CHAPTER 7

Simulation of probability models and statistical inferences

Whenever we represent inferences for a parameter using a point estimate and standard error, we are performing a data reduction. If the estimate is normally distributed, this summary discards no information because the normal distribution is completely defined by its mean and variance. But in other cases it can be useful to represent the uncertainty in the parameter estimation by a set of random simulations that represent possible values of the parameter vector (with more likely values being more likely to appear in the simulation). By *simulation*, then, we mean summarizing inferences by random numbers rather than by point estimates and standard errors.

7.1 Simulation of probability models

In this section we introduce simulation for two simple probability models. The rest of the chapter discusses how to use simulations to summarize and understand regressions and generalize linear models, and the next chapter applies simulation to model checking and validation. Simulation is important in itself and also prepares for multilevel models, which we fit using simulation-based inference, as described in Part 2B.

A simple example of discrete predictive simulation

How many girls in 400 births? The probability that a baby is a girl or boy is 48.8% or 51.2%, respectively. Suppose that 400 babies are born in a hospital in a given year. How many will be girls?

We can simulate the 400 births using the binomial distribution:

R code
```
n.girls <- rbinom (1, 400, .488)
print (n.girls)
```

which shows us what could happen in 400 births. To get a sense of the *distribution* of what could happen, we simulate the process 1000 times (after first creating the vector `n.girls` to store the simulations):

R code
```
n.sims <- 1000
n.girls <- rep (NA, n.sims)
for (s in 1:n.sims){
  n.girls[s] <- rbinom (1, 400, .488)}
hist (n.girls)
```

which yields the histogram in Figure 7.1 representing the probability distribution for the number of girl births. The 1000 simulations capture the uncertainty.[1]

[1] In this example, we performed all the simulations in a loop. It would also be possible to simulate 1000 draws from the binomial distribution directly:

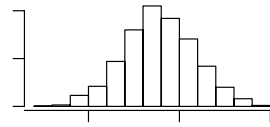

Figure 7.1 *Histogram of 1000 simulated values for the number of girls born in a hospital out of 400 babies, as simulated from the binomial probability distribution with probability 0.488.*

Accounting for twins. We can complicate the model in various ways. For example, there is a 1/125 chance that a birth event results in fraternal twins, of which each has an approximate 49.5% chance of being a girl, and a 1/300 chance of identical twins, which have an approximate 49.5% chance of being girls. We can simulate 400 birth events as follows:

R code

```
birth.type <- sample (c("fraternal twin","identical twin","single birth"),
  size=400, replace=TRUE, prob=c(1/125, 1/300, 1 - 1/125 - 1/300))
girls <- rep (NA, 400)
for (i in 1:400){
  if (birth.type[i]=="single birth"){
    girls[i] <- rbinom (1, 1, .488)}
  else if (birth.type[i]=="identical twin"){
    girls[i] <- 2*rbinom (1, 1, .495)}
  else if (birth.type[i]=="fraternal twin"){
    girls[i] <- rbinom (1, 2, .495)}
}
n.girls <- sum (girls)
```

Here, `girls` is a vector of length 400, of 0's, 1's, and 2's (mostly 0's and 1's) representing the number of girls in each birth event.[2] To approximate the *distribution* of the number of girls in 400 births, we put the simulation in a loop and repeat it 1000 times:

R code

```
n.girls <- rep (NA, n.sims)
for (s in 1:n.sims){
  birth.type <-sample(c("fraternal twin","identical twin","single birth"),
    size=400, replace=TRUE, prob=c(1/125, 1/300, 1 - 1/125 - 1/300))
  girls <- rep (NA, 400)
  for (i in 1:400){
    if (birth.type[i]=="single birth"){
      girls[i] <- rbinom (1, 1, .488)}
    else if (birth.type[i]=="identical twin"){
      girls[i] <- 2*rbinom (1, 1, .495)}
    else if (birth.type[i]=="fraternal twin"){
      girls[i] <- rbinom (1, 2, .495)}
```

`n.girls <- rbinom (n.sims, 400, .488)`
In other settings one can write the simulation as a function and perform the looping implicitly using the `replicate()` function in R, as we illustrate on page 139.

[2] Again, this calculation could also be performed without looping using vector operations in R:
```
girls <- ifelse (birth.type=="single birth", rbinom (400, 1, .488), ifelse (
  birth.type="identical twins", 2*rbinom (400, 1, .495), rbinom (400, 2, .495)))
```
We have used looping in the main text to emphasize the parallel calculation for the 400 birth events, but the vectorized computation is faster and can be more convenient when part of a larger computation.

```
    }
    n.girls[s] <- sum (girls)
  }
```

This nested looping is characteristic of simulations of complex data structures and can also be implemented using custom R functions and the `replicate()` function, as we discuss shortly.

A simple example of continuous predictive simulation

Similarly, we can program R to simulate continuous random variables. For example, 52% of adults in the United States are women and 48% are men. The heights of the men are approximately normally distributed with mean 69.1 inches and standard deviation 2.9 inches; women with mean 63.7 and standard deviation 2.7.

Suppose we select 10 adults at random. What can we say about their average height?

