Chapter 9

Nonlinear Inverse Problems

9.1 PARAMETERIZATIONS

Variables representing the data and model parameters—the *parameterization*—must be selected at the start of any inverse problem. In many instances, this selection is rather *ad hoc*; there might be no strong reasons for selecting one parameterization over another. This can become a substantial issue in nonlinear inverse problems because the answer obtained by solving it is dependent on the parameterization. In other words, the solution is not invariant under nonlinear transformations of the variables. This is in contrast to the linear inverse problem with Gaussian statistics, in which solutions are invariant for any linear reparameterization of the data and model parameters.

As an illustration of this difficulty, consider the problem of fitting a straight line to the data pairs (1, 1), (2, 2), (3, 3), (4, 5). Suppose that we regard these data as (z, d) pairs, where z is an auxiliary variable. The least-squares fit is $d = -0.500 + 1.300z$. On the other hand, we might regard them as (d', z') pairs, where z' is the auxiliary variable. Least squares then gives $d' = 0.457 + 0.743z'$, which can be rearranged as $z' = -0.615 + 1.346d'$. These two straight lines have intercepts that differ by 20% and slopes that differ by 4% (see *MatLab* script gda09_01).

This discrepancy arises from two sources. The first is an inconsistent application of probability theory. In the example above, we alternately assumed that z was exactly known but d contained Gaussian noise and that d was exactly known but z contained Gaussian noise. These are two radically different assumptions about the distribution of errors, so it is no wonder that the solutions are different.

This first source of discrepancy can in theory be avoided completely by recognizing and taking into account the fact that reparameterizing a problem changes the associated probability density functions. For instance, consider an inverse problem in which there is a model parameter m that is known to possess a uniform probability density function $p(m)$ on the interval [0, 1] (Figure 9.1A). If the inverse problem is reparameterized in terms of a new model parameter $m' = m^2$, then the transformed probability density function $p(m')$ can be calculated as

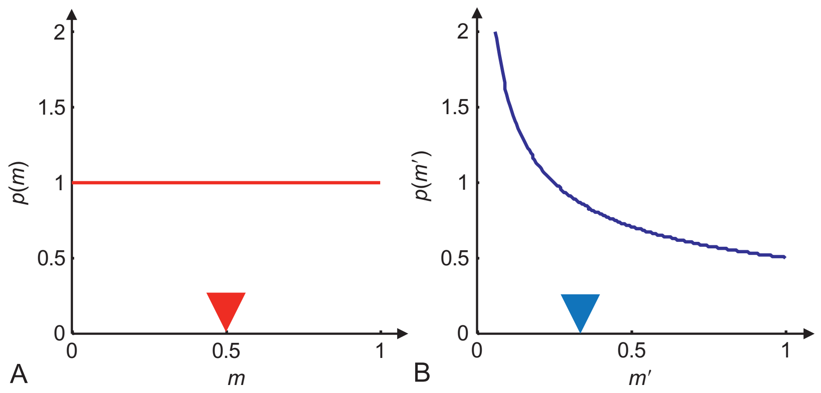

FIGURE 9.1 (A) A probability density function, $p(m)$, that is uniform on the interval $0<m<1$. (B) The corresponding probability distribution, $p(m')$, for the transformation $m'=m^2$. The mean (expectation) of each probability density function is indicated by a triangle. *MatLab* script gda09_02.

$$p(m)\mathrm{d}m = p[m(m')]\left|\frac{\mathrm{d}m}{\mathrm{d}m'}\right|\mathrm{d}m' = p(m')\mathrm{d}m' \quad \text{so} \quad p(m') = \frac{1}{2}m'^{-1/2} \qquad (9.1)$$

The distribution of m' is not uniform (Figure 9.1B), and any inverse method developed to solve the problem under this parameterization must account for this fact.

The second source of discrepancy is more serious. Suppose that we could use some inverse theory to calculate the distribution of the model parameters under a particular parameterization. We could then use Equation (9.1) to find their distribution under any arbitrary parameterization. Insofar as the distribution of the model parameters is the *answer* to the inverse problem, we would have the correct answer in the new parameterization. Probability distributions are invariant under changes of parameterization. However, a distribution is not always the answer for which we are looking. More typically, we need an estimate (a single number) based on a probability distribution (for instance, its maximum likelihood point or mean value).

Estimates are *not* invariant under changes in the parameterization. For example, suppose $p(m)$ has a uniform distribution as above. Then, if $m'=m^2$, $p(m') = \frac{1}{2}m'^{-1/2}$. The distribution in m has no maximum likelihood point, whereas the distribution in m' has one at $m'=m=0$. The distributions also have different means (expectations):

$$\begin{aligned} \langle m \rangle &= \int_0^1 m\, p(m)\mathrm{d}m = \int_0^1 m\mathrm{d}m = 1/2 \\ \langle m' \rangle &= \int_0^1 m' p(m')\mathrm{d}m' = \int_0^1 \frac{1}{2}m'^{1/2}\mathrm{d}m' = 1/3 \end{aligned} \qquad (9.2)$$

Even though m' equals the square of the model parameter m, the mean (expectation) of m' is not equal to the square of the expectation of m, that is, $(1/3) \neq (1/2)^2$.

There is some advantage, therefore, in working explicitly with probability distributions as long as possible, forming estimates only at the last step. If $\mathbf{m}$ and $\mathbf{m}'$ are two different parameterizations of model parameters, we want to avoid as much as possible sequences like

$$\text{p.d.f. for } \mathbf{m} \to \text{estimate of } \mathbf{m} \to \text{estimate of } \mathbf{m}' \tag{9.3}$$

in favor of sequences like

$$\text{pdf for } \mathbf{m} \to \text{pdf for } \mathbf{m}' \to \text{estimate of } \mathbf{m}' \tag{9.4}$$

Note, however, that the mathematics for this second sequence is typically much more difficult than that for the first.

There are objective criteria for the "goodness" of a particular estimate of a model parameter. Suppose that we are interested in the value of a model parameter $\mathbf{m}$. Suppose further that this parameter is either deterministic with a true value or (if it is a random variable) has a well-defined distribution from which the true expectation could be calculated if the distribution were known. Of course, we cannot know the true value; we can only perform experiments and then apply inverse theory to derive an estimate of the model parameter. As any one experiment contains noise, the estimate we derive will not coincide with the true value of the model parameter. But we can at least expect that if we perform the experiment enough times, the estimated values will scatter about the true value. If they do, then the method of estimating is said to be *unbiased*. Estimating model parameters by taking nonlinear combinations of estimates of other model parameters almost always leads to bias.

9.2 LINEARIZING TRANSFORMATIONS

One of the reasons for changing parameterizations is that an inverse problem can sometimes be transformed into a simpler form that can be solved by a known method. The problems that most commonly benefit from such transformations involve fitting exponential and power functions to data. Consider a set of (z, d) data pairs that are thought to obey the model $d_i = m_1 \exp(m_2 z_i)$. By making the transformation

$$m_1' = \log(m_1) \quad \text{and} \quad m_2' = m_2 \quad \text{and} \quad d_i' = \log(d_i) \tag{9.5}$$

we can write the model as the linear equation $d_i' = m_1' + m_2' z_i$, which can be solved by simple least-squares techniques. However, we must assume that the d_i' are independent random variables with a Gaussian probability density function with uniform variance in order to justify rigorously the application of least-squares techniques to this problem. The distribution of the data in their original parameterization must therefore be non-Gaussian. (It must have a *log-normal* probability density function.)

Notice that, if the exponential decays with increasing z for all $m_2 < 0$, the process of taking a logarithm amplifies the scattering of the near-zero points

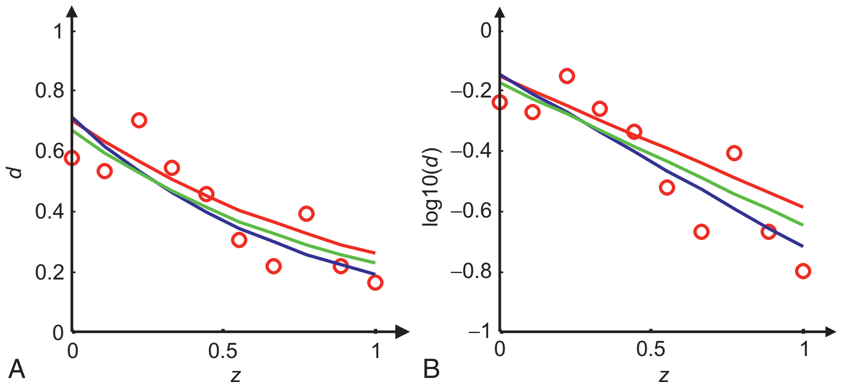

FIGURE 9.2 (A) Least-squares fit to the exponential function, $d_i = m_1 exp(m_2 z_i)$. (Red curve) The true function, $d(z)$, for $(m_1, m_2) = (0.7, 1.0)$. (Red circles) Data that include Normally distributed random noise with zero mean and variance, $\sigma_d^2 = (0.1)^2$. (Green curve) Nonlinear least-squares solution using Newton's method. (Blue curve) Least-squares solution using the linearizing transformation, $\ln(d_i) = \ln(m_1) + m_2 z_i$. Note that the two solutions are different. (B) Log-linear version of the graph in (A). Note that the scatter of the data increases with z, and that the solution based on the linearizing transformation is strongly affected by outliers associated with it. *MatLab* script gda09_03.

that occurs at large z. The assumption that the d_i' have uniform variance, therefore, implies that the data $\mathbf{d}$ were measured with an accuracy that increases with z (Figure 9.2). This assumption may well be inconsistent with the facts of the experiment. Linearizing transformations must be used with some caution.

