Chapter 3. Optimization and Nonlinear Least Squares

1 Introduction

The word **optimization** denotes either the minimization or maximization of a function. Consider a real-valued function g with domain D in R^n. The function g is said to have a *local maximum* at point $x^* \in D$ if there exists a real number $\delta > 0$ such that $g(x) \leq g(x^*)$ for all $x \in D$ satisfying $\|x - x^*\| \leq \delta$. We define a *local minimum* in a similar way, but in the sense that inequality $g(x) \leq g(x^*)$ is reversed. If the inequality $g(x) \leq g(x^*)$ is replaced by a strict inequality

$$g(x) < g(x^*), \quad x \in D, x \neq x^*,$$

we have a strict local maximum; and if the sense of the inequality $g(x) < g(x^*)$ is reversed, we have a strict local minimum. We say that the function g has a *global (absolute) maximum* (strict global maximum) at x^* if $g(x) \leq g(x^*)$, $[g(x) < g(x^*)]$ holds for every $x \in D$. Thus a function may have many local maxima, each with a different value of $g(x)$, say, $g(x_j^0)$, $j = 1, \ldots, k$. The global maximum can always be chosen from among these local maxima by comparing their values and choosing one such that

$$g(x^*) \geq g(x_j^0), \quad j = 1, \ldots, k,$$

where $x^* \in \{x_j^0, j = 1, \ldots, k\}$. It is clear that every global maximum (minimum) is also a local maximum (minimum); however, the converse of this statement is, in general, not true. Only when $g(x)$ is a convex function in R^n and $D \subset R^n$ is a convex set is every local extremum of g at $x \in D$ also a global extremum of g over D.

Consider the following *deterministic optimization problem*:

$$\max_{x \in D \subset R^n} g(x) = g(x^*) = g^*, \tag{1}$$

where $g(x)$ is a real-valued bounded function defined on a closed bounded domain $D \subset R^n$. As an example, consider the following data fitting problem. Suppose that water is being pumped through a container to which an amount of dye has been added. Every few seconds the concentration of dye is measured in the water leaving the container. It is expected that the concentration of dye will decrease linearly over time. Since the measuring equipment maynot be perfectly accurate, it may not be possible to interpret the measurements exactly,

and the mixing may not behave exactly as predicted. The determine the rate at which the concentration decreases, the experimenter would have to approximate the data by a straight line, a line that *best* approximated the data in some sense. A common approach is to employ the method of least squares. We then run into an optimization problem.

The model above is called a **linear model** because it is a linear combination of the model functions 1 and x. *Nonlinear* models are also used. A common example is an exponential model such as $Y = \alpha \exp(\beta T) + \epsilon$. Here the model is a nonlinear function of the parameter β. If a least-squares approach were used, the following optimization problem would be obtained

$$\min_{\alpha,\beta} \sum_{i=1}^{n} [Y_i - \alpha \exp(\beta T_i)]^2.$$

This is called a **nonlinear least-squares problem**.

2 One-Dimensional Optimization

We will begin with a discussion of techniques for functions of one variable. There are several reasons for this:

1. One-dimensional problems are common.
2. Especially efficient and reliable methods are available in one dimension.
3. Many of the ideas used in high-dimensional problems can be illustrated in one dimension.
4. A major step in our n-dimensional algorithm will be the approximate solution of a one-dimensional problem.

We will describe two techniques for one-dimensional problems. The first uses derivative values to attain a high rate of convergence. The second uses only function values, and is reliable but somewhat slow.

2.1 Newton's Method

We consider minimization of a one-dimensional function $g(x)$, without any restrictions on x. We will assume that $g(x)$ has at least two continuous derivatives and that it is bounded

below. If the function is not bounded below, it may not have a local minimum, and so the problem may not make sense.

