appropriate.

You need to have a basic understanding of *simulation* and its uses. Pages 47–51 of the *Statistical Computing* notes will give you that and reference material, but I have included an Appendix as a reminder and as a less technical introduction.

More than any previous module, familiarity with R is essential: we cannot actually do computer-intensive statistics without computers, and I assure you that using R is one of the least painful ways to do it – I wish it had been around when I was a student. Venables & Ripley (2002) is the recommended reference material for R for 'computer-intensive statistics' – Bill Venables and I wrote it[4] because at last in S, S-PLUS and later R we had a vehicle to explain 'modern' statistics, much of which is 'computer-intensive'.

I hope that those of you who are far-from-reluctant R programmers will bring your laptops: see section 7 for what packages you will be using.

3 A motivating example

One lunch time when we were PhD students, Frank Kelly and I looked at a paper in the latest issue of *Biometrika*. Now *Biometrika* was (and is) a very reputable journal, but we were pretty sure the paper was seriously flawed, and we set out to see if we could do something similar but correct.

Suppose you have observed a spatial pattern of trees, say n trees within a sampling area A, and the actual (two-dimensional) locations are $\boldsymbol{x}_i, i = 1, \ldots, n$. The aim was to produce a model (a stochastic process) for the pattern of the trees. What makes this interesting is

(a) there is no definitive Euclidean coordinate system – the trees do not know where North is nor the Greenwich Meridian nor the Equator.

(b) the trees do not have an order, so we have a *set* not a *sequence*.

(c) the trees are not positioned independently – trees compete for light.

(d) trees do not compete just within the study area A but also with trees outside A.

Point (d) makes modelling more difficult – we need to model a process of trees within the whole forest observed only in A – but for simplicity we will ignore that here.

The model we came up with implied a joint density for the sample $\boldsymbol{x} = \{\boldsymbol{x}_i, i = 1, \ldots, n\}$ of

$$f(\boldsymbol{x}; c, R) \propto c^{t(\boldsymbol{x},R)} \qquad c \geq 0 \tag{1}$$

where $t(\boldsymbol{x}, R)$ is the number of 'R-close pairs', the number of (unordered) pairs of trees less than R units apart. (Note that this meets modelling aims (a–c).)

[4] first edition 1994.

At that point we had several questions

1. What do spatial patterns from process (1) typically look like?
2. What values of c are valid? Our theory suggested that $c > 1$ violated point (d), but (1) is a valid pdf – is it modelling anything interesting?

   Note that $c = 0$ *is* interesting – it is a *hard-core* interaction process.

   The original paper was entitled *A Model for Clustering*, and clustering would need $c > 1$.
3. How could we do statistical inference on the parameter c, and perhaps R?
4. How could we write this up and convince others?

At around that time Prof. D. G. Kendall[5] told me

> *you don't really understand a stochastic process until you know how to simulate it*

and perhaps I should add

> *you don't really know how to simulate a stochastic process until you have done so and validated your simulations*

So these points are inter-related – one way to validate the simulation is to be able to do inference on its output.

I did have an idea about how to do the simulation – see section 4.

Inference from (1) superficially looks easy: if we write it as

$$f(\boldsymbol{x}; \theta, R) \propto \exp -\theta t(\boldsymbol{x}, R) \qquad -\infty < \theta \leq \infty \tag{2}$$

it is shown as a canonical exponential family for θ, albeit with an unusual parameter range. The problem is that this is really

$$f(\boldsymbol{x}; \theta, R) = C(\theta, R) \exp -\theta t(\boldsymbol{x}, R) \qquad -\infty < \theta \leq \infty$$

and to find $C(\theta, R)$ we need to do a $2n$-dimensional integral that we can only do numerically – and no reasonable approximations were known to us then (nor now).

However, this tells us that the MLE for c solves[6]

$$E_c\, t(\boldsymbol{X}, R) = t(\boldsymbol{x}, R)$$

where $\boldsymbol{X}$ is a sample from (1) with parameter θ. If we can simulate from (1) we can estimate the LHS as a function of θ and hence get the MLE: for more details see Ripley (1988, §4.6).

