4.5

Consider a truncated Newton method and assume that $\{x^k\}$ converges to a nonsingular local minimum x^*. Assume that the matrices H^k and the directions d^k satisfy

$$\lim_{k\to\infty} \left\| H^k - \nabla^2 f(x^k) \right\| = 0, \qquad \lim_{k\to\infty} \frac{\left\| H^k d^k + \nabla f(x^k) \right\|}{\left\| \nabla f(x^k) \right\|} = 0.$$

Show that $\left\{ \|x^k - x^*\| \right\}$ converges superlinearly.

4.6

Apply Newton's method to minimization of the function $f(x) = \|x\|^3$ and show that it converges linearly to $x^* = 0$. Explain this fact in light of Prop. 1.4.1.

4.7

Apply Newton's method with the trust region implementation to a positive definite quadratic function. Show that the method terminates in a finite number of iterations.

1.5 LEAST SQUARES PROBLEMS

In this section we consider methods for solving least squares problems of the form

$$\begin{aligned} &\text{minimize} \quad f(x) = \tfrac{1}{2}\|g(x)\|^2 = \tfrac{1}{2}\sum_{i=1}^{m} \|g_i(x)\|^2 \\ &\text{subject to} \quad x \in \Re^n, \end{aligned} \tag{5.1}$$

where g is a continuously differentiable function with component functions $g_1, \ldots, g_m$, where $g_i : \Re^n \to \Re^{r_i}$. Usually $r_i = 1$, but it is sometimes notationally convenient to consider the more general case.

Least squares problems are very common in practice. A principal case arises when g consists of n scalar-valued functions and we want to solve the system of n equations with n unknowns $g(x) = 0$. We can formulate this as the least squares optimization problem (5.1) [x^* solves the system $g(x) = 0$ if and only if it minimizes $\frac{1}{2}\|g(x)\|^2$ and the optimal value is zero]. Here are some other examples:

Example 5.1 (Model Construction – Curve Fitting)

Suppose that we want to estimate n parameters of a mathematical model so that it fits well a physical system, based on a set of measurements. In particular, we hypothesize an approximate relation of the form

$$z = h(x, y),$$

where h is a known function representing the model and

$x \in \Re^n$ is a vector of unknown parameters,
$z \in \Re^r$ is the model's output,
$y \in \Re^p$ is the model's input.

Given a set of m input-output data pairs $(y_1, z_1), \ldots, (y_m, z_m)$ from measurements of the physical system that we try to model, we want to find the vector of parameters x that matches best the data in the sense that it minimizes the sum of squared errors

$$\tfrac{1}{2} \sum_{i=1}^{m} \|z_i - h(x, y_i)\|^2.$$

For example, to fit the data pairs by a cubic polynomial approximation, we would choose

$$h(x, y) = x_3 y^3 + x_2 y^2 + x_1 y + x_0,$$

where $x = (x_0, x_1, x_2, x_3)$ is the vector of unknown coefficients of the cubic polynomial.

The next two examples are really special cases of the preceding one.

Example 5.2 (Dynamic System Identification)

A common model for a single input-single output dynamic system is to relate the input sequence $\{y_k\}$ to the output sequence $\{z_k\}$ by a linear equation of the form

$$\sum_{j=0}^{n} \alpha_j z_{k-j} = \sum_{j=0}^{n} \beta_j y_{k-j}.$$

Given a record of inputs and outputs $y_1, z_1, \ldots, y_m, z_m$ from the true system, we would like to find a set of parameters $\{\alpha_j, \beta_j \mid j = 0, 1, \ldots, n\}$ that matches this record best in the sense that it minimizes

$$\sum_{k=n}^{m} \left(\sum_{j=0}^{n} \alpha_j z_{k-j} - \sum_{j=0}^{n} \beta_j y_{k-j} \right)^2.$$

This is a least-squares problem.

