John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

NONLINEAR PARAMETER ESTIMATION:
an Integrated System in BASIC

ISBN 0-88769-009-2

Originally published in 1987 by Marcel Dekker Inc., New York,

under ISBN 0-8247-7819-7

This edition of **Nonlinear Parameter Estimation: an Integrated System in BASIC** is made available electronically in order to provide the material to interested readers. The text has been kept as close to the original as possible, except for correction of known errors and minor adjustments to fit the dual-page format chosen to minimize paper use.

We can be contacted as follows:

Electronic mail to Mary Nash: mnash _at_ nashinfo.com

Telephone: (613) 236 6108

[We do NOT return long-distance calls nor accept collect calls except where there are prior arrangements.]

Electronic mail is our preferred method of communication.

John C. Nash, April 2004

TABLE OF CONTENTS

PREFACE

This book and software collection is intended to help scientists, engineers and statisticians in their work. We have collected various software tools for nonlinear parameter estimation, along with representative example problems, and provided sufficient "glue" in the form of procedures, documentation, and auxiliary program code to allow for relatively easy use of the software system for nonlinear parameter estimation.

We use the expression "relatively easy use", because nonlinear parameter estimation has many traps for the unwary. The problems are, in general, an order of magnitude more difficult or effort-consuming to solve than linear parameter estimation problems. Because nonlinear parameter estimation is related to the solution of nonlinear function minimization problems and to the solution of sets of nonlinear equations, the software we present may also be applied in those areas.

In our presentation, we have tried to provide enough background to the methods to allow for at least an intuitive understanding of their working. However, we have avoided mathematical detail and theorem proving. This is a book about mathematical software and about problem solving rather than mathematics. In partial compensation for this deliberate restriction of the scope of our treatment, we include a quite extensive bibliography. [We had originally intended to update this bibliography on a periodic basis and distribute it on diskette for a small charge. The costs of doing this have prevented us from its execution. However, we remain interested in bibliographic information on nonlinear estimation and hope to eventually provide an Internet form at modest cost to users. In the interim, we welcome contact at the electronic mail addresses on the frontespiece.]

The methods included have been developed over the last 25 years by a number of workers. The implementations presented have been modified over the past 15 years for use on "small" computers. In fact, the IBM PC and compatible machines are quite large by comparison with the earlier computing environments for which the programs were tailored. Despite the availability of quite high-speed personal computers with large random access memories, there are many reasons why compact programs continue to be useful, for example:

- less code to debug
- faster consolidation / compilation
- possibly faster execution
- less disk overhead for storage of code
- more memory for user problem code.

We have aimed to make this collection of software easy to use and to adapt to a variety of situations. Therefore functionality has been a major requirement, as has reliability. Speed has been secondary. If computing time is critical, users may wish to modify the codes to tune their

performance. However, it is quite possible that reliability may be sacrificed in this process. As it is, one of our major worries in presenting this code is that there remain undiscovered weaknesses which a variety of test problems and multiple perusals of the code have failed to reveal. We respectfully request users who discover such weaknesses to communicate them to us so that we may compile (and eventually distribute) corrections or warnings.

To extend the already quite wide-ranging set of examples which are presented in the following pages, we also are preparing a collection of nonlinear parameter estimation and function minimization problems, presented with program code and (partial) solutions as in the present work. This collection, entitled "Nonlinear parameter estimation: More Examples", is available from the second author at the address above for $25 (prepaid). Also included are the abbreviated function minimization codes.

In preparing this work, we have a number of people to whom we wish to express our gratitude:

to Stephen Nash, for private communication of various important ideas, and especially a version of the truncated Newton method

to Don Watts and Doug Bates, for sharing ideas which helped us to focus on the software and its use

to Mary Nash and Neil Smith, for reading our drafts and living with our enthusiasm for sequences of numbers appearing on a screen

to Lucy Yang, for a portion of the text entry of the early drafts of the book.

The Corona PC-21 microcomputer and some of the computer supplies used in the research and preparation of this work were paid for from research grants from the Natural Sciences and Engineering Research Council of Canada.

NOTATION

We will write any function to be minimized as f(B), where the vector B represents the n parameters

(0-3-1) $\underline{B} = (B(1), B(2), ..., B(n))^T$

though in most real-world situations there will be exogenous data, which for convenience will be stored in a matrix Y. We could write our function (and constraints) using this matrix i.e. f(B,Y). However, this tends to clutter the page and lead to confusion wherever the ",Y" is inadvertently omitted, so we will mostly use the simpler form. Furthermore, we will assume users have taken care to choose units for data so that the function evaluation is reasonably scaled, as we shall discuss in Chapter 3.

We shall strive to be consistent in our use of labels for variables and arrays within the book. Moreover, we shall try to carry over the same notations to computer programs so that the usage is readily accessible for modification and application.

1

OVERVIEW AND PURPOSE

1-0. Beginnings

This introduction is intended to set out the background and history of nonlinear modeling, to provide a statement of the goals and purposes of this book, and to present a notational framework for describing nonlinear estimation problems and solution methods.

1-1. History and Motivations

The development of science and technology has chronicled improvements in our description of natural phenomena, particularly description in the form of mathematical models -- equations and inequalities which allow us to predict the evolution and properties of the phenomena of interest. Such mathematical models relate diverse observable or conceptual variables which together translate into a description of the phenomenon itself. The mathematical forms of the relationships are called functions, with a function being the (many-to-one) mapping of the values of several variables onto the single value which is that of the function. Moreover, the functions themselves involve parameters which give to the model its scale, rates of change, units of measurement, and other aspects which distinguish an individual occurrence of a phenomenon from all similar events or systems.

In talking about nonlinear models, we are referring to the kind of equations and inequalities from which a mathematical model is made. Specifically, we refer to the manner in which the parameters enter into the equations or inequalities and/or the difficulties this form of appearance imposes on the scientist wishing to estimate the parameters. We shall expand on this topic below.

Early mathematical models of reality were frequently centered on the physical world. For example, Kepler's Laws of Planetary motion state:

1. The orbit of a planet is an ellipse with the sun at one focus.

2. The radius vector from the sun to the planet sweeps over equal areas in equal time intervals.

3. The squares of the periodic times are proportional to the cubes of the major axes.

Sommerfeld (1964, pages 38-44) gives the classical mechanics derivation of these "laws," but for our purposes it shall suffice to re-phrase the last law as an equation:

(1-1-1) $$T^2 = K a^3$$

where T is the period of the orbit (time for one complete trip around the sun), a is a measure of the size of the ellipse of the orbit, and K is a constant which relates T and a.

Since a appears to the power 3 and T to the power 2, the relationship between T and a is nonlinear, that is, T cannot be expressed as directly proportional to a to the power 1. The parameter K does, however, appear to the power 1 in the model, and the estimation of K could be carried out with calculations

requiring the solution of linear equations.

Early scientists sought to find "natural laws" which were universally applicable. The parameters and equations of such laws appear from the present perspective to have been regarded as exact. The consideration of errors or fluctuations in observations, which would in the course of calculations alter the values of parameters, is generally attributed to Gauss (1809). Furthermore, Gauss astutely chose to presume the errors in his observations to be distributed according to the so-called Normal distribution, a name statisticians now prefer to replace with "Gaussian distribution" in respect for its inventor and in the hope that a term which is misleading to the untrained reader may gradually drop from usage. The Gaussian distribution of errors, when superposed on a mathematical model, allows us to find the set of parameters which maximizes the probability that the actual observations would be seen. In the case where the model is a linear function of the parameters, that is, each parameter occurs to the power 1 and does not multiply or divide any other parameter, such maximum likelihood parameter estimates can be found by solving a set of linear equations, as shown in Neter and Wasserman (1974, pages 48-49).

Many models of reality use functional forms to relate variables which are linear in the parameters. The most widely used of these is the multiple linear regression model, relating variable y with the varables x(i), i = 1, 2, ..., n-1.

(1-1-2) $$y(t) = \sum_{i=1}^{n-1} B(i)\, x(t,i) + B(n) + \mathrm{error}(t)$$

where y(t), x(t,i) and error(t) are the respective values of variable y, variable x(i) and the error at observation t. Parameter B(n), which in some notations multiplies an artificial variable whose observations are all 1, is called the constant. Maximizing the probability of observing specific data for a set of observations t=1,2,...,m under the assumption that the errors are Gaussian distributed with zero mean and variance sigma^2 (read "sigma squared") requires us to find the maximum of the likelihood function for the Gaussian distribution

$$\text{(1-1-3)} \qquad f(\underline{B}) = \prod_{t=1}^{m} \exp(- \text{error}(t)^2) / (2\ \text{sigma}^2)$$

where

$$\text{(1-1-4)} \qquad \underline{B} = (B(1), B(2), \ldots , B(n))^T$$

For convenience, we have left out the normalizing constant in Equation (1-1-3), which does not change the result. The maximization of the function in Equation (1-1-3) is obtained by noting that the logarithm of f(<u>B</u>) takes on its maximum where f(<u>B</u>) does. Thus the probability of observing y and x(i), i=1,2,...,n at their data positions is maximized by minimizing the sum of squared errors -- hence the concept of <u>least squares</u>.

$$\text{(1-1-5)} \qquad \text{LS: minimize } S(\underline{B}) = \sum_{t=1}^{m} \text{error}(t)^2 \quad \text{w.r.t. } \underline{B}$$

Minimizing the sum of squared errors for the linear regression model involves only functions which are linear in the parameters <u>B</u>. This can be accomplished via the solution of a set of simultaneous linear equations, a task which we shall consider as "easy" from our current perspective, despite the variety of obstacles and pitfalls to obtaining good parameter estimates (see Nash, 1979, pages 37-39,53-58, for a discussion of some of these difficulties).

Our present concern will be with models which do not

permit estimation via the solution of a set of simultaneous linear equations. Thus we shall generally not be interested in models which can, by suitable transformations, be estimated using linear equations. For example, the model

$$\text{(1-1-6)} \qquad y(t) = B(1) + \exp(-B(2) + x(t)) + \text{error}(t)$$

which presents the parameter B(2) within a transcendental function can be transformed to the model

$$\text{(1-1-7)} \qquad y(t) = B(1) + B'(2) \exp(x(t)) + \text{error}(t)$$

where

$$\text{(1-1-8)} \qquad B'(2) = \exp(- B(2))$$

which requires only linear equation solving to estimate B(1) and B'(2). B(2) is then recovered as

$$\text{(1-1-9)} \qquad B(2) = - \log(B'(2))$$

On the other hand, the linear regression model (1-1-2) can be rendered highly nonlinear in the parameters by the imposition of the constraint

$$\text{(1-1-10)} \qquad B(2)\ B(5) = B(3)\ B(4)$$

For a problem of this type having n = 6 parameters and m = 26 observations, see Nash (1979, pages 184-186).

Gauss, despite the very limited computational tools available, recognized the difficulties of nonlinear parameter estimation and described a variant of the Newton-Raphson iteration to attempt a solution. This method, called the Gauss-Newton method, is at the foundation of a number of the more common techniques used today for the nonlinear estimation problem. Its failings are primarily in situations where the errors (or residuals) at the minimum sum of squares are still large. Gauss himself was quite clear on this point -- he only envisaged using the method for problems where the errors were relatively small compared to the quantities measured.

To conclude this section, we formalize our definition of "nonlinear parameter." In order to do this, we first state that a function Z(<u>B</u>) which is <u>linear</u> in the parameters B(1), B(2),...,B(n) to be one where each term in the function has at most one parameter, and each parameter appears only to the first degree, that is, to the first power. This means that Z(B) can be written as a linear combination

(1-1-11) $$Z(\underline{B}) = z_0 + \sum_{i=1}^{n} z_i B(i)$$

where the values of the numbers

(1-1-12) z_i , i = 1, 2, ..., n

define the function. A linear equation results by setting Z(B) to zero. In their forthcoming book on nonlinear least squares problems, Bates and Watts (1986) consider the term "nonlinear" to imply that the gradient of the modelling function is not constant. We will consider that the parameters B are nonlinear if the equations which must be solved or functions which must be minimized in order to estimate their values cannot be reduced to a set of simultaneous linear equations. This is closely related to the Bates/Watts definition, but will encompass estimation problems wider in scope than nonlinear least squares.

In some examples, we may have a choice of estimation techniques. A set of parameters which is linear under one choice may nevertheless be nonlinear under another. It is the linearity of the equations used to calculate the estimates which will govern our definition here.

Some problems have a number of parameters which can be considered linear if other parameters are fixed. For example, if we wish to model some phenomenon by a simple exponential growth model

(1-1-13) Z(B(1), B(2), Y(i,2)) = B(1) exp(B(2) * Y(i,2))
for i = 1,2, ...,m

then B(1) can be estimated by solving a linear equation if B(2) is fixed. That is, we could simply average over the observations to give

(1-1-14) $$\sum_{i=1}^{m} Y(i,1) = B(1) \sum_{i=1}^{m} \exp(B(2) * Y(i,2))$$

(Lawton and Sylvestre, 1971). Such problems have interested a number of researchers whose work has attempted to improve the efficiency of algorithms to estimate parameters by using the special structure (see Section 14-3).

1-2. Purposes

This book is an attempt to provide a software collection for the estimation of nonlinear parameters in mathematical models which arise in diverse scientific disciplines. We shall not restrict our attention to any particular area of application. Indeed, there are applications of nonlinear models in practically every area of study -- wherever people attempt to codify their understanding of the operation of a system or the evolution of a phenomenon by means of a mathematical model, the requirement that parameters be estimated may arise. In the situation that these parameters are nonlinear in the sense described in Section 1-1, the ideas to be discussed in this book may be useful.

In what way will our work be useful to others? Clearly, we cannot discuss all aspects of parameter estimation in a volume of this size. Indeed, we have no intention of creating an encyclopaedic masterwork on the subject. Our goal is more modest: to provide researchers in many disciplines with an approach and with the software tools to solve nonlinear parameter estimation problems using small computers.

First, the approach that we propose to nonlinear parameter estimation is via the minimization of loss functions. That is, it is presumed the researcher has some measure of "goodness" of his model and, given two sets of parameters, can state which is "better" from the point of view of this measure or loss function. We fully expect users to want to alter such loss functions to explore different aspects of their problem:

- statisticians may wish to impose different distributional assumptions on the unknown errors in their models, thereby altering the types of estimators and consequent estimation problems;
- chemical engineers may want to hypothesize alternative chemical reaction paths in the model of a reactor to produce an industrial chemical;
- biologists may want to choose different environmental constraints for a population of animals or plants under study.

Within our approach are certain simplifications and precautions. For example,

- we suggest the use of certain checking programs to verify the operation of program code written by users;
- we urge but do not insist, that users scale their parameters and functions to avoid numbers of widely different magnitude in calculations;
- we suggest that crude estimation methods be used to gain insight into the effects of changes in the parameters on the loss functions used, as well as to impose constraints and develop feasible sets of parameters. Some of these methods have fallen into disuse on large mainframe computers because they are inefficient in finding estimates of nonlinear parameters if used as general tools. However, in the context of small computers, they are useful for explorations of the problem.

"Small computer" is a term which is in need of definition. We shall use it to refer to a personal computer or segment of a larger machine which allows interactive problem solving with easy access to program and data files. In this book we present all programs in BASIC, since most microcomputers either come equipped with a BASIC interpreter or one can be bought for them. Furthermore, many minicomputers and mainframes have BASIC sub-systems with similar facilities. We prefer to discuss programs operating in an interpretive environment for the type of parameter estimation tasks in this book. This is because the ability to quickly modify programs, print variables or array elements and, in some systems, change values and continue execution of a program is highly desirable in this particular context. For "production" tasks, or where the problem is so large that an interpreted program is simply too slow and inefficient, compiled versions of the software are more appropriate. In such cases, however, other programming languages and a somewhat different approach may be more suitable. (Depending on the response to the software presented here, we may undertake versions in other programming languages. Turbo Pascal is a likely environment for early consideration.)

The second goal of this book is to present the software tools to actually carry out the estimation tasks. Primarily, these tools are presented as BASIC "subprograms" which minimize nonlinear functions -- either general or specialized to sums of squares -- and which may or may not require derivative information to be provided by the user.

The third goal is to present the rudiments of the software "glue" to integrate the tools above into an easily used system for solving estimation problems. The individual estimation programs are useful and the core of our approach, but they do not lend themselves to easy use by themselves. Extra programs or subprograms are included to aid the user in related tasks such as

- to check program code for the loss function or its derivatives
- to incorporate bounds constraints on parameters
- to plot two-dimensional data
- to determine the computational environment
- to perform postestimation analysis to determine the sensitivity of the parameters.

We do not provide tools for the entry and edit of data, since there are many good, inexpensive tools for this purpose on most small computers.

The fourth goal is to document both the approach and software in a unified and consistent way. Since BASIC uses global variable and array names, this part of our work is extremely important -- should a user employ a variable name already storing a key control value, our software will not work as anticipated and the results may be totally wrong. In this regard, the choice of interpreted BASIC can lose a great deal of the benefit gained in the ease of development of function and derivative subprograms. However, we feel this benefit outweighs the disadvantage of potential variable name confusion. For convenience, we have chosen to work in the Microsoft flavor of BASIC. Although this allows us to use relatively long variable names, we shall restrict ourselves to a single letter or a single letter plus a single digit in conformance with the ISO Minimal BASIC standard (ISO 6373-1984). By so doing, we create a few headaches for ourselves in preparing and testing the programs here, but allow many more users to adapt the software to their own particular

computing environments. Appendix D presents details of the choices we have made.

Our four goals are thus summarized as

1. approach
2. software tools
3. system design and prototype programs
4. documentation of methods and programs

We anticipate that most readers of this book will use it in one of three ways, which represent differing levels of use:

1. as an introduction to nonlinear estimation for non-specialists
2. to learn/use a particular tool
3. to build their own nonlinear parameter estimation system or to integrate our methods with another (statistical or database) package.

Having presented our intentions, it is also pertinent to list some of the aspects of nonlinear parameter estimation we will NOT address. We do not intend to make more than passing reference to specialist techniques in particular disciplines. There are, in fact, many special models for which highly efficient and well-established estimation methods have been derived. However, these do not, we feel, fit well into the development of a general parameter estimation package for small computers. The difficulty is not one of the length or complexity of program code to include a number of such methods -- it is the amount of effort to properly explain and document such a compendium of tools which may have only a limited range of applicability and small target audience.Nor do we intend to discuss at any length the foundations of the statistical theory upon which estimation methods are based. This is an important subject area, but we shall focus our effort on solving problems after they have been posed, rather than developing the formulation from first principles.

1-3. Classic Problems

While Chapter 2 introduces more examples of nonlinear parameter estimation problems, we wish to point out at this very early stage that these tasks are well-established throughout science and technology. Some of the particular cases which have developed a large literature are those of finding models to describe:

- the relationship between plant yield and density of planting;
- the change over time of the size of plants, animals or economic activity -- growth curves;
- the rate of production or disappearance of a chemical or radio-isotope in a chemical or nuclear reaction.

Examples of data for such situations are:

- the data for growth of White Imperial Spanish Onions at Purnong Landing given by Ratkowsky (1983, page 58) and presented in Table 1-3-1 and plotted in Figure 1-3-1;
- the weed infestation data discussed in Section 2-1 and presented in Table 2-1-1 (next chapter) and plotted in Figure 2-1-1;
- data for the decomposition of methyl iodide in alkaline solution given by Moelwyn-Hughes (1961, page 1253) and presented in Table 1-3-2 and plotted in Figure 1-3-2. This problem is discussed in more detail in Section 16-5.

Many of the examples in this book arise in these "classic" tasks, but we shall introduce problems from other situations. Since one of our major goals is to provide software, we shall also present easily formulated test problems to allow methods to be compared. Readers may note that our plotted curves often have a very simple structure. Most, but not all, have been created with the program PLOT.BAS which is described and listed in Chapter 15. While more sophisticated plots could have been used for presentation in this book, we prefer to show the reader the utility of simple graphs which can be created within an inexpensive and commonly available computing environment.

Table 1-3-1. Yield versus density for White Imperial Spanish Onions at Purnong Landing

X = density	Y = yield	X = density	Y = yield
23.48	223.02	70.06	116.31
26.22	234.24	70.45	120.71
27.79	221.68	73.98	134.16
32.88	221.94	73.98	114.48
33.27	197.45	78.67	91.17
36.79	189.64	95.90	101.27
37.58	211.20	96.68	97.33
37.58	191.36	96.68	101.37
41.49	156.62	101.38	97.20
42.66	168.12	103.72	87.12
44.23	197.89	104.51	81.71
44.23	154.14	105.68	76.44
51.67	153.26	108.03	87.10
55.58	142.79	117.82	84.54
55.58	126.17	127.21	69.09
57.93	167.95	134.26	64.40
58.71	144.54	137.39	66.81
59.50	151.30	151.87	63.01
60.67	130.52	163.61	55.45
62.63	125.30	166.35	62.54
67.71	114.05	184.75	54.68

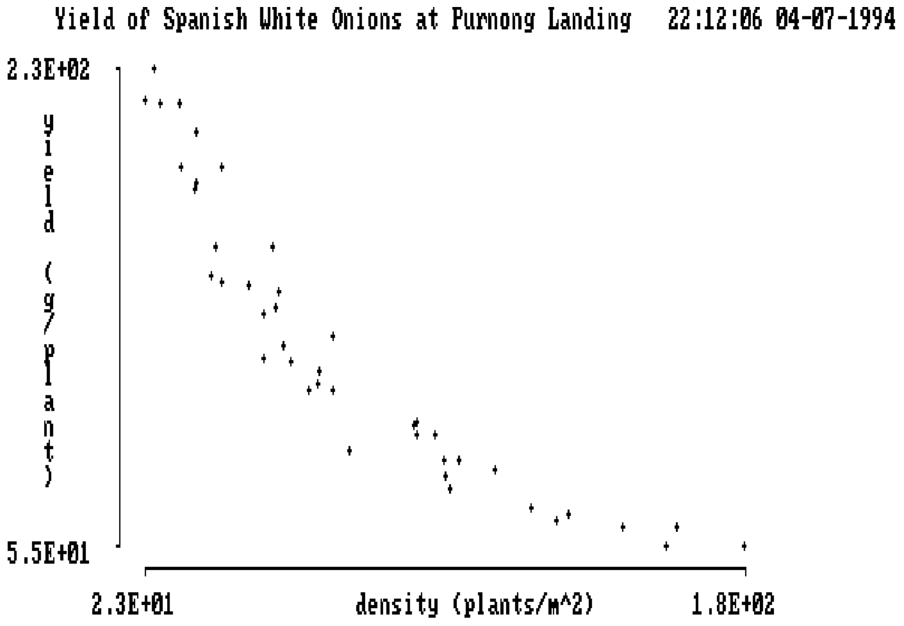


Figure 1-3-1. Plot of the "ONION" data set in Table 1-3-1.

Table 1-3-2. The fractional decomposition of methyl iodide in aqueous alkaline solution at temperature 342.8 K

t=time in minutes	observed percent completion
0	0
3.5	16.8
8.5	36.0
13.5	50.7
19.5	64.3
26.5	75.2
37.5	85.1
infinity	100.0

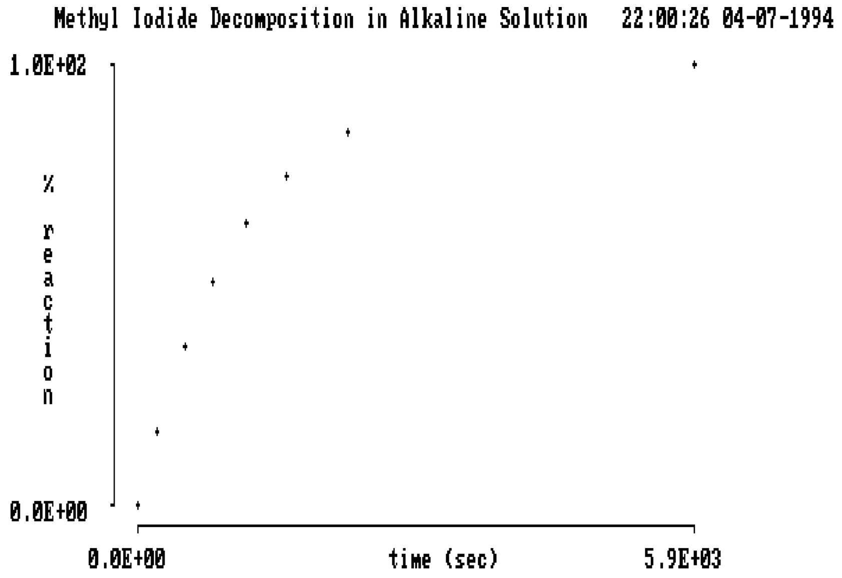


Figure 1-3-2. Plot of the percent completion of decomposition of methyl iodide with time.

1-4. Landmarks in Nonlinear Estimation Methods

This section presents a brief historical overview of the topic of nonlinear estimation. We make no claim to being exhaustive, and wish only to provide a perspective for the material we will present in later chapters.

Isaac Newton provided scientists and mathematicians with much food for thought. His developments in the calculus were not only theoretical, and from the late 17th century to the beginning of the 19th century, Newton's method (also referred to as the Newton-Raphson method) was the only approach to the solution of nonlinear equations, and hence of parameter estimation problems. However, it cannot be said that there are many examples of such problems being solved. It is only when Gauss and others, taking advantage of the developments in instrumentation which accompanied the industrial revolution, began to calculate orbital parameters for planets and other astronomical objects that we see attempts made to develop the computational techniques. Gauss (1809), by recognizing that the calculations for the Newton method is simplified if the residuals (model minus observation) were small, developed the Gauss-Newton method for nonlinear least squares problems.

In large measure, it can be stated that most function minimization methods which use the gradient of a function still owe much to Newton, and most nonlinear least squares methods still find their origins in the Gauss-Newton method. The developments which have occurred have mainly improved the Newton and Gauss methods in

- increasing their ability to obtain correct solutions when presented with poor guesses for the parameter estimates, since initial values of the parameters must be provided so that the iteration can proceed;
- reducing the computational effort required within the Newton and Gauss-Newton methods either in the calculation of derivatives or in the solution of the iteration equations;
- reducing the storage requirements of the methods, that is, the amount of information needed to operate them.

Particular developments of note in the progress of nonlinear estimation techniques follow.

Cauchy (1848) suggested the steepest descents method, that is, an iteration wherein the gradient is used as a

search direction, the loss function is reduced along the gradient direction, and the process is repeated from a new (lower) point. In practice, this is NOT a good method in general, but incorporating elements of it can be useful for ensuring progress of a method. Thus, many of the approaches we shall suggest will make use of occasional explicit or implicit steepest descent steps.

Levenberg (1944) suggested that the iteration equations for the Gauss-Newton method could be stabilized by modifying the iteration equations to include a steepest-descent component. While Levenberg's work was largely ignored by practitioners, its rediscovery by Marquardt (1963) and the distribution of a computer program implementing the ideas were perhaps the biggest practical contribution to nonlinear parameter estimation to date.

A somewhat different approach was used by Hartley (1961), who modified the Gauss-Newton method by treating the solution of the iteration equations as a search vector (see Chapter 11).

From the point of view of function minimization, as opposed to nonlinear least squares, the developments of digital computers prompted workers to try direct search methods for function minimization. Methods were proposed by, among others, Rosenbrock (1960), Hooke and Jeeves (1961) and Box (1957). Of these methods, the Hooke and Jeeves method survives almost in its original form and is presented in Chapter 5, while the "simplex" method of Box was improved by Nelder and Mead (1965) to a form similar to that which appears in Chaper 6. Other methods, such as that of Rosenbrock, appear to have fallen into disuse.

Gradient methods have also been improved to provide efficient techniques which avoid the computational effort of Newton's method and the slow convergence of Steepest Descents. Davidon (1959) showed how to build up the Newton solution from a sequence of gradient steps. Thus the work at each iteration of his "variable metric" method was comparable to that of Steepest Descents but convergence was more like that of Newton's method. Fletcher and Powell (1965) published what has become one of the most popular algorithms of this type. In Chapter 10 we use a variant by Fletcher (1970).

Variable metric (or quasi-Newton) methods still require a lot of data storage to operate, and for functions in a large number of parameters we would still be required to use steepest descents if Fletcher and Reeves (1964) had not noticed that ideas from iterative approaches to the solution of linear algebraic equations could be applied to nonlinear minimization. The resulting conjugate gradients method, modified and adapted, appears in Chapter 8. A further refinement along somewhat similar lines resulted in the truncated-Newton method (S.G.Nash, 1982; Dembo et al., 1982). See Chapter 9.

The nonlinear least squares problem has received special attention from time to time. Of note is the program NL2SOL of Dennis, Gay and Welsch (1981), which uses the secant method. While the length and complexity of this code mean that we have not considered including it here, it should be kept in mind by those working in FORTRAN environments on larger computers.

An important area of research has been the attempt to understand the statistical meaning and consequences of nonlinearity in models and loss functions. Bard (1974) remains a source of information and inspiration in this area. On the topic of nonlinear least squares, Donald Watts and his associates have done much to advance understanding of the subject (for example, in the forthcoming book by Bates and Watts, 1986). In a very different fashion, work on generalized linear models, as illustrated in the book by

McCullagh and Nelder (1983), will have an important place in nonlinear modeling.

This brief survey ignores a great deal of research work which is important in the field. For example, no mention has been made of the efforts spent in incorporating constraints in nonlinear function minimization methods.

1-5. Statistical Approaches

Underlying all scientific and technological developments are the many years of tedious observation, data analysis and modeling and model testing needed to establish the "laws" of nature. With all observations, an element of error creeps into data. For the most part we will ignore (unexplained) systematic errors in observations, for example, due to equipment or the observational environment. Note, however, that systematic errors may be uncovered by careful analysis of the problem and data (Havriliak and Watts, 1985). We will also generally assume that observations with obviously wrong data have been eliminated from the problems we shall study. This task -- the detection and removal of outliers -- is one which cannot be taken lightly. However, we believe that it is outside our present purview, despite being a topic of great importance. (For a review, see Beckman and Cook, 1983.)

In the data left after systematic errors and (supposedly) accidental outliers have been removed, there will then remain some level of "error" from an ideal form which is due to the uncontrollable fluctuations in the system under study or the observational equipment. Ultimately, because of the Heisenberg uncertainty principle (Schiff, 1955, page 7), there is a limit to the control of such observational errors, and we are forced to take a statistical view of the fluctuations.

A simplified description of the statistical approach to estimation is as follows:

1. specify the mathematical form of the model of the system under study;
2. specify the nature of the fluctuations about this model (error distribution);
3. derive the relationship between the observed data and the unknown parameters of the model;
4. use this relationship to determine the most probable parameter values or those which minimize the deviation between predicted and observed behavior of the system.

Note that the first two of these four steps require that we <u>assume</u> a great deal about the system, namely, the form of the functional relationship describing the system of interest as well as the distribution of errors made in observing it. Clearly, one needs to have quite strong reasons for making such assumptions. Such reasons may be based on well-established theories, or the necessity of obtaining results, but the assumptions should ultimately be tested by comparison with the consequences of other possibilities. In some cases, it may be possible to show that the resultant model is relatively insensitive to the particular functional form of the model or error distribution suggested. In other situations, we may have to make a difficult decision between alternative models with different interpretations if they are accepted as reliable descriptions of the properties and behavior of the system under study. More observations and experiments are then required to resolve the uncertainties.

Given a form for the model function and for the error

distribution, it is straightforward to design estimators for the parameters of the model. It is not our intention to explain statistical estimation theory, but the following example of maximum likelihood estimators illustrates one application of the methods we present later.

Let us suppose that the model function at the i-th observation yields a residual function (that is, deviation between observed and predicted behavior)

(1-5-1) $r(\underline{Y}_i^T, \underline{B})$ or $r(i,\underline{B})$

where Y_i^T represents the i-th row of the matrix of observed data, Y.

The error distribution function describes the relative probability that a given value of a residual is observed. If the true model parameters $\underline{B}^*$ are known, then the error distribution function describes the relative probability that an observation $\underline{Y}^T$ is recorded. Alternatively, we may consider the observed data to be given facts, so that the error distribution allows the relative probability (or likelihood) of different parameter sets to be compared.

With respect to the residual $r(i,\underline{B})$, we will write the probability density function as

(1-5-2) $P(r(i, \underline{B}))$

That is, the integral of this function over all possible values of r is one. The joint probability density that observations i and j are sampled i.e., that they are independently drawn at random in a sample of size 2, is

(1-5-3) $P(r(i, \underline{B})) * P(r(j, \underline{B}))$

and the joint probability density of an entire sample of observations is the product of the density functions for each real observation. The <u>most probable</u> set of parameters $\underline{B}$ is the set which maximizes this probability density. If the function can be written down, then methods for function maximization can be applied. (Usually we minimize the negative of the product of probability densities, or, frequently, the negative of the logarithm of this product to convert the functional form to a summation.)