So once we can produce simulations we can make a lot of progress on our statistical questions – at least if we can do so within the computing resources available.

[5] Prof. Wilfrid Kendall's late father, at the time Professor of Mathematical Statistics at Cambridge.

[6] note $E_0\, t(\boldsymbol{X}, R) = 0 \leq t(\boldsymbol{x}, R)$.

4 Markov Chain Monte Carlo

My idea for simulation was to make use of a *spatial birth-and-death* process which we had recently heard about in some graduate lectures[7] from Dr Chris Preston. Here is a stripped-down version.

Suppose we have a population of N objects in study area A. Each of them has an exponential lifetime until death, and objects are being born at a given rate. Both the rate of birth and the rate of death can depend on the current crowding. This is clearly a Markov process, and fairly clearly stationary under modest conditions (as the more points there are, the faster the overall rate of death, so the population cannot explode). Also, we can choose (in several ways) the birth and death rates so that the stationary distribution is the variable-n version of (1).

The idea of using a Markov process to simulate from its stationary distribution was new to us (but not new to the world) and is now called *Markov Chain Monte Carlo*. The term is appropriate, for it seems *Monte Carlo sampling* arose as a code word at Los Alamos during WWII—and MCMC is usually attributed to a five-author paper (Metropolis *et al.*, 1953) from a group at Los Alamos published in 1953. What they proposed is now known as *Metropolis sampling*.[8]

Suppose we restrict our birth-and-death process to just n or $n-1$ objects. With n objects we can only have a death, and we take a uniform death rate (so pick an object at random, and kill it). Now we have $n-1$ objects, so we must have a birth, and add point ξ with density

$$f(\xi \mid \boldsymbol{x}) \propto \frac{c^{t(\boldsymbol{x} \cup \{\xi\}, R)}}{c^{t(\boldsymbol{x}, R)}} \quad . \tag{3}$$

Now $t(\boldsymbol{x} \cup \{\xi\}, R)$ is the number of R-close pairs when we add an object at ξ, and this is the number of R-close pairs in $\boldsymbol{x}$ (constant) plus the number of objects in $\boldsymbol{x}$ which are R-close to ξ. More verbosely, we consider adding a new object, count the number of existing objects within distance R of the new object, and arrange to keep the new object with probability proportional to c raised to the power of the count.

By combining a death and a birth as a single step we have a Markov chain on sets of n objects, and by the methods discussed in the *Applied Stochastic Processes* module we showed that its stationary distribution is (1).

This is in fact a quite general way to sample from a multivariate distribution, and I published it as such in Ripley (1979). It was rediscovered by and published in Geman & Geman (1984) as the *Gibbs sampler*. The general description for the joint distribution of n variables is to select one variable at random, and re-sample it from its conditional distribution given all the others.

It may not be obvious that we have made progress, as (3) also has an omitted normalizing constant. However, this is just a 2-dimensional integral, and there are methods of simulation which just need an integrable upper bound such as *rejection sampling* (see Appendix A). For $c \leq 1$ it is clear that the RHS of (3) is bounded by 1 (and we may be able to find a better

[7] which he never really published, so going to graduate lectures can sometimes pay off in ways you never imagine! Even boring ones ....

[8] Metropolis is the first of five authors in alphabetical order, and one of the others is very famous: so if this is fair is a story for another time.

bound). So you might like to check for yourself[9] that one way to sample from (3) is to draw a uniformly distributed object from A, and accept it with probability $c^{\{t(\boldsymbol{x}\cup\{\xi\})-t(\boldsymbol{x})\}}$ or try again. (In case anyone wonders why we do not apply rejection sampling for the whole process – for the sort of datasets we are interested in the probability of acceptance would be astronomically small.)

There are all sorts of practical considerations, but this provided us with a way to simulate our process for the sort of parameter values we were interested in that was just about fast enough for our limited computing resources[10] as lowly students, and hence to begin to answer our questions.

