6 Nonlinear Estimation

In this final chapter of the volume, we describe the set of numerical procedures by which ARIMA model parameters are estimated. These procedures are relatively more complicated than the procedures used to estimate the parameters of many statistical models used by social scientists. We have left this material until last because some readers may not be interested in the details of estimation. Models and parameter estimates can be competently interpreted without reference to the mechanical routines of estimation. Those readers who wish only to interpret the results of time series research may thus not wish to invest the time and effort required to master this material. Those readers who plan to estimate ARIMA models in the course of instruction or research, on the other hand, will find that this material is absolutely essential.

The statistical models most commonly used by social scientists are *linear* models. For a finite set of variables, $\{Y, X_1, \ldots, X_n, u\}$, a linear model is defined as any linear combination of those variables. The familiar linear prediction equation, for example,

$$Y = b_0 + b_1X_1 + \ldots + b_nX_n + u,$$

is a linear combination of a finite set of variables and, thus, is a linear model. In contrast, the conventional log-linear version of this same prediction equation,

$$Y = b_0(X_1)^{b_1} \ldots (X_n)^{b_n}(u),$$

is *not* a linear combination and, thus, is *not* a linear model. If the model is log-transformed, however:

$$\text{Ln}(Y) = \text{Ln}(b_0) + b_1\text{Ln}(X_1) + \ldots + b_n\text{Ln}(X_n) + \text{Ln}(u),$$

a linear model is defined, though not in terms of the original set of variables. Finally, the model

$$Y = \text{Ln}(b_0)\,[\text{Ln}(X_1)]^{b_1} \ldots [\text{Ln}(X_n)]^{b_n}[\text{Ln}(u)],$$

is *intrinsically* nonlinear. This model is not a linear combination of the variables nor is there any transformation which will result in a linear combination.

When a model is linear (or when it can be made linear with a transformation), maximum likelihood estimates of its parameters can be obtained as analytical solutions to a set of equations. This procedure is a straightforward one which could be executed (though tediously) with paper and pencil. When a model is intrinsically nonlinear, however, as the general ARIMA model is, analytical solutions for parameter estimates are ordinarily undefined. Numerical solutions must be obtained, usually by means of an iterative algorithm wherein parameter estimates are successively refined to a desired degree of precision. While these numerical algorithms could theoretically be executed with paper and pencil, the sheer number of iterative computations required is so large as to render this strategy practically impossible. Estimation of ARIMA model parameters is hence, by definition, dependent on sophisticated computer software.

Our discussion of ARIMA parameter estimation will begin with the logic of maximum likelihood estimation (MLE). After developing this logic for a simple problem, we demonstrate that the MLEs of ARIMA parameters are identical with the least-squares estimates. Due to this identity, MLEs of ARIMA parameters can be derived through a numerical minimization of the sum of squares function. We illustrate the general principles of least-squares estimation by plotting the sum of square functions for several time series analyzed in earlier chapters. The most commonly used numerical algorithms for minimizing the sum of squares function are discussed, and at the end of the chapter we describe the characteristics of several ARIMA software packages.

6.1 Maximum Likelihood Estimation

Consider a coin-flip experiment in which the probability of realizing k heads in N flips is given by the probability density function:

$$P(\text{k heads in N flips}) = \frac{N!}{k!\,(N-k)!}\,p^k(1-p)^{N-k}$$

for p = probability of heads on any flip. If a fair coin is flipped 10 times with the result of 6 heads, the probability of this result is:

$$P(\text{6 heads in 10 flips}) = \frac{10!}{6!\,4!}\,(.5)^6(.5)^4$$
$$= .205065.$$

This probability may now be used to test the null hypothesis that the coin is fair, that is, that p = .5.

Statistical inferences of this sort are called *classical* inferences. In contrast, *Bayesian* inferences, of which MLE is a special case, use the empirical outcome of an experiment to derive a set of model parameters (p in this case) which is most likely to have generated those outcomes. Again, if a coin is flipped 10 times with the result of 6 heads, a *likelihood function*, L, is defined as:

$$L = (p)\,(p)\,(p)\,(p)\,(p)\,(p)\,(1-p)\,(1-p)\,(1-p)\,(1-p)$$
$$= p^6(1-p)^4.$$

This likelihood function is identical with the classical probability density function except for a multiplicative constant which in this case is the term

$$\frac{N!}{k!\,(N-k)!}.$$

The likelihood function describes a relationship between the model parameter, p, and the likelihood that a particular value of the parameter has generated the set of outcomes. Substituting successively larger values of p into the likelihood function, starting with p = 0.0, the relationship emerges as:

p	likelihood
0.0	0.000000
.1	.000006
.2	.000026
.3	.000175
.4	.000531
.5	.000977
.6	.001194
.7	.000953
.8	.000491

.9	.000053
1.0	0.000000

a parabola whose maximum value occurs when p = .6. *This is the MLE of p, that is, the value of p which is most likely to generate 6 heads in 10 flips.*

But there is no guarantee that the likelihood function attains its maximum value at exactly p = .6. It is possible, for example, that the maximum occurs at p = .59 or .61. To derive the *exact* maximum of the likelihood function, a differential calculus operation is required. In the general case, the likelihood for the result of k heads in N flips is:

$$L = p^k(1-p)^{N-k}.$$

The calculus operation will be simplified, however, if the likelihood function is log-transformed:

$$Ln(L) = kLn(p) + (N-k)Ln(1-p).$$

The log-likelihood function is linear and, thus, will always be easier to differentiate than the likelihood function.

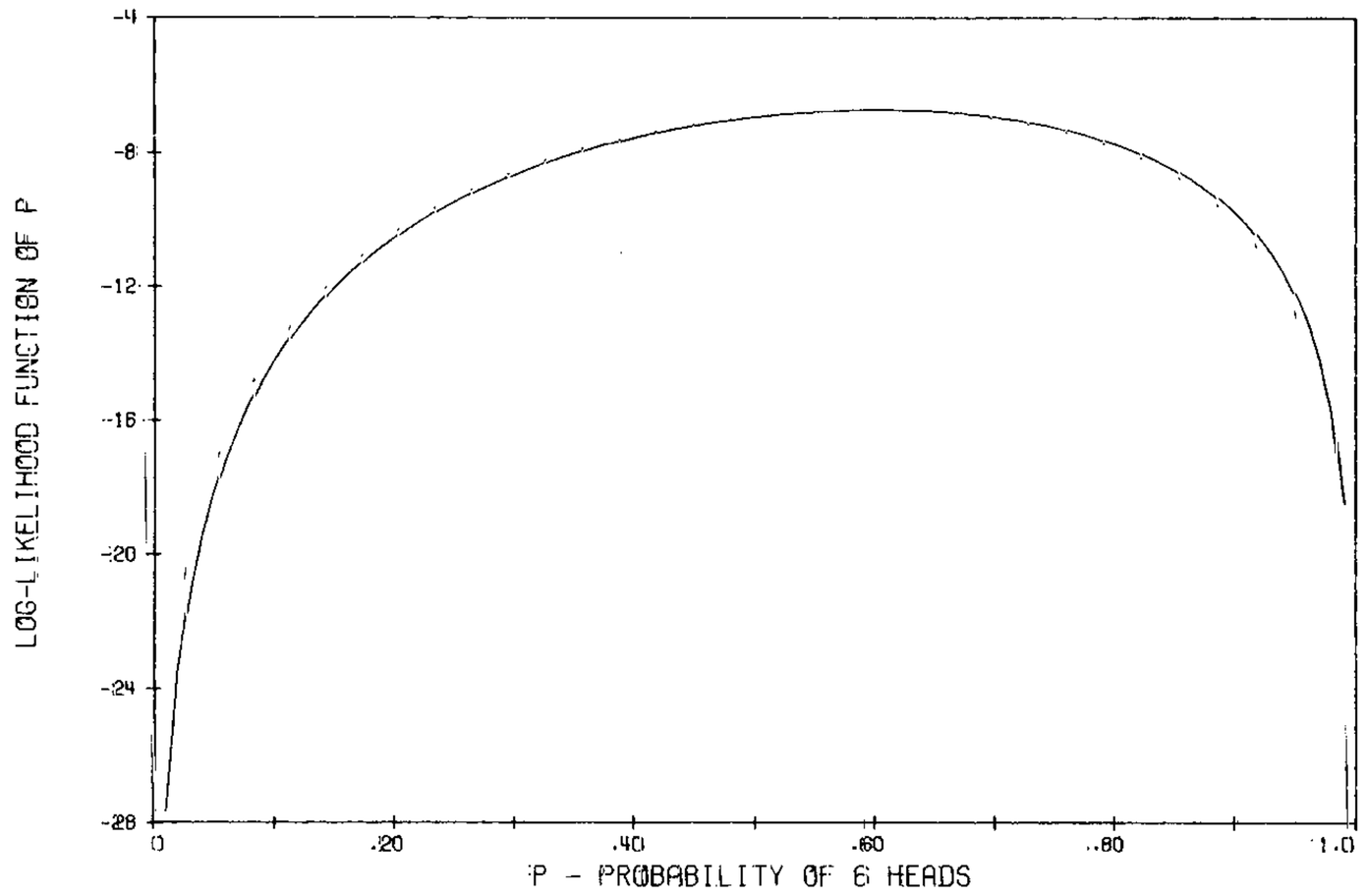

FIGURE 6.1(a) Log-Likelihood Function for a Coin-Flip Experiment: 6 Heads in 10 Flips

As the log-transformation is a monotonic linear transformation, this simplification will have no bearing on the outcome. Figure 6.1(a) shows the log-likelihood function for the result of 6 heads in 10 flips. The log-likelihood function shown here is unimodal with a clearly defined maximum point. While we will not do so here, it is easily demonstrated that the likelihood and log-likelihood functions are maximized by the same value of p.

To obtain the MLE of p, the log-likelihood function is differentiated with respect to p. This operation is:

$$\frac{\partial\,[Ln(L)]}{\partial\,p} = k/p - (N-k)/(1-p).$$

The solution to this derivative

$$\frac{\partial\,[Ln(L)]}{\partial\,p} = 0$$

$$k/p - (N-k)/(1-p) = 0$$

$$k - kp = pN - pk$$

$$p = k/N$$

is the MLE of p. Substituting k = 6 and N = 10 into this formula, results in p = .6.

We do not require a calculus background of the reader but the principles of calculus maximization should be clear from the log-likelihood function shown in Figure 6.1(a). The log-likelihood function (and the likelihood function itself) increases until p = k/N and then decreases. Calculus maximization is a simple method of discovering the exact point at which the log-likelihood function attains its maximum value.

The logic of MLE may now be generalized to the more difficult problem of obtaining ARIMA parameter estimates. In the coin-flip experiment, binomial outcomes, heads or tails, were observed and MLE procedures were used to obtain a value of the parameter p which was most likely to have generated those outcomes. In a time series analysis, white noise outcomes (random shocks) are observed and MLE procedures are used to obtain values of ARIMA parameters which are most likely to have generated those shocks. Of course, the random shocks are not really observed but we will assume for the moment that they are.

Denoting the first random shock of a time series as a_1, the classical probability density function is:

$$P(a_1) = \frac{e^{-(a_1^2/2\sigma_a^2)}}{\sigma_a \sqrt{2\pi}}.$$

This formidable appearing expression is merely the "Normal curve," the probability density function associated with all Normally distributed variables. As white noise has the statistical property

$$a_t \sim NID(0,\sigma_a^2),$$

the Normal (or Gaussian) function is implied.

