10

Variance Estimation, the Effective Sample Size, and the Bootstrap

10.1 Introduction

An important statistical consequence of positive spatial autocorrelation is that it inflates the variance of the test statistic, and this may in turn result in an inflated Type I error rate in significance tests (Section 3.3). Let us review and summarize the discussion in that section. Suppose we have obtained a sample $\{Y_1, Y_2, ..., Y_n\}$ from a normally distributed, spatially autocorrelated population, and we are testing the null hypothesis $H_0 : \mu = 0$ against the alternative $H_a : \mu \neq 0$ (Equation 3.6). Recall from Equation 3.9 that the variance σ^2 of the population from which the Y_i are drawn can be estimated by the sample variance s^2, which satisfies

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (Y_i - \bar{Y})^2. \tag{10.1}$$

We use the t statistic (Equation 3.11), $t = \bar{Y} / s\{\bar{Y}\}$, where $s^2\{\bar{Y}\} = s^2 / n$ and $s\{\bar{Y}\} = \sqrt{s^2\{\bar{Y}\}}$, to carry out the test. Reproducing Equation 3.17,

$$\text{var}\{\bar{Y}\} = \frac{\sigma^2}{n} + \frac{2}{n} \sum_{i \neq j} \text{cov}\{Y_i, Y_j\}, \tag{10.2}$$

which indicates that if $\text{cov}\{Y_i, Y_j\} > 0$ (i.e., if the values are autocorrelated), then $s^2\{\bar{Y}\}$ underestimates the true variance of the mean (see Section 3.5.2 for a more complete discussion). Thus, the denominator of the t statistic is smaller than it should be, so that the value of t is artificially inflated, resulting in an increased probability of rejecting the null hypothesis (i.e., an increased Type I error rate).

Cressie (1991, p. 14) provides a very simple example of this effect that illustrates some important points. Suppose the Y_i represent the values of an autocorrelated random process that takes place along a line, with the n sites arranged in numerical order. Suppose the process is such that the covariance between the values of any two sites is given by

$$\text{cov}\{Y_i, Y_j\} = \sigma^2 \rho^{|i-j|}, \; i = 1, 2, ..., n, \tag{10.3}$$

where $0 < \rho < 1$. In this case, the variance of the mean can be worked out exactly. It turns out (Cressie, 1991, p. 14) that it satisfies

$$\text{var}\{\bar{Y}\} = \frac{\sigma^2}{n}\left[1 + \left(\frac{2\rho}{1-\rho}\right)\left(1 - \frac{1}{n}\right) - \frac{2}{n}\left(\frac{\rho}{1-\rho}\right)^2\left(1 - \rho^{n-1}\right)\right]. \tag{10.4}$$

Figure 10.1a shows a plot of $\text{var}\{\bar{Y}\}$ in Equation 10.4 as a function of ρ for $n = 25$ and $\sigma^2 = 1$. As expected, $\text{var}\{\bar{Y}\}$ increases with increasing ρ, slowly at first but dramatically for values of ρ near 1.

There is a second way to view the effect of positive autocorrelation on the variance of the mean. Recall that an intuitive explanation of the inflated Type I error rate in a significance test involving positively autocorrelated data is that each data value contains information about the values of surrounding data, and therefore cannot be counted as a single independent sample. Thus, the *effective sample size* is less than the sample size n. For given values of the population variance σ^2 of the Y and variance of the mean $\text{var}\{\bar{Y}\}$, we can make this notion precise by defining the effective sample size n_e to be the value that satisfies the equation

$$\text{var}\{\bar{Y}\} = \frac{\sigma^2}{n_e}. \tag{10.5}$$

In the simple example of Equation 10.4, the effective sample size can be computed exactly. It has the form (Cressie, 1991, p. 15)

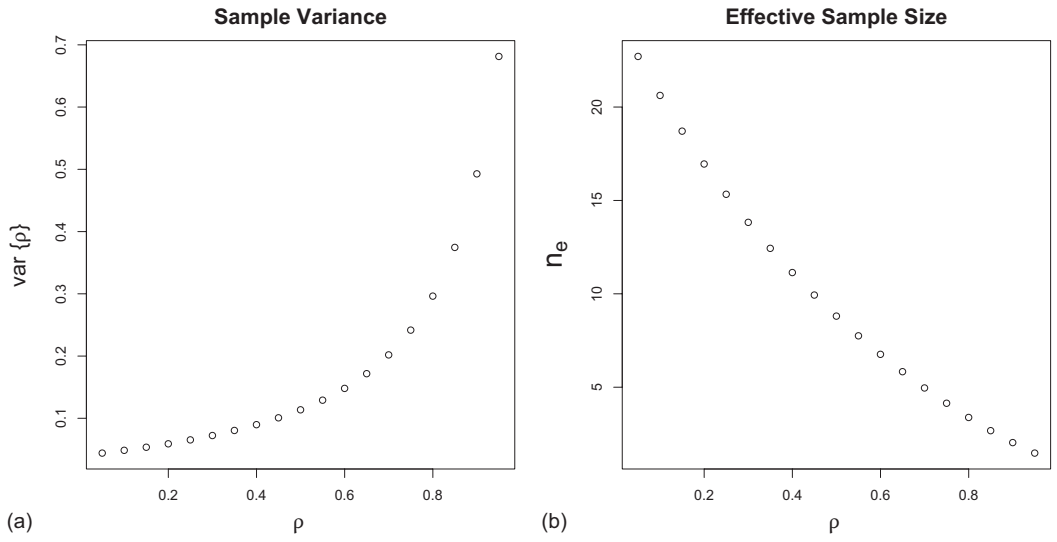

FIGURE 10.1
(a) Variance of the mean defined by Equation 10.3 for $n = 25$ and $\sigma^2 = 1$; (b) Effective sample size for the same values.

$$n_e = \frac{\sigma^2}{\text{var}\{\bar{Y}\}} = n\left[1+\left(\frac{2\rho}{1-\rho}\right)\left(1-\frac{1}{n}\right)-\frac{2}{n}\left(\frac{\rho}{1-\rho}\right)^2\left(1-\rho^{n-1}\right)\right]^{-1}. \tag{10.6}$$

The effective sample size for this example is shown in Figure 10.1b. It declines from a value of 25 for $\rho = 0$ to a value of approximately one for ρ near one.

Now let us consider the general case, in which we are given a sample $\{Y_1, Y_2, ..., Y_n\}$ of spatially autocorrelated data. If there is substantial spatial autocorrelation, then we know that $s^2\{\bar{Y}\}$ may underestimate the true variance of the mean. Suppose, however, that we could obtain a second, independent estimate $\text{var}_{est}\{\bar{Y}\}$ of the variance of the mean. If this second estimate is more accurate than $s^2\{\bar{Y}\}$, then we can insert $\text{var}_{est}\{\bar{Y}\}$ into Equation 10.5 to obtain an estimate $\hat{n}_e$ of the effective sample size n_e. We define $\hat{n}_e$ to be the value satisfying

$$\text{var}_{est}\{\bar{Y}\} = \frac{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}{\hat{n}_e}. \tag{10.7}$$

In other words, $\hat{n}_e$ is our estimate of the sample size that would yield a variance $s^2\{\bar{Y}\}$ if the data were independent rather than spatially autocorrelated. Solving for $\hat{n}_e$ yields

$$\hat{n}_e = \frac{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}{\text{var}_{est}\{\bar{Y}\}}. \tag{10.8}$$

As an example of the effect of spatial autocorrelation on the results of a hypothesis test involving real data, consider EM38 electrical conductivity measurements taken in the furrows and beds of Field 1 of Data Set 4 on two separate dates, April 25, 1995, and May 20, 1995. As usual, we create a data frame `data.Set4.1` from the contents of the file *Set4.196sample.csv* as described in Appendix B.4. Some measurements were not made on April 25, so these data records will be eliminated.

```
> EM425B <- data.Set4.1$EM38B425[!is.na(data.Set4.1$EM38B425)]
> EM520F <- data.Set4.1$EM38F520[!is.na(data.Set4.1$EM38B425)]
```