Be aware that there are often a lot of choices in designing MCMC methods – the one I have sketched is good for $c < 1$, but not for $c > 1$, for example. People have since found ways using MCMC to simulate it exactly (Berthelsen & Møller, 2003).

So far I have described using a Markov chain to obtain a single sample from a stochastic process by running it for an infinite number of steps. In practice we run it for long enough to get close to equilibrium (called a 'burn-in' period) and then start sampling every $m \geq 1$ steps. We can estimate any distributional quantity *via* the law of large numbers

$$\frac{1}{N}\sum_{i=1}^{N} h(X_{mi}) \to E\, h(\boldsymbol{X})$$

for any m, so if $h()$ is cheap to compute we may as well average over all steps. In practice we often take m large enough so that samples are fairly dissimilar—in the spatial process we used $m = 2n$.

There are many practical issues. Where do we start? How do we know when we are 'close to equilibrium'? And so on. Note that the issue of whether we are yet close to equilibrium is critical if we are simulating to get an idea of how the stochastic process behaves – Geman & Geman (1984) based all their intuition on processes which were far from equilibrium, but incorrect intuition led to interesting statistical methods.

References

MCMC can be approached from wide range of viewpoints – from theoretical to practical, as a general technique or purely Bayesian, and at different levels (especially in probability background). Texts which have interesting perspectives include Gamerman & Lopes (2006) Gelman *et al.* (2004), Gilks *et al.* (1996) and Robert & Casella (2004). As a topic in simulation, it is covered in Ripley (1987) and Dagpunar (2007), and as a method of integration in Evans & Swartz (2000). The simulation books, Gilks *et al.* (1996) and Gamerman & Lopes (2006) are perhaps the most approachable unless you have a strong probability background (when I suggest Robert & Casella (2004)).

For those unfamiliar with applied Bayesian work, Albert (2007) and Gelman *et al.* (2004) provide accessible introductions to the computational aspects with non-trivial worked examples.

[9] this is an example of rejection sampling, so you need write it in the standard form (as given for example in Appendix A) and check the distribution it samples from is indeed the correct one.

[10] a few CPU seconds a day on a mainframe far slower than today's PCs: see Ripley (2005).

Wasserman (2004, Chapter 24) gives a 9-page overview.

Robert & Casella (2010) is a recent fairly short book on both simulation and MCMC with supporting R code in package `mcsm`. I find its selection of material rather too biased towards the research interests of its authors.

Ripley (1988) and Møller & Waagepetersen (2003) cover inference for spatial point processes in detail.

The background Markov processes theory is covered in the chapters by Roberts and Tierney in Gilks *et al.* (1996), and its practical relevance in Roberts & Rosenthal (1998).

Some of these books have earlier first editions, and those are not recommended as there have been substantial changes – MCMC is still a young subject.

5 Bootstrapping

Suppose we were interested in inference about the correlation coefficient of n iid pairs (x_i, y_i) for moderate n, say 15. (Efron (1982) apparently was. To be quite specific, look at the US Law School data in Appendix B.) If we assume that the samples are jointly normally distributed, we might know that there is some distribution theory (using Fisher's inverse tanh transform), but suppose we do not wish to assume normality?

We could do a simulation experiment: repeat R times sampling n pairs and compute their correlation, which gives us a sample of size R from the population of correlation coefficients. **But** to do so, we need to assume both a family of distributions for the pairs **and** a particular parameter value.

Efron's idea was to create more samples by resampling the data. This sounds preposterous[11], but Efron was astute enough to give it an appealing name, *the bootstrap* after the (fictional) adventures of (the historical) Baron Münchhausen,[12] perhaps the source (in German) of the phrase

> *to pull / lift / raise oneself up by one's bootstraps*

(which the Oxford English Dictionary references to Joyce's *Ulysses*.)