R code
```
sex <- rbinom (10, 1, .52)
height <- ifelse (sex==0, rnorm (10, 69.1, 2.9), rnorm (10, 64.5, 2.7))
avg.height <- mean (height)
print (avg.height)
```

To simulate the distribution of `avg.height`, we loop the simulation 1000 times:

R code
```
n.sims <- 1000
avg.height <- rep (NA, n.sims)
for (s in 1:n.sims){
  sex <- rbinom (10, 1, .52)
  height <- ifelse (sex==0, rnorm (10, 69.1, 2.9), rnorm (10, 64.5, 2.7))
  avg.height[s] <- mean (height)
}
hist (avg.height, main="Average height of 10 adults")
```

What about the maximum height of the 10 people? To determine this, just add the following line within the loop:

R code
```
  max.height[s] <- max (height)
```

and before the loop, initialize `max.height`:

R code
```
max.height <- rep (NA, n.sims)
```

Then, after the loop, make a histogram of `max.height`.

Simulation in R using custom-made functions

The coding for simulations becomes cleaner if we express the steps for a single simulation as a function in R. We illustrate with the simulation of average heights. First, the function:

R code
```
Height.sim <- function (n.adults){
  sex <- rbinom (n.adults, 1, .52)
  height <- ifelse (sex==0, rnorm (10, 69.5, 2.9), rnorm (10, 64.5, 2.7))
  return (mean(height))
}
```

(For simplicity we have "hard-coded" the proportion of women and the mean and standard deviation of men's and women's heights, but more generally these could be supplied as arguments to the function.)

We then use the `replicate()` function to call `Height.sim()` 1000 times:

R code
```
avg.height <- replicate (1000, Height.sim (n.adults=10))
hist (avg.height)
```

See Section 20.5 for a more elaborate example of the use of functions in R, in a fake-data simulation to perform a power calculation.

7.2 Summarizing linear regressions using simulation: an informal Bayesian approach

In a regression setting, we can use simulation to capture both predictive uncertainty (the error term in the regression model) and inferential uncertainty (the standard errors of the coefficients and uncertainty about the residual error). We first discuss the simplest case of simulating prediction errors, then consider inferential uncertainty and the combination of both sources of variation.

Simulation to represent predictive uncertainty

We illustrate predictive uncertainty with the problem of predicting the earnings of a 68-inch-tall man, using model (4.2) on page 63.

Obtaining the point and interval predictions automatically. The predictive estimate and confidence interval can easily be accessed using the regression software in R:

R code
```
x.new <- data.frame (height=68, male=1)
pred.interval <- predict (earn.logmodel.3, x.new, interval="prediction",
  level=.95)
```

and then exponentiating to get the predictions on the original (unlogged) scale of earnings:

R code
```
exp (pred.interval)
```

Constructing the predictive interval using simulation. We now discuss how to obtain predictive intervals "manually" using simulations derived from the fitted regression model. In this example it would be easier to simply use the `predict()` function as just shown; however, simulation is a general tool that we will be able to apply in more complicated predictive settings, as we illustrate later in this chapter and the next.

- The point estimate for log earnings is $8.4+0.017\cdot 68-0.079\cdot 1+0.007\cdot 68\cdot 1 = 9.95$, with a standard deviation of 0.88. To put these on the original (unlogged) scale, we exponentiate to yield a geometric mean of $e^{9.95} = 21000$ and a geometric standard deviation of $e^{0.88} = 2.4$.

  Then, for example, the 68% predictive interval is $[21000/2.4,\ 21000 \cdot 2.4] = [8800, 50000]$, and the 95% interval is $[21000/2.4^2,\ 21000 \cdot 2.4^2] = [3600, 121000]$
- The simulation prediction is a set of random numbers whose logarithms have mean 9.95 and standard deviation 0.88. For example, in R, we can summarize the predictive distribution using the following command:

R code
```
pred <- exp (rnorm (1000, 9.95, .88))
```

which tells R to draw 1000 random numbers from a normal distribution with mean 9.95 and variance 0.88, and then exponentiate these values.

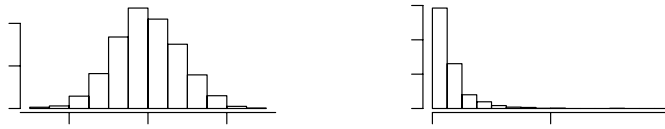

Figure 7.2 *Histogram of 1000 simulated values from the predictive distribution of the earnings of a 68-inch-tall man from a fitted regression model, on the logarithmic and original scales.*

We can display the simulations as a histogram (see Figure 7.2) and also compute various numerical summaries, for example,

- mean: `mean(pred)`
- median: `quantile(pred,.5)`
- 50% interval: `quantile(pred,c(.25,.75))`
- 95% interval: `quantile(pred,c(.025,.975))`

(These calculations ignore uncertainty in the regression parameters and thus are only approximate; we describe a more complete computational procedure later in this section.)

Why do we need simulation for predictive inferences?

For many purposes, point estimates, standard errors, and the intervals obtained from the `predict()` function in R are sufficient because the Central Limit Theorem ensures that for all but the smallest sample sizes and for reasonably well-behaved error distributions, coefficient estimates are approximately normally distributed (see page 14). Accounting for the uncertainty in the standard-error estimates, the t-distribution with $n-k$ degrees of freedom (where k is the number of predictors in the model) is a reliable approximation for the appropriate uncertainty distribution for the coefficients. Analytic procedures can also be used to get uncertainty for linear combinations of parameters and predictions. (An example of a linear combination of predictions is to use one of the models in Chapter 3 to predict the average test score of a group of 100 children whose mothers' educations and IQs are known.)

