Chapter 8

Monte Carlo methods

Monte Carlo (MC) methods are statistical methods that are based on the computer-generated random numbers. Random numbers can be used directly to asses some features of complicated random model, or they can be used in drawing randomized samples of the existing data. The latter case is called resampling. When Monte Carlo is is used to create so-called Markov chains, the method is called MCMC and that is not dealt with in this chapter. In any case, the inference with MC (or MCMC) is based on the averaged descriptive statistics of the data that results from the MC procedure.

8.1 Random number generation

Before MC methods can be used, we must have procedure that can generate random numbers from the desired distribution. In some cases the probability model can consist of an algorithm that is difficult to describe with parametric distribution. In that case, the algorithm itself can be used to create samples that obey the unknown distribution when some random input is created. In the common case, however, we know the parametric distribution from which we want to create random numbers.

Even the creation of uniform random integers is somewhat complicated if we want the *pseudorandom numbers* to come from a sequence that will seem random. First of all, the length of the period, i.e. the length of unique sequence of integers, should be large. Second, the integers should pass any test of uniformity. Third, there should not be detectable autocorrelation in the sequence. All the uniform number generators in the modern computing environments should be reliable nowadays. For example, the Mersenne twister algorithm was developed in 1997 and has very good random properties.

The pseudorandom number generators always generate random integers. The conversion to uniform real numbers between 0 and 1, $U \sim \mathcal{U}(0,1)$, is done by dividing

the random integer by the largest possible integer (plus one) in the system. Usually the generators can return the lower limit of 0.0 (naturally, with very low probability), but not the upper limit of 1.0.

Since the uniform distribution should be well implemented in almost all systems, and it is hard to implement yourself, we will concentrate on the creation of continuous random numbers from more complicated distributions using the uniform numbers as an input.

8.1.1 Inversion method

The most general algorithm for random numbers is the inversion method or inverse transform method. It is based on the following deduction. Let U be random number from $\mathcal{U}(0,1)$. Let us compute $\mathrm{F}^{-1}(U)$, where $\mathrm{F}^{-1}(\cdot)$ is the inverse cumulative probability function of the desired distribution. If we compute the cumulative probability of $\mathrm{F}^{-1}(U)$ being less than x, we can see that

$$\mathrm{P}\left(\mathrm{F}^{-1}(U) \leq x\right) \Leftrightarrow \mathrm{P}\left(\mathrm{F}\left(\mathrm{F}^{-1}(U)\right) \leq \mathrm{F}(x)\right) \text{ [by applying F to both sides]}$$
$$\Leftrightarrow \mathrm{P}\left(U \leq \mathrm{F}(x)\right) \Leftrightarrow \mathrm{F}(x) \quad (8.1)$$

because U is uniform, so the probability of $U \leq y$ is y when y is between zero and one. A graphical example is shown in Fig. 8.1.

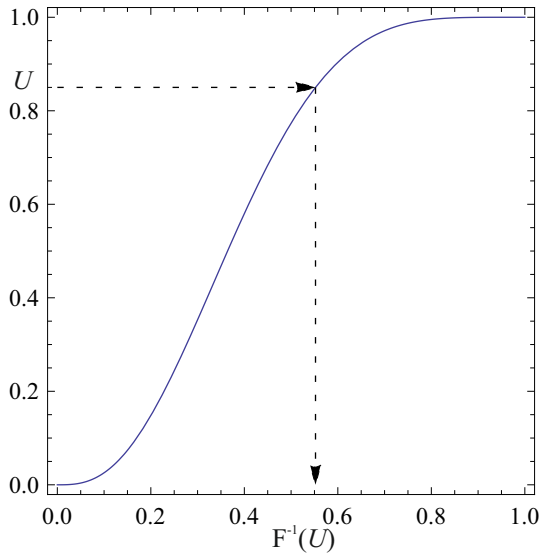

Figure 8.1: Example of the inverse transform method with Beta(3,5)-distribution.

The inverse transform method is valid, in theory, for all distributions. The problem is that the inverse cdf does not exist in closed from for all the distributions, for example the cdf and the inverse cdf for normal distribution are not closed-form functions.