It happens that the April 25 bed readings are very similar to the furrow readings made on May 20. Here is the code to plot histograms of these two data sets.

```
> h1 <- hist(EM425B, breaks = seq(50,100,5), plot = FALSE)
> h2 <- hist(EM520F, breaks = seq(50,100,5), plot = FALSE)
> plot(h1$mids,h1$density, xlim = c(50,100), ylim = c(0,0.06),
+     type = "o", cex.main = 2, cex.lab = 1.5, # Fig. 10.2
+     xlab = "EC (mS/M)", ylab = "Frequency",
+     main = "Histograms of EM38 Readings")
> lines(h2$mids,h2$density, xlim = c(50,100),
+     type = "o", lty = 2)
> legend(80, 0.05, c("4/25 Bed", "5/20 Furrow"), lty = c(1,2))
```

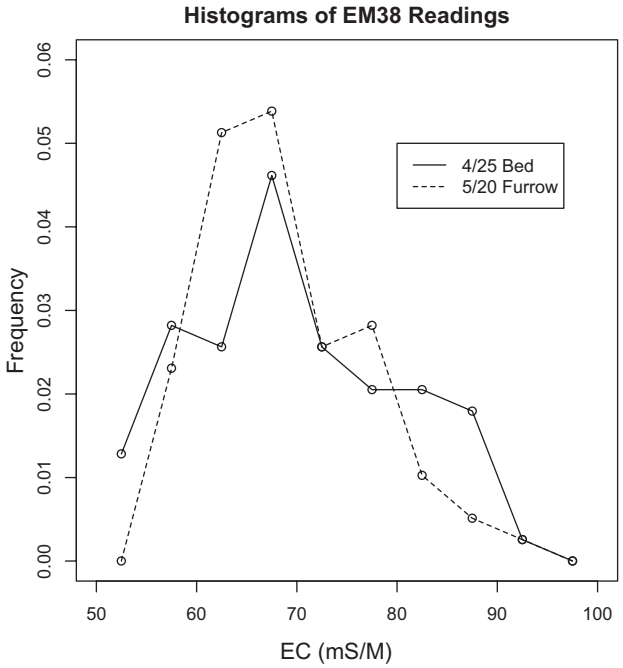

FIGURE 10.2
Histograms of the EM38 samples from Field 4.1.

The result is shown in Figure 10.2. The means of the April 25 bed and May 20 furrow EM38 readings are 70.38 and 69.24 mS/M, respectively. Based on this small difference and on the histograms in Figure 10.2 one might expect that a test of the null hypothesis that the two means are equal would not be rejected. Here are the results of a paired t test of the null hypothesis of equality.

```
> t.test(EM425B,EM520F, paired = TRUE, alternative = "two.sided")
        Paired t-test
data: EM425B and EM520F
t = 1.9629, df = 77, p-value = 0.05327
```

The test returns a suspiciously low p value, considering the variability displayed in Figure 10.2, and we might suspect that spatial autocorrelation has reduced the effective sample size.

We can easily compute the value of s^2 in Equation 10.1: for the paired t test it is the variance of the difference between the two data vectors, which is $s^2 = 26.37$. This is the variance estimate assuming that the EM38 readings are spatially independent. However, in order to obtain the value of n_e we need the second estimate $\text{var}_{est}\{\overline{Y}\}$ of the variance of the mean. This chapter discusses a simple and often remarkably effective resampling method for doing this called the *bootstrap*. The name comes from the phrase "pulling oneself up by one's bootstraps," because this is what the bootstrap method effectively does. The method was developed by Bradley Efron (1979), who showed that by resampling a data set many times, one could generate an approximation of the population distribution of that data set that is sufficiently accurate to permit an estimate of the variance of the mean (as well as other statistics). The basic bootstrap is described in the next section. A complication arises when dealing with autocorrelated data in that the basic bootstrap does not preserve the correlation structure

of the data. Section 10.3 introduces the bootstrap for autocorrelated data by describing two methods for dealing with time series data, and Section 10.4 extends the ideas developed in Section 10.3 to spatially autocorrelated data. This chapter is a bit of a detour in that none of the analyses contribute directly to the resolution of any of the research problems posed for the four data sets. Rather, the discussion introduces a powerful method that we will use to good effect in subsequent chapters. Section 10.5 provides a resolution to the EM38 comparison problem that we use as an illustration throughout this chapter.

10.2 Bootstrap Estimation of the Standard Error

The bootstrap method (Efron, 1979) is described by Efron and Tibshirani (1993). It is a remarkably robust method of estimating variance statistics from the sample data itself. The reasoning behind the method is that the sample provides the best available information about the population itself. Therefore, to estimate statistics associated with the variability of the population, one draws a sample from the sample, that is, a *resample*, and computes the statistics associated with the variability of this resample. Bootstrapping can be used to estimate a variety of statistics associated with distributions; in this chapter, we will use it to estimate the standard error.

In order to demonstrate the bootstrap, we will begin with a finite population whose parameters are known exactly, and whose standard error $\sigma/\sqrt{n}$ we can estimate as $s/\sqrt{n}$. The objective will be to show that we can generate a standard error estimate as good as $s/\sqrt{n}$ directly from the data. Our population has a size of 1,000 and is generated from a lognormal distribution. The lognormal distribution is chosen because it is very highly skewed, and thus far from normal, as can be seen in the histogram of Figure 10.3a.

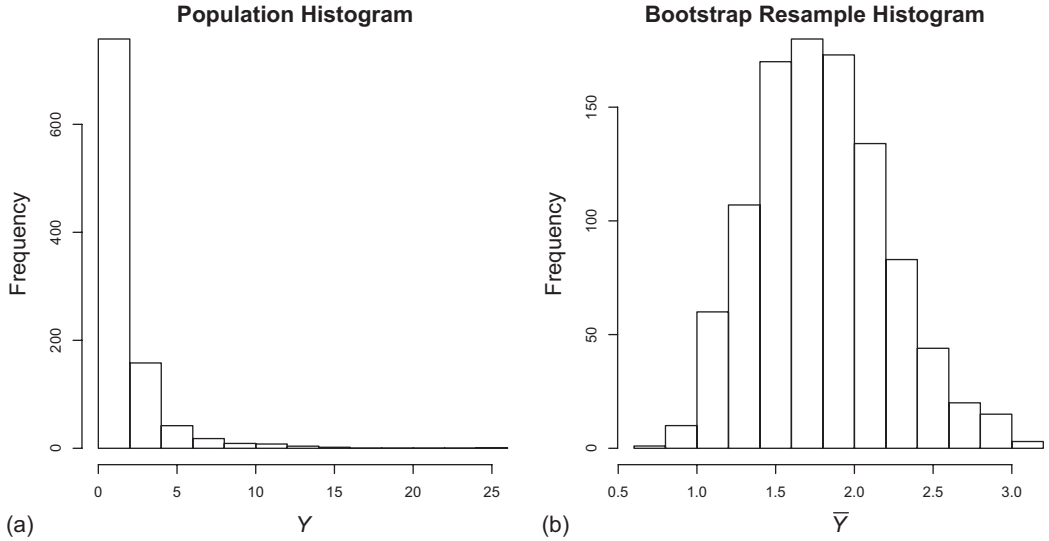

FIGURE 10.3
(a) Histogram of the finite population of size 1,000 used to demonstrate the bootstrap; (b) Histogram of the means of the bootstrap resamples.

```
> set.seed(123)
> pop.Y <- rlnorm(1000)
> print(Y.bar <- mean(pop.Y))
[1] 1.671687
> print(pop.var <- sum((pop.Y - Y.bar)^2) / 1000)
[1] 4.44692
```

The parameters of the population are $\mu = 1.671687$ and $\sigma^2 = 4.44692$.

We draw a sample of size 20 (without replacement) from the population. The variability parameter we will be interested in is the standard error $\sigma/\sqrt{n}$, which is estimated by the sample standard error $se = sd\{\bar{Y}\} = s/\sqrt{n}$.

```
> set.seed(123)
> print(sample.Y <- sample(pop.Y, size = 20), digits = 3)
 [1] 5.160 0.141 0.583 1.611 0.503 0.325 3.048 0.354 1.896 1.545
[11] 7.439 0.128 2.747 1.106 1.293 1.576 4.464 0.812 0.262 0.914
> print(mean(sample.Y), digits = 3)
[1] 1.80
> print(sd(sample.Y) / sqrt(20), digits = 3)
[1] 0.431
> print(sqrt(pop.var / 20), digits = 3)
[1] 0.472
```

This sample contains the best information available about the population. The sample mean is $\bar{Y} = 1.80$ and the sample standard error is $s\{\bar{Y}\} = 0.431$. The true value of $\sigma/\sqrt{n}$ is 0.472.