For more general predictions, however, the easiest and most reliable way to compute uncertainties is by simulation. For example, suppose we have a 68-inch-tall woman and a 68-inch-tall man, and we would like to use model (4.2) to predict the difference of their earnings. As a point estimate, we can use the difference of the point predictions: $\exp(8.4 + 0.017 \cdot 68 - 0.079 \cdot 1 + 0.007 \cdot 68 \cdot 1) - \exp(8.4 + 0.017 \cdot 68 - 0.079 \cdot 0 + 0.007 \cdot 68 \cdot 0) = 6900$. The simplest way to get a standard error or uncertainty interval for this prediction is to use simulation:

R code

```
pred.man <- exp (rnorm (1000, 8.4 + .017*68 - .079*1 + .007*68*1, .88))
pred.woman <- exp (rnorm (1000, 8.4 + .017*68 - .079*0 + .007*68*0, .88))
pred.diff <- pred.man - pred.woman
pred.ratio <- pred.man/pred.woman
```

We can summarize the distribution of this difference using a histogram or numerical summaries such as `mean(pred.diff)`, `quantile(pred.ratio,c(.25,.75))`, and so forth.

More generally, simulation is valuable because it can be used to summarize *any*

function of estimated and predicted values. This is important partly for practical purposes in summarizing predictions and also because it allows us to fit complicated models in which the ultimate objects of interest are more complicated than a set of regression coefficients or linear combination of coefficients. Simulation will also be crucial when working with nonlinear models such as logistic regression.

Simulation to represent uncertainty in regression coefficients

The usual summary of a fitted regression gives standard errors along with estimates for each coefficient, and these give a sense of the uncertainty in estimation (see Figure 3.7 on page 40). When going beyond inferences for individual coefficients, however, it is helpful to summarize inferences by simulation, which gives us complete flexibility in propagating uncertainty about combinations of parameters and predictions.

For classical linear regressions and generalized linear models, we implement these simulations using the `sim()` function in R. For example, if we do

R code
```
n.sims <- 1000
fit.1 <- lm (log.earn ~ height + male + height:male)
sim.1 <- sim (fit.1, n.sims)
```

then `sim.1$beta` is a matrix with 1000 rows and 4 columns (representing 1000 independent simulations of the vector $(\beta_0, \beta_1, \beta_2, \beta_3)$), and `sim.1$sigma` is a vector of length 1000 (representing the estimation uncertainty in the residual standard-deviation parameter σ).

We can check that these simulations are equivalent to the regression computations, for example by the following commands in R, which print the mean, standard deviation, and 95% interval for the coefficient for `height` in the fitted model:

R code
```
height.coef <- sim.1$beta[,2]
mean (height.coef)
sd (height.coef)
quantile (height.coef, c(.025,.975))
```

For a more interesting example, consider the question: In this interaction model, what can be said about the coefficient of height among men? We cannot directly answer this question using the regression output: the slope for men is a sum of the `height` and `height:male` coefficients, and there is no simple way to compute its standard error given the information in the regression table. The most direct approach is to compute the 95% interval directly from the inferential simulations:

R code
```
height.for.men.coef <- sim.1$beta[,2] + sim.1$beta[,4]
quantile (height.for.men.coef, c(.025,.975))
```

The result is $[-0.003, 0.049]$, that is, $[-0.3\%, 4.9\%]$. Statistical significance is not the object of the analysis—our conclusions should not be greatly changed if, for example, the 95% interval instead were $[0.1\%, 5.3\%]$—but it is important to have a sense of the uncertainty of estimates, and it is convenient to be able to do this using the inferential simulations. The powers of inferential simulations are demonstrated more effectively when combined with prediction, as illustrated in Section 7.3.

Details of the simulation procedure

To get `n.sims` simulation draws (for example, 1000 is typically more than enough; see Chapter 17), we apply the following procedure based on Bayesian inference (see Chapter 18).

1. Use classical regression of n data points on k predictors to compute the vector $\hat{\beta}$ of estimated parameters, the unscaled estimation covariance matrix V_β, and the residual variance $\hat{\sigma}^2$.
2. Create `n.sims` random simulations of the coefficient vector β and the residual standard deviation σ. For each simulation draw:
   (a) Simulate $\sigma = \hat{\sigma}\sqrt{(n-k)/X}$, where X is a random draw from the χ^2 distribution with $n-k$ degrees of freedom.
   (b) Given the random draw of σ, simulate β from a multivariate normal distribution with mean $\hat{\beta}$ and variance matrix $\sigma^2 V_\beta$.

   These simulations are centered about the estimates $\hat{\beta}$ and $\hat{\sigma}$ with variation representing estimation uncertainty in the parameters. (For example, approximately 68% of the simulations of β_1 will be within ± 1 standard error of $\hat{\beta}_1$, approximately 95% will be within ± 2 standard errors, and so forth.)

These steps are performed automatically by our R function `sim()`, which pulls out $n, k, \hat{\beta}, V_\beta, \hat{\sigma}$ from the fitted linear model and then performs a loop over the n_{sims} simulations:

```
for (s in 1:n.sims){                                              R code
  sigma[s] <- sigma.hat*sqrt((n-k)/rchisq(1,n-k))
  beta[s,] <- mvrnorm (1, beta.hat, V.beta*sigma[s]^2)
}
```

The `sim()` function then returns the vector of simulations of σ and the $n_{\text{sims}} \times k$ matrix of simulations of β:

```
return (list (beta=beta, sigma=sigma))                            R code
```

The list items are given names so they can be accessed using these names from the simulation object. The function works similarly for generalized linear models such as logistic and Poisson regressions, adjusting for any overdispersion by using the standard errors of the coefficient estimates, which are scaled for overdispersion if that is included in the model.

Informal Bayesian inference

Bayesian inference refers to statistical procedures that model unknown parameters (and also missing and latent data) as random variables. As described in more detail in Section 18.3, Bayesian inference starts with a *prior distribution* on the unknown parameters and updates this with the *likelihood* of the data, yielding a *posterior distribution* which is used for inferences and predictions.