If one were interested only in classical inference (testing a null hypothesis that all model parameters were zero, for example), inferences could follow directly from this function. To obtain MLEs of the ARIMA parameters, however, a likelihood function is required. The likelihood function for a_1 is:

$$L(a_1) = \frac{e^{-(a_1^2/2\sigma_a^2)}}{\sigma_a}.$$

This likelihood function differs from the probability density function only by the multiplicative constant, $1/\sqrt{2\pi}$.

The second shock of the series, a_2, has an identical likelihood function. Because each shock of a white noise process is independent of every other shock, the likelihood function for a_1 *and* a_2 is simply the *product* of their individual likelihoods. That is,

$$L(a_1,a_2) = [L(a_1)\,L(a_2)] = \frac{e^{-(a_1^2/2\sigma_a^2)}}{\sigma_a} \times \frac{e^{-(a_2^2/2\sigma_a^2)}}{\sigma_a}$$

$$= \frac{e^{-(a_1^2 + a_2^2)/2\sigma_a^2}}{\sigma_a^2}.$$

In the same way, and for the same reasons, the likelihood function for all N shocks of a time series is:

$$L(a_1, \ldots, a_N) = \frac{e^{-(\sum_{i=1}^{N} a_i^2)/2\sigma_a^2}}{\sigma_a^N}.$$

The log-likelihood function is thus:

$$Ln(L) = -\sum_{i=1}^{N} a_i^2/2\sigma_a^2 - N\,Ln(\sigma_a).$$

The utility of log-transforming the likelihood function is apparent in this case. The log-likelihood function is the simple sum of two terms. Moreover, the second term of the log-likelihood function,

$$-N\,Ln(\sigma_a),$$

will be negative whenever $\sigma_a > 1$; will be zero whenever $\sigma_a = 1$; and will positive whenever $\sigma_a < 1$. But the first term of the log-likelihood function,

$$-\sum_{i=1}^{N} a_i^2/2\sigma_a^2,$$

will always be negative. The maximum of the log-likelihood function will thus occur when the sum of squared shocks is at its minimum. In other words,

$$\text{Max}\left[-\sum_{i=1}^{N} a_i^2/2\sigma_a^2 - N\,Ln(\sigma_a)\right] = \text{Min}\left[\sum_{i=1}^{N} a_i^2\right].$$

This important result means that *the MLEs of ARIMA parameters are the "least-squares" estimates*. To obtain MLEs of ARIMA parameters, then, a set of parameters must be selected so as to minimize the sum of squared random shocks.

The same calculus procedure used to maximize the log-likelihood function can be used to minimize the sum of squares function. First, the sum of squares function must be written in a way that explicitly represents each parameter. Noting that the general ARIMA (p,0,q) model (for a stationary time series or a differenced nonstationary series) is:

$$(1 - \phi_1 B - \ldots - \phi_p B^p) z_t = (1 - \theta_1 B - \ldots - \theta_q B^q) a_t,$$

then the random shock, a_t, is defined as:

$$a_t = \frac{1 - \phi_1 B - \ldots - \phi_p B^p}{1 - \theta_1 B - \ldots - \theta_q B^q} z_t.$$

So the sum of squares function may be expressed as:

$$\sum_{i=1}^{N} a_{ii}^2 = \sum_{ii=1}^{N} \left[\frac{1 - \phi_1 B - \ldots - \phi_p B^p}{1 - \Theta_1 B - \ldots - \Theta_q B^q} z_i\right]^2.$$

We will call this expression the *conditional* sum of squares function. Denoting a vector of ARIMA parameter estimates as

$$\hat{\beta} = \{\hat{\phi}_p, \hat{\phi}_p, \hat{\Theta}_q, \hat{\Theta}_Q, \hat{\delta}_r, \hat{\omega}_s\},$$

we will denote the conditional sum of squares function as $S(\hat{\beta})$. This conditional sum of square function is conditional upon a specific set of parameter estimates.[1] The function will change its value as the parameter estimates change. The *absolute minimum* of this function, its smallest possible value, that is, will occur when solved derivatives,

$$0 = \frac{\partial S(\hat{\beta})}{\partial \hat{\phi}_p} = \frac{\partial S(\hat{\beta})}{\partial \hat{\phi}_p} = \frac{\partial S(\hat{\beta})}{\partial \hat{\Theta}_q} = \frac{\partial S(\hat{\beta})}{\partial \hat{\Theta}_Q} = \frac{\partial S(\hat{\beta})}{\partial \hat{\delta}_r} = \frac{\partial S(\hat{\beta})}{\partial \hat{\omega}_s},$$

are taken as the parameter estimates. Because these solved derivatives minimize the sum of squares function, they are the MLEs of the ARIMA parameters.

When a model is nonlinear, however, these derivatives are not easily solved. To illustrate this point, consider the simple *linear* model with

$$Y_i = b_0 + b_1 X_i + u_i$$

for the i^{th} case. The sum of the squared disturbance terms for this model is:

$$\sum_{ii=1}^{N} u_i^2 = \sum_{i=1}^{N} (Y_i - b_0 - b_1 X_i)^2.$$

The sum of squares is minimized when its solved derivatives are taken as the parameter estimates. These derivatives are:

$$\frac{\partial S(\hat{\beta})}{\partial \hat{b}_0} = -2 \sum_{i=1}^{N} (Y_{ii} - \hat{b}_0 - \hat{b}_1 X_i)$$

$$\frac{\partial S(\hat{\beta})}{\partial \hat{b}_1} = -2X_i \sum_{i=1}^{N} (Y_i - \hat{b}_0 + \hat{b}_1 X_i).$$

These solved derivatives are called the "normal equations" of a model. Whenever a model is linear, its normal equations will be linear. In this particular case, there are two linear normal equations in two unknowns, $\hat{b}_0$ and $\hat{b}_1$. The equations may thus be solved for the unknowns to give MLEs of the parameters.

But now consider the simple ARIMA (0,0,1) model,

$$y_t = (1 - \Theta_1 B)a_t$$

or

$$a_t = \sum_{k=0}^{\infty} \Theta_1^k y_{t-k}.$$

The sum of squared shocks for this model is:

$$\sum_{t=1}^{N} a_t^2 = \sum_{t=1}^{N} \left[\sum_{k=0}^{\infty} \Theta_1^k y_{t-k}\right]^2$$

and the solved derivative is:

$$\frac{\partial S(\hat{\beta})}{\partial \hat{\Theta}} = 2 \sum_{t=1}^{N} \left[\sum_{k=0}^{\infty} \hat{\Theta}_1^k y_{t-k}\right] \left[k \sum_{k=0}^{\infty} \hat{\Theta}_1^{k-1} y_{t-k}\right] = 0.$$

This normal equation is nonlinear. In all cases, a nonlinear model will have nonlinear normal equations which cannot be solved with analytical methods.

In the next section, we will describe numerical (as opposed to analytical) methods for minimizing the $S(\hat{\beta})$ function. Meanwhile, Figures 6.1(b) and 6.1(c) illustrate the logic of minimization. Figure 6.1(b) is a plot of the sum of squares function for an ARIMA (0,1,1) model fit to the IBM stock price time series originally presented in Section 2.1 (Series B). This model is:

$$(1 - B)Y_t = z_t$$
$$z_t = (1 - \Theta_1 B)a_t.$$

It has only one parameter and the x-axis (the horizontal axis, that is) gives all

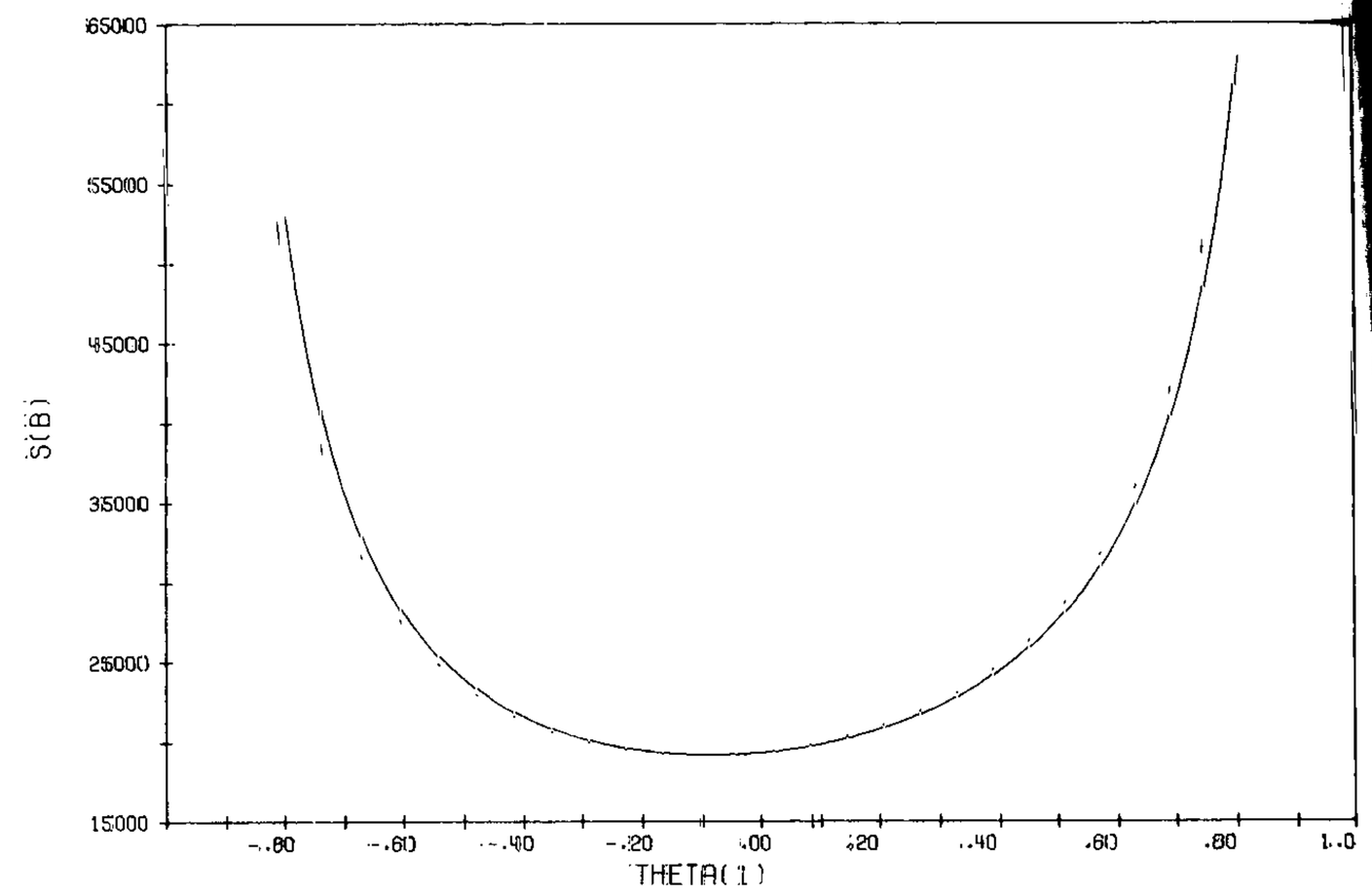

FIGURE 6.1(b) S ($\hat{\beta}$) for an ARIMA (0,1,1) Model Fit to Series B

values of $\hat{\Theta}_1$ within the bounds of invertibility. The y-axis of the plot (the vertical axis, that is) gives the corresponding value of the sum of squares function, $S(\hat{\Theta}_1)$. The minimum value of this sum of squares function appears to be at or near zero. In other words, $\hat{\Theta}_1 \simeq 0$ minimizes the $S(\hat{\Theta}_1)$ function. This approximate value will be the least-squares estimate of Θ_1. It will also be the MLE because, as has been demonstrated, the least-squares estimates and the MLEs of ARIMA parameters are identical.