Suppose we sample from $(x_i, y_i), i = 1, \ldots, n$, taking a sample of size n with replacement. Then unless we are extraordinarily (un)lucky we will get a sample which contains some of the original data points more than once, and some not at all. Such a sample is called a *bootstrap* (re)sample, and we can do this R times to get R independent bootstrap samples.

How can we use the bootstrap resamples to do inference on the original problem, and under what circumstances (if any) is such inference valid? Note that the bootstrap resample is unlike the original sample in many ways, perhaps most obvious that (with very high probability) it contains ties whereas correlation coefficients are usually computed for continuous quantities.

The lectures will go into more details of when and how, but bootstrapping is most commonly used

[11] and after many years of thought, some have concluded that it is.

[12] `http://en.wikipedia.org/wiki/Baron_Munchhausen`: the spelling of his name in English is variable.

- as part of a procedure to produce more accurate confidence intervals for parameters than might be obtained by classical methods (including those based on asymptotic normality). Often this is very similar to using more refined asymptotic statistical theory, and I once heard bootstrapping described as

  'a way to do refined asymptotics without employing the services of Peter Hall'

- to allieviate biases.

The basic idea is that if we have a statistic $t(\boldsymbol{x})$ of interest, the distribution of $t(\boldsymbol{x}^*) - t(\boldsymbol{x})$ for a bootstrap sample $\boldsymbol{x}^*$ ought to be similar to the distribution of $t(\boldsymbol{x}) - E\,t(\boldsymbol{X})$ where $\boldsymbol{X}$ is a sample from the unknown stochastic generating mechanism. So we can use the variability of the statistics from the bootstrap samples to mimic the variability of the original sampling.

The basic bootstrap has a very appealing simplicity, but just as in survey sampling and simulation experiments (see Appendix A), it is normally best to do the sampling in a less uncontrolled manner. Also, if we want a sample from a continuous distribution we can use the *smoothed bootstrap* (a sample from a density estimate rather than from the original data) and thereby avoid ties.

In part because it is so simple to describe and easy to do, bootstrapping has become popular and is often used when it is not valid. For a careful account by two of the pioneers of the subject, see Davison & Hinkley (1997). Wasserman (2004, Chapter 8) gives a 9-page overview.

Because the term is popular, it sometimes used for other things. For example *parametric bootstrapping* is a new term for some aspects of an old idea, *simulation-based inference*. One way to think of (standard) bootstrapping is that that it is iid sampling from the empirical cumulative distribution function $\widehat{F}$, and sampling from $F_{\widehat{\theta}}$ for the fitted member of some parametric family F_θ is sometimes called 'parametric bootstrapping'.

6 Large datasets

This is an important topic, but not one for which you need to do much preparation ... just think about two questions:

1. How many items would you consider to be a (a) *large* and (b) *huge* data set?
2. What fields are you aware of that generate huge datasets? Which (if any) are important to you?

(You will be asked for your answers, anonymously, at the module.)

Visualization of large datasets is an important topic – see Unwin *et al.* (2006) for the viewpoints of the Augsburg school.

You might like to look at `http://www.stats.ox.ac.uk/~ripley/Cox80.pdf`, the lecture version of Ripley (2005).

7 Software

R ships with the recommended package `boot` which was written to support Davison & Hinkley (1997). This should already be installed, but Debian/Ubuntu Linux users may have to install this package separately.

Markov Chain Monte Carlo is definitely computer intensive and so often needs to be computed efficiently. This can need careful consideration of the MCMC sampler, and efficient implemenation of the chosen algorithm. This mitigates against general-purpose code, and as a result much software has been written for MCMC work on specific problems. We will make use of some examples from package `MCMCpack`. For less critical work, MCMC samplers can be written in R itself, and we will see examples from Albert (2007) with associated R package `LearnBayes` and Gelman *et al.* (2004): the latter has an appendix on Gelman's somewhat eclectic approach which is available on-line at `http://www.stat.columbia.edu/~gelman/bugsR/software.pdf`.