For a general function, there will be no formula for the point that minimizes $g(x)$. The idea behind Newton's method is to approximate $g(x)$ by a simpler function that we can minimize, and use the minimizer of the simpler function as the new estimate of the minimizer of $g(x)$. The process is then repeated from this new point. Since a linear function does not have a finite minimum, a quadratic function is the simplest that can be used. To form a quadratic approximation, let x_k be the current estimate of the solution x^*, and consider a Taylor series expansion of g about the point x_k:

$$g(x_k + t) = g(x_k) + tg'(x_k) + \frac{1}{2}g''(x_k) + \cdots.$$

The original minimization problem can be approximated using a Taylor series expansion

$$\begin{aligned} g(x^*) &= \min_x g(x) = \min_t g(x_k + t) \\ &= \min_t \left[g(x_k) + tg'(x_k) + \frac{1}{2}g''(x_k) + \cdots \right] \\ &\approx \min_t \left[g(x_k) + tg'(x_k) + \frac{1}{2}g''(x_k) \right]. \end{aligned}$$

Now that the higher-order terms in the series have been ignored, this last problem is just a minimization of a quadratic function of t. We hope that minimizing this quadratic approximation will be an effective substitute for directly minimizing $g(x)$. To minimize the quadratic, take the derivative with respect to t and set it equal to zero giving

$$t = -\frac{g'(x_k)}{g''(x_k)}.$$

Since t is an approximation to the step that would take us from x_k to the solution x^* of the original problem, and the algorithm is defined by the formula

$$x_{k+1} = x_k - \frac{g'(x_k)}{g''(x_k)}.$$

It is called **Newton's method**.

The principle advantage of Newton's method is that it converges quickly, with a quadratic rate of convergence. That is, if x_k is sufficiently close to x^* and if $g''(x_k) > 0$, then

$$|x_{k+1} - x^*| \leq \beta |x_k - x^*|^2$$

where β is a non-negative constant that depends on the function being minimized. This is not difficult to show. We expand $g^{'}(x)$ in a Taylor series, again about the point x_k

$$g^{'}(x) = g^{'}(x_k) + g^{(2)}(x_k)(x - x_k) + \frac{1}{2}g^{(3)}(\eta)(x - x_k)^2,$$

where η is some unknown point between x and x_k. If we let $x = x^*$ the solution, then $g^{'}(x^*) = 0$:

$$0 = g^{'}(x_k) + g^{(2)}(x_k)(x^* - x_k) + \frac{1}{2}g^{(3)}(\eta)(x^* - x_k)^2.$$

We now divide this equation by $g^{(2)}(x_k)$ and rearrange

$$x^* - \left(x_k - \frac{g^{'}(x_k)}{g^{(2)}(x_k)}\right) = \frac{g^{(3)}(\eta)}{2g^{(2)}(x_k)}(x^* - x_k)^2.$$

By the formulas for the Newton iteration, the left-hand side is just $x^* - x_{k+1}$. We then have $\beta = |g^{(3)}(\eta)/[2g^{(2)}(x_k)]|$. This is just the definition of quadratic convergence.

The assumption that $g^{(2)}(x^*) > 0$ is not unusual. If $g^{(2)}(x^*) < 0$, then x^* is a maximizer and not a minimizer of $g(x)$. If $g^{(2)}(x^*) = 0$, then $g^{'}(x)$ has a multiple zero at x^*. In this case, Newton's method will converge linearly.

Newton's method, when it works, is remarkably effective at minimizing a function. However, it is not guaranteed to converge. It can fail in several ways:

1. Each iteration of Newton's method is based on approximation a function $g(x)$ by the first three terms of its Taylor series. If this is not a good approximation to the function, then there is no reason to believe that x_{k+1} will be closer to x^* than x_k is. As an example, consider a function which can be approximated by a linear function well at the left side of x^*. If x_k falls in that neighborhood, x_{k+1} will be farther away from x^* than x_k is.
2. Newton's method could converge to a maximum of $g(x)$ rather than a minimum.
3. A final disadvantage of Newton's method is that it requires the computation of derivatives of $g(x)$ in addition to just function values; this may be difficult or impossible in a complicated real-world problem.