Figure 6.1(c) shows a plot of the sum of squares function for the Directory Assistance time series analyzed in Section 3.1. The model used for this plot is:

$$Y_t = \frac{1 - \Theta_{12}B^{12}}{(1 - B)(1 - B^{12})} a_t + \omega_0 I_{147}.$$

It has two parameters. The sum of squares, $S(\hat{\Theta}_{12}, \hat{\omega}_0)$, is thus a three-dimensional surface. The x-axis of this plot gives a range of values for $\hat{\omega}_0$ while the y-axis gives the range of values for $\hat{\Theta}_{12}$. The contours of this plot give the height or depth of the sum of squares surface for each pair of

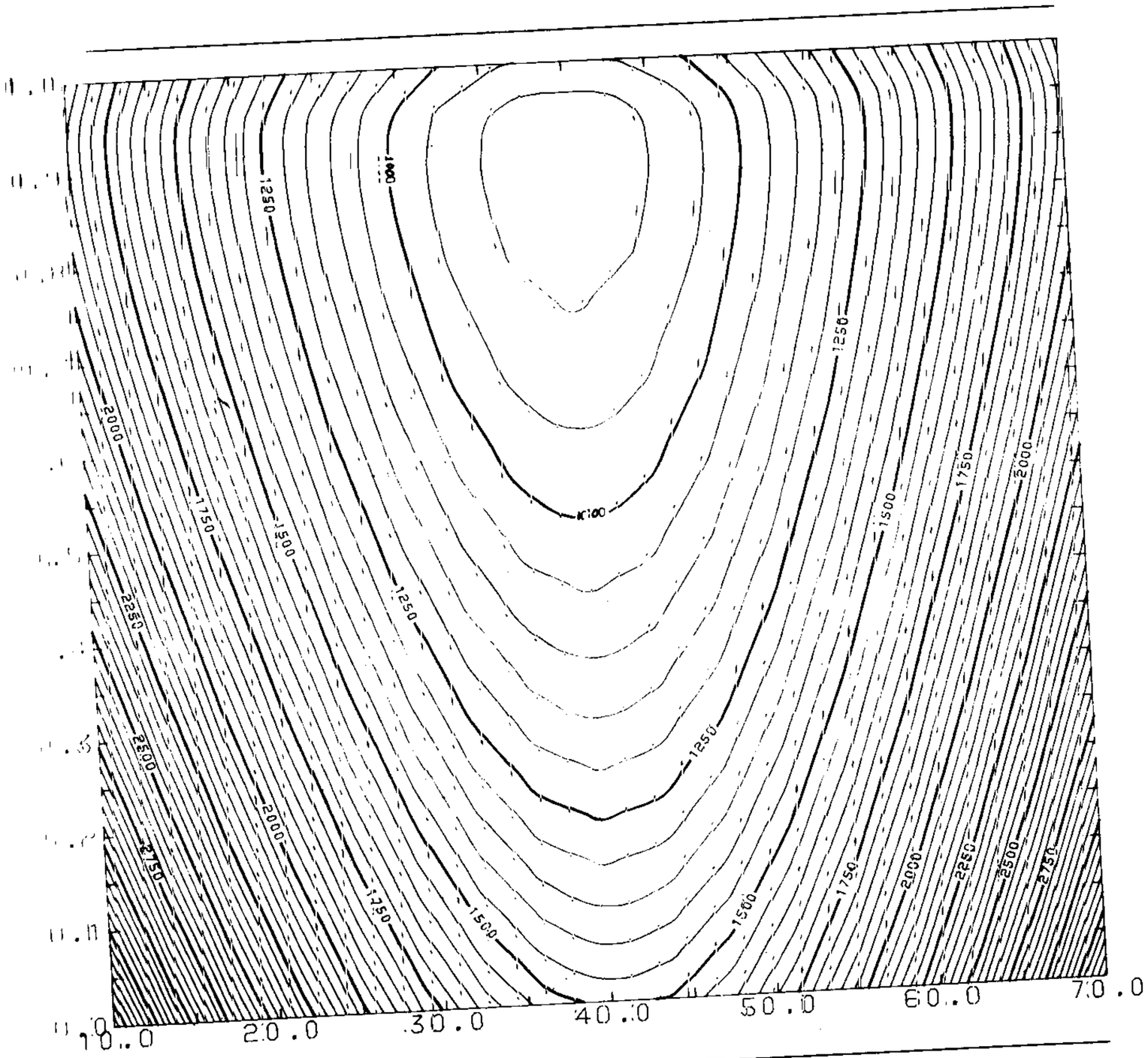

FIGURE 6.1(c) S ($\hat{\beta}$) for the Directory Assistance Example of Section 3.1

parameter estimates $(\hat{\Theta}_{12}, \hat{\omega}_0)$. The plot can be interpreted in the same manner as a topological map. The estimation *problem* in this case, put simply, is to find a pair of parameter estimates which minimize the sum of squares function. The minimum occurs in the "depression" or "valley" at the top center of the plot, corresponding approximately with $\hat{\Theta}_{12} \simeq .85$ and $\hat{\omega}_0 \simeq 40{,}000$. These parameter estimates minimize the sum of squares function.

6.2 Numerical Solutions to the S($\hat{\beta}$) Minimum

The problem of ARIMA parameter estimation is to discover a set of parameter estimates which minimizes the sum of squares function. Discov-

ering this set of parameter estimates is no simple task, however. The general ARIMA model is nonlinear, so calculus methods may not be used. Instead, numerical methods are required.

To illustrate the basic concepts of numerical solutions, consider the problem of discovering the square root of a positive number, A. The problem is to discover a number x such that

$$x^2 = A.$$

A numerical algorithm for this problem is:

$$x_{n+1} = 1/2(x_n + A/x_n).$$

An algorithm in this context is defined as an iterative numerical procedure; subscripts in the algorithm refer to the number of the iteration. To execute this algorithm, a "guess" for the value of x_1 is plugged into the equation to obtain a value for x_2; this value of x_2 is then plugged into the equation to obtain a value for x_3, and so forth. After n iterations, the value of x_{n+1} will be very close to the square root of A. For example, let A = 20 and let x_1 = 10. Then,

$$x_2 = 1/2(10 + 20/10) = 6$$
$$x_3 = 1/2(6 + 20/6) = 4.66667$$
$$x_4 = 1/2(4.66667 + 20/4.66667) = 4.47619$$
$$x_5 = 1/2(4.47619 + 20/4.47619) = 4.47214$$

and so forth. The change in x from iteration to iteration becomes smaller and smaller as the value of x comes closer and closer to the solution. It can be demonstrated that this algorithm *converges* to $\sqrt{A}$ that is,

$$\lim_{n\to\infty} x_n = \sqrt{A}.$$

Convergence is one of the most important properties of a numerical algorithm. If an algorithm fails to converge on the proper solution, it is practically worthless. More important, some algorithms converge at a faster rate than others. An algorithm that takes millions of iterations to converge to an acceptable solution may also be practically worthless.

This particular algorithm could obviously be continued indefinitely, with successive values of x_n approaching nearer and nearer the limit of $\sqrt{A}$. If A is not a perfect square, of course, no value of x_n will ever be exactly equal to

$\sqrt{A}$. For this reason, we require a measure of *precision*. Such a measure is ϵ which is defined as:

$$\epsilon_n = \left| x_n^2 - A \right|.$$

In this example, for successive iterations of the algorithm

n	x_n	ϵ_n
1	10.00000	80.00000
2	6.00000	16.00000
3	4.66667	1.77778
4	4.47619	.03628
5	4.47214	.00004

With only four iterations, x_n is precise to the fourth decimal place. The required degree of precision, the value of ϵ_n that is, will ordinarily be determined a priori by situational needs. When extremely precise estimates are required, the value of ϵ_n will be set arbitrarily small to reflect those requirements and the algorithm will be run until the desired value of ϵ_n is realized.

What has not been made clear is that the number of iterations required to achieve a given value of ϵ_n depends also on the *initial value*, x_1. In this case, had the initial value been x_1 = 15 instead of x_1 = 10, more iterations would have been required:

n	x_n	ϵ_n
1	15.00000	205.00000
2	8.16667	46.69444
3	5.30782	8.17295
4	4.53792	.59274
5	4.47261	.00424
6	4.47214	.00001.

And in general, the farther the initial value is from the solution, the more iterations required to achieve an estimate of a given precision.

There are actually many suitable algorithms for solving the square root of A. The algorithm demonstrated here was selected only because it illustrates the basic concepts of numerical methods, including *convergence*, *precision*, and the relationship between precision and the *initial value* of the algorithm. These same concepts apply as well to the algorithms used for ARIMA parameter estimation. Numerical solutions to the minimization of $S(\hat{\beta})$ start

with some likely guess for each ARIMA parameter. A sum of squares is evaluated conditional upon these guesses. With an appropriate algorithm, successive parameter values converge on a set of estimates which minimize the sum of squares function. The algorithm continues until a reasonable degree of precision is achieved. In most cases, parameter estimates precise to the fourth decimal place are adequate. The number of iterations required to achieve this degree of precision depends, of course, on the starting values of the algorithm, that is, on the initial guesses made for each parameter.

But unlike the simple problem of numerically solving a square root, the problem of solving the minimum $S(\hat{\beta})$ requires a relatively sophisticated algorithm. We will now describe two algorithms which have been developed for ARIMA estimation. The first, a *grid search* algorithm, was used in some of the earlier time series analysis programs such as *TMS* (Bower et al., 1974). As we will demonstrate, the grid search method is not ideally suited to ARIMA estimation. The second method we will discuss, the Marquardt algorithm has proved well suited to ARIMA estimation. Marquardt's algorithm, or variations of it, are used in most contemporary software packages. It is an extremely sophisticated method, however, so a full development would require a level of mathematical sophistication (not to mention a tolerance for esoteric detail) beyond that of most readers. Our development and description will consequently emphasize the broader conceptual and practical issues. Finally, our discussion will use examples in which the ARIMA models have only one or two parameters. The $S(\hat{\beta})$ functions for these models can be depicted graphically as two-dimensional curves or three-dimensional surfaces. Most readers will gain a better understanding of the principles of estimation when presented with graphic examples. It is a simple matter to generalize these principles to models with $n-1$ parameters and $S(\hat{\beta})$ functions in n-space, however.