The BUGS family of programs have provided a more general approach in which Bayesian models of a fairly broad class can be expressed in the BUGS language, which the program 'compiles' to a fairly efficient Gibbs Sampler and provides simple facilities to run simulations of the compiled model. We will make use of the R package `BRugs` which includes OpenBUGS, the latest member of the BUGS family that is currently available only for Windows. JAGS[13] is a program written in C++ that re-implements the BUGS language, and provides a way for non-Windows users (and especially Linux and Mac users) to use the BUGS language, and we will also take a look at that.

The JAGS user manual[14] contains a brief introduction to the BUGS language. Those interested in software should take a quick look before the module.

There was a new release of JAGS, 2.0.0, on April 26, but that is buggy so use either 2.1.0 or 1.0.3/4.

The results of a MCMC simulation are all of similar form: at each iteration we collect (the word *monitor* is often used in the BUGS literature) the values of all the variables of interest, so the raw results are a multivariate time series. All the software we will be using has been written to produce output suitable for analysis by the R package `coda`.

If you are bringing your laptop with you, please install R packages `boot` (if not already installed), `coda`, `LearnBayes` and `MCMCpack`, and if you are using Windows `BRugs`. Optionally (but recommended for non-Windows users), install `JAGS` and the corresponding R package `rjags` – note that the version number of this must match that of `JAGS`. All the R packages mentioned are available from CRAN except `BRugs`, which is available from CRANextras and hence should be found from the 'Install packages' menu item on a Windows installation of R. If you have installed these in the past, please ensure that the versions you bring are current.

[13] `http://www-fis.iarc.fr/~martyn/software/jags/`, but note the links to version 2.
[14] `http://sourceforge.net/projects/mcmc-jags/files/Manuals/2.0`.

References

Albert, J. (2007) *Bayesian Computation with R*. New York: Springer.

Berthelsen, K. K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* **30**, 549–564.

Dagpunar, J. (2007) *Simulation and Monte Carlo. With Applications in Finance and MCMC*. Chichester: Wiley.

Davison, A. C. and Hinkley, D. V. (1997) *Bootstrap Methods and Their Application*. Cambridge: Cambridge University Press.

Efron, B. (1982) *The Jackknife, the Bootstrap, and Other Resampling Plans*. Philadelphia: Society for Industrial and Applied Mathematics.

Efron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap*. New York: Chapman & Hall.

Evans, M. and Swartz, T. (2000) *Approximating Integrals via Monte Carlo and Deterministic Methods*. Oxford: Oxford University Press.

Gamerman, D. and Lopes, H. F. (2006) *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Second Edition. London: Chapman & Hall/CRC Press.

Gelman, A., Carlin, J. B., Stern, H. S. and Rubin, D. B. (2004) *Bayesian Data Analysis*. Second Edition. Chapman & Hall/CRC Press.

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721–741.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J. (1996) *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953) Equations of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 1087–1091.

Møller, J. and Waagepetersen, R. (2003) *Statistical Inference and Simulation for Spatial Point Processes*. London: Chapman & Hall/CRC Press.

Ripley, B. D. (1979) Algorithm AS137. Simulating spatial patterns: dependent samples from a multivariate density. *Applied Statistics* **28**, 109–112.

Ripley, B. D. (1981) *Spatial Statistics*. New York: John Wiley and Sons.

Ripley, B. D. (1987) *Stochastic Simulation*. New York: John Wiley and Sons.

Ripley, B. D. (1988) *Statistical Inference for Spatial Processes*. Cambridge: Cambridge University Press.

Ripley, B. D. (2005) How computing has changed statistics. In *Celebrating Statistics: Papers in Honour of Sir David Cox on His 80th Birthday*, eds A. C. Davison, Y. Dodge and N. Wermuth, pp. 197–211. Oxford University Press.

Robert, C. P. and Casella, G. (2004) *Monte Carlo Statistical Methods*. Second Edition. New York: Springer.

Robert, C. P. and Casella, G. (2010) *Introducing Monte Carlo Methods with R*. New York: Springer.

