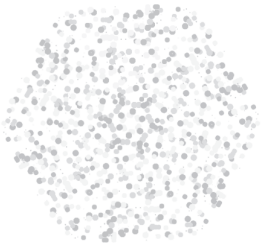
10



MAXIMUM LIKELIHOOD ESTIMATION

> Since the fabric of the universe is most perfect and the work of a most wise Creator, nothing at all takes place in the universe in which some rule of maximum or minimum does not appear.
>
> —Leonhard Euler[1]

Whether by divine design or human preference, problems involving the search for optima are everywhere. To this point, most models have had closed-form solutions for the optimal parameters, but if there is not a nice computational shortcut to finding them, you will have to hunt for them directly. There are a variety of routines to find optima, and Apophenia provides a consistent front-end to many of them via its `apop_maximum_likelihood` function.

Given a distribution $p(\cdot)$, the value at one input, $p(x)$, is *local information*: we need to evaluate the function at only one point to write down the value. However, the optimum of a function is *global information*, meaning that you need to know the value of the distribution at every possible input from $x = -\infty$ up to $x = \infty$ in order to know where the optimum is located.

This chapter will begin with the simple mathematical theory behind maximum likelihood estimation (*MLE*), and then confront the engineering question of how we can find the global optimum of a function by gathering local information about a small number of points. Once the prerequisites are in place, the remainder of the chapter covers MLEs in practice, both for description and testing.

[1] Letter to Pierre-Louis Moreau de Maupertuis, c. 1740–1744, cited in Kline (1980, p 66).

10.1 LOG LIKELIHOOD AND FRIENDS

To this point, we have met many probability distributions, whose PDFs were listed in Sections 7.2 and 9.2. This section takes such a probability distribution $P(\mathbf{X}, \boldsymbol{\beta})$ as given, and from that produces a few variant and derivative functions that will prove to be easier to work with. Also, a reminder: there is a list of notation for the book on page 12.

Let x_1 and x_2 be two *independent, identical draws* (iid) from a distribution. The independence assumption means that the joint probability is the product of the individual probabilities; that is, $P(\{x_1 \text{ and } x_2\}, \boldsymbol{\beta}) = P(x_1, \boldsymbol{\beta}) \cdot P(x_2, \boldsymbol{\beta})$. The assumption of identical distributions (i.e., that both are draws from the same distribution $P(\cdot, \boldsymbol{\beta})$) allows us to write this more neatly, as

$$P(\{x_1 \text{ and } x_2\}, \boldsymbol{\beta}) = \prod_{i=\{1,2\}} P(x_i, \boldsymbol{\beta}).$$

A probability function gives the probability that we'd see the data that we have given some parameters; a *likelihood function* is the probability that we'd see the specified set of parameters given some observed data. The philosophical implications of this distinction will be discussed further below.

There are three basic transformations of the likelihood function that will appear repeatedly, and are worth getting to know.

Define the *log likelihood function* as $LL \equiv \ln P(\mathbf{x}, \boldsymbol{\beta})|_x$, the *score* as its derivative with respect to $\boldsymbol{\beta}$:

$$\mathbf{S} \equiv \begin{bmatrix} \frac{\partial \ln P}{\partial \beta_1} \\ \vdots \\ \frac{\partial \ln P}{\partial \beta_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial LL}{\partial \beta_1} \\ \vdots \\ \frac{\partial LL}{\partial \beta_n} \end{bmatrix},$$

and the *information matrix* as the negation of derivative of the score with respect to $\boldsymbol{\beta}$.

$$\begin{aligned} \mathbb{I} &= -\frac{\partial \mathbf{S}}{\partial \boldsymbol{\beta}} \\ &= -\begin{bmatrix} \frac{\partial^2 LL}{\partial \beta_1^2} & \cdots & \frac{\partial^2 LL}{\partial \beta_n \beta_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 LL}{\partial \beta_1 \beta_n} & \cdots & \frac{\partial^2 LL}{\partial \beta_n^2} \end{bmatrix}. \end{aligned}$$

An example: Bernoulli Say that we have nine draws from a Bernoulli distribution, which you will recall from page 237 means that each draw is one with probability p and is zero with probability $1-p$, and say that in our case five draws were ones and four were zeros. The likelihood of drawing five ones is p^5; the likelihood of drawing four zeros is $(1-p)^4$; putting them together via the independence assumption, the likelihood of an arbitrary value of p given this data set $\mathbf{x}$ is

$$P(\mathbf{x}, p)|_x = p^5 \cdot (1-p)^4.$$

The log likelihood is thus

$$LL(\mathbf{x}, p) = 5\ln(p) + 4\ln(1-p).$$

The score, in this case a one-dimensional vector, is

$$S(\mathbf{x}, p) = \frac{5}{p} - \frac{4}{(1-p)}, \tag{10.1.1}$$

and the information value (a 1×1 matrix) is

$$\mathbb{I}(\mathbf{x}, p) = \frac{5}{p^2} + \frac{4}{(1-p)^2}.$$

Both intuitively and for a number of reasons discussed below, it makes sense to focus on the most likely value of p—that is, the value that maximizes $P(\mathbf{x}, p)$ given our observed data $\mathbf{x}$. Since the log is a monotonic transformation, p maximizes $P(\mathbf{x}, p)$ if and only if it maximizes $LL(\mathbf{x}, p)$. Recall from basic calculus that a necessary condition for a smooth function $f(x)$ to be at a maximum is that $\frac{df}{dx} = 0$—in the case of $LL(\mathbf{x}, p)$, that $S(\mathbf{x}, p) = 0$. Setting Equation 10.1.1 to zero and solving for p gives $p = \frac{5}{9}$.

But a zero derivative can indicate either a maximum or a minimum; to tell which, look at the second derivative. If the second derivative is negative when the first derivative is zero, then the first derivative is about to become negative—the function is beginning to dip downward. At this point, the function must be at a maximum and not a minimum.

Since the information matrix is defined as the negation of the score's derivative, we can check that we are at a maximum and not a minimum by verifying that the information value is positive—and indeed it is: for $p = 5/9$, $\mathbb{I} \approx 4.05$. In more dimensions, the analog is that the information matrix must be *positive definite*; see the exercise on page 269.

To summarize, given $p = \frac{5}{9}$, $P \approx 0.0020649$, $LL \approx -6.182$, $S = 0$, and $\mathbb{I} \approx 4.05$. It is easy to check other values of p to verify that they produce lower probabilities of observing the given data.

$\mathbb{Q}_{10.1}$

Verify the above statements.

- Generate a data set whose matrix element includes five ones and four zeros (in any order). Put the data set in a global variable.
- Produce an output model via `apop_estimate(`*`your_data`*`, apop_bernoulli)`.
- Display the output model via `apop_model_show`; check that the probability is as it should be (i.e., $5/9 = .\overline{555}$).
- Write a function that takes in an `apop_data` struct holding a vector with one item, and returns the log likelihood of that parameter given the global data, using `apop_log_likelihood`.
- Send that function to your `plotafunction` routine from page 191 (with the range $[0, 1]$) and verify that the log likelihood is at its highest where the parameter is $5/9$.

An example: Probit Recall the Probit model from page 283. It specified that an agent would act iff its utility from doing so is greater than a Normally-distributed error term. That is, act with probability

$$P(\mathbf{x}, \boldsymbol{\beta}) = \int_{-\infty}^{\mathbf{x}\boldsymbol{\beta}} \mathcal{N}(y|0,1)dy,$$

where $\mathcal{N}(y|0,1)$ indicates the standard Normal PDF at y (and so the integral is the CDF up to $\mathbf{x}\boldsymbol{\beta}$).

Reversing this, let $\mathbf{x}^A$ be the set of $\mathbf{x}$'s that led to action and $\mathbf{x}^N$ be the set that led to non-action. Then the likelihood of $\boldsymbol{\beta}$ given the data set is

$$P(\mathbf{x}, \boldsymbol{\beta})|_{\mathbf{x}} = \prod_{i\in A} P(\mathbf{x}_i^A, \boldsymbol{\beta}) \cdot \prod_{i\in N} \left(1 - P(\mathbf{x}_i^N, \boldsymbol{\beta})\right),$$

so

$$LL(\mathbf{x}, \boldsymbol{\beta})|_{\mathbf{x}} = \sum_{i\in A} \ln P(\mathbf{x}_i^A, \boldsymbol{\beta}) + \sum_{i\in N} \ln\left(1 - P(\mathbf{x}_i^N, \boldsymbol{\beta})\right).$$

By the way, the logit (Equation 8.3.1, page 284) tells the same story, but it simplifies significantly, to

$$LL(\mathbf{x}, \boldsymbol{\beta})|_{\mathbf{x}} = \sum_{i \in A} \mathbf{x}_i^A \boldsymbol{\beta} - \sum_{\forall i} \ln\left(1 + e^{\mathbf{x}_i \boldsymbol{\beta}}\right),$$

where the first term counts only those who act, while the second includes everybody. [ℚ: Verify the log likelihood using Equation 8.3.1.]