6.2.1 The Grid Search Method

Returning now to the problem of finding a number x such that $x^2 = 20$, suppose that no efficient algorithm is available. A numerical solution could still be obtained (though with considerably more effort) by means of a "grid search" algorithm. To illustrate this method, let $x_1 = 6$ but now

$$x_{n+1} = x_n + \Delta.$$

In other words, the value of x increases with each iteration by a constant, Δ. Taking $\Delta = -.5$, the results of this grid search are:

n	X_n	ϵ_n
1	6.00	16.00
2	5.50	10.25
3	5.00	5.00
4	4.50	.25
5	4.00	4.00
6	3.50	7.75
7	3.00	11.00

and so forth. The smallest value of ϵ occurs when $x = 4.5$. While this estimate of $\sqrt{20}$ is relatively imprecise, it can be used as the basis for a finer search of the ϵ grid.

Figure 6.2(a) shows a plot of the precision function, ϵ, for several values of x. The precision function is a "W" with its smallest values (*minima*) at $x = \pm\sqrt{20}$. From the numerical results, it is clear that $\epsilon = 0$ somewhere in the interval

$$4.0 < x < 5.0.$$

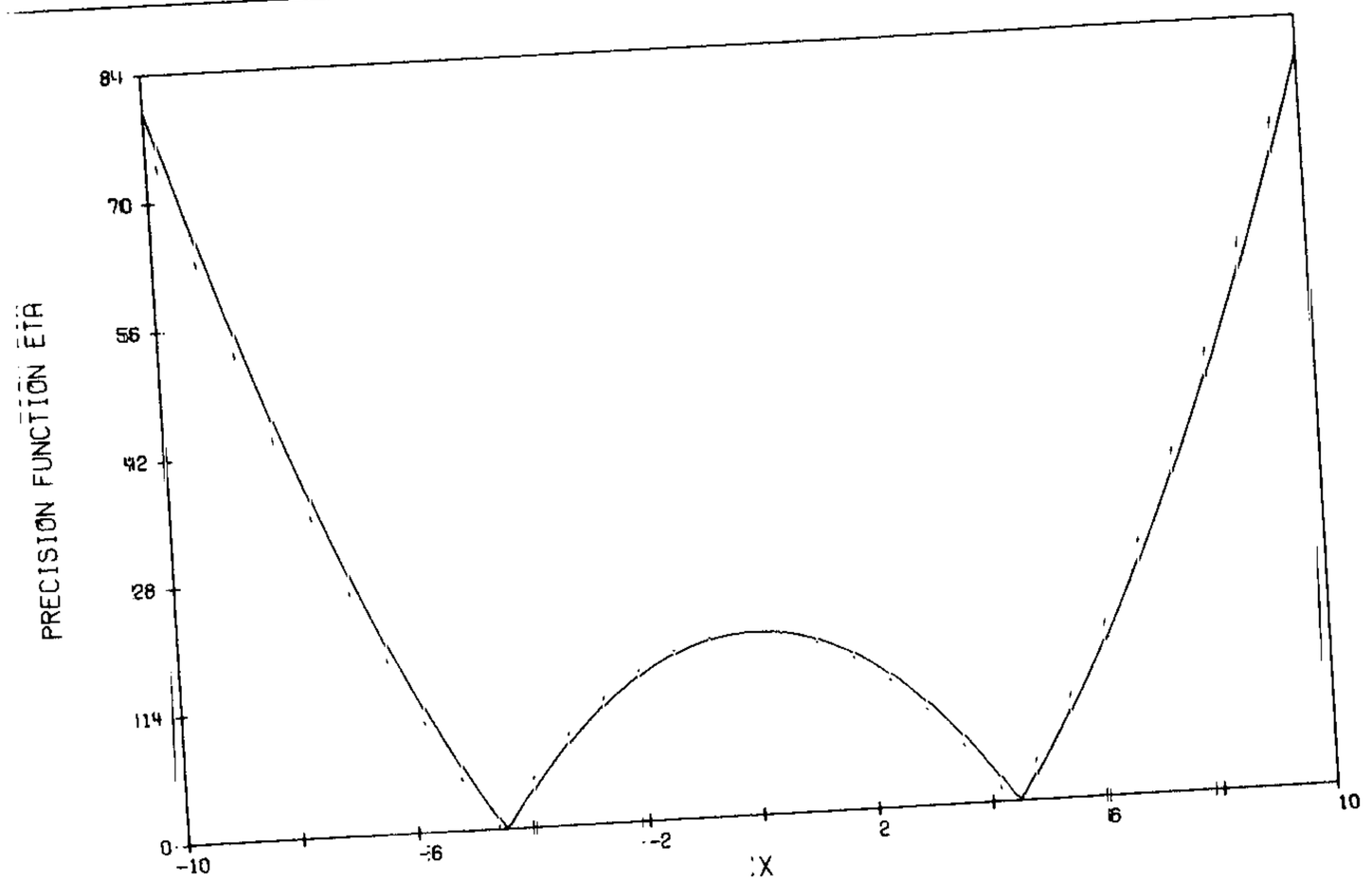

FIGURE 6.2(a) The Precision Function, ϵ

So the grid search could continue with $x_1 = 4.0$ and $\Delta = .1$:

$$x_{n+1} = x_n + .1.$$

The results of this grid search would narrow the interval around $\epsilon = 0$ considerably. By continually narrowing the interval to be searched, this crude method will yield an estimate of $\sqrt{20}$ to any desired degree of precision.

This same crude grid search method can be used to minimize a sum of squares function to obtain MLEs of ARIMA parameters. Referring to the $S(\beta)$ plot for an ARIMA (0,1,1) model of the IBM stock price time series, Figure 6.1(b), it appears that the minimum is at $\hat{\Theta}_1 \simeq 0$. We will now use a grid search to obtain a more precise estimate of Θ_1.

The grid search begins with an initial guess for the value of $\hat{\Theta}_1$. The $S(\hat{\Theta}_1)$ function is then evaluated for this guess. Say that the initial guess is $\hat{\Theta}_{11} = -.75$. The conditional sum of squares function is then evaluated as:

$$S(\hat{\Theta}_1 = -.75) = \sum_{t=1}^{N} (z_t - \hat{z}_t)^2 = \sum_{t=1}^{N} \hat{a}_t^2.$$

The random shock estimates used to evaluate the $S(\hat{\Theta}_1)$ function are computed directly from the differenced time series. The first shock, for example is computed as:

$$z_1 = \hat{a}_1 + .75\hat{a}_0,$$

$$\text{thus} \qquad \hat{a}_1 = z_1 - .75\hat{a}_0.$$

The immediate problem is that the value of $\hat{a}_0$ (an estimate of the random shock which precedes the first time series observation) is unknown. A suitable estimate is often the unconditional expected value of a_0, or zero.[2] Using $\hat{a}_0 = 0$,

$$\hat{a}_1 = z_1 - .75(0) = z_1.$$

The second random shock is then estimated as:

$$\hat{a}_2 = z_2 - .75\hat{a}_1.$$

This value of $\hat{a}_2$ is then used to compute the value of $\hat{a}_3$ and so on until all N random shocks have been estimated. The conditional sum of squares function is then evaluated as the sum of all N squared random shock estimates.

The function evaluated in this way is interpreted as the sum of squares *if* $\hat{\Theta}_{11} = -.75$.

As the grid search continues, the value of $\hat{\Theta}_1$ is increased by the constant Δ and the sum of squares function is reevaluated. Repeating this procedure several times, a sum of squares function is evaluated over a grid of predetermined $\hat{\Theta}_1$ values. Figure 6.1(b), in fact, was generated in this manner.

To efficiently employ the grid search method, the algorithm should be started with a value of $\hat{\Theta}_{11}$ as near the solution as possible. In Figure 6.1(b), for example, much effort was wasted evaluating the conditional sum of squares function for values of $\hat{\Theta}_1$ greater than .1 and less than $-.1$. In addition, the value of Δ (the increment or decrement in $\hat{\Theta}_1$) should be large enough to ensure that the solution will be discovered within a reasonable number of iterations and yet small enough to ensure that a reasonably precise estimate of the solution will be discovered.

The first of these issues is a relatively simple one to deal with in practice. The conditional sum of squares function obviously need not be evaluated for parameter values outside the bounds of stationarity-invertibility. While such values may occasionally minimize the function (usually because the ARIMA model has been incorrectly identified), they are undefined and need not be considered. More important, however, the ACF used to identify the ARIMA model will ordinarily give the analyst some information as to an appropriate set of initial values for the grid search. If nothing else, the ACF will always indicate whether parameter values are positive or negative, large or small.

Having decided upon a *range* of parameter values for the grid search, the analyst must decide upon a reasonable value of Δ. There is a trade-off here between precision and computational expense. Suppose, for example, that a solution precise to the fourth decimal place is required. For this degree of precision, searching a grid on the interval

$$-1 < \hat{\Theta}_1 < +1$$

would require (literally) hundreds of thousands of iterative evaluations. But on the other hand, if a solution precise to the second decimal place is adequate, the same grid search can be completed with only a few thousand iterative evaluations.

To balance the need for precision with the need for computional efficiency, the value of Δ can be varied. For the IBM stock price time series, we have executed a grid search across the entire bounds of invertibility with $\Delta = .1$. This grid search required only 21 evaluations of $S(\hat{\Theta}_1)$. The results,

Increment = .1		Increment = .01	
$\hat{\Theta}_1$	$S(\hat{\beta})$	$\hat{\Theta}_1$	$S(\hat{\beta})$
−1.0	1625637	−0.20	19483.3
−0.9	116298	−0.19	19437.3
−0.8	52987	−0.18	19395.8
−0.7	35519	−0.17	19358.8
−0.6	27948	−0.16	19326.3
−0.5	23929	−0.15	19298.2
−0.4	21595	−0.14	19274.2
−0.3	20222	−0.13	19254.6
−0.2	19483	−0.12	19239.0
−0.1	19220	−0.11	19227.6
0.0	19363	−0.10	19220.2
0.1	19896	−0.09	19216.8
0.2	20851	−0.08	19217.5
0.3	22315	−0.07	19222.0
0.4	24471	−0.06	19230.5
0.5	27694	−0.05	19242.9
0.6	32818	−0.04	19259.2
0.7	42017	−0.03	19279.3
0.8	62856	−0.02	19303.4
0.9	141803	−0.01	19331.3
1.0	2736045	0.00	19363.0

TABLE 6.2 Grid Search for $\hat{\Theta}_1$

listed in Table 6.2 and plotted in Figure 6.1(b), indicate that the solution lies in the interval

$$-.2 < \hat{\Theta}_1 < 0.0.$$

A grid search of this interval with $\Delta = .01$ also requires only 21 evaluations of $S(\hat{\Theta}_1)$. These results, also listed in Table 6.2, indicate that the solution lies in the interval

$$-.1 < \hat{\Theta}_1 < -.08.$$

If greater precision is required, this interval can be searched with $\Delta = .001$. As a general rule, relatively large values of Δ can be used at the start of the grid search to narrow the range of the search. Relatively small values of Δ can then be used to obtain precise estimates of the solution.