8.1.2 Normal distribution

Random numbers from standard normal distribution can be generated with the special transformation, the Box-Muller algorithm. For two uniform numbers U_1, U_2 it holds that the transformation

$$X_1 = \sqrt{-2\log(U_1)}\cos(2\pi U_2), \quad X_2 = \sqrt{-2\log(U_1)}\sin(2\pi U_2) \tag{8.2}$$

produces X_1 and X_2 that are independent and have standard normal distribution. In Sec. 6.1.1 we introduced how to create correlated values from multinormal distribution, but for two-dimensional multinormal distribution there is a shortcut with the Seppo Mustonen -algorithm. It is the same transform to X_1 as the Box-Muller, but X_2 is computed by

$$X_2 = \sqrt{-2\log(U_1)}\sin(2\pi U_2 + \arcsin(\rho)), \tag{8.3}$$

where ρ is the correlation coefficient between X_1 and X_2.

Other special transforms exists, and some of them are based on the way the distribution is originally derived. For example, as we know that sum of squared normal variables has χ^2-distribution, random numbers from χ^2 can be derived simply just by first creating normal random numbers and the summing their squares. Often these kind of transformations are inefficient when the parameters require a lot of source variables per one outcome.

8.1.3 Accept-Reject method

The Accept-Reject method is based on the creation of random coordinates uniformly inside an area (in 2-D) that bounds the pdf of the target distribution. If the coordinate is inside the area bounded by the target pdf, it's x-coordinate is accepted as a random number from the distribution. If not, it is rejected and a new coordinate is created.

The most simple application of the accept-reject method is the 'box-counting' version where random coordinates are created inside the rectangular area that holds the target pdf inside. For this to work, the target pdf must have finite support. For example, the Beta distribution is defined between 0 and 1 — example of box-counting accept-reject algorithm for Beta distribution is shown in Fig. 8.2.

The box-counting comes less effective in multiple dimensions and in cases where the support of the distribution is very wide, because the number of the rejected point grows. The effectiveness can be improved by finding an envelope that has smaller reject-area outside the target pdf than the rectangle. In general, any 'envelope pdf' $g(\cdot)$ can be used in accept-reject method, if only we can find constant c so that

$$\mathrm{f}(x) \leq c\,\mathrm{g}(x) \quad \forall x \tag{8.4}$$

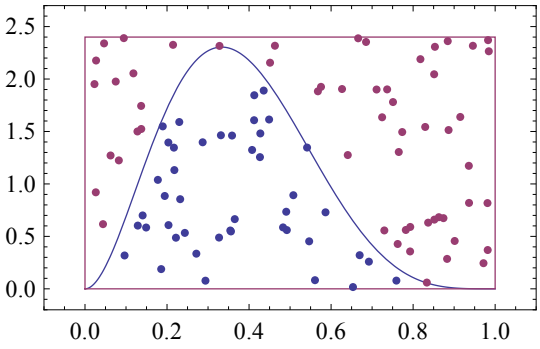

Figure 8.2: Beta(3,5)-distribution and box-counting accept-reject algorithm. The x-coordinates of the blue points have the desired distribution.

If this condition can be fulfilled, the X that is generated from the envelope pdf g can be accepted if

$$U \leq \frac{\mathrm{f}(X)}{c\,\mathrm{g}(X)}, \tag{8.5}$$

where U is from $\mathcal{U}(0,1)$, and rejected otherwise. The box-counting is a simple version of this where the envelope is also a uniform distribution, so that $c\,\mathrm{g}(x) = c$. If the envelope is very close to the target distribution, only a small fraction of the random numbers must be rejected. For the envelope method to work we naturally need to have such a distribution g that it is easy to create random numbers from it.

The Gamma distribution is one example of a distribution that can be simulated by the envelope accpet-reject algorithm efficiently. The trick is that Gamma(α,β)-variables are easy and fast to create if α is an integer. For other α, the Gamma distribution with integer α can be used as an envelope with suitable choice of β and constant c. Example is shown in Fig. 8.3.

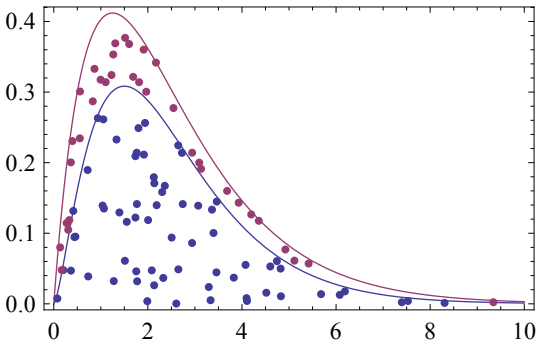

Figure 8.3: Gamma(2.5,1) distribution and a suitable envelope for envelope accept-reject algorithm.