Newton's method is modefied in two ways to make it more reliable. First, the quadratic approximation must always have a finite minimum. If this is not true, then a different quadratic function must be used to approximate the function, one with a sensible (finite)

minimum value. This is only a temporary modification to the method; at the next iteration we will attempt to use the standard Newton method again. Second, the function value must be improved at every iteration, that is, the new iterate x_{k+1} must satisfy $g(x_{k+1}) < g(x_k)$; this can be guaranteed by choosing $x_{k+1} = x_k + \alpha p$ for some $\alpha \in (0,1)$ so as to approximately minimize $g(x_k + \alpha p)$ as a function of α. These ideas will be discussed in greater detail in the section on n-dimensional optimization.

The requirement for derivative calculations can be relaxed at the cost of extra function evaluations. The derivative $g^{'}(x_k)$ can be approximated by the forward-difference formula

$$g^{'}(x_k) \approx \frac{g(x_k + h) - g(x_k)}{h}$$

for $h \approx \sqrt{\epsilon_{mach}}$. Here ϵ_{mach} is the so-called machine constant. For example, in a four-digit floating point arithmetic

$$1.000 \times 10^{00} + 1.000 \times 10^{-04} = 1.0001 \rightarrow 1.000 \times 10^{00}.$$

The smallest floating point number that we can add to 1.0 and obtain a floating point result larger than 1.0 is called a **machine epsilon**, written ϵ_{mach}. The actural value of ϵ_{mach} depends upon whether arithmetic is rounded or chopped. On our four digit machine $\epsilon_{mach} = 0.001$ if the arithmetic is chopped, and $\epsilon_{mach} = 0.0005$ if it is rounded. Machine epsilon determines the relative accuracy of the computer arithmetic. For 32-bit floating point arithmetic, $\epsilon_{mach} = 2^{-22} \approx 1.2 \cdot 10^{-7}$ if rounding is used. Thus it is hopeless to expect more than seven decimal digits accuracy from any floating point calculation. Of a computer which uses binary arithmetic, floating point number are typically represesented in memory by 32 binary digits or bits. In the IEEE Standard, 24 bits are set aside for the mantissa and 8 bits for the exponent; these bits must also store the signs of the mantissa and the exponent.

When $|g^{'}(x_k)|$ is small relative to $|g(x_k)|$ the error in the forward-difference estimate can be large. In this case the substraction $g(x_k + h) - g(x_k)$ can produce a large relative error. In minimization algorithms, since $g^{'}(x^*) = 0$, this will happen as the algorithm converges. When $|g^{'}(x_k)|$ is small a more accurate central-difference approximation should be used, such as

$$g^{'}(x_k) \approx \frac{g(x_k + h) - g(x_k - h)}{2h}$$

which is accurate to $O(h^2)$ in exact arithmetic. To balance the approximation error and the

rounding error, we have

$$h = \left(\frac{3|g(x_k)|\epsilon_{mach}}{|g^{(3)}(x_k)|} \right)^{1/3}.$$

2.2 Golden-section Search

Suppose that g is a real-valued function defined on $[0, 1]$ and that $g(x)$ strictly decreases for $x \leq x^*$ and that $g(x)$ strictly increases for $x^* \leq x$. Such a function is called **unimodal**. Notice that a unimodal function need not be smooth or even function.. Suppose that we are allowed to evaluate the function k times, where $k > 1$ is given. The problem is to find a set of points $x_1, x_2, \ldots, x_k$ and corresponding function values $g(x_1), g(x_2), \ldots, g(x_k)$, and then determine that the optimal value of g lies in the interval $x_{i-1} \leq x^* \leq x_{i+1}$ for some I. Such an interval is called the **interval of uncertainty**. How can we use these evaluations to locate x^* with the smallest possible interval of uncertainty? To derive such a plan, we shall assume that the function F will be evaluated sequentially, that is, at one point at a time.

A theory for chosing sequentially search plans was started by J. Kiefer in the early 1950s. The algorithm for an optimal strategy will produce k test sequentially, which we choose to number $x_k, x_{k-1}, \ldots, x_2, x_1$. At the first stage two points x_k and x_{k-1} are chosen simultaneously, with $x_{k-1} < x_k$. If $g(x_{k-1}) \geq g(x_k)$, then the interval of uncertainty is reduced to $[x_{k-1}, 1]$, and we already have the value of $g(x_k)$ to help further refine the interval of uncertainty.

