CHAPTER 8

Simulation for checking statistical procedures and model fits

This chapter describes a variety of ways in which probabilistic simulation can be used to better understand statistical procedures in general, and the fit of models to data in particular. In Sections 8.1–8.2, we discuss *fake-data simulation*, that is, controlled experiments in which the parameters of a statistical model are set to fixed "true" values, and then simulations are used to study the properties of statistical methods. Sections 8.3–8.4 consider the related but different method of *predictive simulation*, where a model is fit to data, then replicated datasets are simulated from this estimated model, and then the replicated data are compared to the actual data.

The difference between these two general approaches is that, in fake-data simulation, estimated parameters are compared to true parameters, to check that a statistical method performs as advertised. In predictive simulation, replicated datasets are compared to an actual dataset, to check the fit of a particular model.

8.1 Fake-data simulation

Simulation of fake data can be used to validate statistical algorithms and to check the properties of estimation procedures. We illustrate with a simple regression model, where we simulate fake data from the model, $y = \alpha + \beta x + \epsilon$, refit the model to the simulated data, and check the coverage of the 68% and 95% intervals for the coefficent β.

First we set up the true values of the parameters—which we arbitrarily set to $\alpha = 1.4$, $\beta = 2.3$, $\sigma = 0.9$—and set up the predictors, which we arbitrarily set to $(1, 2, 3, 4, 5)$:

R code

```
a <- 1.4
b <- 2.3
sigma <- 0.9
x <- 1:5
n <- length(x)
```

We then simulate a vector y of fake data and fit a regression model to these data. The fitting makes no use of the true values of α, β, and σ.

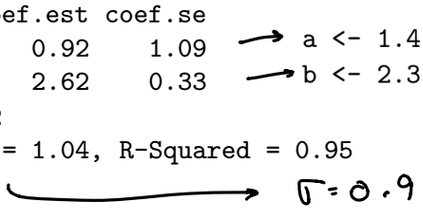
R code

```
y <- a + b*x + rnorm (n, 0, sigma)
lm.1 <- lm (y ~ x)
display (lm.1)
```

Here is the regression output:

R output

```
lm(formula = y ~ x)
            coef.est coef.se
(Intercept)     0.92    1.09
x               2.62    0.33
  n = 5, k = 2
  residual sd = 1.04, R-Squared = 0.95
```



Comparing the estimated coefficients to the true values 1.4 and 2.3, the fit seems reasonable enough: the estimates are not exact but are within the margin of error. We can perform this comparison more formally by extracting from the regression object the estimate and standard error of β (the second coefficient in the model):

R code
```
b.hat <- coef (lm.1)[2]             # "b" is the 2nd coef in the model
b.se <- se.coef (lm.1)[2]           # "b" is the 2nd coef in the model
```

and then checking whether the true β falls within the estimated 68% and 95% confidence intervals obtained by taking the estimate ± 1 or ± 2 standard errors (recall Figure 3.7 on page 40):

R code
```
cover.68 <- abs (b - b.hat) < b.se       # this will be TRUE or FALSE
cover.95 <- abs (b - b.hat) < 2*b.se     # this will be TRUE or FALSE
cat (paste ("68% coverage: ", cover.68, "\n"))
cat (paste ("95% coverage: ", cover.95, "\n"))
```

So, the confidence intervals worked once, but do they have the correct coverage probabilities? We can check by embedding the data simulation, model fitting, and coverage checking in a loop and running 1000 times:[1]

R code
```
n.fake <- 1000
cover.68 <- rep (NA, n.fake)
cover.95 <- rep (NA, n.fake)
for (s in 1:n.fake){
  y <- a + b*x + rnorm (n, 0, sigma)
  lm.1 <- lm (y ~ x)
  b.hat <- coef (lm.1)[2]
  b.se <- se.coef (lm.1)[2]
  cover.68[s] <- abs (b - b.hat) < b.se
  cover.95[s] <- abs (b - b.hat) < 2*b.se
}
cat (paste ("68% coverage: ", mean(cover.68), "\n"))
cat (paste ("95% coverage: ", mean(cover.95), "\n"))
```

The following appears on the console:

R output
```
68% coverage:  0.61
95% coverage:  0.85
```
→ 61% of the 95% intervals covered the value 2.3
→ 85% of the 95% intervals covered the value 2.3

That is, `mean(cover.68)` = 0.61 and `mean(cover.95)` = 0.85. This does not seem right: only 61% of the 68% intervals and 85% of the 95% intervals covered the true parameter value!