This maximum likelihood approach to parameter estimation is an important class of estimators. However, its dependence on a specific probability density function for the error distribution should be kept clearly in mind. We maximize the likelihood of occurrence of a set of parameters under the assumed error distribution and under specific assumptions regarding the validity of the observations.

Other authors have explored these statistical aspects of nonlinear parameter estimation in more detail than we shall here. Bard (1974, Chapter 4) surveys a variety of statistical estimation techniques. Ratkowsky (1983), while restricting attention to errors which in one problem are independent of each other and all drawn from the same ("identical") Gaussian distribution, covers a number of important classes of nonlinear models in some detail. McCullagh and Nelder (1983) approach the task from the view of generalized linear models. While their approach and motivations are different from our own, their success in solving difficult modeling problems deserves the wide respect it has earned. The programs we present could be used within the McCullagh/Nelder formulation, which requires a sound understanding of relatively advanced statistics. The approaches we present are frequently less elegant routes to solutions of parameter estimation problems. They also admit constraints in a way which we believe to be easier for users to apply, but provide generally less statistical information about the parameters and hence models estimated. Research is continuing in both areas and one may hope to see a comprehensive and constructive synthesis of the two approaches in the not-too-distant future.

1-6. Software Approaches

Methods for nonlinear estimation have been a popular subject for research, as the bibliography will substantiate. Software for particular methods has also been announced from time to time. (For a review to 1973, see Chambers, 1973.) We divide the forms of this software into three classes:

- software packages
- parts of a program library
- individual programs or subroutines

Packages allow the user the greatest facility in solving parameter estimation problems. That is, a package provides for data entry, edit and transformation, the development of the model function and error distribution and the estimation of parameters. The price paid for convenience is usually the effort to learn the operation of the package, the cost of acquiring or using it and the general lack of flexibility.

Packages which allow nonlinear parameters to be estimated are:

- GLIM, a generalized linear modeling package (McCullagh and Nelder, 1983, page 239);
- GENSTAT, a statistical computing language (McCullagh and Nelder, 1983, page 239); and
- SAS, via the NLIN function (SAS User's Guide, 1982).

Libraries of computer programs have been an important source of tools for solving nonlinear estimation problems. Four of the major libraries which are widely available are:

- the NAG library, which includes several nonlinear optimization methods, including at least one for nonlinear least squares (Because the library changes slightly at each release, we refer the reader to the latest documentation for precise information.);

- the IMSL library, which similarly offers several routines for nonlinear optimization and least squares, for example, the nonlinear least squares routine RNSSQ in the STAT/PC-LIBRARY (IMSL, 1984; Nash, 1986a; Nash, 1986b);
- the Harwell library, which is especially rich in nonlinear optimization routines;
- BMDP, which offers routines PAR and P3R to solve nonlinear least squares problems (BMDP, 1981, page 289ff.).

A number of individual programs have been publicized over the years. Many of these are linked to particular subject areas. For example, on pharmacokinetics, programs have been reported or discussed by Duggleby (1984a,1984b), Sedman and Wagner (1976), Pfeffer (1973), Pedersen (1977), Brown and Manno (1978), Peck and Barrett (1979) and Landriani, Guardabasso and Rocchetti (1983). For physics, James and Roos (1975) propose a polyalgorithm (several methods in one program) for the fitting and analysis of parameters. Using simple examples from chemistry, Caceci and Cacheris (1984) presented a Nelder-Mead polytope program in PASCAL. (This listing may appear heavily weighted towards chemical kinetics, either general or pharmacological. We believe that programs have been developed in other fields, but that they may not have been published. For example, we have had enquiries for our own work from engineers who were working on defense projects who did not wish to discuss the nature of the problems they were solving.)

One of the more widely discussed individual programs for nonlinear estimation is that of Marquardt (1964,1966). A simplified version of Marquardt's method has been presented as a step-and-description algorithm by Nash (1979). Bard (1974, Appendix G) lists several other nonlinear estimation

programs, but omits the more recent program NL2SOL of Dennis, Gay and Welsch (1979).

To conclude this section, we would stress that nonlinear parameter estimation cannot be fully automated because of the important decisions which must be made concerning model and error distribution forms, the rejection of outliers and the analysis of results. There is here the necessity for a partnership between user, software and machine if nonlinear estimation problems are to be efficiently solved. No one piece of software is likely to satisfy all requirements. Our own preference is to have at hand a number of software building blocks which can be quickly put together to approach a given problem. This approach is reflected in the programs presented later in the book.

1-7. Characterization of Problems

In this section we shall specifically define the classes of problems which will be of interest to us. We will use the vector

$$\underline{B} = (B(1), B(2), \ldots, B(n))^T = (B_1, B_2, \ldots, B_n)^T \qquad (1\text{-}7\text{-}1)$$

to represent the set of parameters to be estimated.

In estimating the parameters in nonlinear models, one of the most important general problems is that of finding the minimum of a function of n parameters $\underline{B}$ which is itself the sum of the squares of m functions -- the nonlinear least squares problem (NLLS).

<u>Problem NLLS: Nonlinear least squares.</u>

Find the set of parameters $\underline{B}$ which minimizes the loss function

$$(1\text{-}7\text{-}2) \qquad f(\underline{B}) = \sum_{i=1}^{m} [r(i,\underline{B})]^2$$

The functions r(i,B) are called residuals.

By taking the first derivatives of f(B) and setting them to zero in the classical method for finding the stationary points of a function (Protter and Morrey, 1964, Chapter 20), that is, the maxima, minima or saddle points, we obtain the nonlinear equations problem (NLE).

Problem NLE: Nonlinear equations or rootfinding.

Find the set of parameters B which causes the set of equations

$$(1\text{-}7\text{-}3) \qquad r(i,\underline{B}) = 0 \qquad i = 1,2,\ldots,m$$

Frequently m = n, so that the number of equations equals the number of parameters.

It is also possible to consider parameter estimation problems in which the loss function f(B) does not appear in the form of a sum of squares, in which case we may wish to consider the general unconstrained function minimization problem (FMIN).

Problem FMIN: Unconstrained function minimization.

Find the set of parameters B which minimizes f(B). This may be a local minimum i.e.

$$(1\text{-}7\text{-}4) \qquad f(\underline{B}) < f(\underline{B} + \underline{delta})$$

for all vectors delta such that length(delta) < deltasize.

Alternatively, we may pose the much more difficult problem of finding the global minimum of the function, that is, the B where it takes on its lowest value for all possible sets of parameters.

Problems FMIN and NLLS can have constraints imposed on the allowed parameters B. Such constraints may be in the form of m3 equations, or equality constraints,

$$(1\text{-}7\text{-}5) \qquad c(i,\underline{B}) = 0 \qquad i = 1,2,\ldots,m3$$

or appear as inequalities

$$(1\text{-}7\text{-}6) \qquad c(i,\underline{B}) \geq 0 \qquad i = m3+1, m3+2,\ldots,m4.$$

Of the inequalities, one class is so common that our methods will generally be set up to take account of them. This is the class of bounds, that is, constraints of the form

$$(1\text{-}7\text{-}7) \qquad 2 \leq B(3) \leq 4$$

We rewrite (1-7-7) as two constraints to obtain the general form (1-7-6)

$$(1\text{-}7\text{-}8) \qquad \begin{aligned} B(3) &\geq 2 \\ -B(3) &\geq -4 \end{aligned}$$

The general nonlinear function minimization problem can now be written down as follows:

Problem NLP: General function minimization or nonlinear programming.

Given data Y, minimize f(B, Y) with respect to B under the constraints

$$(1\text{-}7\text{-}5) \qquad c(i,\underline{B}) = 0 \qquad i = 1,2,\ldots,m3$$

and

$$(1\text{-}7\text{-}6) \qquad c(i,\underline{B}) \geq 0 \qquad i = m3+1, m3+2,\ldots,m4.$$

It is straightforward to write down this problem, and not too difficult to actually specify all the functional forms needed for a number of practical problems. However, finding a solution is not trivial, and at this point we have no guarantee that a solution even exists. Moreover, the specific expression of a particular problem is open to a number of choices. For example, a single equality constraint,

B(1) + B(2) * B(3) = 4

is equivalent to the two inequalities

$B(1) + B(2) * B(3) - 4 \geq 0$

and

$- B(1) - B(2) * B(3) + 4 \geq 0$

A more serious problem arises when the constraints are such that there is NO solution, for example, if

$B(1) + B(2) * B(3) \geq 5$

and

$B(1) + B(2) * B(3) \leq 3$

are supposed to be satisfied simultaneously. In this case the pair of constraints is infeasible.

In general, each equality constraint should allow one parameter to be calculated from the rest, thereby reducing the dimensionality of the problem which has to be solved. In practice, it may be difficult to determine if each of the constraints imposes an independent relationship on the set of parameters $\underline{B}$. However, for linear constraints, this is a well-understood procedure.

A categorization of constraints concerns their number. When there are very many constraints, particularly linear ones, it is common to approach the function minimization problem by exploring the possible feasible points. That is, we do not change the parameters $\underline{B}$ primarily to reduce the function f($\underline{B}$), but to move between points which are feasible with respect to the set of constraints. The general name given to this process is mathematical programming, which has unfortunate consequences for the development of indexes to the mathematical and computing literature.

Linear programming (LP) involves the minimization of a linear function with respect to linear constraints.

Integer programming (IP) problems have parameters $\underline{B}$ which are integers. The problems are usually of the LP type. Mixed-Integer Programming problems involve both integer and floating-point parameters, and are among the more difficult to solve efficiently.

Quadratic programming (QP) problems have a quadratic function to be minimized subject to linear constraints. They are extremely common, arising in many statistical and image processing applications.

Nonlinear programming (NLP) is the general function minimization problem defined above, though the term is usually reserved for cases where there are a large number of constraints or parameters.

In this book, we will concentrate on problems having only a few explicit constraints. We shall approach such problems from the point of view of the loss function which we shall attempt to minimize. Constraints will be included by either altering the function so that the solution of an unconstrained problem coincides with that of the constrained one, or by forcing our search into the feasible region. Generally we expect that there will be only a few constraints in addition to bounds on the parameters.

As indicated, the various problems above are quite closely related:

1. Problem NLE is a problem NLLS with the loss function required to be zero at the minimum;

2. Classical calculus solution of problem FMIN sets the partial derivatives of the loss function to zero, a problem of type NLE. Note that additional conditions must be imposed on the solution to avoid local maxima and saddle points (Gill, Murray and Wright, 1981, pp 63-65);

3. Problem NLLS is clearly a special case of problem FMIN.

Just as the problems are closely linked, so are the methods for their solution. A discussion of these relationships is given in Section 3.2 and in some of the examples throughout the book.

Gill, Murray and Wright (1981) give problem FMIN without constraints the label UCP, while with constraints, as NCP. Conn (1985) calls it problem NLP as we have done here.

1-8. Outline of Intentions

In Section 1-2 we have presented the general purposes of this book and software. Now we will discuss what types of material in relation to nonlinear parameter estimation methods we hope to present in the following chapters.

1. We will deal primarily with methods for unconstrained problems. Bounds on parameters will be an important exception to this generalization. We will handle other constraints by methods mentioned briefly in Section 3-2 and detailed in examples.

2. The methods we present in this book are intended to be simple but effective. We will sacrifice machine cycles for user convenience and will not be overly bothered if convergence of an iteration is several or even many times slower than the "best" method for a problem. We will, however, wish the methods we employ to allow users to make steady progress toward solving real-world problems, and to do this without a great deal of difficulty.

3. We desire our methods to be useful on a variety of personal computers. This should include machines such as the Apple II family, the IBM-PC family, various CP/M machines and portables such as the Radio Shack TRS-80 Model 100. In this incarnation of our methods, we have used BASIC as the programming language. To allow for file input/output, we

have used the Microsoft dialect of BASIC (Microsoft, 1983) for those parts of the code which require file I/O functions. However, for the most part our code conforms to the International Standard ISO 6373-1984 for Minimal BASIC (Appendix D).

4. Our methods should be capable of handling most nonlinear parameter estimation problems which present themselves to users. There will surely be problems too extreme, too large, or too specialized for our methods, but by and large, we expect that the package of techniques presented herein to be comprehensive of most user problems.

5. Algorithm presentation is not a priority in this work, though algorithms are of course imbedded in the software. Other works, including our own, (Nash, 1979) present the details of how many of the algorithms work. Here we shall be satisfied with a general perspective on parameter estimation and function minimization methods.

6. Despite our intention NOT to present algorithms as such, we shall try to point out the important details of coding of parameter estimation programs, and of the loss functions and their derivatives, along with the background reasons for these details, which make a great deal of difference to the performance and success of programs and their users.

7. We hope that our work will provide a useful and inexpensive learning tool for this field.

COVER SHEET

Chapter 2

Chapter title: Examples of nonlinear parameter estimation problems

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

EXAMPLES OF NONLINEAR PARAMETER ESTIMATION PROBLEMS

2-0. Examples -- an Overview

In this chapter we consider some situations which give rise to nonlinear parameter estimation problems. By presenting such situations, we aim to

- show the diversity of origins of nonlinear parameter estimation problems;
- illustrate the fact that diverse contexts may give rise to estimation problems which are essentially equivalent in the model functions, the data sets to which they are fitted, and the estimation methods used;
- provide several specific problems which will be used to illustrate the methods presented in later chapters.

We strongly urge readers to keep the context of the problem in mind throughout the solution process. As developers of methods, we seek to unify problems so that a common approach and a common software package can be applied to solve them. As practitioners, we must adapt the approach to take account of the specific nature of each problem. This means that we must be cognizant of the implicit additional facts which apply, such as

- the range of values possible for variables and parameters
- the meaning which may be attached to different values of variables or parameters
- the general size of the loss function which is to be expected at the solution.

Generalized software, such as that presented later, cannot take account of all such pieces of information without considerable extra program code, and documentation, as well as more work for the user to specify the information, so that it can be used by the program.

In our approach, we have allowed bounds to be imposed on the parameters, but other forms of specific information are left to the user to include in the program code he/she must provide for the loss function. This represents one choice of the balance point between the developers and users of software which we feel minimizes the total effort expended in constructing software and using it.

2-1. Growth and Decay Models

In practically all areas of the physical, biological and social sciences as well as their applications in engineering, medicine or management, situations arise where growth or decay of some variable is observed. The modeling of such a phenomenon may be important in order that the processes underlying the changes may be understood or the change predicted.

Example A: Weed infestation.

An experimental soil plot is observed each year at the same date and the number of weed plants per square metre is recorded. The plot is initially bare of plants, so in year 1 we are observing the results of seeding by wind, insects or other agents. Thereafter, some seeding from established weeds will take place. After some years the plot will be choked with weeds - - there will be no soil surface available for more plants - - and the numbers per square metre will level off. Data for this problem was initally supplied to one of us (JCN) by Mr. D. Hobbs of Agriculture Canada (Nash, 1979, pages 120-121). The model suggested for this problem was a sigmoidal or s-shaped curve, in particular the logistic function, which gives a nonlinear least squares problem with the residual expressed as

(2-1-1) $r(i,Y) = B(1)/[1 + B(2) * \exp(i*B(3))] - Y_{i1}$

where i is the observation number, hence the year after the start of the experiment in which the obsevation was was made, and Y_{i1} is the number of weeds per square metre observed.

Readers may note that the residual is written

(2-1-2) residual = expected - observed

which is the negative of traditional forms. (We are in good company with both Gauss and Legendre according to McCullagh and Nelder, 1983, page 7.) However, we recommend that residuals be expressed in this way so that the derivatives of the residual with respect to the parameters $\underline{B}$ are the same as those of the model function with respect to $\underline{B}$. One opportunity for a sign error to occur is thereby avoided.

The data for the Hobbs problem, as we shall call it, is given in Table 2-1-1. We shall use this example throughout the book to illustrate the behaviour of the various methods. This data is plotted in Figure 2-1-1.

Table 2-1-1. Weed infestation data (source D. Hobbs)

Year	Weeds per square metre
1	5.308
2	7.24
3	9.638
4	12.866
5	17.069
6	23.192
7	31.443
8	38.558
9	50.156
10	62.948
11	75.995
12	91.972

There are a number of other functional forms for sigmoidal growth curves (Ratkowsky, 1983, chaper 4). These use a time variable (e.g. "i" in Equation (2-1-1)) as well as the observed dependent variable (e.g. weeds per square metre). We will use

Y_{i1} = observed dependent variable
Y_{i2} = time (independent) variable

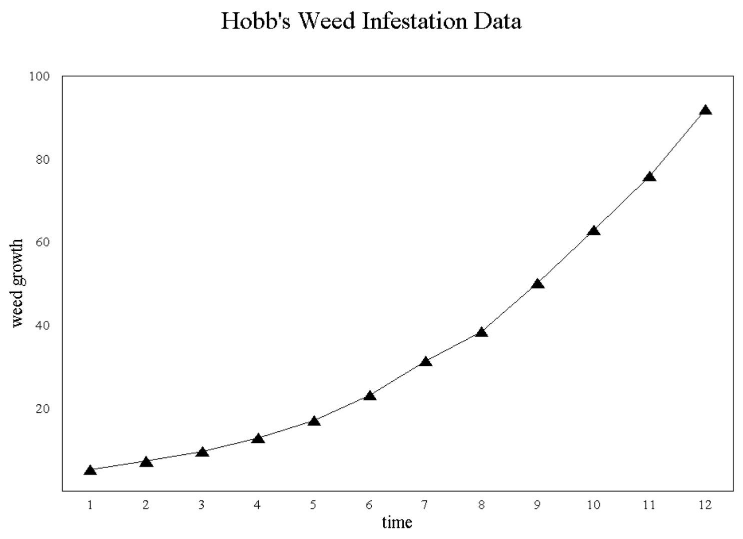


Figure 2-1-1. A plot of weed infestation versus time, using the data of Table 2-1-1.

In our notation, Ratkowsky's sigmoidal curves are logistic:

(2-1-3) $r(i,Y) = B(1)/[1 + \exp(B(2)-B(3)*Y_{i2})] - Y_{i1}$

(Note that this is equivalent to (2-1-1) except that B(2) has been replaced by exp(B(2)) and B(3) by -B(3).)

Gompertz:

(2-1-4) $r(i,Y) = B(1)*\exp[-\exp(B(2)-B(3)*Y_{i2})] - Y_{i1}$

Richards:

(2-1-5) $r(i,Y) = B(1)/[1+\exp(B(2)-B(3)*Y_{i2})^{(1/B(4))}] - Y_{i1}$

Morgan-Mercer-Flodin (MMF):

(2-1-6) $r(i,Y) = [B(2)*B(3)+B(1)*Y_{i2}^{B(4)}]/[B(3)+Y_{i2}^{B(4)}]-Y_{i1}$

Weibull-type:

(2-1-7) $r(i,Y) = B(1) - B(2)*\exp(-B(3)*Y_{i2}^{B(4)}) - Y_{i1}$

Each of these forms has properties which may make it more or less suitable for a particular situation, but all are continuous functions of the time variable Y_{i1}. One of us (JCN) has also suggested that discrete functions may be more appropriate in situations such as that of the Hobbs weed infestation problem (Nash, 1977), as plant reproduction takes place in discrete cycles.

The sigmoidal or s-shaped curve, either for growth or decay, is one functional shape of interest. However, other forms of growth or decay are also of interest. One of the most important of these is the exponential family, which is explored in the next example.

Example B: Chemical reactions.

Bard (1974, pages 123-133) presents the problem of modeling a simple first-order chemical reaction (see Laidler, 1965, Chapter 1 for a discussion of kinetic laws) which converts compound A into compound B. Letting

Y_{i1} = fraction of compound A remaining at time point i

Y_{i2} = i-th time point recorded

Y_{i3} = absolute temperature

we write the model for this reaction as

(2-1-8) $r(i,Y) = \exp[-B(1)*Y_{i2}* \exp(-B(2)/Y_{i3})] - Y_{i1}$

The data for this problem is given in Table 2-1-2. In this example we have a simple exponential decay of the fraction of compound A remaining providing the temperature is fixed. The reaction proceeds more quickly the higher the temperature. This model is for a process where it is reasonable to think of time as a continuous variable.

Table 2-1-2. The data for the chemical reaction example

Experiment number i	Time (hr) Y_{i2}	Temperature (degrees K) Y_{i3}	Fraction A remaining Y_{i1}
1	0.1	100	0.980
2	0.2	100	0.983
3	0.3	100	0.955
4	0.4	100	0.979
5	0.5	100	0.993
6	0.05	200	0.626
7	0.1	200	0.544
8	0.15	200	0.455
9	0.2	200	0.225
10	0.25	200	0.167
11	0.02	300	0.566
12	0.04	300	0.317
13	0.06	300	0.034
14	0.08	300	0.016
15	0.1	300	0.066

When a phenomenon takes place at discrete time intervals, for example, agricultural production, investment or accounting activity, school or university graduations, it makes sense to model the phenomenon by a function which is discrete in time. Other phenomena which may be better modelled with discrete functions involve investment decisions leading to improved production efficiencies (Nash and Teeter, 1975).

Example C: Discrete time growth or decay models.

A trivial example is the depreciation of an asset by the declining balance method. Each year a fixed percentage of the asset value is depreciated. In Canada, computer hardware typically carries a 30% Capital Cost Allowance rate for taxation purposes, so that 30% of the remaining value is a permitted depreciation expense. This implies that the undepreciated value is 70%, so the "model" for undepreciated value after t years is

(2-1-9) $\quad$ (Undepreciated value) = (Initial value) * 0.7^t

Such a model provides for an exponential decay. It is easy to convert to the exponential form

(2-1-10) $\quad$ (Undepreciated value) =
(Initial value) * exp(t*ln(0.7))

However, t only takes on integer values.

Discrete models for sigmoidal curves are also possible (Nash, 1977). As an example, consider the fraction of households adopting a new technology which is highly beneficial, such as electricity or telephone service. Assume no household abandons this technology once adopted. If the fraction of households having the technology is x, assume that the probability of adoption is proportional to x. Therefore

(2-1-11) $\quad x_{t+1} - x_t = k * x_t$

where k is similar to a rate constant. Figure 2-1-2 shows the shape of the model function if x_0= 0.05 for different values of the rate constant.

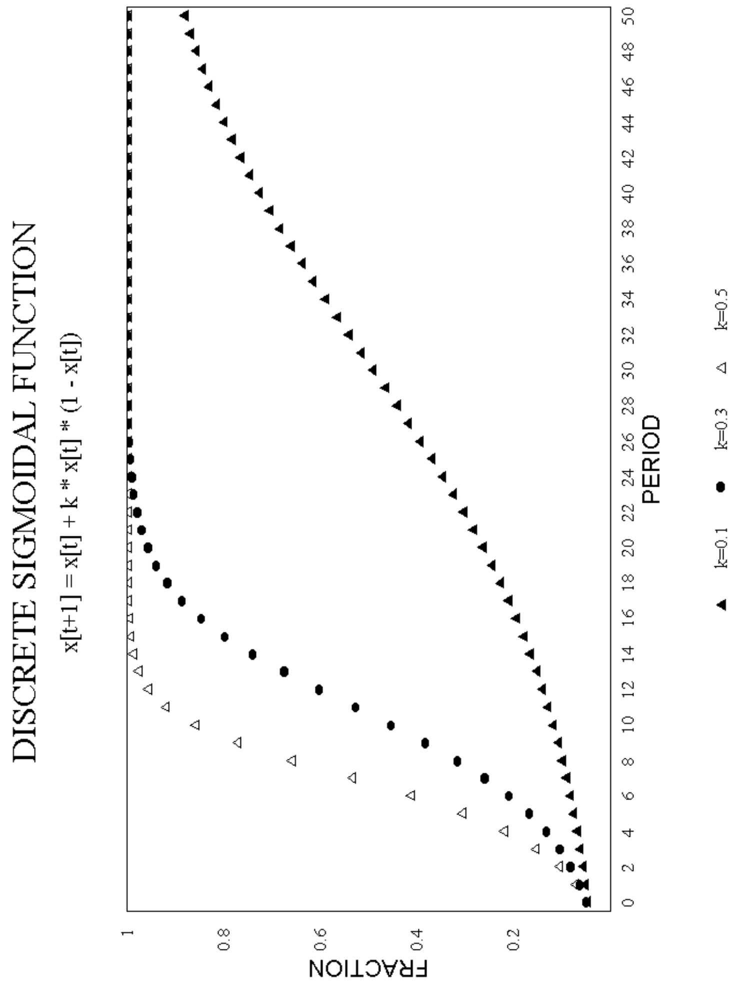


Figure 2-1-2. Shape of the discrete sigmoidal funtion defined by (2-1-11) for different values of the rate constant k.

Yet another application of growth curves is in forecasting market development (Makridakis, Wheelwright and McGee, 1983; Meade, 1984; Meade, 1985, and references in these articles). Some of these problems are included in the example data sets in Chapter 18.

2-2. Equilibrium Models in Sciences and Engineering

A number of natural phenomena describe the real world in some equilibrium state, with parameters controlling the position of the equilibrium point. We consider several examples.

The Van der Waals equation of state describes the relationship between Pressure (P), Temperature (T) and Volume (V) for m moles of a gas (Eggers et al., 1964, page 68). The equation is

(2-2-1) $(P + B(1) * m^2 / V^2) (V - B(2) * m) = mRT$

where R is the gas constant (8.3143 joules/ (degree mole)) and B(1) and B(2) are parameters. Given data for P, V, T, and m, we could estimate B(1) and B(2) (which are clearly involved nonlinearly in the model Equation (2-2-1)).

A closely related phenomenon is the viscosity of gases (Moelwyn-Hughes, 1961, Chapter 14).

Another physical property which may be modelled by nonlinear functions of some parameters is the dielectric constant. The study of the dielectric constant of certain polymers by Havriliak and Watts (1985) is a notable work in this area.

In engineering, Lo et al. (1982) use a nonlinear estimation technique to estimate the state of an electric power distribution system. We shall not present details of these three examples because of the considerable background information required to develop the related estimation

problems.

In the engineering of machine tools, such as lathes and milling machines, the wear on the cutting tool is dependent on the over-all distance the tool travels across the workpiece surface as well as the temperature generated by friction in cutting. The dependency on the materials from which the tool and workpiece are made will be ignored here; we will only consider models for a given pair of materials. For such a situation, F. W. Taylor developed a model in 1906 which uses two constants, B(1) and B(2), which will be referred to as the Taylor exponent and "cutting speed (in feet per minute) for 1 minute tool life", which are characteristic of the tool for a given pair of tool / workpiece materials (Schey, 1977, page 232). The model is

(2-2-2) $v * t^{B(1)} = B(2)$

where v is the cutting speed in feet per minute and t is the tool life (in minutes). Increasing the cutting speed lowers the tool life. The traditional solution for B(1) and B(2) is to plot v versus t on log-log graph paper. However, the relationship can be modelled directly if nonlinear estimation methods are available.

A more general mechanical phenomenon which is modelled by a nonlinear function is that of damped vibrations, which could also be thought of as a case of growth of decay phenomena. The form of such models (Beer and Johnston, 1972, page 864) is

(2-2-3) $y(t) = B(1) * \exp(-B(2)*t) * \sin(B(3)*t+B(4))$

where the notation has been altered to correspond to our form for nonlinear models. Such models are generally quite awkward to estimate, especially if there is any contamination of one frequency of vibration (embedded in the parameter B(3)) with some other vibrational mode.

2-3. Maximum Likelihood Estimation

Statistical estimation techniques presuppose rather more underlying structure for the errors between observed data and "true" values for the variables considered. If a probability distribution for these errors is known (or assumed), then maximum likelihood estimation may be used.

Let us first consider just one variable, Y, subject to error, and suppose that its values are modelled by

(2-3-1) $y_i = z(i,\underline{B},X) + error_i$

where z is some function of the parameters $\underline{B}$ and the other data X. If the errors are distributed according to the probability density function u(error) then the most probable error has a size which is found by maximizing u(error) with respect to error.

When we have M observations, we note that

(2-3-2) $error_i = z(i,\underline{B},X) - y_i$

and the joint probability density of observing all M observations is

(2-3-3) $$L(\underline{B},\underline{y},X) = \prod_{i=1}^{M} u(error_i)$$

(see Equation (1-1-3) for an example). Note that the errors are implicit functions of the parameters by virtue of (2-3-2).

By adjusting the parameters $\underline{B}$, the likelihood function L( ) can be maximized. Frequently the log-likelihood, that is, the logarithm of this function, is preferred, since the product in (2-3-3) is then converted to a sum, with potential algebraic simplifications of the overall problems. In this book we present algorithms for minimizing functions of several parameters. Therefore, we will minimize

(2-3-4) $-L(\underline{B},\underline{y},X)$ or $-\log L(\underline{B},\underline{y},X)$

with respect to $\underline{B}$.

Bard (1974) presents a fairly complicated study of maximum likelihood estimation of the Klein econometric model. Here we will look at a somewhat simpler example. This concerns the estimation of the mean and variance parameters of a Gaussian distribution (normal distribution) from sample data for M observations drawn from the population which is presumed to have normally distributed values y(i). The classical calculus solution for this estimation problem gives the parameters as

(2-3-5) $$B(1) = \text{mean} = \text{y-bar} = \left\{ \sum_{i=1}^{M} y(i) \right\} / M$$

(2-3-6) $$B(2) = \text{variance} = \text{sigma}^2 = \sum_{i=1}^{M} (y(i) - B(1))^2 / M$$

Note that Equation (2-3-6) gives an estimator which is biased, and the usual estimator divides the sum of squared deviations from the mean by (M-1). Having an analytical formula for the maximum likelihood estimates of mean and variance parameters provides a simple test of our algorithms. Using the definition of y-bar given in Equation (2-3-5), the objective function to be minimized is

(2-3-7) $$-\log L(\underline{B},\underline{y}) = -M * \log(\text{sqrt}(B(2)))$$
$$- \left\{ \sum_{i=1}^{M} (y(i) - \text{y-bar})^2 + M*(\text{y-bar} - B(1))^2 \right\} / (2*B(2))$$

2-4. Robust Estimation Techniques

Traditional estimation techniques, such as least squares or maximum likelihood, may be sensitive to situations where the assumptions about the error structure are not true. For example, it is common to assume that errors between observed data and true (but unknown) model follow a Gaussian distribution. In such cases, least squares and maximum likelihood estimation generally give rise to the same estimation calculations. When the Gaussian error assumption does not hold for all observations, maximum likelihood is no longer a useful technique unless the error distribution can be specified. Statisticians have therefore attempted to develop other techniques of estimation which do not rely on a distributional assumption. Such robust estimation techniques continue to be a topic of much research. Following Goodall (1983), we consider minimizing objective functions which are sums of functions of the residuals $r(i,\underline{B})$, $i = 1, 2, ..., m$.

For least squares, the function of the residual used is just the square of the residual, that is,

(2-4-1) $f = r^2$

The Huber (k) function modifies this to

(2-4-2) $f = r^2/2 \qquad |r| \leq k$
$\qquad = k\,|r| - k^2/2 \qquad |r| > k$

Tukey's biweight function uses

(2-4-3) $f = [1 - (1-r^2)^3] / 6 \qquad |r| \leq 1$
$\qquad = 1/6 \qquad |r| > 1$

Belsley, Kuh and Welsch (1980, page 232) show the use of the Huber (k) in estimating a housing-price equation.

2-5. Constraints Introducing Nonlinearity

While the subject of constraints will be dealt with in more depth later, it is important to note that apparently linear models may be made nonlinear by the introduction of constraints. Furthermore, some techniques for handling

constraints may introduce nonlinearity. As an example, consider a linear model Nash (1979, page 18) of the form

(2-5-1) $X(i,\underline{B}) = B(1) + B(2) * Y_{i2} + B(3) * Y_{i3} + B(4) * Y_{i4}$

which is to be fitted to the data in series

(2-5-2) Y_{i1}

subject to the constraint on the parameters $\underline{B}$ which requires

(2-5-3) $B(4) = B(2) * B(3)$

The revised model, found by substituting (2-5-3) in (2-5-1) is

(2-5-4) $Z(i,\underline{B}) = B(1) + B(2) * Y_{i2} + B(3) * Y_{i3}$
$\qquad + B(2) * B(3) * Y_{i4}$

which is now obviously nonlinear in the parameters. As discussed in Nash (1979, pages 184ff), this may yield a relatively difficult problem (Nash, 1976, Problem 35).

