Chapter 5: Checking Resampling Results

Naomi Bush has just finished her doctoral dissertation examining the flashing synchronization of the Malaysian firefly, *Pteroptyx malaccae*. The central part of her work concerns a hypothesis test on whether one fly's flashing rate influences another fly's rate. She has done the calculations, of course, using resampling.

She is walking home from her laboratory, having just sent off the dissertation for binding. Her head is high as she contemplates whether or not she will feign surprise when she gets the telephone call from the Nobel Committee in Stockholm.

As she gets closer to home, her confidence wanes. Like all good scientists she looks at her results with skepticism. "How do I know whether I used enough trials in my computations? Did I have enough data for the computations to give reliable results? How did I know that I didn't make a programming error? Perhaps the flies don't really influence one another in the way I described!"

Questions like these surround any computation and really any answer to any statistical question. In this section we discuss some ways to answer these questions.

5.1 How many trials to use

Just about every example in this book involves repeating a resampling trial many times. The programs contain lines like:

```
Ntrials=1000;
for trials=1:Ntrials
```

```
    resample and compute your statistic for this trial
    tally the result for this trial
end
```

You should understand that resampling is done at random and therefore the result of any single trial is somewhat random. By combining the results of many trials, we reduce this randomness somewhat, but it is still there to some extent. By increasing the number of trials we reduce the effect of randomness.

For example, consider Example **??** where we wanted to know the 1% percentile of the total lifetime of the four light bulbs in the package. If we re-run this example 10 times, we get several different answers:

```
4614 4780 4810 4475 4544 4475 4660 4300 4660 4780
```

All of the answers are close to one another. Are they "close enough?" This is the question of precision.[1] The answer to this question depends on what you want to use the answer for. Without knowing the particular use for the answer to your statistical question, we can't tell you if a given answer is precise enough. We can, however, tell you how to estimate the precision of your answer and how to make the precision better if you feel you need to do this.

1. Compute the quantity you're interested in. For the light bulbs this was the 1% percentile of the total lifetime of the bulbs in the package.

2. Repeat the calculation 5 or 10 times. Look to see if the spread of values is acceptable for you for your purpose. (If the values are all exactly the same, and resampling is involved, then something may be wrong with your program.) For Example **??**, repeating the calculation produced a spread of values ranging from 4300 to 4810 hours. This might not be precise enough. Since the plan is to mark the package with a guarantee that total lifetime is greater than 4500 hours, if 4300 hours is the true 1% percentile the company will have more refunds to pay than it intended. On the other hand, if 4810 were correct, the company could earn additional sales by marking its package with a guarantee of 4800 hours.

3. If the spread in step (2) is unacceptably large, then increase the number of *trials* used in your calculation. As a rule of thumb, if

[1] Later, we'll consider the matter of how accurate the answer is; whether or not the answers are systematically wrong.

you want to reduce the spread by a factor of N, you need to use N^2 as many trials. For example, suppose you are using 1000 trials and conclude that your result is too imprecise. You want to improve the precision by a factor of 3, that is, make the spread 1/3 as big. Since N is 3, you should try 9 times as many trials, or 9000 trials.

We repeated the Example **??** computations using 10,000 trials. To get an idea of the spread of values with an increased number of trials, we repeated the runs 10 times — altogether 100,000 trials — and got the following results (which have been sorted into ascending order):

```
4453 4497 4497 4497 4497 4520 4544 4544 4544 4544
```

It seems that 4500 is a pretty good answer. However, many of the values are repeated multiple times. This is a sign of trouble; there might not be enough data in the first place. See Sec. **??**

How many trials should you use when first doing a calculation? Here are some hints:

- When writing a new program, use a very small number of trials (say, 2 or 10) for the purpose of debugging. Everyone makes mistakes when typing a program; once you have gotten the program working to the point where it doesn't produce error messages or give obviously incorrect answers, then increase the number of trials.
- When computing confidence intervals, the number of trials needed depends on the confidence level you want to use. You can use the following guidelines (which are based on references [**?**] and [**?**]):

Confidence level	99%	95%	90%	60%	Std. Err.
Number of trials	5000	1000	500	100	50

"Std. Err." refers to the "standard error" which is the standard deviation of a sample statistic.