To apply the bootstrap method, we repeatedly, a total of B times, sample *with replacement* from the sample. Each "sample of the sample," or resample, has the same size as the original sample (in our example, this size is 20). The sampling must be done with replacement, because otherwise each resample would simply consist of a reordering of the 20 original values. Let the samples be denoted $S_i(Y)$, $i = 1,...,B$. Let $\bar{S} = \Sigma S_i(Y)/B$ be the mean of the bootstrap resamples. The bootstrap estimate $\hat{s}_{boot}\{\bar{Y}\}$ of the standard error is then obtained as the standard error of the bootstrap resamples, and is given by the formula (Efron and Tibshirani, 1993, p. 13).

$$\hat{s}_{boot}\{\bar{Y}\} = \sqrt{\frac{\sum_{i=1}^{B}(S_i(Y) - \bar{S})^2}{B-1}}. \tag{10.9}$$

If this seems a bit confusing, read on to the end of the section and then come back.

The principle of applying the formula for the standard error to the resampled data is known as the *plug-in principle* (Efron and Tibshirani, 1993, p. 35; Sprent, 1998, p. 35). There is some discussion over whether the number in the denominator should be B or $B-1$, but, as Efron and Tibshirani (1993, p. 43) point out, it usually doesn't make much difference. Here are the results of three successive resamples. The function `boot.sample()` is created in the first two lines to draw the resample.

```
> boot.sample <- function(x) sample(x, size = length(x),
+     replace = TRUE)
> # One bootstrap resample
> print(b <- boot.sample(sample.Y), digits = 3)
 [1] 0.812 1.106 2.747 0.914 1.106 1.293 7.439 0.128 0.325 0.583
```

```
[11] 0.914 0.262 1.106 1.576 5.160 1.545 1.576 0.503 3.048 0.503
> print(mean(b), digits = 3)
[1] 1.63
> # A second bootstrap resample
> print(b <- boot.sample(sample.Y), digits = 3)
  [1] 0.583 1.896 1.896 0.354 1.611 0.583 0.503 1.545 0.325 0.812
[11] 5.160 1.896 1.576 0.583 0.128 0.503 0.583 1.576 0.812 0.354
> print(mean(b), digits = 3)
[1] 1.16
> # A third bootstrap resample
> print(b <- boot.sample(sample.Y), digits = 3)
 [1] 1.106 0.141 0.354 0.325 4.464 1.896 4.464 4.464 1.576 1.896
[11] 1.576 2.747 1.293 5.160 1.545 0.503 0.354 2.747 0.354 0.583
> print(mean(b), digits = 3)
[1] 1.88
```

You can see that each mean of the resamples is different, and so we can generate a lot of them and compute the variance. The square root of this variance is the bootstrap estimate of the standard error.

Now we are ready to draw a large number of resamples. As with the Monte Carlo simulations done in earlier chapters, we can accomplish the repeated resampling in R by creating a function and then replicating that function. Unlike Monte Carlo simulation, we don't usually need a really large number of resamples; Efron and Tibshirani (1993, p. 54) state that 200 is in almost all cases sufficient to achieve an accurate estimate of the standard error. We will start with 1000 to be on the safe side. The replicated resampling is shown here.

```
> sample.mean <- function(x){
+     Y <- sample(x, size = length(x), replace = TRUE)
+     mean(Y)
+ }
> set.seed(123)
> boot.dist <- replicate(1000, sample.mean(sample.Y))
> print(mean(boot.dist), digits = 3)
[1] 1.79
> print(sd(boot.dist), digits = 3)
[1] 0.419
```

Figure 10.3b shows the histogram of the distribution of means of the resamples. As expected based on the central limit theorem (Larsen and Marx, 1986, p. 322), the resampling distribution is approximately normal, even though the population distribution is lognormal. Also as expected from the central limit theorem, the mean of the means of the resamples is not a very good estimate of the population mean, but it is a good estimate of the sample mean. The standard error of the resample of means, however, does provide an estimate of the population standard error of 0.472 that is almost as good as the sample standard error of 0.431. This illustrates the remarkable fact that simply sampling with replacement many times from an original sample (i.e., by resampling), one can generate a surprisingly accurate estimate of the standard error, and therefore of the variance, of the population.

The bootstrap doesn't always work, but it is often very effective. The accuracy of the result is constrained by the extent to which the sample represents the population. The advantage of the method might not be evident from this example, since the standard error can easily be calculated directly from the data and is a good estimate of the standard deviation of the mean, but such an easy calculation is not always possible. The great advantage

of the bootstrap is that it can be applied in cases where there is no easy way to estimate the standard deviation of the mean directly.

It is not necessary to write one's own function for bootstrapping: R has the function `boot()` in the package `boot` (Canty and Riplyy, 2017) that carries out a bootstrap estimate of the standard error.

```
> library(boot)
> set.seed(123)
> mean.Y <- function(Y,i) mean(Y[i])
> boot.Y <- boot(sample.Y, mean.Y, R = 1000)
> print(sd(boot.Y$t), digits = 3)
[1] 0.421
```

The function `boot()` typically employs a user defined function as shown. There is one feature that must be explained. In the third line, the function `mean.Y()` is defined so it can be passed as an argument of the function `boot()` in the fourth line. In addition to the first argument `Y` of the function `mean.Y()`, which identifies an array consisting of the data to be resampled, the definition of the function *must include* a second argument that is an index, in this case `i`, of this array. The index argument must be represented in the function definition as shown in line 3.

There are two sources of randomness in the bootstrap, the original random sample from the population and the randomness of the resampling. In the example above, the first source of randomness led to the large difference between the sample mean and the population mean. The difference between the estimated standard errors of the homemade bootstrap and the function `boot()` illustrates the second effect. When a reasonably large number of resamples are computed, the main source of variability is the first of the two, which depends on the sample size. It is not necessarily the case that increasing the number of resamples increases the accuracy of the bootstrap estimate (Exercise 10.2). This is particularly true for small sample sizes (i.e., small values of n), because a small sample may not be very representative of the population.

Finally, recall that there is another statistical method, the permutation method, in which the sample is repeatedly resampled (Section 3.3). Let us compare the two methods so that we can eliminate confusion between them. In Section 3.3, the permutation test was introduced as a way to test the hypothesis that two samples come from the same population. In Section 4.3, the method was used by the function `joincount.mc()` to test the null hypothesis that high weed levels are not spatially autocorrelated against the alternative hypothesis that they are positively autocorrelated, and in Section 4.4 it was used by the function `moran.mc()` to carry out the same test on detrended sand content data. The permutation test involves sampling the data values *without replacement* in order to rearrange their configuration. One computes the statistic for each rearrangement, and then tabulates the number of rearrangements for which the computed value is more extreme than that of the observed data. If the observed statistic is more extreme than all but a few of the rearrangements, then we reject the null hypothesis. If it is more of a "typical" value, then we cannot reject. Thus, the permutation method involves repeated sampling of the data without replacement in order to rearrange them and in so doing carry out a statistical test in which the null hypothesis is that the data are randomly arranged. The bootstrap method involves repeated sampling of the data *with* replacement to estimate the variability associated with a statistic. For a more in-depth comparison of the bootstrap and permutation methods, see Efron and Tibshirani (1993, p. 202).

10.3 Bootstrapping Time Series Data

10.3.1 The Problem with Correlated Data

In Section 3.4.1, we studied a model of the form

$$\begin{aligned} Y_i &= \mu + \eta_i, \\ \eta_i &= \lambda \eta_{i-1} + \varepsilon_i,\ i = 2, 3, \ldots, \end{aligned} \tag{10.10}$$

where Y_1 and η_1 are given, the $\varepsilon_i \sim N(0, \sigma^2)$ are independent, and $-1 < \lambda < 1$. When $\lambda > 0$, a test of the null hypothesis that the mean μ of the distribution of the Y_i is zero had an inflated Type I error rate. The variance of the Y_i, and therefore the standard error, is inflated because the Y_i are autocorrelated, and therefore the t statistic based on the assumption of independent Y_i is artificially high, so that a t test using this statistic will have an increased probability of rejecting the null hypothesis. Based on the discussion in the previous section it seems like it might be possible to construct a bootstrap estimate of the standard error and thus obtain a test with an improved Type I error rate. This idea is explored in the present section. We first verify that the bootstrap produces an adequate t statistic with uncorrelated data. Here is a t test of a random sample of size 100.

```
> set.seed(123)
> Y <- rnorm(100)
> t.test(Y, alternative = "two.sided")$p.value
[1] 0.3243898
```

Now we carry out the test on the same data set using a bootstrap estimate of the standard error, computing the t statistic by hand and using the function `pt()` to compute the p value.

```
> library(boot)
> mean.Y <- function(Y,i) mean(Y[i])
> boot.Y <- boot(Y, mean.Y, R = 1000)
> t.boot <- mean(Y) / sd(boot.Y$t)
> print(p <- 2 * (1 - pt(q = abs(t.boot),
+    df = length(Y) - 1)))
[1] 0.3302481
```

The results are similar.