Our problem is that the ± 1 and ± 2 standard-error intervals are appropriate for the normal distribution, but with such a small sample size our inferences should use the t distribution, in this case with 3 degrees of freedom (5 data points, minus 2 coefficients estimated; see Section 3.4). We repeat our simulation but using t_3 confidence intervals:

R code
```
n.fake <- 1000
cover.68 <- rep (NA, n.fake)
cover.95 <- rep (NA, n.fake)
t.68 <- qt (.84, n-2)
t.95 <- qt (.975, n-2)
for (s in 1:n.fake){
  y <- a + b*x + rnorm (n, 0, sigma)
```

[1] This and other loops in this chapter could also be performed implicitly using the `replicate()` function in R, as illustrated on pages 139 and 147.

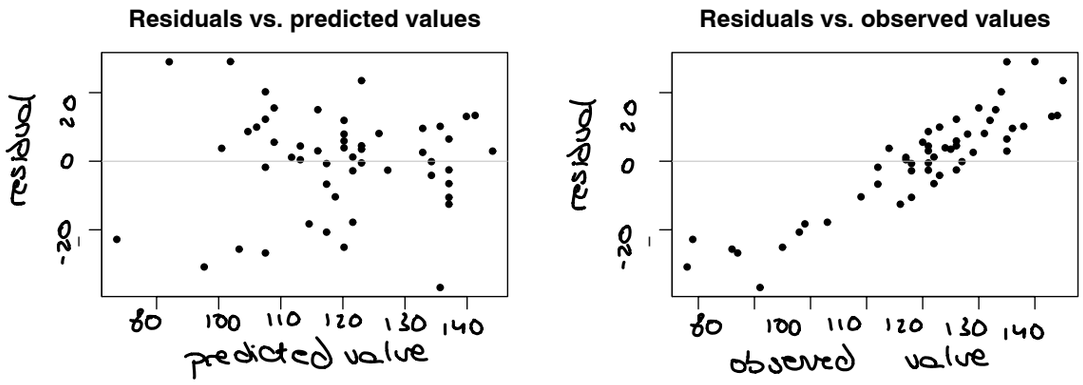


Figure 8.1 *From a model predicting final exam grades from midterms: plots of regression residuals versus predicted and versus observed values. The left plot looks reasonable but the right plot shows strong patterns. How to understand these? An exploration using fake data (see Figure 8.2) shows that, even if the model were correct, we would expect the right plot to show strong patterns. The plot of residuals versus observed thus does not indicate a problem with the model.*

```
  lm.1 <- lm (y ~ x)
  b.hat <- coef (lm.1)[2]
  b.se <- se.coef (lm.1)[2]
  cover.68[s] <- abs (b - b.hat) < t.68*b.se
  cover.95[s] <- abs (b - b.hat) < t.95*b.se
}
cat (paste ("68% coverage ", mean(cover.68), "\n"))
cat (paste ("95% coverage: ", mean(cover.95), "\n"))
```

and now we obtain coverages of 67% and 96%, as predicted (within the expected level of variation based on 1000 simulations; see Exercise 7.10).

8.2 Example: using fake-data simulation to understand residual plots

For another illustration of the power of fake data, we simulate from a regression model to get insight into residual plots, in particular, to understand why we plot residuals versus fitted values rather than versus observed values (see Section 3.6).