Unlike the binomial example above, we can not find the optimum of the log likelihood function for either the probit or logit using just a derivative and some quick algebra. We will instead need to use a search method such as those discussed later in this chapter.

※ A DIGRESSION: THE PHILOSOPHY OF NOTATION

The probability function has a *frequentist* interpretation: if you give me a fixed distribution, the story behind it, and a fixed parameter β, then after a few million draws, x will occur $P(x, \beta)|_{\beta} \cdot 100$ percent of the time. The likelihood function has no such interpretation, because we assume that the data was produced using one model that had a fixed β, that we happen to be ignorant of. There is no mysterious pool of β's from which ours was drawn with some probability.

Thus, the probability of x given β (and a model) is in many cases an objectively verifiable fact; the likelihood of β given x (and a model) is a subjective construct that facilitates various types of comparison among β's. The integral over all x is always one (i.e., for any fixed β, $\int_{\forall x} P(x, \beta) dx = 1$). The integral over all β of the likelihood function, however, could be anything.

Ronald Aylmer Fisher, the famed eugenicist and statistician whose techniques appear throughout this book, was vehement about keeping a clear distinction: "... [I]n 1922, I proposed the term 'likelihood,' in view of the fact that, with respect to [the parameter], it is not a probability, and does not obey the laws of probability, while at the same time it bears to the problem of rational choice among the possible values of [the parameter] a relation similar to that which probability bears to the problem of predicting events in games of chance.... Whereas, however, in relation to psychological judgment, likelihood has some resemblance to probability, the two concepts are wholly distinct...." (Fisher, 1934, p 287) See Pawitan (2001) for more on the interpretation of likelihood functions.

But as a simple practical matter, the probability of x given fixed parameter β is $P(x, \beta)$, and the likelihood of β given fixed data x is the very same $P(x, \beta)$. At the computer, there is no point writing down separate functions `p(x, beta)` and `likelihood(beta, x)`—a single function will serve both purposes. Just fix x to produce a likelihood function over β, and fix β to get a probability distribution of values of x.

We have two choices for notation, both of which can lead to confusion. The first is to use two distinct function names for probability and likelihood—$P(x)$ and $L(x)$ are typical—which clarifies the philosophical differences and leaves it to the reader to recognize that they are numerically identical, and that both are functions of x and β. The second option is to use a single function for both, which clarifies the computational aspects, but leaves the reader to ponder the philosophical implications of a single function that produces an objective probability distribution when viewed one way and a subjective likelihood function when viewed the other way.[2] Because this book is centrally focused on computation, it takes the second approach of listing both probability and likelihood using the same $P(x, \boldsymbol{\beta})$ form.

MORE ON LL, S, AND $\mathbb{I}$ The log of the likelihood function has a number of divine properties, which add up to making the log likelihood preferable to the plain likelihood in most cases—and wait until you see what the score can do.

First, due to all that exponentiation in the distributions of Sections 7.2 and 9.2, $\ln P$ is often much easier to deal with, yet is equivalent to $P(\cdot)$ for most of our purposes—notably, if we have found a maximum for one, the we have found a maximum for the other.

Also, consider calculating an iid likelihood function given a thousand data points. The probability of observing the data set will have the form $\prod_{i=1}^{1000} P(x_i)$. Since each $P(x_i) \in (0, 1]$, this product is typically on the order of 1×10^{-1000}. As discussed on page 137, such a number is too delicate for a computer to readily deal with. Taking logs, each value of p_i is now a negative number (e.g., $\ln(0.5) \approx -0.69$ and $\ln(0.1) \approx -2.3$), and the product above is now a sum:

$$\ln\left[\prod_{i=1}^{1000} P(x_i)\right] = \sum_{i=1}^{1000} \ln\left(P(x_i)\right).$$

Thus, the log likelihood of our typical thousand-point data set is on the order of -1000 instead of 1×10^{-1000}—much more robust and manageable. You saw an example of these different scales with the nine-point sample in the Bernoulli example, which had $p \approx 0.002$ but $LL \approx -6$.

Analytically, the maximum of the log likelihood function is useful for two reasons with four names: the Cramér–Rao lower bound and the Neyman–Pearson lemma. It all begins with this useful property of the score:[3]

[2] There are consistent means of describing subjective probability that accommodate both ways of slicing $P(x, \beta)$. The *subjectivist* approach (closely related to the *Bayesian* approach) takes all views of $P(x, \beta)$ as existing only in our minds—no matter how you slice it, there need not be a physical interpretation. The axiomatic approach, led by Ramsey, Savage, and de Finetti, posits a few rules that lead to 'consistent' subjective beliefs when followed, but places no further constraints on either probability or likelihood. Again, once both probability and likelihood are accepted as subjective beliefs, there is less reason to distinguish them notationally.

[3] All proofs here will be in the case where β is a scalar. Proofs for the multidimensional case are analogous but

Theorem 10.1.1. If $P(\mathbf{x}, \beta)$ satisfies certain regularity conditions as described in the footnote,[4] then for any statistic $f(x)$,

$$E_x(S \cdot f) = \frac{\partial E_x(f)}{\partial \beta}.$$

That is, the score is a sort of derivative machine: the expected value of the score times a statistic is equivalent to the derivative of the expected value of the statistic. Finding an optimum requires finding the point where the derivative is zero and the second derivative is negative, and this theorem gives us an easy trick for finding those derivatives. The next few pages will show how this trick is used.

When reading this theorem, it is worth recalling the sleight-of-notation from page 257: $f(x)$ is a function only of the data, but $E_x(f(x))$ (where x is produced using a certain model and parameters) is a function only of the parameters.

※**Proof:** The expected value of the score times the statistic is

$$\begin{aligned}
E_x(S \cdot f) &= \int_{\forall x} S(\beta) f(x) P(x, \beta) dx \\
&= \int_{\forall x} \frac{\partial \ln P(x, \beta)}{\partial \beta} f(x) P(x, \beta) dx && (10.1.2) \\
&= \int_{\forall x} \frac{\frac{\partial P(x,\beta)}{\partial \beta}}{P(x, \beta)} f(x) P(\beta, x) dx && (10.1.3) \\
&= \int_{\forall x} f(x) \frac{\partial P(x, \beta)}{\partial \beta} dx \\
&= \frac{\partial \left(\int_{\forall x} f(x) P(x, \beta) dx \right)}{\partial \beta} && (10.1.4) \\
&= \frac{\partial E_x(f(x))}{\partial \beta} && (10.1.5)
\end{aligned}$$

require more involved notation.

[4]Equation 10.1.4 of the proof uses the claim that $\int f \cdot \frac{dP}{d\beta} dx = \frac{d}{d\beta} \int f \cdot P dx$. If we can't reverse the integral and derivative like this, none of this applies.

The common explanation for when the switch is valid is in the case of any exponential family; the definition of an exponential family will not be covered in this book, but rest assured that it applies to the Normal, Gamma, Beta, Binomial, Poisson, et cetera—just about every distribution but the Uniform.

But it also applies more generally: we need only uniform convergence of the PDF as its parameters go to any given limit (Casella & Berger, 1990, Section 2.4). Roughly, this is satisfied for any PDF whose value and derivative are always finite. For those who prefer the exponential family story, note that any PDF can be approximated arbitrarily closely by a sum of exponential-family distributions (Barron & Sheu, 1991), so for any distribution that fails, there is an arbitrarily close distribution that works. For example, the Uniform$[\beta_1, \beta_2]$ distribution fails because of the infinite slope at either end, but a distribution with a steep slope up between $\beta_1 - 1\text{e}{-10}$ and β_1 and a steep slope down between β_2 and $\beta_2 + 1\text{e}{-10}$ works fine.

The sequence of events consisted of substituting in the definition of the score (Equation 10.1.2), then substituting the familiar form for the derivative of the log (Equation 10.1.3), and canceling out a pair of $P(x, \beta)$'s. At this point, the simple weighting $P(x)$ (first introduced on page 221) has been replaced with the weighting $dP(x)$. Before, if x_1 was twice as likely as x_2 (i.e., $P(x_1) = 2P(x_2)$), then $f(x_1)$ would get double weighting. Now, if the slope of $P(x)$ at x_1 is twice as steep as the slope at x_2, then $f(x_1)$ gets double weighting.

The final steps state that, under the right conditions, the integral of $f(x)$ using a measure based on $dP(x)$ is equivalent to the derivative of the integral of $f(x)$ using a measure based on $P(x)$. Equation 10.1.4 switched the integral and derivative, using the assumptions in the theorem's footnote, and Equation 10.1.5 recognized the integral as an expectation under the given probability density. ♦

Corollary 10.1.2.

$$E(S) = 0.$$

Proof: Let the statistic $f(x)$ be the trivial function $f(x) = 1$. Then Theorem 10.1.1 tells us that $E(S \cdot 1) = \partial E(1)/\partial\beta = 0$.