9.3 ERROR AND LIKELIHOOD IN NONLINEAR INVERSE PROBLEMS

Suppose that the data $\mathbf{d}$ in an inverse problem has a possibly non-Gaussian probability density function $p(\mathbf{d}; \langle\mathbf{d}\rangle)$, where $\langle\mathbf{d}\rangle$ is the mean and the semicolon is used to indicate that $\langle\mathbf{d}\rangle$ is just a parameter in the probability density function for $\mathbf{d}$. The principle of maximum likelihood—that the observed data are the most probable data—holds regardless of the form of $p(\mathbf{d}; \langle\mathbf{d}\rangle)$. As long as the theory is explicit, we can assume that the theory predicts the mean of the data, that is, $\langle\mathbf{d}\rangle = \mathbf{g}(\mathbf{m})$ and write the likelihood $L(\mathbf{m})$ as

$$L(\mathbf{m}) = \log p(\mathbf{d}^{\text{obs}}; \mathbf{m}) = c - \frac{1}{2} E(\mathbf{m}) \tag{9.6}$$

Here c is some constant, $E(\mathbf{m})$ is some function, and the factor of 1/2 has been introduced so that $E(\mathbf{m})$ equals the weighted prediction error in the Gaussian case; that is, $E(\mathbf{m}) = [\mathbf{d}^{\text{obs}} - \mathbf{g}(\mathbf{m})]^T[\text{cov}\ \mathbf{d}]^{-1}[\mathbf{d}^{\text{obs}} - \mathbf{g}(\mathbf{m})]$. Thus, it is always possible to define an "error" $E(\mathbf{m})$ such that minimizing $E(\mathbf{m})$ is equivalent to maximizing the likelihood $L(\mathbf{m})$. However, although $E(\mathbf{m})$ is the total prediction error in the Gaussian case, it does not necessarily have all the properties of the total prediction error in other cases. For instance, it may not be zero when $\mathbf{d}^{\text{obs}} = \mathbf{g}(\mathbf{m})$.

There is no simple means for deciding whether a nonlinear inverse problem has a unique solution. Consider the very simple nonlinear model $d_i = m_1^2 + m_1 m_2 z_i$ with Gaussian data. This problem can be linearized by the transformation of variables $m_1' = m_1^2, m_2' = m_1 m_2$ and can therefore be solved by the least-squares method if $N \geq 2$. Nevertheless, even if the primed parameters are unique, the unprimed ones are not: if $\mathbf{m}^{\text{est}}$ is a solution that minimizes the prediction error, then $-\mathbf{m}^{\text{est}}$ is also a solution with the same error. In this instance, the error $E(\mathbf{m})$ has two minima of equal depth.

We must examine the global properties of the prediction error $E(\mathbf{m})$ in order to determine whether the inverse problem is unique. If the function has but a single minimum point $\mathbf{m}^{\text{est}}$, then the solution is unique. If it has more than one minimum point, the solution is nonunique, and *a priori* information must be added to resolve the indeterminacy. The error surface of a linear problem is always a paraboloid (Figure 9.3), which can have only a simple range of shapes. An arbitrarily complex nonlinear inverse problem can have an arbitrarily complicated error. If $M = 2$ or 3, it may be possible to investigate the shape of the surface by graphical techniques (Figure 9.4). For most realistic problems this is infeasible.

9.4 THE GRID SEARCH

One strategy for solving a nonlinear inverse problem is to exhaustively consider "every possible" solution and pick the one with the smallest error $E(\mathbf{m})$. Of course, it is impossible to examine "every possible" solution; but it is possible to examine a large set of trial solutions. When the trial solutions are drawn from

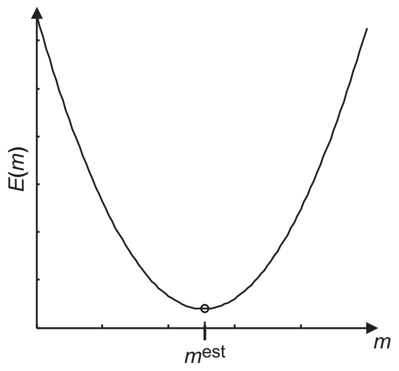

FIGURE 9.3 L_2 prediction error $E(m)$, as a function of model parameter m, for a typical linear inverse problem. The solution m^{est} minimizes the error. In the linear case, $E(m)$ is always a paraboloid. *MatLab* script gda09_04.

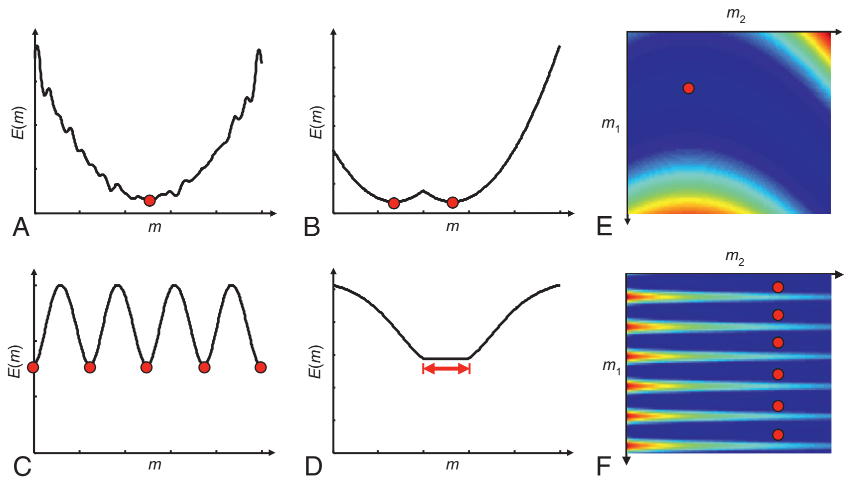

FIGURE 9.4 (A–D) Prediction error, E, as a function of a single model parameter, m. (A) A single minimum (red dot) corresponds to an inverse problem with a unique solution. (B) Two solutions. (C) Many well-separated solutions. (D) Finite range of solutions (red arrow). (E and F) Error (colors) as a function of two model parameters, m_1 and m_2. (E) A single solution, with the minimum occurring within a nearly flat valley. (F) Many well-separated solutions. *MatLab* scripts gda09_05 and gda09_06.

a regular grid in model space, this procedure is called a *grid search*. Grid searches are most practical when

1. The total number of model parameters is small, say $M<7$. The grid is M-dimensional, so the number of trial solutions is proportional to L^M, where L is the number of trial solutions along each dimension of the grid.
2. The solution is known to lie within a specific range of values, which can be used to define the limits of the grid.
3. The forward problem $\mathbf{d}=\mathbf{g}(\mathbf{m})$ can be computed rapidly enough that the time needed to compute L^M of them is not prohibitive.
4. The error function $E(\mathbf{m})$ is smooth over the scale of the grid spacing, Δm, so that the minimum is not missed through the grid spacing being too coarse.

As an example, we use *MatLab* to solve the nonlinear problem $d(x_i)=\sin(\omega_0 m_1 x_i)+m_1 m_2$. The data are assumed to be Gaussian and uncorrelated with uniform variance so that the error is $E(m_1,m_2)=\|\mathbf{d}^{\text{obs}}-\mathbf{d}^{\text{pre}}(\mathbf{m})\|_2$. The script below has three sections: the first section defines the grid of trial $\mathbf{m}$ values; the second evaluates $E(\mathbf{m})$ for every trial $\mathbf{m}$ and stores the results in a matrix `E`; and the third searches the matrix for the smallest value of E and calculates $\mathbf{m}^{\text{est}}$ on the basis of its row and column indices. A useful by-product of the grid search is a tabulation of E on the grid, which can be turned into an informative plot (Figure 9.5).

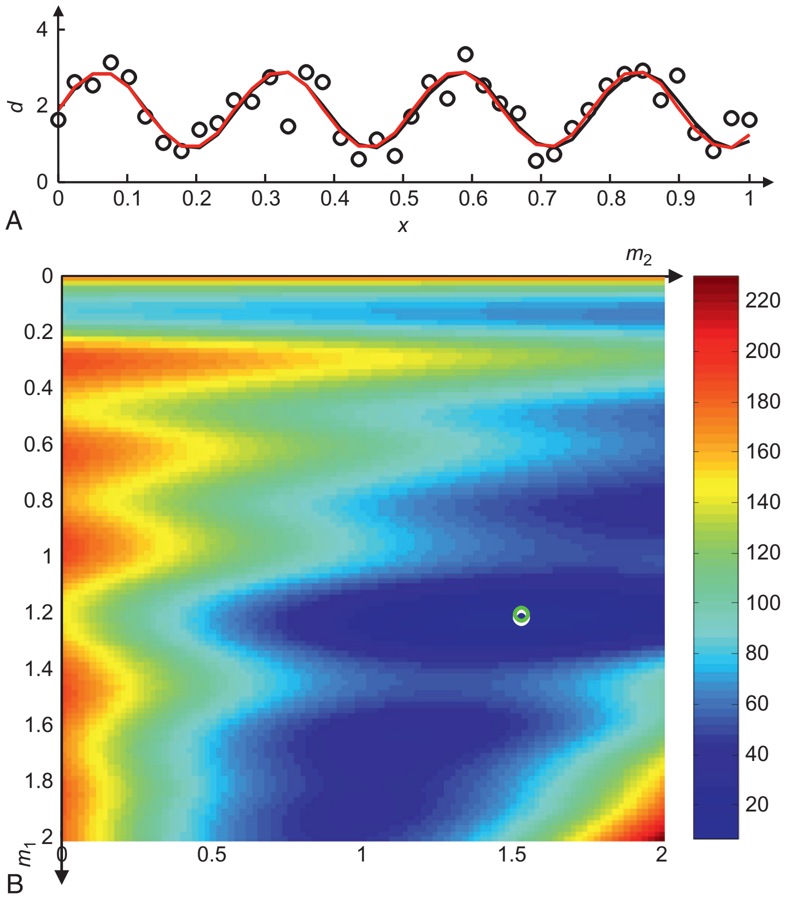

FIGURE 9.5 A grid search is used to solve the nonlinear curve-fitting problem, $d_i(x_i) = \sin(\omega_0 m_1 x_i) + m_1 m_2$. (A) The true data (black curve) are for $m_1 = 1.21$, $m_2 = 1.54$. The observed data (black circles) have additive noise with variance, $\sigma_d^2 = (0.4)^2$. The predicted data (red curve) are based on results of the grid search. (B) Error surface (colors), showing true solution (green circle) and estimated solution (white circle). *MatLab* script gda09_07.

```
% 2D grid of m's
L = 101;
Dm = 0.02;
m1min=0;
m2min=0;
m1a = m1min+Dm*[0:L-1]';
m2a = m2min+Dm*[0:L-1]';
m1max = m1a(L);
m2max = m2a(L);
```

```
% grid search, compute error, E
E = zeros(L,L);
for j = [1:L]
for k = [1:L]
    dpre = sin(w0*m1a(j)*x) + m1a(j)*m2a(k);
    E(j,k) = (dobs-dpre)'*(dobs-dpre);
end
end

% find the minimum value of E
[Erowmins, rowindices] = min(E);
[Emin, colindex] = min(Erowmins);
rowindex = rowindices(colindex);
m1est = m1min+Dm*(rowindex-1);
m2est = m2min+Dm*(colindex-1);
```