3 Optimization in Many Dimensions

The local minimization of a function of n variables is so important that algorithms have been devised for it over 140 years. Many of them are based on the following metaphor: Image the graph of a function as a landscape with hills and valleys. The bottoms of the valleys represent minima of the function. If we are not at a minimum, then a natural idea is to move downhill until we find one.

3.1 Method of Steepest Descent

The method of steepest descent is one of the oldest methods for determining the minimum of a function. It was proposed by A. Cauchy in 1845. It is seldom used today because it is often slow to converge.

Let the vector $(x_1, \ldots, x_n)^T$ be denoted by $\mathbf{x}$, and assume that the function $G(\mathbf{x})$ has continuous partial derivatives of several orders. Let $\bigtriangledown G = \bigtriangledown G(\mathbf{x})$ denote the gradient of G at $\mathbf{x}$, the vector whose i-th component is $(\bigtriangledown G)_i(\mathbf{x}) = \partial G/\partial x_i$. It is known that for any given point $\mathbf{x}_k$ where the gradient vector $\bigtriangledown G(\mathbf{x}_k)$ is not zero that $\bigtriangledown G(\mathbf{x}_k)$ defines the direction of maximum local increase in $G(\mathbf{x})$. Thus if we want to proced most rapidly downhill on the surface $G(\mathbf{x})$, we should consider $\mathbf{x}$ in the direction $-\bigtriangledown G(\mathbf{x}_k)$ from $\mathbf{x}_k$. This choice of direction defines the method of steepest descent.

For fixed $\mathbf{x}$ and varying α, $\mathbf{x} - \alpha \bigtriangledown G$ is a ray through $\mathbf{x}$. It is known that $-\bigtriangledown G(\mathbf{x})$ is a *downhill* direction for the function $G(\mathbf{x})$ at the point $\mathbf{x}$ in the sense that for small enough positive α, the function value will decrease if we move a small distance along the direction $-\bigtriangledown G(\mathbf{x})$, i.e., it is guaranteed that $G(\mathbf{x} - \alpha - \bigtriangledown G) < G(\mathbf{x})$ for *small* $\alpha > 0$. Cauchy proposed searching the half line defined by $\mathbf{x} - \alpha \bigtriangledown G$ $(0 < \alpha < \infty)$ for a minimum value of G. This is a one-dimensional minimization of the type discussed in previous section. Having found this minimum, one starts over and searches along the half line of steepest descent from the new $\mathbf{x}$.

Theoretical analysis has predicted that Cauchy's method will converge extremely slowly in some cases, and experience has confirmed this in many practical cases. The poor behavior of the steepest descent method is illustrated in the following example.

Example. We will use steepest descent to solve

$$\text{minimize } G(x_1, x_2, x_3) = x_1^2 + 10x_2^2 + 100x_3^2$$

with starting guess $(1,1,1)$. The solution is $\mathbf{x}^* = (0,0,0)$. The gradient of this function is given by

$$\bigtriangledown G(\mathbf{x}) = (2x_1, 20x_2, 200x_3)^T.$$

At each iteration, the new point is determined by solving the one dimensional problem

$$\min_{\alpha} G(\mathbf{x} - \alpha \bigtriangledown G(\mathbf{x})) = (x_1 - 2\alpha x_1)^2 + 10(x_2 - 20\alpha x_2)^2 + 100(x_3 - 200\alpha x_3)^2.$$

This is a quadratic function in α, and can be minimized by taking the derivative and setting it equal to zero, giving $\alpha = \frac{x_1^2+10^2x_2^2+10^4x_3^2}{2(x_1^2+10^3x_2^2+10^6x_3^2)}$. The first few iterations of the algorithm are shown in the table below.