Another type of constraint is that the parameter B(k) is a proportion, that is, it must lie in the interval [0,1). This is clearly satisfied by the quantity

(2-5-5) $B(k) = b^2 / (1 + b^2)$

which has value zero at b = 0 and 1 as b tends to infinity in magnitude. Using b instead of B(k) as our parameter again introduces a nonlinearity. If such problems arise frequently, some method for constrained linear estimation (probably least squares) is desirable. For once-in-a-while problems, the substitution (2-5-4) may suffice along with an unconstrained function minimization method. Alternatively, the bounds zero and one could be prescribed in a gradient minimization approach as described in Section 7-3. An alternative substitution uses the square of the trigonometric sine or cosine function.

2-6. Loss Functions Which are Sums of Squares

A very large class of problems is that of nonlinear least squares (Problem NLLS, Section 1-7), where the loss function is a sum of squared residuals (Equation (1-1-5)). We do not say "sum of squared errors", since the "true" errors are unknown. However, the i-th residual $r(i,\underline{B})$ is the deviation between the observed data and its model (or calculated) value for the i-th observation. Frequently, the i-th observation is a vector of several independent and one dependent variable so that we may write

(2-6-1) $r(i,\underline{B}) = Z(Y(i,2),Y(1,3),...,\underline{B}) - Y(i,1)$

where Z( ) is the modeling function.

While some authors consider only this form of nonlinear least squares, we wish to stress two of its deficiencies:

1. it does not include forms which cannot be written as a difference between model and observation;

2. the traditional "observed minus model" form

(2-6-1a) $r(i,\underline{B}) = Y(i,1) - Z(Y(i,2),Y(1,3),...,\underline{B})$

introduces the potential for sign errors in evaluating derivatives of the residuals, as we have already noted in Section 2-1.

The derivatives of the residuals lead to the Jacobian matrix J.

(2-6-2) J_{ij} = partial derivative of $r(i,\underline{B})$ w.r.t. B(j)

Evaluating the Jacobian elements is an important step in many nonlinear least squares algorithms. These are frequently combined into the sum of squares and cross products (SSCP) matrix

$$\text{SSCP matrix} = J^T J$$

Particular problems with the sum of squares loss function center upon the possibility that one or more large residuals may be responsible for the bulk of the function

value. This is especially problematic if we are seeking to solve a nonlinear equations problem (NLE) by minimizing the residuals to (hopefully) zero. In solving equations, we clearly would like a true zero for the residual, and not "zero relative to some large number". One possibility in such cases is to scale the residuals, that is, instead of minimizing

(2-6-3) $f(\underline{B}) = \underline{r}^T(\underline{B})\ \underline{r}(\underline{B}) = \underline{r}^T\ \underline{r}$

we minimize

(2-6-4) $f1(B) = \underline{r}^T W^2\ \underline{r}$

where W^2 is some diagonal matrix with positive elements on the diagonal (squares of the weights).

The difficulty here is in assigning the weights in a way which does not essentially alter the problem by assigning an artificial importance, or lack thereof, to one or more of the observations/residuals.

The relationship between NLLS and FMIN can be cast into program code as the BASIC segment SUMSQR given below in Listing 2-6-1. Note that we compute the residuals one at a time in the scalar variable R1, a saving of (m-1) storage locations. Likewise, the whole (m x n) Jacobian matrix is not needed to form the gradient

(2-6-5) $\underline{g} = J^T\underline{r}(\underline{B})$

We compute the rows of J one at a time and accumulate them into $\underline{g}$, the gradient. This particular implementation could be more computationally efficient by saving the residuals in a vector $\underline{R}$. As things stand, we compute the residual in evaluating the sum of squares function AND in computing the gradient. If the residuals are computed BEFORE any attempt is made to accumulate the gradient, then the following changes will avoid the extra calculation at the expense of a storage vector R( ):

```
---- DIM R(M): REM EXTRA STORAGE
2065 LET R(I) = R1: REM SAVE RESIDUAL TO AVOID RECOMPUTING
2560 LET R1 = R(I): REM REPLACES GOSUB 3500
```

Note that vector R( ) must be dimensioned to length at least M.

Listing 2-6-1. Code segment SUMSQR.BAS to transform a sum of squared residuals into a general loss function.

```
              SUMSQR.BAS

 38 DIM D(50): REM for derivatives (row of Jacobian)
 39 LET J6=1: REM to display residuals for POSTGEN, POSTVM
2000 LET I3=0: REM set computability flag
2010 REM COMPUTATION OF SUM OF SQUARES LOSS FUNCTION FROM RESIDUALS
2020 REM CALLS:
2030 REM    RESIDUAL R1 -- line 3500
2040 REM INPUTS:
2050 REM    M   -- the number of data points
2060 REM OUTPUTS:
2070 REM    F   -- the value of the function
2080 REM    I3  -- flag for non-computability of function
2090 REM
2100 LET F=0
2110 FOR I=1 TO M
2120 GOSUB 3500: REM evaluate I-th residual
2130 IF I3<>0 THEN RETURN: REM function not computable if I3<>0
2140 LET F=F+R1*R1: REM calculate sum of squares
2150 NEXT I
2160 RETURN
2500 REM COMPUTATION OF GRADIENT FROM RESIDUALS AND JACOBIAN
2501 REM CALLS:
2502 REM    RESIDUAL R1  -- line 3500
2503 REM    JACOBIAN D() -- line 4000
2504 REM INPUTS:
2505 REM    M   -- the number of data points
2506 REM    N   -- the number of parameters
2507 REM OUTPUTS:
2508 REM    G() -- the value of the gradient (N elements)
2509 REM
2510 FOR J=1 TO N
2520 LET G(J)=0: REM initialize the gradient
2530 NEXT J
2540 FOR I=1 TO M: REM note that residual is evaluated first
2550 GOSUB 3500: REM evaluate I-th residual
2560 GOSUB 4000: REM evaluate Jacobian for I-th residual
2570 FOR J=1 TO N
2580 LET G(J)=G(J)+2*R1*D(J): REM calculate gradient
```

```
2590 NEXT J
2600 NEXT I
2610 RETURN

Line no.    Referenced in line(s)
 3500      2120  2550 -- compute I-th residual
 4000      2560 -- compute I-th row of Jacobian

 Symbol     Referenced in line(s)
D(N)         38  2580 -- vector for I-th row of Jacobian
F          2100  2140 -- sum of squared residuals (function)
G(N)       2520  2580 -- gradient vector
I          2110  2150  2540  2600 -- index for residuals
I3         2000  2130  -- failure flag (set to 1 on failure)
J          2510  2520  2530  2570  2580  2590 -- loop index
J6           39 -- set to 1 to cause residuals to be printed
           by post-solution analysis routines
M          2110  2540 -- number of residuals
N          2510  2570 -- number of parameters
R1         2140  2580 -- value of I-th residual
=============================================================
LINES: 41      BYTES: 1329     SYMBOLS: 12     REFERENCES: 28
```

We have not detailed how the weights W should be established in Equation (2-6-4). This task is not trivial since

- large weights may introduce mixtures of large and small numbers into the solution calculations, giving rise to a potential loss of information, or to slow convergence of the algorithms;

- weights which are too small may result in failure of "solutions" to satisfy the conditions which underly the introduction of weights.

In many instances, the choice of weights will be reasonably clear from the context of the problem at hand. In other cases, we may need to try several sets of choices and to analyze the results carefully. In all situations, the tentative solutions should be looked at only as candidates for acceptance for the solution of the original problem.

COVER SHEET

Chapter 3

Chapter title: Exploring the problem

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

3

EXPLORING THE PROBLEM

3-0. Overview -- Properties of Problems

In this chapter we discuss the ways in which a nonlinear parameter estimation problem should be set up and adjusted to avoid unnecessary difficulties. To this end, we introduce programs to test the function and derivative subroutines and the residual and Jacobian subroutines. While most problems can be resolved without resorting to adjustment of the problem, knowledge of the characteristics of the problem is an essential ingredient for efficient solution, and may be the key to resolving those particular cases which are beyond the normal limits of the hardware or software at hand.

Another aspect of resolving problems is the inclusion of those constraints, explicit or implicit, which attend ALL real problems. The inclusion of a general mechanism for all constraints is beyond the scope of the present work. Indeed, much research into such a general framework is ongoing as this is being written. However, a large variety of simple techniques exists for inclusion of the types of constraints which commonly arise, and we discuss some of these. Such constraint inclusion methods mainly involve changes to the function and derivative subroutines via the use of penalties or substitutions. However, bounds on parameters are so important a class of constraints that our methods include these directly as part of the method.

3-1. The Function and its Scaling

All the parameter estimation techniques to be discussed in this book are based on the concept of a loss function. That is, we assume that a "good" set of parameter values has a measurable or calculable value of some "badness" or loss function which is lower than a "bad" set of parameters. Note that estimation methods based on moments or other ideas may often be cast into this form as well. For example, suppose that we wish to equate various moments of some statistic to some "known" values. The parameters B(j), j=1,2,..., N, which describe the system in question must be estimated by making the moment functions Q equal to their values v, that is,

(3-1-1) $Q(k,\underline{B}) = v(k)$ for k = 1,2,...,limit-on-k

The limit-on-k may be assumed to be equal to N, giving a nonlinear equations problem (Problem NLE) or may be greater or less than N, giving respectively an overdetermined or underdetermined system of nonlinear equations for which solutions $\underline{B}$ may or may not exist. Since the differences

(3-1-2) $R(k,\underline{B}) = Q(k,\underline{B}) - v(k)$

form a set of residuals for a nonlinear least squares problem (NLLS), we can attempt a solution with a method to minimize a sum of squared functions. The user must, of course, accept the results of such a method only if satisfied that the final residuals are sufficiently small that equality may be presumed in Equation (3-1-1).

Deciding what constitutes "equality" is a difficult task

-- one which cannot be delegated to computer programs. It is all too easy to write down nonlinear estimation loss functions in a way which causes their values to be nearly equal for a large class of unacceptable or even inadmissible parameter sets. Indeed, it is common to observe that the natural expression of a model or loss function is a particularly poor way to code such functions for computer evaluation. While the central ideas of nonlinearity are discussed in Chapter 13, we feel the stability of computations important enough to devote some space at this early stage of the book to mechanisms by which poor scaling and rapid variation of the loss function may be discovered. Correction of scaling difficulties in the function is usually straightforward though tedious. However, discovering appropriate transformations of functions to remove non-essential nonlinearity requires a great deal of ingenuity in most cases.

Scaling of a function concerns both its domain -- the sets of parameters $\underline{B}$ for which the function may be evaluated -- and its range -- the possible values the function may take on. The most common failures or inefficiencies in nonlinear function minimization in general, and nonlinear parameter estimation in particular, are, apart from programming errors, usually related to poor scaling in either of these areas. Often we use sums of squared functions as a measure of goodness of fit or for the resolution of systems of nonlinear equations. The very process of squaring may lead to very large numbers or very small numbers.

In this discussion, we will use the general unconstrained function $f(\underline{B}) = f(B(1),B(2),...,B(N))$. Constraints may also be imposed on the parameters $\underline{B}$.

What are desirable features of a well-scaled function for the purposes of its minimization? In general we would like

- the function to vary smoothly with changes in the parameters (continuity and smoothness)
- the function to have reasonable values in a range which requires us to copy neither exponents nor leading or trailing zeroes (or digits with no significance)
- the function to have well-defined derivatives over the domain of parameters of interest and these (partial) derivatives to have roughly similar magnitudes with respect to each parameter
- the parameters to take on similar magnitudes over the domain of interest
- the domain of interest (or feasible region) to be easily defined.

The above criteria are not at all strictly necessary for successful solution of parameter estimation problems, but they help us to avoid some of the numerical difficulties and human errors which attend problems of the type we are trying to solve. We can modify functions to obtain some of the desirable characteristics above (removing the non-essential nonlinearities) but will not be able to avoid all the difficulties. The main tool for modification of the function is scaling.

First, since the loss function is to be observed as decreasing with the progress of our estimation methods, it should have a large RELATIVE variation over the domain of parameters of interest. A simple translation, that is,

(3-1-3) $f1(\underline{B}) = f(\underline{B}) - F6$

where F6 is some constant, will allow the common leading digits of the evaluated function to be removed. However, this is NOT the ideal way to evaluate our new loss function f1, since we have lost leading significant digits in the subtraction. A better approach is to perform the subtraction IMPLICITLY if this is possible. The example below illustrates this idea.

Second, it is useful to not have to carry too many digits or have to copy down exponents. The display of a large number of digits, most of which may have no meaning in the context of the problem at hand may lead users to believe the results of an estimation procedure are "accurate". The potential for human error increases with the amount of information to be read or copied. Thus we recommend that the function be scaled so that all magnitudes are between 0 and 100 for ease of viewing, plotting and transcribing. This is accomplished by the multiplier, F5, so

(3-1-4) $f2(\underline{B}) = F5 * f1(\underline{B}) = F5 * f(\underline{B}) - F6$

where the scaling factor F5 is most commonly a power of 10 or some other simple conversion factor which essentially defines the units in which the loss function is measured. Note that the components of the gradient of F2() are F5 times those of F().

We note also that F5 and F6 may be chosen so that the loss function is always positive, since otherwise its minimization results in the displayed magnitude (that is, absolute value) increasing, which may mislead us into thinking that the function is increasing. Since it is within our capability to arrange for the function to have a small magnitude and to decline in absolute value as it is minimized, we should take the opportunity to scale the function appropriately to so convenience ourselves.

Example of function scaling.

The test function NG1 (Nash, 1976, problem #34) is designed to have "large" residuals in nonlinear least squares calculations. This function is defined as the sum for i=1,2,...,20 of

(3-1-5) $r(i,\underline{B}) = (B(1) + i*B(2)/5 - \exp(i/5))^2 + (B(3) + B(4)*\sin(i/5) - \cos(i/5))$

The minimum value of the sum of squares is 85822 at the point

$\underline{B} \underline{\sim} (-11.594, 13.204, -0.40344, 0.23678)^T$

For nonlinear least squares problems, there is little which can be done to reduce the magnitude of the loss function other than a common scaling factor applied to the residuals. This scaling factor will be the square root of F5 above. For function minimization problems, subtracting F6=85800 will leave smaller numbers in the final iterations of algorithms. Of course, we will not always have at hand a reasonable approximation to the function minimum, though a great many practical problems arise as members of a series of similar problems, so that reasonable estimates are availble.

3-2. Including Constraints

We have already seen how constraints arise in nonlinear parameter estimation problems in Section 2-5. The task here is to show how the estimation problem may be formulated to incorporate these constraints. Moreover, the inclusion of the constraints must be carried out in such a way that our software is still able to operate correctly.

First, constraints will be classified as in Section 1-7, since the method of handling them depends largely on their nature and influence on the estimation problem. We shall follow many other authors and categorize constraints as equality or inequality constraints and as linear or nonlinear constraints.

Recall that an equality constraint is a functional relationship between the parameters $\underline{B}$ of the form

(3-2-1) $c_i(\underline{B},Y) = 0$ for $i = 1,2,...,m3$

where the index i is provided to allow for more than one constraint. A simple example of an equality constraint is the condition that the sum of our monthly expenditures and savings must equal our income and borrowings. Many practical problems involve conservation laws of this type leading to equality constraints.

An inequality constraint is a functional relationship between the parameters $\underline{B}$ of the form

(3-2-2) $c_j(\underline{B},Y) \geq 0$ for $j = m3+1,m3+2,...,m4$

where we have changed the index to j to indicate that the functions

c_i and c_j

are not the same. The use of the "greater than or equal" symbol $\geq$ in the constraint (3-2-2) is a matter of convenience. If it is required that the function be strictly greater than zero, then a new function

(3-2-3) $c\text{-new}_j(\underline{B},Y) = c_j(\underline{B},Y) - \text{epsilon}$

could be used, where epsilon is a very small number relative to the magnitudes involved in the function and constraints.

As with the loss function, we shall not bother to write down the data matrix "Y" except where it is needed for clarity (Section 0-3).

In principle, we can use the m3 equality constraints to remove m3 of the parameters from the problem. That is, we can solve for m3 of the parameters $\underline{B}$ in terms of the remaining (n-m3). Especially when the functions $c_i(\underline{B})$ are linear in the parameters, we essentially have a set of linear equations to solve, and then can substitute these results into the loss function to leave an unconstrained loss function in fewer parameters. For problems with only a few constraints, this explicit substitution avoids the creation of extra program code to handle the constraints. Many nonlinear parameter estimation problems have such a structure, and we recommend direct substitution to remove (or eliminate) equality constraints wherever possible. Where the constraint equations are nonlinear in the parameters, this direct approach is even more valuable in avoiding expensive program code (and often computation time).

When a large number of linear equality constraints are present, then the solution of the constraint equations is better carried out within the overall loss function minimization program. Following the discussion of Gill, Murray and Wright (1981, Section 5.1) we recommend various forms of the projected gradient method. That is, all matrices and vectors used in solving the minimization problem are projected into a feasible subset of the parameter space. Thus, only values of the parameters which satisfy the equality constraints can enter in the minimization. Various ways of projecting the vectors and matrices into the feasible subspace exist. Note that they require that the initial point $\underline{B}_0$ must be feasible, i.e. satisfy the constraints. Moreover, it is necessary to verify that there are no redundancies in the constraints before the projectors are constructed. Since we do not envisage many problems in parameter estimation having a large number of linear equality constraints, we have not included special code for the solution of these problems.

Inequality constraints generally give rise to greater difficulty. Ignored here are the special case of bounds and masks, which are treated specially in our codes, and discussed in Sections 7-3 and 15-2. Inequality constraints do not usually reduce the dimensionality of the estimation problems unless the solution lies on a constraint boundary. In such case the constraint is said to be active at the minimum. For example, the minimum of

(3-2-4) $B(1)^2$ subject to $B(1) \geq 1$

is clearly at $B(1) = 1$.

One approach to minimizing functions subject to inequality constraints is to add positive amounts to the loss function whenever the constraint is violated. Such penalty functions are of the form

(3-2-5) $c_j(\underline{B})^2 \ W_j \ H(c_j(\underline{B}))$ for $j = m3+1,\ldots,m4$

where W_j is some positive weight assigned to the inequality constraint indexed by j and H(x) is the Heaviside function

(3-2-6) $H(x) = 1$ if $x \geq 0$

$= 0$ if $x < 0$

Penalty functions can also be used in the case of equality constraints, but we do not recommend this practice, since the "feasible region" is so narrow, yielding a very inefficient minimization process. For inequality constraints which are inactive, the minimization process proceeds unhindered. Note that it is common to terminate the minimization on the infeasible side of an active constraint because the penalty functions allows the constraint boundary to be "soft" to the extent that W_j is not infinite. Increasing the weights introduces a potential descaling of the problem. Nevertheless, the Sequential Unconstrained Minimization Technique (SUMT) has had a great deal of use since its introduction by Fiacco and McCormick (see Fiacco and McCormick, 1968 for a development). Note that the penalized loss function remains differentiable with respect to the parameters.

Another penalty-like approach is the barrier function, which forces all acceptable test points to be within the feasible region. Gill, Murray and Wright (1983, Section 6.2) present a discussion of a variety of differentiable and non-differentiable forms for inequality constraints. Here we shall generally use such penalty and barrier techniques as sparingly as possible. Since we deal primarily with interactive computing environments, we will often prefer to ignore constraints when it is clear that in the region of interest they are not active. In some cases we may be able to transform our problem so that the constraint is automatically satisfied, or use simple bounds on the parameters to approximately satisfy the constraint. For problems of a one-time-only nature, a simple barrier approach to the constraint is useful in conjunction with a direct search minimization (Chapter 4). Here the loss function is assigned a very large value whenever the constraint is violated. The constrained loss function is thus non-differentiable, so its minimization cannot be attempted with a gradient method.

An example of a parameter estimation problem where the constraints are more than simple bounds is presented in Chapter 18.

3-3. Scaling of Parameters and Data

In general, we recommend that the values of the parameters $\underline{B}$ which are of interest should lie in a common range. Usually it is convenient to arrange that all the B's should be between 1.0 and 10.0. This can be done by simple scalings or translations, that is,

```
(3-3-1)     NEW-B(k) = Scaling(k) * B(k)
```

or

```
(3-3-2)     NEW-B(k) = Scaling(k) * B(k) - Shift(k)
```

In order to simplify the coding of scalings in any programs, we will use a single array S dimensioned N by 2, having the purpose

```
(3-3-3)     Scaling(k) = S(k,1)
            Shift(k)   = S(k,2)   for k = 1,2,...,N
```

Since it is useful to work with variables in their natural

units, we can then use the tranformations

```
(3-3-4)     SCALE:    B(k) := S(k,1) * B(k) - S(k,2)
(3-3-5)     DESCALE:  B(k) := (B(k) + S(k,2)) / S(k,1)
```

In practice, we prefer to scale and descale outside the programs and to learn to work with the scaled parameters. This allows us to avoid continual to-ing and fro-ing inside our programs. Following this general principle, none of the codes in this book will use the scaling and descaling mechanism of Equations (3-3-4) and (3-3-5). Instead, explicit scaling factors will appear in the problem program code.

Benefits of scaling the parameters arise in several ways. First, most methods for minimizing functions explicitly or implicitly combine the parameters in certain ways when "searching" for better parameter sets. For example, the Hooke and Jeeves method uses an axial search about the current "best" point (set of parameters). The Nelder - Mead polytope method calculates a centroid or average of a number of points. Most other methods we discuss use a form of line search along a vector $\underline{T}$ to find a new point

$$(3\text{-}3\text{-}6)\qquad \underline{\text{NEW-B}} = \underline{B} + \text{step} * \underline{T}$$

which may result in numbers of vastly different magnitudes being added together if the parameters are poorly scaled. When the parameters are indeed poorly scaled, the danger is that information contained in one of the parameters is "lost" as numbers with different magnitudes are combined. While many of the gradient methods are relatively resistant to this computational fault, it is quite obvious from the design and program code of the direct search methods (Hooke and Jeeves and Nelder - Mead) that such scaling is extremely important to their success.

Example: The Hobbs weed infestation problem.

The Hobbs weed infestation data presented in Table 2-1-1 are presumed to follow a logistic growth curve

```
      b(1)/(1+b(2)*exp(-i*b(3)))
```

where i is the index of the observation (year of growth). The parameters are presented in lower case to represent unscaled parameters -- that is, as the problem might be first written down. Upper case representation of the parameters will indicate the scaled form which is used in calculations. From ad hoc methods of estimating the parameters (see Section 3-9), we are given the information that the parameters in this model are approximately

```
       b(1)=200,  b(2)=50,  b(3)=-0.3
```

In this case, we do not bother with translations of the parameter values (i.e. S(k,2)=0 for k=1,2,3) but do use the simple "power of 10" scalings

```
       S(1,1)=0.01,  S(2,1)=0.1,  S(3,1)=10
```

so that our model is now

```
       100*B(1)/(1+0.1*B(2)*exp(-10*i*B(3)))
```

Listing 3-3-1 presents the program code used to carry out the estimation of the parameters in this model. This includes the code to supply the data and to evaluate the residuals and derivatives for the problem.

Listing 3-3-1. HOBBS.RES program code

```
             HOBBS.RES           04-17-1986   20:31:49

 30 DIM Y(12,2),B(3),X(3),O(3,3)
 3000 PRINT "HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017"
 3010 PRINT #3,"HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017"
 3020 LET P$="HOBBS.RES 3 parameter logistic fit"
 3030 LET M=12: REM 12 data points
 3040 LET N=3: REM 3 parameters
 3050 REM note that we DIM Y(12,2) to allow residuals to be saved
3060 RESTORE: REM reset data pointer
3070 FOR I=1 TO M
3080 READ Y(I,1): REM weed growth at time point I
3090 NEXT I
3100 REM now set the bounds on the parameters
3110 LET O(1,1)=0: REM positive asymptote
3120 LET O(1,2)=100: REM loose upper bound
3130 LET O(1,3)=1: REM not masked
3140 LET O(2,1)=0: REM positive to avoid zero divide
3150 LET O(2,2)=100: REM loose bounds on second parameter
3160 LET O(2,3)=1: REM not masked
3170 LET O(3,1)=0: REM positive exponential parameter
3180 LET O(3,2)=30: REM upper bound & test on exponential argument
3190 LET O(3,3)=1: REM not masked
3200 PRINT " FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(2)*I))"
3210 PRINT #3," FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(2)*I))"
3220 PRINT
3230 PRINT #3,
3240 RETURN
3250 DATA 5.308, 7.24, 9.638, 12.866, 17.069
3260 DATA 23.192, 31.443, 38.558, 50.156, 62.948
3270 DATA 75.995, 91.972
3500 LET I3=0: REM start with computable function
3510 IF ABS(.1*I*B(3))>50 THEN 3550: REM not computable if exponent big
3520 LET R1=100*B(1)/(1+10*B(2)*EXP(-.1*B(3)*I))-Y(I,1)
3530 LET Y(I,2)=R1: REM save the residual value
3540 RETURN
3550 LET I3=1: REM cannot compute the residual
3560 RETURN
4000 LET D(1)=100/(1+10*B(2)*EXP(-.1*B(3)*I)): REM Hobbs
4010 LET D(2)=-B(1)*.1*D(1)*D(1)*EXP(-.1*B(3)*I)
4020 LET D(3)=-.1*D(2)*B(2)*I
4030 RETURN

    Line no.    Referenced in line(s)
     3550      3510

     Symbol     Referenced in line(s)
    B(           30  3510  3520  4000  4010  4020 -- parameters
    D(         4000  4010  4020 -- row I of Jacobian
    I          3070  3080  3090  3510  3520  3530  4000  4010  4020
               -- counter for residuals
    I3         3500  3550 -- flag set to 1 if residual not computable
    M          3030  3070 -- number of residuals
    N          3040 -- number of parameters
    O(           30  3110  3120  3130  3140  3150  3160  3170  3180
               3190 -- masks and bounds array
    P$         3020 -- problem name
    R1         3520  3530 -- value of residual I
    X(           30 -- value of "best" parameters
Y(          30  3080  3520  3530 -- data array
=================================================================
LINES: 40      BYTES: 1631     SYMBOLS: 12     REFERENCES: 42
```

Even from the viewpoint of the gradient methods, vastly different magnitudes of the parameters will give rise to difficulties in the calculation of gradients or in the approximation of the Hessian (that is, second derivatives) via finite-difference approximation. Here we are attempting to replace the analytic expression for a derivative by some approximating number

(3-3-7) Approx. Derivative = (f($\underline{B}$+h*$\underline{T}$) - f($\underline{B}$)) / h

for some choice of h and $\underline{T}$, a search vector of unit length. The difficulty is that h must be chosen so that it is scaled to the size of the parameters -- that is, to make some fractional change in the parameters -- and yet reflect the possibility that one or more of the parameters may have a zero value at the minimum of the function.

When the search vector $\underline{T}$ is along a parameter axis, a simple strategy such as

(3-3-8) h = tol * (ABS(B(k)) + tol)

(Nash, 1979, Chapter 18, page 179) for a derivative with respect to the k'th parameter, using a fraction tol to set the relative size of the step, works reasonably well. However, when the derivative is needed along a search direction which is not a parameter axis, we must be much more careful. Such caution is most easily concentrated in an attempt to scale the parameters, rather than in an attempt to correct for scaling deficiencies within function or derivative program code.

3-4. Nonlinear Transformations

For nonlinear parameter estimation methods which make use of the partial derivatives of the loss function, scaling is especially important in the evaluation of these derivatives. As we have just discussed, this scaling is concentrated mainly in the magnitudes of the parameters $\underline{B}$ with a secondary measure available via the function (as in Section 3-1).

The derivatives themselves cannot be scaled directly without transforming the function, but <u>linear</u> transformations of the parameters $\underline{B}$ and function f($\underline{B}$) can be of value in overcoming some of the difficulties which arise in practical computations.

If the nonlinear estimation practitioner is prepared to transform the function, then it may be possible to remove the nonlinearity in the problem. The minimum (or more precisely stationary point) of a function whose gradient is linear in the parameters can be found by solving a single set of linear equations to set the gradient to zero. For both linear and nonlinear minimization problems, the first order conditions for a minimum are that the gradient be zero, that is,

(3-4-1) $\underline{g}(\underline{B}) = \underline{0}$

If the loss function is quadratic in the parameters $\underline{B}$, the gradient $\underline{g}$ is linear in $\underline{B}$. The second derivatives of such a function are all constants. That is, the gradient has no curvature with respect to $\underline{B}$, so that the solution of (3-4-1) is straightforward. Transformations have been a popular method for overcoming nonlinearities because they simplify the solution process (see, for example, Draper and Smith, 1981, Chapter 5, or Johnston, 1972, Chapter 3). However, it is important to realize that the transformation process may change the problem which is solved. For example, the Cobb-Douglas production function in economics (Intrilligator, 1978, pages 266ff) is written

$$Z(i,Y,\underline{B}) = Y(i,1)^{B(1)} * Y(i,2)^{B(2)} * Y(i,3)^{B(3)}$$

which is transformed by logarithms to a linear form. However, minimizing the sum of squared deviations between the logarithm of this model and the logarithm of observed data it is supposed to represent implies that we believe that the error in the model is multiplicative in nature. To minimize the deviations between Z( ) and the observed data requires that we solve a nonlinear set of equations. The estimates obtained by solving the log-transformed problem may be greatly different from those found by a nonlinear least squares estimation. We have encountered one case where the signs of one estimated parameter was different under the two approaches to its estimation.

3-5. A Simple Example

Nonlinear estimation not only implies that the gradient has curvature with respect to the parameters. The difficulties which may be encountered in minimizing the functions considered in this book arise because the gradient may vary widely in magnitude over the parameter space of interest. We shall illustrate this with a very simple example,

(3-5-1) $f(B) = [\log (2 * B)]^2$

which is plotted, together with its first derivative (gradient) and second derivative (Hessian) in Figure 3-5-1. Clearly, we can reparametrize this problem using

(3-5-2) $Q = \log (2B)$

to obtain a very simple parabola with obvious minimum at Q = 0 or B = 0.5. The luxury of such a reparametrization is seldom available, and it is worth noting how the nonlinearity, together with the bound

(3-5-3) $B > 0$

imposed because log() is not defined for non-positive arguments, creates a relatively "difficult" minimization problem.

Anticipating Chapter 7, we will use the Newton iteration

(3-5-4) $\text{B-new} = B - f'(B) / f''(B)$
$= B * [1 - 2*\log(2*B)] / [1 - \log(2*B)]$

It is easy to show that this iteration is not defined when B = 0.5 * exp(1), and diverges away from the solution for B greater than this value. Worse, for values of B

(3-5-5) $0.5 * \exp(0.5) < B < 0.5 * \exp(1)$
$(\underset{\sim}{}0.8244) \qquad (\underset{\sim}{}1.3591)$

the next iterate, B-new, is less than zero. Only for

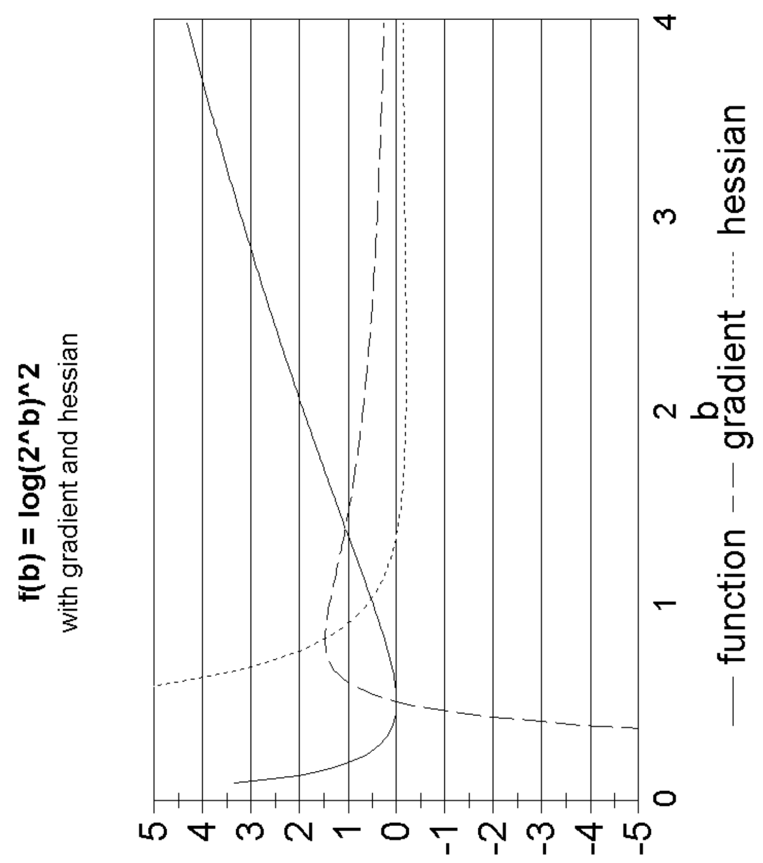
(3-5-6) $0 < B < 0.5 * \exp(0.5) \underset{\sim}{} 0.8244$

is the iteration well-defined and convergent. In later chapters, we shall show ways in which this region may be enlarged by changing the iteration algorithm (in the variable metric, conjugate gradients, truncated Newton, and Marquardt methods), that is, by altering what is accomplished by Equation (3-5-4).