Example 5.3 (Neural Networks)

A least squares modeling problem that has received a lot of attention is provided by *neural networks*. Here the model is specified by a multistage system, also called a *multilayer perceptron*. The kth stage consists of n_k *activation units*, each of which is a single input-single output mapping of a given form $\phi : \Re \mapsto \Re$ to be described shortly. The output of the jth activation unit of the $(k+1)$st stage is denoted by x_{k+1}^j and the input is a linear function of the output vector $x_k = (x_k^1, \ldots, x_k^{n_k})$ of the kth stage. Thus

$$x_{k+1}^j = \phi\left(u_k^{0j} + \sum_{s=1}^{n_k} x_k^s u_k^{sj}\right), \qquad j = 1, \ldots, n_{k+1}, \tag{5.2}$$

where the coefficients u_k^{sj} (also called *weights*) are to be determined.

Suppose that the multilayer perceptron has N stages, and let u denote the vector of the weights of all the stages:

$$u = \{u_k^{sj} \mid k = 0, \ldots, N-1,\ s = 0, \ldots, n_k,\ j = 1, \ldots, n_{k+1}\}.$$

Then, for a given vector u of weights, an input vector x_0 to the first stage produces a unique output vector x_N from the Nth stage via Eq. (5.2). Thus, we may view the multilayer perceptron as a mapping h that is parameterized by u and transforms the input vector x_0 into an output vector of the form $x_N = h(u, x_0)$. Suppose that we have m sample input-output pairs $(y_1, z_1), \ldots, (y_m, z_m)$ from a physical system that we are trying to model. Then, by selecting u appropriately, we can try to match the mapping of the multilayer perceptron with the mapping of the physical system. A common way to do this is to minimize over u the sum of squared errors

$$\tfrac{1}{2}\sum_{i=1}^{m} \|z_i - h(u, y_i)\|^2.$$

In the terminology of neural network theory, the process of finding the optimal weights is known as *training the network*.

Common examples of activation units are functions such as

$$\phi(\xi) = \frac{1}{1+e^{-\xi}}, \qquad \text{(sigmoidal function)},$$

$$\phi(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}}, \qquad \text{(hyperbolic tangent function)},$$

whose gradients are zero as the argument ξ approaches $-\infty$ and ∞. For these functions ϕ, it is possible to show that with a sufficient number of activation units and a number of stages $N \geq 2$, a multilayer perceptron can approximate arbitrarily closely very complex input-output maps; see [Cyb89].

Neural network training problems can be quite challenging. Their cost function is typically nonconvex and involves multiple local minima. For large