Iteration	$\mathbf{x}$	$G(\mathbf{x})$
0	(1.0000,1.0000,1.0000)	111.0000
1	(0.9899,0.8991,-0.0091)	9.0718
2	(0.8981,0.0661,0.0751)	1.4148
3	(0.8890,0.0593,-0.0016)	0.8258
4	(0.7368,-0.0422,0.0255)	0.6260
5	(0.7287,-0.0375,0.0027)	0.5458
6	(0.6706,-0.0076,0.0190)	0.4863

The convergence is slow. After another 180 iterations, reasonable results are obtained: $\mathbf{x} = (1.07 \times 10^{-4}, 0, 0.301 \times 10^{-6})^T$ and $G(\mathbf{x}) = 1.24 \times 10^{-9}$.

We now give an interpretation why the above minimization problem takes so long to converge. For the ease of illustration, we consider a two-dimension surface with a contour which looks like an elong ellipse. Recall that the steepest descent will move to the direction changing most quickly. Therefore, the path to the minimum at the center proceeds very slowly by a route that gradually works down the valley, oscillating back and forth in the local gradient directions. There is great need to accelerate the convergence.

3.2 Newton's Method and Some Modification

Newton's method (also called the Newton-Raphson method) is a widely used and often-studied method for minimization. The method requires use of both the gradient vector and the Hessian matrix in computations; hence it places more burden on the user to supply derivatives of the objective function than does the steepest descent method.

We will write the Taylor series in matrix/vector form. In two dimensions, the second-order Taylor series approximation is

$$\begin{aligned} G(x_1+p_1, x_2+p_2) &\approx G(x_1,x_2) + p_1\frac{\partial G(x_1,x_2)}{\partial x_1} + p_2\frac{\partial G(x_1,x_2)}{\partial x_2} \\ &+ \frac{1}{2}\left(p_1^2\frac{\partial^2 G(x_1,x_2)}{\partial x_1^2} + 2p_1p_2\frac{\partial^2 G(x_1,x_2)}{\partial x_1 \partial x_2} + p_2^2\frac{\partial^2 G(x_1,x_2)}{\partial x_2^2}\right). \end{aligned}$$

Let $\bigtriangledown^2 G$ be the constant matrix of second partial derivatives of G at $\mathbf{x}_k$-the so-called **Hessian matrix**:

$$\bigtriangledown^2 G_{ij} = \frac{\partial G(\mathbf{x})}{\partial x_i \partial x_j}.$$

If the notations for the gradient and Hessian matrix are used, we can write down Taylor series in many dimensions which takes the form

$$Q(\mathbf{p}) = G(\mathbf{x}) \approx G(\mathbf{x}) + \mathbf{p}^T \bigtriangledown G(\mathbf{x}) + \frac{1}{2}\mathbf{p}^T \bigtriangledown^2 G(\mathbf{x})\mathbf{p}.$$

When $\mathbf{x}_k$ is close to $\mathbf{x}^*$ we can expect that the above quadratic function will approximate $G(\mathbf{x})$. To obtain the step $\mathbf{p}$, we now minimize this quadratic as a function of $\mathbf{p}$ by forming its gradient with respect to $\mathbf{p}$

$$\bigtriangledown_{\mathbf{p}} Q(\mathbf{p}) = \bigtriangledown_{\mathbf{p}}\left(\mathbf{p}^T \bigtriangledown G(\mathbf{x}) + \frac{1}{2}\mathbf{p}^T \bigtriangledown^2 G(\mathbf{x})\mathbf{p}\right) = \bigtriangledown G(\mathbf{x}) + \bigtriangledown^2 G(\mathbf{x})\mathbf{p}$$

and setting it equal to zero

$$\bigtriangledown^2 G(\mathbf{x})\mathbf{p} = - \bigtriangledown G(\mathbf{x}).$$

This is a set of n linear equations in the n unknowns $\mathbf{p} = (p_1, \ldots, p_n)^T$. These linear equations are called the **Newton equations**. If $\bigtriangledown^2 G(\mathbf{x})$ is positive definite, this suggests the general iterative scheme

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p} = \mathbf{x}_k - [\bigtriangledown^2 G(\mathbf{x}_k)]^{-1} \bigtriangledown G(\mathbf{x}).$$

When $G(\mathbf{x})$ is closely approximated by $Q(\mathbf{p})$ in the neighborhood of $\mathbf{x}^*$, convergence will normally be at a quadratic rate if the Hessian is positive definite at each step.