Scaling the parameters can do nothing to alter the fundamental difficulty posed by the narrowness of the convergence region in the example above. Whatever scaling is applied will only change the numerical values of the limits on B which lead to a convergent iteration. Nevertheless, given the nature of finite precision arithmetic, and the human tendency to start many iterations with starting values of 1.0, scaling may "stretch out" a region wherein the iteration is convergent.

In general, in this book we shall not transform the functions which are to be minimized to estimate nonlinear parameters. It is our opinion that users will prefer to work with familiar functional forms and that well-chosen transformations, while extremely effective in resolving difficulties, generally demand a considerable human effort to apply. While this situation will probably change as symbolic manipulation becomes more widely available, for the present we will concentrate on improving our algorithms so that they may cope, albeit clumsily, with difficult problems in their untransformed state.

(opposite)
Figure 3-5-1. Plots of the function (3-5-1) and its derivatives. [Note: This is a combination of the graphs in the original 1987 edition.]



3-6. A Framework for Parameter Estimation

The discussion so far has concerned the scaling and the properties of the loss functions used to estimate parameters in models of phenomena or in curves to fit data. It is appropriate at this point to set out the framework within which parameter estimation problems will be solved. This framework will be reflected in programs to be presented later. We will in a loose sense follow this framework in establishing programs, and in Chapter 15 a batch procedure will be presented to tie the parts together (on MS-DOS machines, at least). We do not believe that such integration can be provided without considerable interaction with operating system software. While it is our hope that users will find our programs suitable for operation on a range of computers, we will provide a consolidation procedure only for the IBM PC and similar computer systems.

The structure we envisage for parameter estimation problems is as follows.

1. Setup
   - data collection, validation by humans, entry
   - data checking by machines, edit, scaling
   - choice of modeling loss function and constraints
   - coding of modeling loss function
   - scaling of modeling loss function
   - setting of bounds on parameters

2. Choice of method
   - direct search if derivatives not meaningful
   - coding of derivative calculations
   - testing of derivative calculation code
   - input of method-dependent information (stepsizes, tolerances)

3. Finding parameters
   - initial parameter guesses
   - solution phase
   - restart from different initial parameters

4. Post-solution analysis
   - computation of measures of dispersion
   - reporting of loss function and/or residual values
   - reporting of measures of effort for solution method(s) used
   - rescaling of parameters (this must be carried out explicitly by the user in our codes)
   - calculations particular to the problem at hand required by the user

In Listing 3-3-1 we have already shown the program code which must be provided by the user for the Hobbs problem. This example may be considered a prototype of code for solving estimation problems under the structure outlined above. Indeed, we frequently use HOBBS.RES or other problem codes as a starting point when preparing code for a new problem.

3-7. Testing Derivative Calculations

We have mentioned that the user must provide program code for the calculation of the partial derivatives (that is, the gradient) of $f(\underline{B})$ with respect to the B(j), j=1,1,....n. Numerical approximations can be used, and for one-time use may save the human effort, so we provide subroutines (Section 7-5) for this purpose. However, for loss functions which arise frequently, it is usually worthwhile computing the derivatives from analytic expressions. Unfortunately, the human errors made in such calculations are a frequent cause of overall failure to find a solution or of inefficient or incorrect operation of the methods employed.

Here we present a program to test derivative calculations by comparing the results of program code based on the analytic expressions for the gradient with numerical approximations based on forward difference approximations.

The forward difference approximation to the j'th component of the gradient $\underline{g}(\underline{B})$ (the partial derivative of $f(\underline{B})$ with respect to B(j)) is

(3-7-1) $g(j,\underline{B}) \underset{\sim}{} [f(\underline{B}') - f(\underline{B})]/h$

where

(3-7-2) $B'(i) = B(i) + h$ if $i = j$
$= B(i)$ otherwise,

and h is a stepsize parameter.

The forward difference approximation (3-7-1) may be a poor approximation to the true gradient component:

1. if the stepsize h is too large, the straight-line between $(\underline{B},f(\underline{B}))$ and $(\underline{B}',f(\underline{B}'))$ is not a good approximation to the surface of f between these points.

2. if the stepsize h is too small, the subtraction $[f(\underline{B}') - f(\underline{B})]$ may result in extensive digit cancellation which may similarly compromise the gradient approximation.

More discussion on this subject can be found in Froberg (1965, chapter 9), Gill, Murray and Wright (1981, section 8.6) or Nash (1979, section 18.2).

In the program FGTEST below, we use several values for the stepsize h. Since the parameters $\underline{B}$ have supposedly been scaled to have a reasonable magnitude, for example, between 0.1 and 10 in absolute value, absolute values of h of 1E-2, 1E-4, 1E-6, 1E-8 and 1E-10 are used in the forward difference approximation to allow stepsize effects to be observed. In addition, a flexible stepsize is also applied which depends

on the parameter value. This flexible step corresponds to the step-size used in the subroutines to compute derivatives by numerical approximation. Therefore, if it is known that the analytic expressions for derivatives are correct, FGTEST (and the corresponding code RJTEST for Jacobian element testing) can be used to evaluate the success of numerical approximations for different sets of parameters. When computing the j'th gradient component, we use

(3-7-3) h = eps * (ABS(B(J)) + eps)

where ABS() is the absolute value or modulus function and eps is the square root of the machine precision (see Appendix E).

FGTEST and RJTEST reset any parameter outside of the bounds to the nearest bound (unless it is masked). Masks must be set outside of the main program in the user-supplied problem code (see Section 15-3). If the function is not computable at the specified set of parameters (point in the parameter space), the program lets the user choose a new point. If the addition of the step to a parameter, i.e. B'(J), makes the function not computable at the new point, the numerical derivative for parameter J is set to -9999.

The output program FGTEST is illustrated in Listing 3-7-1 for the Hobbs function, using <u>unscaled</u> parameters to illustrate the difficulties of numerically approximating derivatives. Listing 3-7-2 shows the results for a scaled function at the equivalent evaluation point. To obtain a loss function from the Hobbs residuals, we have augmented the program code HOBBS.RES (Listing 3-3-1) with the code SUMSQR (Listing 2-6-1). Listing 3-7-3 gives the program code and cross-reference tables for FGTEST.

Listing 3-7-1. Analytic and numerical derivative calculations for the unscaled Hobbs problem.

```
FGTEST -- FUNCTION AND DERIVATIVE TESTER - 19851104
PROBLEM: HOBBU.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   10000
-100  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 200
B( 2 )= 50
B( 3 )= .3

FUNCTION =  158.2325

GRADIENT AND APPROXIMATIONS

 ANALYTIC   H=10^(- 2 ) H=10^(- 4 ) H=10^(- 6 ) H=10^(- 8 ) FLEXISTEP

-1.784E+01 -1.783E+01  -1.923E+01   0.000E+00   0.000E+00  -1.749E+01
 4.825E+01  4.827E+01   4.791E+01   0.000E+00   0.000E+00   4.880E+01
-2.456E+04 -1.495E+04  -2.447E+04  -2.499E+04   0.000E+00  -2.358E+04

NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)
        WHERE E8 =  10 *SQRT(MACHINE PRECISION)

ANOTHER TEST POINT ?Y
B( 1 )= 100
B( 2 )= 10
B( 3 )= .1

FUNCTION =  10685.29

GRADIENT AND APPROXIMATIONS

 ANALYTIC   H=10^(- 2 ) H=10^(- 4 ) H=10^(- 6 ) H=10^(- 8 ) FLEXISTEP

-1.009E+02 -1.009E+02  -1.074E+02   0.000E+00   0.000E+00  -1.008E+02
 7.835E+02  7.831E+02   7.715E+02   9.766E+02   0.000E+00   7.822E+02
-8.234E+04 -8.286E+04  -8.233E+04  -8.301E+04  -9.766E+04  -8.236E+04

NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)
        WHERE E8 =  10 *SQRT(MACHINE PRECISION)

        --------------------------------------------------
```

Listing 3-7-2. Analytic and numerical derivative calculations for the scaled Hobbs problem.

```
FGTEST -- FUNCTION AND DERIVATIVE TESTER - 19851104
PROBLEM: HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
-100  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3

FUNCTION =  158.2325

GRADIENT AND APPROXIMATIONS

 ANALYTIC   H=10^(- 2 ) H=10^(- 4 ) H=10^(- 6 ) H=10^(- 8 ) FLEXISTEP

-1.784E+03 -1.733E+03  -1.783E+03  -1.709E+03   0.000E+00  -1.749E+03
 4.825E+02  4.857E+02   4.813E+02   3.510E+02   0.000E+00   4.880E+02
-2.456E+03 -2.363E+03  -2.454E+03  -2.594E+03   0.000E+00  -2.359E+03

NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)
        WHERE E8 =  10 *SQRT(MACHINE PRECISION)

ANOTHER TEST POINT ?Y
B( 1 )= 1
B( 2 )= 1
B( 3 )= 1

FUNCTION =  10685.29

GRADIENT AND APPROXIMATIONS

 ANALYTIC   H=10^(- 2 ) H=10^(- 4 ) H=10^(- 6 ) H=10^(- 8 ) FLEXISTEP

-1.009E+04 -1.006E+04  -1.009E+04  -9.766E+03   0.000E+00  -1.008E+04
 7.835E+03  7.798E+03   7.832E+03   6.836E+03   0.000E+00   7.822E+03
-8.234E+03 -8.240E+03  -8.232E+03  -8.789E+03   0.000E+00  -8.237E+03

NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)
        WHERE E8 =  10 *SQRT(MACHINE PRECISION)

        --------------------------------------------------
```

Listing 3-7-3. FGTEST - a program to test function and derivative sub-programs.

```
         FGTEST.BAS        05-18-1986   20:11:38

     10 REM DEFDBL A-H,N-Z: REM !! double precision
     20 INPUT "CONSOLE IMAGE FILE ([CR] = NUL) ";F$
     30 REM to accommodate dimensions from problem file
     40 REM
     50 IF LEN(F$)=0 THEN LET F$="NUL"
     60 OPEN F$ FOR OUTPUT AS 3
     70 PRINT "FGTEST -- FUNCTION AND DERIVATIVE TESTER - 19851104"
     80 PRINT #3,"FGTEST -- FUNCTION AND DERIVATIVE TESTER - 19851104"
     90 REM
     100 DIM G(25),T(25),W(25,5)
     110 REM CALLS:
     120 REM     FUNCTION F(B) -- line 2000
     130 REM     GRADIENT G(B) -- line 2500
     140 REM     SETUP         -- line 3000
     150 REM     ENVRON (COMPUTING ENVIRONMENT) -- line 7120
     160 REM
     170 REM INPUTS TO THE PROGRAM:
     180 REM    B() -- parameter values where gradient will be tested
     190 REM    N   -- the number of parameters in the function F(B)
     200 REM    O( , ) -- masks and bounds
     210 REM OUTPUT FROM THE PROGRAM:
     220 REM    F0  -- function value at initial parameters
     230 REM    G() -- computed analytic gradient
     240 REM    W(,)-- numerical gradients
     250 REM
     260 GOSUB 7120: REM computing environment for E9
     270 LET E8=E5*SQR(E9): REM basic stepsize parameter
     280 GOSUB 3000: REM setup
     290 PRINT "PROBLEM: ";P$
     300 PRINT #3,"PROBLEM: ";P$
     310 PRINT "ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";
     320 INPUT X$
     330 PRINT #3,"ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";X$
     340 IF X$="N" THEN 450
     350 IF X$="n" THEN 450
     360 GOSUB 1210: REM display bounds information
     370 FOR J=1 TO N
     380 PRINT "B(";J;")=";
     390 INPUT B(J)
     400 PRINT #3,"B(";J;")=";B(J)
     410 NEXT J
     420 PRINT
     430 PRINT #3,
     440 GOSUB 1320: REM  check parameters within bounds
     450 FOR I=1 TO N
```

```
460 LET T(I)=B(I): REM save parameter I
470 NEXT I
480 LET I3=0: REM flag non-computable function with I3=1
490 GOSUB 2000: REM compute function
500 IF I3=0 THEN 540: REM check if in feasible region
510 PRINT "FUNCTION IS NOT COMPUTABLE"
520 PRINT #3,"FUNCTION IS NOT COMPUTABLE"
530 GOTO 310
540 PRINT "FUNCTION = ";F
550 PRINT #3,"FUNCTION = ";F
560 PRINT
570 PRINT #3,
580 LET F0=F: REM save this value
590 GOSUB 2500: REM compute derivatives in G()
600 FOR J=1 TO N: REM loop over gradient components
610 IF O(J,3)=0 THEN 750: REM masked
620 LET H=.01: REM initial stepsize
630 FOR I1=1 TO 5: REM loop over stepsize values
640 LET B(J)=T(J)+H
650 GOSUB 2000: REM compute function at new point
660 IF I3=0 THEN 700
670 LET W(J,I1)=-9999: REM flag non-computable at new point
680 LET I3=0: REM reset flag
690 GOTO 710
700 LET W(J,I1)=(F-F0)/H: REM compute derivative
710 LET B(J)=T(J): REM reset function
720 LET H=H*.01: REM adjust stepsize downwards
730 IF I1>3 THEN LET H=E8*(ABS(B(J))+E8): REM flexistep
740 NEXT I1
750 NEXT J
760 PRINT "GRADIENT AND APPROXIMATIONS"
770 PRINT
780 PRINT #3,"GRADIENT AND APPROXIMATIONS"
790 PRINT #3,
800 PRINT "ANALYTIC";
810 PRINT #3,"ANALYTIC";
820 FOR I1=1 TO 4
830 PRINT TAB(12*I1) "H=10^(-";2*I1;")";
840 PRINT #3,TAB(12*I1) "H=10^(-";2*I1;")";
850 NEXT I1
860 PRINT TAB(60) "FLEXISTEP"
870 PRINT
880 PRINT #3,TAB(60) "FLEXISTEP"
890 PRINT #3,
900 FOR J=1 TO N
910 IF O(J,3)<>0 THEN 960
920 PRINT "*** B(";J;") MASKED AT ";B(J)
930 PRINT #3,"*** B(";J;") MASKED AT ";B(J)
940 GOTO 1070
950 REM PRINT G(J);: REM standard print
```

```
 960 PRINT USING "##.###^^^^";G(J);: REM !!
 970 PRINT #3,USING "##.###^^^^";G(J);: REM !!
 980 FOR I1=1 TO 5
 990 PRINT TAB(12*I1);
 1000 PRINT #3,TAB(12*I1);
 1010 REM PRINT W(J,I1);: REM standard print
 1020 PRINT USING "##.###^^^^"; W(J,I1);: REM !!
 1030 PRINT #3,USING "##.###^^^^"; W(J,I1);: REM !!
 1040 NEXT I1
 1050 PRINT
 1060 PRINT #3,
 1070 NEXT J
 1080 PRINT
 1090 PRINT "NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)"
 1100 PRINT "        WHERE E8 = ";E5;"*SQRT(MACHINE PRECISION)"
 1110 PRINT
 1120 INPUT "ANOTHER TEST POINT ?";X$
 1130 PRINT #3,
1140 PRINT #3,"NOTE -- FLEXISTEP USES  STEPSIZE = E8*(ABS(B(J))+E8)"
 1150 PRINT #3,"        WHERE E8 = ";E5;"*SQRT(MACHINE PRECISION)"
 1160 PRINT #3,
 1170 PRINT #3,"ANOTHER TEST POINT ?";X$
 1180 IF X$="Y" THEN 360
 1190 IF X$="y" THEN 360
 1200 STOP
 1210 PRINT "bounds on parameters for problem"
 1220 PRINT #3,"bounds on parameters for problem"
 1230 FOR J=1 TO N
 1240 IF O(J,3)=0 THEN 1280
 1250 PRINT O(J,1);" <=   b(";J;")  <=  ";O(J,2)
 1260 PRINT #3,O(J,1);" <=   b(";J;")  <=  ";O(J,2)
 1270 GOTO 1300
 1280 PRINT " ****  b(";J;") WILL BE MASKED (FIXED) ****"
 1290 PRINT #3," ****  b(";J;") WILL BE MASKED (FIXED) ****"
 1300 NEXT J
 1310 RETURN
 1320 REM check bounds and set up
 1330 FOR J=1 TO N
 1340 IF O(J,3)=0 THEN  1450
 1350 IF B(J)>O(J,1) THEN  1400
 1360 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
 1370 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
 1380 LET B(J)=O(J,1)
 1390 LET O(J,3)=-2: REM lower bound active
 1400 IF B(J)<O(J,2) THEN 1450
 1410 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
 1420 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
 1430 LET B(J)=O(J,2)
 1440 LET O(J,3)=-1: REM upper bound active
 1450 NEXT J
```

```
 1460 RETURN

 Line no.    Referenced in line(s)
   310       530
   360      1180  1190
   450       340   350
   540       500
   700       660
   710       690
   750       610
   960       910
  1070       940
  1210       360
  1280      1240
  1300      1270
  1320       440
  1400      1350
  1450      1340  1400
  2000       490   650 -- subroutine to calculate the loss function
  2500       590 -- subroutine to calculate gradient
  3000       280 -- subroutine to set up problem
  7120       260 -- computing environment subroutine

 Symbol     Referenced in line(s)
B(          390   400   460   640   710   730   920   930  1350  1360
           1370  1380  1400  1410  1420  1430 -- the parameter vector
E5         270  1100  1150 -- a value used for scaled comparisons (= 10)
E8          270   730 -- the basic stepsize parameter
E9          270 -- the machine precision
F           540   550   580   700 -- a temporary function value
F$           20    50    60 -- the file for console image
F0          580   700 -- the loss function value
G(          100   960   970 -- the gradient
H          620   640   700   720   730 -- the stepsize for the numerical
            approximation to the gradient
I           450   460   470 -- a loop control counter
I1          630   670   700   730   740   820   830   840   850
            980   990  1000  1020  1030  1040 -- a loop control counter
I3          480   500   660   680 -- flag for function not computable
J           370   380   390   400   410   600   610   640   670
            700   710   730   750   900   910   920   930   960
            970  1020  1030  1070  1230  1240  1250  1260  1280
           1290  1300  1330  1340  1350  1360  1370  1380  1390
           1400  1410  1420  1430  1440  1450 -- a loop control counter
N           370   450   600   900  1230  1330 -- number of parameters
           in loss function
O(          610   910  1240  1250  1260  1340  1350  1360  1370
           1380  1390  1400  1410  1420  1430  1440 -- bounds and
           masks information storage
P$          290   300 -- the problem name
```

```
    T(         100   460   640   710 -- temporary storage of the
               parameter vector
    W(         100   670   700  1020  1030 -- the numerical
               approximations to the gradient
    X$         320   330   340   350  1120  1170  1180  1190 --
                temporary user input
=======================================================================
    LINES: 146      BYTES: 4760     SYMBOLS: 38     REFERENCES: 167
```

3-8. Testing Residual and Jacobian Calculations

Because nonlinear least squares problems are such a large class within the general area of nonlinear parameter estimation, we include the following driver program RJTEST in Listing 3-8-3 to test code written to generate residuals and their derivatives, that is, the elements of the Jacobian matrix. Because 5n estimates of the Jacobian are made for each of the m residuals, some additional diagonistics have been added to aid the user as compared to FGTEST (see Section 3-7). If either the relative deviation of the interpolation between the two best fits of the numerical approximations to the analytical Jacobian or one quarter of the relative deviation of either of the best fits exceeds a user-specified tolerance, the appropriate output is marked by two stars and the bell is rung.

Listing 3-8-3. RJTEST - a program to test residual and Jacobian sub-programs.

```
              RJTEST.BAS        05-18-1986   20:12:18

     10 REM DEFDBL A-H,N-Z: REM !! double precision
     15 INPUT "CONSOLE IMAGE FILE ([CR] = NUL) ";F$
     20 IF LEN(F$)=0 THEN LET F$="NUL"
     25 OPEN F$ FOR OUTPUT AS 3
     30 REM dimensions of y,x,b,o are merged in here from problem
     35 REM
     40 REM
     45 PRINT "RJTEST -- RESIDUAL AND JACOBIAN TESTER - 19851104"
     50 PRINT #3,"RJTEST -- RESIDUAL AND JACOBIAN TESTER - 19851104"
```

```
55 REM CALLS:
60 DIM D(25),T(25),W(25,5),S(6)
65 REM     SETUP         -- line 3000
70 REM     RESIDUAL      -- line 3500
75 REM     JACOBIAN D(G) -- line 4000
80 REM     ENVRON (COMPUTING ENVIRONMENT) -- line 7120
85 REM
90 REM INPUTS TO THE PROGRAM:
95 REM    B() -- parameter values where Jacobian will be tested
100 REM    M   -- number of data points
105 REM    N   -- the number of parameters in the function F(B)
110 REM    T7  -- tolerance for relative deviation of derivative
115 REM    O( , ) -- masks and bounds
120 REM OUTPUT FROM THE PROGRAM:
125 REM    R0  -- residual value at initial parameters
130 REM    D() -- computed analytic Jacobian
135 REM    W(,)-- numerical Jacobians
140 REM    S9  -- the sum of the squared residuals
145 REM    S(6)-- a flag that the relative deviation of the derivative
150 REM           is unacceptable
155 REM
160 GOSUB 7120: REM computing environment for E9
165 LET E8=E5*SQR(E9)
170 PRINT "Enter a tolerance for acceptable relative deviation between"
175 PRINT " analytic derivative and nearest approximation (e.g. 0.005)"
180 INPUT " tolerance = ";T7
185 PRINT #3,"Enter a tolerance for acceptable relative deviation between"
190 PRINT #3," analytic derivative and nearest approximation (e.g. 0.005)"
195 PRINT #3," tolerance = ";T7
200 LET M=0: REM set number of data points in case not initialized
205 GOSUB 3000: REM setup
210 IF M>0 THEN 220
215 INPUT "number of data points=";M
220 PRINT "ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";
225 INPUT X$
230 PRINT #3,"ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";X$
235 IF X$="N" THEN 285
240 IF X$="n" THEN 285
245 GOSUB 785: REM display bounds information
250 FOR J=1 TO N
255 PRINT "B(";J;")=";
260 INPUT B(J)
265 PRINT #3,"B(";J;")=";B(J)
270 NEXT J
275 PRINT #3,
280 GOSUB 840: REM check parameters are within bounds
285 FOR I=1 TO N
290 LET T(I)=B(I): REM save parameter I
295 NEXT I
300 LET S9=0: REM accumulator for sum of squares
```

```
305 FOR I=1 TO M: REM loop over data points
310 LET I3=0: REM flag non-computable function with I3=1
315 GOSUB 3500: REM residual calculation
320 IF I3=0 THEN 340: REM check if in feasible region
325 PRINT "RESIDUAL NOT COMPUTABLE"
330 PRINT #3,"RESIDUAL NOT COMPUTABLE"
335 GOTO 220
340 PRINT "RESIDUAL ";I;" = ";R1
345 PRINT #3,"RESIDUAL ";I;" = ";R1
350 LET S9=S9+R1*R1: REM accumulate sum of squares
355 LET R0=R1
360 GOSUB 4000: REM compute Jacobian in D()
365 FOR J=1 TO N: REM loop over Jacobian components
370 IF O(J,3)=0 THEN 440: REM masked
375 LET H=.01: REM initial stepsize
380 FOR J1=1 TO 5: REM loop over stepsize values
385 LET B(J)=T(J)+H
390 GOSUB 3500: REM compute residual at new point
395 IF I3=0 THEN 415
400 LET W(J,J1)=-9999: REM flag non-computable Jacobian
405 LET I3=0: REM reset flag
410 GOTO 420
415 LET W(J,J1)=(R1-R0)/H: REM compute numerical Jacobian
420 LET B(J)=T(J): REM reset parameter
425 LET H=.01*H: REM adjust stepsize downward
430 IF J1>3 THEN LET H=E8*(ABS(T(J))+E8): REM flexistep
435 NEXT J1
440 NEXT J
445 PRINT "JACOBIAN COMPONENTS AND APPROXIMATIONS"
450 PRINT "ANALYTIC";
455 PRINT #3,"JACOBIAN COMPONENTS AND APPROXIMATIONS"
460 PRINT #3,"ANALYTIC";
465 FOR J1=1 TO 4
470 PRINT TAB(12*J1) "H=10^(-";2*J1;")";
475 PRINT #3,TAB(12*J1) "H=10^(-";2*J1;")";
480 NEXT J1
485 PRINT TAB(60);"FLEX"
490 PRINT #3,TAB(60);"FLEX"
495 FOR J=1 TO N
500 IF O(J,3)<>0 THEN 520
505 PRINT "*** B(";J;") MASKED AT ";B(J)
510 PRINT #3,"*** B(";J;") MASKED AT ";B(J)
515 GOTO 590
520 GOSUB 680: REM find error level
525 PRINT USING "#.###^^^^";D(J);: REM !! Microsoft Basic
530 PRINT #3,USING "#.###^^^^";D(J);: REM !! Microsoft Basic
535 FOR J1=1 TO 5
540 PRINT TAB(12*J1);
545 PRINT USING "#.###^^^^";W(J,J1);: REM !! Microsoft Basic
550 PRINT #3,TAB(12*J1);
```

```
555 PRINT #3,USING "#.###^^^^";W(J,J1);: REM !! Microsoft Basic
560 NEXT J1
565 IF S(6)=0 THEN 580
570 PRINT " **";CHR$(7);
575 PRINT #3," **";
580 PRINT
585 PRINT #3,
590 NEXT J
595 PRINT
600 PRINT #3,
605 NEXT I: REM end loop over data points
610 PRINT
615 PRINT "sum of squared residuals = ";S9
620 PRINT
625 PRINT
630 INPUT "ANOTHER TEST POINT ?";X$
635 PRINT #3,
640 PRINT #3,"sum of squared residuals = ";S9
645 PRINT #3,
650 PRINT #3,
655 PRINT #3,"ANOTHER TEST POINT ?";X$
660 IF X$="Y" THEN 245
665 IF X$="y" THEN 245
670 STOP
675 REM find magnitude of error in analytic derivative
680 FOR J1=1 TO 5
685 LET S(J1)=W(J,J1)-D(J): REM deviation from analytic value
690 NEXT J1
695 LET S(6)=0: REM derivative OK
700 FOR J1=1 TO 4: REM sorting errors
705 FOR J2=J1+1 TO 5
710 IF ABS(S(J1))<=ABS(S(J2)) THEN 730
715 LET T0=S(J1)
720 LET S(J1)=S(J2)
725 LET S(J2)=T0
730 NEXT J2
735 NEXT J1: REM end of sort
740 REM now test the values
745 LET T0=(S(1)+S(2))/2: REM interpolated errors of "best" fit
750 LET T1=ABS(D(J))+E9: REM stabilized denominator
755 IF ABS(T0)/T1 < T7 THEN 770
760 LET S(6)=1: REM failure flag
765 RETURN
770 IF ABS(S(1))/T1 > 4*T7 THEN 760: REM too big
775 IF ABS(S(2))/T1 > 4*T7 THEN 760: REM too big
780 RETURN
785 PRINT "bounds on parameters for problem"
790 PRINT #3,"bounds on parameters for problem"
795 FOR J=1 TO N
800 IF O(J,3)=0 THEN 820
```

```
805 PRINT O(J,1);" <=   b(";J;")  <=  ";O(J,2)
810 PRINT #3,O(J,1);" <=   b(";J;")  <=  ";O(J,2)
815 GOTO 830
820 PRINT " ****  b(";J;") WILL BE MASKED (FIXED) ****"
825 PRINT #3," ****  b(";J;") WILL BE MASKED (FIXED) ****"
830 NEXT J
835 RETURN
840 REM check bounds and set up
845 FOR J=1 TO N
850 IF O(J,3)=0 THEN  905
855 IF B(J)>O(J,1) THEN  880
860 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
865 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
870 LET B(J)=O(J,1)
875 LET O(J,3)=-2: REM lower bound active
880 IF B(J)<O(J,2) THEN  905
885 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
890 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
895 LET B(J)=O(J,2)
900 LET O(J,3)=-1: REM upper bound active
905 NEXT J
910 RETURN

Line no.    Referenced in line(s)
  220       210   335
  245       660   665
  285       235   240
  340       320
  415       395
  420       410
  440       370
  520       500
  580       565
  590       515
  680       520
  730       710
  760       770   775
  770       755
  785       245
  820       800
  830       815
  840       280
  880       855
  905       850   880
 3000       205 -- subroutine to set up problem
 3500       315   390 -- subroutine to calculate residuals
 4000       360 -- subroutine to evaluate Jacobian
 7120       160 -- computing environment subroutine

 Symbol     Referenced in line(s)
```

```
 B(          260   265   290   385   420   505   510   855   860
           865   870   880   885   890   895 -- the parameter vector
 D(           60   525   530   685   750 -- the Jacobian vector
 E5          165 -- a value used for scaled comparisons (= 10)
 E8          165   430 -- the basic stepsize parameter
 E9          165   750 -- the machine precision
 F$           15    20    25 -- the file for console image
 H           375   385   415   425   430 -- the stepsize for the
            numerical approximation to the Jacobian
 I          285   290   295   305   340   345   605 -- a loop counter
 I3         310   320   395   405 -- flag for non-computable function
 J           250   255   260   265   270   365   370   385   400
             415   420   430   440   495   500   505   510   525
             530   545   555   590   685   750   795   800   805
             810   820   825   830   845   850   855   860   865
             870   875   880   885   890   895   900   905 -- a loop
            control counter
 J1          380   400   415   430   435   465   470   475   480
             535   540   545   550   555   560   680   685   690
             700   705   710   715   720   735 -- a loop counter
 J2          705   710   720   725   730 -- a loop control counter
 M           200   210   215   305 -- the number of data points
 N          250   285   365   495   795   845 -- number of parameters
            in loss function
 O(          370   500   800   805   810   850   855   860   865
             870   875   880   885   890   895   900 -- bounds and
            masks information storage
 R0          355   415 -- residual value at initial parameters
 R1          340   345   350   355   415 -- temporary storage of
            the residual
 S(           60   565   685   695   710   715   720   725   745
             760   770   775 -- a flag that the relative deviation
            of the derivative is unacceptable
 S9         300   350   615   640 -- the sum of the squared residuals
 T(           60   290   385   420   430 -- temporary storage of the
            parameter vector
 T0         715   725   745   755 -- interpolated error of "best" fit
 T1          750   755   770   775 -- stabilized denominator
 T7          180   195   755   770   775 -- user-specified tolerance
            for acceptable relative deviation
 W(           60   400   415   545   555   685 -- numerical
            approximations to the Jacobian
 X$          225   230   235   240   630   655   660   665 --
            temporary user input
=================================================================
  LINES: 182     BYTES: 6096     SYMBOLS: 49     REFERENCES: 228
```

3-9. Initial Parameter Estimates

The methods described in this book are all iterative in nature and all of them require some starting values for the parameters to be estimated. In general, experienced practitioners recommend that the initial estimates used be the "best" guess available. However, recognizing that the users of our programs may not have a great deal of experience with either nonlinear models or the estimation techniques, we have tried to make our programs very tolerant of poor initial parameter values. For users wanting to explore the possibility of alternative solutions to particular problems, it is important to use initial parameter values well away from a reported estimate.

There are a number of ways to develop initial parameter values. In many applications, there is a wealth of information from similar situations. For example, in chemical kinetics, where it is desired to estimate rate constants, the chemist will usually have background information on similar reactions and will have at least order of magnitude values for the rate constants. In situations where the rate constant is being estimated over a range of conditions of temperature, pressure or other conditions, it is likely that very good starting values can be provided for experiments which are run late in the study simply by crude extrapolation of parameter estimates from early experiments.

For a number of problems, the model may be linearized by applying transformations, or by ignoring constraints, or by fixing one parameter. As an example, consider the Hobbs 3-parameter logistic problem of Section 2-1. One model for this problem is given by the residual, at time period i, of the form

(3-9-1) B(1) / (1+exp(B(2)-i*B(3))) - Y(i,1)

where Y(i,1) is the observed value of the growth phenomenon

for time point i. In this case, Y(i,1) is the number of weeds per square metre observed in year i.