Part 2 of this book discusses how Bayesian inference is appropriate for multilevel modeling—in which it is natural to fit probability distributions to batches of parameters. For the classical models considered in Part 1, Bayesian inference is simpler, typically starting with a "noninformative" or uniform prior distribution on the unknown parameters. We will not explore the technical issues further here except to note that the simulations presented here correspond to these noninformative prior distributions. It can also be helpful to think of these simulations as representing configurations of parameters and predictions that are compatible with the observed data—in the same sense that a classical confidence interval contains a range of parameter values that are not contradicted by the data.

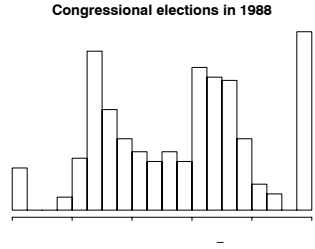

Figure 7.3 *Histogram of congressional election data from 1988. The spikes at the left and right ends represent uncontested Republicans and Democrats, respectively.*

7.3 Simulation for nonlinear predictions: congressional elections

We illustrate nonlinear predictions in the context of a model of elections for the U.S. Congress. We first construct a model to predict the 1988 election from the 1986 election. Then we apply the model to predict 1990 from 1988. (It is convenient, when learning about a method, to predict outcomes that have already occurred, so the predictions can be compared to reality.)

Background

The United States is divided into 435 congressional districts, and we define the outcome y_i, for $i = 1, \ldots, n = 435$ to be the Democratic Party's share of the two-party vote (that is, excluding the votes for parties other than the Democrats and the Republicans) in district i in 1988. Figure 7.3 shows a histogram of the data y.

How can the variation in the data be understood? What information would be relevant in predicting the outcome of a congressional election? First of all, it is useful to know whether both parties are contesting the election; the spikes at the two ends of the histogram reveal that many of the elections were uncontested. After that, it would seem to make sense to use the outcome of the most recent previous election, which was in 1988. In addition, we use the knowledge of whether the *incumbent*—the current occupant of the congressional seat—is running for reelection.

Our regression model has the following predictors:

- A constant term
- The Democratic share of the two-party vote in district i in the previous election
- Incumbency: an indicator that equals $+1$ if district i is currently (as of 1988) occupied by a Democrat who is running for reelection, -1 if a Republican is running for reelection, and 0 if the election is *open*—that is, if neither of the two candidates is currently occupying the seat.

Because the incumbency predictor is categorical, we can display the data in a single scatterplot using different symbols for Republican incumbents, Democratic incumbents, and open seats; see Figure 7.4a.

We shall fit a linear regression. The data—number of votes for each candidate—are discrete, so it might at first seem appropriate to fit a generalized linear model such as an overdispersed binomial. But the number of votes within each district is large enough that the vote proportions are essentially continuous, so nothing would be gained by attempting to model the discreteness in the data.

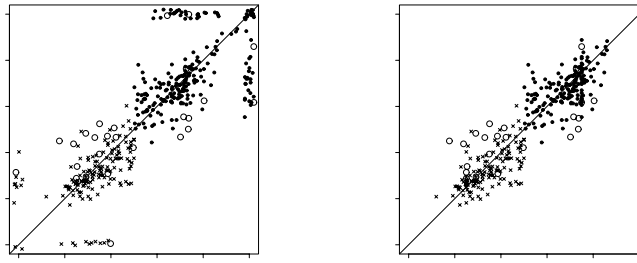

Figure 7.4 *(a) Congressional election data from 1986 and 1988. Crosses correspond to elections with Republican incumbents running in 1988, dots correspond to Democratic incumbents, and open circles correspond to open seats. The "incumbency" predictor in the regression model equals 0 for the circles, +1 for the dots, and −1 for the crosses. Uncontested election outcomes (at 0 and 1) have been jittered slightly. (b) Data for the regression analysis, with uncontested 1988 elections removed and uncontested 1986 election values replaced by 0.25 and 0.75. The $y = x$ line is included as a comparison on both plots.*

Data issues

Many of the elections were uncontested in 1988, so that $y_i = 0$ or 1 exactly; for simplicity, we exclude these from our analysis. Thus, we are predicting the 1988 results given the outcome in the previous election and the knowledge of (a) whether the incumbent is running for reelection, and (b) whether the election will be contested. Primary elections are typically in September, and so it is reasonable to expect to have this information about two months before the November general election. We also exclude any elections that were won by third parties, yielding $n = 343$ congressional elections for our analysis.

In addition, many elections were uncontested in 1986, so the previous election outcome X_{2i} is 0 or 1 exactly. It would be possible to simply include these in the model as is; however, instead we impute the value 0.25 for uncontested Republicans and 0.75 for uncontested Democrats, which are intended to represent approximately the proportion of votes received by the Democratic candidate had the election actually been contested. (More generally, we can impute random values from the distribution of contested election outcomes preceding an uncontested race, but for our purposes here the simple imputation is suficient.) The adjusted dataset is displayed in Figure 7.4b.