One problem with Newton's method is that the Hessian may not be positive definite at each iteration. Thus the method requires modification to insure that the resultant method is acceptable but still retains the desirable characteristics of Newton's method. As a simple illustration, consider the one-dimensional example

$$G(x) = 2x^3 - 5x^2 + 4x$$

with initial guess $x_0 = 0$. Then $G(x_0) = 0$. We compute the Newton step using

$$p = -[\bigtriangledown^2 G(\mathbf{x}_k)]^{-1} \bigtriangledown G(\mathbf{x}) = -(12x_0 - 10)^{-1}(6x_0^2 - 10x_0 + 4) = 0.4.$$

Then $G(x_0 + p) = G(0.4) = 0.9280 > G(x_0)$. In fact, it is straightforward to check that $G(x_0 + \alpha p) > G(x_0)$ for all $\alpha \in (0, 1]$. Thus p is not a descent direction.

Suppose that the inverse Hessian matrix $[\bigtriangledown^2 G(\mathbf{x}_k)]^{-1}$ is positive definite, that is, it satisfies the condition $\mathbf{z}^T[\bigtriangledown^2 G]^{-1}\mathbf{z} > 0$ for all $\mathbf{z} \neq 0$. In this case, the Newton direction is guaranteed to be a downhill direction since

$$\begin{aligned} G(\mathbf{x} + \epsilon\mathbf{p}) &= G(\mathbf{x}) + \bigtriangledown G(\mathbf{x})(\epsilon\mathbf{p}) + O(\epsilon^2) \\ &= G(\mathbf{x}) + \epsilon \bigtriangledown G^T(- \bigtriangledown^2 G^{-1} \bigtriangledown G) + O(\epsilon^2) \\ &= G(\mathbf{x}) - \epsilon \bigtriangledown G^T \bigtriangledown^2 G^{-1} \bigtriangledown G + O(\epsilon^2). \end{aligned}$$

Several modifications have been proposed to remedy the problem that the Hessian may not be positive definite at each iteration. All incorporate a nonconstant step size to help avoiding stepping across a minimum and possibly moving outside the neighborhood where a quadratic function gives an adequate fit. We will now describe some of the proposed modification.

Since few functions will have a positive Hessian matrix for all $\mathbf{x}$, some strategy is required to guarantee a descent direction. One successful idea, due to Gill and Murray (1974), is to replace $\bigtriangledown^2 G$ by a positive-definite matrix $\bar{\bigtriangledown}^2 G$. Greenstadt (1967) suggests a strategy to replace all negative eigenvalues of Hessian matrix with their absolute value and replacing zero eigenvalues with a small positive number. Marquardt (1963) and Levenberg (1944) suggest another modification which is described by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_i[\bigtriangledown^2 G(\mathbf{x}_k) + \tau_i \mathbf{I}_n]^{-1} \bigtriangledown G(\mathbf{x}_k)$$

where $\tau_i \geq 0$ is chosen so that $\bigtriangledown^2 G(\mathbf{x}_k) + \tau_i \mathbf{I}_n$ is positive definite. Note that as τ_i approaches infinity the above mentioned direction approaches that of steepest descent, and when τ_i is small the direction specified is close to the Newton direction. Thus the direction used is an interpolation between the Newton and steepest descent direction.

To avoid stepping across a minimum and possibly moving outside the neighborhood where a quadratic function gives an adequate fit, we can adjust the step size by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_i[\bigtriangledown^2 G(\mathbf{x}_k)]^{-1} \bigtriangledown G(\mathbf{x}_k).$$

The stationary point of

4 Linear Statistical Models

Linear models form the core of classical statistics, and Splus provides extensive facilities to fit and manipulate them. The main function for fitting linear models is *lm*. The usual linear model is written as

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{e} \tag{2}$$

where $\mathbf{X}$ is a fixed known $n \times p$ matrix containing $n \geq p$ values for each of p independent variables, $\mathbf{Y}$ is an $n \times 1$ vector of observed values of the dependent variable y, and $\mathbf{e}$ is a vector of random disturbances under the usual conditions of zero mean and covariance matrix $\sigma^2\mathbf{I}_n$. The estimate of unknown parameter β is usually obtained by finding $[\mathbf{b}]$ that minimizes $\|\mathbf{Y} = \mathbf{X}\mathbf{b}\|^2$, where $\|\cdot\|$ denotes the Euclidean norm.