If the model were exact, we can set the residual to zero, then transform it to give

(3-9-2) B(1)/Y(i,1) - 1 = exp(B(2)-i*B(3))

or

(3-9-3) log(B(1)/Y(i,1) - 1) = B(2) - i*B(3)

Recalling that B(1) is an upper bound to the growth curve, it is now possible to obtain initial estimates of B(2) and B(3) from simple linear regression (Stevens, 1951). Such an analysis for several values of B(1) is more than adequate to provide some initial values for the three parameters. Table 3-9-1 gives the results of the analysis.

Table 3-9-1. Initial estimates for the parameters in the Hobbs weed infestation problem obtained by setting B(1) and calculating B(2) and B(3) from a trend line. The sum of squares reported is for the <u>original</u> untransformed residuals. Note that we are using an unscaled form of the residual in Equation (3-9-1)

B(1) chosen	calculated B(2)	calculated B(3)	residual sum of squares
1	3.738571	4.355923	256.455
2	5.020068	3.129566	2.735687
3	7.16966	2.931647	26.42764
4	9.375299	2.84654	54.23337

COVER SHEET

Chapter 4

Chapter title: Direct search methods

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

4 DIRECT SEARCH METHODS

4-0. Overview -- Direct search methods

In this chapter we present some methods for finding the minima of functions which require only that we are able to provide a mechanism for computing the value of the function at any point in the space of its arguments. The arguments of the function are the parameters we wish to estimate.

4-1. Motivations

Methods for function minimization which need only function values are clearly "easier" to use than those requiring first and possibly second derivatives of the function as well, since it is usually we, the users, who must provide the program code to compute the necessary derivatives. While some workers have reported success in using automated derivative calculation (Shearer and Wolfe, 1985) via symbolic manipulation packages, these facilities are generally not conveniently available to users of personal computers. Our viewpoint is that such users are also unable to count on assistance of mathematically competent helpers to derive or verify the formulas for the required derivatives. We shall therefore wish to simplify as much as possible the task of preparing and adapting the parameter estimation problem to the hardware and software available, and in this context direct search function minimization techniques are an important class of tools.

As will be observed in our examples, we pay for the convenience of avoiding the mental effort of programming the computation of derivatives. The algorithms for direct search methods have generally slower convergence than techniques based on gradient information. Nevertheless, the overall elapsed time to a solution and the effort expended by the user are usually low enough, even with very slow progress during the actual computations, that we generally apply direct search methods as the first tool for nonlinear parameter estimation in our own day-to-day work.

Besides the saving of personal time required to solve a problem, direct search methods may offer other advantages:

- If the method is based on heuristic principles and NOT on approximation of the loss function by a smooth surface such as a paraboloid, then the estimation of nonlinear parameters can involve the use of discontinuous or non-smooth loss functions. That is, the function and/or its derivatives may be discontinuous. Such functional behavior may arise if a constraint is imposed by a barrier function added to the loss function. That is, the loss function is assigned a very large value whenever a particular constraint is violated. Such barrier functions are very easy to implement in computer programs, but they create an obvious discontinuity in the loss function surface. The fact that some direct search minimization methods are able to handle discontinuities is a further convenience

allowing less human energy to be expended in solving the parameter estimation problem.
- The results of parameter estimation problems which arise in business, government or industry may have to be described or explained to non-technical audiences such as management or review committees. In a teaching environment, students whose backgrounds do not involve calculus cannot comprehend easily the processes of function minimization via gradient methods. However, we have found that certain direct search methods are relatively easy to describe. The Hooke and Jeeves and Nelder - Mead methods are based on simple heuristic pattern searches which can be demonstrated graphically for functions of two parameters.
- Frequently, we do not have much experience with the problem or the resulting loss function used to perform the estimation. It is useful in such circumstances to be able to explore the functional surface in different regions of the parameter space in order to gain this experience. Direct search methods, with appropriate modification to print or plot the functional surface, provide a convenient mechanism for such explorations.

So far, we have talked of direct search function minimization methods without formally defining them. A direct search method for minimizing a function of several parameters is a technique which compares the value of the function at a sequence of points in the parameter space and which attempts to generate a new member of the sequence of points which has a lower functional value. In performing the comparison of points, the method may use only the sign of the difference in function values, that is, it may only be concerned with whether point A is higher (has a larger function value) than point B, or it may use the relative size of the function values. In the latter case, we anticipate that the method will be more sensitive to the presence of discontinuities in the function.

4-2. Background

Direct search methods were among the earliest "automatic function minimization" methods reported. Box (1957) introduced the idea of Evolutionary Operation to optimize the performance of a manufacturing plant. A set of points forming the vertices of a hypercube in the parameter space is set up about the current "best" point and the "performance" (that is, loss function) is evaluated at these points. In Box's work, multiple evaluations of performance were used since the "function" was evaluated by experimental measurement. If the current "best" point is still lowest in functional value, then the hypercube is contracted, else we move to the new "best" point and develop a new hypercube.

Box's ideas are straightforward, but involve a lot of computational effort, since the function must be evaluated 2^n ("2 to the power n") times for each hypercube given a problem in n parameters. Nevertheless, similar ideas are at the base of the Nelder - Mead method which will be presented later.

While Box searched over a set of points in the n - dimensional space, other workers suggested defining a set of n search directions. From a current estimate of the function minimum, that is, some initial guess or "best" point, a linear search along one of the search directions is tried. If this results in a new estimate of the minimum, this replaces the "best" point, then we try a search along another search direction. Clearly, it is useful to have independent, and possibly orthogonal, search directions.

This is the basis of a number of methods, for example, those of Rosenbrock (1960), Hooke and Jeeves (1961) and Davies, Swann and Campey (Swann, 1964). The detailed differences between these approaches are of minor interest in the present discussion. It is sufficient to point out that the Hooke and Jeeves method has been selected for inclusion because it offers very simple program code, limited working storage requirements, and reasonable reliability in finding the minima of functions.

More detailed information can be found in Swann (1972) and Dixon (1972). We have deliberately not mentioned methods which try to apply the ideas of gradient function minimization methods by building up a picture of the functional surface. A fairly detailed treatment of one family of such methods is given by Brent (1973). As these approaches are more complicated in concept, and usually in algorithmic detail, than the heuristic methods we present, we shall not discuss them further, though it is important to note that such approaches may yield much better performance on smooth loss functions. Where the loss function is believed to be smooth but the derivatives are awkward to evaluate we prefer to use explicit numerical approximations to the derivatives rather than to build implicit gradient techniques, such as that of Brent.

4-3. Implementation Details

In actually programming a direct search method, there are a number of detailed choices which must be made in order to construct a working program. These choices alter the operational characteristics of programs considerably, so are a necessary topic for consideration.

The first issue we shall address is how to decide when to stop the program. In general, from a numerical analyst's point of view, we wish to allow the program to continue as long as progress can be made in reducing the objective function. Therefore a convergence test could be made using the rule:

<u>CONTINUE THE MINIMIZATION PROGRAM UNTIL NO LOWER FUNCTION VALUE IS FOUND.</u>

In practice, this may lead to unnecessary computational effort. First, for problems with zero as the true minimum of the function, we may approach this minimum via a sequence of function values

$$r, r^2, r^3, \ldots$$

so that the iterates are smaller and smaller but not zero until an underflow occurs, typically many iterations after the user would accept the function value as zero. For example, suppose r = 0.1, which is a typical stepsize reduction factor in the Hooke and Jeeves algorithm, and

$$f(\underline{B}) = \sum_{j=1}^{n} | B(j)| \qquad (4\text{-}3\text{-}1)$$

is to be minimized from a starting point having all the parameters equal to 1. The algorithm takes at least n and up to 2n function evaluations for each axial search. Ignoring the potential accidental arrival at the true minimum and any pattern move to an advantageous region of the parameter space, it is reasonable to predict that the lowest point found after each k major cycles of the Hooke and Jeeves algorithm will be proportional to $(0.1)^k$.

Listing 4-3-1 shows some edited actual output of minimizing the function given in Equation (4-3-1).

Listing 4-3-1. Minimization of the function described in Equation (4-3-1) by the Hooke and Jeeves method. The output has been edited to illustrate the proportionality of the current best function value to powers of the stepsize reduction factor.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
12:40:28      01-19-1986
NUMBER OF PARAMETERS =  4
ABS.FN - SUM OF ABSOLUTE PARAMETERS
Hooke and Jeeves -- 19851018
STEP-SIZE = 1
STEP-SIZE REDUCTION FACTOR = .1
INITIAL FUNCTION VALUE = 13.332
 71           1.332        STEPSIZE= .1
             ( cf. (0.1^1) * 13.332 )
 141          .1319998     STEPSIZE= .01
             ( cf. (0.1^2) * 13.332 )
 211          1.199983E-02 STEPSIZE= 9.999999E-04
             ( cf. (0.1^3) * 13.332 )
 281          1.685694E-07 STEPSIZE= 9.999999E-05
             ( cf. (0.1^4) * 13.332 )
 289          1.685694E-07 STEPSIZE= 9.999999E-06
 297          1.685694E-07 STEPSIZE= 9.999999E-07
 305          1.685694E-07 STEPSIZE= 9.999999E-08
 305          1.685694E-07 STEPSIZE= 9.999999E-09
  ELAPSED SECS= 74  AFTER 0  GRAD & 305  FN EVAL
FUNCTION MINIMUM =  1.685694E-07
PARAMETER ESTIMATES
B( 1 ) = -3.655441E-08
B( 2 ) = -4.400499E-08
B( 3 ) = -4.400499E-08
B( 4 ) = -4.400499E-08
           -----------------------------------------
```

The situation in this example, where the minimum has one or more parameters at zero, is common enough that we should account for it in any convergence test. Another frequent occurrence, which is particularly important for the Hooke and Jeeves algorithm, is that the stepsize is so small that parameters with larger magnitudes are unaltered in the axial search. Clearly it is a waste of effort in such cases to compute the objective function. This leads us to the following procedure for deciding when to stop.

1. If the current parameter is unaltered by addition of

the current stepsize, do not compute the function, and continue the axial search with the next parameter.

2. If none of the parameters have been changed during the entire axial search, stop.

The test for unchanged parameters in the first part of this procedure uses an equality test between (B(j) + stepsize) and B(j). Such equality tests between floating point numbers are frequently considered improper by programming instructors, because there is a danger that very small quantities will not have identical internal representations, and so never satisfy the equality. To avoid any such difficulties, we compare

[(B(j) + stepsize) + E5] with [B(j) + E5]

where E5 is an additive scaling factor. Throughout our own work we use a value E5 = 10. This ensures that very small quantities are "equal" relative to the additive scaling E5, as they are simply shifted off to the right of the arithmetic registers.

It should be noted that this procedure more or less assumes that problems have been set up so that parameters have a maximum magnitude of 10 over the region of interest (see Chapter 3).

For the Nelder - Mead polytope method, which is the second direct search technique included in this book, a different convergence criterion is required because parameter adjustment is not accomplished via movements along axial directions in the parameter space. This minimization method attempts to minimize a function of n parameters by considering (n+1) sets of parameters at once. The method attempts to replace the set of parameters with the highest function value with a new point (set of parameters) of lower function value by use of a set of heuristic search rules. Clearly, no progress can be made if no one point is the "highest", so that polytope methods generally use convergence

tests based on the "closeness" of the "highest" and "lowest" points in the current polytope. We have kept this practice.

The convergence tests for the Hooke and Jeeves (HJ) and the Nelder - Mead (NM) methods are thus very different, and in part account for the markedly different behavior of these methods when applied to the same function minimization problem. Our experience is that the Hooke and Jeeves method is generally slower than the Nelder - Mead, but that its progress is more predictable.

The second implementation detail important to the success of direct search methods is the size of the initial search step. Both the HJ and NM techniques start with axial steps which are the same size along all parameter directions. While a form of automatic scaling could be used, employing ideas similar to those upon which stepsizes for numerical derivatives are based (see Sections 7-5 and 11-8), we prefer to scale the parameters explicitly as in Chapter 3. Users should therefore be aware that the programs NM and HJ may perform poorly on functions for which the parameters have widely differing scale.

The size of the initial search step should be large enough to allow the search procedure to make rapid progress towards lower function values. However, it should not be so large that the search stepsize must be reduced many times to achieve convergence. In part, this simply reflects the value of good initial guesses to the parameters. However, since one of the applications of direct search methods is the finding of alternative sets of parameters having similar values of the loss function, it is clear that the initial stepsize should not be too small. When all parameters are of a magnitude in the range (1, 10), we recommend a value near 1 for the initial search stepsize.

A third implementation detail concerns the incorporation of masks and bounds. Masked parameters must not be altered in the parameter estimation process. In many situations, users wish to supply estimates of parameters obtained outside the framework of automatic loss function minimization. Upper and lower bounds on parameter estimates should not be violated. (We permit masked parameters to be "out of bounds", but parameters are otherwise reset to the nearest bound by our driver program.)

In the HJ method, bounds are fairly easily incorporated into the mechanisms used to generate new search points. In the NM approach, masks can be introduced fairly easily, but explicit accounting for the bounds results in a more cumbersome program code. We found that simply setting the loss function very large whenever a bound is violated, thereby imposing a barrier constraint, resulted in very slow convergence. The effect is that the polytope "bounces off" the bound constraints. The code given in Chapter 6 therefore has an alternative approach wherein elements of the parameter vector $\underline{B}$ which violate bounds are reset to the nearest bound. This may also result in some slowing of the progress of the algorithm towards a minimum.

5

THE HOOKE AND JEEVES METHOD

5-0. Overview -- Hooke and Jeeves

The Hooke and Jeeves (1961) method has been one of the most successful of the direct search methods in practical use, yet it is also one of the simplest to understand, program and use. To underline its utility, we note that it was the major function minimization code of at least one major North American chemical company until after 1980. One numerical study (Eason and Fenton, 1972, 1973) concluded it was the most generally successful single program code over the set of test problems used. In our personal experience, the Hooke and Jeeves Pattern Search (HJ) method has the following strengths and weaknesses.

Strengths:

1. The HJ method has a very short code, and is suitable for implementation on such extremely small computers as the Radio Shack Pocket Computer (also known as the Sharp PC1211) and other programmable calculators.

2. The minimum working storage requirement is only 2 n-vectors for a problem of size n, plus a few scalar variables for controlling the algorithm. (Our program includes a matrix of size 3*n elements to allow masks and bounds to be imposed on the parameters. We also include some extra code and variables to permit control of the program output.)

3. The approach -- axial search and pattern move -- are highly intuitive and easily explained to those with little mathematical sophistication. This renders the HJ method suitable for classroom use with students whose major studies are outside optimization.

4. The algorithm seems to be highly reliable in finding local minima, despite its simplicity.

Weaknesses:

1. The axial search of the HJ method may use as many as 2*n function evaluations in each major cycle of the algorithm. This may result in a program for the HJ method making very slow progress towards a minimum or in its taking many function evaluations to discover it has been started near a local minimum.

2. It is possible to devise functions which increase along all axial directions, but which can be reduced along 'diagonal' search directions. An example of this failure is presented below in Section 5-2.

3. In part because of the overhead of the axial search, the HJ method seems to be inefficient for large numbers of parameters.

3. Because of the geometric basis of the pattern move, the HJ method is quite negatively influenced by badly scaled functions. This is also illustrated by an example in Section 5-3.

4. The manner in which the HJ method is generally terminated, which simply compares the current axial search stepsize to some lower limit, implies that the parameters are determined to a pre-determined absolute precision, that is, number of 'decimals', rather than to a common relative (i.e. percentage) error.

Overall, we prefer to use the HJ method as a tool for exploring the functional surface to discover its properties. Rarely is this the method chosen for final parameter estimation, since convergence to the final (full-precision) results can be at times painfully slow. However, for specialized situations, it may be the method of choice. Aided by Maurice Lafleur, one of us (JCN) developed the HJ method so that it could be implemented in a single-chip microcomputer for real-time optimization of processes. In this approach, the parameters were 1-byte (i.e. 8-bit) integers and the function had a 1-byte mantissa and 1-byte sign/exponent so that the whole code could more easily be adapted to an 8-bit microprocessor such as the Mostek 6502, Intel 8080, or Zilog Z80.

In a more general context, the HJ method is most suitable as a tool for quickly testing code to compute the function and to establish its magnitude and scaling properties. Because the axial search explores the function along axial directions, this is akin to a grid search, with the added advantage that if the process converges quickly to a minimum we need not bother to use other methods unless parameter dispersion estimates based on gradient information are wanted (Section 13-4). These are not generally developed from the final axial search data, which would mean storing an additional 2*n function values.

5-1. Hooke and Jeeves Source-code

The following program code implements the Hooke and Jeeves pattern search. This code includes the facility to handle masks and bounds on the parameters. Masks are dealt with by simply not changing any masked parameter. Bounds are imposed in both axial search and pattern move sections of the method by simply returning a parameter value to the nearest bound whenever the axial step or pattern move violates a bound.

Listing 5-1-1. The Hooke and Jeeves Code.

```
             HJ.BAS           04-09-1986   20:49:01

40 DIM T(25): REM DIMension only needed for POSTGEN, not HJ
1000 PRINT "Hooke and Jeeves -- 19851018"
1004 PRINT #3,"Hooke and Jeeves -- 19851018"
1008 REM CALLS:
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     ENVRON (computing  environment) -- line 7120
1020 REM
1024 REM INPUTS TO THE ROUTINE:
1028 REM    B() -- a vector of initial parameter estimates
1032 REM    S1  -- initial stepsize for search (set to 1 if zero)
1036 REM    S2  -- stepsize reduction factor, which is applied to
1040 REM           S1 when axial search fails (set to 0.1 if zero)
1044 REM    N   -- the number of parameters in the function F(B)
1048 REM    O( , ) -- masks and bounds
1052 REM    I5  -- the number of masked parameters. must be zero if
1056 REM           masks are not used
1060 REM OUTPUT FROM THE ROUTINE:
1064 REM    X() -- a vector of final parameter estimates for the
1068 REM          values of the parameters which minimize the function.
1072 REM    F0  -- value of the function at the minimum
1076 REM    I8  -- number of gradient evaluations (unchanged)
1080 REM    I9  -- number of function evaluations
1084 REM
1088 IF S1<=0 THEN LET S1=1: REM !! warning -- is variable undefined?
1092 IF S2<=0 THEN LET S2=.1: REM !!  ditto
1096 LET J8=40: REM no of fn eval before parameter display
1100 LET J7=1: REM counter for parameter display
1104 GOSUB 7120: REM computing environment
1108 GOSUB 1440: REM copy B() into X() (lowest point so far)
1112 PRINT "STEP-SIZE =";S1
1116 PRINT "STEP-SIZE REDUCTION FACTOR =";S2
```

```
1120 PRINT #3,"STEP-SIZE =";S1
1124 PRINT #3,"STEP-SIZE REDUCTION FACTOR =";S2
1128 GOSUB 1456: REM compute function in F, set I3<>0 if not possible.
1132 IF I3<>0 THEN 1412
1136 PRINT "INITIAL FUNCTION VALUE =";F
1140 PRINT #3,"INITIAL FUNCTION VALUE =";F
1144 LET F1=F: REM store function value at base point
1148 LET F0=F: REM store lowest function value so far
1152 GOSUB 1252: REM axial exploratory search
1156 IF I6=2*(N-I5) THEN 1176: REM parameters unchanged in axial search
1160 IF F0>=F1 THEN 1176: REM test for a lower function value
1164 LET F1=F0: REM update function value at base
1168 GOSUB 1368: REM pattern move
1172 GOTO 1152: REM repeat axial search
1176 FOR J=1 TO N: REM is B() still the current base point?
1180 IF B(J)<>X(J) THEN 1192: REM test for changes in parameters
1184 NEXT J: REM in above test look for equality since B:=X at base
1188 GOTO 1208: REM reduce step-size as search has not reduced function
1192 GOSUB 1424: REM copy X into B (B() is now at the base point)
1196 PRINT " RETURN TO BASE POINT ";
1200 PRINT #3," RETURN TO BASE POINT ";
1204 GOTO 1152: REM try another axial search
1208 LET S1=S1*S2: REM reduce step-size
1212 PRINT
1216 PRINT I9,F0,"STEPSIZE=";S1
1220 PRINT #3,
1224 PRINT #3,I9,F0,"STEPSIZE=";S1
1228 GOSUB 1476: REM display parameters
1232 IF I6<2*(N-I5) THEN 1152: REM convergence test (no altered param's)
1236 PRINT
1240 PRINT #3,
1244 RETURN: REM function minimization complete
1248 REM axial exploratory search subroutine
1252 LET I6=0: REM counter for number of unchanged parameters
1256 FOR J=1 TO N
1260 IF O(J,3)=0 THEN 1344: REM parameter J is masked
1264 LET S3=B(J): REM store parameter value
1268 IF S3=O(J,2) THEN 1284: REM avoid change if at upper bound
1272 LET B(J)=S3+S1: REM step forward
1276 IF B(J)>O(J,2) THEN LET B(J)=O(J,2): REM bound -- assume +VE step
1280 IF B(J)+E5<>S3+E5 THEN 1292: REM test equality relative to E5
1284 LET I6=I6+1
1288 GOTO 1300: REM now try negative step
1292 GOSUB 1456: REM function evaluation
1296 IF F<F0 THEN 1340: REM test if function value < current lowest value
1300 IF S3=O(J,1) THEN 1328: REM no change if at lower bound
1304 LET B(J)=S3-S1: REM step backward
1308 IF B(J)<O(J,1) THEN LET B(J)=O(J,1): REM lower bound
1312 IF B(J)+E5=S3+E5 THEN 1328: REM test equality
1316 GOSUB 1456: REM function evaluation
```

```
1320 IF F<F0 THEN 1340
1324 GOTO 1332
1328 LET I6=I6+1: REM count number of times parameter unchanged
1332 LET B(J)=S3: REM restore original parameter (not by addition!!)
1336 GOTO 1344
1340 LET F0=F: REM store new lowest function value
1344 NEXT J
1348 PRINT " AXIAL SEARCH F0=";F0
1352 PRINT #3," AXIAL SEARCH F0=";F0
1356 GOSUB 1476: REM print parameters
1360 RETURN: REM end axial search
1364 REM PATTERN MOVE
1368 FOR J=1 TO N
1372 IF O(J,3)=0 THEN 1396: REM masked parameter
1376 LET S3=2*B(J)-X(J): REM element of new base point
1380 LET X(J)=B(J): REM store current point
1384 IF S3<O(J,1) THEN LET S3=O(J,1): REM lower bound check
1388 IF S3>O(J,2) THEN LET S3=O(J,2): REM upper bound check
1392 LET B(J)=S3: REM store new base point
1396 NEXT J
1400 PRINT " PMOVE ";
1404 PRINT #3," PMOVE ";
1408 RETURN: REM end pattern move
1412 PRINT "FUNCTION NOT COMPUTABLE AT INITIAL POINT"
1416 PRINT #3,"FUNCTION NOT COMPUTABLE AT INITIAL POINT"
1420 STOP
1424 FOR J=1 TO N: REM copy X into B
1428 LET B(J)=X(J)
1432 NEXT J
1436 RETURN
1440 FOR J=1 TO N: REM copy B into X
1444 LET X(J)=B(J)
1448 NEXT J
1452 RETURN
1456 LET I3=0: REM compute function -- reset failure flag
1460 GOSUB 2000: REM user routine
1464 IF I3<>0 THEN LET F=B9: REM large value assigned
1468 LET I9=I9+1: REM function evaluation counter
1472 RETURN
1476 IF J8=0 THEN RETURN: REM no parameter display
1480 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1484 LET J7=INT(I9/J8)+1: REM parameter display control
1488 PRINT "parameters";
1492 PRINT #3,"parameters";
1496 FOR J=1 TO N
1500 LET Q$=""
1504 IF B(J)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1508 IF O(J,2)-B(J)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1512 IF O(J,3)=0 THEN LET Q$="M": REM masked
1516 PRINT "  ";B(J);Q$;
```

```
1520 PRINT #3,"  ";B(J);Q$;
1524 IF 5*INT(J/5)<>J THEN 1544
1528 PRINT
1532 PRINT "";
1536 PRINT #3,
1540 PRINT #3,"";
1544 NEXT J
1548 PRINT
1552 PRINT #3,
1556 RETURN

     Line no.    Referenced in line(s)
      1152      1172  1204  1232
      1176      1156  1160
      1192      1180
      1208      1188
      1252      1152
      1284      1268
      1292      1280
      1300      1288
      1328      1300  1312
      1332      1324
      1340      1296  1320
      1344      1260  1336
      1368      1168
      1396      1372
      1412      1132
      1424      1192
      1440      1108
      1456      1128  1292  1316
      1476      1228  1356
      1544      1524
      2000      1460 -- subroutine to calculate the loss function
      7120      1104 -- computing environment subroutine

      Symbol     Referenced in line(s)
     B(         1180  1264  1272  1276  1280  1304  1308  1312  1332
                1376  1380  1392  1428  1444  1504  1508  1516  1520
                -- the parameter vector
     B9         1464 -- a "large" number (supplied by ENVRON)
     E5        1280  1312 -- a value used for scaled comparisons (= 10)
     E9         1504  1508 -- the machine precision
     F          1136  1140  1144  1148  1296  1320  1340  1464
                -- the loss function value
     F0         1148  1160  1164  1216  1224  1296  1320  1340  1348
                1352 -- the lowest value found for the loss function
     F1         1144  1160  1164 -- a temporary function value
     I3         1132  1456  1464 -- flag for function not computable
     I5         1156  1232 -- the number of masked parameters
     I6        1156  1232  1252  1284  1328 -- a counter for the number
```

```
             of parameters which are unchanged in a step
I9           1216  1224  1468  1480  1484 -- counter for function
             evaluations performed
J            1176  1180  1184  1256  1260  1264  1268  1272  1276
             1280  1300  1304  1308  1312  1332  1344  1368  1372
             1376  1380  1384  1388  1392  1396  1424  1428  1432
             1440  1444  1448  1496  1504  1508  1512  1516  1520
             1524  1544 -- a loop control counter
J7           1100  1480  1484 -- parameter display control index
J8           1096  1476  1480  1484 -- parameters are displayed
             every J8 function evaluations (no display if J8=0)
N            1156  1176  1232  1256  1368  1424  1440  1496
             -- number of parameters in loss function
O(           1260  1268  1276  1300  1308  1372  1384  1388  1504
             1508  1512 -- bounds and masks information storage
Q$           1500  1504  1508  1512  1516  1520 -- used to hold
             display information regarding masks and active bounds
S1           1088  1112  1120  1208  1216  1224  1272  1304
             -- the current stepsize for axial search
S2           1092  1116  1124  1208
             -- the stepsize reduction factor
S3           1264  1268  1272  1280  1300  1304  1312  1332  1376
             1384  1388  1392 -- a temporary variable
T(             40 -- a temporary vector needed by POSTGEN.BAS
X(           1180  1376  1380  1428  1444 -- storage for the
             "best" parameters found so far
===================================================================
LINES: 142      BYTES: 5688     SYMBOLS: 44     REFERENCES: 190
```

5-2. Use of the Hooke and Jeeves Method

We first apply the Hooke and Jeeves method to the solution of the Hobbs 3-parameter logistic estimation problem which was presented in Section 2-1. Listing 5-2-1 shows an edited output for this problem.

Listing 5-2-1. Edited output of the application of HJ to the Hobbs weed infestation problem.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
20:48:19      04-17-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(2)*I))
```

```
HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
 are masks or bounds to be set or altered ([cr] = no) n
Hooke and Jeeves -- 19851018
STEP-SIZE = 1
STEP-SIZE REDUCTION FACTOR = .1
INITIAL FUNCTION VALUE = 158.2325
 AXIAL SEARCH F0= 127.9983
 PMOVE  AXIAL SEARCH F0= 127.9983
 RETURN TO BASE POINT  AXIAL SEARCH F0= 127.9983

 19            127.9983     STEPSIZE= .1
 AXIAL SEARCH F0= 17.68197
 47            11.01658     STEPSIZE= .01
 AXIAL SEARCH F0= 9.869432
 131           3.668904     STEPSIZE= 9.999999E-04
 AXIAL SEARCH F0= 3.65716
 287           2.606807     STEPSIZE= 9.999999E-05
 AXIAL SEARCH F0= 2.606771
 405           2.58762      STEPSIZE= 9.999999E-06
 AXIAL SEARCH F0= 2.587601
 449           2.587578     STEPSIZE= 9.999999E-07
 455           2.587578     STEPSIZE= 9.999999E-08
 455           2.587578     STEPSIZE= 9.999999E-09

  ELAPSED SECS= 356  AFTER 0  GRAD & 455  FN EVAL
CALCULATED FUNCTION MINIMUM =  2.587578
PARAMETER ESTIMATES
B( 1 ) =  1.960722
B( 2 ) =  4.90423
B( 3 ) =  3.135595
POSTGEN.BAS 851118 -- GENERAL POST SOLUTION ANALYSIS
LOSS FUNCTION PER DEGREE OF FREEDOM (SIGMA^2) = .2875087
B( 1 )= 1.960722E+00  step= 6.78E-03  f-, f+  2.8790E+00  2.8789E+00
B( 2 )= 4.904230E+00  step= 1.69E-02  f-, f+  2.7143E+00  2.7090E+00
B( 3 )= 3.135595E+00  step= 1.08E-02  f-, f+  3.7853E+00  3.7885E+00
        best function value found is            2.5876E+00

radii of curvature for surface along axial directions
  & tilt angle in degrees

for B( 1 )  R. OF CURV. =  1.579E-04   tilt =    0.34444
for B( 2 )  R. OF CURV. =  2.403E-03   tilt =    9.03852
```

```
    for B( 3 )  R. OF CURV. =  1.012E-04   tilt =   -8.41148

    RESIDUALS
     1.395655E-02 -3.010845E-02  9.538555E-02  .2129488  .3976593
    -5.178452E-02 -1.099423  .7219544 -.1027107 -.346302
     .649704 -.2977905
```

A second example of the HJ method is provided by Caceci and Cacheris (1984), who wish to estimate a 2-parameter model for a chemical reaction. This is discussed in more detail in Section 16-1, and the output presented below in Listing 5-2-2.

Listing 5-2-2. Edited output of the application of HJ to the Caceci and Cacheris kinetic problem.

```
    DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
    21:05:36      04-17-1986
    BYTE problem (Caceci & Cacheris, 1984)
    860406
     6  data points
     2  parameters
     2             data series
    Substrate concentration
    data series  1
     1.68  3.33  5  6.67  10
     20
    Reaction velocity
    data series  2
     .172  .25  .286  .303  .334
     .384
    CHEM.RES kinetic problem 860102 + DATAFILE b:byte.CHM
    bounds on parameters for problem
     0  <=   b( 1 )  <=   100
     0  <=   b( 2 )  <=   100
    ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
    B( 1 )= 1
    B( 2 )= 1
    Hooke and Jeeves -- 19851018
    STEP-SIZE = 1
    STEP-SIZE REDUCTION FACTOR = .1
    INITIAL FUNCTION VALUE = 1.75074
     AXIAL SEARCH F0= .524701
     PMOVE  AXIAL SEARCH F0= .524701
     RETURN TO BASE POINT  AXIAL SEARCH F0= .524701
```

```
 10            .524701      STEPSIZE= .1
 AXIAL SEARCH F0= .2970126
 38            5.501088E-04 STEPSIZE= .01
 AXIAL SEARCH F0= 5.445365E-04
 85            2.879175E-04 STEPSIZE= 9.999999E-04
 AXIAL SEARCH F0= 2.727602E-04
 178           1.675951E-04 STEPSIZE= 9.999999E-05
 AXIAL SEARCH F0= 1.675943E-04
 229           1.675779E-04 STEPSIZE= 9.999999E-06
 233           1.675779E-04 STEPSIZE= 9.999999E-07
 236           1.675779E-04 STEPSIZE= 9.999999E-08
 236           1.675779E-04 STEPSIZE= 9.999999E-09

  ELAPSED SECS= 152  AFTER 0  GRAD & 236  FN EVAL
CALCULATED FUNCTION MINIMUM =  1.675779E-04
PARAMETER ESTIMATES
B( 1 ) =  2.453777
B( 2 ) =  .4238997
POSTGEN.BAS 851118 -- GENERAL POST SOLUTION ANALYSIS
LOSS FUNCTION PER DEGREE OF FREEDOM (SIGMA^2) = 4.189447E-05
B( 1 )= 2.453777E+00  step= 8.48E-03  f-, f+  1.6807E-04  1.6812E-04
B( 2 )= 4.238997E-01  step= 1.48E-03  f-, f+  1.7386E-04  1.7401E-04
       best function value found is             1.6758E-04

radii of curvature for surface along axial directions
  & tilt angle in degrees

for B( 1 )  R. OF CURV. =  1.403E+02   tilt =   -0.00017
for B( 2 )  R. OF CURV. =  3.425E-01   tilt =   -0.00300

RESIDUALS
2.762079E-04 -5.940482E-03 -1.64777E-03  6.894738E-03  6.378383E-03
-6.424636E-03
```

COVER SHEET

Chapter 6

Chapter title: The Nelder - Mead polytope method

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

6

THE NELDER - MEAD POLYTOPE METHOD

6-0. Overview -- Nelder - Mead

The Nelder - Mead polytope method is one of the most successful direct search techniques for finding the minimum of a function f($\underline{B}$) of n parameters B(j), j=1,2,...n. Actually, this is not one but a family of closely related methods which apply simple heuristics to modify a set of (n+1) points

(6-0-1) $\underline{B}_k$, k = 1,2,...,n+1

in the n-dimensional space having as its axes the parameters. For a function of 2 parameters, this means that the usual x,y axes become

(6-0-2) x = B(1), y = B(2)

with it possible to think of a z axis representing the function value. The (n+1) = 3 points form a triangle in the x,y space.