We illustrate with a simple model predicting final exam scores from midterms in an introductory statistics class:

```
lm.1 <- lm (final ~ midterm)
```

R code

yielding

```
            coef.est coef.se
(Intercept)     64.5    17.0
midterm          0.7     0.2
  n = 52, k = 2
  residual sd = 14.8, R-Squared = 0.18
```

R output

We construct fitted values $\hat{y} = X\hat{\beta}$ and residuals $y - X\hat{\beta}$:

```
n <- length (final)
X <- cbind (rep(1,n), midterm)
predicted <- X %*% coef (lm.1)
```

R code

Figure 8.1 shows the residuals from this model, plotted in two different ways: (a) residuals versus fitted values, and (b) residuals versus observed values. The first

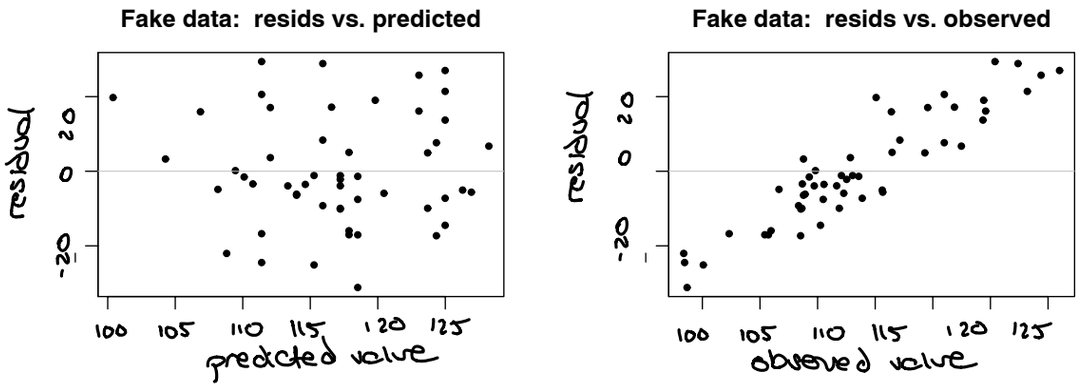


Figure 8.2 *From fake data: plots of regression residuals versus predicted and versus observed values. The data were simulated from the fitted family of regression models, and so we know that the strong pattern in the right panel does not represent any sort of model failure. This is an illustration of the use of fake data to evaluate diagnostic plots. Compare to the corresponding plots of real data in Figure 8.1.*

plot looks reasonable: the residuals are centered around zero for all fitted values. But the second plot looks troubling.

It turns out that the first plot is what we should be looking at, and the second plot is misleading. This can be understood using probability theory (from the regression model, the errors ϵ should be independent of the predictors x, not the data y) but a perhaps more convincing demonstration uses fake data, as we now illustrate.

For this example, we set the regression coefficients and residual standard error to reasonable values given the model estimates, and then simulate fake data:

R code
```
a <- 65
b <- 0.7
sigma <- 15
y.fake <- a + b*midterm + rnorm (n, 0, 15)
```

Next we fit the regression model to the fake data and compute fitted values and residuals:

R code
```
lm.fake <- lm (y.fake ~ midterm)
predicted.fake <- X %*% coef (lm.fake)
resid.fake <- y.fake - predicted.fake
```

(The predicted values could also be obtained in R using `fitted(lm.fake)`; here we explicitly multiply the predictors by the coefficients to emphasize the computations used in creating the fake data.) Figure 8.2 shows the plots of `resid.fake` versus `predicted.fake` and `y.fake`. These are the sorts of residual plots we would see *if the model were correct*. This simulation shows why we prefer, as a diagnostic plot, to view residuals versus predicted rather than observed values.

8.3 Simulating from the fitted model and comparing to actual data

So far we have considered several uses of simulation: exploring the implications of hypothesized probability models (Section 7.1); exploring the implications of statistical models that were fit to data (Sections 7.2–7.4); studying the properties of statistical procedures by comparing to known true values of parameters (Sections 8.1–8.2). Here we introduce yet another twist: simulating replicated data under the fitted model (as with the predictions in Sections 7.2–7.4) and then comparing these to the observed data (rather than comparing estimates to true parameter values as in Section 8.1).