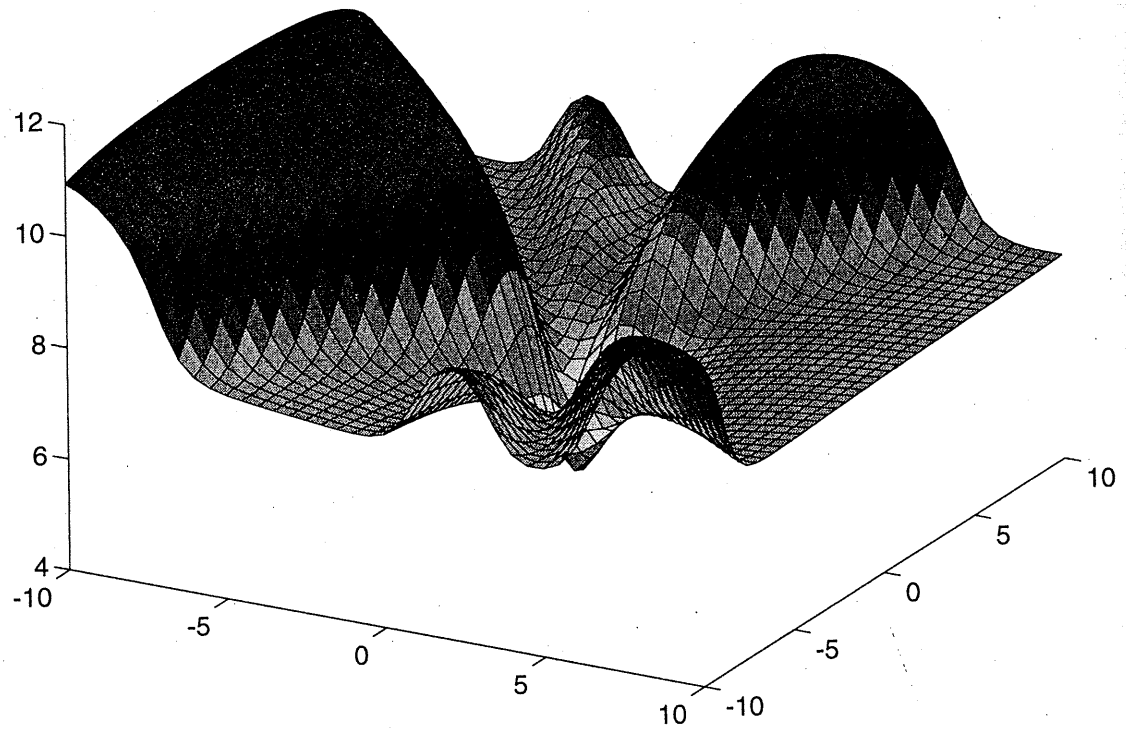

Figure 1.5.1. Three-dimensional plot of a least squares cost function

$$\tfrac{1}{2}\sum_{i=1}^{5}\big(z_i - \phi(u_1 y_i + u_0)\big)^2,$$

for a neural network training problem where there are only two weights u_0 and u_1, five data pairs, and ϕ is the hyperbolic tangent function. The data of the problem are given in Exercise 5.3. The cost function tends to a constant as u is changed along rays of the form $r\overline{u}$, where $r > 0$ and $\overline{u}$ is a fixed vector.

values of the weights u_k^{ij}, the cost becomes "flat". In fact, as illustrated in Fig. 1.5.1, the cost function tends to a constant as u is changed along rays of the form $r\overline{u}$, where $r > 0$ and $\overline{u}$ is a fixed vector. For u near the origin, the cost function can be quite complicated alternately involving flat and steep regions.

The next example deals with an important context where neural networks are often used:

Example 5.4 (Pattern Classification)

Consider the problem of classifying objects based on the values of their char-

acteristics. (We use the term "object" generically; in some contexts, the classification may relate to persons or situations.) Each object is presented to us with a vector y of features, and we wish to classify it in one of s categories $1, \ldots, s$. For example, the vector y may represent the results of a collection of tests on a medical patient, and we may wish to classify the patient as being healthy or as having one of several types of illnesses.

A classical pattern classification approach is to assume that we know the probabilities $p(j|y)$ of an object with feature vector y being of category j, where $j = 1, \ldots, s$. Then we may associate an object with feature vector y with the category $j^*(y)$ having maximum posterior probability, that is,

$$j^*(y) = \arg \max_{j=1,\ldots,s} p(j|y). \tag{5.3}$$

Suppose now that the probabilities $p(j|y)$ are unknown, but instead we have a sample consisting of m object-category pairs. Then we may try to estimate $p(j|y)$ based on the fact that, out of all functions $f_j(y)$ of y, $p(j|y)$ is the one that minimizes the expected value of $(z_j - f_j(y))^2$, where

$$z_j = \begin{cases} 1 & \text{if } y \text{ is of category } j, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, let y_i denote the feature vector of the ith object. For each category $j = 1, \ldots, s$, we estimate the probability $p(j|y)$ as a function of y, by a function $h_j(x_j, y)$ that is parameterized by a vector x_j. The function h_j could be provided for example by a neural network (cf. Example 5.3). Then, we can obtain x_j by minimizing the least squares function

$$\tfrac{1}{2} \sum_{i=1}^{m} \big(z_j^i - h_j(x_j, y_i)\big)^2,$$

where

$$z_j^i = \begin{cases} 1 & \text{if } y_i \text{ is of category } j, \\ 0 & \text{otherwise.} \end{cases}$$

This minimization approximates the minimization of the expected value of $(z_j - f_j(y))^2$. Once the optimal parameter vectors x_j^*, $j = 1, \ldots, s$, have been obtained, we may use them to classify a new object with feature vector y according to the rule

$$\text{Estimated Object Category} = \arg \max_{j=1,\ldots,s} h_j(x_j^*, y),$$

which approximates the maximum posterior probability rule (5.3).