Although grid search methods are conceptually simple and easily executed, they are extremely costly. For the IBM stock price time series, a reasonably precise estimate of Θ_1 was discovered with fewer than 100 evaluations of the $S(\hat{\Theta}_1)$ function. Given access to a computer, these calculations present no real problem. As the number of parameters in the model increases, however, the number of evaluations of $S(\hat{\beta})$ required for a grid search increases exponentially in the parameter space. If a model has only one parameter, and if 100 evaluations gives reasonably precise estimates of the parameter, a grid search is practical. If the model has two parameters, however, the grid search will require 10,000 evaluations (100^2) to obtain parameter estimates of equal precision. And if the model has four parameters, which is not at all uncommon in social science applications, the grid search will require 100,000,000 evaluations of $S(\hat{\beta})$. Executing the grid search in stages, each with a smaller value of Δ as in this example, would reduce the number of evaluations required to, say, several hundred thousand. But even this number of evaluations would be prohibitively expensive.

6.2.2 Marquardt's Algorithm

While useful as a pedagogical tool, grid search methods of estimating ARIMA parameters are virtually worthless in practice. To be sure, any method will require the evaluation of $S(\hat{\beta})$ at a number of points and, in this sense, all methods are similar to the grid search method. To be practically useful, however, a method must have some decision rule to limit the number of points requiring evaluation. An ideal method will superficially scan the $S(\hat{\beta})$ function, locate the general area of the minimum, and then will *converge* on the minimum in as few steps as possible. When the minimum is relatively far away, the method will take large steps, but as the minimum draws nearer, steps will become smaller. In this way, the ideal method will balance the need for precision with the costs of computation.

Marquardt (1963) has proposed an algorithm which, in this sense, is nearly ideal for the estimation of ARIMA parameters. While there are other algorithms which are also nearly ideal, they differ from Marquardt's algorithm only in minor details. The analyst typically encounters a nonlinear estimation algorithm as a "black box" hidden inside the time series analysis computer program. The black box usually performs adequately so long as the analyst supplies appropriate inputs to the program and so long as outputs are intelligently interpreted. To do this, the analyst need only have a general idea of the inner workings of the black box. The esoteric details can be left to computer programmers and numerical analysts. Our development and discussion of Marquardt's algorithm will thus focus on general principles and

practical issues. Those readers who are interested in the detailed workings of the black box are directed to Box and Jenkins (1976: Chapter 7), Makridakis and Wheelwright (1978: Chapter 26), Draper and Smith (1966: Chapter 10) or Nelson (1973: Chapter 5.8) for treatments of these issues in varying detail, orientation, and contexts. Although our discussion will deal only with one version of the Marquardt algorithm, the terminology and principles we develop will be general to nearly all time series software.

To grasp the logic of Marquardt's algorithm, one must at least understand the basic principles of calculus minimization. We have introduced some of these principles in our earlier discussion but we must now give a more explicit introduction. The efficiency of a grid search can be increased substantially by increasing or decreasing the size of a step across the grid (increasing or decreasing the value of Δ, that is) depending upon how near or far away the solution is. An obvious decision rule for optimizing step size involves the derivatives of $S(\hat{\beta})$. If these derivatives were linear, of course, they could be solved to obtain MLEs of the ARIMA parameters. A numerical solution to the minimum $S(\hat{\beta})$ is required *because* these derivatives are either *non*linear, unavailable, or intractable.

But numerical estimates of the derivatives are always available. To illustrate just how these derivatives can be used to optimize step size, consider again the problem of estimating the value of $\sqrt{20}$ through a grid search. Denoting the *size* of the n_{th} step by Δx where

$$\Delta x = x_n - x_{n-1},$$

the corresponding change in the precision function is:

$$\Delta\epsilon = \epsilon_n - \epsilon_{n-1}.$$

Invoking the Mean Value Theorem of differential calculus, it can be demonstrated that the derivative of the precision function at some point between x_{n-1} and x_n is given by the ratio of $\Delta\epsilon$ to Δx. Of course, when the difference between x_{n-1} and x_n is made infinitesimally small, its limit *is* the derivative, that is,

$$\lim_{\Delta x \to 0} \frac{\Delta\epsilon}{\Delta x} = \frac{\partial\epsilon}{\partial x}$$

Now let $x_1 = 7$ and let $\Delta x = .5$. The grid search then has the result:

n	x_n	ϵ_n	$\Delta\epsilon/\Delta x$
1	7.00	29.00	
2	6.50	22.25	13.50
3	6.00	16.00	12.50
4	5.50	10.25	11.50
5	5.00	5.00	10.50
6	4.50	.25	9.50
7	4.00	4.00	− 7.50 .

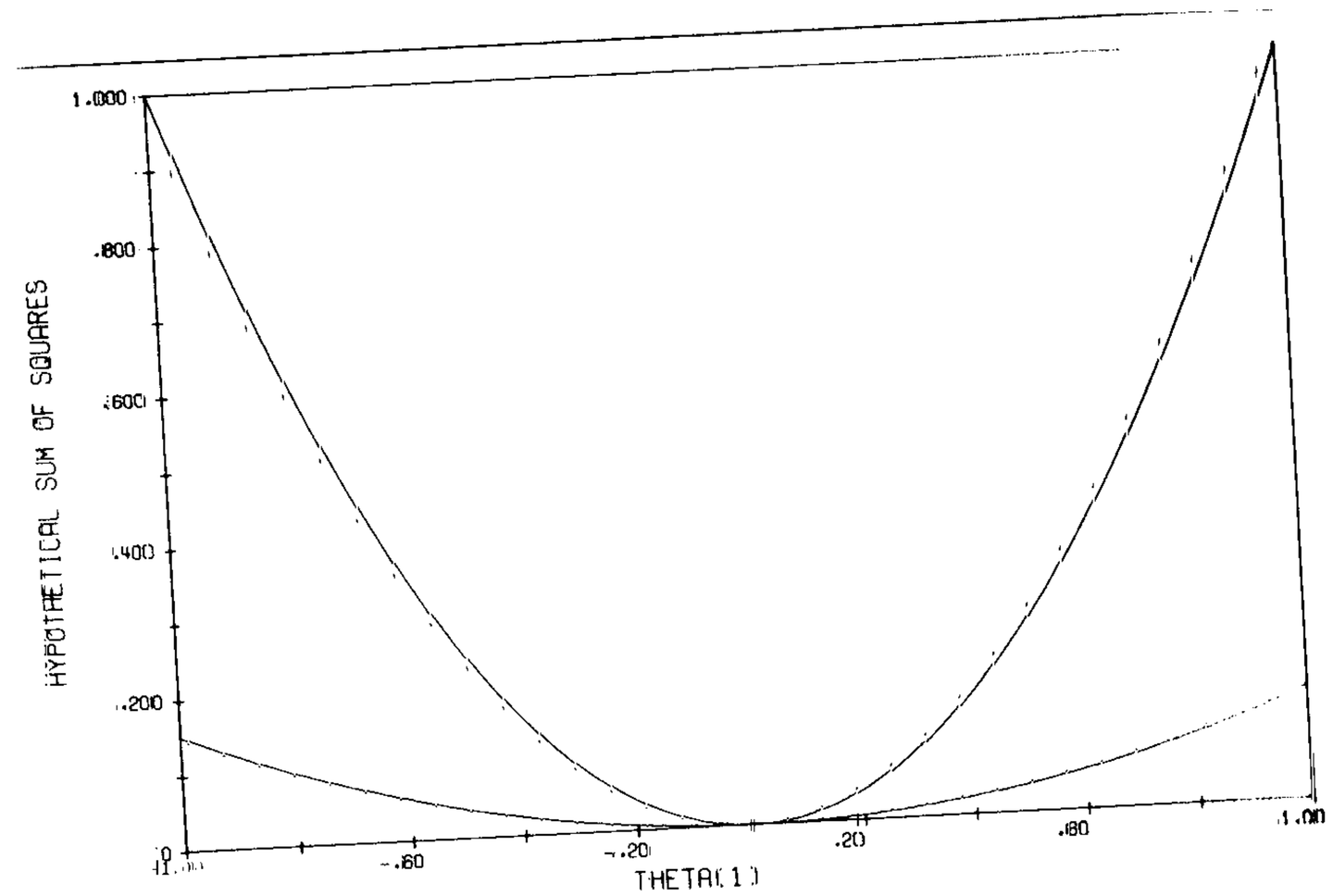

FIGURE 6.2(b) Two Hypothetical Sum of Squares Functions Differing Only in Steepness

The numerical derivative, $\Delta\epsilon/\Delta x$, grows smaller as x approaches the minimum of the precision function; and as x *passes* the minimum, the numerical derivative changes its sign from positive to negative. The derivative is interpreted literally as the slope of a straight line drawn tangent to the precision function at a point between x_{n-1} and x. The numerical derivative thus tells one how *steep* the precision function is at that point. To optimize step size in the grid search, an algorithm can make the size of each step proportional to the steepness of the function, that is, proportional to the numerical derivative. When the function is steep, as it always will be at points far from the minimum, the numerical derivative will be large and positive. Steps should thus be large and positive. As the function becomes

less steep, as it always will be at points near the solution, the numerical derivative will become small and step sizes may be made smaller. Finally, if the numerical derivative changes its sign, the search can stop for this indicates that the solution has been passed.

This method of optimizing the size of a grid search step is called (loosely) *the method of steepest descent*. An obvious shortcoming of this method is that, for the algorithm to be efficient, the $S(\hat{\beta})$ function must be relatively steep at points far away from the solution. Figure 6.2(b) shows two hypothetical functions which vary in steepness. For one of the functions, a steepest descent algorithm may or may not prove efficient, while for the other function, a steepest descent algorithm will probably be no more efficient than a simple grid search.

Strictly speaking, Marquardt's algorithm is a compromise between the steepest descent method and the *Gauss-Newton (iterative linearization)* method. The Gauss-Newton method solves a linear approximation to the nonlinear ARIMA model. Although the general ARIMA model itself is nonlinear, it can be expanded as a Taylor series around some initial guess for the parameters. Representing the general ARIMA model as

$$Y_t = a_t + f(\hat{\beta}, Y_{t-k}),$$

in which $f(\hat{\beta}, Y_{t-k})$ is a function of past observations of the time series and the model parameter estimates and where this function is nonlinear in the parameters, a Taylor series expansion is:

$$Y_t = a_t + f(\hat{\beta}_1, Y_{t-k}) + (\hat{\beta} - \hat{\beta}_1)\left[\frac{\partial f(\hat{\beta}_1, Y_{t-k})}{\partial \hat{\beta}_1}\right] + \ldots,$$

where $\hat{\beta}_1$ is an initial guess for $\hat{\beta}$. The Taylor series is an infinite expansion, of course, but only the first two terms of this expansion are required for the Gauss-Newton algorithm. The truncated Taylor series can be written as

$$Y_t - f(\hat{\beta}_1, Y_{t-k}) = (\hat{\beta} - \hat{\beta}_1)\left[\frac{\partial f(\hat{\beta}_1, Y_{t-k})}{\partial \hat{\beta}_1}\right] + a_t.$$

The quantity on the left-hand side of the operation is simply the difference between the time series observation Y_t and its predicted value given the parameter estimates $\hat{\beta}_1$. The derivative on the right-hand side of the equation is evaluated as a number for each time series observation, Y_t. An ARIMA (0,0,1) model, for example, has the derivative

$$\frac{\partial f(\hat{\beta}_1, Y_{t-k})}{\partial \hat{\beta}_1} = k \sum_{k=0}^{\infty} \hat{\beta}_1^{k-1} Y_{t-k},$$

where $\hat{\beta}_1$ is an initial guess for the parameter Θ_1. Although this derivative is nonlinear, and thus cannot be solved analytically, it can be *evaluated*. Denoting the evaluated derivative at Y_t by c_t, the linear approximation can be written as

$$\hat{a}_t = (\hat{\beta} - \hat{\beta}_1)c_t + a_t,$$

which is a linear model. An estimate of the quantity $(\hat{\beta} - \hat{\beta}_1)$ can be obtained with an OLS solution to the linear approximation. Once obtained, the estimate is interpreted as the *distance* between the MLE of the parameters and the initial guess at those parameters. If this distance is denoted by Δ_1 where

$$\Delta_1 = \hat{\beta} - \hat{\beta}_1,$$

then a second guess for the parameter estimates is:

$$\hat{\beta}_2 = \hat{\beta}_1 + \Delta_1.$$

The nonlinear function and the derivative are then evaluated for this set of parameter estimates, that is, the linear approximation

$$\hat{a}_t = (\hat{\beta} - \hat{\beta}_2)c_t + a_t$$

is solved to obtain a value for Δ_2. The Gauss-Newton algorithm continues iteratively until it converges, that is, until

$$\Delta_n = \epsilon,$$

where ϵ is a specified degree of precision.