$\mathbb{Q}_{10.2}$ Verify that the expected value of the score is zero for a few of the distributions given in Chapter 7, such as the Exponential on page 248. (*Hint*: you will need to calculate the integral of the expected value of the score over the range from zero to infinity; integration by parts will help.)

Lemma 10.1.3. The *information equality*

$$\mathrm{var}(S) = E(S \cdot S) = E(\mathbb{I}).$$

※**Proof:** The first half comes from the fact that $\mathrm{var}(S) = E(S \cdot S) - E(S) \cdot E(S)$, but we just saw that $E(S) = 0$.

For the second half, write out $E(\mathbb{I})$, using the expansion of $S = \left(\frac{\frac{\partial P(x,\beta)}{\partial \beta}}{P(x,\beta)} \right)$ and the usual rules for taking the derivative of a ratio.

$$
\begin{aligned}
E\left[\frac{\partial^2 \ln P(x,\beta)}{\partial\beta}\right] &= E\left[\frac{\partial\left(\frac{\frac{\partial P(x,\beta)}{\partial\beta}}{P(x,\beta)}\right)}{\partial\beta}\right] \\
&= E\left[\frac{P(x,\beta)\frac{\partial^2 P(x,\beta)}{\partial\beta^2} - \left(\frac{\partial P(x,\beta)}{\partial\beta}\right)^2}{\left(P(x,\beta)\right)^2}\right] \\
&= E\left[\frac{\frac{\partial^2 P(x,\beta)}{\partial\beta^2}}{P(x,\beta)} - S\cdot S\right] \\
&= -E\left[S\cdot S\right]
\end{aligned}
$$

$\mathbb{Q}$: Prove the final step, showing that $E\left[\frac{\frac{\partial^2 P(x,\beta)}{\partial\beta^2}}{P(x,\beta)}\right] = 0$. (*Hint*: use the lessons from the proof of Theorem 10.1.1 to write the expectation as an integral and switch the integral and one of the derivatives.) ♦

The information equality will be computationally convenient because we can replace a variance, which can be hard to directly compute, with the square of a derivative that we probably had to calculate anyway.

For the culmination of the sequence, we need the Cauchy–Schwarz inequality, which first appeared on page 229. It said that the correlation between any two variables ranges between -1 and 1. That is,

$$
\begin{aligned}
-1 \le \rho(f,g) = \frac{\operatorname{cov}(f,g)}{\sqrt{\operatorname{var}(g)\operatorname{var}(f)}} &\le 1 \\
\frac{\operatorname{cov}(f,g)^2}{\operatorname{var}(g)\operatorname{var}(f)} &\le 1 \\
\frac{\operatorname{cov}(f,g)^2}{\operatorname{var}(g)} &\le \operatorname{var}(f). \qquad (10.1.6)
\end{aligned}
$$

Lemma 10.1.4. *The Cramér–Rao lower bound*

Let $f(\mathbf{x},\beta)$ be any statistic, and assume a distribution $P(\mathbf{x},\beta)$ that meets the criteria from the prior results. Then

$$
-\frac{\left(\partial E_x\left(f(\mathbf{x},\beta)\right)/\partial\beta\right)^2}{E(\mathbb{I})} \le \operatorname{var}(f(\mathbf{x},\beta)). \qquad (10.1.7)
$$

The proof consists of simply transforming the Cauchy–Schwarz inequality using the above lemmas. Let g in Equation 10.1.6 be the score; then the equation expands to

$$\frac{\left(E_x(f(\mathbf{x}) \cdot S) - E_x(f(\mathbf{x}))E_x(S)\right)^2}{\mathrm{var}(S)} \leq \mathrm{var}(f(\mathbf{x})) \tag{10.1.8}$$

The left-hand side has three components, each of which can be simplified using one of the above results:

- $E_x(f(\mathbf{x}, \beta) \cdot S) = \partial f(\mathbf{x}, \beta)/\partial\beta$, by Theorem 10.1.1.
- Corollary 10.1.2 said $E(S)$ is zero, so the second half of the numerator disappears.
- The information equality states that that $\mathrm{var}(S) = E(\mathbb{I})$.

Applying all these at once gives the Cramér–Rao lower bound.

Further, MLEs have a number of properties that let us further tailor the CRLB to say still more.[5] Let the statistic $f(\mathbf{x})$ be the maximum likelihood estimate of the parameter, $MLE(\mathbf{x}, \beta)$.

- MLEs can be biased for finite data sets, but can be shown to be asymptotically unbiased. This means that for n sufficiently large, $E(MLE(\mathbf{x}, \beta)) = \beta$. Therefore, $\partial MLE(\mathbf{x}, \beta)/\partial\beta = 1$, so the numerator on the left-hand side of Equation 10.1.7 is one.
- It can be proven that maximum likelihood estimators actually achieve the CRLB, meaning that in this case the inequality in Equation 10.1.7 is an equality.
- The information matrix is additive: If one data set gives us $\mathbb{I}_1$ and another gives us $\mathbb{I}_2$, then the two together produce $\mathbb{I}_{\mathrm{total}} = \mathbb{I}_1 + \mathbb{I}_2$. For a set of iid draws, each draw has the same amount of information (i.e., the expected information matrix, which is a property of the distribution, not any one draw of data), so the total information from n data points is $n\mathbb{I}$.

The end result is the following form, which we can use to easily calculate the covariance matrix of the MLE parameter estimates.

$$\mathrm{var}(MLE(\mathbf{x}, \beta)) = \frac{1}{nE_x(\mathbb{I})}. \tag{10.1.9}$$

Equation 10.1.9 makes MLEs the cornerstone of statistics that they are. Given that MLEs achieve the CRLB for large n, they are asymptotically efficient, and if we want to test the parameter estimates we find via a t or F test, there is a relatively

[5] See Casella & Berger (1990, pp 310–311) for formal proofs of the statements in this section.

easy computation for finding the variances we would need to run such a test.[6] For many models (simulations especially), we want to know whether the outcome is sensitive to the exact value of a parameter, and the information matrix gives us a sensitivity measure for each parameter.

HOW TO EVALUATE A TEST A hypothesis test can be fooled two ways: the hypothesis could be true but the test rejects it, or the hypothesis could be false but the test fails to reject it.

There is a balance to be struck between the two errors: as one rises, the other falls. But not all tests are born equal. If a hypothesis has a 50–50 chance of being true, then the coin-flip test, 'heads, accept; tails, reject' gives us a 50% chance of a Type I error and a 50% chance of a Type II error, but in most situations there are tests where both errors are significantly lower than 50%. By any measure we would call those better tests than the coin-flipping test.

Evaluation vocab

Here are some vocabulary terms; if you are in a stats class right now, you will be tested on this:

Likelihood of a *Type I error* $\equiv \alpha$: rejecting the null when it is true.

Likelihood of a *Type II error* $\equiv \beta$: accepting the null when it is false.

Power $\equiv 1 - \beta$: the likelihood of rejecting a false null.

Unbiased: $(1 - \beta) \geq \alpha$ for all values of the parameter. I.e., you are less likely to accept the null when it is false than when it is true.

Consistent: the power $\rightarrow 1$ as $n \rightarrow \infty$.

A big help in distinguishing Type I from Type II error is that one minus the Type II error rate has the surprisingly descriptive name of *power*. To a high-power telescope, every star is slightly different—some things that seem like stars are even galaxies or planets. But to a low-power lens like the human eye, everything just looks like a little dot. Similarly, a high-power test can detect distinctions where a low-power test fails to reject the null hypothesis of no difference. Or, for the more cynical: since most journals have limited interest in publishing null results, a high-power test increases the odds that you will be able to publish results. As you can imagine, researchers are very concerned with maximizing power.

THE NEYMAN–PEARSON LEMMA The Neyman–Pearson lemma (Neyman & Pearson, 1928a,b) states that a *likelihood ratio test* will have the minimum possible Type II error—the maximum power—of any test with a given level of α. After establishing this fact, we can select a Type I error level and be confident that we did the best we could with the Type II errors by using a likelihood ratio test.

[6] Of course, if we run one of the tests from Chapter 9 derived from the CLT, then we need to make sure the CLT holds for the maximum likelihood estimate of the statistic in question. This could be a problem for small data sets; for large data sets or simulations based on millions of runs, it is less of a concern.

Likelihood ratios Say the cost of a test that correctly accepts or rejects the hypothesis is zero, the cost to a Type I error is C_I, and the cost to a Type II error is C_{II}. Then it is sensible to reject H_0 iff the expected cost of rejecting is less than the expected cost of not rejecting. That is, reject H_0 iff $C_I P(H_0|\mathbf{x}) < C_{II} P(H_1|\mathbf{x})$. We can translate this cost-minimization rule into the ratio of two likelihoods.