Fitting the model

First we fit the regression model in R: we label the adjusted variables as `vote.88`, `vote.86`, `incumbency.88`, subset them to include only the elections that were contested in 1988, and fit a linear model:

```
fit.88 <- lm (vote.88 ~ vote.86 + incumbency.88)
```

R code

Displaying yields

```
             coef.est coef.se
(Intercept)      0.20    0.02
vote.86          0.58    0.04
```

R output

sim	σ	β_0	β_1	β_2	$\tilde{y}_1$	$\tilde{y}_2$	$\cdots$	$\tilde{y}_{55}$	$\cdots$	$\tilde{y}_{435}$	$\sum_i I(\tilde{y}_i > 0.5)$
1	.065	.19	.62	.067	.69	.57	$\cdots$	NA	$\cdots$	.79	251
2	.069	.25	.50	.097	.75	.63	$\cdots$	NA	$\cdots$	.76	254
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1000	.067	.23	.51	.089	.73	.57	$\cdots$	NA	$\cdots$	.69	251
median	.068	.20	.58	.077	.73	.65	$\cdots$	NA	$\cdots$	.72	253
mean	.067	.20	.58	.078	.73	.65	$\cdots$	NA	$\cdots$	.72	252.4
sd	.003	.02	.04	.007	.07	.07	$\cdots$	NA	$\cdots$	.07	3.1

Figure 7.5 *Simulation results for the congressional election forecasting model. The predicted values $\tilde{y}_i$ correspond to the 1990 election. The NAs are for a district that was uncontested in 1990, so it was not predicted by the regression model.*

```
incumbency.88     0.08    0.01
  n = 343, k = 3
  residual sd = 0.067, R-Squared = 0.88
```

This model has serious problems, as can be seen, for example, by careful examination of the plot of residuals or even of the before-after plot in Figure 7.4b (for example, the jump between the average y-values just below and just above $x = 0.5$ is not completely fit by the `incumbency.88` predictor). Better models can be fit to these data (see Exercise 9.13), but the simple regression fit here is sufficient to demonstrate the principles of simulation-based predictive inference.

Simulation for inferences and predictions of new data points

The first five columns of Figure 7.5 show a set of simulation results for the parameters in the fitted model. We use these, along with the data from 1988 and incumbency information in 1990, to predict the district-by-district election outcome in 1990. We start by creating a new matrix of predictors, $\tilde{X}$:

R code

```
n.tilde <- length (vote.88)
X.tilde <- cbind (rep(1,n.tilde), vote.88, incumbency.90)
```

We then simulate $n_{\text{sims}} = 1000$ predictive simulations of the vector of $\tilde{n}$ new data points with $\tilde{n} \times k$ matrix of predictors $\tilde{X}$. For each simulation, we compute the predicted value $\tilde{X}\beta$ and add normal errors:

R code

```
n.sims <- 1000
sim.88 <- sim (fit.88, n.sims)
y.tilde <- array (NA, c(n.sims, n.tilde))
for (s in 1:n.sims){
  y.tilde[s,] <- rnorm (n.tilde, X.tilde %*% sim.88$beta[s,],
    sim.88$sigma[s])}
```

This last matrix multiplication works because `X.tilde` is a $\tilde{n} \times 3$ matrix and `sim.88$beta` is a $n_{\text{sims}} \times 3$ matrix; thus the selected row, `sim.88$beta[s,]`, is a vector of length 3, and the product `X.tilde%*%sim.88$beta[s,]` is a vector of length $\tilde{n}$ that represents the vector of predicted values for that particular simulation draw.

Predictive simulation for a nonlinear function of new data

For the congressional elections example, we perform inference on the summary measure $\sum_{i=1}^{n} I(\tilde{y}_i > 0.5)$—the number of elections won by the Democrats in 1990, by summing over the rows in the matrix:[3]

```
dems.tilde <- rowSums (y.tilde > .5)
```
R code

The last column of Figure 7.5 shows the results. Each row shows the outcome of a different random simulation.

The lower lines of the table in Figure 7.5 show the median, mean, and standard deviation of each simulated outcome. The means and medians of the parameters σ and β are nearly identical to the point estimates (the differences are due to variation because there are only 1000 simulation draws). The future election outcome in each district has a predictive uncertainty of about 0.07, which makes sense since the estimated standard deviation from the regression is $\hat{\sigma} = 0.07$. (The predictive uncertainties are slightly higher than $\hat{\sigma}$, but by only a very small amount since the number of data points in the original regression is large, and the x-values for the predictions are all within the range of the original data.)

Finally, the entries in the lower-right corner of Figure 7.5 give a predictive mean of 252.4 and standard error of 3.1 for the number of districts to be won by the Democrats. This estimate and standard error *could not* simply be calculated from the estimates and uncertainties for the individual districts. Simulation is the only practical method of assessing the predictive uncertainty for this nonlinear function of the predicted outcomes.

Incidentally, the actual number of seats won by the Democrats in 1990 was 262. This is more than 3 standard deviations away from the mean, which suggests that the model is not quite applicable to the 1990 election—this makes sense since it does not allow for national partisan swings of the sort that happen from election to election.

Implementation using functions

We could also compute these predictions by writing a custom R function:

```
Pred.88 <- function (X.pred, lm.fit){
  n.pred <- dim(X.pred)[1]
  sim.88 <- sim (lm.fit, 1)
  y.pred <- rnorm (n.pred, X.pred %*% t(sim.88$beta), sim.88$sigma)
  return (y.pred)
}
```
R code

and then creating 1000 simulations using the `replicate()` function in R:

```
y.tilde <- replicate (1000, Pred.88 (X.tilde, fit.88))
```
R code

To predict the total number of seats won by the Democrats, we can add a wrapper:

```
dems.tilde <- replicate (1000, Pred.88 (X.tilde, fit.88) > .5)
```
R code

Computing using `replicate()` (or related functions such as `apply()` and `sapply()`) results in faster and more compact R code which, depending on one's programming experience can appear either simpler or more mysterious than explicit looping. We sometimes find it helpful to perform computations both ways when we are uncertain about the programming.