For the simpler case where there are just two categories, say A and B, a similar formulation is to hypothesize a relation of the following form between feature vector y and category of an object:

$$\text{Object Category} = \begin{cases} A & \text{if } h(x, y) = 1, \\ B & \text{if } h(x, y) = -1, \end{cases}$$

where h is a given function and x is an unknown vector of parameters. Given a set of m data pairs $(z_1, y_1), \ldots, (z_m, y_m)$ of representative objects of known category, where y_i is the feature vector of the ith object, and

$$z_i = \begin{cases} 1 & \text{if } y \text{ is of category } A, \\ -1 & \text{if } y \text{ is of category } B, \end{cases}$$

we obtain x by minimizing the least squares function

$$\tfrac{1}{2} \sum_{i=1}^{m} \big(z_i - h(x, y_i)\big)^2.$$

The optimal parameter vector x^* is used to classify a new object with feature vector y according to the rule

$$\text{Estimated Object Category} = \begin{cases} A & \text{if } h(x^*, y) > 0, \\ B & \text{if } h(x^*, y) < 0. \end{cases}$$

There are several other variations on the above theme, for which we refer to the specialized literature.

1.5.1 The Gauss-Newton Method

Let us consider now specialized methods for minimizing the least squares cost $(1/2)\|g(x)\|^2$, starting with the most commonly used method, the *Gauss-Newton method*. Given a point x^k, the pure form of the Gauss-Newton iteration is based on linearizing g to obtain

$$\tilde{g}(x, x^k) = g(x^k) + \nabla g(x^k)'(x - x^k)$$

and then minimizing the norm of the linearized function $\tilde{g}$:

$$\begin{aligned} x^{k+1} &= \arg \min_{x \in \Re^n} \tfrac{1}{2} \|\tilde{g}(x, x^k)\|^2 \\ &= \arg \min_{x \in \Re^n} \tfrac{1}{2} \big\{ \|g(x^k)\|^2 + 2(x - x^k)' \nabla g(x^k) g(x^k) \\ &\qquad + (x - x^k)' \nabla g(x^k) \nabla g(x^k)' (x - x^k) \big\}. \end{aligned}$$

Assuming that the $n \times n$ matrix $\nabla g(x^k) \nabla g(x^k)'$ is invertible, the above quadratic minimization yields

$$x^{k+1} = x^k - \big(\nabla g(x^k) \nabla g(x^k)'\big)^{-1} \nabla g(x^k) g(x^k). \tag{5.4}$$

Note that if g is already a linear function, we have $\|g(x)\|^2 = \|\tilde{g}(x, x^k)\|^2$, and the method converges in a single iteration. Note also that the direction

$$-\big(\nabla g(x^k) \nabla g(x^k)'\big)^{-1} \nabla g(x^k) g(x^k)$$

used in the above iteration is a descent direction since $\nabla g(x^k)g(x^k)$ is the gradient at x^k of the least squares cost function $(1/2)\|g(x)\|^2$ and $\big(\nabla g(x^k)\nabla g(x^k)'\big)^{-1}$ is a positive definite matrix.