Recall Bayes's rule from page 258:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

To apply Bayes's rule to the rejection test, set $A = H_0$ and $B = \mathbf{x}$, so $P(A|B) = P(H_0|\mathbf{x})$ and $P(B|A) = P(\mathbf{x}|H_0)$ (and similarly for H_1). Then:

$$C_I P(H_0|\mathbf{x}) < C_{II} P(H_1|\mathbf{x}) \tag{10.1.10}$$

$$C_I \frac{P(\mathbf{x}|H_0)P(H_0)}{P(\mathbf{x})} < C_{II} \frac{P(\mathbf{x}|H_1)P(H_1)}{P(\mathbf{x})} \tag{10.1.11}$$

$$c < \frac{P(\mathbf{x}|H_1)}{P(\mathbf{x}|H_0)} \tag{10.1.12}$$

Inequality 10.1.10 is the rejection rule from above; Inequality 10.1.11 uses Bayes's rule to insert the likelihood functions; Inequality 10.1.12 does some cross-division, canceling out the $P(\mathbf{x})$'s and defining the *critical value* $c \equiv C_I P(H_1)/C_{II} P(H_0)$, i.e., everything that doesn't depend on $\mathbf{x}$. If you tell me the shape of $P(\cdot|H_1)$ and $P(\cdot|H_0)$ and some number $\alpha \in (0, 1)$, then I can give you a value of c such that Inequality 10.1.12 is true with probability α.[7] The test will then be: gather the data, calculate the likelihood ratio on the right-hand side of Inequality 10.1.12, and reject H_0 iff the inequality is true.

The Neyman–Pearson lemma states that this test is the 'best' in the sense that for a Type I error fixed at α, the LR test minimizes the probability of a Type II error.[8] So we can design any test we like by just fixing α at a value with which we are comfortable (custom says to use 95 or 99%) and calculating a few likelihood functions, and we are assured that we did the best we could regarding Type II errors. Most standard tests can be expressed in a likelihood ratio form, and so Type II errors pretty much never get mentioned, since they're considered taken care of.[9]

[7] Alternatively, you could give me a ratio of costs C_I/C_{II} and I could again give you a value of c. Thus, one could draw a relation between the relative costs and the choice of α.

[8] For a proof, see e.g. Amemiya (1994, pp 189–191).

[9] Every test has a Type I and Type II error, but thanks to the Neyman–Pearson Lemma, we just describe a test using the Type I level, with phrases like *a test with 5% p-value*. The introductory chapter of Hunter & Schmidt (2004) is an excellent essay on how such description can be severely misleading. The extreme-case test *always fail to reject the null* has a 0% Type I error rate, but if the null hypothesis is false, then it is wrong 100% of the time.

- For any sufficiently well-specified model, you can find the probability that a given data set was produced via the model.
- If the data set consists of independent and identically distributed elements, then the likelihood is a product with one term for each data point. For both computational and analytic reasons, the log likelihood is easier to work with; the product then becomes a sum.

- The parameters that maximize the likelihood function for a model (or identically, maximize the log likelihood) will have the minimum variance among all unbiased estimators. The variance is a known quantity, given by the Cramér–Rao lower bound.
- Type I and Type II errors are complementary: as one goes up, the other generally goes down. However, given a Type I error level, different tests will have different Type II error levels. The Neyman–Pearson lemma guarantees that a likelihood ratio test has the minimum probability of Type II error for a fixed Type I error level.

10.2 DESCRIPTION: MAXIMUM LIKELIHOOD ESTIMATORS

Apophenia provides one function to find a model's optimum, `apop_maximum_likelihood`—but what a function it is. It provides a standardized interface to several types of optimization routines that take very different approaches toward finding optima. You will have to provide a log likelihood function, but if you are unable to provide the derivatives, the maximization routines will find them for you. Since the Cramér–Rao lower bound tells us the variance of a most-likely parameter, `apop_maximum_likelihood` will return a parametrized `apop_model` with the variances, along with other useful information. This section gives an overview of some standard optimization methods, and how to choose among them to raise the odds that they will find an optimum for your functions.

You may be wondering why you need to know these details. Isn't finding the optimum of a likelihood function a solved problem?

The answer is decidedly no. Most standard optimization algorithms are built to work well with a smooth, closed-form, globally concave function, such as finding the value of x that maximizes $f(x) = -x^2$. If your function more-or-less meets such conditions, the odds are good that the default optimization routine in any stats package of your choosing will work fine. But *anything* that produces a likelihood value could be a model: a simulation could be a model, where the likelihood is a function of how well the model matches a real-world data set. A dynamic program-

ming problem could be a model. The consumer choosing among goods at the end of Chapter 4.7 was a model. If your model has a stochastic element, has multiple equilibria, or otherwise fails to fulfill the expectation of being a simple globally concave function, then you will need to tailor a method and settings around the problem at hand.[10]

```
1  #include <apop.h>
2
3  double sin_square(apop_data *data, apop_model *m){
4      double x = apop_data_get(m->parameters, 0, -1);
5          return -sin(x)*gsl_pow_2(x);
6  }
7
8  apop_model sin_sq_model ={"-sin(x) times x^2",1, .p = sin_square};
```

Listing 10.1 A model to be optimized. Online source: `sinsq.c`.

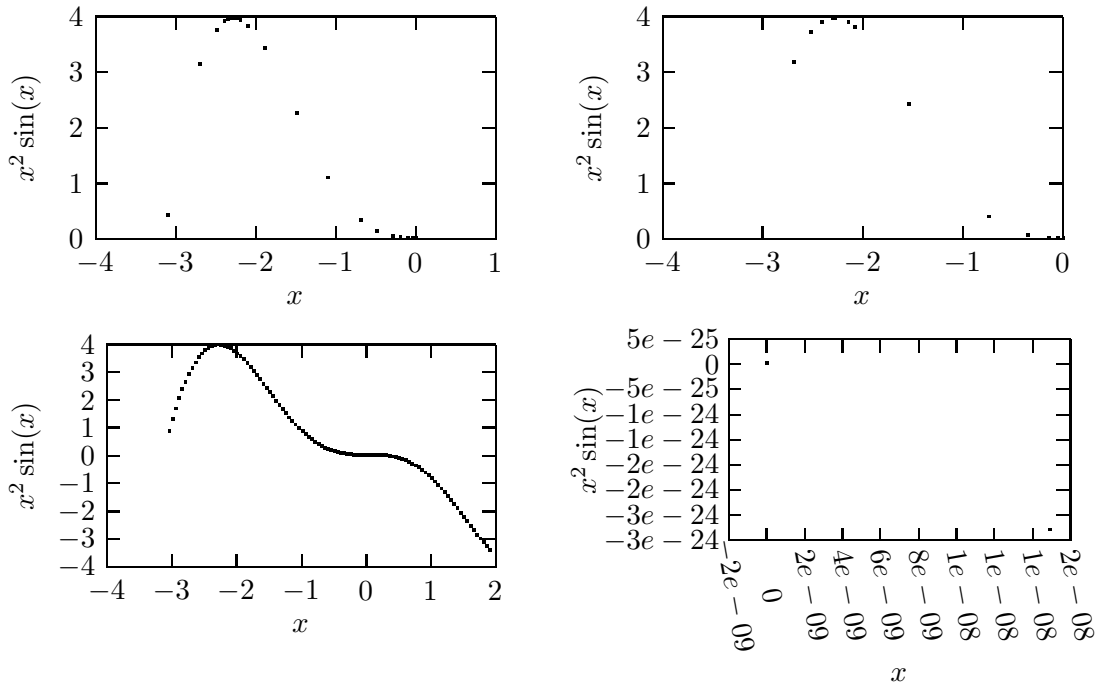


Figure 10.2 Top row: The simplex method and conjugate gradients; bottom row: simulated annealing and root search. [Online source: `localmax_print.c`]

[10]The problem of finding an optimum is so broad that there are large sections of optimization research that this book does not mention. For example, there is the broad and active field of *combinatorial optimization*, which covers questions like the optimal ordering of n elements given an optimization function (which is a problem with $n!$ options). See, e.g., Papadimitriou & Steiglitz (1998) for more on combinatorial optimization methods.

```
1  #include <apop.h>
2
3  apop_model sin_sq_model;
4
5  void do_search(int m, char *name){
6    double p[] = {0};
7    double result;
8      Apop_settings_add(&sin_sq_model, apop_mle, starting_pt, p);
9      Apop_settings_add(&sin_sq_model, apop_mle, method, m);
10     apop_model *out = apop_maximum_likelihood(NULL, sin_sq_model);
11     result = gsl_vector_get(out->parameters->vector, 0);
12     printf("The %s algorithm found %g.\n", name, result);
13 }
14
15 int main(){
16     apop_opts.verbose ++;
17     Apop_settings_add_group(&sin_sq_model, apop_mle, &sin_sq_model);
18     do_search(APOP_SIMPLEX_NM, "N-M Simplex");
19     do_search(APOP_CG_FR, "F-R Conjugate gradient");
20     do_search(APOP_SIMAN, "Simulated annealing");
21     do_search(APOP_RF_NEWTON, "Root-finding");
22 }
```

Listing 10.3 Using `apop_maximum_likelihood` and four types of method to solve for a maximum. Compile with `sinsq.c`. Online source: `localmax.c`.