- Using more trials is better than using fewer trials, so long as the speed and memory size of the computer are not causing a problem.

5.2 How Much Data is Needed

Imagine a researcher who, with great effort, has collected one data point. Without other information, you know nothing about the reliability of this single data point. Neither conventional inference procedures nor

resampling will produce very useful results: all the resamples will be identical.

How about when there are 2 data points? Is that enough?

The question, "How many data points do I need?" is like the question, "How many trials should I use?" An answer to both questions is, "More is better." This answer isn't much use when talking about the number of data points needed: you can't always collect more data, either because you don't have the time, the energy, or the money.

There are, of course, many situations in which one data point is entirely sufficient. This occurs when you know from previous experience what is the variability of the measurement and you know that this variability is so small that it can be ignored. For example, when measuring the air pressure in a car tire, we generally take just one measurement. The precision of the gauge is good enough that averaging multiple measurements would not improve the measurement in any practically meaningful way.

The situations we're considering in this section are those in which the variability of the measurement is unknown, and we want to use the spread of several measurements in order to infer something about the variability of a single measurement.

Here are three different answers to the question of how many data points you need to get reliable results:

1. Rule of thumb:

- Don't expect reliable results if you have 5 or fewer data points.
- If you have more than 20 data points, you should be pretty safe, unless your data include outliers or if you think the system you are studying sometimes — but rarely — produces "way out" values.

2. Simulation.

How did we construct the rule of thumb in (1)? We did a simulation. If you can construct a relevant simulation model of the process that generates your data, you can construct your own rules of thumb that will be more relevant to your own cases.

Here's how we developed the rules of thumb in (1):

(a) We assumed that people are interested in quantities such as 95% confidence intervals of the mean of n samples.

(b) We assumed that data are distributed in a normal-shaped distribution. If your data are different, this assumption can give misleading results.

(c) We constructed a way of generating data like (b) and made many such data sets, each with n data points. (See page **??** for documentation on NORMAL.)

Figure 1:
Results from a simulation showing how the actual confidence level compares to the confidence level that was requested for different sizes of data sets. The actual confidence level is defined to be the fraction of times that the computed confidence interval contains the true mean of the distribution from which the data sets were drawn. Three different requested confidence levels are shown: 95%, 90%, and 60%. It can be seen that the actual confidence level is somewhat below the requested level, but that the discrepancy becomes small as the size of the data set increases. This graph is based on a simulation of data drawn from a normal distribution.

(d) For each of the data sets in (c), we used resampling to find 95% confidence intervals of the mean of the n data points. We call each of these calculations a "run" which consists of 1000 resampling trials.

(e) We know that the true mean of the data in (c) is zero. (We know this because of a theoretical understanding of how computer random number generators work.) If the resampling confidence-interval procedure is working prefectly, about 95% of the runs in (d) should give a confidence interval that includes zero. But the procedure doesn't work perfectly, in part because a small sample of data can give a misleading indication of the mean and spread of the process generating the data. Figure **??** shows how often the confidence error of the mean of a randomly generated data set contains the true mean, as a function of the size of the data set. This is the actual confidence level of the interval. The figure indicates that for very small data sets the confidence intervals are system-

atically too small, that is, the actual confidence level is below the requested confidence level.

If your data set is too small, and it is impractical to collect more data, and you really do want to have the correct confidence interval, there is one thing you can do, called *recalibration*. In brief, this consists of asking for confidence interval at a higher level of confidence, say 99% percentile, but then interpreting the results as giving the confidence interval at the lower level of confidence. For details on how to do this, see reference [**?**]. To illustrate how calibration works, suppose that you had a data set of 10 samples drawn from a normal distribution. You want a 90% confidence interval. Looking at Figure **??** you see that if you ask for a 90% confidence interval you will be getting what is really at an 85% confidence level. By asking intead for a 95% confidence level, the computations will produce, on average, what is really a 90% level.

3. Jackknife-after-bootstrap

If you aren't sure what type of simulation to use in (2), then here is a general method for deciding whether you have enough data to justify confidence in your results. Note that this method won't tell you how much data you need, just whether you have enough, and note also that a conclusion that you have enough data is not definitive.