Like the steepest descent method, the Gauss-Newton method is quite efficient for some types of nonlinear functions and some estimation situations. But the Gauss-Newton method, too, may be inefficient for other types of functions and other situations. For example, the Gauss-Newton algorithm may oscillate, producing positive and negative values of Δ_n before converging slowly to an acceptably precise set of parameter estimates. More important, since the Gauss-Newton algorithm solves a linear approximation to the nonlinear ARIMA model, any given iteration may reduce the linear sum of squares while increasing the nonlinear sum of squares.

In short, neither the method of steepest descent nor the Gauss-Newton method is ideal for ARIMA parameter estimation. The major practical dif-

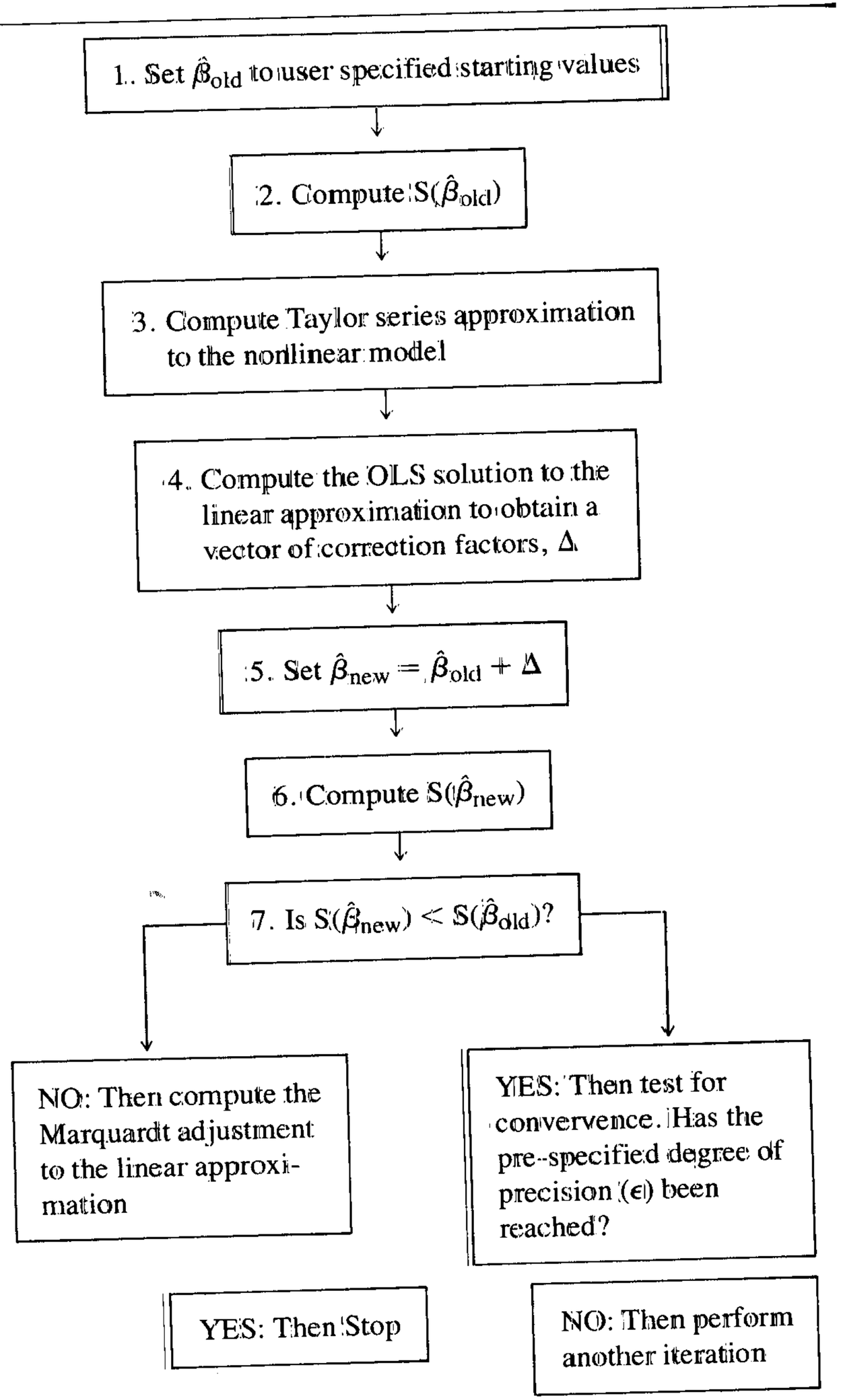

FIGURE 6.2(c) A Flowchart Diagram of Marquardt's Algorithm

ference between these two methods is the manner in which a step size (Δ) is selected for each iteration. To correct the deficiencies of the Gauss-Newton method, Marquardt incorporated several features of the steepest descent method into the algorithm. The result, often referred to as "Marquardt's compromise," combines the strongest features of each method.

Figure 6.2(c) presents the general principles of Marquardt's algorithm in a flow-chart format. The algorithm begins with a set of initial guesses, or starting values, for each parameter of the model. A truncated Taylor series expansion around these initial values is used as a linear approximation to the nonlinear model. The linear approximation is solved by OLS regression to obtain a vector of correction factors, Δ. A new set of parameter estimates is formed by adding the correction factors to the old parameters estimates, that is,

$$\hat{\beta}_{new} = \hat{\beta}_{old} + \Delta.$$

The sum of squares function is then evaluated for the new parameter estimates. If the new sum of squares is smaller than the old sum of squares, the new parameter estimates are substituted for the old. Convergence is tested, and if the specified degree of precision has not been achieved, another iteration is executed.

As described here, and as diagramed in Figure 6.2(c), the Marquardt algorithm is almost identical with the Gauss-Newton algorithm. Indeed, the two methods are identical *so long as each iteration results in a reduced sum of squares*. When an iteration does *not* result in a reduction of the sum of squares, that is, when

$$S(\hat{\beta}_{old}) < S(\hat{\beta}_{new}),$$

the iteration does not end. Instead, the linear approximation to the nonlinear function is adjusted and a new vector of correction factors is estimated. This adjustment is too complicated (and has too many variations) to be described in detail here. However, it is essentially an interpolation between the Gauss-Newton and steepest descent correction factors. This adjustment procedure is often called the "test-point" solution. Depending upon the shape of the nonlinear $S(\hat{\beta})$ function, the adjustment procedure may be repeated a number of times until a set of parameter values is found which reduces the sum of squares. Only then is the iteration completed, the convergence tests executed, and a new iteration (possibly) begun.

Figure 6.2(d) shows a contour plot of the sum of squares function for the Sutter County Workforce series analyzed in Section 2.12.1. The ARIMA

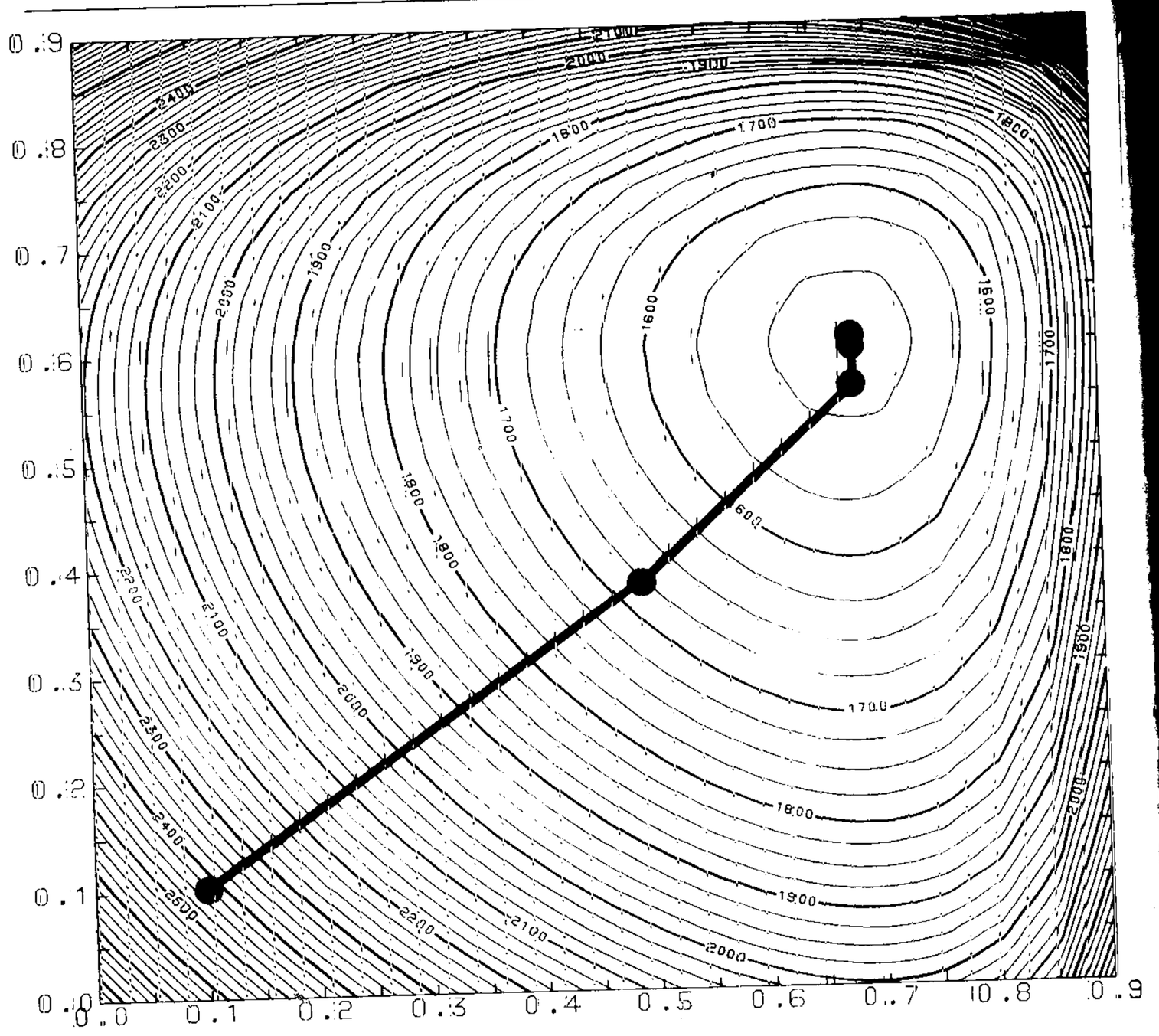

FIGURE 6.2(d) $S(\hat{\beta})$ for the Sutter County Workforce Example of Section 2.12.1: Marquardt's Algorithm "Finds" the Minimum in Four Steps

$(0,1,1)(0,1,1)_{12}$ model on which this plot is based, gives ranges for the two moving average parameters along its x- and y-axes. Superimposed on this plot is the path taken by Marquardt's algorithm to obtain a solution precise to the third decimal place. Note that the steps taken by the algorithm in its first and second iterations are relatively large while the steps taken in its third and fourth iterations are relatively small. The size of each step is roughly proportional to the steepness of the function at that point. The statistics for each of the four iterations are:

Iteration	$\hat{\Theta}_1$	$\hat{\Theta}_{12}$	$S(\hat{\Theta}_{11}) \times 10^{-4}$
Starting Values	.1000	.1000	2430.0
1	.3828	.4856	1678.0
2	.5590	.6839	1542.0
3	.5970	.6812	1539.0
4	.6031	.6800	1539.0 .