Roberts, G. O. and Rosenthal, J. S. (1998) Markov-chain Monte Carlo: Some practical implications of theoretical results. *Canadian Journal of Statistics* **26**, 5–31.

Unwin, A., Theus, M. and Hofmann, H. (2006) *Graphics of Large Datasets. Visualizing a Million*. Springer.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth Edition. New York: Springer-Verlag.

Wasserman, L. (2004) *All of Statistics. A Concise Course in Statistical Inference*. Springer.

A Simulation

This appendix provides a minimal introduction to simulation.

Simulation means (here) the use of computer-generated data from specified stochastic mechanisms: an earlier term was *Monte Carlo methods*. This is often done to try things out, for example to find out if the approximate (asymptotic?) distribution of a test statistic or the coverage property of a confidence interval procedure holds for a realistic simulation of the problem of interest. The advantage of a simulation is that (barring mistakes) you do know the true stochastic mechanism which generated the data.

Random Numbers

The fundamental building block for stochastic simulations is *random numbers*, meaning the generation of independent identically-distributed $U(0, 1)$ random variates.

Occasionally genuine random numbers are used, from physical sources (e.g. electronic noise). In 1955 RAND published a book of 10^6 random digits and 10^5 normally-distributed random numbers, and in the 1980's George Marsaglia distributed a CD-ROM of random numbers. However, physical random numbers have drawbacks:

1. They can be far too slow to generate or read in.
2. If generated on the fly the simulation is not repeatable.
3. You rely on the physical mechanism being implemented perfectly, which in the case of the RAND tables was not the case.

There are fairly convenient ways to get physical random numbers – see R packages `accuracy` and `random`, as well as commercial chips – but point 3 still applies.

As a result, the vast majority of random numbers used are *pseudo-random* generated from a *seed* by a deterministic algorithm which makes a new number out of the fine details of the last number (or the last few numbers).

Not so long ago, users needed to know how this was done[15] as it was usually done insufficiently well. Nowadays you can probably assume that the PRNG[16] in a major statistical system is good enough, although you should be aware that all PRNGs have some systematic departures from randomness, and it is a good idea in critical studies to compare results from more than one PRNG. (R has several: see function `RNGkind`.)

To get pseudo-random numbers in R, call the function `runif`. R saves the *seed* as variable `.Random.seed` in the workspace, so each session starts using the pseudo-random number stream where the last one left off *if you save the workspace*. If you want reproducible results, it is recommended to call the function `set.seed(i)` with $1 \leq$ `i` < 10000 to choose a pre-selected seed.[17] Then if you want to re-run the results (for example to collect more details or to use a different analysis), just call `set.seed(i)` again for the same `i`.

[15] the references will tell you if you are interested.

[16] Pseudo-Random Number Generator.

[17] There are many more available, but small numbers are easier to remember.

Other Random Variables

You will probably find that the system you are using has functions to generate random variables from the common non-uniform distributions: R has `rnorm`, `rpois` .... However, it is helpful to understand the basic principles, as they are the same as are needed to simulate from more complex problems.

The basic task is to turn a stream $(U_i, i = 1, 2, \ldots)$ of random numbers into samples from the specified stochastic mechanism. This should be done fast enough and without undue sensitivity to the fine details of the random numbers.

Perhaps the simplest way to specify a univariate random variable is via its CDF F. Then if $X \sim F$ and F is continuous, $F(X) \sim U(0, 1)$. Inverting this shows that $F^{-1}(U)$ has CDF F if $U \sim U(0, 1)$. For a discrete distribution this still holds if we define

$$F^{-1}(x) = \min\{x \,|\, F(x) \geq u\}$$

This is known as *inversion*, and is often a good general method if F^{-1} is known[18] or a fast approximation is available. For example, the exponential distribution has $F(x) = 1 - \exp -\lambda x$, so $F^{-1}(U) = -(1/\lambda)\log(1 - U)$, which can be simplified slightly as $1 - U$ and U have the same distribution. For a discrete distribution inversion amounts to searching a table of cumulative probabilities.