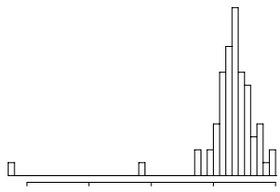


Figure 8.3 *Histogram of Simon Newcomb's measurements for estimating the speed of light, from Stigler (1977). The data represent the amount of time required for light to travel a distance of 7442 meters and are recorded as deviations from* 24,800 *nanoseconds.*

Example: comparing data to replications from a fitted normal distribution

The most fundamental way to check model fit is to display replicated datasets and compare them to the actual data. Here we illustrate with a simple case, from a famous historical dataset that did not fit the normal distribution. The goal of this example is to demonstrate how the lack of fit can be seen using predictive replications.

Figure 8.3 shows the data, a set of measurements taken by Simon Newcomb in 1882 as part of an experiment to estimate the speed of light. We (inappropriately) fit a normal distribution to these data, which in the regression context can be done by fitting a linear regression with no predictors:

```
light <- lm (y ~ 1)
```
R code

The next step is to simulate 1000 replications from the parameters in the fitted model (in this case, simply the constant term β_0 and the residual standard deviation σ):

```
n.sims <- 1000
sim.light <- sim (light, n.sims)
```
R code

We can then use these simulations to create 1000 fake datasets of 66 observations each:

```
n <- length (y)
y.rep <- array (NA, c(n.sims, n))
for (s in 1:n.sims){
  y.rep[s,] <- rnorm (n, sim.light$beta[s], sim.light$sigma[s])
}
```
R code

Visual comparison of actual and replicated datasets. Figure 8.4 shows a plot of 20 of the replicated datasets, produced as follows:

```
par (mfrow=c(5,4))
for (s in 1:20){
  hist (y.rep[s,])
}
```
R code

The systematic differences between data and replications are clear. In more complicated problems, more effort may be needed to effectively display the data and replications for useful comparisons, but the same general idea holds.

Checking model fit using a numerical data summary. Data displays can suggest more focused test statistics with which to check model fit, as we illustrate in Section 24.2. Here we demonstrate a simple example with the speed-of-light measurements. The graphical check in Figures 8.3 and 8.4 shows that the data have some extremely low values that do not appear in the replications. We can formalize this check by defining a *test statistic*, $T(y)$, equal to the minimum value of the data, and then calculating $T(y^{\text{rep}})$ for each of the replicated datasets:

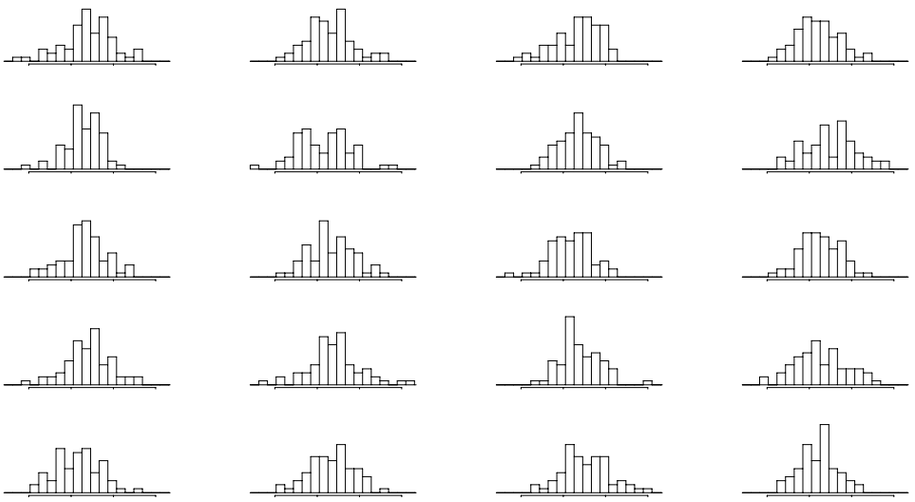


Figure 8.4 *Twenty replications, y^{rep}, of the speed-of-light data from the predictive distribution under the normal model; compare to observed data, y, in Figure 8.3. Each histogram displays the result of drawing 66 independent values y_i^{rep} from a common normal distribution with mean and standard deviation (μ, σ) estimated from the data.*