As long as the (n+1) points are NOT all in one plane (or hyperplane in more than 3 dimensions), the polytope is the geometric figure having these points as its vertices. Another name for this construct is a <u>simplex</u>, and many workers call the present approach to minimizing a function the "Nelder - Mead Simplex algorithm". Unfortunately, this gives rise to confusion with the Linear (and other) Programming algorithms of Dantzig and others (Gass, 1964), so we shall use the name "polytope" here.

If we sort the function values at the vertices of the polytope, we can label three vertices in particular:

H - the "highest", having the largest function value
L - the "lowest", or best estimate of the minimum so far
N - the "next to highest", having the second largest function value.

The different members of the family of polytope algorithms apply slightly different heuristics to try to find "lower" points on the functional surface. These are described in detail by Nash (1979, chapter 14), Parkinson and Hutchinson (1972) and others. Here we shall avoid mathematical detail in favour of a conceptual description.

The various tactics used in polytope methods have acquired names which describe the geometric operations applied to the polytope.

1. Reflection: the centroid (or "average") of all points other than H is formed. We call this point C in the (n+1) dimensional space. We then "reflect" H through C, that is, move twice the distance from H to C along the line H-C to a new point R.

2. Expansion: the point R is not magical, and if R is "lower" in function value than the current lowest point L, we continue to extend the line H-C to some new point E where the function is again evaluated.

3. Contraction (or reduction): when the point R is not "lower" than point L, we can try other points on the line H-C-R.

a. a "low-side" contraction tries a point between C and R if the function value at R is less than that at H.
b. a "high-side" contraction tries a point between H and C if R is "higher" than H.

4. General shrinkage: point L is retained. All other vertices are moved towards L along the edges of the polytope.

The tactics used by Nelder and Mead lead to the following sequence of calculations:

1. Points C and R are found, and the function at R is calculated;

2. One of the following situations holds

a. R is higher than H. A point P1 between C and H is found and the function computed (high-side reduction);
   if P1 is lower than H, P1 replaces H, otherwise a general shrinkage is applied.
b. R is lower than H but higher than N. A point P2 between C and R is found and the function computed (low- side reduction);
   if P2 is lower than R, then P2 replaces H, otherwise R replaces H.
c. R is lower than N but higher than L, then R replaces H.
d. R is lower than L. Find point E and evaluate the function,
   if E is lower than R (which is already lower than L), then E replaces H in the polytope, otherwise R replaces H.

In the above description we have avoided the issue of ties in the function values. In implementing a polytope algorithm, the case of equal function values must be taken into account, forcing a decision to be made regarding the specific tactical path taken. Similar decisions are needed regarding ties in the ranking of the function values of the points in the polytope.

The result of applying the operations above to the polytope is a new polytope to which the same tactics can be applied. The work involved in each cycle is

- calculation of centroid
- reflection and 1 function evaluation in case 2c.
- reflection and movement to an expansion or contraction point and two function evaluations in cases 2a, 2b, and 2d.
- movement of n vertices of the polytope, evaluation of the function at these n new points and re-ranking of the function values in the case that 2a fails to find a point lower than H.

Particular implementations of polytope methods may differ conceptually in the following ways:

1. the size of the step taken along line H-C to find point R (the reflection coefficient, alpha);
2. the size of the step taken to extend line H-C-R to point E (the expansion coefficient, gamma);
3. the size of the step taken along line H-C-R from C to point S1 or S2 (the contraction coefficient, beta);
4. the ratio of the distance from L to a "new" vertex to the distance from L to the corresponding vertex in a shrinkage (the shrinkage coefficient, which is usually chosen to be the same as the contraction coefficient beta);
5. the particular ways in which ties between function values are broken, as mentioned above;

6. the manner in which the initial polytope is generated (Parkinson and Hutchinson, 1972);
7. the criterion used to decide when the minimum has been found;
8. possible additional tactics such as
    a. translating the points of the polytope to "follow" an extension point E and allowing such extension to be repeated within a major cycle (Parkinson and Hutchinson, 1972)
    b. "moving" more than one point of the polytope at one time (Evans and Craig, 1979)
    c. use of more than (n+1) points in the geometric structure used for the search (M.J.Box, 1965)
    d. evaluating the function at the centroid C or at additional search points near a supposed minimum (Nash, 1979, chapter 14).
    e. avoiding the comparisons needed to find the "next to highest" vertex of the polytope.

With the array of possibilities, readers should be cautious of statements, such as, "the Nelder - Mead polytope is better (worse) than (some other method)", since generalization from one set of specific tactics and choices of control information (alpha, beta and gamma) is nearly impossible. However, it does appear that this family of methods is in some overall sense very reliable in finding the minima of functions despite differences at the tactical level.

What is more difficult to determine is the relative efficiencies of different choices of tactics over the broad range of possible functions to be minimized. In this book we have decided to present a fairly conservative variant of the polytope method.

To avoid the comparisons needed to find the function value at vertex N, which is needed in order to decide whether the low-side contraction is to be performed, we arbitrarily compare the function at the reflection point R with the weighted average of the function values at the lowest and highest vertices. That is, we compare $f(\underline{B})$ with

(1 - beta) * f_high + beta * f_low

where we have used the contraction coefficient beta to control the comparison.

While we have indicated that minor differences in the choice of tactics are generally unimportant to the success of these methods in finding the minimum of a function, we would caution that it IS important to program the operations carefully. For example, the shrinkage operation will move each vertex, V, other than the lowest, L, to a new point between V and L which is gamma * (distance L to V) from L. If all the vertices are stored in matrix W(,) so that the i'th component of the j'th vertex is in W(i,j), then the new position of vertex j (that is, the new value of parameter B(i) for i = 1,2,...,n at this vertex is given by

(6-0-3) B(i) = W(i,L) + gamma * (W(i,V) - W(i,L))

Note that this equation is also valid if V = L (for lazy programmers) and also that it can be rewritten as

(6-0-4) B(i) = (1-gamma) * W(i,L) + gamma * W(i,V)

We prefer to actually use (6-0-3) and compute the difference

(W(i,V) - W(i,L))

which can be used as a crude measure of the "size" of the polytope. This "size" clearly should be reduced in a shrinkage operation, and if it is not reduced, there is no sense in continuing. Such situations may arise as follows when the method has "nearly" converged.

Suppose we have a computing environment allowing 3-digit decimal arithmetic which rounds and have

gamma = 0.5

(6-0-5) W(i,L)= 5.07

W(i,V)= 5.08

Both formulas (6-0-3) and (6-0-4) give B(i) = 5.08, so that W(i,V) has not been moved. The "size" test is then useful, since many functions (especially with penalty or barrier constraints) may have values which are different at L and V. Other practitioners suggest convergence tests based on the variance of the function values of the current polytope, that is

$$\text{(6-0-6)} \quad \text{test} = \sum_{k=1}^{n} (f(\underline{B}_k) - \text{f\_bar})^2/n$$

where

$$\text{(6-0-7)} \quad \text{f\_bar} = \sum_{k=1}^{n} f(\underline{B}_k)/n$$

The variance, "test", is then compared to some tolerance. Unfortunately, such measures are sensitive to the scaling of the function, the presence of discontinuities and the precision of floating-point arithmetic available. Our own preference is to proceed with the polytope algorithm until

1. $f(\underline{B}_H) \underset{\sim}{} f(\underline{B}_L)$, that is, there is no "highest" or "lowest" point (the comparison we implement involves a tolerance based on the size of the initial function value calculated and the machine precision);

2. shrinkage fails to reduce the polytope size; or

3. a user-determined count on the number of function evaluations is exceeded. This last condition is left to the user to implement if desired.

This approach allows the Nelder - Mead polytope method to be applied to problems where there are discontinuities in the function arising from the problem at hand or from barrier constraints which have been added.

The ability of the polytope method to cope with such problems, along with its overall robustness to a wide range of problem types, are its major strengths compared to other direct search methods. Compared to the Hooke and Jeeves method, the Nelder - Mead polytope method is generally more efficient for problems involving a moderate number of parameters. We have dealt with problems of up to 41 parameters (Nash, 1984a, page 229; Nash, 1981) with relative ease. As the number of parameters n increases, it is clear that the amount of working storage increases as n^2. However, the workload does not necessarily increase at this rate, and we have noted some problems for which NM is surprisingly efficient.

From a simple geometric perspective, it is clear that well-scaled problems will be easier to handle. If the function changes very rapidly near the minimum for one parameter, then the polytope will be "flattened" along the axis corresponding to that parameter, with consequences for the test points used in the algorithm. For example, suppose we have a function of 2 parameters B(1) and B(2). If the polytope at the start of the cycle is

```
(6-0-8)     W(1,H) = 5.00   W(2,H) = 1.03
            W(1,N) = 7.00   W(2,N) = 1.01
            W(1,L) = 2.00   W(2,L) = 1.02
```

Again we will use 3-digit decimal arithmetic with rounding.

The centroid of points N and L is

```
(6-0-9)     W(1,C) = 4.50   W(2,C) = 1.02 (NOT 0.015)
```

The reflection point, using a reflection coefficient of 1, is

```
            W(1,R) = W(1,C) + 1 * (W(1,C) - W(1,H))
                   = 4.50 + (4.50 - 5.0)
                   = 4.00
```

```
            W(2,R) = 1.02 + 1 * (1.02 - 1.03)
                   = 1.01
```

whereas the true value should be

```
            W'(2,R) = 1.015 + 1 * (1.015 - 1.03)
                    = 1.015 - 0.015
                    = 1.00
```

Note that in these manipulations the function has NOT appeared except via the ranking of the points. This is at the same time an advantage to the Nelder - Mead method in that non-smooth functions may be dealt with, but a potential source of inefficiency in that information available to us is not used.

A final note on the polytope method concerns the generation estimates of the dispersion (standard errors) of parameters. Spendley (1969) has pointed out that the final polytope contains information which can be used to compute estimates of the dispersion of the parameters. We return to such ideas in Section 13-4.

Other tactics besides those discussed are, of course, of possible utility. Following Parkinson and Hutchinson (1972b), we initially planned to include a modification of the Nelder and Mead strategy to take advantage of cases where the direction of expansion (extension) of the polytope is such that each new point found is lower in function value than any test point so far encountered. Repeated extension of the search line is permitted until the test point does not give a new point which is the "lowest so far." At this stage in the minimization process, the polytope is very much distorted in the direction of the search line (the reflection of the "old" high vertex through the centroid of the other vertices). If we simply continue the traditional strategy with such a distorted polytope, many cycles of contraction of the polytope, either by high-side or low-side contractions or general shrinkage of the polytope, may be needed before more progress can be made towards a minimum of the function. The rationale for this statement is that the distortion in the shape of the polytope introduced by the extension will result in the next reflection operation giving a test point more or less in the direction of the extension. Since the extension has just been halted because no new "low" point could be found, a change in direcion of search is indicated. That is, we need to change the search direction.

One approach to providing this direction, suggested by Parkinson and Hutchinson (1972b), is to move all the points of the polytope nearer the new estimate of the function minimum, that is, nearer the last successful extension test point E'. This translation of the polytope is accomplished by moving every point of the polytope but the "old" highest point (by now replaced by the last successful extension point) along a line parallel to the extension direction a distance equal to that from point R to point E'. This translation is relatively easy to perform, but costs us n function evaluations to find the "heights" of the newly defined polytope vertices. Since the extension operation is the only way that the polytope can grow in size, we only perform the translation if at least two successful extensions have been performed. That is, we count the extension operations, and note that the last will have been unsuccessful in finding a new "low" test point.

This extension/translation strategy is supported mainly on heuristic grounds. For functions which are relatively well-scaled, it is expected to improve the performance of the polytope method. However, as we have mentioned, there are so many different sets of choices to be made in the implementation of a polytope minimization program that we cannot provide a statistical justification of any particular implementation. In the example problems we tried, the extension/translation modification appeared not to be invoked

enough to outweigh the extra overheads it imposed, and in the program presented below we have stayed with simpler tactics.

The Nelder - Mead method does not lend itself as well as the Hooke and Jeeves method to the incorporation of bounds constraints. The parameters are not adjusted one at a time as in the axial search, so the approach we have taken is to return the parameter value to the nearest bound each time a bound constraint is violated. This has the unfortunate effect that the polytope can "flatten" against the bound constraint, leading to premature convergence of the program. Generally, the axial search used to develop measures of parameter dispersion will find a lower point (set of parameter values), but reliance on such a device is inelegant. We have noted that such premature convergence may be avoided in some cases by using a relatively small stepsize and initial parameter values "far" from bounds to build the original polytope.

Masks also present additional work in developing a Nelder - Mead program, since each masked parameter reduces the dimensionality of the polytope. While this is helpful in reducing the computational workload for the computer, and introduces no fundamental theoretical difficulty, the nuisance of the detail to which we must pay attention can easily allow errors to enter the code.

6-1. Nelder - Mead Source-code

```
      NM.BAS           04-20-1986   11:15:07

40 DIM W(26,27): REM should be (N+1) by (N+2) at least
44 DIM T(25): REM for tilts in POSTGEN
1000 PRINT "NM -- nelder-mead polytope method -- 851110"
1004 PRINT #3,"NM -- nelder-mead polytope method -- 851110"
1008 REM CALLS
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     ENVRON (computing environment) -- line 7120
```

```
1020 REM
1024 REM INPUTS TO THE ROUTINE
1028 REM    B() -- a vector of initial parameter estimates
1032 REM    N   -- the number of parameters in the function F(B)
1036 REM    O( , ) -- masks and bounds
1040 REM    S1  -- the stepsize to build polytope
1044 REM    I5  -- the number of masked (fixed) parameters
1048 REM
1052 REM OUTPUT FROM THE ROUTINE
1056 REM    X() -- a vector of final parameter estimates for the
1060 REM           values of the parameters which minimuze the function.
1064 REM    F0  -- value of the function at the minimum
1068 REM    I8  -- the number of gradient evaluations (unchanged)
1072 REM    I9  -- the number of function evaluations
1076 REM
1080 REM need arrays B(N),W(N+1,N+2) and output vector X(N)
1084 REM but X() can be deleted by minor programming changes
1088 REM X() is output here for compatibility with other programs
1092 IF S1>0 THEN 1104: REM assume set outside program if positive
1096 INPUT "stepsize to build polytope =";S1
1100 PRINT #3,"stepsize to build polytope =";S1
1104 LET S5=1: REM reflection factor
1108 LET S7=1.2: REM extension factor
1112 LET S6=.5: REM reduction and shrink factor
1116 PRINT "strategy -- refln, extnn and redn factors ";S5;S7;S6
1120 PRINT #3,"strategy -- refln, extnn and redn factors ";S5;S7;S6
1124 GOSUB 7120: REM computing environment
1128 LET J8=40: REM no of fn eval before parameter display
1132 LET J7=1: REM counter for parameter display
1136 GOSUB 1692: REM initial function value
1140 IF I3=0 THEN 1160
1144 PRINT "function not computable at initial point"
1148 PRINT #3,"function not computable at initial point"
1152 STOP
1156 REM now build polytope and put in the values of the other points
1160 PRINT "function at initial guess =";F
1164 PRINT #3,"function at initial guess =";F
1168 LET N2=N-I5: REM polytope has dimensionality n - no. of masks
1172 IF N2=0 THEN STOP: REM done if all parameters fixed
1176 LET F9=ABS(F)+1: REM save adjusted value for use in convergence test
1180 FOR K=1 TO N: REM build polytope
1184 LET W(K,1)=B(K): REM put initial point in first column
1188 NEXT K
1192 LET W(N+1,1)=F: REM to save function value
1196 LET K1=1: REM first parameter to be altered
1200 FOR J=2 TO (N2+1): REM loop over columns
1204 FOR K=1 TO N
1208 LET W(K,J)=B(K)
1212 NEXT K
1216 IF O(K1,3)=0 THEN LET K1=K1+1: REM masked parameter
```

```
1220 LET B(K1)=B(K1)+S1: REM step to build polytope
1224 GOSUB 1692: REM compute function
1225 IF B(K1)<>W(K1,J)+S1 THEN PRINT "RESET PARM.";K1;" TO ";B(K1)
1226 IF B(K1)<>W(K1,J)+S1 THEN PRINT #3,"RESET PARM.";K1;" TO ";B(K1)
1227 LET F3=B(K1)
1228 LET B(K1)=W(K1,J): REM reset the parameter
1232 LET W(K1,J)=F3
1236 REM note how we avoid addition and subtraction on B()
1240 LET W(N+1,J)=F: REM function value
1244 PRINT "vertex ";J;" = ";F
1248 PRINT #3,"vertex ";J;" = ";F
1252 LET K1=K1+1: REM increment for next parameter
1256 NEXT J
1260 LET K$="build"
1264 LET F0=W(N+1,1): REM ranking -- smallest function value in F0
1268 LET F3=F0: REM largest function value to be in F3
1272 LET K1=1
1276 LET K3=1
1280 FOR J=2 TO (N2+1)
1284 LET F=W(N+1,J)
1288 IF F>=F0 THEN 1308
1292 REM new lowest function value found, so save it and index
1296 LET F0=F
1300 LET K1=J
1304 GOTO 1320
1308 IF F<F3 THEN 1320
1312 LET K3=J: REM index of new highest
1316 LET F3=F
1320 NEXT J
1324 REM done ranking loop -- now test for convergence
1328 IF F3<=F0+E9*F9 THEN 1648
1332 REM use machine precision E9 for test
1336 PRINT K$;TAB(14) I9;TAB(20) F0;TAB(40) F3
1340 PRINT #3,K$;TAB(14) I9;TAB(20) F0;TAB(40) F3
1344 REM print low and high fn. vals.
1348 GOSUB 1744: REM display parameters
1352 FOR K=1 TO N: REM build centroid in rightmost col. of W( )
1356 LET F=-W(K,K3): REM subtract out highest point
1360 FOR J=1 TO (N2+1)
1364 LET F=F+W(K,J)
1368 NEXT J
1372 LET W(K,N2+2)=F/N2: REM note mod. for masks
1376 NEXT K
1380 REM centroid now stored in col. (N2+2) of W()
1384 FOR K=1 TO N: REM reflection operation
1388 IF O(K,3)=0 THEN 1396: REM masked
1392 LET B(K)=(1+S5)*W(K,N2+2)-S5*W(K,K3): REM reflection point in B()
1396 NEXT K
1400 LET K$="refl'n"
1404 GOSUB 1692: REM compute function at reflection point
```

```
1408 REM now try the different cases
1412 IF F<F3 THEN 1492: REM lower than highest?
1416 LET K$="hi-contr'n"
1420 GOSUB 1664: REM contraction
1424 IF F<F3 THEN 1536: REM contraction successful?
1428 REM shrink polytope towards lowest vertex
1432 LET S8=0: REM old polytope size measure in shrink operation
1436 LET S9=0: REM new polytope size measure
1440 FOR J=1 TO (N2+1)
1444 IF J=K1 THEN 1476
1448 FOR K=1 TO N
1452 IF O(K,3)=0 THEN 1472: REM masked
1456 LET F=W(K,J)-W(K,K1)
1460 LET S8=S8+ABS(F)
1464 LET W(K,J)=S6*F+W(K,K1): REM new vertex of polytope
1468 LET S9=S9+ABS(W(K,J)-W(K,K1))
1472 NEXT K
1476 NEXT J
1480 LET K$="shrink"
1484 IF S9>.5*(1+S6)*S8 THEN 1624: REM test that shrink has worked
1488 GOTO 1264
1492 FOR K=1 TO N
1496 LET W(K,K3)=B(K): REM reflection point is saved in old highest
1500 NEXT K
1504 LET W(N+1,K3)=F: REM with function value
1508 IF F<(1-S6)*F3+S6*F0 THEN 1532
1512 REM use wtd avg of lowest and highest for "next to highest"
1516 LET K$="lo-contr'n"
1520 GOSUB 1664: REM contraction
1524 IF F>W(N+1,K3) THEN 1264: REM restart if not good
1528 GOTO 1536: REM new point as low as H, so save
1532 IF F<W(N+1,K1) THEN 1556: REM lower than best
1536 FOR K=1 TO N
1540 LET W(K,K3)=B(K): REM save new point from line reductions or refl'n
1544 NEXT K
1548 LET W(N+1,K3)=F: REM save function value
1552 GOTO 1264: REM and restart cycle
1556 LET K$="exten'n"
1560 LET W(N+1,K3)=F
1564 REM col. k3 of work array will be made to store lowest point so far
1568 FOR K=1 TO N
1572 LET W(K,K3)=B(K): REM refln point or extn point
1576 IF O(K,3)=0 THEN 1588: REM masked
1580 LET B(K)=B(K)+S7*(W(K,K3)-W(K,N2+2))
1584 REM extension from reflection point
1588 NEXT K
1592 GOSUB 1692: REM function value at extension point
1596 IF F>=W(N+1,K3) THEN 1264
1600 REM if not lower, try again (refln already in place)
1604 FOR K=1 TO N
```

```
1608 LET W(K,K3)=B(K): REM save new low
1612 NEXT K
1616 LET W(N+1,K3)=F
1620 GOTO 1264
1624 PRINT "shrink operation has failed to reduce polytope sufficiently"
1628 PRINT #3,"shrink operation has failed to reduce polytope sufficiently
1632 PRINT "size measure before shrink operation =";S8
1636 PRINT #3,"size measure before shrink operation =";S8
1640 PRINT "size measure after  shrink operation =";S9
1644 PRINT #3,"size measure after  shrink operation =";S9
1648 FOR K=1 TO N
1652 LET X(K)=W(K,K1): REM copy parameters back
1656 NEXT K
1660 RETURN: REM return to main program
1664 FOR K=1 TO N: REM contraction operation
1668 IF O(K,3)=0 THEN 1676: REM masked
1672 LET B(K)=(1-S6)*W(K,K3)+S6*W(K,N2+2): REM reduce along line H-C
1676 NEXT K
1680 GOSUB 1692: REM compute function
1684 RETURN: REM end contraction
1688 REM test bounds before calling function
1692 LET I3=0: REM turn off flag before testing parameters
1696 LET I9=I9+1: REM to count function evaluations
1700 FOR J1=1 TO N
1704 IF O(J1,3)=0 THEN 1716: REM masked?
1708 IF B(J1)>=O(J1,1) THEN 1712: REM lower bound
1709 LET B(J1)=O(J1,1): REM BUT DIDN'T FIX W()
1710 LET K$="<"+K$: REM to indicate lower bound active
1711 GOTO 1716
1712 IF B(J1)<=O(J1,2) THEN 1716: REM upper bound
1713 LET B(J1)=O(J1,2)
1714 LET K$=">"+K$: REM to indicate upper bound active
1716 NEXT J1: REM done tests, so compute function
1720 GOSUB 2000: REM user function sub-program
1724 IF I3=1 THEN 1736: REM non-computable function at B
1728 RETURN
1732 LET I3=1: REM out of bounds
1736 LET F=B9: REM use large value of function
1740 RETURN
1744 IF J8=0 THEN RETURN: REM no parameter display
1748 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1752 LET J7=INT(I9/J8)+1: REM parameter display control
1756 PRINT "parameters";
1760 PRINT #3,"parameters";
1764 FOR J=1 TO N
1768 LET Q$=""
1772 IF W(J,K1)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1776 IF O(J,2)-W(J,K1)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1780 IF O(J,3)=0 THEN LET Q$="M": REM masked
1784 PRINT "  ";W(J,K1);Q$;
```

```
1788 PRINT #3,"  ";W(J,K1);Q$;
1792 IF 5*INT(J/5)<>J THEN 1812
1796 PRINT
1800 PRINT "          ";
1804 PRINT #3,
1808 PRINT #3,"          ";
1812 NEXT J
1816 PRINT
1820 PRINT #3,
1824 RETURN
```

Line no.	Referenced in line(s)
1104	1092
1160	1140
1264	1488 1524 1552 1596 1620
1308	1288
1320	1304 1308
1396	1388
1472	1452
1476	1444
1492	1412
1532	1508
1536	1424 1528
1556	1532
1588	1576
1624	1484
1648	1328
1664	1420 1520
1676	1668
1692	1136 1224 1404 1592 1680
1712	1708
1716	1704 1711 1712
1736	1724
1744	1348
1812	1792
2000	1720 -- subroutine to calculate the loss function
7120	1124 -- computing environment subroutine

Symbol	Referenced in line(s)
B(	1184 1208 1220 1225 1226 1227 1228 1392 1496 1540 1572 1580 1608 1672 1708 1709 1712 1713 -- the parameter vector
B9	1736 -- a "large" number (supplied by ENVRON)
E9	1328 1772 1776 -- the machine precision
F	1160 1164 1176 1192 1240 1244 1248 1284 1288 1296 1308 1316 1356 1364 1372 1412 1424 1456 1460 1464 1504 1508 1524 1532 1548 1560 1596 1616 1736 -- the loss function value
F0	1264 1268 1288 1296 1328 1336 1340 1508 -- the lowest value found for the loss function
F3	1227 1232 1268 1308 1316 1328 1336 1340 1412 1424 1508 -- highest function value in current polytope
F9	1176 1328 -- adjusted initial function value
I3	1140 1692 1724 1732 -- flag for function not computable
I5	1168 -- the number of masked parameters
I9	1336 1340 1696 1748 1752 -- counter for function evaluations performed
J	1200 1208 1225 1226 1228 1232 1240 1244 1248 1256 1280 1284 1300 1312 1320 1360 1364 1368 1440 1444 1456 1464 1468 1476 1764 1772 1776 1780 1784 1788 1792 1812 -- a loop control counter
J1	1700 1704 1708 1709 1712 1713 1716 -- a loop counter
J7	1132 1748 1752 -- parameter display control index
J8	1128 1744 1748 1752 -- parameters are displayed every J8 function evaluations (no display if J8=0)
K	1180 1184 1188 1204 1208 1212 1352 1356 1364 1372 1376 1384 1388 1392 1396 1448 1452 1456 1464 1468 1472 1492 1496 1500 1536 1540 1544 1568 1572 1576 1580 1588 1604 1608 1612 1648 1652 1656 1664 1668 1672 1676 -- a loop counter
K$	1260 1336 1340 1400 1416 1480 1516 1556 1710 1714 -- indicator string for type of polytope operation
K1	1196 1216 1220 1225 1226 1227 1228 1232 1252 1272 1300 1444 1456 1464 1468 1532 1652 1772 1776 1784 1788 -- an index variable (often refers to the lowest vertex in the polytope)
K3	1276 1312 1356 1392 1496 1504 1524 1540 1548 1560 1572 1580 1596 1608 1616 1672 -- an index variable (often refers to the highest vertex in the polytope)
N	1168 1180 1192 1204 1240 1264 1284 1352 1384 1448 1492 1504 1524 1532 1536 1548 1560 1568 1596 1604 1616 1648 1664 1700 1764 -- number of parameters in loss function
N2	1168 1172 1200 1280 1360 1372 1392 1440 1580 1672 -- number of dimensions in polytope
O(	1216 1388 1452 1576 1668 1704 1708 1709 1712 1713 1772 1776 1780 -- bound/mask information storage
Q$	1768 1772 1776 1780 1784 1788 -- used to hold display information regarding masks and active bounds
S1	1092 1096 1100 1220 1225 1226 -- stepsize used to build the initial or restart polytope
S5	1104 1116 1120 1392 -- polytope reflection factor
S6	1112 1116 1120 1464 1484 1508 1672 -- polytope reduction and shrinkage factor
S7	1108 1116 1120 1580 -- polytope extension factor
S8	1432 1460 1484 1632 1636 -- "size" of polytope before shrink operation
S9	1436 1468 1484 1640 1644 -- "size" of polytope after shrink operation
T(	44 -- a temporary vector needed by POSTGEN.BAS

```
    W(          40  1184  1192  1208  1225  1226  1228  1232  1240
              1264  1284  1356  1364  1372  1392  1456  1464  1468
              1496  1504  1524  1532  1540  1548  1560  1572  1580
              1596  1608  1616  1652  1672  1772  1776  1784  1788
              -- an array to store the polytope and function values
    X(        1652 -- storage for the "best" parameters found so far
=======================================================================
    LINES: 218      BYTES: 8360     SYMBOLS: 56     REFERENCES: 378
```

6-2. Use of the Nelder - Mead Method

Again we use the Hobbs weed infestation example (Section 2-1) to illustrate the use of a method. From the suggested starting point of (2,5,3), a compiled version of the program code given in the previous section required 31 seconds and 129 sum of squares evaluations to find a suggested minimum value of the sum of squares loss function of 2.587251 with parameter estimates

B(1) = 1.961705
B(2) = 4.909004
B(3) = 3.135799

In interpreted GWBASIC version 1.12.03, 130 seconds and 110 function evaluations were required to find a sum of squares of 2.587286 and

B(1) = 1.961601
B(2) = 4.909224
B(3) = 3.135929

Both calculations were carried out under MS-DOS version 2.11 on a No-Name IBM-PC compatible computer. (Note that Microsoft does not use exactly the same arithmetic or special function procedures in the interpreted and compiled versions

of their BASIC programming language processors.)

From a starting point $(1, 1, 1)^T$ the Nelder - Mead method converges to approximately the same minimum of the sum of squares loss function, and does so relatively quickly. Listing 6-2-1 shows a part of the screen output for this calculation, which was performed with a compiled program.