(*MatLab* script gda09_07)

9.5 THE MONTE CARLO SEARCH

The Monte Carlo search is a modification of the grid search in which the trial solutions are randomly generated, in contrast to being drawn from a regular grid. In its pure form, where each trial solution is generated independently of previous ones, it has only minor advantages over the grid search. In a later section, we will show that it can be improved by the introduction of correlation between successive trial solutions.

The accompanying *MatLab* script, which implements the algorithm, has two sections: the first section generates an initial trial solution and its corresponding error; the second is a loop that randomly generates a trial solution, calculates its corresponding error, and accepts it if the error is less than the previously accepted solution.

```
% initial guess and corresponding error
mg=[1,1]';
dg = sin(w0*mg(1)*x) + mg(1)*mg(2);
Eg = (dobs-dg)'*(dobs-dg);

% randomly generate pairs of model parameters and check
% if they further minimize the error
ma = zeros(2,1);
for k = [1:Niter]

    % randomly generate a solution
    ma(1) = random('unif',m1min,m1max);
    ma(2) = random('unif',m2min,m2max);
```

```
        % compute its error
        da = sin(w0*ma(1)*x) + ma(1)*ma(2);
        Ea = (dobs-da)'*(dobs-da);

        % adopt it if it is better
        if( Ea < Eg )
            mg=ma;
            Eg=Ea;
        end
    end
```

(*MatLab* script gda09_08)

An example is shown in Figure 9.6.

9.6 NEWTON'S METHOD

A completely different approach to solving the nonlinear inverse problem is to use information about the shape of the error $E(\mathbf{m})$ in the vicinity of a trial solution $\mathbf{m}^{(p)}$ to devise a better solution $\mathbf{m}^{(p+1)}$. One source of shape information is

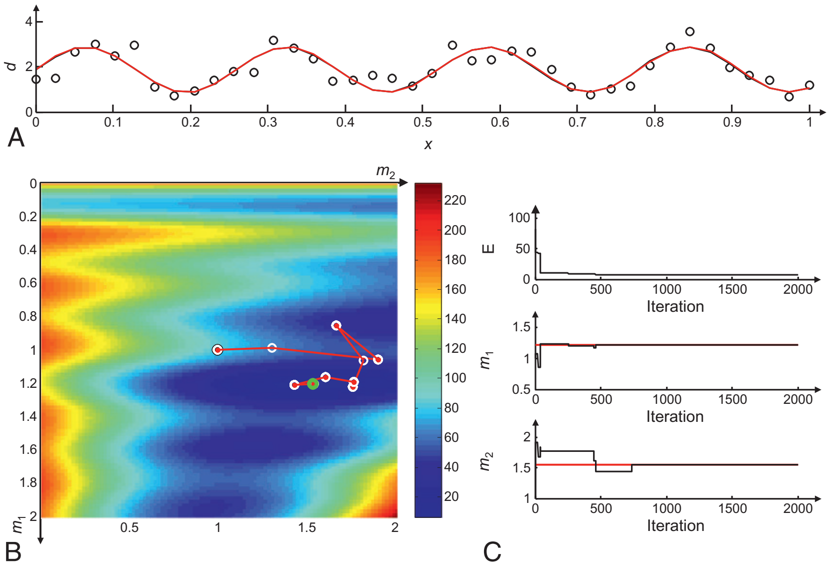

FIGURE 9.6 A Monte Carlo search is used to solve the same nonlinear curve-fitting problem as in Figure 9.5. (A) The observed data (black circles) are computed from the true data (black curve) by adding random noise. The predicted data (red curve) are based on the results of the method. (B) Error surface (colors), showing true solution (green circle), and a series of improved solutions (white circles connected by red lines) determined by the method. (C) Plot of error E and model parameters m_1 and m_2 as a function of iteration number. *MatLab* script gda09_08.

the derivatives of $E(\mathbf{m})$ at a trial solution $\mathbf{m}^{(p)}$, as Taylor's theorem indicates that the entire function can be built up from them. Expanding $E(\mathbf{m})$ in a Taylor series about the trial solution and keeping the first three terms, we obtain the parabolic approximation

$$E(\mathbf{m}) \approx E(\mathbf{m}^{(p)}) + \sum_{i=1}^{M} b_i\left(m_i - m_i^{(p)}\right) + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} B_{ij}\left(m_i - m_i^{(p)}\right)\left(m_j - m_j^{(p)}\right)$$

$$\text{with} \quad b_i = \left.\frac{\partial E}{\partial m_i}\right|_{\mathbf{m}^{(p)}} \quad \text{and} \quad B_{ij} = \left.\frac{\partial^2 E}{\partial m_i \partial m_j}\right|_{\mathbf{m}^{(p)}} \tag{9.7}$$

Here $\mathbf{b}$ is a vector of first derivatives and $\mathbf{B}$ is a matrix of second derivatives of $E(\mathbf{m})$, evaluated at the trial solution $\mathbf{m}^{(p)}$. These derivatives can be calculated either by analytically differentiating $E(\mathbf{m})$, if its functional form is known, or by approximating it with finite differences

$$\left.\frac{\partial E}{\partial m_i}\right|_{\mathbf{m}^{(p)}} \approx \frac{1}{\Delta m}\left\{E(\mathbf{m}+\Delta\mathbf{m}^{(\mathbf{i})}) - E(\mathbf{m})\right\}$$

$$\left.\frac{\partial^2 E}{\partial m_i \partial m_j}\right|_{\mathbf{m}^{(p)}} \approx \begin{cases} \dfrac{1}{(\Delta m)^2}\{E(\mathbf{m}+\Delta\mathbf{m}^{(\mathbf{i})}) - 2E(\mathbf{m}) + E(\mathbf{m}-\Delta\mathbf{m}^{(\mathbf{i})})\} & (i=j) \\ \dfrac{1}{4(\Delta m)^2}\{E(\mathbf{m}+\Delta\mathbf{m}^{(\mathbf{i})}+\Delta\mathbf{m}^{(\mathbf{j})}) - E(\mathbf{m}+\Delta\mathbf{m}^{(\mathbf{i})}-\Delta\mathbf{m}^{(\mathbf{j})}) & \\ -E(\mathbf{m}-\Delta\mathbf{m}^{(\mathbf{i})}+\Delta\mathbf{m}^{(\mathbf{j})}) + E(\mathbf{m}-\Delta\mathbf{m}^{(\mathbf{i})}-\Delta\mathbf{m}^{(\mathbf{j})})\} & (i \neq j) \end{cases} \tag{9.8}$$

Here $\Delta\mathbf{m}^{(\mathbf{i})}$ is a small increment Δm in the ith direction; that is, $\Delta\mathbf{m}^{(\mathbf{i})} = \Delta m\,[0, \ldots, 0, 1, 0, \ldots, 0]^{\mathrm{T}}$, where the ith element is unity and the rest are zero. Note that these approximations are computationally expensive, in the sense that E must be evaluated many times for each instance of $\mathbf{b}$ and $\mathbf{G}$.

We can now find the minimum by differentiating this approximate form of $E(\mathbf{m})$ with respect to m_q and setting the result to zero:

$$\frac{\partial E(\mathbf{m})}{\partial m_q} = 0 = b_q + \sum_{j=1}^{M} B_{qj}\left(m_j - m_j^{(p)}\right) \quad \text{or} \quad \mathbf{m} - \mathbf{m}^{(p)} = -\mathbf{B}^{-1}\mathbf{b} \tag{9.9}$$

In the case of uncorrelated Gaussian data with uniform variance and the *linear* theory $\mathbf{d}=\mathbf{Gm}$, the error is $E(\mathbf{m}) = [\mathbf{d}-\mathbf{Gm}]^{\mathrm{T}}[\mathbf{d}-\mathbf{Gm}]$, from whence we find that $\mathbf{b} = -2\mathbf{G}^{\mathrm{T}}(\mathbf{d}-\mathbf{Gm}^{(p)})$, $\mathbf{B} = 2\mathbf{G}^{\mathrm{T}}\mathbf{G}$ and $\mathbf{m} = [\mathbf{G}^{\mathrm{T}}\mathbf{G}]^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{d}$, which is the familiar least-squares solution. In this case, the result is independent of the trial solution and is exact.