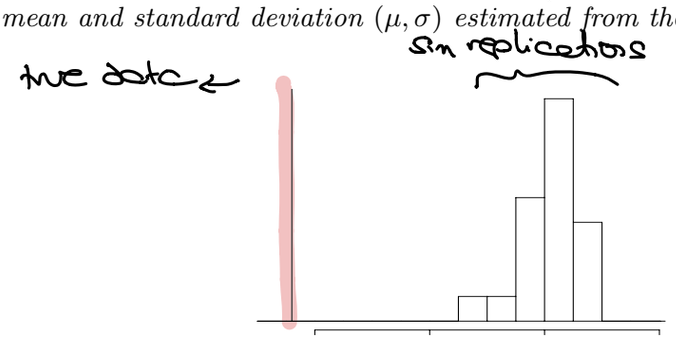


Figure 8.5 *Smallest observation of Newcomb's speed-of-light data (the vertical line at the left of the graph), compared to the smallest observations from each of 20 posterior predictive simulated datasets displayed in Figure 8.4.*

R code

```
Test <- function (y){
  min (y)
}
test.rep <- rep (NA, n.sims)
for (s in 1:n.sims){
  test.rep[s] <- Test (y.rep[s,])
}
```

We then plot a histogram of the minima of the replicated datasets, with a vertical line indicating the minimum of the observed data:

R code

```
hist (test.rep, xlim=range (Test(y), test.rep))
lines (rep (Test(y), 2), c(0,n))
```

Figure 8.5 shows the result: the smallest observations in each of the hypothetical replications are all much larger than Newcomb's smallest observation, which is indicated by a vertical line on the graph. The normal model clearly does not capture the variation that Newcomb observed. A revised model might use an asymmetric contaminated normal distribution or a symmetric long-tailed distribution in place of the normal measurement model.

Example: zeroes in count data

For a more complicated example, we consider a study of the effect of integrated pest management on reducing cockroach levels in urban apartments. In this experiment, the treatment and control were applied to 160 and 104 apartments, respectively, and the outcome measurement y_i in each apartment i was the number of roaches caught in a set of traps. Different apartments had traps for different numbers of days, and we label as u_i the number of trap-days. The natural model for the roach counts is then $y_i \sim \text{Poisson}(u_i \exp(X_i\beta))$, where X represents the regression predictors (in this case, a pre-treatment roach level, a treatment indicator, and an indicator for whether the apartment is in a "senior" building restricted to the elderly, and the constant term). The logarithm of the exposure, $\log(u_i)$, plays the role of the "offset" in the Poisson regression (see model (6.3) on page 111).

We fit the model

```
glm.1 <- glm (y ~ roach1 + treatment + senior, family=poisson,
  offset=log(exposure2))
```
R code

which yields

```
            coef.est coef.se
(Intercept)    -0.46    0.02
roach1          0.24    0.00
treatment      -0.48    0.02
senior         -0.40    0.03
  n = 264, k = 4
  residual deviance = 11753.3, null deviance = 17354 (difference = 5600.7)
```
R output

The treatment appears to be effective in reducing roach counts—we shall return to this issue in a later chapter with a fuller exploration of this study. For now, we are simply interested in evaluating the model as a description of the data, without worrying about causal issues or the interpretation of the coefficients.

Comparing the data, y, to a replicated dataset, y^{rep}. How well does this model fit the data? We explore by simulating a replicated dataset y^{rep} that might be seen if the model were true and the study were performed again:

```
n <- length (y)
X <- cbind (rep(1,n), roach1, treatment, senior)
y.hat <- exposure2 * exp (X %*% coef (glm.1))
y.rep <- rpois (n, y.hat)
```
R code

We can compare the replicated data y^{rep} to the original data y in various ways. We illustrate with a simple test of the number of zeroes in the data:

```
print (mean (y==0))
print (mean (y.rep==0))
```
R code

which reveals that 36% of the observed data points, but none of the replicated data points, equal zero. This suggests a potential problem with the model: in reality, many apartments have zero roaches, but this would not be happening if the model were true, at least to judge from one simulation.