The method is based on the idea that if you have enough data, then dropping one data point shouldn't change the answer very much. On the other hand, if there's not enough data then a single point might have a lot of influence. This suggests that one can analyze the situation by repeating the calculation as many times as there are observations, leaving out each of the data points in turn. The resulting set of answers will reflect how much influence each data point has. If some points have a lot of influence, we expect there to be a wide spread in the set of answers and we can quantify the spread of the results using the standard deviation. This method goes under the technical name *jackknife-after-bootstrap*. (See reference [**?**].) This procedure has been implemented as the program JAB in Resampling Stats.

To illustrate, consider the data set

```
≫  data = [1 2 3 4 5];
```

The mean of `data` is 3, but we are interested in the confidence intervals. Considering that we have only 5 data points, we decide to compute 50% confidence intervals. We use the program CONFINTERVALS which carries out the resampling:

```
≫  confintervals(data, 'mean(#)', .5)
  ans:    2.4 3.4
```

But how reliable are these intervals themselves? To examine this question, we use jackknife-after-bootstrap:

```
≫  jab(data, 'confintervals(#, ''mean(#)'', .5)')
  ans:    1.7968 3.1532
          2.7929 4.2071
```

The lower end of the 50% confidence interval is (with 95% confidence) somewhere between 1.8 and 3.2. The upper end is between 2.8 and 4.2.

Suppose the data contained an outlier.

```
≫  data2 = [1 2 3 4 50];
```

Running `jab` on data2, we get a different picture.

```
≫  jab(data2, 'confintervals(#, ''mean(#)'', .5)')
  ans:    2.0429 3.4571
          0.2084 33.8916
```

We see that we have essentially no idea where the upper end of the 50% confidence interval is.

Example 1: Enough light-bulb data?

Pity the corporate executive who decided to offer a money-back guarantee on packages of light bulbs in Example **??**. The guarantee is that the total lifetime of the bulbs will be 4500 hours or greater. The executive's decision was based on a set of data of the lifetimes of lightbulbs tested in the company's quality control lab,

```
≫  data = [ 2103 2786 2543 1987 7 2380 3102 2452 3453 2543];
```

The executive used a 98% confidence interval on the sum of the lifetimes of the 4 light bulbs in a package:

```
≫  confintervals(data,'sum(#(1:4))',.98)
```

giving the interval 4497 hours to 12443. The number of trials was set fairly high, so these numbers seem reliable and setting the guarantee at 4500 hours seems like a safe bet: only 1% of packages will fail the guarantee. But is the data set large enough? We try jackknife-after-bootstrap:

```
≫  jab(data, 'confintervals(#,''sum(#(1:4))'',.98)')
2355 7402
11713 13080
```

We see that the left boundary of the 98% confidence interval is likely somewhere between 2355 and 7402 hours (with 95% confidence). This indicates that the value of 4500 is not very reliable. Since in this case the corporate executive is counting on not having too many violations

of the guarantee, there is a great deal of uncertainty about whether this will happen. Although 4500 was a reasonable 1% percentile for the exact values in the data set, this data set is only a small sample of all the lightbulbs produced by the factory. One light bulb, with lifetime 7 hours, has had a great influence on the results. The data set implies that such short-lived bulbs are only 10% of the output of the light-bulb factory, but the data set is too small to know that 10% with any reliability. Using jackknife-after-bootstrap provides an indication of this.

Having committed himself to a guarantee of 4500 hours, and having realized that his conclusions are not reliable, the executive starts to worry about how bad the damage might be. He wonders what fraction of packages will have bulbs with a total lifetime less than 4500 hours. The file `less4500.m` contains commands for doing a resampling simulation of this:

```
function res = less4500(values)
z = starttally;
for trial = 1:1000
   lifetime = sum(sample(4,data));
   tally lifetime z;
end
res = proportion(z<4500);
```

Running this script

```
≫  less4500(data)  ⇒  ans:   0.008
```

The result is approximately 1%, as the executive foresaw. However, jackknife-after-bootstrap tells a more informative story:

```
≫   jab(data, 'less4500(#)')
  ans:     -0.0014 0.0220
```

Given these data, the fraction of packages in violation of the warranty might be as high as 2.2% (with 95% confidence). It's time for the executive to cross his fingers. (The negative lower bound given by JAB doesn't make sense in terms of a fraction of lightbulbs. It arises because JAB bases its calculation on the standard deviation of the spread of jackknifed values.)