8.2 Resampling methods

The resampling methods are procedures that recycle the existing data in some random manner, i.e. draw random (re)samples of the data. If the original sample is a good representation of the unknown sample space, then the random resamples also estimate the properties of the sample space well. Resampling methods have troubles with small and biased samples, but then again, this is true for more or less all the statistical methods. We will introduce bootstrap, permutation tests and cross-validation here. The so-called jackknife is also a resampling method for variance estimation, but the bootstrap is more general and preferable in many cases, so the jackknife method is not dealt here.

8.2.1 Bootstrap

The bootstrap method was developed by Bradley Efron in 1979 as an extension to jackknife. The name refers to phrase "pull oneself up from one's bootstraps", and suits the method quite well. The initial situation for bootstrap is that we have only one random sample of the interesting phenomena, $\boldsymbol{y}$, and no other information. However, the sample should represent the total sample space. If so, we could draw new samples $\boldsymbol{y}_i^*$ from $\boldsymbol{y}$, and they should also represent the sample space. These resamples should be drawn *with replacement* from the original sample, and have the same size.

From the original sample we can compute a value for an estimator of interest, $\hat{\theta}$. With bootstrap we can asses the uncertainty, e.g. the variance or the confidence intervals, of the estimator. If we compute the same estimator value for every bootstrap sample, θ_i^*, the empirical distribution of θ_i^*'s should estimate the true distribution of $\hat{\theta}$. The inference about $\hat{\theta}$ can be made based on the empirical distribution by descriptive statistics.

For example, we have a sample of 10 numbers from the exponential distribution in Fig. 8.4. The mean $\overline{y}$ is 1.09. Without knowing that the underlying distribution is exponential, one could compute the variance or the confidence interval to the mean using normal approximation. The resulting CI will be symmetrical about the mean. However, exponential distribution is skewed to right, and thus the real CI is not symmetrical.

The histogram in Fig. 8.4 is drawn from the 40,000 means computed from 40,000 bootstrap samples of $\boldsymbol{y}$. Their distribution is slightly skewed to right, as it should be. The bootstrap CI can be computed from the ordered values of $\overline{y}_i^*$. For 95 % CI we will take the 1,000th (2.5 %) value and the 39,000th (97.5 %) value of sorted bootstrap means, and end up with a CI of $(\overline{y}^*_{[1000]}, \overline{y}^*_{[39000]}) = (0.507, 1.879)$. This CI is shown in the figure with gray vertical lines, and it is clearly nonsymmetric around the mean with red vertical line.

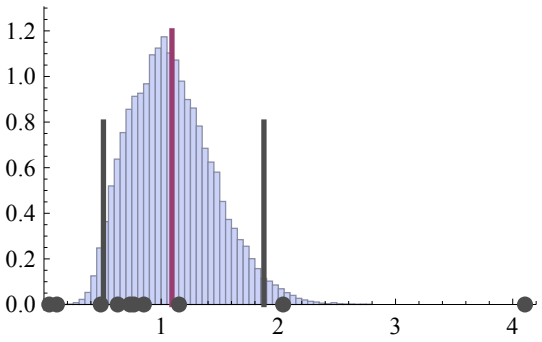

Figure 8.4: Ten samples (gray dots), their mean (red line), histogram of 40,000 bootstrap means and bootstrap CI for the mean (gray lines).

The great advantage with bootstrap is that variances or CI's can be derived for any estimator as easily as for mean, for example for median. The number of bootstrap samples should be large in order to smooth out the finite sampling effects. What is 'large' depends on the problem, but with modern computers the speed is usually not an issue, so 10,000, 50,000 or even 100,000 could be used as the number of bootstrap samples.

Bootstrap can be used also in regression problems (LM and NLM), but then the bootstrap sampling should be used for *residuals* instead of the original data. The procedure is such that first the standard LM or NLM is fitted, and estimates $\boldsymbol{b}$, fitted values $\hat{y}_i$ and residuals e_i are received. Then, bootstrap dataset is formed by adding randomly chosen residual value e_j to each $\hat{y}_i$, thus creating new dataset with $(\boldsymbol{x}_i, \hat{y}_i + e_j)$. The same regression analysis is computed, and bootstrap values $\boldsymbol{b}_k^*$ are received. This is repeated, and inference is based on the distribution of $\boldsymbol{b}_k^*$'s.

8.2.2 Cross-validation

Resampling methods are often quite simple and straightforward ideas that are easy to implement if only the computing power is not an issue. This is true with the bootstrapping, and is especially true with the cross-validation.

Cross-validation is practical with methods involving some kind of prediction, and the accuracy of the prediction interest us. For example, LM or NLM can be used to predict the values of the dependent variable for given explanatory variable. The CI of the prediction can be computed using the residual variance of the model, i.e. the observed errors. However, the residual variance gives too optimistic estimate. The model is fitted to exactly the same observations from which the residuals are computed, and this introduces *overfitting* if we consider new observations. Actually, this can be taken into account analytically in LM, but in NLM or in general linear models this is not possible.