Comparing the data y to 1000 replicated datasets y^{rep}. To perform this model check more formally, we simulate 1000 replicated datasets y^{rep}, which we store in a matrix:

R code
```
n.sims <- 1000
sim.1 <- sim (glm.1, n.sims)
y.rep <- array (NA, c(n.sims, n))
for (s in 1:n.sims){
  y.hat <- exposure2 * exp (X %*% sim.1$beta[s,])
  y.rep[s,] <- rpois (n, y.hat)
}
```

For each of these replications, we then compute a test statistic: the proportion of zeroes in the (hypothetical) dataset:

R code
```
Test <- function (y){
  mean (y==0)
}
test.rep <- rep (NA, n.sims)
for (s in 1:n.sims){
  test.rep[k] <- Test (y.rep[s,])
}
```

The 1000 values of `test.rep` vary from 0 to 0.008—all of which are much lower than the observed test statistic of 0.36. Thus the Poisson regression model does not replicate the frequency of zeroes in the data.

Checking the overdispersed model

We probably should have just started with an overdispersed Poisson regression:

R code
```
glm.2 <- glm (y ~ roach1 + treatment + senior, family=quasipoisson,
  offset=log(exposure2))
```

which yields

R output
```
glm(formula = y ~ roach1 + treatment + senior, family = quasipoisson,
    offset = log(exposure2))
             coef.est coef.se
(Intercept)    -0.46    0.17
roach1          0.24    0.03
treatment      -0.48    0.20
senior         -0.40    0.27
  n = 264, k = 4
  residual deviance = 11753.3, null deviance = 17354 (difference = 5600.7)
  overdispersion parameter = 66.6
```

As discussed in Section 6.2, the coefficient estimates are the same as before but the standard errors are much larger, reflecting the variation that is now being modeled.

Again, we can test the model by simulating 1000 replicated datasets:

R code
```
n.sims <- 1000
sim.2 <- sim (glm.2, n.sims)
y.rep <- array (NA, c(n.sims, n))
for (s in 1:n.sims){
  y.hat <- exposure2 * exp (X %*% sim.2$beta[s,])
  a <- y.hat/(overdisp-1)            # Using R's parameterization for
  y.rep[s,] <- rnegbin (n, y.hat, a) # the negative-binomial distribution
}
```

and again computing the test statistic for each replication. This time, the proportion of zeroes in the replicated datasets varies from 18% to 48%, with a 95% interval

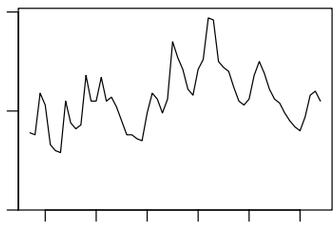


Figure 8.6 *Time series of U.S. unemployment rates from 1947 to 2004. We fit a first-order autoregression to these data and then simulated several datasets, shown in Figure 8.7, from the fitted model.*

of $[0.22, 0.40]$. The observed 36% fits right in, which tells us that this aspect of the data is reasonably fit by the model.

However, other aspects of the data might not be so well fit, as could be discovered by looking at other test statistics. We discuss this in Chapter 24 in the context of a more elaborate example.

8.4 Using predictive simulation to check the fit of a time-series model

Predictive simulation is more complicated in time-series models, which are typically set up so that the distribution for each point depends on the earlier data. We illustrate with a simple autoregressive model.

Fitting a first-order autoregression to the unemployment series

Figure 8.6 shows the time series of annual unemployment rates in the United States from 1947 to 2004. We would like to see how well these data are fit by a first-order autoregression, that is, a regression on last year's unemployment rate. Such a model is easy to set up and fit:[2]

R code
```
n <- length (y)
y.lag <- c (NA, y[1:(n-1)])
lm.lag <- lm (y ~ y.lag)
```

yielding the following fit:

R output
```
             coef.est coef.se
(Intercept) 1.43     0.50
y.lag       0.75     0.09
  n = 57, k = 2
  residual sd = 0.99, R-Squared = 0.57
```

This information is potentially informative but does not tell us whether the model is a reasonable fit to the data. To examine fit, we will simulate replicated data from the fitted model.