Listing 10.1 presents a simple model, consisting of the equation $-x^2 \sin(x)$. This is a very simple equation, but it has an infinite number of local modes. As $x \to \infty$, the value of the function at these modes rises in proportion to x^2, so there is no global maximum. Figure 10.2 shows various attempts to search the function, one of which gives a very good picture of the shape of the curve.

- On lines 3–6, the function is defined. Because the system is oriented toward data analysis, the function takes in a data set and an `apop_model` holding the parameters. In this case, the data set is simply ignored.
- Line eight declares a new model with some of the information the MLE function will need, like the name, the number of parameters (one) and the probability function.

Listing 10.3 does the optimization four ways.

- The `apop_model` struct can hold an array of groups of settings. Line 17 adds to the model a group of MLE-appropriate settings. That group includes things like the starting point, method, tolerance, and many other details that you will see below.

- Lines eight and nine change the `starting_pt` and `method` elements of the model's MLE settings group to `p` and `m`, respectively.
- All the real work is done by line ten, which calls the maximization. The subsequent two lines interrogate the output.
- The `main` routine does the optimization with four different methods.

By inserting `Apop_settings_add(&sin_sq_model, apop_mle, trace_path, "`*outfile*`")` somewhere around line eight or nine, the system writes down the points it tests during its search for an optimum; you can then produce plots like those in Figure 10.2. You can already see disparate styles among the four methods: the simplex and conjugate gradient methods are somewhat similar in this case, but simulated annealing is much more exhaustive—you can clearly see the shape of the curve—while the root search barely leaves zero before deciding that it is close enough.

Chapter 11 will demonstrate another method of producing a picture of a function via random walk, but in the meantime, simulated annealing and `trace_path` provide a pretty effective way to get a graph of a complex and unfamiliar function.

When you run the program, you will see that asking four different methods gives you three different answers, which leads to an important lesson: *never trust a single optimization search*. By redoing the optimization at different points with different methods, you can get a better idea of whether the optimum found is just a local peak or the optimum for the entire function. See below for tips on restarting optimizations.

METHODS OF FINDING OPTIMA

Here is the problem: you want to find the maximum of $f(\mathbf{x})$, but only have time to evaluate the function and its derivative at a limited number of points. For example, $f(\mathbf{x})$ could be a complex simulation that takes in a few parameters and runs for an hour before spitting out a value, and you would like to have an answer this week.

Here are the methods that are currently supported by Apophenia. They are basically a standardization of the various routines provided by the GSL. In turn, the GSL's choice of optimization routines bears a close resemblance to those recommended by Press *et al.* (1988), so see that reference for a very thorough discussion of how these algorithms work, or see below for some broad advice on picking an algorithm. Also, Avriel (2003) provides a thorough, mathematician-oriented overview of optimization, and Gill *et al.* (1981) provides a more practical, modeler-oriented overview of the same topics.

Simplex method—Nelder–Mead `[APOP_SIMPLEX_NM]` For a d dimensional search, this method draws a polygon with $d+1$ corners and, at each step, shifts the corner of the polygon with the smallest function value to a new location, which may move the polygon or may contract it to a smaller size. Eventually, the polygon should move to and shrink around the maximum. When the average distance from the polygon midpoint to the $d+1$ corners is less than `tolerance`, the algorithm returns the midpoint.

This method doesn't require derivatives at all.

Conjugate gradient Including:
Polak–Ribiere `[APOP_CG_PR]`
Fletcher–Reeves `[APOP_CG_FR]`
Broyden–Fletcher–Goldfarb–Shanno `[APOP_CG_BFGS]`

Begin by picking a starting point and a direction, then find the minimum along that single dimension, using a relatively simple one-dimensional minimization procedure like Newton's Method. Now you have a new point from which to start, and the conjugate gradient method picks a new single line along which to search.

Given a direction vector $\mathbf{d}_1$, vector $\mathbf{d}_2$ is *orthogonal* iff $\mathbf{d}_1'\mathbf{d}_2 = 0$. Colloquially, two orthogonal vectors are at right angles to each other. After doing an optimization search along $\mathbf{d}_1$, it makes intuitive sense to do the next one-dimensional search along an orthogonal vector. However, there are many situations where this search strategy does not do very well—the optimal second direction is typically not at right angles to the first.

Instead, a *conjugate gradient* satisfies $\mathbf{d}_1'\mathbf{A}\mathbf{d}_2 = 0$, for some matrix $\mathbf{A}$. Orthogonality is the special case where $\mathbf{A} = \mathbf{1}$. For quadratic functions of the form $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}'\mathbf{A}\mathbf{x} - \mathbf{b}\mathbf{x}$, a search along conjugate gradients will find the optimum in as many steps as there are dimensions in $\mathbf{A}$. However, your function probably only approximates a quadratic form—and a different quadratic form at every point, meaning that for approximations at points one and two, $\mathbf{A}_1 \neq \mathbf{A}_2$. It is not necessary to actually calculate $\mathbf{A}$ at any point, but the quality of the search depends on how close to a quadratic form your function is; the further from quadratic, the less effective the method.

Polak–Ribiere and Fletcher–Reeves differ only in their choice of method to build the next gradient from the prior; Press *et al.* (1988) recommend Polak–Ribiere.

The BFGS method is slightly different, in that it maintains a running best guess about the *Hessian*, and uses that for updating the gradients.[11] However, the same

[11] Formally, one could argue that this means that it is not a conjugate gradient method, but I class it with the

general rules about its underlying assumptions hold: the closer your function is to a fixed function with smooth derivatives, the better BFGS will do.

Here is the pseudocode for the three algorithms. The variables in `teletype` are settings that can be tuned before calling the routine, as on lines 8 and 9 of Listing 10.3. For all routines, set `verbose` to `1` to see the progress of the search on screen.

Start at $p_0 =$ `starting_pt`, and gradient vector g_0.
While ($p_i \cdot g_i <$ `tolerance`$|p_i||g_i|$)
    Pick a new candidate point, $p_c = p_i + g_i \cdot$ `step_size`$/|g_i|$.
    If $p_c > p_i$
        $p_{i+1} \leftarrow p_c$
    Else
        $p_{i+1} \leftarrow$ the minimum point on the line between p_i and p_c (to within `tolerance`^{12}).
    Find g_{i+1} using a method-specific equation.

We are guaranteed that $p_{i+1} \neq p_i$ because we followed the gradient uphill. If the function continues uphill in that direction, then p_c will be the next point, but if the function along that path goes uphill and then downhill, the next point will be in between p_i and p_c.

The step to calculate g_{i+1} is the only step that differs between the three methods. In all three cases, it requires knowing derivatives of the function at the current point. If your model has no `dlog_likelihood` function, then the system will use a numerical approximation.

Root finding Including:
Newton's method `[APOP_RF_NEWTON]`
Hybrid method `[APOP_RF_HYBRID]`
Hybrid method; no internal scaling `[APOP_RF_HYBRID_NOSCALE]`

A root search finds the optimum of the function by searching for the point where the first derivative of the function to be maximized is zero. Notice that this can't distinguish between maxima and minima, though for most likelihood functions, the minima are at $\beta = \pm\infty$.[13]

FR and PR methods because all three routines bear a very close resemblance, and all share the same pseudocode.

[12]Do not confuse the tolerance in these algorithms with a promise (for example, the promise in a Taylor expansion) that if the tolerance is τ and the true value of the parameter is β, then the MLE estimate will be such that $|\hat{\beta}_{\text{MLE}} - \beta| < \tau$. There is no way to guarantee such a thing. Instead, the tolerance indicates when the internal measure of change (for most algorithms, $\Delta f(\beta)$) is small enough to indicate convergence.

[13]In fact, it is worth making sure that your models do not have a divergent likelihood, where the likelihood increases as $\beta \to \infty$ or $-\infty$. Divergent likelihoods are probably a sign of a misspecified model.

Let $f(\mathbf{x})$ be the function whose root we seek (e.g., the score), and let ∇f be the matrix of its derivatives (e.g., the information matrix). Then one step in Newton's method follows

$$x_{i+1} \leftarrow x_i - (\nabla f(x_i))^{-1} f(x_i).$$

You may be familiar with the one-dimensional version, which begins at a point x_1, and follows the tangent at coordinate $(x_1, f(x_1))$ down to the $x-$axis; the equation for this is $x_2 = x_1 - f(x_1)/f'(x_1)$, which generalizes to the multidimensional version above.