To ensure descent, and also to deal with the case where the matrix $\nabla g(x^k)\nabla g(x^k)'$ is singular (as well as enhance convergence when this matrix is nearly singular), the method is often implemented in the modified form

$$x^{k+1} = x^k - \alpha^k\big(\nabla g(x^k)\nabla g(x^k)' + \Delta^k\big)^{-1}\nabla g(x^k)g(x^k), \qquad (5.5)$$

where α^k is a stepsize chosen by one of the stepsize rules that we have discussed, and Δ^k is a diagonal matrix such that

$$\nabla g(x^k)\nabla g(x^k)' + \Delta^k : \text{positive definite.}$$

For example, Δ^k may be chosen in accordance with the Cholesky factorization scheme outlined in Section 1.4. An early proposal, known as the *Levenberg-Marquardt method*, is to choose Δ^k to be a positive multiple of the identity matrix. With these choices of Δ^k, it can be seen that the directions used by the method are gradient related, and the convergence results of Section 1.2 apply.

Relation to Newton's Method

The Gauss-Newton method bears a close relation to Newton's method. In particular, assuming each g_i is a scalar function, the Hessian of the cost $(1/2)\|g(x)\|^2$ is

$$\nabla g(x^k)\nabla g(x^k)' + \sum_{i=1}^{m}\nabla^2 g_i(x^k)g_i(x^k), \qquad (5.6)$$

so it is seen that the Gauss-Newton iterations (5.4) and (5.5) are approximate versions of their Newton counterparts, where the second order term

$$\sum_{i=1}^{m}\nabla^2 g_i(x^k)g_i(x^k) \qquad (5.7)$$

is neglected. Thus, in the Gauss-Newton method, we save the computation of this term at the expense of some deterioration in the convergence rate. If, however, the neglected term (5.7) is relatively small near a solution, the convergence rate of the Gauss-Newton method is satisfactory. This is often true in many applications such as for example when g is nearly linear, and also when the components $g_i(x)$ are small near the solution. In the case when $m = n$ and we try to solve the system $g(x) = 0$, the neglected term is zero at a solution. In this case, assuming $\nabla g(x^k)$ is invertible, we have

$$\big(\nabla g(x^k)\nabla g(x^k)'\big)^{-1}\nabla g(x^k)g(x^k) = \big(\nabla g(x^k)'\big)^{-1}g(x^k),$$

and the pure form of the Gauss-Newton method (5.4) takes the form

$$x^{k+1} = x^k - \big(\nabla g(x^k)'\big)^{-1} g(x^k),$$

which is identical to Newton's method for solving the system $g(x) = 0$ [rather than Newton's method for minimizing $\|g(x)\|^2$]. Thus, the convergence rate is typically superlinear in this case, as discussed in Section 1.4.

1.5.2 Incremental Gradient Methods*

Let us return to the model construction Example 5.1, where we want to find a vector $x \in \Re^n$ of model parameters based on data obtained from a physical system. Each component g_i in the least squares formulation is referred to as a *data block*, and the entire function $g = (g_1, \ldots, g_m)$ is referred to as the *data set*.

In many problems of interest where there are many data blocks, the Gauss-Newton method may be ineffective, because the size of the data set makes each iteration very costly. For such problems it may be more attractive to use an incremental method that does not wait to process the entire data set before updating x; instead, the method cycles through the data blocks in sequence and updates the estimate of x after each data block is processed.

For example, given x^k we may obtain x^{k+1} at the last step of the following algorithm

$$\psi_i = \psi_{i-1} - \alpha^k h_i, \qquad i = 1, \ldots, m, \tag{5.8}$$

where

$$\psi_0 = x^k,$$

$\alpha^k > 0$ is a stepsize, and the direction h_i is the gradient of the ith data block,

$$h_i = \nabla g_i(\psi_{i-1}) g_i(\psi_{i-1}). \tag{5.9}$$

This method, assuming the same stepsize α^k is used for all i, can be written as

$$x^{k+1} = x^k - \alpha^k \sum_{i=1}^{m} \nabla g_i(\psi_{i-1}) g_i(\psi_{i-1}). \tag{5.10}$$

It may be viewed as an incremental version of the steepest descent method, which is

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k) = x^k - \alpha^k \sum_{i=1}^{m} \nabla g_i(x^k) g_i(x^k).$$