Simulating replicated datasets

Using a point estimate of the fitted model. We first simulate replicated data in a slightly simplified way, using the following point estimate from the fitted model:

[2] Another option is to use some of the special time-series features in R, but it is simpler for us here to just fit as an ordinary regression.

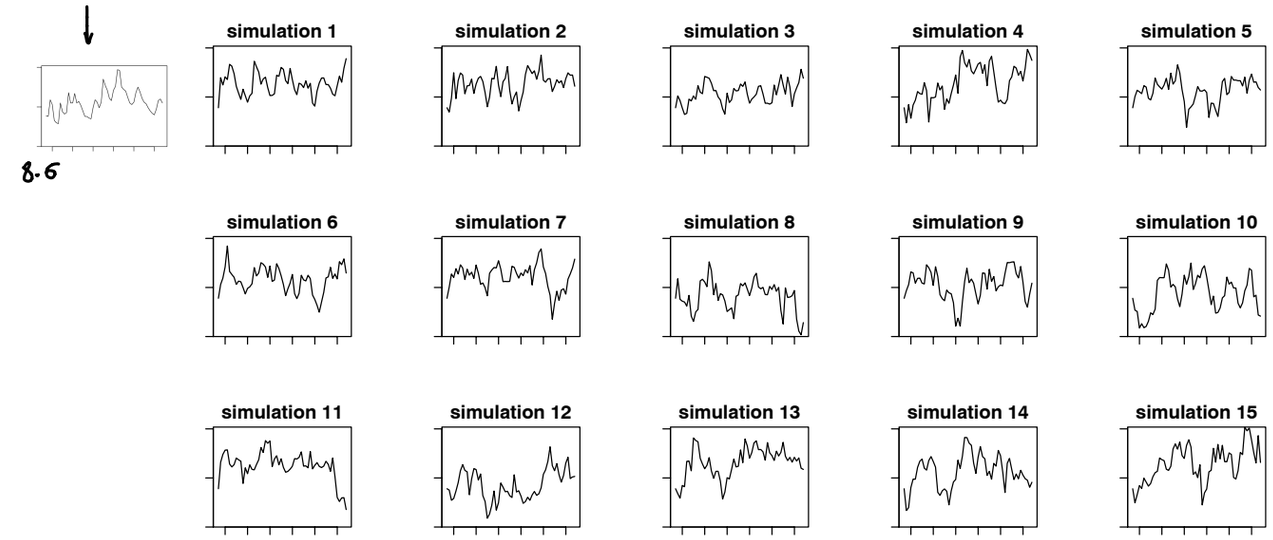


Figure 8.7 *Simulated replications of the unemployment series from the fitted autoregressive model. The replications capture many of the features of the actual data in Figure 8.6 but show slightly more short-term variation.*

R code
```
b.hat <- coef (lm.lag)              # vector of 2 regression coefs
s.hat <- sigma.hat (lm.lag)         # residual standard deviation
```

We start each of the simulated time series at the observed value y_1 (the actual unemployment rate in 1947) and then use the model, step by step, to simulate each year's value from the last:

R output
```
n.sims <- 100
y.rep <- array (NA, c(n.sims, n))
for (s in 1:n.sims){
  y.rep[s,1] <- y[1]
  for (t in 2:n){
    prediction <- c (1, y.rep[s,t-1]) %*% b.hat
    y.rep[s,t] <- rnorm (1, prediction, s.hat)
  }
}
```

Including the uncertainty in the estimated parameters. It is slightly better to propagate the estimation uncertainty by using simulations from the fitted model (as in Section 7.2), and then using these draws of β and σ to simulate replicated datasets:

R code
```
lm.lag.sim <- sim (lm.lag, n.sims)  # simulations of beta and sigma
for (s in 1:n.sims){
  y.rep[s,1] <- y[1]
  for (t in 2:n){
    prediction <-  c (1, y.rep[s,t-1]) %*% lm.lag.sim$beta[s,]
    y.rep[s,t] <- rnorm (1, prediction, lm.lag.sim$sigma[s])
  }
}
```

Visual and numerical comparisons of replicated to actual data

Our first step in model checking is to plot some simulated datasets, which we do in Figure 8.7, and compare them visually to the actual data in Figure 8.6. The 15 simulations show different patterns, with many of them capturing the broad

features of the data—its range, lack of overall trend, and irregular rises and falls. This autoregressive model clearly can represent many different sorts of time-series patterns.