The hybrid edition of the algorithm imposes a region around the current point beyond which the algorithm does not venture. If the tangent sends the search toward infinity, the algorithm follows the tangent only as far as the edge of the trust region. The basic hybrid algorithm uses a trust region with a different scale for each dimension, which takes some extra gradient-calculating evaluations and could make the trust region useless for ill-defined functions; the no-scaling version simply uses the standard Euclidian distance between the current and proposed point.

The algorithm repeats until the function whose zero is sought (e.g., the score) has a value less than `tolerance`.

Simulated annealing `[APOP_SIMAN]` A controlled random walk. As with the other methods, the system tries a new point, and if it is better, switches. Initially, the system is allowed to make large jumps, and then with each iteration, the jumps get smaller, eventually converging. Also, there is some decreasing probability that if the new point is *less* likely, it will still be chosen. One reason for allowing jumps to less likely parameters is for situations where there may be multiple local optima. Early in the random walk, the system can readily jump from the neighborhood of one optimum to another; later it will fine-tune its way toward the optimum. Other motivations for the transition rules will be elucidated in the chapter on Monte Carlo methods.

Here is the algorithm in greater detail, with setting names in appropriate places:

Start with temp = `t_initial`
Let $x_0 \leftarrow$ `starting_point`
While temp $\geq$ `t_min`
    Repeat the following `iters_fixed_T` times:
        Draw a new point x_t, at most `step_size` units away from x_{t-1}.
        Draw a random number $r \in [0, 1]$.
        If $f(x_t) > f(x_{t-1})$
            Jump to the new point: $x_{t+1} \leftarrow x_t$.
        Else if $r < exp(-(x_{t-1} - x_t)/(\mathtt{k} \cdot \text{temp}))$
            Jump to the new point: $x_{t+1} \leftarrow x_t$.
        Else remain at the old point: $x_{t+1} \leftarrow x_{t-1}$.
    Cool: Let temp $\leftarrow$ temp/`mu_t`

Unlike with the other methods, the number of points tested in a simulated annealing run is not dependent on the function: if you give it a specification that reaches `t_min` in 4,000 steps, then that is exactly how many steps it will take. If you know your model is globally convex (as are most standard probability functions), then this method is overkill; if your model is a complex interaction, simulated annealing may be your only bet. It does not use derivatives, so if the derivatives do not exist or are ill-behaved, this is appropriate, but if they are available either analytically or via computation, the methods that use derivative information will converge faster.

If your model is stochastic, then methods that build up a picture of the space (notably conjugate gradient methods) could be fooled by a few early bad draws. Simulated annealing is memoryless, in the sense that the only inputs to the next decision to jump are the current point and the candidate. A few unrepresentative draws could send the search in the wrong direction for a while, but with enough draws the search could eventually meander back to the direction in which it should have gone.

Global v local optima As you saw in the case of $-x^2 \sin(x)$, none of the methods guarantee that the optimum found is the global optimum, since there is no way for a computer to have global knowledge of a function $f(x)$ for all $x \in (-\infty, \infty)$. One option is to restart the search from a variety of starting points, in the hope that if there are multiple peaks, then different starting points will rise to different peaks.

The simulated annealing algorithm deals well with multiple peaks, because its search can easily jump from the neighborhood of one peak to that of another. In fact, as the number of steps in the simulated annealing algorithm $\rightarrow \infty$, the algorithm can be shown to converge to the global optimum with probability one. However, calculating an infinite number of steps tends to take an unreasonable amount of time, so you will need to select a time-to-confidence trade-off appropriate to your situation.

$\mathbb{Q}_{10.3}$ Lines 13 and 14 of Listing 10.3 set the key parameters of the method and the starting point. Try various values of both. Which do a better job of jumping out toward the larger modes? [*Bonus*: rewrite the program to take command-line switches using `getopt` so you can do this exercise from a batch file.]

$\mathbb{Q}_{10.4}$ The `econ101` models from Chapter 4 provide the relatively rare situation where we have an optimization and the analytic values. [Hopefully your own simulations have at least one special case where this is also true.] This is therefore a fine opportunity to try various methods, values of delta, step size, tolerance, method, et cetera Do extreme prices or preferences create problems, and under which optimization settings?

RESTARTING To reiterate a recommendation from above: *never trust a single optimization search*. But there are a number of ways by which you can order your multiple searches.

- You could start with a large step size and wide tolerance, so the search jumps around the space quickly, then restart with smaller step size and tolerance to hone in on a result.
- Along a similar vein, different search methods have different stopping criteria, so switching between a simplex algorithm and a conjugate gradient algorithm, for example, may lead to a better optimum.
- If you suspect there could be multiple local optima, then try different starting points—either different extremes of the space, or randomly-generated points.
- If you are running a constrained optimization, and one of the constraints binds, then there may be odd interactions between the penalty and the function being optimized. Try a series of optimizations with the penalty declining toward zero, to see if the optimum gets closer to the boundary.
- The `apop_estimate_restart` function will help you to run sequences of optimizations; see the online reference for details.

- ➤ Given an appropriate `apop_data` set and `apop_model`, the `apop_maximum_likelihood` function will apply any of a number of maximum-searching techniques to find the optimal parameters.
- ➤ No computational method can guarantee a global optimum, because the computer can only gather local information about the function. Restarting the search in different locations may help to establish a unique optimum or find multiple optima.
- ➤ You can try various methods in sequence using `apop_estimate_restart`. You can also use the restarting technique to do a coarse search for the neighborhood of an optimum, and then a finer search beginning where the coarse search ended.

10.3 MISSING DATA

Say that your data set is mostly complete, but has an NaN in observation fifty, column three. When you run a regression, the NaN's propagate, and you wind up with NaN's all over the parameter estimates. How can you fill in the missing data?

We could turn this into an MLE problem: we seek the most likely values to fill in for the NaN's, based on some model of how the data was generated.

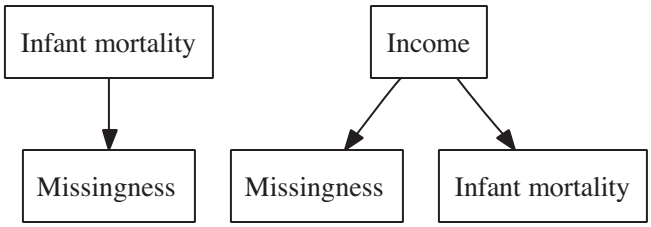


Figure 10.4 Two different flows of causation. At left is non-ignorable missingness: the value of the infant mortality statistic determines whether infant mortality data will be missing. At right is MAR: low income causes high infant mortality, and causes infant mortality data to be missing. [online source: `mar.dot`]

But first, we need to distinguish among three types of missingness. Data are *missing completely at random* (MCAR) when the incidence of missing data is uncorrelated to every variable in the data set. This is truly haphazard error: somebody tripped over the meter's power cord, or one of the surveyors was drunk. The cause of the missing data is nowhere in the data set.

Data are *missing at random* (MAR) when the incidence of missing data in column i is uncorrelated to the existing data in column i once we condition on the observed data in all other columns. For example, poor countries tend to have bad demographic data, so the incidence of a missing infant mortality rate is correlated to low GNP per capita. Once we have conditioned on GNP per capita, there is no reason to expect that missingness is correlated to infant mortality. The cause of the missing values in column i is something in the data set, but not column i. As in the right-hand diagram in Figure 10.4, there is no flow of causation from infant mortality to missingness.

Conversely, say that governments are embarrassed by high infant mortality rates. Statistics bureaux are under orders to measure the rate, but release the measure only if it falls below a certain threshold. In this case, the incidence of missing data is directly related to the value of the missing data. The cause of the missing data is the value of the data. This is known as *missing not at random* (MNAR) or *non-ignorable missingness*, and is a serious problem because it implies bias almost by definition.

There are many methods for dealing with censored or otherwise non-ignorable missingness discussed in many sources, such as Greene (1990). For a full discussion of the many types of missing data, see Allison (2002).

Listwise deletion One option for dealing with NaN's that are MCAR is listwise deletion. The idea here is supremely simple: if a row is missing data for any variable, then throw out the entire row. This is conceptually simple, does not impose any additional structure or model on the data, and can be executed in one line of code:

```
apop_data *nan_free_data = apop_data_listwise_delete(dirty_data);
```

Alternatively, see page 105 for the syntax to do listwise deletion on the SQL side.

But listwise deletion isn't always appropriate. As a worst-case situation, say that a survey has a hundred questions, and everybody filled out exactly 99 of them. By listwise deletion, we would throw out the entire data set.

But with listwise deletion, the data set is going to be shorter, meaning that we lose information, and if data is MAR (not MCAR), then throwing out observations with missing data means biasing the information among other variables. In the example above, if we throw out countries with missing infant mortality data, we would mostly be throwing out countries with low GDP per capita.