In the case of uncorrelated Gaussian data with uniform variance and the nonlinear theory $\mathbf{d}-\mathbf{g}(\mathbf{m})=0$, the error is $E(\mathbf{m}) = [\mathbf{d}-\mathbf{g}(\mathbf{m})]^{\mathrm{T}}[\mathbf{d}-\mathbf{g}(\mathbf{m})]$, from which we conclude

$$\mathbf{b} = -2\mathbf{G}^{(p)\mathrm{T}}[\mathbf{d} - \mathbf{g}(\mathbf{m}^{(p)})] \quad \text{and} \quad \mathbf{B} \approx 2[\mathbf{G}^{(p)\mathrm{T}}\mathbf{G}^{(p)}] \quad \text{with} \quad G_{ij}^{(p)} = \left.\frac{\partial g_i}{\partial m_j}\right|_{\mathbf{m}^{(p)}}$$

$$\text{so} \quad \mathbf{m} - \mathbf{m}^{(p)} \approx [\mathbf{G}^{(p)\mathrm{T}}\mathbf{G}^{(p)}]^{-1}\mathbf{G}^{(p)\mathrm{T}}[\mathbf{d} - \mathbf{g}(\mathbf{m}^{(p)})] \tag{9.10}$$

Note that we have omitted the term involving the gradient of **G** in the formula for **B**. In a linear theory, **G** is constant and its gradient is zero. We assume that the theory is sufficiently close to linear that here too it is insignificant.

The form of Equation (9.10) is very similar to simple least squares. The matrix $\mathbf{G}^{(\mathbf{p})}$, which is the gradient of the model $\mathbf{g}(\mathbf{m})$ at the trial solution, acts as a data kernel. It can be calculated analytically, if its functional form is known, or approximated by finite differences (see Equation (9.8)). The generalized inverse $\mathbf{G}^{-g} = [\mathbf{G}^{(p)\mathrm{T}}\mathbf{G}^{(p)}]^{-1}\mathbf{G}^{(p)\mathrm{T}}$ relates the deviation of the data $\Delta\mathbf{d} = [\mathbf{d} - \mathbf{g}(\mathbf{m}^{(p)})]$ from what is predicted by the trial solution to the deviation of the solution $\Delta\mathbf{m} = \mathbf{m} - \mathbf{m}^{(p)}$ from the trial solution, that is, $\Delta\mathbf{m} = \mathbf{G}^{-g}\Delta\mathbf{d}$. However, because it is based on a truncated Taylor Series, the solution is only approximate; it yields a solution that is improved over the trial solution, but not the exact solution. However, it can be iterated to yield a succession of improvements

$$\mathbf{m}^{(p+1)} = \mathbf{m}^{(p)} + [\mathbf{G}^{(p)\mathrm{T}}\mathbf{G}^{(p)}]^{-1}\mathbf{G}^{(p)\mathrm{T}}[\mathbf{d} - \mathbf{g}(\mathbf{m}^{(p)})] \tag{9.11}$$

until the error declines to an acceptably low level (Figure 9.7). Unfortunately, while convergence of $\mathbf{m}^{(p)}$ to the value that globally minimizes $E(\mathbf{m})$ is often

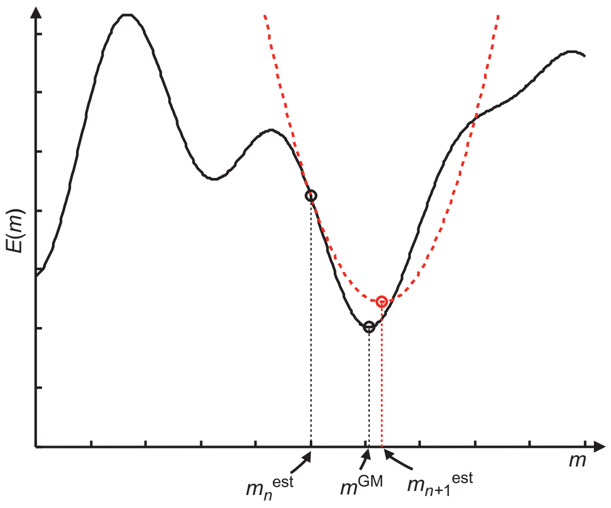

FIGURE 9.7 The iterative method locates the global minimum m^{GM} of the error $E(m)$ (black curve) by determining the paraboloid (red curve) that is tangent to E at the trial solution m_n^{est}. The improved solution m_{n+1}^{est} is at the minimum (red circle) of this paraboloid and, under favorable conditions, can be closer to the solution corresponding to the global minimum than is the trial solution. *MatLab* script gda09_09.

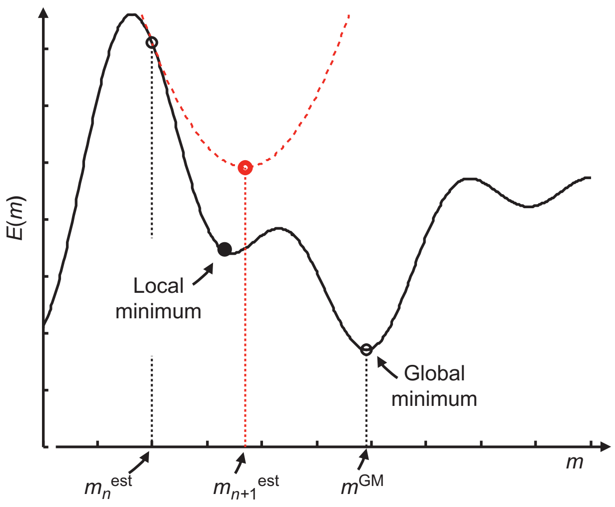

FIGURE 9.8 If the trial solution, m_n^{est}, is too far from the global minimum m^{GM}, the method may converge to a local minimum. *MatLab* script gda09_10.

very rapid, it is not guaranteed. One common problem is the solution converging to a local minimum instead of the global minimum (Figure 9.8). A critical part of the algorithm is picking the initial trial solution $\mathbf{m}^{(1)}$. The method may not find the global minimum if the trial solution is too far from it.

As an example, we consider the problem from Section 9.4, in which $g_i(\mathbf{m}) = \sin(\omega_0 m_1 x_i) + m_1 m_2$. The matrix of partial derivatives is

$$\mathbf{G}^{(p)} = \begin{bmatrix} \omega_0 x_1 \cos\left(\omega_0 x_1 m_1^{(p)}\right) + m_2^{(p)} & m_1^{(p)} \\ \omega_0 x_2 \cos\left(\omega_0 x_2 m_1^{(p)}\right) + m_2^{(p)} & m_1^{(p)} \\ \vdots & \vdots \\ \omega_0 x_N \cos\left(\omega_0 x_N m_1^{(p)}\right) + m_2^{(p)} & m_1^{(p)} \end{bmatrix} \tag{9.12}$$

In *MatLab*, the main part of the script is a loop that iteratively updates the trial solution. First, the deviation $\Delta\mathbf{d}$ (`dd` in the script) and the data kernel $\mathbf{G}$ are calculated, using the trial solution $\mathbf{m}^{(p)}$. Then the deviation $\Delta\mathbf{m}$ is calculated using least squares. Finally, the trial solution is updated as $\mathbf{m}^{(p+1)} = \mathbf{m}^{(p)} + \Delta\mathbf{m}$. In *MatLab*

```
% initial guess and corresponding error
mg=[1,1]';
dg = sin(w0*mg(1)*x) + mg(1)*mg(2);
Eg = (dobs-dg)'*(dobs-dg);

% iterate to improve initial guess
Niter = 20;
```

```
G = zeros(N,M);
for k = [1:Niter]

    dg = sin(w0*mg(1)*x) + mg(1)*mg(2);
    dd = dobs-dg;
    Eg=dd'*dd;

    G = zeros(N,2);
    G(:,1) = w0 * x .* cos( w0 * mg(1) * x ) + mg(2);
    G(:,2) = mg(2)*ones(N,1);

    % least squares solution
    dm = (G'*G)\(G'*dd);

    % update
    mg = mg+dm;
end
```

(*MatLab* script gda09_11)

The results of this script are shown in Figure 9.9. This exemplary script iterates a fixed number of times, as specified by the variable `Niter`. A more elegant approach would be to perform a test that terminates the iterations when the error declines to an acceptable level or when the solution no longer changes significantly from iteration to iteration.

9.7 THE IMPLICIT NONLINEAR INVERSE PROBLEM WITH GAUSSIAN DATA

We now generalize the iterative method of the previous section to the general case of the implicit theory $\mathbf{f}(\mathbf{d}, \mathbf{m}) = 0$, where $\mathbf{f}$ is of length $L \leq M + N$. We assume that the data $\mathbf{d}$ and *a priori* model parameters $\langle \mathbf{m} \rangle$ have Gaussian distributions with covariance [cov $\mathbf{d}$] and [cov $\mathbf{m}]_A$, respectively. If we let $\mathbf{x} = [\mathbf{d}^T, \mathbf{m}^T]^T$, we can think of the *a priori* distribution of the data and model as a cloud in the space $S(\mathbf{x})$ centered about the observed data and mean *a priori* model parameters, with a shape determined by the covariance matrix [cov $\mathbf{x}$] (Figure 9.10). The matrix [cov $\mathbf{x}$] contains [cov $\mathbf{d}$] and [cov $\mathbf{m}$] on diagonal blocks, as in Equation (5.27). In principle, the off-diagonal blocks could be made nonzero, indicating correlation between observed data and *a priori* model parameters. However, specifying *a priori* constraints is typically an *ad hoc* procedure that one can seldom find motivation for introducing such a correlation. The *a priori* distribution is therefore

$$p_A(\mathbf{x}) \propto \exp\left\{ -\frac{1}{2}[\mathbf{x} - \langle \mathbf{x} \rangle]^T [\text{cov}\ \mathbf{x}]^{-1} [\mathbf{x} - \langle \mathbf{x} \rangle] \right\} \tag{9.13}$$