Fisher (1947) published a dataset of H.G.O. Holck giving the heart weights, body weights and sex of 47 female and 97 male adult cats. These had been used in bioassay experiments with the muscle-relaxing drug digitalis. It appears that cats with a body weight of at least 2 kilograms were selected. Fisher's interest in the data was to study the variation of heart weight with body weight and to discover if this differs by sex. Consider fitting separate linear regression models for male and female cats. In Splus, we can do it by the following:

attach(*cats*)

catsF ¡- lm(Hwt Bwt, data = cats, subset = Sex ==”F”)

The second line fits a linear regression for the females. When Splus looks for an object, it searches through a sequence of places known as the *search list*. Usually the first entry in the search list is the .Data subdirectory of the current working directory. The names of the places currently making up the search list are given by invoking the function search(). Extra directories, data frames or lists can be added to the list with the function *attach* and removed with the function *detach*. The dataset we just mention is given in data frame cats in the library with three columns, Sex, Bwt and Hwt.

The main arguments for the function *lm* are

$$lm(formula, data, weights, subset, na.action)$$

where

- formula: dependent variable regressor(s) separated by +,
- data: an optional data frame,
- weights: a vector of positive weights, if non-uniform weights are needed,
- subset: a vector specifying a subset of the data is to be used (by default all rows are used)
- na.action: a function specifying how missing values are to be handled (by default, missing values are not allowed).

A typical mathematical treatment of this subject finds [**b**] as a solution to the *normal equations*

$$\mathbf{X}^t\mathbf{X}\mathbf{b} = \mathbf{X}^t\mathbf{Y} \tag{3}$$

and often utilizes the vector of residuals

$$\mathbf{Z} = \mathbf{Y} - \mathbf{X}\mathbf{b} \tag{4}$$

in subsequent discussion. The inverse of $\mathbf{X}^t\mathbf{X}$, either $\mathbf{X}^t\mathbf{X}^{-1}$ or $\mathbf{X}^t\mathbf{X}^-$, is another quantity of direct interest in these discussions.

In Splus, the function *solve* inverts matrices and solve systems of linear equations; solve(A) inverts A and solve(A,b) solves A(If the system is over-determined, the least-squares fit is found, but matrices of less than full rank give an error.) The function *chol* returns the Choleski decomposition $A = U^tU$ of a nonnegative definite symmetric matrix. (Note that this is an unusual convention; more commonly the lower-triangular form $B = LL^t$ with $L = U^t$ is used.) Function *backsolve* solves upper triangular systems of matrices, and is often used in conjunction with *chol*.

Two different basic approaches are used for regression computations. One approach is to form the normal equations and proceed with these equations as a starting point. This approach is rejected by some people because it may be very difficult to form the matrices in (4) to a high degree of accuracy. The other approach is to avoid formation of the normal equations. Instead, computations proceed based on the given data ($\mathbf{X}$, $\mathbf{Y}$) and utilize orthogonal triangularization in the process of performing regression analysis.

5 General Minimization and Maximum Likelihood Estimation

The waiting times between eruptions in the data *faithful* are strongly bimodal. Try the following program in Splus:

- attach(faithful)
- hist(waiting, prob=T, xlim=c(35,105), ylim=c(0,0.045), style.bar="old")
- wait.dns ¡- density(waiting, 200, width=10)
- lines(wait.dns, lty=2)

From inspection of this figure a mixture of two normal distributions would seem to be a reasonable descriptive model for the marginal distribution of waiting times. Useful references for mixture models include Everitt and Hand (1981), Titterington, Smith and Markov (1985) and McLachlan and Basford (1988). Everitt and Hand (1981) describe the EM algorithm for fitting a mixture model, which is simple.