ML imputation This is where maximum likelihood estimation comes in (Dempster *et al.*, 1977). Let the missing data be $\boldsymbol{\beta}$, and the existing data (with holes) be $\mathbf{X}$, as usual. Then our goal is to find the most likely value of $\boldsymbol{\beta}$. The first step is to specify a model from which the data was allegedly generated, so that we can evaluate the likelihood of any given β. The norm is that the completed data set has a Multivariate Normal distribution, meaning that the n columns of the data are distributed as an n-dimensional bell curve with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ. However, it may make sense for your data to take on any of a number of other forms. But given a parametrized distribution, one could search for the data points that are most likely to have occurred.

In the `data-corruption.db` database, you will find Transparency International's Corruption Perceptions Index from 2003–2006. Because the index depends on about a dozen hard-to-gather measures, there are many missing data points. Listing 10.5 goes through the entire process of filling in those data points, by pulling the data from the database, reducing it to an estimable subset via listwise deletion, fitting a Multivariate Normal to the subset, and then filling in the NaN's in the full data set via maximum likelihood. It may run slowly: filling in about eighty NaN's means a search in an 80-dimensional space. For more missing data than this, you are probably better off finding a means of dividing the data set or otherwise incrementally filling in the blanks.

You are encouraged to look at the results and decide whether they seem plausible.

```
#include <apop.h>

int main(){
    apop_db_open("data-corruption.db");
    apop_data *corrupt = apop_db_to_crosstab("cpi", "country", "year", "score");
    apop_data *clean = apop_data_listwise_delete(corrupt);
    apop_model *mlv = apop_estimate(clean, apop_multivariate_normal);
    apop_ml_imputation(corrupt, mlv);
    apop_crosstab_to_db(corrupt, "cpi_clean", "country", "year", "score");
}
```

Listing 10.5 Filling in NaN's via a Multivariate Normal. Online source: `corrupt.c`.

For example, would you use the data for Yugoslavia? Is a Multivariate Normal the most appropriate model of how the data was formed?

On lengthy surveys, few if any people successfully fill out the entire form. In the worst case, we may have 100 questions, and all subjects answered 99 of them. Listwise deletion would throw out every subject. In this case, the best bet is *pairwise deletion*: calculate the mean of each vector by just ignoring NaN's, and the covariance of each pair of columns by removing only those observations that have an NaN for one of those two variables. Pairwise deletion can introduce odd biases in the covariance matrix, so it should be used as a last resort.

10.4 TESTING WITH LIKELIHOODS

In order to test a hypothesis regarding a statistic, we need to have a means of describing its theoretical distribution. When the statistic is an MLE, the standard means of doing this is via two interlocking approximations: a Taylor expansion and a Normal approximation. This is convenient because the Normal approximation proves to be innocuous in many settings, and the Taylor expansion involves the same cast of characters with which we have been dealing to this point—LL, S, and $\mathcal{I}$.

USING THE INFORMATION MATRIX

The Cramér–Rao lower bound gives us a value for the variance of the parameters: the inverse of the expected information matrix. Given a variance on each parameter, we can do the same t and F tests as before.

It is even easier if we make one more approximation. The *expected* information matrix is an expectation over all possible parameters, which means that it is a property of the model, not of any one set of parameters. Conversely, the *estimated*

information matrix is the derivative of the score around the most likely values of the parameters. We can expect that it is different for different parameter estimates.

Efron & Hinkley (1978) found that for most of the applications they consider, the inverse of the estimated information matrix is preferable as an estimator of the variance as the expected information matrix. For *exponential family* distributions, the two are identical at the optimum. From a computational perspective, it is certainly preferable to use the estimated information, because it is a local property of the optimum, not a global property of the entire parameter space. Simulated annealing does a decent job of sampling the entire space, but the other methods go out of their way to not do so, meaning that we would need to execute a second round of data-gathering to get variances. Apophenia's maximum likelihood estimation returns a covariance matrix constructed via the estimated information matrix.

The covariance matrix provides an estimate of the stability of each individual parameter, and allows us to test hypotheses about individual parameters (rather than tests about the model as a whole, as done by the likelihood ratio methods discussed below).[14] However, there are a number of approximations that had to be made to get to this point. Basically, by applying a t test, we are assuming that a few million draws of a parameter's MLE (generated via a few million draws of new data) would be asymptotically Normally distributed. We already encountered this assumption earlier: when testing parameters of a linear regression we assume that the errors are Normally distributed. So the same caveats apply, and if you have a means of generating several data sets, you could test for Normality; if you do not, you could use the methods that will be discussed in Chapter 11 to bootstrap a distribution; or if you are working at a consulting firm, you could just assume that Normality always holds.

There is no sample code for this section because you already know how to run a t test given a statistic's mean and its estimated variance.

USING LIKELIHOOD RATIOS We can use likelihood ratio (LR) tests to compare models. For example, we could claim that one model is just like another, but with the constraint that $\beta_{12} = 0$, and then test whether the constraint is actually binding via the ratio of the likelihood with the constraint and the likelihood without. Or, say that we can't decide between using an OLS model or a probit model; then the ratio of the likelihood of the two models can tell us the confidence with which one is more likely than the other.

[14] In Klemens (2007), I discuss at length the utility of the variance of the MLE as a gauge of which of a simulation's parameters have a volatile effect on the outcome and which have little effect.

A loose derivation As intimated by the Neyman–Pearson lemma, the ratio of two likelihoods is a good way to test a hypothesis. Given the ratio of two likelihoods P_1/P_2, the log is the difference $\ln(P_1/P_2) = LL_1 - LL_2$.

Now consider the Taylor expansion of a log likelihood function around $\hat{\beta}$. The Taylor expansion is a common means of approximating a function $f(x)$ via a series of derivatives evaluated at a certain point. For example, the second-degree Taylor expansion around seven would be $f(x) \approx f(7)+(x-7)f'(7)+(x-7)^2 f''(7)/2+\epsilon$, where ϵ is an error term. The approximation is exactly correct at $x = 7$, and decreasingly precise (meaning ϵ gets larger) for values further from seven. In the case of the log likelihood expanded around $\hat{\beta}$, the Taylor expansion is

$$LL(\beta) = LL(\hat{\beta}) + (\beta - \hat{\beta})LL'(\hat{\beta}) + \frac{(\beta - \hat{\beta})^2}{2}LL''(\hat{\beta}) + \epsilon.$$

As per the definitions from the beginning of the chapter, the derivative in the second term is the score, and the second derivative in the third term is $-\mathbb{I}$. When $\hat{\beta}$ is the optimum, the score is zero. Also, as is the norm with Taylor expansions, we will assume $\epsilon = 0$. Then the expansion simplifies to

$$LL(\beta) = LL(\hat{\beta}) - \frac{(\beta - \hat{\beta})^2}{2}\mathbb{I}(\hat{\beta}). \tag{10.4.1}$$

Typically, the likelihood ratio test involves the ratio of an unrestricted model and the same model with a constraint imposed. Let LL_c be the constrained log likelihood; then we can repeat Equation 10.4.1 with the constrained log likelihood:

$$LL_c(\beta) = LL_c(\hat{\beta}) - \frac{(\beta - \hat{\beta})^2}{2}\mathbb{I}_c(\hat{\beta}).$$

Now the hypothesis: the constraint is not binding, and therefore both constrained and unconstrained optimizations find the same value of $\hat{\beta}$. Then

$$\begin{aligned} -2(LL(\beta) - LL_c(\beta)) &= 2LL_c(\hat{\beta}) - 2LL(\hat{\beta}) + (\beta - \hat{\beta})^2\mathbb{I}(\hat{\beta}) - (\beta - \hat{\beta})^2\mathbb{I}_c(\hat{\beta}) \\ &= (\beta - \hat{\beta})^2 \left(\mathbb{I}(\hat{\beta}) - \mathbb{I}_c(\hat{\beta})\right) \end{aligned} \tag{10.4.2}$$

The second equation follows from the first because having the same value for $\hat{\beta}$ for constrained and unconstrained optimizations means that $LL(\hat{\beta}) = LL_c(\hat{\beta})$.

But we still haven't said anything about the distribution of $-2(LL(\beta) - LL_c(\beta))$. Consider the case of the Normal distribution with fixed σ (so the only free param-

eter is the mean μ); there, the Score is

$$S(\mathbf{x}, \mu) = \sum_i (x_i - \mu)/\sigma^2. \tag{10.4.3}$$

$\mathbb{Q}$: Verify this by finding $\partial LL(\mathbf{x}, \mu)/\partial\mu$ using the probability distribution on page 241.

The right-hand side of Equation 10.4.3 takes the familiar mean-like form upon which the CLT is based, and so is is Normally distributed. Since $E(\mathbb{I}) = E(S \cdot S)$, and a Normally-distributed statistic squared has a χ^2 distribution, Expression 10.4.2 has a χ^2 distribution.