We now try using the bootstrap estimate of the standard error on a t test involving autocorrelated data. Recall that in Section 3.3 we carried out a Monte Carlo simulation of a test of the null hypothesis $\mu = 0$ in Equation 10.10 in which the value of μ had been set at zero. When λ had the value 0.4, the Type I error rate in 10,000 simulations was 0.1936, well above the α value of 0.05. As we did in Chapter 3, in the present simulation study, we first carry out the simulation with uncorrelated data.

```
> set.seed(123)
> ttest <- function(){
+   Y <- rnorm(100)
+   t.ttest <- t.test(Y,alternative = "two.sided")
+   TypeI <- as.numeric(t.ttest$p.value < 0.05)
+ }
```

```
> U <- replicate(10000, ttest())
> mean(U)
[1] 0.0488
```

Next, we run the same simulation except that we use autocorrelated data. As we did in Section 3.5.1, we drop the first 10 values to avoid the effect of an initial transient. Also, since $\mu = 0$ implies that $Y_i = \eta_i$, we express the model in terms of Y rather than η, writing Equations 10.10 as

$$Y_i = \lambda Y_{i-1} + \varepsilon_i,\ i = 2, 3, \ldots, . \tag{10.11}$$

The value of λ is set to 0.4.

```
> lambda <- 0.4 #Autocorrelation term
> set.seed(123)
> ttest <- function(lambda){
+   Y <- numeric(110)
+   for (i in 2:110) Y[i] <- lambda * Y[i - 1] + rnorm(1, sd = 1)
+   Ysamp <- Y[11:110]
+   t.ttest <- t.test(Ysamp,alternative = "two.sided")
+   TypeI <- as.numeric(t.ttest$p.value < 0.05)
+ }
> U <- replicate(10000, ttest(lambda))
> mean(U)
[1] 0.1944
```

As expected, the Type I error rate is inflated. Now run the same simulation but compute the standard deviation of the t statistic using the bootstrap method. First, we create a function `t.boot()` to carry out a t test using the bootstrap estimate of the standard error. To save time, the bootstrap only uses 200 resamples, but based on the results of Exercise 10.2, this should not cause a serious problem.

```
> t.boot <- function(Y){
+     mean.Y <- function(Y,i) mean(Y[i])
# Bootstrap estimate of the variance
+     boot.Y <- boot(Y, mean.Y, R = 200)
+     se <- sd(boot.Y$t)
+     t.stat <- mean(Y) / se
# t test based on the bootstrap s.e. estimate
+     p <- 2 * (1 - pt(q = abs(t.stat),df = length(Y) - 1))
+     return(c(p, se))
+ }
```

Next, we modify our function `ttest()` to use the bootstrap standard error. We will have the function return the two standard error estimates as well.

```
> ttest <- function(lambda){
+     Y <- numeric(110)
+     for (i in 2:110) Y[i] <- lambda * Y[i - 1] + rnorm(1, sd = 1)
+     Ysamp <- Y[11:110]
+     t.ttest <- t.boot(Ysamp)
+     TypeI <- as.numeric(t.ttest[1] < 0.05)
+     Yse <- sd(Ysamp) / 10
+     Yse.boot <- t.ttest[2]
+     return(c(TypeI, Yse, Yse.boot))
+ }
```

TABLE 10.1

Mean Standard Error and Error Rate for Three Monte Carlo simulations of the Test of the Null Hypothesis of Zero Mean for Equation 10.1, $n = 100$

ρ		0.0	0.2	0.4	0.6	0.8
Uncorrected	mean *se*	0.099	0.101	0.107	0.122	0.157
	E.R.	0.051	0.114	0.206	0.337	0.527
Block bootstrap	mean *se*	0.093	0.113	0.146	0.206	0.346
	E.R.	0.093	0.099	0.109	0.126	0.182
Parametric bootstrap	mean *se*	0.098	0.122	0.161	0.236	0.441
	E.R.	0.059	0.062	0.0680	0.080	0.108

Note: The standard error or independent samples would be $s\{\bar{Y}\} = 1/\sqrt{100} = 0.1$. The symbol *se* represents the bootstrap estimate of the standard error, denoted $\hat{s}_{boot}\{\bar{Y}\}$ in the text (e.g., Equation 10.9). Statistics are based on 10,000 simulations.

Now we are ready to carry out the Monte Carlo simulation.

```
> set.seed(123)
> U <- replicate(10000, ttest(lambda))
> mean(U[1,]) # Error rate
[1] 0.2062
```

The result is disappointing: the Type I error rate produced by the bootstrap standard error estimate is no better than that produced by the ordinary sample standard error estimate. Indeed, as indicated in the first two rows of Table 10.1, the bootstrap standard error does not change with increasing values of the autocorrelation parameter λ, and therefore the error rate increases with λ.

What happened? The problem is that the inflated standard error is due to the autocorrelation structure of the Y_i. By randomly resampling the sequence of Y_i, the bootstrap resamples break up this autocorrelation structure so that the bootstrap resamples are not autocorrelated and therefore reproduce the incorrect estimate of the standard error. We can see this by printing out the average sample standard error and the average of the bootstrap estimate of the standard error.

```
> mean(U[2,]) # Avg std error
[1] 0.1079641
> mean(U[3,]) # Avg bootstrap s.e.
[1] 0.1073135
```

To effectively use the bootstrap method to estimate the standard error for autocorrelated data, some way has to be found to generate bootstrap resamples that preserve the autocorrelation structure of the data. There are two commonly used approaches: the block bootstrap and the parametric bootstrap. We will describe each of these in turn in the next two subsections.

10.3.2 The Block Bootstrap

The problem in the previous subsection is that the ordinary bootstrap disrupts the correlation structure of autocorrelated data. One way to partially preserve this structure is, instead of resampling the sample data for individual values, to resample the data in blocks,

each of some specified length l (Efron and Tibshirani, 1993, p. 99). More complex methods involve moving the block along the time series in a continuous manner (Efron and Tibshirani, 1993, p. 101), but we will not be concerned with these. The advantage of block resampling is that serial autocorrelation is preserved within each block and is only disrupted at the boundaries between the blocks. At these block boundaries, the data will in general be uncorrelated.

To create a block bootstrap system for serially autocorrelated data, the first step is to create the blocks. We will do this by placing the data into a matrix in which each block is a column of the matrix. By default, the function `matrix()` places a vector of values into a matrix by columns. We will take our autocorrelated sample of 100 values from the previous subsection and split it into 10 blocks of 10 values each. The first 10 values of *Y*, which are the initial transient, are ignored.

```
> lambda <- 0.4
> set.seed(123)
> Y <- numeric(110)
> for (i in 2:110) Y[i] <- lambda * Y[i - 1] + rnorm(1, sd = 1)
> Ysamp <- Y[11:110]
> n.blocks <- 10
> blocks <- matrix(Ysamp, ncol = n.blocks)
```

Next, we create a function `block.samp()` to randomly sample the blocks with replacement and covert them into a vector.

```
> block.samp <- function(n.blocks, blocks){
+     cols <- sample(1:ncol(blocks), n.blocks, replace = TRUE)
+     as.vector(blocks[,cols])
+ }
```

Here is a simple example of the application of `block.samp()`.

```
> set.seed(123)
> print(A <- matrix(1:9,3))
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> print(B <- block.samp(6, A))
 [1] 1 2 3 7 8 9 4 5 6 7 8 9 7 8 9 1 2 3
```

Next, we create a function `mean.b.boot()` that calls the function `block.samp()` and computes the mean of the resulting sample.

```
> mean.b.boot <- function(n.blocks, blocks){
+     mean(block.samp(n.blocks, blocks))
+ }
```

For example, if we apply `mean.b.boot()` to the simple matrix A, we get the following result.

```
> set.seed(123)
> mean.b.boot(6, A)
[1] 5.5
> mean(B)
[1] 5.5
```