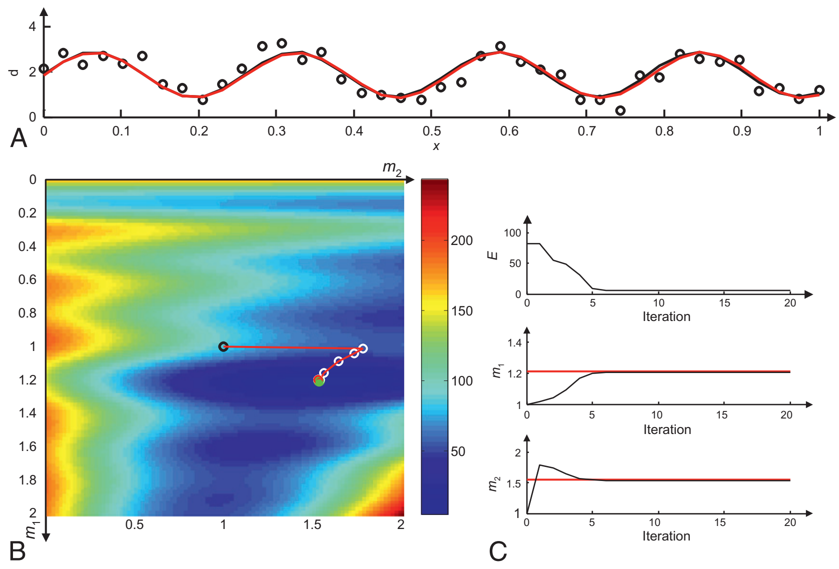

FIGURE 9.9 Newton's method (linearized least squares) is used to solve the same nonlinear curve-fitting problem as in Figure 9.5. (A) The observed data (black circles) are computed from the true data (black curve) by adding random noise. The predicted data (red curve) are based on the results of the method. (B) Error surface (colors), showing true solution (green dot), and a series of improved solutions (white circles connected by red lines) determined by the method. (C) Plot of error E and model parameters, m_1 and m_2 as a function of iteration number. *MatLab* script gda09_11.

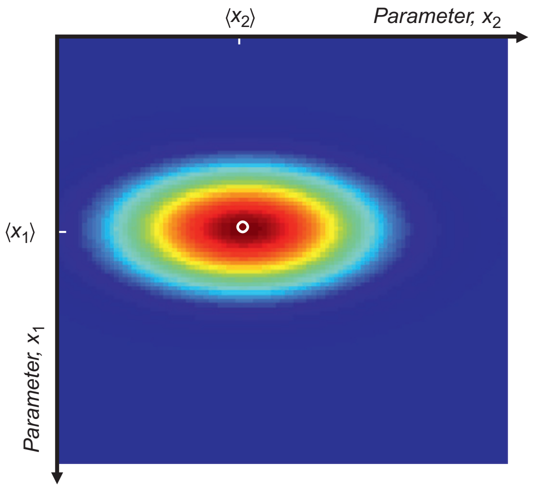

FIGURE 9.10 The data and model parameters are grouped together in a vector, **x**. The prior information for **x** is then represented as a probability density function (colors) in the $(M+N)$-dimensional space, $S(\mathbf{x})$. *MatLab* script gda09_12.

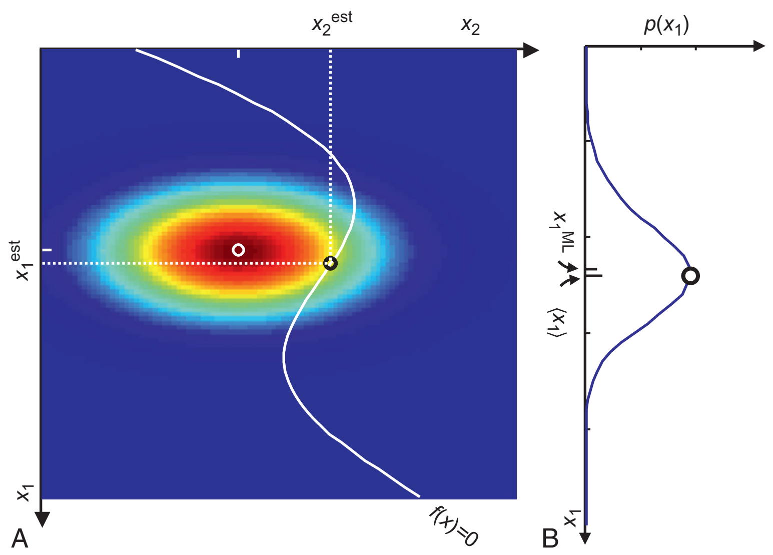

FIGURE 9.11 (A) The estimated solution $\mathbf{x}^{est}$ (black circle) is at the point on the surface $\mathbf{f}(\mathbf{x}) = 0$ (white curve) where the *a priori* probability density function (colors) attains its largest value. (B) The probability density function $p(x_1)$ evaluated along the surface, as a function of position x_1. As the function is nonnormal, its mean $\langle x_1 \rangle$ may be distinct from its mode x_1^{ML} (although in this case they are similar). *MatLab* script gda09_13.

where $\langle \mathbf{x} \rangle = [(\mathbf{d}^{obs})^T, \langle \mathbf{m} \rangle^T]^T$ is a vector containing the observed data and *a priori* model parameters.

The theory $\mathbf{f}(\mathbf{x}) = 0$ defines a surface in $S(\mathbf{x})$ on which the predicted data and estimated model parameters $\mathbf{x}^{est} = [\mathbf{d}^{pre\ T}, \mathbf{m}^{est\ T}]^T$ must lie. The probability distribution for $\mathbf{x}^{est}$ is, therefore, $p_A(\mathbf{x})$, evaluated on this surface (Figure 9.11). If the surface is plane, this is just the linear case described in Chapter 5 and the final distribution is Gaussian. On the other hand, if the surface is very "bumpy," the distribution on the surface will be very non-Gaussian and may even possess several maxima (Figure 9.12).

One approach to estimating the solution is to find the maximum likelihood point of $p_A(\mathbf{x})$ on the surface $\mathbf{f}(\mathbf{x}) = 0$ (Figure 9.11). This point can be found without explicitly determining the distribution on the surface. One just maximizes $p_A(\mathbf{x})$ with the constraint that $\mathbf{f}(\mathbf{x}) = 0$. One should keep in mind, however, that the maximum likelihood point of a non-Gaussian distribution may not be the most sensible estimate that can be made from that distribution. Gaussian distributions are symmetric, so their maximum likelihood point always coincides with their mean value. In contrast, the maximum likelihood point can be arbitrarily far from the mean of a non-Gaussian distribution (Figure 9.12). Computing the mean, however, requires one to compute explicitly the distribution on the surface and then take its expectation (a much more difficult procedure).

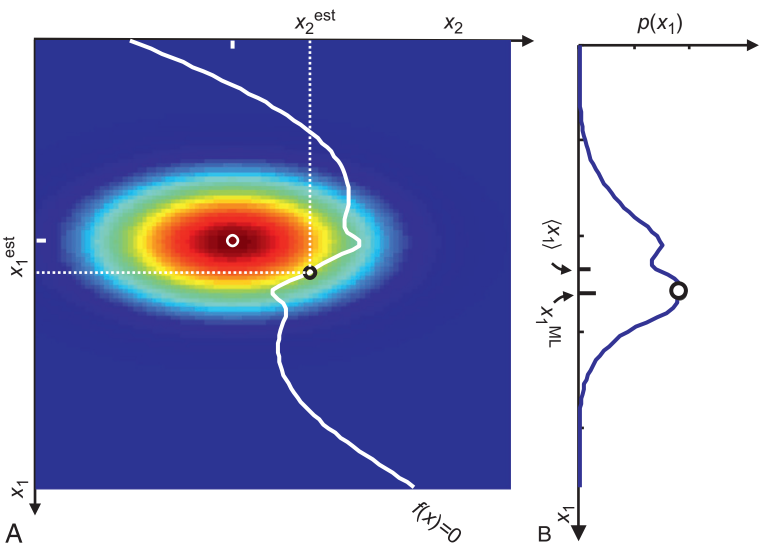

FIGURE 9.12 (A) A highly nonlinear inverse problem corresponds to a complicated surface $\mathbf{f}(\mathbf{x})=0$ (white curve). (B) The probability density function $p(x_1)$ evaluated along the surface, as a function of position x_1. It may have several peaks, and as it is nonnormal, its mean $\langle x_1 \rangle$ may be distinct from its mode x_1^{ML}. *MatLab* script gda09_14.