And in fact, this holds for much more than a Normal likelihood function (Pawitan, 2001, p 29). Say that there exists a transformation function $t(x, \beta)$ such that $P(x, \beta) \cdot t(x, \beta)$ is Normally distributed. Then

$$\frac{P(x, \beta) \cdot t(x, \beta)}{P_c(x, \beta) \cdot t(x, \beta)} = \frac{P(x, \beta)}{P_c(x, \beta)}.$$

Instead of canceling out the transformation here, we could also cancel it out in the log likelihood step:

$$LL(x, \beta) + t(x, \beta) - LL_c(x, \beta) - t(x, \beta) = LL(x, \beta) - LL_c(x, \beta).$$

Either way, Expression 10.4.2 is the same with or without the transformation—which means the untransformed version is also $\sim \chi^2$. So provided the likelihood function is sufficiently well-behaved that $t(x, \beta)$ could exist, we don't have to worry about deriving it. This is a specific case of the *invariance principle* of likelihood functions, that broadly says that transformations of the likelihood function do not change the information embedded within the function.

This is what we can use to do the likelihood ratio tests that the Neyman–Pearson lemma recommended. We find the log likelihood of the model in its unconstrained and constrained forms, take two times the difference, and look up the result in the χ^2 tables.

The LR test, constraint case As above, the typical likelihood ratio test involves the ratio of an unrestricted and a restricted model, and a null hypothesis that the constraint is not binding. Let P be the (not-log, plain) likelihood of the overall model, and P_c be the likelihood of a model with K restrictions, such as K parameters fixed as zero.

In this context, the above discussion becomes

$$-2 \ln \frac{P}{P_c} = -2[\ln P - \ln P_c] \sim \chi^2_K. \tag{10.4.4}$$

In modeling terms, the unrestricted model could be any of the models discussed earlier, such as the `apop_probit`, `apop_normal`, or even `apop_ols`, because the

OLS parameters ($\boldsymbol{\beta}_{\mathrm{OLS}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{y}$) can be shown to be identical to the maximum likelihood estimate of $\boldsymbol{\beta}$.

```
1  #include "eigenbox.h"
2
3  double linear_constraint(apop_data * d, apop_model *m){
4        apop_data *constr = apop_line_to_data((double[]) {0, 0, 0, 1}, 1, 1, 3);
5        return apop_linear_constraint(m->parameters->vector, constr, 0);
6  }
7
8  void show_chi_squared_test(apop_model *unconstrained, apop_model *constrained, int
         constraints){
9        double statistic = 2 * (unconstrained->llikelihood - constrained->llikelihood);
10       double confidence = gsl_cdf_chisq_P(statistic, constraints);
11       printf("The Chi squared statistic is: %g, so reject the null of non-binding constraint "
12            "with %g%% confidence.\n", statistic, confidence*100);
13 }
14
15 int main(){
16     apop_data *d = query_data();
17     apop_model *unconstr = apop_estimate(d, apop_ols);
18     apop_model_show(unconstr);
19
20     Apop_settings_add_group(&apop_ols, apop_mle, &apop_ols);
21     Apop_settings_add(&apop_ols, apop_mle, starting_pt, unconstr->parameters->vector->
           data);
22     Apop_settings_add(&apop_ols, apop_mle, use_score, 0);
23     Apop_settings_add(&apop_ols, apop_mle, step_size, 1e-3);
24     apop_ols.estimate = NULL;
25     apop_ols.constraint = linear_constraint;
26     apop_model *constr = apop_estimate(d, apop_ols);
27     printf("New parameters:\n");
28      apop_vector_show(constr->parameters->vector);
29     show_chi_squared_test(unconstr, constr, 1);
30 }
```

Listing 10.6 Comparing three different models using likelihood ratios: an OLS model, an OLS model with constraint, and a logit model. Online source: `lrtest.c`.

Listing 10.6 presents an unconstrained and a constrained optimization. It uses the query from page 267 that produces a data set whose outcome variable is males per 100 females, and whose independent variables are population and median age. The question *is the coefficient on median age significant?* can be rephrased to: *if we constrain the median age coefficient to zero, does that have a significant effect on the log likelihood?*

- The unconstrained optimization, on line 17, is the ordinary least squares model (which, as above, finds the MLE).

- Lines 20–24 mangle the base OLS model into a constrained model estimated via maximum likelihood. By setting the `estimate` element to `NULL` the estimation on line 25 uses the default method, which is maximum likelihood estimation.
- The constraint function is on line 3, and it uses the `apop_linear_constraint` function to test that an input vector satisfies a constraint expressed as an `apop_data` set of the type first introduced on page 152. In this case, the constraint is $0 < \beta_3$; since the unconstrained OLS estimation finds that $\beta_3 < 0$, this is a binding constraint.
- Line four uses a new syntactic trick: anonymous structures. The type-in-parens form, `(double [])`, looks like a type cast, and it basically acts that way, declaring that the data in braces is a nameless array of `doubles`. The line is thus equivalent to two lines of code, as at the top of the `main` routine in the `ftest.c` program on page 311:

```
double tempvar[] = {0, 0, 0, 1};
apop_line_to_data(tempvar, 1, 1, 3);
```

  But we can get away with packing it all onto one line and not bothering with the temp variable. When used in conjunction with designated initializers, anonymous structs can either convey a lot of information onto one line or make the code an unreadable mess, depending on your æsthetic preferences.
- By commenting out the constraint-setting on line 24, you will have an unconstrained model estimated via maximum likelihood, and can thus verify that the OLS parameters and the MLE parameters are identical.
- You will recognize the function on line nine as a direct translation of Expression 10.4.4. It is thus a test of the claim that the constraint is not binding, and it rejects the null with about the same confidence with which the t test associated with the linear regression rejected the null that the third parameter is zero.
- The statistic on line nine, $LL - LL_c$, is always positive, because whatever optimum the constrained optimization found could also be used by the unconstrained model, and the unconstrained model could potentially find something with even higher likelihood. If this term is negative, then it is a sign that the unconstrained optimization is still far from the true optimum, so restart it with a new method, new starting point, tighter tolerances, or other tweaks.

Be sure to compare the results of the test here with the results of the F test on page 311.

The LR test, non-nested case The above form is a test of two *nested* models, where one is a restricted form of the other, so under the hypothesis of the nonbinding constraint, both can find the same estimate $\hat{\beta}$ and so both can conceivably arrive at the same log likelihood. If this is not the case,

then the cancellation of the first part of the Taylor expansion in Equation 10.4.2 does not happen.

In this case (Cox, 1962; Vuong, 1989), the statistic and its distribution is

$$\frac{\ln \frac{P_1}{P_2} - E\left(\ln \frac{P_1}{P_2}\right)}{\sqrt{n}} \sim \mathcal{N}(0,1). \tag{10.4.5}$$

The denominator is simply the square root of the sample size. The first part of the numerator is just $LL_1 - LL_2$, with which we are very familiar at this point. The expected value is more problematic, because it is a global value of the log likelihoods, which we would conceivably arrive at by a probability-weighted integral of $LL_1(\beta) - LL_2(\beta)$ over the entire space of βs.

Alternatively, we could just assume that it is zero. That is, the easiest test to run with Expression 10.4.5 is the null hypothesis of no difference between the expected value of the two logs.

```
#define TESTING
#include "dummies.c"

void show_normal_test(apop_model *unconstrained, apop_model *constrained, int n){
    double statistic = (unconstrained->llikelihood - constrained->llikelihood)/sqrt(n);
    double confidence = gsl_cdf_gaussian_P(fabs(statistic), 1); //one-tailed.
    printf("The Normal statistic is: %g, so reject the null of no difference between models "
        "with %g%% confidence.\n", statistic, confidence*100);
}

int main(){
    apop_db_open("data-metro.db");
    apop_model *m0 = dummies(0);
    apop_model *m1 = dummies(1);
    show_normal_test(m0, m1, m0->data->matrix->size1);
}
```

Listing 10.7 Compare the two Metro ridership models from page 282 Online source: `lrnonnest.c`.

Listing 10.7 reads in the code for the two OLS estimations of Washington Metro ridersiop from page 282, one with a zero-one dummy and one with a dummy for the year's slope.

- If you flip back to the `dummies.c` file, you will see that the `main` function is wrapped by a preprocessor if-then statement: `#ifndef TESTING`. Because `TESTING` is defined here, the `main` function in that file will be passed over.
- Therefore, the next line can read in `dummies.c` directly, without ending up with two `main`s.

- The `main` function here simply estimates two models, and then calls the `show_normal_test` function, which is a translation of Expression 10.4.5 under the null hypothesis that $E(LL_{\text{OLS}} - LL_{\text{logit}}) = 0$.

  Remember, a number of approximations underly both the nested and non-nested LR tests. In the nested case, they are generally considered to be innocuous and are rarely verified or even mentioned. For the non-nested probit and logit models, their log likelihoods behave in a somewhat similar manner (as $n \to \infty$), so it is reasonable to apply the non-nested statistic above. But for two radically different models, like an OLS model versus an agent-based model, the approximations may start to strain. You can directly compare the two log-likelihoods, and the test statistic will give you a sense of the scale of the difference, but from there it is up to you to decide what these statistics tell you about the two disparate models.