The algorithm reached this solution quickly. More complicated models and models whose sum of squares functions are not so ideally shaped (symmetrical and steep with a well-defined minimum, that is) might have required many more iterations before achieving a solution of this precision. The algorithm stopped in this case because the relative change in the sum of squares function from the third to fourth iterations was less than the specified degree of precision, $\epsilon = .001$. Had we required a greater degree of precision, say $\epsilon = .0001$ for a relative change in the sum of squares, the algorithm would not have stopped after the fourth iteration. And of course, had we specified a more realistic set of starting values, say $\hat{\Theta}_1 = \hat{\Theta}_{12} = .5$, convergence would have been achieved in fewer than four iterations.

6.3 Monitoring and Controlling the Algorithm

In practice, Marquardt's algorithm converges to a reasonably precise set of ARIMA parameter estimates within relatively few iterations, usually 5 to 15. The grid search, steepest descent, and Gauss-Newton methods, in contrast, may require hundreds or even thousands of iterations to achieve a solution of the same precision. From time to time, however, Marquardt's algorithm may iterate 50 or 100 times. Computational costs not withstanding, the parameter estimates obtained in such a case are generally suspect and certainly are of a lower quality than parameter estimates ordinarily must be. Most software packages print out the details of estimation. By monitoring these details, the analyst can often spot statistical anomalies and make adjustments to guarantee the quality of parameter estimates.

Fortunately, the rate at which Marquardt's algorithm converges (and hence, whether it will converge at all) depends on a few key factors which, for the most part, are controlled by the analyst. These factors include (1) the degree of precision specified by the analyst, (2) the quality of the starting values supplied to the algorithm by the analyst, and (3) the empirical adequacy of the ARIMA model identified by the analyst. The key here is the analyst. The parameter estimates given by the algorithm may often be no

better than the information given the algorithm by the analyst.

Of these three factors, the least important is the degree of precision specified. Most computer programs stop iterating when any of four stopping criteria is satisfied: (1) a specified maximum number of iterations (say 50) have been executed; (2) the last iteration has resulted in a minimal relative change in the sum of squares evaluation, that is,

$$\left| \frac{S(\hat{\beta}_{new}) - S(\hat{\beta}_{old})}{S(\hat{\beta}_{old})} \right| \leq \epsilon$$

(3) the last iteration has resulted in a minimal relative change in the parameter estimates, that is,

$$\left| \frac{\hat{\beta}_{new} - \hat{\beta}_{old}}{\hat{\beta}_{old}} \right| \leq \epsilon$$

or (4) when the last iteration has resulted in a minimal ratio of the initial sum of squares to the final sum of squares.

Relaxing the tolerances of any stopping criterion will obviously result in a more rapid convergence to a less precise solution. Conversely, tightening any of the stopping criteria will result in a slower convergence to a more precise solution. Most computer programs provide default stopping criteria which ensure an adequate balance between precision and efficiency. *The analyst should always know exactly what these default stopping criteria are. More important, on any estimation, the analyst should always know why the algorithm stopped.* If the algorithm stops because it has converged on a solution, there is no problem. It may happen, however, that the algorithm did not converge, but rather stopped because a specified maximum number of iterations had been executed. The parameter estimates printed out in this situation would be of extremely poor quality. Beyond this, however, precision in effect is a given. It has been determined by the analyst a priori and should not be changed just to make the algorithm converge more quickly.

The starting values for parameter estimates are another matter. It should be intuitively clear that the rate of convergence will be optimal when the starting values supplied by the analyst are near the solution. What is not intuitively clear is that faulty starting values may lead to faulty parameter estimates. Although information from model identification may be used to estimate starting values, we have found that almost any naive value for the ϕ_p, ϕ_p, Θ_q, Θ_Q, and δ_r parameters of a general model will result in rapid

convergence. Of course, the values supplied should always lie within the bounds of stationarity, invertibility, and stability.

Specification of a starting value for the ω_s parameter of a general model is more problematic. The ω-parameter is analogous to the unstandardized slope parameter of a linear regression model: both are expressed in the metric of the independent variable. To ensure convergence, the starting value should be as close as possible to the final estimate. At the very least, the starting value should be of the same order of magnitude as the final estimate. In the Directory Assistance analysis of Section 3.1, for example, the parameter $\hat{\omega}_0$ is approximately -40,000. The sum of squares function for this estimation is plotted in Figure 6.1(c). Examining that plot, the reader may see how crucial a realistic starting value for $\hat{\omega}_0$ is. An unrealistic starting value would result in an extremely large initial sum of squares which would cause many problems for the algorithm. On the other hand, any starting value of the same magnitude (for example, $-10{,}000$) will be adequate. When in doubt, the analyst may use OLS estimates of the ω-parameters as starting values.

Many software packages automatically compute adequate starting values for most analyses. Nonetheless, the analyst should remain sensitive to this issue and should make it a point to know the heuristic mechanisms used in the computer program. The program should display the starting values and these should be routinely inspected and compared to the final parameter estimates. An initial sum of squares, computed from the starting values, that is highly divergent from the sum of squares on the first iteration may indicate that one or more of the starting values is inappropriate.

Finally, the most crucial information supplied to the algorithm by the analyst is the ARIMA model. If the model is inappropriate, then the sum of squares function may be ill-formed with no solution clearly defined. Although a solution may be defined in a technical sense, the $S(\hat{\beta})$ function may be "flat" or "defective," in which case the algorithm may converge slowly or not at all and parameter estimates will be unstable.

The most graphic illustration of this problem is the case of parameter redundancy. As noted in Section 2.7, ARIMA (p,0,q) models may be reduced to simpler ARIMA (p,0,0), ARIMA (0,0,q), or ARIMA (0,0,0) models for certain values of the ϕ- and Θ-parameters. When parameters are redundant, the $S(\hat{\beta})$ function may not have one clearly defined minimim, but rather may have several minima, each associated with a particular configuration of the redundant parameters. When the algorithm attempts to solve a function of this sort, it may oscillate between the several minima without ever converging. The most common case of parameter redundancy may be

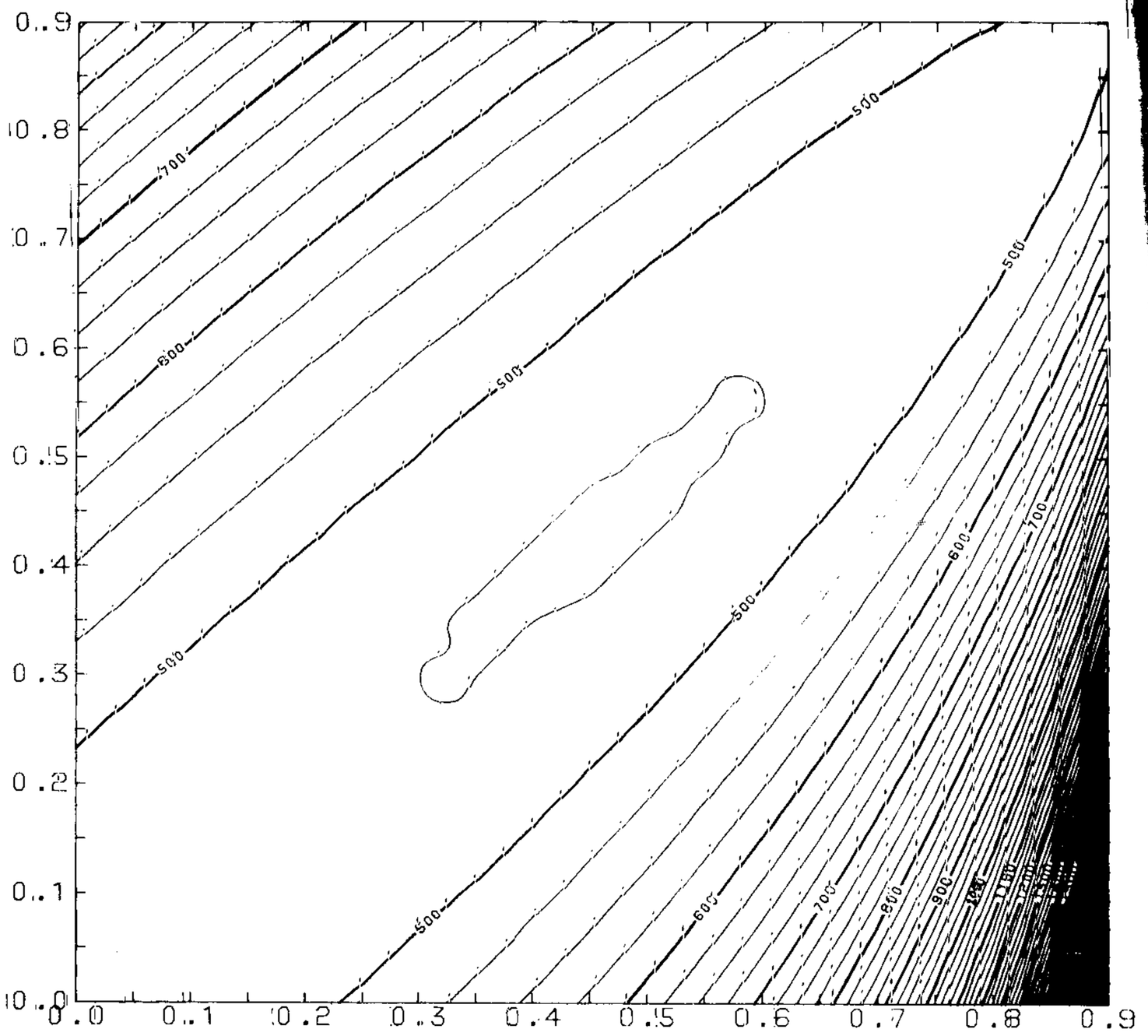

FIGURE 6.3 $S(\hat{\beta})$ for an ARIMA (1,0,1) Process, $\phi_1 \approx \theta_1$

the ARIMA (1,0,1) model in which $\phi_1 \simeq \Theta_1$. Figure 6.3 shows a contour plot of the sum of squares function for the model

$$(1 - .5B)y_t = (1 - .5B)a_t.$$

Note that the minimum of this function is not a clearly defined point, but rather is a long valley extending along the axis of $\hat{\phi}_1 = \hat{\Theta}_1$. Since the two parameters are redundant, many values of $\hat{\phi}_1$ and $\hat{\Theta}_1$ will minimize this function. A *unique* solution is undefined. Marquardt's algorithm in this case will produce a variety of solutions depending upon the starting values supplied it. If the algorithm is given $\hat{\phi}_1 \simeq \hat{\Theta}_1 \simeq .4$ for starting values, for example, it will produce different final estimates than if it is given $\hat{\phi}_1 \simeq \hat{\Theta}_1 \simeq .6$ for starting values. If this point is unclear, the reader should compare

this sum of squares plot with the one shown in Figure 6.2(d). That sum of squares function is nearly ideal for the same reasons that this one is not.