These caveats aside, we proceed with the calculation of the maximum likelihood point by minimizing the argument of the exponential in $p_A(\mathbf{x})$ with the constraint that $\mathbf{f}(\mathbf{x})=0$ (adapted from Tarantola and Valette, 1982):

$$\text{minimize} \quad \Phi = [\mathbf{x} - \langle \mathbf{x} \rangle]^T [\text{cov}\ \mathbf{x}]^{-1} [\mathbf{x} - \langle \mathbf{x} \rangle] \quad \text{subject to } \mathbf{f}(\mathbf{x}) = 0 \tag{9.14}$$

The Lagrange multiplier equations are

$$\frac{\partial \Phi}{\partial x_i} - \sum_{j=1}^{L} 2\lambda_j \frac{\partial f_j}{\partial x_i} = 0 \quad \text{or} \quad [\mathbf{x} - \langle \mathbf{x} \rangle]^T [\text{cov}\ \mathbf{x}]^{-1} = \mathbf{F}^T \boldsymbol{\lambda} \tag{9.15}$$

where $\boldsymbol{\lambda}$ is a vector of Lagrange multipliers and $\mathbf{F}$ is the matrix of derivatives $F_{ij} = \frac{\partial f_i}{\partial x_j}$. The Lagrange multipliers can be determined by premultiplying the transformed equation by $\mathbf{F}$ as

$$\mathbf{F}[\mathbf{x} - \langle \mathbf{x} \rangle] = \mathbf{F}[\text{cov}\ \mathbf{x}]\mathbf{F}^T \boldsymbol{\lambda} \tag{9.16}$$

and then premultiplying $\left\{\mathbf{F}[\text{cov}\ \mathbf{x}]\mathbf{F}^T\right\}^{-1}$ as

$$\boldsymbol{\lambda} = \left\{\mathbf{F}[\text{cov}\ \mathbf{x}]\mathbf{F}^T\right\}^{-1} \mathbf{F}[\mathbf{x} - \langle \mathbf{x} \rangle] \tag{9.17}$$

Substitution into the transpose of the original equation yields

$$[\mathbf{x} - \langle \mathbf{x} \rangle] = [\text{cov}\ \mathbf{x}]\mathbf{F}^T \left\{\mathbf{F}[\text{cov}\ \mathbf{x}]\mathbf{F}^T\right\}^{-1} \mathbf{F}[\mathbf{x} - \langle \mathbf{x} \rangle] \tag{9.18}$$

which must be solved simultaneously with the constraint equation $\mathbf{f}(\mathbf{x}) = 0$. These two equations are equivalent to the single equation

$$[\mathbf{x} - \langle\mathbf{x}\rangle] = [\text{cov } \mathbf{x}]\mathbf{F}^{\mathrm{T}}\left\{\mathbf{F}[\text{cov } \mathbf{x}]\mathbf{F}^{\mathrm{T}}\right\}^{-1}\{\mathbf{F}[\mathbf{x} - \langle\mathbf{x}\rangle] - \mathbf{f}(\mathbf{x})\} \tag{9.19}$$

as the original two equations can be recovered by premultiplying this equation by $\mathbf{F}$. The form of this equation is very similar to the linear solution of Equation (5.37); in fact, it reduces to it in the case of the exact linear theory $\mathbf{f}(\mathbf{x}) = \mathbf{F}\mathbf{x}$. As the unknown $\mathbf{x}$ appears on both sides of the equation and $\mathbf{f}$ and $\mathbf{F}$ are functions of $\mathbf{x}$, this equation may be difficult to solve explicitly. We now examine an iterative method of solving it. This method consists of starting with some initial trial solution, say, $\mathbf{x}^{(p)}$, where $p = 1$, and then generating successive approximations as

$$\mathbf{x}^{(p+1)} = \langle\mathbf{x}\rangle + [\text{cov } \mathbf{x}]\mathbf{F}^{(p)\mathrm{T}}\left\{\mathbf{F}^{(p)}[\text{cov } \mathbf{x}]\mathbf{F}^{(p)\mathrm{T}}\right\}^{-1}\{\mathbf{F}^{(p)}[\mathbf{x}^{(p)} - \langle\mathbf{x}\rangle] - \mathbf{f}(\mathbf{x}^{(p)})\} \tag{9.20}$$

The superscript on $\mathbf{F}^{(p)}$ implies that it is evaluated at $\mathbf{x}^{(p)}$. If the initial guess is close enough to the maximum likelihood point, the successive approximations will converge to the true solution $\mathbf{x}^{\text{est}}$; else it may converge to a local minimum.

If the theory is explicit (i.e., if $\mathbf{f}(\mathbf{x}) = \mathbf{d} - \mathbf{g}(\mathbf{m}) = 0$) and if the data and *a priori* model parameters are uncorrelated, the iterative formula can be rewritten as

$$\mathbf{m}^{(p+1)} = \langle\mathbf{m}\rangle + \mathbf{G}_{(p)}^{-g}\{\mathbf{d} - \mathbf{g}(\mathbf{m}^{(p)}) + \mathbf{G}^{(p)}[\mathbf{m}^{(p)} - \langle\mathbf{m}\rangle]\} \tag{9.21a}$$

$$\begin{aligned}\mathbf{G}_{(p)}^{-g} &= [\text{cov } \mathbf{m}]_{\mathrm{A}}\mathbf{G}^{(p)\mathrm{T}}\left\{\mathbf{G}^{(p)}[\text{cov } \mathbf{m}]_{\mathrm{A}}\mathbf{G}^{(p)\mathrm{T}} + [\text{cov } \mathbf{d}]\right\}^{-1} \\ &= \left\{\mathbf{G}^{(p)\mathrm{T}}[\text{cov } \mathbf{d}]^{-1}\mathbf{G}^{(p)} + [\text{cov } \mathbf{m}]_{\mathrm{A}}^{-1}\right\}^{-1}\mathbf{G}^{(p)\mathrm{T}}[\text{cov } \mathbf{d}]^{-1}\end{aligned} \tag{9.21b}$$

or solved using simple least squares

$$\begin{bmatrix} [\text{cov } \mathbf{d}]^{-1/2}\mathbf{G}^{(p)} \\ [\text{cov } \mathbf{m}]_{\mathrm{A}}^{-1/2}\mathbf{I} \end{bmatrix}\mathbf{m}^{(p+1)} = \begin{bmatrix} [\text{cov } \mathbf{d}]^{-1/2}\left\{\mathbf{d} - \mathbf{g}\left(\mathbf{m}^{(p)}\right) + \mathbf{G}^{(p)}\mathbf{m}^{(p)}\right\} \\ [\text{cov } \mathbf{m}]_{\mathrm{A}}^{-1/2}\langle\mathbf{m}\rangle \end{bmatrix} \tag{9.21c}$$

Here $[\mathbf{G}^{(p)}]_{ij} = \partial g_i / \partial m_j$ is evaluated at $\mathbf{m}^{(p)}$ and the generalized inverse notation has been used for convenience. The two versions of the generalized inverse in Equation (9.21b)—one with the form of the minimum length solution and the other with the form of the least-squares solution—are equivalent, as was shown in Equation (5.39). Equation (9.21a–9.21c) is the nonlinear, iterative analog to the linear, noniterative formulas stated for the linear inverse problem in Equation (5.37), except that in this case the theory has been assumed to be exact. One obtains formulas for an inexact theory with covariance [cov $\mathbf{g}$] with the substitution [cov $\mathbf{d}$] $\rightarrow$ [cov $\mathbf{d}$] + [cov $\mathbf{g}$] and for *a priori* information of the form $\mathbf{Hm}=\mathbf{h}$ with the substitutions $\mathbf{I}\rightarrow\mathbf{H}$, $\langle\mathbf{m}\rangle\rightarrow\mathbf{h}$, and $[\text{cov}\ \mathbf{m}]_A \rightarrow [\text{cov}\ \mathbf{h}]_A$.

Provided the problem is overdetermined, Equation (9.21a–9.21c) reduces to Newton's method in the limit where $\langle\mathbf{m}\rangle \rightarrow 0$, $[\text{cov}\ \mathbf{m}]_A^{-1} \rightarrow \mathbf{0}$, and $[\text{cov}\ \mathbf{d}]^{-1} \rightarrow \mathbf{I}$

$$\Delta\mathbf{m} = \mathbf{G}^{-g}\Delta\mathbf{d} + (\mathbf{I} - \mathbf{R}^{(p)})\mathbf{m}^{(p)} \rightarrow \mathbf{G}^{-g}\Delta\mathbf{d} \tag{9.22}$$

as the resolution matrix $\mathbf{R}^{(p)} = \mathbf{G}_{(p)}^{-g}\mathbf{G}^{(p)} \rightarrow \mathbf{I}$ in the overdetermined case. However, Equation (9.21a–9.21c) indicates that Newton's method cannot be *patched* for underdetermined problems merely by using a damped least-squares version of $\mathbf{G}_{(p)}^{-g}$. The problem is that damped least squares drives $\Delta\mathbf{m}$ toward zero, rather than (as is more sensible) driving $\mathbf{m}^{(p+1)}$ toward zero. Instead, *MatLab* scripts should solve Equation (9.21c), using the `bicg()` solver together with the `weightedleastsquaresfcn()` function.

In a linear problem, the error $E(\mathbf{m})$ is a paraboloid and the estimated model parameters $\mathbf{m}^{\text{est}}$ are at its minimum. While a nonlinear problem has an error with a more complicated shape, it may still be approximately paraboloid in the vicinity of its minimum. This is the region of the space of model parameters where the inverse problem behaves linearly and the probability density function $p(\mathbf{m})$ is approximately Gaussian in shape. If the patch is big enough to encompass a large percentage of the total probability, then one might use the linear formula

$$[\text{cov}\ \mathbf{m}^{\text{est}}] = \mathbf{G}_{(p)}^{-g}[\text{cov}\ \mathbf{d}]\mathbf{G}_{(p)}^{-g\text{T}} + [\mathbf{I} - \mathbf{R}^{(p)}][\text{cov}\ \mathbf{m}]_A[\mathbf{I} - \mathbf{R}^{(p)}]^{\text{T}} \tag{9.23}$$

where p is the final iteration, to calculate approximate variances of the model parameters. Whether 95% confidence intervals inferred from these variances are correct will depend upon the size of the patch because only the central part of the probability density function, and not its tails, is approximately Gaussian. For this reason, estimates of confidence intervals based on the Bootstrap method (Section 9.11) are usually preferred.

The same caveat applies to interpretations of the resolution matrices $\mathbf{N}^{(p)}$ and $\mathbf{R}^{(p)}$. As the problem is nonlinear, they do not describe the true resolution of the problem. On the other hand, they give the resolution of a linear problem that is in some sense close to the nonlinear one.