[3] We could also calculate this sum in a loop as

```
dems.tilde <- rep (NA, n.sims)
for (s in 1:n.sims){
  dems.tilde[s] <- sum (y.tilde[s,] > .5)}
```

Combining simulation and analytic calculations

In some settings it is helpful to supplement simulation-based inference with mathematical analysis. For example, in the election prediction model, suppose we want to estimate the probability that the election in a particular district will be tied, or within one vote of being exactly tied. (This calculation is relevant, for example, in estimating the probability that an individual vote will be decisive, and comparing these probabilities can be relevant for parties' decisions for allocating campaign resources.)

Consider a district, i, with n_i voters. For simplicity we suppose n_i is even. This district's election will be tied if the future vote outcome, $\tilde{y}_i$, is exactly 0.5. We have approximated the distribution of $\tilde{y}$ as continuous—which is perfectly reasonable given that the n_i's are in the tens or hundreds of thousands—and so a tie is equivalent to $\tilde{y}_i$ being in the range $[\frac{1}{2} - \frac{1}{2n_i}, \frac{1}{2} + \frac{1}{2n_i}]$.

How can we compute this probability by simulation? The most direct way is to perform many predictive simulations and count the proportion for which $\tilde{y}_i$ falls in the range $0.5 \pm 1/(2n_i)$. Unfortunately, for realistic n_i's, this range is so tiny that thousands or millions of simulations could be required to estimate this probability accurately. (For example, it would not be very helpful to learn that 0 out of 1000 simulations fell within the interval.)

A better approach is to combine simulation and analytical results: first compute 1000 simulations of $\tilde{y}$, as shown, then for each district compute the proportion of simulations that fall between 0.49 and 0.51, say, and divide by $0.02n_i$ (that is, the number of intervals of width $1/n_i$ that fit between 0.49 and 0.51). Or compute the proportion falling between 0.45 and 0.55, and divide by $0.1n_i$. For some districts, the probability will still be estimated at zero after 1000 simulation draws, but in this case the estimated zero is much more precise.

Estimated probabilities for extremely rare events can be computed in this example using the fact that predictive distributions from a linear regression follow the t distribution with $n-k$ degrees of freedom. We can use 1000 simulations to compute a predictive mean and standard deviation for each $\tilde{y}_i$, then use tail probabilities of the t_{340} distribution (in this example, $n = 343$ and $k = 3$) to compute the probability of falling in the range $0.5 \pm 1/(2n_i)$.

7.4 Predictive simulation for generalized linear models

As with linear regression, we simulate inference for generalized linear models in two steps: first using `sim()` to obtain simulations for the coefficients, then simulating predictions from the appropriate model, given the linear predictor.

Logistic regression

We illustrate for one of the models from Chapter 5 of the probability of switching wells given the distance from the nearest safe well.

Simulating the uncertainty in the estimated coefficients. Figure 7.6a shows the uncertainty in the regression coefficients, computed as follows:

R code

```
sim.1 <- sim (fit.1, n.sims=1000)
plot (sim.1$beta[,1], sim.1$beta[,2], xlab=expression(beta[0]),
  ylab=expression(beta[1]))
```

and Figure 7.6b shows the corresponding uncertainty in the logistic regression curve, displayed as follows:

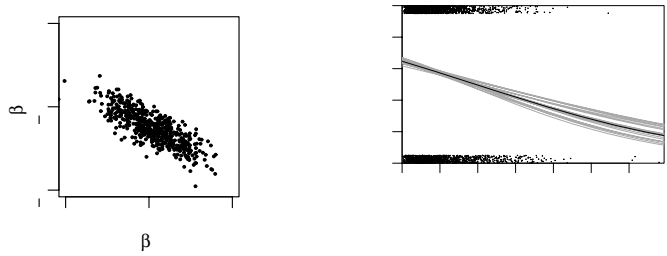

Figure 7.6 *(a) Uncertainty in the estimated coefficients β_0 and β_1 in the logistic regression,* $\Pr(\textit{switching wells}) = \textit{logit}^{-1}(\beta_0 - \beta_0 \cdot \texttt{dist100})$. *(b) Graphical expression of the best-fit model,* $\Pr(\textit{switching wells}) = \textit{logit}^{-1}(0.61 - 0.62 \cdot \texttt{dist100})$, *with (jittered) data overlain. Light lines represent estimation uncertainty in the logistic regression coefficients, corresponding to the distribution of β shown to the left. Compare to Figure 5.9 on page 89.*

sim	β_0	β_1	$\tilde{y}_1$	$\tilde{y}_2$	$\cdots$	$\tilde{y}_{10}$
1	0.68	−0.007	1	0	$\cdots$	1
2	0.61	−0.005	0	0	$\cdots$	1
⋮	⋮	⋮	⋮	⋮	⋱	⋮
1000	0.69	−0.006	1	1	$\cdots$	1
mean	0.61	−0.006	0.60	0.59	$\cdots$	0.52

Figure 7.7 *Simulation results for ten hypothetical new households in the well-switching example, predicting based only on distance to the nearest well. The inferences for (β_0, β_1) are displayed as a scatterplot in Figure 7.6a. The bottom row—the mean of the simulated values of $\tilde{y}_i$ for each household i—gives the estimated probabilities of switching.*

R code
```
plot (dist, switch)
for (s in 1:20){
  curve (invlogit (sim.1$beta[s,1] + sim.1$beta[s,2]*x), col="gray",
    add=TRUE)}
curve (invlogit (fit.1$coef[1] + fit.1$coef[2]*x), add=TRUE)
```