Recall that the matrix `blocks` contains the same data as the sample `Ysamp` and therefore has the same mean.

```
> mean(Ysamp)
[1] 0.05088889
> mean(blocks)
[1] 0.05088889
```

Therefore, we can conduct a block bootstrap t test by generating a set of B resamples by applying the function `mean.b.boot()` B times and then, using the plug-in principle, estimating the standard error by the standard deviation of the vector of resampled block bootstrap means. We set B at 200.

```
> set.seed(123)
> u <- replicate(200, mean.b.boot(10, blocks))
> t.stat <- mean(blocks) / sd(u)
> print(p <- 2 * (1 - pt(q = abs(t.stat),
+     df = nrow(blocks)*ncol(blocks) - 1)))
[1] 0.772062
```

Now we are ready to construct a Monte Carlo simulation of a t test of autocorrelated data using the block bootstrap. First, we create a function `t.b.boot()` that carries out a single t test using the block bootstrap.

```
> t.b.boot <- function(lambda){
+ # Generate the sample, eliminating the initial transient
+     Y <- numeric(110)
+     for (i in 2:110) Y[i] <- lambda * Y[i - 1] + rnorm(1, sd = 1)
+     Ysamp <- Y[11:110]
+ # Break into blocks
+     n.blocks <- 10
+     blocks <- matrix(Ysamp, ncol = n.blocks)
+ # Compute the bootstrap resample
+     U <- replicate(200, mean.b.boot(n.blocks, blocks))
+ # Carry out the t-test
+     t.stat <- mean(blocks) / sd(U)
+     p <- 2 * (1 - pt(q = abs(t.stat),
+       df = nrow(blocks)*ncol(blocks) - 1))
+     return(c(as.numeric(p < 0.05), sd(U)))
+ }
```

Now we carry out the Monte Carlo simulation.

```
> set.seed(123)
> lambda <- 0.4
> U <- replicate(10000, t.b.boot(lambda))
> mean(U[1,]) # Error rate
[1] 0.1089
> mean(U[2,]) # Mean est. std. error
[1] 0.1459619
```

The results of several such simulations are given in the second row of Table 10.1. As shown in the table, the block bootstrap actually has about the same or even a higher error rate than the uncorrected t test for uncorrelated or weakly autocorrelated data. For values of λ greater

than 0.4, the block bootstrap outperforms the uncorrected t test, although the error rate is still inflated by a factor of two to four. This relatively poor result is due in part to the interruption of the correlation structure on the boundaries of the blocks, and in part to the fact that a relatively small number of independent entities (the blocks) are involved in the bootstrap simulation. There is a trade-off: the smaller the number of boundaries, the smaller the number of independent entities. The parametric bootstrap, discussed in the next section, avoids this problem but does bring another problem of its own.

10.3.3 The Parametric Bootstrap

Efron and Tibshirani (1993, p. 92) describe a second method for structured data such as time series that has come to be called the *parametric bootstrap* (Davison and Hinckley, 1997, p. 389). The reason for this term is that, unlike the block bootstrap, the parametric bootstrap assumes a model for the random process, estimates the parameter or parameters of the model, and uses these estimated parameters to construct the bootstrap resamples. Suppose we have a data sequence $\{Y_1, Y_2, ..., Y_n\}$ that we assume has been generated by a first-order autoregressive process of the form of Equation 10.11, and we wish to test the null hypothesis $\mu = 0$. We can fit the Y_i with the first-order autoregressive model (Equation 10.11) and compute the least squares estimate $\hat{\lambda}$ for λ. Let $\{\hat{\varepsilon}_i\}$ be the set of residuals of this estimate. The $\hat{\varepsilon}_i$ are given by the equation

$$\hat{\varepsilon}_i = Y_i - \hat{\lambda} Y_{i-1},\ i = 1, 2, \ldots . \tag{10.12}$$

The errors ε_i in Equation 10.11 are independent. Even, if we assume (and this is a very important assumption) that the model (Equation 10.11) provides an accurate description of the data set $\{Y_i\}$, the error estimates $\hat{\varepsilon}_i$ in Equation 10.12 will not be independent because they sum to zero. However, the correlation structure associated with Equation 10.11 will be removed. Therefore, one can generate a bootstrap sequence with an error structure that approximates the error structure of the solution of Equation 10.12 by setting Y_1 equal to $\hat{Y}_1$ and then computing the remaining Y_i iteratively according to the equation

$$\hat{Y}_i = \hat{\lambda} \hat{Y}_{i-1} + \tilde{\varepsilon}_i \ \ i = 2, 3, ..., n \tag{10.13}$$

where at each step $\tilde{\varepsilon}_i$ is selected randomly with replacement from the set of random variables $\{\hat{\varepsilon}_i\}$, $i = 1, \ldots, n$. In other words, the bootstrap errors are selected randomly in the same way as the original bootstrap values are selected, by sampling from the original data set with replacement, and then they are fitted into the parametric model of the random process.

R contains built-in facilities for estimating the parameters of time series such as Equation 10.10. These are described by Venables and Ripley (2002, Ch. 14), and the reader is referred to that reference for further discussion. We first use the R function `ts()` to convert the vector `Ysamp` generated in the previous section into an R time series. Then we use the function `arima()` to construct a first-order autoregressive integrated moving average model based on the time series data.

```
> Ysamp.ts <- ts(Ysamp)
> Ysamp.ar <- arima(Ysamp.ts, c(1,0,0))
```

Next, we obtain our estimate $\hat{\lambda}$ and our set of residuals $\hat{\varepsilon}_i$.

```
> print(lambda.hat <- coef(Ysamp.ar)[1], digits = 3)
  ar1
0.401
> e.hat <- residuals(Ysamp.ar)
```

Now we construct a function to generate the simulated time series based on Equation 10.13.

```
> para.samp <- function(Ysamp, e.hat, lambda.hat){
+     Y.sim <- numeric(length(Ysamp))
+     Y.sim[1] <- Ysamp[1]
+     for (i in 2:length(Ysamp)){
+       ei.boot <- sample(e.hat, 1)
+       Y.sim[i] <- lambda.hat * Y.sim[i - 1] + ei.boot
+   }
+   return(mean(Y.sim))
+ }
```

In theory, the series `Y.sim` has the same autocorrelation structure as `Ysamp`, so that it can function as a bootstrap resample. Thus, we can use the function `replicate()` to carry out our bootstrap resampling.

```
> set.seed(123)
> U <- replicate(200,para.samp(Ysamp, e.hat, lambda.hat))
```

Using the plug-in principle, we can obtain an estimate of the standard error as the standard deviation of the mean of the bootstrap resamples.

```
> print(sd(U), digits = 3)
[1] 0.151
```

By comparison, the estimate $s\{\bar{Y}\} = s/\sqrt{n}$, where $n = 100$, gives us the estimate we would expect from an independent sample.

```
> print(sd(Ysamp) / 10, digits = 3)
[1] 0.0986
```

The Monte Carlo simulation of the t test using the parametric bootstrap is set up in the same way as that of the block bootstrap, and the code is not shown (if you try to run this, you will find that it takes a *very* long time). The results are shown in the last two rows of Table 10.1. The parametric bootstrap outperforms the block bootstrap at all values of λ and has a reasonable error rate for all but the highest value.

The success of the parametric bootstrap method may be highly dependent on the adequacy of the parametric model in simulating the data. There has been relatively little study of the effects of misspecification of the model on the accuracy of the parametric bootstrap. It is important to emphasize that the parametric model used in the test whose results are reported in Table 10.1 is the actual model itself, so there is no misspecification error. That is, we are estimating the parameters of the model of a data set that is actually the same as the model used to generate the data set in the first place. The existence of misspecification error in a real application could seriously impact the performance of the method.

10.4 Bootstrapping Spatial Data

10.4.1 The Spatial Block Bootstrap

The preceding section introduced the bootstrap estimate of the standard error of data sets with temporally autocorrelated errors. As with other aspects of the analysis of spatial data, bootstrapping data with spatially autocorrelated errors proceeds in a completely analogous way. Where the time series data is specified by Equation 10.10, the spatial error model is specified by Equation 3.25, which is repeated here as

$$\begin{aligned} Y &= \mu + \eta \\ \eta &= \lambda W \eta + \varepsilon. \end{aligned} \tag{10.14}$$

Assuming that the spatial weights matrix is row normalized (Equation 3.22), the values of λ range between –1 and 1. Once again, we will test the null hypothesis $\mu = 0$ against the alternative $\mu \neq 0$. A spatial block bootstrap can be developed that is precisely analogous to the time series block bootstrap of Section 10.3.