Listing 6-2-1. Part of the screen output for estimation of the parameters of the Hobbs weed infestation problem using the Nelder - Mead polytope minimization method.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
22:12:34       04-17-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(2)*I))

HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 1
B( 2 )= 1
B( 3 )= 1
 are masks or bounds to be set or altered ([cr] = no)
NM -- nelder-mead polytope method -- 851110
stepsize to build polytope = 1
strategy -- refln, extnn and redn factors  1  1.2  .5
function at initial guess = 10685.29
vertex  2  =  4173.159
vertex  3  =  15809.71
vertex  4  =  3136.244
build         4    3136.244            15809.71
hi-contr'n    6    3136.244            10996.78
refl'n        7    3136.244            10685.29
hi-contr'n    9    3136.244            5601.207
lo-contr'n    11   2588.995            5387.446
refl'n        12   2588.995            4173.159
refl'n        13   2588.995            3327.629
refl'n        14   2588.995            3239.982
hi-contr'n    16   2129.675            3136.244
              . . . . . . . .

refl'n        196  2.588759            2.590579
hi-contr'n    198  2.587906            2.589543
hi-contr'n    200  2.587718            2.589185
```

```
 parameters   1.961206    4.908348    3.136249
   ELAPSED SECS= 46  AFTER 0  GRAD & 202  FN EVAL
 CALCULATED FUNCTION MINIMUM =  2.587718
 PARAMETER ESTIMATES
 B( 1 ) =  1.961206
 B( 2 ) =  4.908348
 B( 3 ) =  3.136249
 POSTGEN.BAS 851118 -- GENERAL POST SOLUTION ANALYSIS
 LOSS FUNCTION PER DEGREE OF FREEDOM (SIGMA^2) = .2875243
B( 1 )= 1.961206E+00  step= 6.78E-03  f-, f+  2.8559E+00  2.9034E+00
B( 2 )= 4.908348E+00  step= 1.70E-02  f-, f+  2.8021E+00  2.6464E+00
B( 3 )= 3.136249E+00  step= 1.08E-02  f-, f+  3.7403E+00  3.8358E+00
        best function value found is             2.5877E+00

 radii of curvature for surface along axial directions
   & tilt angle in degrees

 for B( 1 )  R. OF CURV. =  7.582E-03   tilt =  -74.04022
 for B( 2 )  R. OF CURV. =  2.187E-01   tilt =   77.71503
 for B( 3 )  R. OF CURV. =  9.038E-03   tilt =  -77.21677

 RESIDUALS
  1.126575E-02 -3.324604E-02  9.184265E-02  .2091293  .3938236
 -5.518723E-02 -1.101715  .7217407 -9.967804E-02 -.3387413
  .6630478 -.2778015
```

7

MINIMIZATION USING THE GRADIENT

7-0. Overview -- Gradient Minimization

In this chapter we introduce methods for minimizing a function of several parameters using information about the gradient of the function as well as the function values. Such methods originated very early in the literature -- Newton's method in the seventeenth century, its sum of squares variant with Gauss (1809) and descent methods were discussed by Cauchy (1848). Nevertheless, the more successful algorithms for automatic function minimization based on gradient ideas are slightly more recent than the direct search methods we have discussed in Chapter 4.

As we have noted, computation of the first derivatives of a function requires the user to provide program code. If there are n parameters, then the gradient has n components (the n first partial derivatives of the function with respect to each of the parameters). Each derivative is itself a function of the n parameters, so the amount of work could be as much as (n + 1) times as great as simply calculating the function value. Usually, however, there are simplifications, so that computing the function and gradient may typically involve only 2 to 3 times as much work as calculating the function value.

Second derivatives of the function, needed for the Hessian matrix in a true Newton's method, require the calculation of the n*n = n^2 second derivatives. Even with the simplifications which may be possible, it is clear that computation of the Hessian will require work, time, and program code which we may prefer to avoid.

In our experience spanning nearly 20 years with nonlinear parameter estimation problems, the derivative computation program code is very prone to errors. Better than 80% of the reported failures of our algorithms for function minimization or nonlinear least squares can be traced to such mistakes. Roughly the same percentage applies to delays in work we carry out ourselves in this area. As we have pointed out, it may be possible to arrange for automatic preparation of the derivative evaluation code (Bard, 1970, pages 323-324; Shearer and Wolfe, 1985), but we cannot assume that our readers will have such facilities. As an alternative, we presented programs for testing the function and its derivatives in Sections 3-7 and 3-8. We caution users to check derivative evaluation code and scalings of the function and parameters whenever the convergence behaviour of an algorithm is unsatisfactorily slow.

7-1. Motivation and Characterization

There are a great many function minimization methods based on the gradient. Cauchy (1848) suggested the steepest descents method. This uses the now well-known calculus result that the vector which is the negative of the gradient is locally the direction of the greatest rate of reduction of the function. That is, at point $\underline{X}$, the negative gradient

(7-1-1) $\underline{t} = -\underline{g}(\underline{X})$

where

(7-1-2) $g_k(\underline{B})$ = Partial derivative of $f(\underline{B})$ w.r.t $B(k)$

is the direction which locally reduces the function most rapidly. This is easily demonstrated by the Taylor expansion of $f(\underline{B})$ about $\underline{X}$ where

(7-1-3) $\underline{B} = \underline{X} + \text{stepsize} * \underline{t}$

(7-1-4) $f(\underline{B}) = f(\underline{X}) + \text{stepsize} * \underline{g}(\underline{X})^T \underline{t}$
$+ \text{stepsize}^2 * \underline{t}^T H \underline{t} + \ldots$

where H is the Hessian matrix whose elements are the second partial derivatives of the function with respect to the parameters, evaluated at $\underline{X}$. If the stepsize is small, then only the second term in the expansion is significant, so that choosing t as in (7-1-1) gives the steepest descent.

When the stepsize is not small, the "steepest descent" direction may very soon lead uphill. A classic failing of steepest descents can be readily visualized. Suppose a function of two parameters is such that its surface is a long thin valley. If we are situated on the side of the valley, the steepest descent is toward the floor of the valley. However, the minimum may be some way along the floor of the valley, so that using only the gradient can result in a rather ineffective function minimization method.

To improve on steepest descents, we may try to introduce search directions which span the dimensionality of the space on which the function is defined. Ideally, such directions would be orthogonal to each other. In practice, some set of conjugate directions is developed. One of the more successful approaches considers the minimization of a quadratic function

(7-1-5) $f(\underline{B}) = \underline{B}^T A \underline{B} - \underline{B}^T \underline{C}$

which is minimized by the solution of the linear equations

(7-1-6) $A \underline{B} = \underline{C}$

A is a symmetric, positive definite matrix in the sample problem (7-1-5). Hestenes and Stiefel (1949) developed the very compact conjugate gradients method for solving such linear equation problems (7-1-6) based on ideas of minimizing (7-1-5). Fletcher and Reeves (1964) turned the idea around to minimize general nonlinear functions by this method by considering them to be "locally quadratic". Clearly, this will not always be a reasonable approximation. Nevertheless, the approach does generate a useful set of search directions. Moreover, it is very compact in program code and especially in working storage, since it manages to require only the last search direction and the current gradient to generate the next search direction. This new search direction is supposed to be conjugate to its predecessors. For a space of n parameters, we can have just n conjugate directions. For a quadratic function, it can be shown that the minimum will have been found after just n iterations providing exact one-dimensional minima are computed along each of the search directions and then used as the starting point for the next iteration. In practice, it is not uncommon to find the minimum of a function in far fewer than n iterations, though poor scaling or extreme nonlinearity will cause convergence to be much slower.

Better search directions can be calculated if more information is kept, so that a better picture of the function is available to the minimizing method. Such modifications of conjugate gradients methods yield algorithms very similar to those which result from modifications of Newton's method, which is the other starting point for developing function minimization methods based on the gradient.

Newton's method is designed to solve a set of nonlinear Equations

(7-1-7) $\underline{g}(\underline{B}) = \underline{0}$ (null vector)

for the set of parameters $\underline{B}$. If the set of functions $\underline{g}(\underline{B})$ is the gradient of f($\underline{B}$), then Equations (7-1-7) are the first order conditions for a local minimum (or maximum or saddle point). Using the Hessian matrix H, Newton's method finds a sequence of iterates $\underline{X}$ which is developed by solving the following Equations

(7-1-8) $H\,\underline{t} = -g(\underline{X})$

for $\underline{t}$ and then using this shift to find the next iterate

(7-1-9) $\underline{X}' = \underline{X} + \underline{t}$

Newton's method requires the calculation of n partial derivatives of f($\underline{B}$) for the gradient $\underline{g}$ and n(n+1)/2 second partial derivatives for the Hessian matrix H. The symmetry of the Hessian reduces the amount of work from n*n to n(n+1)/2, but the level of effort to apply Newton's method to a problem is still formidable.

The calculations involved in solving (7-1-8) for the step vector $\underline{t}$ are straightforward as long as H is not singular. However, the update (7-1-9) must usually be safeguarded against situations where H is "nearly singular". Even so, the program required to implement such a modified Newton's method is not large. The main objection to its use is the large amount of derivative calculation code which must be written by the user. While we do not present program code for a modified Newton's method, we recommend its use in the particular case where the same estimation problem is to be solved many times. In this case the effort to program and verify the derivative calculations is repaid by the (usual) rapidity of solution. As programs for symbolic differentiation which can generate the required program code automatically become more widely available, Newton-like methods may gain in popularity.

To avoid computing analytic second derivatives, finite-difference approximation of the Hessian could be employed. Using expressions similar to those in the test programs FGTEST and RJTEST of Sections 3-7 and 3-8, an approximate Hessian could be constructed. Note that this would require (n+1) gradient evaluations at each major iteration. (Some saving of effort might be possible by using the symmetry of the Hessian, but an objectionably large amount of work is still required.) Davidon (1959) noted that it is possible to "build up" the Hessian as points are discovered which lower the function value. Such quasi-Newton Methods -- also referred to as variable metric methods -- have proven highly successful in practice. Indeed, the implementation of similar ideas by Fletcher and Powell (1963) has made the FLEPOMIN program one of of the most widely used function minimizing programs. It has been recommended by Bard (1967; 1970; 1974, section 5-12) for nonlinear estimation problems.

Quasi-Newton (or variable metric) methods still require an approximation to the Hessian to be stored. For certain problems having a very large number of parameters, this may render them impossible to use because the accessible memory of the computer available is too small. ("Accessible" is an important word here, since many popular programming language processors, such as Microsoft BASIC and Turbo Pascal (vn. 3) cannot use more than a 64k segment of memory in an IBM-PC or compatible machine, even if much more physical memory is present.) In such cases, researchers have tried to find function minimization methods with the convergence properties of Newton-like methods and the storage requirements of the conjugate gradients method. The truncated Newton method offers these properties (Dembo and Steihaug, 1983; Nash, 1982; 1985a & 1985b). The fundamental approach is to solve the Newton Equations (7-1-8) by the linear conjugate gradients algorithm, noting

1. that H is never needed explicitly, but only as a matrix-vector product

(7-1-10) $\underline{v} = H\ \underline{U}$

2. that the elements of H needed can be created as new points are tested in the function minimization.

As mentioned above, the resulting programs bear a strong resemblance to some of the variants of the nonlinear conjugate gradients methods.

To complete this survey of gradient methods, we consider problems which involve the minimization of a function which is a sum of squared terms. Here we can specialize most of the previous methods, but, in particular, Newton's method. Gauss (1809) noted that for a function which is of this special form, the Hessian has two parts:

1. representing a sum of terms in which only first partial derivatives of the residuals appear;
2. containing products of the residuals and their second derivatives.

By arguing that the residuals ought to be small, so that the second set of terms could be ignored for his problem, Gauss derived a highly efficient method for parameter estimation -- the Gauss-Newton method. As we shall see in Chapter 11, some important modifications of these ideas are needed to safeguard this approach against badly scaled problems or those with "large" residuals.

7-2. Perspective

To summarize the results of the previous section, we present a block diagram of the relationships between the various methods as Figure 7-2-1. More details are presented in the chapters describing the specific methods.

The criteria for choice allow us to eliminate gradient methods when the loss function is <u>non-smooth</u> in the region of interest in the parameter space. In such cases the Nelder - Mead polytope method (NM) is generally preferred, with the Hooke and Jeeves method (HJ) as a possibility if storage is extemely limited and rate of convergence unimportant. If the loss function is smooth, we could still use a gradient method but approximate the required derivatives numerically. Our preference here is to use the variable metric method VM (Chapter 10) with the code segment NUMGRAD to compute gradients when a general loss function must be minimized, since VM appears to require fewer gradient evaluations than either the conjugate gradients method CG or truncated Newton method TN. For sum of squares loss functions, the code segment NUMJAC may be used with the Marquardt - Nash code MRT. In all the work in this book, forward difference approximations to derivatives have been used. That is, we compute a derivative at point <u>X</u> by taking the difference of the function at (<u>X</u> + <u>step</u>) and the function at <u>X</u>. In some cases, central difference approximations may be preferable. These use differences between the function at (<u>X</u> + <u>hstep</u>) and (<u>X</u> - <u>hstep</u>). We have not included modifications of our codes for central difference approximations, and in our own work have tolerated possible slower convergence or extra human effort rather than prepare the necessary modified programs.

When the number of parameters is very large, either conjugate gradients or truncated Newton methods are appropriate. We suspect that the latter ought to be more efficient in general, but have provided both methods here so that comparisons may be made. The structure of these methods is such that TN will usually require relatively more gradient evaluations than function evaluations, while CG generally

uses several function evaluations for each gradient computed.

For a modest number of parameters and a loss function which can be put in a sum of squares form, the overwhelming weight of experience suggests that a Marquardt method (Sections 11-4 and 11-5) is the most efficient. When the loss function is not a sum of squares form, and the number of parameters to be estimated is modest, the variable metric method is the candidate of choice. If problems are well-scaled, the variable metric method also performs satisfactorily for sum of squares loss functions. Therefore, some users may find it to be the tool of choice for the bulk of their nonlinear parameter estimation work.

```
Steepest descents --> Apply conjugacy --> Conjugate gradients
  (Cauchy, 1847)      of search            (1964) (Chapter 8)
                      directions

Newton's method --> Use sequential finite --> Variable metric
 (c. 1690)          difference approximation    method (1960)
                    of Hessian
                --> Use linear cg algorithm --> truncated
                    to solve sequentially       Newton
                    approximated Newton         method (1982)
                    equations
                --> Approximate Hessian     --> Gauss-Newton
                    for sum of squared          method
                    residuals by ignoring       (c. 1809)
                    second derivatives
                --> Combine steepest        --> Marquardt's
                    descents and Gauss-         method
                    Newton direction            (1963)
```

Figure 7-2-1. Relationships between various gradient methods for function minimization.

7-3. Masks and Bounds

In all the gradient minimization methods presented in this book, we shall proceed at each iteration towards a minimum of a function f(B) from some starting point X using a search direction t and some stepsize. That is, trial points of the form

(7-3-1) B = X + stepsize * t

are generated and the function value f(B) evaluated.

In a large number of real-world situations, bounds can be provided on the parameters B so that

(7-3-2) lower(k) $\le$ B(k) $\le$ upper(k)

for k = 1,2,...,n. It is also common that users may wish to fix certain of the parameters in some estimation trials. That is, one or more of the parameters may be masked so that it is not changed during the iterations. This is a useful possibility if there is reason to suspect that a parameter genuinely has some value which can be provided from information arising outside the data and estimating loss function. It allows a number of "what if" scenarios to be evaluated easily.

Masks can be incorporated in several ways.

1. We could provide an index vector of "active" parameters. Inactive parameters are left unchanged in each iteration.

2. A set of equality constraints could be imposed on the loss function.

3. Lower and upper bounds on the parameters to be masked could be set equal to each other.

4. The parameters to be masked could be chosen to be those at the "end" of the vector of parameters. Suppose that the general problem has n parameters, but there are several

masked parameters, leaving n' parameters to be varied to minimize the loss function. Then parameters B(n'+1), B(n'+2),.....B(n) will not be altered if the minimization routine is applied to minimize a function of n' parameters. Note that the function sub-program still needs to use all n parameters. Only n' gradient components need to be computed.

5. If the gradient component is zero in the direction of one parameter, say B(k), then B(k) could be masked simply by altering the gradient sub-program to set g(k) to zero. In practice, this may result in singularity of the Hessian approximation for methods from the Newton family of algorithms unless these have been coded so that such situations cause no difficulty.

In this book, we will illustrate these methods, though we strongly advise users NOT to use the approach of forcing lower and upper bounds together. Such approaches may lead to a false convergence of gradient methods which cannot find a "downhill" search direction because the search direction t is parallel to one of the masked parameter axes.

Bounds are handled by noting that the stepsize can be adjusted to avoid points B outside the bounds. We assume that the search direction t is such that the stepsize is positive. Consider the task of finding the maximum stepsize, maxstep, from X along a search direction t which does not violate the bounds. Since

(7-3-3) lower(k) $\le$ X(k) + stepsize * t(k) $\le$ upper(k)

we consider each parameter in turn and note that there are three possibilities:

1. t(k) = 0, in which case there is no restriction on the stepsize;

2. t(k) > 0, in which case only the upper bound is restrictive, and limits stepsize to

(7-3-4) stepsize $\le$ [upper(k) - X(k)] / t(k)

3. t(k) < 0, in which case only the lower bound is

restrictive, limiting the stepsize to

(7-3-5) stepsize $\leq$ [lower(k) - X(k)] / t(k)

The maximum stepsize, maxstep, is therefore the smallest of the stepsizes allowed for each set of parameter bounds.

In the next section we provide a generalized parameter step subroutine incorporating masks and bounds.

7-4. A General Parameter Step Algorithm

To implement some of the ideas of the previous section, we need to define information storage mechanisms for the bounds and masks. We have chosen to put this information, which is not always required, in a single array labelled by the letter O (oh). This is used as follows:

1. The lower bound on parameter k is

(7-4-1) O(k,1) = lower(k)

2. The upper bound on parameter k is

(7-4-2) O(k,2) = upper(k)

3. A mask on parameter k is specified by setting

(7-4-3) O(k,3) = 0 (zero)

If O(k,3) is positive, no mask is applied. O(k,3) may be thought of as a scaling factor for gradient component G(k), and could possibly be used in this way. We set O(k,3) = 1 for unmasked parameters. Within the gradient minimization programs, we also use the third column of O(,) to store information about active constraints. We do this by setting O(k,3)=-2 if B(k) is at its lower bound. O(k,3)=-1 if B(k) is at its upper bound. This information is useful for convergence testing when bounds constraints are active.

The generalized step procedure outlined below has been incorporated in all our gradient minimization methods, with minor changes in detail for programming convenience.

1. By testing each dimension of the search vector, determine if the stepsize needs to be reduced in order that bounds are not violated. Reduce the stepsize if necessary.

2. Adjust each parameter by adding the stepsize determined above to the appropriate element of the search vector. Note that masked parameters are not altered.

3. Count the number of parameters which have been changed so that (new_parameter + E5) is different from (old_parameter + E5), where E5 is a relative scaling factor (we use a value of 10). Parameters deemed to be unchanged are reset to the "old" value to avoid possible drift of the parameters from values for which function values have been calculated.

Note that other forms of the stepping procedure are possible, and some authors provide more than one stepping routine (Schnabel et al., 1985).

7-5. Finite Difference Approximation of the Gradient

In using gradient-based function minimization techniques, it is usually up to the user to provide program code for not only the objective function but also its first partial derivatives, that is, the components of the gradient vector $\underline{g}(\underline{B})$. This task is frequently tiresome and error-prone. For the user not concerned with computation time, it may be less work to employ numerical approximations to the gradient components. Following Nash (1979, page 179ff), we use the following simple but quite effective forward difference approximation for a single component of the gradient:

(7-5-1) $g(\underline{B})_j = [f(\underline{B} + E3 * \underline{e}_j) - f(\underline{B})] / h$

where $\underline{e}_j$ is the j-th column of a unit matrix of order n and

(7-5-2) E3 = E4 * (ABS(B(j)) + E4)

and E4 is the square root of the machine precision for the available floating-point arithmetic times a heuristic scaling value (we use 10). This idea is incorporated in the following program segment NUMGRAD.BAS displayed in Listing 7-5-1. Note that we do not increment the function evaluation counter I9 in this code, since the gradient evaluation counter I8 is incremented within the function minimization methods. The true number of function evaluations should be computed as the sum of I9 and N*I8, while the number of gradients evaluated analytically is zero.

Listing 7-5-1. NUMGRAD.BAS, a program segment to calculate forward difference approximations to the gradient.

```
              NUMGRAD.BAS       04-20-1986   12:07:56

     2500 REM NUMGRAD--NUMERICAL CALCULATION OF GRADIENT
     2501 REM CALLS:
     2502 REM    FUNCTION F -- line 2000
     2503 REM
     2504 REM INPUTS:
     2505 REM    E5   -- a number for relative equality tests
     2506 REM    E9   -- the machine precision
     2507 REM    B()  -- the parameter values
     2508 REM    F    -- the function value for the given B()
     2509 REM    N    -- the number of parameters
     2510 REM    O(,) -- the parameter bounds and masks (see Sect. 3-7)
     2511 REM
     2512 REM OUTPUTS:
     2513 REM    G()  -- the numerical approximations to the gradient
     2514 REM
     2515 LET E4=E5*SQR(E9): REM step for numerical gradient
     2518 LET F4=F: REM store original fn
     2520 FOR J9=1 TO N: REM loop over B
     2525 IF O(J9,3)=0 THEN 2580: REM check for masks
     2530 LET D0=B(J9): REM save parameter
     2535 LET E3=E4*(ABS(D0)+E4)
     2540 LET B(J9)=B(J9)+E3
     2550 GOSUB 2000: REM fn evaluation (no check here for computability)
     2560 LET G(J9)=(F-F4)/E3
     2570 LET B(J9)=D0: REM restore parameter
     2580 NEXT J9
     2590 LET F=F4: REM restore original fn
     2600 RETURN
```

```
     Line no.    Referenced in line(s)
      2000      2550 -- the loss function subroutine
      2580      2525

      Symbol     Referenced in line(s)
     B(         2530  2540  2570 -- the parameter vector
     D0         2530  2535  2570 -- used to save parameter value
     E3         2535  2540  2560 -- stepsize for gradient approximation
     E4         2515  2535 -- common stepsize factor
     E5         2515 -- relative change factor (usually = 10)
     E9         2515 -- the machine precision
     F          2518  2560  2590 -- the loss function value
     F4         2518  2560  2590 -- to save function value on entry
     G(         2560 -- the gradient vector
     J9         2520  2525  2530  2540  2560  2570  2580 -- loop counter
     N          2520 -- the number of parameters
     O(         2525 -- the mask and bound array
=======================================================================
     LINES: 29      BYTES: 984     SYMBOLS: 14     REFERENCES: 31
```

COVER SHEET

Chapter 8

Chapter title: The conjugate gradients method

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

8

THE CONJUGATE GRADIENTS METHOD

8-0. Overview -- Conjugate Gradients

The conjugate gradients (cg) method first appeared in the early years of automatic digital computers as a method for solving sets of linear equations -- usually having a large, sparse, symmetric and positive- definite coefficient matrix -- where the computer to be used had a very limited memory capacity. Fletcher and Reeves (1964) applied the ideas of earlier workers, notably Hestenes and Stiefel (1952), to the minimization of a general nonlinear function by considering that the general function could be locally approximated by a quadratic function which the cg method could minimize.

The linear cg algorithm is still important in a number of situations where sets of linear equations must be solved. In recent years, pre-conditioned variants of the method have been shown to be highly efficient for solving extremely large sets of linear equations. In the present work, we shall consider solving approximate Newton equations by techniques similar to conjugate gradients to obtain the truncated Newton method of S. G. Nash (1982, 1985).

In specifics, we wish direction $\underline{t}$ to be independent of previous directions. Fletcher and Reeves (1964) showed that the following sequence of operations should generate conjugate directions for a function f(B) which is quadratic in B, that is,

(8-0-1) $$f(\underline{B}) = \underline{B}^T H \underline{B} + \underline{V}^T\underline{B}$$

where V is a constant vector and H is a constant matrix. Dixon (1972, page 60) presents a flowchart of this method.

Fletcher-Reeves method:

Step 0: Set an initial point B

Step 1: Evaluate the gradient g; let t = g.

Step 2: Find (approximately?) a minimum of f(B + s*t) with respect to s. (Linear search)

Step 3: If no improvement can be made in the function, then stop. (Various termination criteria can be applied.) Under some circumstances, goto Step 1 to "restart".

Step 4: let g' = g(B + s*t)
let beta = $\underline{g}'^T \underline{g}' / \underline{g}^T \underline{g}$
let t = g + beta * t
Then goto Step 2. (Generate next direction)

This algorithm is very similar to that of Hestenes and Stiefel (1952), used in TN.BAS to solve sets of linear equations approximately, as discussed in the next chapter. The algorithm is not restricted, however, to quadratic functions. Indeed, though there are interesting theoretical results for quadratic objective functions, conjugate gradients can be applied to general nonlinear function. This application to nonlinear function minimization results in

quite short program codes and very small working storage requirements. In the code presented below, only four vectors of length n are used for unconstrained problems of order n. The general approach is as follows.

1. Generate a search direction t. (Steps 1 or 4 above.)
2. Search along t from the lowest point found so far, X, to find an approximate minimum along that direction. (Step 2 above.)

Variants of the formulas used by Fletcher and Reeves to generate conjugate directions (Step 4 above) have been suggested by Polak and Ribiere (1969) and Powell (1975a, 1975b).

8-1. Conjugate Gradient Source-code

The source-code for the conjugate gradient algorithm is displayed in Listing 8-1-1. To allow the user to monitor the progress of the method, symbols are displayed to indicate the following actions:

>	- "uphill" search dirn
%	- no progress in step
*	- stepsize being reduced
\	- interpolation has failed

Listing 8-1-1. The Conjugate Gradients Code.

```
        CG.BAS           05-16-1986   15:41:09

40 DIM T(25),G(25)
1000 PRINT "CG -- conjugate gradients minimizer -- 851231"
1004 PRINT #3,"CG -- conjugate gradients minimizer -- 851231"
1008 REM CALLS:
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     GRADIENT G(B)  -- line 2500
1020 REM     ENVRON (computing environment) -- line 7120
1024 REM
1028 REM INPUTS TO THE ROUTINE:
1032 REM    B() -- a vector of initial parameter estimates
1036 REM    N   -- the number of parameters in the function F(B)
1040 REM  O( , ) -- masks and bounds; assume set on input
1044 REM    I2  -- number of cycles before restart.
1048 REM           Set to N-I5+2 if I2=0 on entry.
1052 REM    I5  -- the number of masked parameters. Must be zero if
1056 REM           masks are not used.
1060 REM OUTPUT FROM THE ROUTINE:
1064 REM    X() -- a vector of final parameter estimates for the
1068 REM          values of the parameters which minimize the function.
1072 REM    F0  -- value of the function at the minimum
1076 REM    G() -- value of the gradient at the minimum
1080 REM    I8  -- the number of gradient evaluations
1084 REM    I9  -- the number of function evaluations
1088 REM
1092 REM uses n-vectors B (parameters),T,A,G,X
1096 GOSUB 7120: REM computing environment
1100 LET E8=.0001: REM for acceptable function test
1104 LET C4=1.7: REM stepsize increase factor
1108 LET C3=.3: REM stepsize reduction factor
1112 LET T9=N*E9: REM convergence tolerance
1116 LET J8=10: REM no of fn eval before parameter display
1120 LET J7=1: REM counter for parameter display
1124 LET S2=1: REM initial step length
1128 IF I2=0 THEN LET I2=N-I5+2: REM number of cycles until restart
1132 LET I9=I9+1
1136 PRINT "controls e8,c3,i2,c4";E8;C3;I2;C4
1140 PRINT #3,"controls e8,c3,i2,c4";E8;C3;I2;C4
1144 GOSUB 2000: REM compute function
1148 IF I3=0 THEN 1164: REM function cannot be computed
1152 PRINT "FUNCTION NOT COMPUTABLE"
1156 PRINT #3,"FUNCTION NOT COMPUTABLE"
1160 STOP
1164 LET F0=F: REM save function value
1168 PRINT "INITIAL FUNCTION VALUE =";F0
1172 PRINT #3,"INITIAL FUNCTION VALUE =";F0
1176 LET T9=T9*SQR(1+ABS(F0)): REM adjust size of tolerance
```

```
1180 FOR I=1 TO N: REM top of steepest descent cycle
1184 LET T(I)=0
1188 NEXT I
1192 LET G2=0: REM "old" gradient norm
1196 FOR I1=1 TO I2: REM main loop
1200 PRINT "GRADS ";I8;"  FNS ";I9;"  FN VALUE=";F0;" CYCLE ";I1;
1204 PRINT #3,"GRADS ";I8;"  FNS ";I9;"  FN VALUE=";F0;" CYCLE ";I1;
1208 GOSUB 1616: REM compute gradient & check constraints
1212 FOR I=1 TO N
1216 LET X(I)=B(I): REM save best parameter values
1220 NEXT I
1224 PRINT " G^2=";G9
1228 PRINT #3," G^2=";G9
1232 GOSUB 1676: REM display parameters
1236 IF G9<T9*T9 THEN 1344: REM done if gradient small
1240 LET G1=0
1244 IF G2>0 THEN G1=G9/G2
1248 REM if not, compute new search direction
1252 LET G2=G9: REM save "old" gradient norm
1256 LET T2=0: REM projection of gradient on search dirn
1260 FOR J=1 TO N
1264 IF O(J,3)=1 THEN 1272: REM free parameter
1268 LET T(J)=0: REM masked parameter or active constraint
1272 LET T(J)=T(J)*G1-G(J): REM G(J)=0 if B(J) constrained
1276 LET T2=T2+G(J)*T(J)
1280 NEXT J
1284 IF T2<0 THEN 1300
1288 PRINT ">";: REM symbol > implies "uphill" search dirn
1292 PRINT #3,">";
1296 GOTO 1344: REM don't uphill search in this version
1300 LET S3=0
1304 LET J5=I9: REM marker to ensure a step is taken
1308 LET S1=S2: REM S2 contains initial stepsize from last iter'n
1312 GOSUB 1548: REM step along search direction
1316 IF I6<N-I5 THEN 1360: REM check if any progress made
1320 IF I9=J5 THEN 1352: REM always make step large enough on 1st pass
1324 PRINT "%";: REM symbol % means no progress in step
1328 PRINT #3,"%";
1332 FOR J=1 TO N
1336 LET B(J)=X(J): REM ensure B reset exactly
1340 NEXT J: REM cannot proceed downhill
1344 IF I1>1 THEN 1180: REM so try steepest descent
1348 GOTO 1540: REM else converged
1352 LET S2=S2*10+1/B9: REM 1/B9 ensures eventual adequate size
1356 GOTO 1308: REM and retry step
1360 LET I9=I9+1
1364 GOSUB 2000: REM compute function
1368 IF I3=1 THEN 1376: REM computable function? No, then reduce step
1372 IF F<F0+T2*S8*E8 THEN 1392: REM acceptable point?
1376 PRINT "*";: REM symbol * means stepsize being reduced
```

```
1380 PRINT #3,"*";
1384 LET S2=C3*S8: REM reduce stepsize
1388 GOTO 1308
1392 LET S3=S8: REM save best stepsize in line search
1396 LET S4=S1: REM stepsize on entry == s8 if no bound hit
1400 LET F1=F: REM and save value
1404 LET S1=2*((F1-F0)-T2*S3): REM quadratic inverse interpolation
1408 IF S1=0 THEN 1460: REM cannot use interpolation
1412 LET S1=-T2*S3*S3/S1: REM interpolation point
1416 IF S1<=0 THEN 1460: REM ignore interpolation if step is negative
1420 IF S1>S3 AND S8<S4 THEN 1468: REM no use in trying if bounds hit
1424 GOSUB 1548: REM step to interp posn
1428 IF I6=N-I5 THEN 1468: REM any progress? (masks counted)
1436 LET I9=I9+1
1440 GOSUB 2000
1444 IF I3=1 THEN 1460
1448 IF F>=F1 THEN 1460: REM any lower function value?
1452 LET F0=F: REM save good value
1456 GOTO 1480
1460 PRINT "\";: REM symbol \ means interpolation has failed
1464 PRINT #3,"\";
1468 LET S1=S3: REM reset the stepsize
1472 GOSUB 1548: REM and the parameters
1476 LET F0=F1: REM save best function value
1480 LET S2=C4*S2: REM increase stepsize
1484 IF S2>1 THEN LET S2=1: REM but limit to 1
1488 FOR J=1 TO N: REM check bounds
1492 IF O(J,3)<=0 THEN 1528: REM masked or constrained
1496 IF (B(J)-O(J,1))<=E9*(ABS(O(J,1))+1) THEN LET O(J,3)=-2
1500 IF -(B(J)-O(J,2))<=E9*(ABS(O(J,2))+1) THEN LET O(J,3)=-1
1504 REM the tolerance in this test may need adjustment (??too small)
1508 IF O(J,3)=1 THEN 1528
1512 IF O(J,3)=-1 THEN PRINT " upper "; ELSE PRINT " lower ";: REM !!
1516 IF O(J,3)=-1 THEN PRINT #3," upper "; ELSE PRINT #3," lower ";
1520 PRINT "bound imposed on parameter ";J
1524 PRINT #3,"bound imposed on parameter ";J
1528 NEXT J
1532 NEXT I1: REM end of main loop
1536 GOTO 1180: REM restart with another main loop
1540 RETURN: REM done!
1544 REM GENSTEP--stepping subroutine with masks and bounds
1548 LET S8=S1: REM to save S1
1552 FOR J=1 TO N
1556 IF O(J,3)=0 THEN 1580: REM mask check
1560 IF T(J)=0 THEN 1580: REM zero test
1564 LET K9=1: REM default lower bound
1568 IF T(J)>0 THEN K9=2: REM upper bound check
1572 IF S8<=(O(J,K9)-X(J))/T(J) THEN 1580: REM test if within bound
1576 LET S8=(O(J,K9)-X(J))/T(J): REM new maximum stepsize
1580 NEXT J
```

```
1584 LET I6=0: REM counter for parameters unchanged in step
1588 FOR J=1 TO N
1592 IF O(J,3)=0 THEN 1608: REM mask check
1596 LET B(J)=X(J)+S8*T(J)
1600 IF E5+X(J)<>E5+B(J) THEN 1608: REM scaled comparison
1602 LET B(J)=X(J): REM reset to avoid "drift"
1604 LET I6=I6+1
1608 NEXT J
1612 RETURN
1616 LET G9=0: REM gradient norm
1620 LET I8=I8+1
1624 GOSUB 2500: REM compute gradient
1628 FOR K=1 TO N
1632 IF O(K,3)=0 THEN 1660: REM masked
1636 IF O(K,3)>0 THEN 1664: REM free
1640 IF (O(K,3)+1.5)*G(K)<0 THEN 1660: REM L.M. non-negative
1644 LET O(K,3)=1: REM free parameter
1648 PRINT "freeing parameter ";K
1652 PRINT #3,"freeing parameter ";K
1656 GOTO 1664
1660 LET G(K)=0: REM active constraint on parameter B(k)
1664 LET G9=G9+G(K)*G(K)
1668 NEXT K
1672 RETURN
1676 IF J8=0 THEN RETURN: REM no parameter display
1680 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1684 LET J7=INT(I9/J8)+1: REM increment parameter display control
1688 PRINT "parameters";
1692 PRINT #3,"parameters";
1696 FOR J=1 TO N
1700 LET Q$=""
1704 IF B(J)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1708 IF O(J,2)-B(J)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1712 IF O(J,3)=0 THEN LET Q$="M": REM masked
1716 PRINT "  ";B(J);Q$;
1720 PRINT #3,"  ";B(J);Q$;
1724 IF 5*INT(J/5)<>J THEN 1744
1728 PRINT
1732 PRINT "          ";
1736 PRINT #3,
1740 PRINT #3,"          ";
1744 NEXT J
1748 PRINT
1752 PRINT #3,
1756 RETURN