Soley for the purposes of illustration, we imagine the situation if the executive had asked for more data, and had received a second data set back that was identical to the original. Now the executive has twice as much data, but since the distribution of the data is unchanged the original conclusion of 4500 hours for a 1% percentile of the total package lifetime would be unchanged. But with twice as much data, jab indicates

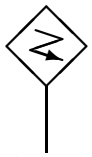
a narrower range of possible values.

```
≫  jab(concat(data,data), 'less4500(#)')
  ans:    0.0017 0.0187
```

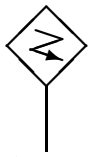
Trying this yet again, with 4 times as much data, the range is only a bit narrower: .03% to 1.6%. This still isn't enough data to be confident that the actual rate of guarantee violations will be less than 1%.

Whether a particular data set is large enough depends importantly on what you are calculating. In the light-bulb example (Example **??**), the data are not adequate for computing the fraction of packages that have a lifetime under 4500 hours to the accuracy required by the demands of corporate finance (that is, "don't lose money."). However, these same data might be quite adequate for computing a meaningful *median* or a *10% percentile* lifetime. Unfortunately, these are not a useful quantities for setting the bulb-lifetime quarantee. The point here is not that you should pick statistics that you're not interested in just because you happen to have enough data to make a meaningful estimation; rather, you should realize that the question of "enough data?" may have different answers for different statistics. If outliers in your data imply that the mean is not reliable, the median might be. (Or, the reverse might hold in some circumstances).

The computations in this section can take much, much longer than the other resampling computations. This is because each resampling computation is computed 10, or 100, or even 1000s of times, depending on the length of the data set.



There are no guarantees with the jackknife-after-bootstrap. If the jackknife-after-bootstrap shows no problem with the size of your dataset, that may just mean that your too-small data set didn't include any troublesome values, such as the 7 in the lightbulb data. This doesn't mean that the troublesome cases don't exist, just that your sample happened not to include them yet. A larger sample might. As a real-world example of this, consider the space-shuttle data of Example **??**: the fact that there was not a crash of the shuttle in first 24 flights does not mean that a crash was impossible.



5.3 Testing your programs

When you write a computer program using Resampling Stats or any other computer language, you intend it to perform some calculation and give an answer. You likely don't know the answer, otherwise you wouldn't need to write the program. How do you know whether the program is giving the right answer?

In the previous two sections we discussed problems that can arise from using too few trials or too few data points. Here we consider problems in the calculation itself and how to avoid them. Of course it would be impossible for use to tell you how to avoid all problems, but we can point out some of the common sources of problems and what to do about them.

1. Write instructions in a file so that you can edit and revise them and so that you have a record of what you did. When using MATLAB you can type commands at the prompt. For simple commands, such as
   ≫ `mean(data)`
   typing at the prompt is fine. When commands become complicated, though, it is much better to put the commands in a file and then execute those commands by giving the name of a file. Instructions for this are given in Sec. **??** (Steps 12 and 13). By putting commands in a file you avoid the need to retype the entire sequence each time you find a mistake; retyping almost invariably leads to further mistakes. Also, if you don't put commands in a file you will be reluctant to test them properly, which is essential to making your results reliable.

2. Test your program on a problem where you know the answer. Such testing is essential if you are to have any confidence that your programs are performing reliably. Of course, it's generally not easy to figure out some relevant data set where you know the answer. One systematic way to do this is to use simulation as in Sec. **??**.

3. Avoid writing unnecessary programs. For instance, if you want to compute confidence intervals of some statistic, Resampling Stats provides a general-purpose program, `confintervals`. To use this program you need to give three pieces of information: your data set, a means to compute the statistic, and a confidence level as in
   ≫ `confintervals( mydata, 'median(#)', 0.90 )`
   will compute the 90% confidence interval of the median of `mydata`.

Another general-purpose program described previously in this section, JAB, carries out the jackknife-after-bootstrap.

4. Look at intermediate results. For instance, rather than just looking at the percentiles of the tallying vector in a confidence interval calculation, look at the values of the tallying vector itself. If you see, for example, that there are only 2 points in the tallying vector, you can suspect a bug in your program: you haven't conducted enough trials.