We will proceed in the same manner as we did with time series data in Section 10.3. We will run Monte Carlo simulations of a test of the null hypothesis $\mu = 0$ on an artificial data set satisfying Equation 10.14 in which the value of μ is zero. The data set is generated as described in Section 3.5.3, by first generating a vector ε of independent pseudorandom numbers from a standard normal distribution and then plugging this into Equation 3.26, $\eta = (I - \lambda W)^{-1}\varepsilon$. Under the null hypothesis $\mu = 0$, and therefore the first of Equations 10.14 becomes $Y = \eta$. Therefore, we have

$$Y = (I - \lambda W)^{-1}\varepsilon. \tag{10.15}$$

The transformation of Equation 10.15 is applied to a 24 by 24 lattice and then a two-layer border is trimmed to avoid edge effects. The Monte Carlo simulation is therefore carried out on a 20 by 20 cell square lattice.

We can use the array feature of R to develop our sampling code. We briefly mentioned in Chapter 2 that R is capable of constructing arrays of more than two dimensions. To create the block resampling system, we employ a three-dimensional array of blocks, so that the blocks are stacked on top of each other like a raster brick or a Rubik's cube. We will then sample these blocks with replacement.

We illustrate the method using a 6 by 6 lattice with 2 by 2 = 4 blocks, each of size 3 by 3. The cell numbering is set up as follows.

```
> n.row <- 6
> n.block <- 2
> print(A <- matrix(1:n.row^2, nrow = n.row, byrow = TRUE))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    7    8    9   10   11   12
[3,]   13   14   15   16   17   18
[4,]   19   20   21   22   23   24
[5,]   25   26   27   28   29   30
[6,]   31   32   33   34   35   36
```

This cell numbering scheme mimics the one generally used with data on a lattice, in which the index value starts in the northwest corner and moves across the rows and down the columns. In R, the natural array numbering is by columns, so we will work with the transpose of A, which we define explicitly to increase the transparency of the code.

```
> At <- t(A)
```

The size of each block is 6 / 2 = 3. We define a three-dimensional array B consisting of four 3 by 3 matrices.

```
> blk.size <- n.row / n.block
> B <- array(0, c(blk.size, blk.size, n.block^2))
```

Next, we fill the elements of B with the corresponding elements of the transpose of A. The first 3 by 3 element of B (the upper left corner of the transpose At) is displayed.

```
> k <- 1
> for(i in 1:n.block){
+     for(j in 1:n.block){
+         B[,,k] <- At[(1+(i-1)*blk.size):(i*blk.size),
+           (1+(j-1)*blk.size):(j*blk.size)]
+         k <- k+1
+ }}
> B[,,1]
     [,1]   [,2]  [,3]
[1,]    1      7    13
[2,]    2      8    14
[3,]    3      9    15
```

Next, we create a sample with replacement of the block number, which is the third index of B.

```
> set.seed(123)
> print(i.samp <- sample(1:n.block^2, replace = TRUE))
[1] 2 4 2 4
```

Next, we create another three-dimensional array of the same size as B and fill it with the resampled blocks of B.

```
> Ct <- matrix(0, nrow = n.row, ncol = n.row)
> k <- 1
> for(i in 1:n.block){
+     for(j in 1:n.block){
+         Ct[(1+(i-1)*blk.size):(i*blk.size),
+             (1+(j-1)*blk.size):(j*blk.size)] <- B[,,i.samp[k]]
+         k <- k+1
+ }}
```

Finally, we transpose back to get our block resampled version of the original lattice A.

```
> print(C <- t(Ct))

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   19   20   21   19   20   21
```

```
[2,]   25   26   27   25   26   27
[3,]   31   32   33   31   32   33
[4,]   22   23   24   22   23   24
[5,]   28   29   30   28   29   30
[6,]   34   35   36   34   35   36
```

The blocks are sampled columnwise, which does not affect the bootstrap distribution.

The block bootstrap is implemented by modification of the uncorrected bootstrap analogous to that of the previous section on time series. Code similar to that just displayed is used to construct a three-dimensional array based on the transpose of the data lattice, and the third dimension of the array is then sampled with replacement. A weakness of the block bootstrap, at least as implemented here, is that it is limited to rectangular data lattices.

The Monte Carlo simulation is accomplished by developing a function `t.b.boot()` that is called by the function `replicate()`.

```
> t.b.boot <- function(lambda){
+ # Generate the sample, eliminating the boundary
+    Y <- IrWinv %*% rnorm(24^2)
+    Ysamp <- matrix(Y, nrow = 24, byrow = TRUE)[3:22,3:22]
+ # Break into blocks
+    Yt <- t(Ysamp)
+    n.block <- 4
+    blk.size <- 20 / n.block
+    blocks <- array(0, c(blk.size,blk.size,n.block^2))
+    k <- 1
+    for(i in 1:n.block){
+       for(j in 1:n.block){
+          blocks[,,k] <- Yt[(1+(i-1)*blk.size):(i*blk.size),
+             (1+(j-1)*blk.size):(j*blk.size)]
+          k <- k+1
+    }}
+ #Compute the bootstrap resample
+    U <- replicate(200, mean.b.boot(n.block, blocks))
+ # Carry out the t-test
+    t.stat <- mean(blocks) / sd(U)
+    p <- 2 * (1 - pt(q = abs(t.stat),
+      df = nrow(Ysamp)*ncol(Ysamp) - 1))
+    return(c(as.numeric(p < 0.05), sd(U)))
+ }
```

The function creates the random field on a 24 by 24 and carries out the block resampling on a 20 by 20 sub-grid. It calls the function `mean.b.boot()`, which in turn calls `block.samp()` to generate the block resample and then computes the mean.

```
> mean.b.boot <- function(n.block, blocks){
+    mean(block.samp(n.block, blocks))
+ }
```

The function `block.samp()` uses the matrix code shown above to generate the block resample.

```
> # Block sample function
> block.samp <- function(n.block, blocks){
```

```
+     i.samp <- sample(1:n.block^2, replace = TRUE)
+     Ct <- matrix(0, nrow = 20, ncol = 20)
+     k <- 1
+     blk.size <- 20 / n.block
+     for(i in 1:n.block){
+        for(j in 1:n.block){
+           Ct[(1+(i-1)*blk.size):(i*blk.size),
+              (1+(j-1)*blk.size):(j*blk.size)] <- blocks[,,i.samp[k]]
+           k <- k+1
+     }}
+     return(t(Ct))
+ }
```

Here is a Monte Carlo simulation for $\lambda = 0.4$.

```
> set.seed(123)
> lambda <- 0.4
> nlist <- cell2nb(24, 24)
> IrWinv <- invIrM(nlist, lambda)
> U <- replicate(10000, t.b.boot(lambda))
> mean(U[1,]) # Error rate
[1] 0.1008
> mean(U[2,]) # Mean est. std. error
[1] 0.07199279
```

The results of a set of simulations for increasing values of λ are shown in Table 10.2. Similar to the case with time series data (Table 10.1), the error rate of the block bootstrap corrected test is actually higher than that of the uncorrected bootstrap test for small values of λ. As λ increases the error rate of the block bootstrap corrected test increases, but not as dramatically as that of the uncorrected test. This poor performance of the block bootstrap was also observed by Davison and Hinkley (1997, p. 399). Efron and Tibshirani (1993, p. 101) note that the block size can have a major impact on the performance of the block bootstrap. In addition, Davison and Hinkley (1997, p. 426) point out that one might expect the performance of the spatial block bootstrap to be worse than that of

TABLE 10.2

Mean Standard Error and Error Rate for Three Monte Carlo simulations of the Test of the Null Hypothesis of Zero Mean for Equation 10.1 for a 20 by 20 Spatial Lattice

ρ		0.0	0.2	0.4	0.6	0.8
Uncorrected	mean(se)	0.049	0.050	0.052	0.057	0.070
	E.R.	0.049	0.109	0.198	0.338	0.531
Block bootstrap	mean(se)	0.048	0.057	0.072	0.100	0.171
	E.R.	0.076	0.085	0.100	0.123	0.173
Parametric bootstrap	mean(se)	0.050	0.061	0.080	0.116	0.220
	E.R.	0.053	0.055	0.058	0.064	0.073