Predictive simulation using the binomial distribution. Now suppose, for example, that we would like to predict the switching behavior for $\tilde{n}$ new households, given a predictor matrix $\tilde{X}$ (which will have $\tilde{n}$ rows and, in this example, two columns, corresponding to the constant term and the distance to the nearest safe well). As with linear regression, we can use simulation to account for the predictive uncertainty. In this case, we use the binomial distribution to simulate the prediction errors:

R code
```
n.tilde <- nrow (X.tilde)
y.tilde <- array (NA, c(n.sims, n.tilde))
for (s in 1:n.sims){
  p.tilde <- invlogit (X.tilde %*% sim.1$beta[s,])
  y.tilde[s,] <- rbinom (n.tilde, 1, p.tilde)
}
```

Figure 7.7 shows an example set of `n.sims` = 1000 simulations corresponding to `n.tilde` = 10 new households.

Predictive simulation using the latent logistic distribution. An alternative way to simulate logistic regression predictions uses the latent-data formulation (see Section

5.3). We obtain simulations for the latent data $\tilde{z}$ by adding independent errors $\tilde{\epsilon}$ to the linear predictor, and then convert to binary data by setting $\tilde{y}_i = 1$ if $\tilde{z}_i > 0$ for each new household i:

R code
```
y.tilde <- array (NA, c(n.sims, n.tilde))
for (s in 1:n.sims){
  epsilon.tilde <- logit (runif (n.tilde, 0, 1))
  z.tilde <- X.tilde %*% t(sim.1$beta) + epsilon.tilde
  y.tilde[s,] <- ifelse (z.tilde>0, 1, 0)
}
```

Other generalized linear models

We can do similar computations with Poisson regression: inference just as before, and predictive simulations using `rpois()`.

For overdispersed Poisson regression, the function `rnegbin()` samples from the negative binomial distribution. Another option is to sample first from the gamma, then the Poisson. For overdispersed binomial, simulations from the beta-binomial distribution can be obtained by drawing first from the beta distribution, then the binomial.

Compound models

Simulation is the easiest way of summarizing inferences from more complex models. For example, as discussed in Section 6.7, we can model earnings from height in two steps:

$$\Pr(\text{earnings}>0) = \text{logit}^{-1}(-3.76 + 0.08 \cdot \text{height} + 1.70 \cdot \text{male})$$
$$\text{If earnings}>0, \text{ then earnings} = \exp(8.15 + 0.02 \cdot \text{height} + 0.42 \cdot \text{male} + \epsilon),$$

with the error term ϵ having a normal distribution with mean 0 and standard deviation 0.88.

We can simulate the earnings of a randomly chosen 68-inch tall man. We first show, for simplicity, the simulation ignoring uncertainty in the regression coefficients:

R code
```
fit.1a <- glm (earn.pos ~ height + male, family=binomial(link="logit"))
fit.1b <- lm (log.earn ~ height + male, subset = earnings>0)
x.new <- c (1, 68, 1)      # constant term=1, height=68, male=1

n.sims <- 1000
prob.earn.pos <- invlogit (coef(fit.1a) %*% x.new)
earn.pos.sim <- rbinom (n.sims, 1, prob.earn.pos)
earn.sim <- ifelse (earn.pos.sim==0, 0,
  exp (rnorm (n.sims, coef(fit.1b) %*% x.new, sigma.hat(fit.1b))))
```

More generally, we can use the simulated values of the coefficient estimates:

R code
```
sim.1a <- sim (fit.1a, n.sims)
sim.1b <- sim (fit.1b, n.sims)
for (s in 1:n.sims){
  prob.earn.pos <- invlogit (sim.1a$beta %*% x.new)
  earn.pos.sim <- rbinom (n.sims, 1, prob.earn.pos)
  earn.sim[s] <- ifelse (earn.pos.sim==0, 0
    exp (rnorm (n.sims, sim.1b$beta %*% x.new, sim.1b$sigma)))
}
```

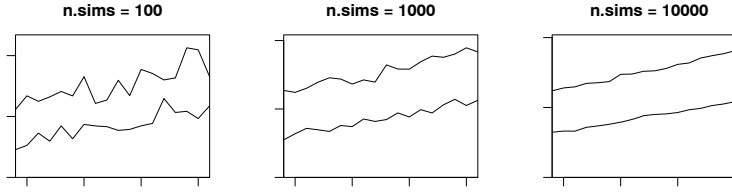

Figure 7.8 *Mean predicted earnings as a function of height and sex for the two-stage model (logistic regression for the probability earnings being positive, followed by linear regression for the logarithms of positive earnings), as computed using 100, 1000, or 10000 simulations.*

Now suppose we want to understand this compound model by plotting the mean predicted earnings as a function of height and sex. We can first put the computations into a function:

R code
```
Mean.earn <- function (height, male, sim.a, sim.b){
  x.new <- c (1, height, male)
  prob.earn.pos <- invlogit (sim.a$beta %*% x.new)
  earn.pos.sim <- rbinom (n.sims, 1, prob.earn.pos)
  earn.sim <- ifelse (earn.pos.sim==0, 0,
    exp (rnorm (n.sims, sim.b$beta %*% x.new, sim.b$sigma)))
  return (mean(earn.sim))
}
```

and then evaluate the function in a loop using the `sapply()` function:[4]

R code
```
heights <- seq (60, 75, 1)
mean.earn.female <- sapply (heights, Mean.earn, male=0, sim.1a, sim.1b)
mean.earn.male <- sapply (heights, Mean.earn, male=1, sim.1a, sim.1b)
```

The plots of `mean.earn.female` and `mean.earn.male` versus `heights` appear in Figure 7.8, for three different values of n_{sims}. The general pattern is clear from 100 simulations, but more simulations are helpful to avoid being distracted by random noise.