A great deal of ingenuity has been expended in using stochastic mechanisms to get a desired distribution. For example, counting events in a Poisson process gives a Poisson-distributed random variable, and a Poisson process can easily be simulated by adding up exponential random variables,[19] at least provided the Poisson mean is not very large. Another ingenious idea is due to Box & Muller (1958):

1. Generate $U_1 \sim U(0, 1)$ and set $\Theta = 2\pi U_1$.
2. Generate $U_2 \sim U(0, 1)$ and set $E = -\log U_2$, $R = \sqrt{2E}$.
3. Return $X = R\cos\Theta$, $Y = R\sin\Theta$.

Then X, Y are independent standard normal variates: if we only want one we can throw Y away or keep it in our pocket for next time we are asked.

One other general principle is worth knowing, that of *rejection sampling*. Suppose we know how to simulate from probability density (or mass function) g with the same support[20] as f such that f/g is bounded by $M < \infty$. Then we can create a sample from f by

Repeat
  Generate Y from g.
  Generate $U \sim U(0, 1)$.
until $MU \leq f(Y)/g(Y)$.

The expected number of tries is M, so this works best if g and f are closely matched: it is often used to make approximate methods of simulation exact.

[18] it is the quantile function, given in R as `qnorm` etc.

[19] and adding up $-\log U_i$ can be replaced by multiplying U_i.

[20] $g > 0$ wherever $f > 0$.

Another advantage of rejection sampling is that we don't need to know f, only a function $f' \propto f$ and a bound on f'/g. This allows us to ignore normalizing constants which may be awkward to compute, or even not known explicitly.

The references (especially Devroye, 1986) cover a plethora of techniques and applications of them to standard distributions. It is unlikely these days that you will need to actually implement them: look around for existing implementations in whichever programming language you are using. R has methods for generating most of the standard distributions (and some not so standard ones) in its basic `stats` package, and contributed packages provide many more, especially packages `Runuran`, `SuppDists` and `distrSim`.

Simulation Experiments

The most important thing to remember when you are using simulation is that you are **performing an experiment**. So, the experiment should be designed, and the data it produces should be analysed.

The main point in the design of a simulation experiment is to eliminate inessential variation and so make the results as precise as possible. The first and most important idea is to compare ideas on the same simulation runs, sometimes known as using **common random numbers** although it is just an application of *blocking*. Other ideas to reduce variability include

control variates: estimate the difference between the result of interest and a similar one which can be obtained analytically and also estimated from the data.

antithetic variates: deliberately introduce negative correlations between runs: rarely useful.

importance sampling: simulate from a different distribution and adjust. Useful to make rare events less rare, for example.

The more complicated the problem the less likely these are to be useful, but they should be borne in mind if a simulation experiment is taking too long.

The main difficulty with *analysis* is dependence in the results. Many simulations are of events through time, and so the results from a time series – there are specialized methods of analysis for such time series. Note that this applies to simulations by MCMC.

Reference books

Bratley, P., Fox, B. L. and Schrage, L. E. (1987) *A Course in Simulation*. Second Edition. Springer.

Devroye, L. (1986) *Non-uniform Random Variate Generation*. Springer.
[Out of print, web edition at `http://www.nrbook.com/devroye/`.]

Hörmann, W., Leydold, J. and Derflinger, G. (2004) *Automatic Nonuniform Random Variate Generation*. Springer.

Ripley, B. D. (1987) *Stochastic Simulation*. Wiley.

B R exercises

These R exercises can be approached in two ways. For those of you comfortable with R, please try them in the form given here, making use of your R references and the on-line help. For others, both a sheet of hints and a step-by-step guide are available on the module's website.

Bootstrapping

Efron (1979, 1982) gives the following data on admissions to 15 US Law Schools

```
LSAT:
576 635 558 578 666 580 555 661 651 605 653 575 545 571 594
GPA:
3.39 3.30 2.81 3.03 3.44 3.07 3.00 3.43 3.36 3.13 3.12 2.74 2.76 2.88 2.96
```

The data are also to be found in Efron & Tibshirani (1993, pp. 19–20) with more background, including that these are actually a sample from a population of size 82.