Note: The standard error for independent samples would be $se = 1/\sqrt{400} = 0.05$. The symbol *se* represents the bootstrap estimate of the standard error, denoted $\hat{s}_{boot}\{\bar{Y}\}$ in the text (e.g., Equation 10.9). Statistics are based on 10,000 simulations. The block bootstrap uses 25 blocks, each of size 4 by 4.

the time series block bootstrap for data sets of the same size. Comparing Tables 10.1 and 10.2 indicates a slightly poorer performance of the spatial block bootstrap. The reason for this is related to a phenomenon known as the "curse of dimensionality" (Bellman, 1957), discussed in this context by Hastie et al. (2017, p. 23). The problem is that as the dimension of the space in which the sampling takes place increases, a greater portion of the sample region near a block boundary also increases. Intuition for this can be gained by printing out a vector representing an individual block of the time series and a matrix representing a block of the spatial block bootstrap. We will use a total of 400 data records and 16 blocks. Here is sequence of random numbers that simulates a time series of length 400 / 16 = 25.

```
> print(rnorm(25), digits = 2)
 [1]   0.450   2.092  0.035  0.733 0.118   0.772  1.837 1.376  -0.544
[10]  -1.069  -0.430 -0.111 -0.034 0.629  -0.298 -0.271 0.988  -0.216
[19]   0.755  -0.683 -0.729 -1.018 0.249  -0.721  0.649
```

Here is matrix of random numbers that simulates a single simulated spatial block.

```
> print(matrix(rnorm(25), nrow = 5), digits = 2)
      [,1]    [,2]   [,3]   [,4]    [,5]
[1,] -1.54  -0.031  2.752  -1.59 -0.0094
[2,]  1.22  -0.574 -0.127  -0.15  0.6734
[3,] -3.28  -0.941 -1.663   0.35 -1.2237
[4,] -0.15  -1.053 -0.092  -1.96 -2.2888
[5,] -2.96  -0.122 -0.434   0.46  0.8126
```

Recall that the autocorrelation structure of the data is disrupted at the block boundaries. Of the 25 points in the simulated time series block, only 2 (the first and last in the sequence) are on the boundary. The simulated spatial data block has 16 boundary points and only 9 interior points. Therefore, a larger fraction of the data records is associated with disrupted autocorrelation structure in the case of the spatial block bootstrap than in that of the time series block bootstrap.

10.4.2 The Parametric Spatial Bootstrap

One can construct a parametric bootstrap for spatial data, analogous to that constructed for time series data in Section 10.3.3. As in the time series case, one fits a parametric model to the data and then samples with replacement from the residuals of the model to generate the bootstrap resamples. The parametric bootstrap procedure for a spatial error model (Section 3.5.3; these will be discussed in Chapter 13) is to fit the data to a model of the form of Equation 10.14 and compute the residuals $\hat{\varepsilon}_i$. One then resamples these residuals to generate bootstrap resamples $\tilde{\varepsilon}_i$. Under the conditions of the null hypothesis, $Y = (I - \lambda W)^{-1}\varepsilon$, so we may apply the plug-in principle to obtain

$$\tilde{Y} = (I - \lambda W)^{-1}\tilde{\varepsilon}. \tag{10.16}$$

One then computes the sample mean of the $\tilde{Y}$. This procedure is repeated many times, and the variance of these means becomes the bootstrap variance estimate for use in computing $\hat{n}_e$ via Equation 10.8.

Similar to previous analyses, to analyze the parametric bootstrap we will create a 24 by 24 grid and excise the outer two grid cells around the boundary. Once again, we will use

Monte Carlo simulation. First, we create a function `t.para.boot()` that will be called by the function `replicate()`. The function returns the outcome of the *t* test and the standard error.

```
> # Monte Carlo simulation of parametric bootstrap
> t.para.boot <- function(IrWinv, nlist.small, W){
+ # Generate the sample, eliminating the boundary
+    Y <- IrWinv %*% rnorm(24^2)
+    Y.samp <- matrix(Y, nrow = 24, byrow = TRUE)[3:22,3:22]
+ # Fit a spatial error model to the data
+    Y.mod <- errorsarlm(as.vector(Y.samp) ~ 1, listw = W)
+    e.hat <- residuals(Y.mod)
+    lambda.hat <- Y.mod$lambda
+    IrWinv.mod <- invIrM(nlist.small, lambda.hat)
+ # Bootstrap resampling
+    u <- replicate(200,para.samp(e.hat, IrWinv.mod))
+ # Carry out the t-test
+    t.stat <- mean(Y.samp) / sd(u)
+    p <- 2 * (1 - pt(q = abs(t.stat),
+     df = nrow(Y.samp)*ncol(Y.samp) - 1))
+    return(c(as.numeric(p < 0.05), sd(u)))
+ }
```

The function has three arguments: `IrWinv`, `nlist.small`, and `W`. The first is the matrix $(I - \lambda W)^{-1}$ on the 24 by 24 lattice, the second is the neighbor list of the 20 by 20 lattice, and the third is the `listw` object of the 20 by 20 lattice (see Section 3.5.2). The `sp` function `errorsarlm()`, discussed in Chapter 13, is used to estimate the parameter λ. This estimate is used to construct a matrix of the form $(I - \lambda W)^{-1}$ on the 20 by 20 lattice. The residuals are then resampled, and the resampled errors are used to generate via Equation 10.16 a set of values Y_i, whose mean is computed. This process is bootstrapped B times (in this case, $B = 200$) to generate the bootstrap distribution.

The function `t.para.boot()` calls a function `para.samp()` to generate the resampled data using Equation 10.16. Here is the code for that function.

```
> para.samp <- function(e.hat, IrWinv.mod){
+   e.samp <- sample(e.hat, length(e.hat), replace = TRUE)
+ +   Y.boot <- IrWinv.mod %*% e.samp
+   return(mean(Y.boot))
+ }
```

The Monte Carlo simulation is carried out for $\lambda = 0.4$ as follows (as with the code in Section 10.3.3, this takes a *really* long time to run!).

```
> lambda <- 0.4
> nlist <- cell2nb(24, 24)
> IrWinv <- invIrM(nlist, lambda)
> nlist.small <- cell2nb(20, 20)
> W <- nb2listw(nlist.small)
> U <- replicate(10000, t.para.boot(IrWinv, nlist.small, W))
> mean(U[1,]) # Error rate
[1] 0.0576
> mean(U[2,]) # Mean est. std. error
[1] 0.07996112
```

TABLE 10.3

Power of the Test of the Methods for Several Values of δ, with $\lambda = 0.6$

δ	0.125	0.25	0.375	0.5
Uncorrected	0.554	0.862	0.983	0.999
Block bootstrap	0.295	0.667	0.920	0.994
Parametric bootstrap	0.203	0.569	0.875	0.986

Note: Based on 10,000 Monte Carlo simulations.

As was the case with time series data, the parametric bootstrap outperforms the block bootstrap at all values of λ (Table 10.2), and, as with the case of time series data, one must interpret these results with caution, recognizing that in the test of the parametric bootstrap there is no misspecification error, and therefore the results of this test may give an unrealistically optimistic picture of the method.

10.4.3 Power of the Tests

The block bootstrap and especially the parametric bootstrap have been shown to be reasonably effective at reducing the Type I error rate for spatially correlated data. One might ask whether these methods have adequate power. Recall (Section 3.3) that the power of a test is defined as 1 – Pr{Type II error}, where a Type II error is the failure to reject the null hypothesis when it is false. Monte Carlo experiments were carried out in which a constant δ was added to the spatially autocorrelated data of Equation 10.16 with a value of λ equal to 0.6. Simulations were conducted of t tests using no correction, the block bootstrap, and the parametric bootstrap. The results are shown in Table 10.3. The methods lack the power of the simple t test but do have reasonable power. The parametric bootstrap is actually slightly less powerful than the block bootstrap in this particular simulation.

10.5 Application to the EM38 Data

Both the block bootstrap and the parametric bootstrap were tested on the data shown in Figure 10.2 (Section 10.1) in a paired t test of the null hypothesis that the mean of the difference between the distributions of the EM38 samples is zero. The block bootstrap works best when applied to a regularly spaced rectangular region, and this particular data set fits that criterion. For our test, the bottom row and eastern column of the sample were removed to create a rectangular lattice of size 6 by 12 cells. No trend removal was carried out, since it was assumed that the data followed the same trend on the two dates, meaning that the difference between them has no trend.