Another example is a classification procedure. Let us say that we have a dataset and we use that to form (i.e. train) our classification scheme. We can try to estimate the

error rate the classifier does by letting it classify our training data, but again, the estimate will be too optimistic because the classification is tuned with exactly these data.

The solution is to leave out one part of the data from the model estimation, and use the model to predict the values for the left-out data. The prediction error is then computed using the errors computed with the left-out data. The usual problem is, of course, that we seldom have huge amounts of data available, and the fitted model will perform worse when estimated with smaller training data than with all the available data. The cross-validation, especially with the so-called *leave-one-out* procedure, is the best compromise between large training set and realistic error estimation. In leave-one-out, one repeatedly leaves one observation out from the training set, estimates the model, and computes the prediction error for the one left-out observation. This is then repeated for all the observations, or at least for a large number, and the mean prediction error is computed from these numbers.

Cross-validation can be done for larger dedicated data than one (k-fold cross-validation), but usually the leave-one-out is the most accurate estimate.

8.2.3 Permutation tests

Permutation tests can be used in cases where we have two or more datasets, and the null hypothesis claims that these should come from the same distribution. A test for medians can be used as an example. Let us have two sets, $\boldsymbol{y}$ and $\boldsymbol{x}$, and we want to test if the medians of the groups are the same with certain statistical significance. The null hypothesis for this test is that the medians are the same.

If the null hypothesis is correct, we could divide the data randomly into new groups $\boldsymbol{y}_i^*$ and $\boldsymbol{x}_i^*$ with the sizes $n_y$and n_x. The difference between the medians is recorded. Again, if the null hypothesis is correct, the difference between the medians between the original sets, $\hat{d} = m_y - m_x$, should be 'common' in the set of all median differences d_i^* computed from the randomly divided groups. If not, the original division was somehow 'special' and the probability of receiving such groups and such difference in median is very small. In the latter case, the null hypothesis can be rejected.

The decision between 'common' and 'special' can be based on the distribution of d_i^*'s in similar manner as in traditional test theory. The p-value of the test is the proportion of d_i^*'s that are as large or larger than our $\hat{d}$. If the p-value is small, i.e. less than 5 %, the null hypothesis can be rejected.

An example of this kind of permutation test for the medians of two groups is shown in Fig. 8.5. The two datasets have both sizes of 30, and they dome from exponential distributions with intensity λ of 1.0 (group 1) or 2.0 (group 2). The difference between the medians is 0.353. With 40,000 random permutations of the groups we can find that only 2.16 % of the median differences in randomly divided groups have

values larger than 0.353. Therefore, the p-value for one-sided test ($H_1 : m_1 > m_2$) is 2.16 % and for two-sided test ($H_1 : m_1 \neq m_2$) 4.32 %. In both cases, the null hypothesis can be rejected — the groups do not have equal medians.

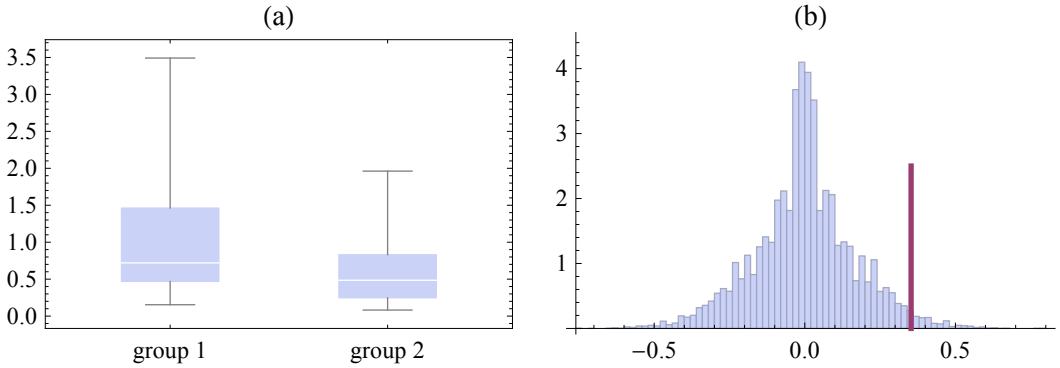

Figure 8.5: Permutation test for equal medians. Box-and-whiskers plot of the two groups in (a), and histogram of the permuted median differences in (b), where red vertical line show the observed median difference.