Looking carefully at Figure 8.7, we see one pattern in all these replicated data that was not in the original data in 8.6, and that is a jaggedness, a level of short-term ups and downs that contrasts to the smoother appearance of the actual time series.

To quantify this discrepancy, we define a test statistic that is the frequency of "switches"—the number of years in which an increase in unemployment is immediately followed by a decrease, or vice versa:

R code
```
Test <- function (y){
  n <- length (y)
  y.lag <- c (NA, y[1:(n-1)])
  y.lag2 <- c (NA, NA, y[1:(n-2)])
  sum (sign(y-y.lag) != sign(y.lag-y.lag2), na.rm=TRUE)
}
```

As with the examples in the previous section, we compute this test for the data and for the replicated datasets:

R code
```
print (Test(y))
test.rep <- rep (NA, n.sims)
for (s in 1:n.sims){
  test.rep[s] <- Test (y.rep[s,])
}
```

The actual unemployment series featured 23 switches. Of the 1000 replications, 97% had more than 23 switches, implying that this aspect of the data was not captured well by the model.

8.5 Bibliographic note

Fake-data simulation is commonly used to validate statistical models and procedures. Two recent papers from a Bayesian perspective are Geweke (2004) and Cook, Gelman, and Rubin (2006). The predictive approach to model checking is described in detail in Gelman et al. (2003, chapter 6) and Gelman, Meng, and Stern (1996), deriving from the ideas of Rubin (1984). Gelman (2004a) connects graphical model checks to exploratory data analysis (Tukey, 1977). Examples of simulation-based model checking appear throughout the statistical literature, especially for highly structured models; see, for example, Bush and Mosteller (1955) and Ripley (1988).

8.6 Exercises

1. Fitting the wrong model: suppose you have 100 data points that arose from the following model: $y = 3 + 0.1x_1 + 0.5x_2 + \text{error}$, with errors having a t distribution with mean 0, scale 5, and 4 degrees of freedom.We shall explore the implications of fitting a standard linear regression to these data.

   (a) Simulate data from this model. For simplicity, suppose the values of x_1 are simply the integers from 1 to 100, and that the values of x_2 are random and equally likely to be 0 or 1.[3] Fit a linear regression (with normal errors) to these

[3] In R, you can define `x.1 <- 1:100`, simulate `x.2` using `rbinom()`, then create the linear predictor, and finally simulate the random errors in `y` using the `rt()` function.

data and see if the 68% confidence intervals for the regression coefficients (for each, the estimates ± 1 standard error) cover the true values.

(b) Put the above step in a loop and repeat 1000 times. Calculate the confidence coverage for the 68% intervals for each of the three coefficients in the model.

(c) Repeat this simulation, but instead fit the model using t errors (see Exercise 6.6).

2. Predictive checks: using data of interest to you, fit a model of interest.

(a) Simulate replicated datasets and visually compare to the actual data.

(b) Summarize the data by a numerical test statistic, and compare to the values of the test statistic in the replicated datasets.

3. Using simulation to check the fit of a time-series model: find time-series data and fit a first-order autoregression model to it. Then use predictive simulation to check the fit of this model as in Section 8.4.

4. Model checking for count data: the folder `risky.behavior` contains data from a study of behavior of couples at risk for HIV; see Exercise 6.1.

(a) Fit a Poisson regression model predicting number of unprotected sex acts from baseline HIV status. Perform predictive simulation to generate 1000 datasets and record both the percent of observations that are equal to 0 and the percent that are greater than 10 (the third quartile in the observed data) for each. Compare these values to the observed value in the original data.

(b) Repeat (a) using an overdispersed Poisson regression model.

(c) Repeat (b), also including ethnicity and baseline number of unprotected sex acts as input variables.