Here is the code for a block bootstrap resample. First, we modify the function `block.samp()`, which constructs and samples the blocks, from that of Section 10.4.1 to accept rectangular lattices.

```
> block.samp <- function(n.blockx, n.blocky, blk.size, blocks){
+ # Resample the block index
+    i.samp <- sample(1:(n.blockx*n.blocky), replace = TRUE)
+ # Construct the 3D array of blocks
+    Ct <- matrix(0, nrow = n.blocky * blk.size,
+       ncol = n.blockx * blk.size)
+    k <- 1
+ # Place resampled blocks in the array
+    for(i in 1:n.blocky){
+       for(j in 1:n.blockx){
+          Ct[(1+(i-1)*blk.size):(i*blk.size),
+             (1+(j-1)*blk.size):(j*blk.size)] <- blocks[,,i.samp[k]]
+          k <- k+1
+    }}
+ # Since we are just computing the mean, we wouldn't realy need
+ # to compute the transpose here
+    return(t(Ct))
+ }
```

The function `mean.b.boot()` is unchanged, beyond passing the extra arguments describing the lattice.

```
> mean.b.boot <- function(n.blockx, n.blocky, blk.size, blocks){
+    mean(block.samp(n.blockx, n.blocky, blk.size, blocks))
+ }
```

After loading the data as described in Appendix B.4, we construct the Y matrix, deleting column 7 and row 13 from the data.

```
> EM38.rect <- data.Set4.1[which(data.Set4.1$Column <= 6 &
+    data.Set4.1$Row <= 12),]
> EM38.rect$EMDiff <- with(EM38.rect, EM38B425 - EM38F520)
> Ysamp <- matrix(EM38.rect$EMDiff, nrow = 12, byrow = TRUE)
> Yt <- t(Ysamp)
```

We will use a 3 by 3 block, meaning that there are two blocks in the east–west direction and four blocks in the north–south direction.

```
> n.blockx <- 4
> n.blocky <- 2
> blk.size <- 3
```

Next, we create the array of blocks.

```
> blocks <- array(0, c(blk.size,blk.size,n.blockx * n.blocky))
> k <- 1
> for(i in 1:n.blocky){
+    for(j in 1:n.blockx){
+       blocks[,,k] <- Yt[(1+(i-1)*blk.size):(i*blk.size),
+          (1+(j-1)*blk.size):(j*blk.size)]
+          k <- k+1
+ }}
>
```

Now we are ready to carry out the test.

```
> set.seed(123)
> u <- replicate(200, mean.b.boot(n.blockx, n.blocky, blk.size, blocks))
> # Carry out the t-test
> Y.bar <- mean(Ysamp)
> se.boot <- sd(u)
> t.stat <- Y.bar / se.boot
> print(p <- 2 * (1 - pt(q = abs(t.stat),
+    df = nrow(Ysamp)*ncol(Ysamp) - 1)))
[1] 0.2262729
```

We can compute the effective sample size using Equation 10.8.

```
> print(ne.hat.bb <- var(EM38.rect$EMDiff) / se.boot^2)
[1] 15.68952
```

The effective sample size as estimated by the block bootstrap is about $\hat{n}_e = 16$, compared with the sample size of $n = 72$.

Turning to the parametric bootstrap, the function `para.samp()`, which uses the objects `e.hat` and `IrWinv.mod`, is unchanged from Section 10.4.2.

```
> para.samp <- function(e.hat, IrWinv.mod){
+   e.samp <- sample(e.hat, length(e.hat), replace = TRUE)
+   Y.boot <- IrWinv.mod %*% e.samp
+   return(mean(Y.boot))
+ }
```

To carry out the parametric bootstrap, we first construct the objects describing the geometry of the lattice.

```
> library(spdep)
> set.seed(123)
> coordinates(EM38.rect) <- c("Easting", "Northing")
> nlist.w <- dnearneigh(EM38.rect, d1 = 0, d2 = 61)
> W <- nb2listw(nlist.w, style = "W")
```

Next, we construct the spatial error model of the data and compute its residuals and parameter estimate.

```
> Y.mod <- errorsarlm(EMDiff ~ 1, data = EM38.rect, listw = W)
> e.hat <- residuals(Y.mod)
> print(lambda.hat <- Y.mod$lambda)
  lambda
0.533286
> IrWinv.mod <- invIrM(nlist.w, lambda.hat)
```

Now we carry out the bootstrap resample.

```
> U <- replicate(200,para.samp(e.hat, IrWinv.mod))
```

Finally, we are ready to carry out the t test and estimate the effective sample size.

```
> se.para <- sd(U)
> t.stat <- mean(EM38.rect$EMDiff) / se.para
> print(p <- 2 * (1 - pt(q = abs(t.stat),
+       df = length(EM38.rect$EMDiff) - 1)))
[1] 0.1489184
> print(ne.hat.para <- var(EM38.rect$EMDiff) / se.para^2)
[1] 22.42287
```

The block bootstrap produces a p value of 0.22, and the parametric bootstrap produces a p value of 0.14. Recall that in Section 10.1 the naïve t test gave us a p value of 0.053.

In summary, the results indicate that there is no valid evidence to indicate that a significant difference exists between the means of the two EM38 data sets.

There is absolutely no agronomic importance associated with the issue of whether or not the mean of the EM38 values sampled from the bed on April 25, 1996 does or does not happen to equal the mean of the EM38 values sampled from the furrow on May 20, 1996 (aside possibly from the question of consistency of the EM38 instrument). This was just a convenient question to ask to illustrate the methods. A great deal of caution must be exercised, however, in interpreting the results of similar tests in the case that these methods are applied to questions that do have some importance. The methods are relatively new and have not been sufficiently tested for a substantial body of literature to have accumulated. Using this example as an illustration, the strongest inferences that can be made are that the data cannot be assumed to satisfy the independence of errors assumption associated with classical statistical analysis, and that the effective sample size is considerably reduced by spatial autocorrelation.

10.6 Further Reading

This chapter makes use of the bootstrap only to estimate the standard error. This is the simplest use of this tool, but there are many other uses. Efron and Tibshirani (1993), Manly (1997), and Sprent (1998) provide good entry level discussions. At a slightly higher level are Efron (1979) and Davison and Hinkley (1997). Both Efron and Tibshirani (1993) and Davison and Hinkley (1997) discuss the block bootstrap and the parametric bootstrap. Davison and Hinckley (1997) provide an extensive discussion of the spatial block bootstrap, and Zhu and Morgan (2004) have used it to carry out a statistical analysis of the distribution of nematodes in a field in Wisconsin. Lahiri and Zhu (2006) apply the block bootstrap to another agricultural problem. Kutner et al. (2005, p. 458) and Fox (1997, p. 493) discuss the application of bootstrapping to regression analysis to generate confidence intervals and carry out hypothesis tests involving the regression coefficients. Rizzo (2008) provides R code for the bootstrap method.

Exercises

10.1 Generate a population of 10,000 values from a standard normal distribution. Draw a sample of size 10 from this population and compute the standard error. Use the function `boot()` to compute the bootstrap standard error based on 50 resamples, and compare the two standard errors.

10.2 Compare accuracy of the bootstrap in problem 10.1 with 20, 50, 1000, and 10,000 resamples.

10.3 Enclose the work you did in Exercise 10.1 in a function and use the function `replicate()` to replicate the bootstrap process 1000 times. Compute the average difference between the bootstrap standard error estimate and the population standard error, and plot a histogram of these differences.

10.4 Repeat Problem 10.2 for a population of 10,000 values generated from a lognormal distribution. What can you say about the effect of asymmetry of the population distribution on the outcome of the bootstrapping?

10.5 Using the data from Field 4.1, test the null hypothesis that the expected value of the EM38 data measured in the bed (EM38425B) is the same as that of the furrow (EM38425F) using a parametric bootstrap correction based on a spatial lag model. How does your result compare with that obtained in the text using the spatial error model? (This exercise may require reading of Chapter 13).

10.6 (a) Carry out a Monte Carlo simulation of the test of the null hypothesis $\mu = 0$ against the alternative $\mu \neq 0$ for a spatially autocorrelated data set similar to that of Section 10.4.2 in which you repeatedly generate a set of pseudorandom numbers that satisfy this parametric model, and use the plug-in principle to estimate the standard error. (b) Compare the results with those obtained in the text.