9.8 GRADIENT METHOD

Occasionally, one encounters an inverse problem in which the error $E(\mathbf{m})$ and its gradient $[\nabla E]_i = \mathrm{d}E/\mathrm{d}m_i$ are especially easy to calculate. It is possible to solve the inverse problem using this information alone, as the unit vector

$$v = -\frac{\nabla E}{|\nabla E|} \tag{9.24}$$

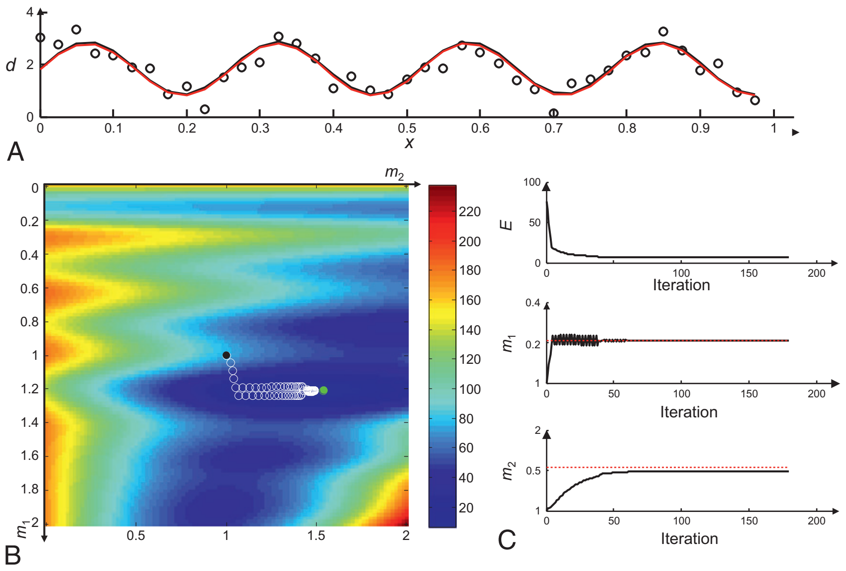

FIGURE 9.13 Gradient method is used to solve the same nonlinear curve-fitting problem as in Figure 9.5. (A) The observed data (black circles) are computed from the true data (black curve) by adding random noise. The predicted data (red curve) are based on the results of the method. (B) Error surface (colors), showing true solution (green dot), and a series of improved solutions (white circles) determined by the method. (C) Plot of error E and model parameters m_1 and m_2 as a function of iteration number. *MatLab* script gda09_15.

points in the direction in which the error E is minimized. Thus a trial solution $\mathbf{m}^{(j)}$ can be improved to $\mathbf{m}^{(j+1)} = \mathbf{m}^{(j)} + \alpha \boldsymbol{\nu}$, where α is a positive number. The only problem is that one does not immediately know how large α should be. Too large and the minimum may be skipped over; too small and the convergence will be very slow. *Armijo's rule* provides an acceptance criterion for α:

$$E\left(\mathbf{m}^{(k+1)}\right) \le E\left(\mathbf{m}^{(k)}\right) + c\alpha\boldsymbol{\nu}^{\mathrm{T}}\nabla E\big|_{m^{(k)}} \tag{9.25}$$

Here c is an empirical constant in the range (0, 1) that is usually chosen to be about 10^{-4}. One strategy is to start the iteration with a "largish" value of α and to use it as long as it passes the test, but to decrease it whenever it fails, say using the rule $\alpha \rightarrow \alpha/2$. An example is shown in Figure 9.13.

9.9 SIMULATED ANNEALING

The Monte Carlo method (Section 9.5) is completely *undirected*. The whole model space is sampled randomly so that the global minimum of the error *eventually* is found, provided that enough trial solutions are examined. Unfortunately, the number of trial solutions that need be computed may be very large—perhaps millions.

In contrast, Newton's method (Section 9.6) is completely directed. Local properties of the error—its slope and curvature—are used to determine the optimal improvement to a trial solution. Convergence to the global minimum of the error, when it occurs, is rapid and just a few trial solutions need be computed—perhaps just ten. Unfortunately, the method may only find a local minimum of the error that corresponds to an incorrect estimate of the model parameters.

The *simulated annealing* method combines the best features of these two methods. Like the Monte Carlo method, it samples the whole model space and so can avoid getting stuck in local minima. And like Newton's method, it uses local information to direct the sequence of trial solutions. Its development was inspired by the physical annealing of metals (Kirkpatrick et al., 1983), where an orderly minimum-energy crystal structure develops within the metal as it is slowly cooled from a red hot state. Initially, when the metal is hot, atomic motions are completely dominated by random thermal fluctuations, but as the temperature is slowly lowered, interatomic forces become more and more important. In the end, the atoms become a crystal lattice that represents a minimum-energy configuration.

In the simulated annealing algorithm, a parameter T is the analog to temperature and the error E is the analog to energy. Large values of T cause the algorithm to behave like a Monte Carlo search; small values cause it to act in a more directed way. As in physical annealing, one starts with a large T and then slowly decreases it as more and more trial solutions are examined. Initially, a very large volume of model space is randomly sampled, but the search becomes increasingly directed as it progresses.

The simulated annealing algorithm starts with a trial solution $\mathbf{m}^{(p)}$ with corresponding error $E(\mathbf{m}^{(p)})$. A test solution $\mathbf{m}^*$ with corresponding error $E(\mathbf{m}^*)$ is then generated that is in the neighborhood of $\mathbf{m}^{(p)}$, say by adding to $\mathbf{m}^{(p)}$ an increment $\Delta\mathbf{m}$ drawn from a Gaussian distribution. The test solution is always accepted as the new trial solution $\mathbf{m}^{(p+1)}$ when $E(\mathbf{m}^*) \leq \mathrm{E}(\mathbf{m}^{(p)})$, but it is also sometimes accepted even when $E(\mathbf{m}^*) > E(\mathbf{m}^{(p)})$. To decide the later case, a test parameter

$$t = \frac{\exp\{-E(\mathbf{m}^*)/T\}}{\exp\{-E(\mathbf{m}^{(p)})/T\}} = \exp\left\{-\frac{[E(\mathbf{m}^*) - E(\mathbf{m}^{(p)})]}{T}\right\} \tag{9.26}$$

is computed. Then, a random number r that is uniformly distributed on the interval [0, 1] is generated and the solution $\mathbf{m}^*$ is accepted if $t > r$. When T is large, the parameter t is close to unity and $\mathbf{m}^*$ is almost always accepted, regardless of the value of the error. This corresponds to the "thermal motion" case where the space of model parameters is explored in an undirected way. When T is small, the parameter t is close to zero and $\mathbf{m}^*$ is almost never accepted. This corresponds to the directed search case, as then the only solutions that decrease the error are accepted. An astute reader will recognize that this is just the Metropolis-Hastings algorithm (Section 2.8) applied to the *Boltzmann* probability density function

$$p(\mathbf{m}) \propto \exp\left\{\frac{-E(\mathbf{m})}{T}\right\} \tag{9.27}$$

The maximum likelihood point of this probability density function corresponds to the point of minimum error, regardless of the value of the parameter T. However, T controls the width of the probability density function, with a larger T corresponding to a wider function. At first, T is high and the distribution is wide. The sequence of realizations samples a broad region of model space that includes the global minimum and, possibly, local minima as well. As T is decreased, the probability density function becomes increasingly peaked at the global minimum. In the limit of $T=0$, all the realizations are at the maximum likelihood point; that is, the sought-after point that minimizes the error.

Note that when $T=2$, we recover the probability density function $p(\mathbf{d}^{\text{obs}}; \mathbf{m})$, as defined in Section 9.3. An alternate strategy for "solving" the inverse problem is to stop the cooling at this temperature and then to produce a large set of realizations of this distribution (see Section 1.4.4). Either the entire set of solutions, itself, or a single parameter derived from it, such as the mean, can be considered the "solution" of the inverse problem.

In *MatLab*, the main part of the Metropolis-Hastings algorithm is a loop that computes the current value of T, randomly computes a $\Delta\mathbf{m}$ to produce the solution $\mathbf{m}^*$ and a corresponding error $E(\mathbf{m}^*)$, and applies the Metropolis rules to either accept or reject $\mathbf{m}^*$.

```
Dm = 0.2;
Niter=400;
for k = [1:Niter]

    % temperature falls off with iteration number
    T = 0.1 * Eg0 * ((Niter-k+1)/Niter)^2;

    % randomly pick model parameters and evaluate error
    ma(1) = random('Normal',mg(1),Dm);
    ma(2) = random('Normal',mg(2),Dm);
    da = sin(w0*ma(1)*x) + ma(1)*ma(2);
    Ea = (dobs-da)'*(dobs-da);

    % accept according to Metropolis rules
    if( Ea < Eg )
        mg=ma;
        Eg=Ea;
        p1his(k+1)=1;
    else
        p1 = exp( -(Ea-Eg)/T );
        p2 = random('unif',0,1);
        if( p1 > p2 )
```

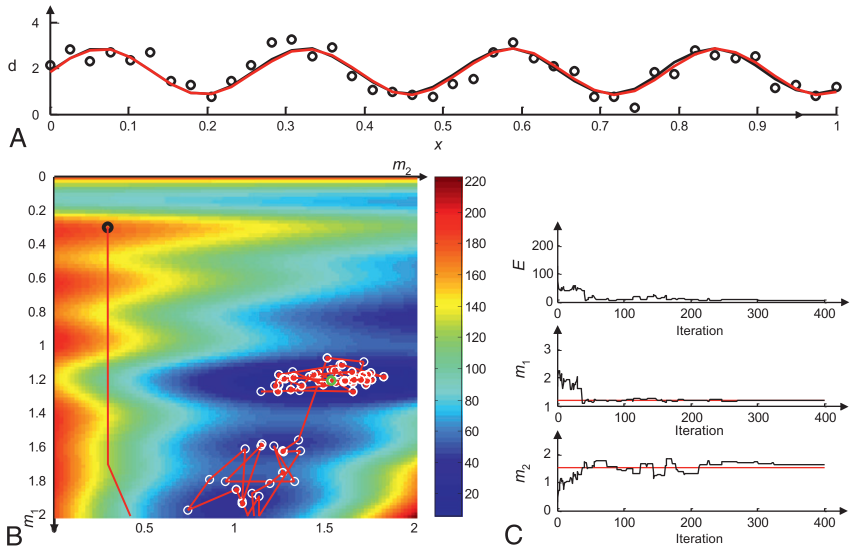

FIGURE 9.14 Simulated annealing is used to solve the same nonlinear curve-fitting problem as in Figure 9.5. (A) The observed data (black circles) are computed from the true data (black curve) by adding random noise. The predicted data (red curve) are based on the results of the method. (B) Error surface (colors), showing true solution (green circle), and a series of solutions (white circles connected by red lines) determined by the method. (C) Plot of error E and model parameters m_1 and m_2 as a function of iteration number. *MatLab* script gda09_16.

```
            mg=ma;
            Eg=Ea;
        end
    end
end
```

(*MatLab* script gda09_16)

An example is shown in Figure 9.14.