Reminder: If r is the correlation coefficient from a bivariate normal population with true correlation ρ, then Fisher showed that atanh(r) is approximately distributed as $N(\text{atanh}(\rho), 1/(n-3))$.

Ex 1 Enter these data into R as a data frame `law`, plot them and compute the correlation coefficient.

Use Fisher's theory to give a 95% confidence interval for ρ assuming normality.

Ex 2 Create a bootstrap sample. Here is one simple way to do it:

```
law[sample(1:15, replace=TRUE), ]
```

Now compute its correlation coefficient.

Repeat 1000 times and put in a vector, making use of the `replicate` function. Now take a look at the bootstrap distribution on both correlation and `atanh` scales.

Ex 3 Another way is to make use of function `boot` in the recommended R package of the same name.

Function `boot` is complex so please read its help thoroughly. (We will be using it extensively in the module.)

The following should get you started.

```
library(boot)
stat <- function(x, ind)  cor(law[ind, ])[1,2]
out <- boot(law, stat, R=1000)
out
plot(out)
```

Use the help to work out what you are being shown here.

Ex 4 How about a confidence interval for ρ? We can use function `boot.ci` to produce several different confidence intervals – do so on both correlation and `atanh` scale.

Ex 5 Cox and Lewis (1966) reported 799 time intervals between pulses on a nerve fibre. The dataset can be downloaded from
http://www.stat.cmu.edu/~larry/all-of-statistics/=data/nerve.dat
and is used by Wasserman (2004, pp. 98, 111).

Use the bootstrap to get confidence intervals for the median and skewness of these data. You can either write your own R function to compute the skewness, or get one from contributed package `e1071` and about ten others. To see if you have one installed, use

```
help.search("skewness")
```

Spatial patterns and MCMC

Ex 6 The Strauss process (1) and ways to simulate it are contained in R packages `spatial` and `spatstat`.

Let us consider the Swedish pines data from Ripley (1981), described in Venables & Ripley (2002, §15.3). Retrieve and plot it in R by

```
library(MASS)
library(spatial)
pines <- ppinit("pines.dat")
eqscplot(pines, xlim = c(0, 10), ylim = c(0, 10), xlab = "", ylab = "")
```

How many points are there? How might you describe the pattern? By the way, the coordinates are in metres.

Venables & Ripley (2002, p. 443) suggest that $R = 0.7$ and $c = 0.15$ are reasonable estimates. Use function `Strauss` to simulate with these parameter values and compare a plot with the real data.

Ex 7 Wasserman (2004, p. 412) gives an example, and the following is based on his (incorrect) R code for his Figure 24.2.

```
metrop <- function(N, b)
{
    x <- numeric(N)
    for(i in 2:N){
        y <- rnorm(1, x[i-1], b)
        r <- (1+x[i-1]^2)/(1+y^2)
        u <- runif(1)
        x[i] <- ifelse(u < r, y, x[i-1])
    }
    x
}

par(mfrow=c(3,1))
N <- 1000
for(b in c(0.1, 1, 10)) {
  plot(metrop(N, b), type="l", xlab="", ylab="")
  abline(h=0, col="grey")
}
```

The aim here is to simulate from the Cauchy distribution as the stationary distribution of a Markov chain.

(a) The Markov chain being simulated here is a random walk in which moves are accepted with probability `r` and otherwise rejected. Write down a formal description of the stochastic process.

(b) What is the rôle of the parameter `b`? Why did Wasserman decide to show the three values he did?

(c) Where did `N = 1000` come from? Is it a reasonable value? Experiment with changing it.

(d) Compare the output with the intended Cauchy distribution.

(e) (More ambitious.) If you are familiar with Markov chains, try to prove that there is a stationary distribution and that it is Cauchy.

(f) This is not a serious exercise in sampling from a Cauchy distribution – find out how it is done efficiently in real applications.