Multicollinearity is a related issue. Because Marquardt's algorithm solves a linear approximation to the nonlinear function, time series programs print out many of the statistics usually associated with linear regression models. For example, most time series programs print out a correlation matrix of parameter estimates. This matrix should always be inspected as a quality control check on parameter estimates. If two parameters are highly correlated (say at the .7 level or higher), an inappropriately specified model may be indicated. When ARIMA parameters are highly correlated, the $S(\hat{\beta})$ function will have a shape similar to the one shown in Figure 6.3(a); that is, the minimum may be a long, shallow valley rather than a steep hole.

Given the wide range of models in the general ARIMA class, it is difficult to generalize about the effects of an inappropriately specified model on estimation. Fortunately, the empirical model-building strategies we developed in Chapters 2, 3, and 5 will nearly always lead to the identification of a parsimonious, statistically adequate model. If these strategies are followed routinely, and if common sense is exercised, it is unlikely that the analyst will encounter estimation problems due to model inappropriateness. The analyst nonetheless should be sensitive to this possibility.

6.4 Time Series Software

As the use of time series analysis in social research proliferates, so does the development of time series software. Computer programs for time series analysis (especially for managerial forecasting applications) have long been available commercially. It is only recently, however, that this software has become available to the academic community. When we started work on this volume in 1976, only one generalized Box-Jenkins program was widely available in academic computing centers. Only two years later, several programs (including the major statistical packages such as SPSS, BMDP, and SAS) either have Box-Jenkins capabilities or are planning to incorporate Box-Jenkins capabilities in the near future.

In this section, we will briefly review the computational requirements of Box-Jenkins time series analysis and then describe several available programs. Our intent is *not* to recommend a specific program, but rather to point out issues and discuss the manner in which these issues are handled by several programs. Our discussion of software will avoid the specific details of various programs. Development in this field is proceeding rapidly, so any description of the current state of the art will soon be outdated. Although many commercial time-sharing firms provide for ARIMA modeling, we

have restricted our discussion to software currently available to acade computing facilities. In an appendix to this chapter, we give the names a addresses of several software suppliers. The reader is urged to contact the organizations directly for a description of current offerings. This list ha been compiled from our personal knowledge, so the inclusion or omission o any program implies nothing about its quality or availability. Finally, th American Statistical Association regularly evaluates statistical software and we suggest that the reader consult these reports.

Speaking generally, computer programs for Box-Jenkins time series analysis are remarkably similar. Throughout this volume, we have referred to the calculation and display of certain statistics, plots, and so forth, and most Box-Jenkins programs provide all of these capabilities. The only major differences among programs tend to be in the formats of input specifications, the presentation of output, and the algorithm used for nonlinear estimation. The model-building strategies presented in earlier chapters always consisted of an identification-estimation-diagnosis sequence and, not surprisingly, most ARIMA time series programs are organized around these functions. Although these three steps are usually executed by the analyst in sequence, there are many interpretive decisions to be made at each step. We have found that flexible, interactive software which facilitates movement within and between these three steps to be optimal. Batch systems are nevertheless adequate and seem to be more widely available at this time.

In discussing time series software, it is useful to distinguish between three different levels of software organization. These are (1) a subroutine library, (2) a "stand-alone" statistical program, and (3) a statistical system. Time series software, generally written in a high-level scientific language such as FORTRAN, is available in each of these forms for computers of varying size and manufacture.

Subroutine Libraries

A subroutine library usually consists of a series of subroutines, each of which performs a discrete and distinct statistical function. One subroutine, for example, might compute the ACF and PACF while another subroutine might plot the ACF and PACF. To use a subroutine library, the user must write a "main program" which reads in the data, calls the appropriate subroutines, stores the intermediate results, and prints the desired output. Subroutine libraries offer great flexibility at the cost of detailed specification and programming.

The International Mathematical and Statistical Library (1979) is a large comprehensive subroutine library available at most research computing cen-

ters. It contains a variety of routines for the identification, estimation, and forecasting of univariate and single-input transfer function ARIMA models. It also contains a useful routine for generating ARIMA time series with known parameters to be used in testing and simulation.

Stand-Alone Statistical Programs

A stand-alone statistical program usually consists of a main program and its associated subroutines. It is designed to accept input from the user in a certain format, perform the necessary calculations, and print the results.

The earliest program designed specifically for interrupted time series analysis was TMS (Bower et al., 1974). This program implemented the methodology developed by Box and Tiao (1965) and elaborated by Glass et al., (1975). While a pioneer in its day, TMS is now generally considered to be obsolete. It is restricted to nonseasonal ARIMA models for the most part; it is restricted to a narrow subset of intervention models; furthermore, its use of a grid search estimation method makes it prohibitively expensive when compared to programs using a variation of the Marquardt algorithm.

In 1972, David J. Pack developed a subroutine library specifically designed for ARIMA analyses; and in intervening years, he has consistently expanded its scope. The most recent version of this program (Pack, 1977) contains two main programs which read input parameters and call appropriate subroutines for computation and display of results. The inclusion of these main programs places the Pack software more properly in the stand-alone program category. We have found these systems to be extremely flexible, easily transportable, and available at a modest cost. The major drawback to the Pack programs is that specification of the input parameters is in fixed fields and can be tedious and confusing to the inexperienced user. Moreover, the documentation is terse and assumes a thorough knowledge of ARIMA models and modeling procedures.

Batch Statistical Systems

A statistical system can be thought of as a series of statistical programs, each performing a certain kind of analysis. The series of programs are linked by a common set of data manipulation capabilities and a unified, "natural" user's input language. From the researcher's point of view, statistical systems are often the easiest to use; but they are also relatively costly to acquire and maintain.

The year 1979 was marked by the introduction of Box-Jenkins capabilities to three major academic and research statistical systems: SPSS (Statistical Package for the Social Sciences), BMDP (Biomedical Computer Pro-

grams), and SAS (Statistical Analysis System). The inclusion of ARIMA statistical procedures in these widely distributed and well-documented systems should greatly facilitate the use of time series analysis by social scientists. Each of these packages features extensive file definition and data management capabilities combined with a keyword syntax for input specifications. In SPSS and SAS, the Box-Jenkins routines are integrated into the unified system whereas in BMDP, they compromise a separate program (BMDP2T) which shares syntax and file structure with other BMDP series programs. Each of these packages supports univariate Box-Jenkins analyses. BMDP2T also supports intervention and transfer function analyses and can be used interactively (see below). SPSS is currently developing transfer function capabilities and SAS is considering them.

Interactive Statistical Systems

The iterative nature of Box-Jenkins time series analysis immediately suggests the desirability of interactive data analysis. In an interactive environment, the researcher ordinarily "converses" with a statistical system by means of a typewriter-like computer terminal. A command is entered from the terminal keyboard, the desired calculations are performed, and the results are immediately displayed and/or printed at the terminal console. The analyst can then enter a new command based on interpretation of the information just received. For example, in the present context, the analyst might estimate a model, perform diagnostic checking on the residuals, and then immediately estimate a revised model based on the diagnostic information just displayed. A procedure of this sort would usually take at least two separate runs using a batch program. Interactive or conversational analysis is also attractive in that input errors can be easily detected by the system and reported to the user for immediate correction. Since communication between the computer and a terminal is usually over regular phone lines, researchers may utilize interactive software, often unavailable locally, from a distant computer facility for a relatively nominal fee.

As we have mentioned, many commercial time-sharing firms offer Box-Jenkins capabilities, but the cost of these services is often prohibitive for academic research and instruction. Network linkages between university computing centers are becoming more popular and one or more members of the EDUNET network may offer ARIMA analysis programs. At present, interactive statistical systems are not distributed as widely as batch systems. Again, development is proceeding rapidly in this field and such systems will probably be more readily available in the future. BMDP2T, described above, also may be operated in an interactive mode. MINITAB is a flexible, general interactive statistical system for research and instruction using small data sets. It can be easily adapted for different sizes and models of computers and presently supports univariate ARIMA analysis. Intervention analysis capabilities are currently under development. SYBIL/RUNNER is an interactive system oriented primarily toward applied managerial forecasting. It includes ARIMA univariate and transfer function capabilities in its extensive forecasting repertoire. SCSS, the SPSS Conversational Statistical System, is a very powerful interactive system that runs on several different models of computers. SCSS does not currently support ARIMA time series analysis but these capabilities are being considered for the future. Finally, SCRUNCH, an interactive impact assessment system developed for the CDC 6000 computer by Richard A. Hay, Jr., was used to conduct many of the analyses for this volume.

For Further Reading

Almost all discussions of nonlinear estimation require a background in calculus. Draper and Smith (1966: Chapter 10) present a solid introduction which is as accessible as possible to the nonmathematician. Pindyck and Rubinfeld (1976: Chapter 15), Nelson (1973: Chapter 5.8), or Granger and Newbold (1977: Chapter 3.5) give briefer developments but at more or less the same level of sophistication. Makridakis and Wheelwright (1978: Chapter 26) work through the arithmetic of a Marquardt solution. This material is perhaps the most accessible treatment of those discussed here.

NOTES TO CHAPTER 6

1. The reader who is familiar with Box and Jenkins (1970: Chapter 7) will realize that the *conditional* sum of squares function is also conditional upon the starting shocks of the estimation, that is, on the values of $a_0, a_{-1}, \ldots, a_{-t}$, which precede the first observation of the time series. We will return to this point later.

2. Cf. Note 1. As noted, the conditional sum of squares function is conditional upon a vector of parameter estimates, $\hat{\beta}$, and also upon a set of initial shocks, $a_0, a_{-1}, \ldots, a_{-t}$. In this case, the $S(\hat{\beta})$ function requires an estimate of only one initial shock to be evaluated: $\hat{a}_0$. An ARIMA (0,0,2) model would require values for $\hat{a}_0$ and $\hat{a}_{-1}$ and an ARIMA (0,0,q) model would require values for $\hat{a}_0, \hat{a}_{-1}, \ldots, \hat{a}_{1-q}$. For *nonseasonal* series of moderate length, say 100 observations or more, the *un*conditional expectation of these shocks (zero) should prove sufficient. For *seasonal* models of any length, however, Box and Jenkins (1976: Chapter 7) recommend that the *conditional* expectation of these shocks be used. The conditional expectation is derived through a recursive procedure known as backforecasting or backcasting. In essence, backcasting is executed by "running" the ARIMA model backward in time to generate forecasts of a_0,