9.10 CHOOSING THE NULL DISTRIBUTION FOR INEXACT NON-GAUSSIAN NONLINEAR THEORIES

As we found in Section 5.2.4, the starting place for analyzing inexact theories is the rule for combining probability density functions to yield the total probability density function p_T. It is built up by combining three probability density functions p_A (*a priori*), p_f or p_g (theory), and p_N (null)

$$p_T(\mathbf{d},\mathbf{m}) = \frac{p_A(\mathbf{d})p_A(\mathbf{m})p_g(\mathbf{d},\mathbf{m})}{p_N(\mathbf{d})p_N(\mathbf{m})} \quad \text{or} \quad p_T(\mathbf{x}) = \frac{p_A(\mathbf{x})p_f(\mathbf{x})}{p_N(\mathbf{x})} \tag{9.28}$$

In the Gaussian case, we have been assuming $p_N \propto$ constant. However, this definition is sometimes inadequate. For instance, if $\mathbf{x}$ is the Cartesian coordinate of an object in three-dimensional space, then $P_N \propto$ constant means that the object could be anywhere with equal probability. This is an adequate definition of the null distribution. On the other hand, if the position of the object is specified by the spherical coordinates $\mathbf{x} = [r, \theta, \phi]^T$, then the statement $P_N \propto$ constant actually implies that the object is near the origin. The statement that the object could be anywhere is $P_N(\mathbf{x}) \propto r^2 \sin(\theta)$. The null distribution must be chosen with the physical significance of the vector $\mathbf{x}$ in mind.

Unfortunately, it is sometimes difficult to find a guiding principle with which to choose the null distribution. Consider the case of an acoustics problem in which a model parameter is the acoustic velocity v. At first sight it may seem that a reasonable choice for the null distribution is $p_N(v) \propto$ constant. Acousticians, however, often work with the acoustic slowness $s = 1/v$, and the distribution $p_N(v) \propto$ constant implies $p_N(s) \propto s^2$. This is somewhat unsatisfactory, as one could, with equal plausibility, argue that $p_N(s) \propto$ constant, in which case $p_N(v) \propto v^2$. One possible solution to this dilemma is to choose a null solution whose *form* is invariant under the reparameterization. The distribution that works in this case is $p_N(v) \propto 1/v$, as this leads to $p_N(s) \propto 1/s$.

Thus, while a non-Gaussian inexact implicit theory can be handled with the same machinery as was applied to the Gaussian case of Section 9.7, more care must be taken when choosing the parameterization and defining the null distribution.

9.11 BOOTSTRAP CONFIDENCE INTERVALS

The probability density function of a nonlinear problem is non-Gaussian, even when the data have Gaussian-distributed error. The simple formulas that we developed for error propagation are not accurate in such cases, except perhaps when the nonlinearity is very weak. We describe here an alternative method of computing confidence intervals for the model parameters that performs the error propagation in an alternative way.

If many *repeat* data sets were available, the problem of estimating confidence intervals could be approached empirically. If we had repeated the experiment 1000 times, each time with the exact same experimental conditions, we would have a group of 1000 data sets, all similar to one another, but each containing a different pattern of observational noise. We could then solve the inverse problem 1000 times, once for each repeat data set, make histograms of the resulting estimates of the model parameters and infer confidence intervals from them.

Repeat data sets are rarely available. However, it is possible to construct an approximate repeat data set by the *random resampling with duplication* of a single data set. The idea is to treat a set of N data as a pool of hypothetical observations and randomly draw N *realized* observations, which together constitute one repeat data set, from them. Even though the pool and the repeat data set are both of length N, they are not the same because the latter contains duplications. It can be shown that this process leads to a group of data sets that approximately have the probability density function of the original data.

As an example, suppose that a data set consists of N pairs of (x_i, d_i) observations, where x_i is an auxiliary variable. In *MatLab*, the resampling is performed as

```
rowindex = unidrnd(N,N,1);
xresampled = x( rowindex );
dresampled = dobs( rowindex );
```

(*MatLab* script gda09_17)

Here, the function `unidrnd(N,N,1)` returns a vector of N uniformly distributed integers in the range 1 through N. The inverse problem is then solved for many such repeat data sets, and the resulting estimates of the model parameters are saved. Confidence intervals are estimates as

```
Nbins=50;
m1hmin=min(m1save);
m1hmax=max(m1save);
Dm1bins = (m1hmax-m1hmin)/(Nbins-1);
m1bins=m1hmin+Dm1bins*[0:Nbins-1]';
m1hist = hist(m1save,m1bins);
pm1 = m1hist/(Dm1bins*sum(m1hist));
Pm1 = Dm1bins*cumsum(pm1);
m1low=m1bins(find(Pm1>0.025,1));
m1high=m1bins(find(Pm1>0.975,1));
```

(*MatLab* script gda09_17)

Here, estimates of the model parameter m_1 for all the repeat data sets have been saved in the vector `m1save`. A histogram `m1hist` is computed from them using the `hist()` function, after determining a reasonable range of m_1 values for its bins. The histogram is converted to an empirical probability density function `pm1`, by scaling it so that its integral is unity, and the cumulative sum function `cumsum()` is used to integrate it to a cumulative probability distribution `Pm1`. The `find()` function is then used to determine the values of m_1 that enclose 95% of the area, defining 95% confidence limits for m_1. An example is shown in Figure 9.15.

9.12 PROBLEMS

9.1. Use the principle of maximum likelihood to estimate the mean μ of N uncorrelated data, each of which is drawn from the same one-sided exponential probability density function $p(d_i) = \mu^{-1} \exp(-d_i/\mu)$ on the interval $(0, \infty)$. Suppose all the N data are equal to unity. What is the variance?

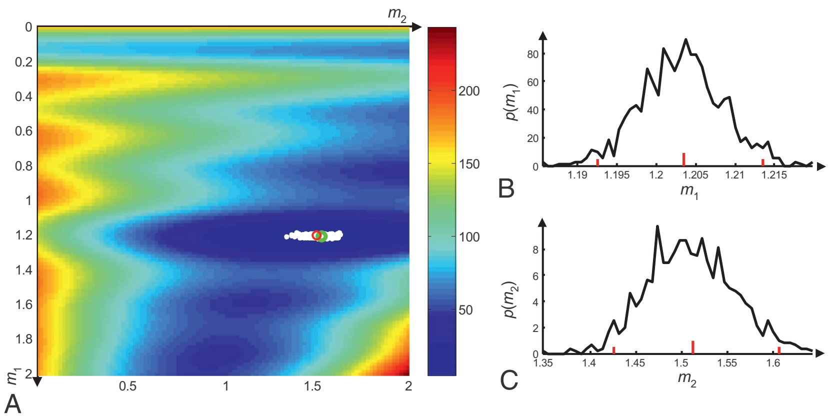

FIGURE 9.15 Bootstrap confidence intervals for model parameters estimated using Newton's method for the same problem as in Figure 9.5. (A) Error surface (colors), showing true solution (red circle), estimated solution (green circle) and bootstrap solutions (white dots). (B) Empirically derived probability density function $p(m_1)$, with m_1^{est} (large red tick) and 95% confidence limits (small red ticks). (C) Same as (B), but for $p(m_2)$. *MatLab* script gda09_17.

9.2. Experiment with the Newton's method example of Figure 9.9, by changing the initial guess `mg` of the solution. Map out the region of the (m_1, m_2) plane for which the solution converges to the global minimum.

9.3. Solve the nonlinear inverse problem $d_i = m_1 z_i^{m_2}$, with z_i an auxiliary variable on the interval (1, 2) and with $\mathbf{m}^{\text{true}} = [3, 2]^T$, using both a linearizing transformation based on taking the logarithm of the equation and by Newton's method and compare the result. Use noisy synthetic data to test your scripts.

9.4. Suppose that the zs in the straight line problem $d_i = m_1 + m_2 z_i$ are considered data, not auxiliary variables. (A) How should the inverse problem be classified? (B) Write a *MatLab* script that solves a test case using an appropriate method.

9.5. A vector **d** is constructed by adding together scaled and shifted versions of vectors **a**, **b**, and **c**. The vector **d**, of length N, is observed. The vectors, **a**, **b**, and **c**, also of length N, are auxiliary variables. They are related by $d_i = Aa_{i+p} + Bb_{i+q} + Cc_{i+r}$, where A, B, and C are unknown constants and p, q, and r are unknown positive integers (assume $a_{i+p} = 0$ if $i+p > N$ and similarly for **b** and **c**). This problem can be solved using a three-dimensional grid search over p, q, and r, only, as A, B, *and* C can be found using least squares once p, q, and r are specified. Write a *MatLab* script that solves a test case for $N = 50$. Assume that the error is $E = \mathbf{e}^T\mathbf{e}$ with $e_i = d_i - (Aa_{i+p} + Bb_{i+q} + Cc_{i+r})$.

REFERENCES

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

Tarantola, A., Valette, B., 1982. Generalized non-linear inverse problems solved using the least squares criterion. Rev. Geophys. Space Phys. 20, 219–232.