7.5 Bibliographic note

Random simulation for performing computations in probability and statistics was one of the first applications of computers, dating back to the 1940s. As computing power became more dispersed since the 1970s, simulation has been used increasingly frequently for summarizing statistical inferences; Rubin (1980) is an early example.

Our simulation-based approach to computation is described in Gelman et al. (2003), and a recent implementation in R appears in Kerman and Gelman (2006). The congressional election analysis in Section 7.3 uses a simplified version of the models of Gelman and King (1990, 1994a).

[4] Alternatively, the looping could be programmed explicitly:
```
heights <- seq (60, 75, 1)
k <- length(heights)
mean.earn.female <- rep (NA, k)
mean.earn.male <- rep (NA, k)
for (i in 1:k) {
  mean.earn.female[i] <- Mean.earn (heights[i], 0, sim.1a, sim.1b)
  mean.earn.male[i] <- Mean.earn (heights[i], 1, sim.1a, sim.1b)
}
```

7.6 Exercises

1. Discrete probability simulation: suppose that a basketball player has a 60% chance of making a shot, and he keeps taking shots until he misses two in a row. Also assume his shots are independent (so that each shot has 60% probability of success, no matter what happened before).
   (a) Write an R function to simulate this process.
   (b) Put the R function in a loop to simulate the process 1000 times. Use the simulation to estimate the mean, standard deviation, and distribution of the total number of shots that the player will take.
   (c) Using your simulations, make a scatterplot of the number of shots the player will take and the proportion of shots that are successes.
2. Continuous probability simulation: the logarithms of weights (in pounds) of men in the United States are approximately normally distributed with mean 5.13 and standard deviation 0.17; women with mean 4.96 and standard deviation 0.20. Suppose 10 adults selected at random step on an elevator with a capacity of 1750 pounds. What is the probability that the elevator cable breaks?
3. Propagation of uncertainty: we use a highly idealized setting to illustrate the use of simulations in combining uncertainties. Suppose a company changes its technology for widget production, and a study estimates the cost savings at $5 per unit, but with a standard error of $4. Furthermore, a forecast estimates the size of the market (that is, the number of widgets that will be sold) at 40,000, with a standard error of 10,000. Assuming these two sources of uncertainty are independent, use simulation to estimate the total amount of money saved by the new product (that is, savings per unit, multiplied by size of the market).
4. Predictive simulation for linear regression: take one of the models from Exercise 3.5 or 4.8 that predicts course evaluations from beauty and other input variables. You will do some simulations.
   (a) Instructor A is a 50-year-old woman who is a native English speaker and has a beauty score of -1. Instructor B is a 60-year-old man who is a native English speaker and has a beauty score of -0.5. Simulate 1000 random draws of the course evaluation rating of these two instructors. In your simulation, account for the uncertainty in the regression parameters (that is, use the `sim()` function) as well as the predictive uncertainty.
   (b) Make a histogram of the difference between the course evaluations for A and B. What is the probability that A will have a higher evaluation?
5. Predictive simulation for linear regression: using data of interest to you, fit a linear regression model. Use the output from this model to simulate a predictive distribution for observations with a particular combination of levels of all the predictors in the regression.
6. Repeat the previous exercise using a logistic regression example.
7. Repeat the previous exercise using a Poisson regression example.
8. Inference for the ratio of parameters: a (hypothetical) study compares the costs and effectiveness of two different medical treatments.
   - In the first part of the study, the difference in costs between treatments A and B is estimated at $600 per patient, with a standard error of $400, based on a regression with 50 degrees of freedom.

- In the second part of the study, the difference in effectiveness is estimated at 3.0 (on some relevant measure), with a standard error of 1.0, based on a regression with 100 degrees of freedom.
- For simplicity, assume that the data from the two parts of the study were collected independently.

Inference is desired for the *incremental cost-effectiveness ratio*: the difference between the average costs of the two treatments, divided by the difference between their average effectiveness. (This problem is discussed further by Heitjan, Moskowitz, and Whang, 1999.)

(a) Create 1000 simulation draws of the cost difference and the effectiveness difference, and make a scatterplot of these draws.

(b) Use simulation to come up with an estimate, 50% interval, and 95% interval for the incremental cost-effectiveness ratio.

(c) Repeat this problem, changing the standard error on the difference in effectiveness to 2.0.

9. Summarizing inferences and predictions using simulation: Exercise 6.5 used a Tobit model to fit a regression with an outcome that had mixed discrete and continuous data. In this exercise you will revisit these data and build a two-step model: (1) logistic regression for zero earnings versus positive earnings, and (2) linear regression for level of earnings given earnings are positive. Compare predictions that result from each of these models with each other.

10. How many simulation draws are needed: take the model from Exercise 3.5 that predicts course evaluations from beauty and other input variables. Use `display()` to summarize the model fit. Focus on the estimate and standard error for the coefficient of beauty.

(a) Use `sim()` with `n.iter` = 10000. Compute the mean and standard deviations of the 1000 simulations of the coefficient of beauty, and check that these are close to the output from `display`.

(b) Repeat with `n.iter` = 1000, `n.iter` = 100, and `n.iter` = 10. Do each of these a few times in order to get a sense of the simulation variability.

(c) How many simulations were needed to give a good approximation to the mean and standard error for the coefficient of beauty?