     Line no.    Referenced in line(s)
      1164      1148
      1180      1344  1536
      1272      1264
```

```
 1300      1284
 1308      1356  1388
 1344      1236  1296
 1352      1320
 1360      1316
 1376      1368
 1392      1372
 1460      1408  1416  1444  1448
 1468      1420  1428
 1480      1456
 1528      1492  1508
 1540      1348
 1548      1312  1424  1472
 1580      1556  1560  1572
 1608      1592  1600
 1616      1208
 1660      1632  1640
 1664      1636  1656
 1676      1232
 1744      1724
 2000      1144  1364  1440 -- subroutine to calculate the loss
           function
 2500      1624 -- subroutine to calculate the gradient
 7120      1096 -- computing environment subroutine

 Symbol     Referenced in line(s)
B(         1216  1336  1496  1500  1596  1600  1602  1704  1708
           1716  1720 -- the parameter vector
B9         1352 -- a "large" number (supplied by ENVRON)
C3         1108  1136  1140  1384 -- stepsize reduction factor
C4         1104  1136  1140  1480 -- stepsize increase factor
E5         1600 -- a value used for scaled comparisons (= 10)
E8         1100  1136  1140  1372 -- tolerance scaling
E9         1112  1496  1500  1704  1708 -- the machine precision
F          1164  1372  1400  1448  1452 -- the loss function value
F0         1164  1168  1172  1176  1200  1204  1372  1404  1452
           1476 -- the lowest value found for the loss function
F1         1400  1404  1448  1476 -- a temporary function value
G(           40  1272  1276  1640  1660  1664 -- the gradient
G1         1240  1244  1272 -- cg update parameter
G2         1192  1244  1252 -- "old" gradient norm
G9        1224  1228  1236  1244  1252  1616  1664 -- gradient norm
I          1180  1184  1188  1212  1216  1220 -- a loop counter
I1         1196  1200  1204  1344  1532 -- counter for steepest
            descent loop
I2        1128  1136  1140  1196 -- number of cycles before restart
           (set to N-I5+2 if I2=0 on entry)
I3         1148  1368  1444 -- flag for function not computable
I5         1128  1316  1428 -- the number of masked parameters
I6         1316  1428  1584  1604 -- a counter for the number of
           parameters which are unchanged in a step
```

```
I8         1200  1204  1620 -- counter for gradient evaluations
           performed
I9         1132  1200  1204  1304  1320  1360  1436  1680  1684
           -- counter for function evaluations performed
J          1260  1264  1268  1272  1276  1280  1332  1336  1340
           1488  1492  1496  1500  1508  1512  1516  1520  1524
           1528  1552  1556  1560  1568  1572  1576  1580  1588
           1592  1596  1600  1602  1608  1696  1704  1708  1712
           1716  1720  1724  1744 -- a loop control counter
J5         1304  1320 -- marker to ensure a step is taken
J7         1120  1680  1684 -- parameter display control index
J8         1116  1676  1680  1684 -- parameters are displayed
           every J8 function evaluations (no display if J8=0)
K          1628  1632  1636  1640  1644  1648  1652  1660  1664
           1668 -- a loop control counter
K9         1564  1568  1572  1576 -- default lower bound
N          1112  1128  1180  1212  1260  1316  1332  1428  1488
           1552  1588  1628  1696 -- number of parameters in loss
           function
O(         1264  1492  1496  1500  1508  1512  1516  1556  1572
           1576  1592  1632  1636  1640  1644  1704  1708  1712
           -- bounds and masks information storage
Q$         1700  1704  1708  1712  1716  1720 -- used to hold
           display information regarding masks and active bounds
S1         1308  1396  1404  1408  1412  1416  1420  1468  1548
           -- the stepsize
S2        1124  1308  1352  1384  1480  1484 -- initial step length
S3         1300  1392  1404  1412  1420  1468 -- best stepsize in
           line search
S4         1396  1420 -- stepsize on entry to inverse quadratic
           interpolation
S8        1372  1384  1392  1420  1548  1572  1576  1596 -- maximum
           stepsize allowed by bounds
T(           40  1184  1268  1272  1276  1560  1568  1572  1576
           1596 -- the search direction
T2         1256  1276  1284  1372  1404  1412 -- projection of
           gradient on search direction
T9         1112  1176  1236 -- convergence tolerance
X(         1216  1336  1572  1576  1596  1600  1602 -- storage for
           the "best" parameters found so far
========================================================================
  LINES: 192      BYTES: 7560     SYMBOLS: 66     REFERENCES: 305
```

8-2. Use of the Conjugate Gradients Method

The CG method, because of its low working storage requirement, is a candidate for problems involving many parameters. Like the truncated Newton method, however, it is also of use with smaller problems. In our experience, it is often very efficient on most problems, and suffers from slow convergence only when extreme nonlinearities or singularity of the Hessian cause the assumptions underlying the method to be violated.

Continuing our use of the Hobbs weed infestation example, Listing 8-2-1 shows the progress of CG in minimizing the sum of squared residuals for this problem. Here we note that CG converges "early" to a sum of squares of 2.611886, instead of the optimal value of approximately 2.587268 (taken from a calculation with MRT, Chapter 11). The parameter values are

```
B( 1 ) =  1.992342  vs. 1.961851
B( 2 ) =  4.957549  vs. 4.90915
B( 3 ) =  3.12174   vs. 3.135703
```

The Hobbs problem, as may be illustrated by plotting various "slices" of the function in different parameter dimensions, is quite badly scaled. Ratkowsky (1983, Chapter 4), demonstrates that the logistic model is often highly nonlinear, both intrinsically and in the manner in which the parameters appear, using measures developed by Bates and Watts (1980) and Box (1971). Therefore, the performance of CG on this problem should not be regarded as "poor", even though it appears to do less well than the other methods in finding the minimum precisely. As we shall point out in Chapter 12, the results presented here are improved slightly by using a different BASIC interpreter. The present example

was prepared using the Microsoft GWBASIC interpreter (version 1.12.03, supplied with a Corona Data Systems PC-21 computer). There are slight but important differences in the arithmetic used by the different Microsoft BASIC interpreters, including the BASICA interpreter supplied on IBM PC systems. These differences can be revealed by clever programming, such as that in the program PARANOIA (Karpinski, 1985; Nash, 1986c). As illustrated here, these differences may occasionally be important.

Two test problems which are more suitable for demonstrating the CG method are BALF, the Brown Almost Linear Function (Moré, Garbow and Hillstrom, 1980) and the CAL4 test function (Stephen Nash, private communication). These test functions are among those discussed in Chapter 18.

BALF, as its name implies, is "almost" linear. It has the property that it can be generated for a variable number of parameters. Here we shall use N = 10 parameters. Table 8-2-1 gives the results of applying CG to this problem. These illustrate the importance of details in the convergence of function minimization methods. The usual convergence test on the current gradient norm, as presented in CG.BAS, compares the square of the gradient norm with a tolerance equal to

```
N * N * E9 * E9 * sqrt(1 + ABS(F0))
```

where E9 is the machine precision and F0 is the value of the function at the initial point. Changing the tolerance to

```
N * E9 * sqrt(1 + ABS(F0))
```

results in much more rapid convergence to a result which is still satisfactory. Because we prefer that our programs attempt to make progress in minimizing a function whenever possible, we recommend that users leave the tolerance at the more conservative value to avoid premature convergence, unless there is sufficient understanding of the problem that the tolerance can be safely increased.

Because the gradient convergence tolerance is scaled by the size of the initial function value, restarting the minimization from a point to which CG has "converged" is a mechanism for refining the estimate of the minimum. This is also illustrated in Table 8-2-1.

The CAL4 test function has a minimum which is not zero, and offers more "realism". An edited output is given below in Listing 8-2-2.

Table 8-2-1. Performance of the conjugate gradients method on the Brown Almost Linear Function. All computations performed on the Maestro PC clone (NEC V20 processor).

Notes	Fns	Grads	Time (1)	Minimum
Single precision code, from starting point B(i)=0.5	120	49	494/-	2.39E-8
with gradient norm tolerance increased in size	12	5	53/6	2.99E-5 (a)
continued from (a)	+14	+6	+62/8	8.99E-7
Interpreted double precision code, from starting point B(i)=0.5	318	142	1572/-	9.84E-27
with gradient norm tolerance increased in size	68	28	342/39	1.44E-16(b)
continued from (b)	+14	+6	+62/-	8.99E-7

(1) Times are given in seconds for execution of the minimization using interpreted/compiled code. A hyphen indicates that the timing was not available.

- -

Listing 8-2-1. Minimization of Hobbs problem using conjugate gradients.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
20:35:41      06-10-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(3)*I))

HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
 are masks or bounds to be set or altered ([cr] = no)
CG -- conjugate gradients minimizer -- 851231
controls e8,c3,i2,c4 .0001  .3  5  1.7
INITIAL FUNCTION VALUE = 158.2325
GRADS  0   FNS  1   FN VALUE= 158.2325  CYCLE  1  G^2= 9448660
*****GRADS  1   FNS  8   FN VALUE= 3.069547  CYCLE  2  G^2= 19202.93
GRADS  2   FNS  10   FN VALUE= 2.787065  CYCLE  3  G^2= 17.62927
parameters   2.057009    4.984586    3.078698
\GRADS  3   FNS  11   FN VALUE= 2.78575  CYCLE  4  G^2= 17.45309
GRADS  4   FNS  13   FN VALUE= 2.686059  CYCLE  5  G^2= 156.7835
*GRADS  5   FNS  16   FN VALUE= 2.678276  CYCLE  1  G^2= 97.19409
                        . . .

\GRADS  18   FNS  45   FN VALUE= 2.624897  CYCLE  4  G^2= 1.064196
GRADS  19   FNS  47   FN VALUE= 2.620261  CYCLE  5  G^2= 355.7879
****\GRADS  20  FNS  53   FN VALUE= 2.619938  CYCLE  1  G^2= 542.9622
parameters   1.991934    4.95768    3.121183
GRADS  21   FNS  55   FN VALUE= 2.611899  CYCLE  2  G^2= .903874
**\GRADS  22   FNS  58   FN VALUE= 2.611896  CYCLE  3  G^2= .8893222
GRADS  23   FNS  60   FN VALUE= 2.611886  CYCLE  4  G^2= .8725742
parameters   1.992342    4.957549    3.12174
***%GRADS  24   FNS  63   FN VALUE= 2.611886  CYCLE  1  G^2= .8725742
*%  ELAPSED SECS= 121  AFTER 25  GRAD & 64  FN EVAL
CALCULATED FUNCTION MINIMUM =  2.611886
PARAMETER ESTIMATES
B( 1 ) =  1.992342
B( 2 ) =  4.957549
B( 3 ) =  3.12174
```

Listing 8-2-2. Minimization of the CAL4 test problem in 15 parameters using conjugate gradients.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
21:26:32      08-01-1986
CAL4.FN SGN FUNCTION 860629
ORDER OF PROBLEM (2<=N<=25)  15
PROBLEM PARAMETER Z9=alpha (0.9 suggested)  .9
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) N
 are masks or bounds to be set or altered ([cr] = no)
CG -- conjugate gradients minimizer -- 851231
controls e8,c3,i2,c4 .0001  .3  17  1.7
INITIAL FUNCTION VALUE = .8935536
GRADS  0   FNS  1   FN VALUE= .8935536  CYCLE  1  G^2= 4.364896
*GRADS  1   FNS  4   FN VALUE= .4727763  CYCLE  2  G^2= .801379
                      . . .

*GRADS  33  FNS  77  FN VALUE= .154806  CYCLE  17  G^2= 2.133127E-06
GRADS  34  FNS  79  FN VALUE= .1548055  CYCLE  1  G^2= 3.239604E-07
****** ELAPSED SECS= 200  AFTER 35  GRAD & 85  FN EVAL
CALCULATED FUNCTION MINIMUM =  .1548055
PARAMETER ESTIMATES
B( 1 ) =  4.430095
B( 2 ) =  3.916481
B( 3 ) =  3.452917
B( 4 ) =  3.033846
B( 5 ) =  2.654313
B( 6 ) =  2.309915
B( 7 ) =  1.996708
B( 8 ) =  1.711095
B( 9 ) =  1.449772
B( 10 ) =  1.20987
B( 11 ) =  .9886648
B( 12 ) =  .7837026
B( 13 ) =  .5928011
B( 14 ) =  .4141218
B( 15 ) =  .2454203
```

9

THE TRUNCATED NEWTON METHOD

9-0. Overview -- Truncated Newton

The truncated Newton method is one of the more recently developed function minimization techniques (Dembo and Steihaug, 1983; Nash, 1982, 1983, 1985a, 1985b) but it has well-established antecedents. The fundamental steps in this modification of the Newton iteration

(7-1-8) $\quad H\,\underline{t} = -\underline{g}(\underline{X})$

(7-1-9) $\quad \underline{X}^1 = \underline{X} + \underline{t}$

are as follows.

1. The search direction $\underline{t}$ is developed by solving Equations (7-1-8) approximately by using the conjugate gradients algorithm (Hestenes and Stiefel, 1952) with pre-conditioning (Golub and Van Loan, 1983, p. 373ff). In the algorithm we present, this pre-conditioning is simplified to a scaling operation which is based on quantities computed during the iteration. To develop the full Hessian matrix H would require the calculation of second partial derivatives of the function f($\underline{B}$) to be minimized, and would, furthermore, require memory space for their storage. However, the conjugate gradients algorithm for the solution of Equations (7-1-8) never needs H itself, but only the

result of matrix multiplication of H with a vector $\underline{u}$. That is, we must supply a rule for computing

(9-0-1) $\underline{v} = H\ \underline{u}$

but do not necessarily require the full matrix H. Further simplifications arise from the observation that the k-th row (or, by symmetry, column) of H is the partial derivative of $\underline{g}(\underline{X})$ with respect to X(k). A numerical approximation to this row (column) using a stepsize s is simply

(9-0-2) $\underline{v}' = \{\underline{g}(\underline{X} + s * \underline{e}(k)) - \underline{g}(\underline{X})\} / s$

where $\underline{e}(k)$ is the k-th column of the unit matrix of order n, that is, a column vector which has zero elements except for element k, which is 1. In the limit of s tending to zero

(9-0-3) $\lim_{s \to 0} \underline{v}' = H\ \underline{e}(k)$

In a similar fashion, we approximate the result of multiplying $\underline{u}$ by H with

(9-0-4) $H\ \underline{u} \underline{\sim} \underline{v} = \{\ \underline{g}(\underline{X} + s\ \underline{u}) - \underline{g}(\underline{X})\} / s$

Thus the effect of matrix multiplication is achieved at a cost of a single gradient evaluation. Note that

- no second derivative program code is required;
- no matrix must be stored.

2. Having developed a search direction, we must update the parameter estimates $\underline{X}$. However, it is possible that the full step to $\underline{X} + \underline{t}$ as in (7-1-9) will result in a point where the function has a higher value. Various line search strategies are possible. Here we will be content with a simple back-tracking technique wherein the test point is

(9-0-5) $\underline{B} = \underline{X} + \text{stepsize} * \underline{t}$

and the stepsize is reduced until

(9-0-6) $f(\underline{B}) < f(\underline{X}) + \text{stepsize} * \text{tol} * \underline{g}^T\ \underline{t}$

is satisfied, where tol is some tolerance used to judge if the point is acceptable (Dennis and Schnabel, 1983,

p. 188ff). More complicated line search procedures can be used (Nash, 1982), but here our emphasis on compact program codes has led us to implement the simpler line search at the possible expense of extra computational effort to minimize functions.

The truncated Newton family of algorithms all make use of the steps above, with possible extensions to the line search. A particular member of this family will depend on the following choices:

- the pre-conditioning approach used in the linear conjugate gradients solution of the Newton equations;
- the organizational variant of the linear conjugate gradients algorithm used;
- the stepsize, s, used in estimating numerical derivatives for the "matrix multiplication" (9-0-4);
- the choice of forward, backward or central difference approximations for estimating the Hessian matrix product (9-0-4), or even some combination of these approximations;
- the maximum number of linear conjugate gradients iterations allowed in each major (Newton) cycle of the method;
- convergence tolerances on the linear conjugate gradients sub-algorithm, the acceptable point test and the overall method.

These choices allow for considerable diversity in the implemented programs. Furthermore, careful program organization can reduce the number of working vectors required to implement the method. It appears that the minimum number of vectors of length n required for the truncated Newton method is nine, which is the number required in our program. Thus, on the basis of working storage, the approach is suitable for problems in a large number of parameters. Indeed, the advantages of truncated Newton methods are:

- small code length;
- small working storage;
- general Newton-like convergence properties.

The major disadvantage we perceive is the necessity of tuning the method to the computer environment at hand. A poor choice of control parameters such as tolerances and iteration limits may degrade performance. Despite this concern, we are of the opinion that the truncated Newton method will become a major tool for function minimization and parameter estimation, especially for problems where n is large. At the time of writing (late 1985), more experience is needed over a wide range of problems to allow "good" implementation choices to be made for different classes of applications.

One implementation decision we have made in the code below is to include a check if bounds on parameters are violated when the Hessian approximation is generated by numerical differentiation along a search direction. That is, we consider that the derivative can only be computed if the user has not placed a bound on one or more parameters that is breached by a step along the search direction. Clearly, we must avoid a "large" stepsize if bounds exist. If the default stepsize violates a bound we reduce the stepsize accordingly. This may make the stepsize so small that a poor estimate of the derivative is obtained. Should the step (D8) be made less than a lower limit (E7), we halt the linear cg process, and use the current solution as the TN step direction. There are, however, cases where this approach may be inappropriate.

9-1. Truncated Newton Source-code

The source-code for the truncated Newton algorithm is displayed in Listing 9-1-1. To allow the user to monitor the progress of the method, symbols are displayed to indicate the following actions:

^ - warning conjugate gradient limit reached
* - stepsize is being reduced

Listing 9-1-1. The Truncated Newton Code.

```
              TN.BAS            06-15-1986   09:42:42

40 DIM H(25),G(25),T(25),E(25),W(25),A(25),R(25)
1000 PRINT "TN - TRUNCATED NEWTON WITH BOUNDS CONSTRAINTS -- 860101"
1004 PRINT #3,"TN - TRUNCATED NEWTON WITH BOUNDS CONSTRAINTS -- 860101"
1008 REM CALLS:
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     GRADIENT G(B)  -- line 2500
1020 REM     ENVRON (COMPUTING ENVIRONMENT) -- line 7120
1024 REM
1028 REM INPUTS TO THE ROUTINE:
1032 REM    B() -- a vector of initial parameter estimates
1036 REM    N   -- the number of parameters in the function F(B)
1040 REM    M9  -- iteration limit (set to 6*N if M9<=0 on entry)
1044 REM    I5  -- the number of masked parameters. Must be zero if
1048 REM           masks are not used.
1052 REM    O() -- bounds and masks array, assumed already set
1056 REM OUTPUT FROM THE ROUTINE:
1060 REM    X() -- a vector of final parameter estimates for the
1064 REM           values of the parameters which minimize the function.
1068 REM    F0  -- value of the function at the minimum
1072 REM    G() -- value of the gradient at the minimum
1076 REM    I8  -- the number of gradient evaluations
1080 REM    I9  -- the number of function evaluations
1084 REM    G9  -- the final gradient norm
1088 REM
1092 REM uses 9 N-vectors apart from bounds O() ( = 3*N )
1096 REM DIM B(N),G(N),X(N),H(N),E(N),T(N),W(N),R(N),A(N)
1100 IF M9>0 THEN 1108: REM M9 is maximum number of iterations
1104 LET M9=6*N: REM choose a reasonable number if not pre-defined
```

```
1108 GOSUB 7120: REM computing environment
1112 LET E7=SQR(E9): REM epsilon for tests
1116 LET E8=.0001: REM acceptable point test tolerance
1120 LET J8=1: REM no of fn eval before display
1124 LET J7=1: REM counter for parameter display
1128 LET T9=N*E5*E9: REM gradient norm tolerance
1132 LET I4=INT(N/2)+1: REM number of cg iterations in inner loop
1136 PRINT "strategy - m9,e7,e8,t9,i4 ";M9;E7;E8;T9;I4
1140 PRINT #3,"strategy - m9,e7,e8,t9,i4 ";M9;E7;E8;T9;I4
1144 LET I9=I9+1
1148 GOSUB 2000: REM evaluate function
1152 IF I3=0 THEN 1168
1156 PRINT "FUNCTION CANNOT BE COMPUTED"
1160 PRINT #3,"FUNCTION CANNOT BE COMPUTED"
1164 STOP
1168 LET F0=F: REM function value
1172 PRINT "INITIAL OR RESTART FUNCTION VALUE = ";F0
1176 PRINT #3,"INITIAL OR RESTART FUNCTION VALUE = ";F0
1180 LET T9=T9*(1+ABS(F0)): REM use the initial function value for scaling
1184 FOR J=1 TO N
1188 LET H(J)=1: REM initialize preconditioning
1192 LET E(J)=1
1196 LET X(J)=B(J): REM and save "best" parameters
1200 NEXT J
1204 FOR I1=1 TO M9: REM  main loop
1208 GOSUB 1772: REM compute gradient and check constraints
1212 PRINT " CYCLE";I1;" GRADIENTS ";I8;" FNS ";I9;" FN = ";F0;
1216 PRINT #3," CYCLE";I1;" GRADIENTS ";I8;" FNS ";I9;" FN = ";F0;
1220 PRINT " G-NORM = ";G9
1224 PRINT #3," G-NORM = ";G9
1228 GOSUB 1840: REM display parameters
1232 FOR J=1 TO N: REM initialize vectors
1236 LET T(J)=0
1240 LET X(J)=B(J): REM save best parameters
1244 LET W(J)=0
1248 LET R(J)=-G(J)
1252 NEXT J
1256 IF G9>T9 THEN 1268: REM done if small gradient norm
1260 IF I1>1 THEN 1172: REM restart to save best parameters in X()
1264 GOTO 1760: REM end
1268 LET  E3=1/I1: REM linear cg convergence tolerance
1272 IF E3>G9 THEN E3=G9: REM use gradient norm if smaller
1276 FOR I7=1 TO I4: REM conjugate gradients loop
1280 LET R2=0: REM RT * Z
1284 FOR J=1 TO N
1288 LET R2=R2+R(J)*R(J)/H(J)
1292 NEXT J
1296 LET B2=0
1300 IF I7>1 THEN LET B2=R2/R1: REM cg update parameter
1304 FOR J=1 TO N
```

```
1308 LET W(J)=R(J)/H(J)+B2*W(J)
1312 NEXT J
1316 REM A * U
1320 LET D8=E7*E5: REM changed from S.G.Nash code
1324 FOR J=1 TO N: REM check stepsize for bounds violations
1328 IF O(J,3)=0 THEN 1348
1332 IF W(J)=0 THEN 1348
1336 LET K2=2: REM pointer to upper bound
1340 IF W(J)<0 THEN LET K2=1: REM or to lower bound
1344 IF (O(J,K2)-B(J))/W(J)<D8 THEN LET D8=(O(J,K2)-B(J))/W(J)
1348 NEXT J
1352 IF D8<E7 THEN 1496: REM stepsize too small
1356 FOR J=1 TO N
1360 LET A(J)=0: REM initialize
1364 IF O(J,3)<=0 THEN 1380: REM mask and active bound check
1368 IF W(J)=0 THEN 1380: REM no sense in trying
1372 LET B(J)=X(J)+D8*W(J)
1376 LET A(J)=G(J): REM not as in SGN code
1380 NEXT J
1384 GOSUB 2500: REM compute gradient in G()
1388 LET I8=I8+1
1392 LET D1=0
1396 LET D2=0
1400 FOR J=1 TO N
1404 IF O(J,3)<=0 THEN 1432: REM mask & bound check
1408 LET T8=A(J)
1412 LET A(J)=(G(J)-A(J))/D8: REM derivative (Hessian) approximation
1416 LET G(J)=T8: REM save gradient at best point
1420 LET B(J)=X(J): REM and reset parameter
1424 LET D1=D1+W(J)*R(J)
1428 LET D2=D2+W(J)*A(J)
1432 NEXT J: REM end A * U
1436 IF D2<=0 THEN 1496: REM UT* A * U
1440 LET A2=R2/D2
1444 LET R9=0: REM norm of update
1448 FOR J=1 TO N
1452 IF O(J,3)=0 THEN 1476: REM mask
1456 LET E(J)=E(J)-R(J)*R(J)/D1+A(J)*A(J)/D2: REM UPDATE
1460 IF E(J)<=0 THEN E(J)=1: REM ensure positive definite
1464 LET T(J)=T(J)+A2*W(J): REM cg step
1468 LET R(J)=R(J)-A2*A(J): REM residual update
1472 LET R9=R9+R(J)*R(J): REM residual norm
1476 NEXT J
1480 LET R9=SQR(R9)/G9
1484 IF R9<E3 THEN 1496: REM test if satisfactory cg soln
1488 LET R1=R2
1492 NEXT I7: REM end linear cg loop
1496 PRINT I7;" CONJUGATE GRADIENT STEPS"
1500 PRINT #3,I7;" CONJUGATE GRADIENT STEPS"
1504 IF I7>I4 THEN PRINT "^";:  REM warning cg limit reached
```

```
1508 IF I7>I4 THEN PRINT #3,"^";:  REM warning cg limit reached
1512 IF D2<>0 THEN 1532: REM update norm UT * A * U
1516 PRINT " UT A U = 0 ";
1520 PRINT #3," UT A U = 0 ";
1524 IF I1=1 THEN 1760: REM at solution? Check if steepest descent
1528 GOTO 1172
1532 LET T2=0: REM projection of gradient on search dirn
1536 FOR J=1 TO N
1540 IF O(J,3)<1 THEN LET T(J)=0
1544 LET T2=T2+G(J)*T(J)
1548 NEXT J
1552 IF -T2<E9*E9 THEN 1524: REM any sense in trying this direction?
1556 LET S1=1: REM step reduction line search
1560 LET S8=S1: REM find max-step and check new constraint
1564 FOR J=1 TO N
1568 IF O(J,3)<=0 THEN 1604: REM masked
1572 IF T(J)=0 THEN 1604: REM  no sense in checking
1576 IF T(J)>0 THEN 1592: REM  check upper bound
1580 LET S7=(O(J,1)-X(J))/T(J): REM distance to lower bd.
1584 IF S7>S8 THEN 1604: REM smaller step in effect
1588 GOTO 1600
1592 LET S7=(O(J,2)-X(J))/T(J): REM distance to upper bd.
1596 IF S8<S7 THEN 1604: REM already smaller step in effect
1600 LET S8=S7
1604 NEXT J
1608 LET I6=0: REM counter for parameters unchanged in step
1612 FOR J=1 TO N
1616 IF O(J,3)=0 THEN 1636: REM mask check
1620 LET B(J)=X(J)+S8*T(J)
1624 IF E5+X(J)<>E5+B(J) THEN 1636
1628 LET B(J)=X(J): REM to avoid drift
1632 LET I6=I6+1
1636 NEXT J: REM end of step
1640 IF I6=N-I5 THEN 1728: REM check if no unchanged parameters
1644 LET I9=I9+1
1648 GOSUB 2000: REM evaluate function
1652 IF I3=1 THEN 1660: REM check if function computable
1656 IF F<F0+T2*S8*E8 THEN 1676: REM acceptable point
1660 LET S1=.2*S8: REM reduce stepsize
1664 PRINT "*";: REM symbol * means stepsize is being reduced
1668 PRINT #3,"*";: REM symbol * means stepsize is being reduced
1672 GOTO 1560
1676 LET F0=F: REM save new lowest value -- end line search
1680 FOR J=1 TO N: REM impose bounds if necessary
1684 IF O(J,3)<=0 THEN 1720: REM masked or already constrained
1688 IF (B(J)-O(J,1))<=E9*(ABS(O(J,1))+1) THEN LET O(J,3)=-2
1692 IF (B(J)-O(J,2))>=E9*(ABS(O(J,2))+1) THEN LET O(J,3)=-1
1696 REM the tolerance in this test may be too small
1700 IF O(J,3)=1 THEN 1720: REM still free
1704 IF O(J,3)=-1 THEN PRINT "upper "; ELSE PRINT "lower ";: REM !!
```

```
1708 IF O(J,3)=-1 THEN PRINT #3,"upper "; ELSE PRINT #3,"lower ";: REM !!
1712 PRINT "bound activated on parameter ";J
1716 PRINT #3,"bound activated on parameter ";J
1720 NEXT J
1724 GOTO 1736: REM end l.m. test
1728 IF I1>1 THEN 1180: REM try again with steepest descent
1732 GOTO 1760: REM done
1736 FOR J=1 TO N
1740 LET H(J)=E(J)
1744 NEXT J
1748 NEXT I1: REM end main loop
1752 PRINT "ABNORMAL END -- too many major cycles"
1756 PRINT #3,"ABNORMAL END -- too many major cycles"
1760 PRINT "EXITING WITH FUNCTION VALUE = ";F0;"  GRADIENT NORM = ";G9
1764 PRINT #3,"EXITING WITH FUNCTION VALUE = ";F0;"  GRADIENT NORM = ";G9
1768 RETURN: REM end tn proper
1772 LET I8=I8+1: REM gradient and its norm
1776 GOSUB 2500: REM gradient
1780 LET G9=0
1784 FOR J=1 TO N: REM adjust gradient for bounds constraints
1788 IF O(J,3)=0 THEN 1816: REM masked
1792 IF O(J,3)>0 THEN 1824: REM free
1796 IF (O(J,3)+1.5)*G(J)<0 THEN 1816: REM L.M. non-negative
1800 LET O(J,3)=1
1804 PRINT "freeing parameter ";J
1808 PRINT #3,"freeing parameter ";J
1812 GOTO 1824
1816 LET G(J)=0
1820 REM PRINT "constraint on parameter ";J;" active"
1824 LET G9=G9+G(J)*G(J)
1828 NEXT J: REM bounds constraints on G imposed
1832 LET G9=SQR(G9)
1836 RETURN
1840 IF J8=0 THEN RETURN: REM no parameter display
1844 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1848 LET J7=INT(I9/J8)+1: REM increment parameter display control
1852 PRINT "parameters";
1856 PRINT #3,"parameters";
1860 FOR J=1 TO N
1864 LET Q$=""
1868 IF B(J)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1872 IF O(J,2)-B(J)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1876 IF O(J,3)=0 THEN LET Q$="M": REM masked
1880 PRINT "  ";B(J);Q$;
1884 PRINT #3,"  ";B(J);Q$;
1888 IF 5*INT(J/5)<>J THEN 1908
1892 PRINT
1896 PRINT "          ";
1900 PRINT #3,
1904 PRINT #3,"          ";
```

```
1908 NEXT J
1912 PRINT
1916 PRINT #3,
1920 RETURN
```

Line no.	Referenced in line(s)
1108	1100
1168	1152
1172	1260 1528
1180	1728
1268	1256
1348	1328 1332
1380	1364 1368
1432	1404
1476	1452
1496	1352 1436 1484
1524	1552
1532	1512
1560	1672
1592	1576
1600	1588
1604	1568 1572 1584 1596
1636	1616 1624
1660	1652
1676	1656
1720	1684 1700
1728	1640
1736	1724
1760	1264 1524 1732
1772	1208
1816	1788 1796
1824	1792 1812
1840	1228
1908	1888
2000	1148 1648 -- subroutine to calculate the loss function
2500	1384 1776 -- subroutine to calculate the gradient
7120	1108 -- computing environment subroutine

Symbol	Referenced in line(s)
A(	40 1360 1376 1408 1412 1428 1456 1468 -- derivative (Hessian) approximation
A2	1440 1464 1468 -- residual update parameter in cg step
B(	1196 1240 1344 1372 1420 1620 1624 1628 1688 1692 1868 1872 1880 1884 -- the parameter vector
B2	1296 1300 1308 -- cg update parameter
D1	1392 1424 1456 -- used to update preconditioning
D2	1396 1428 1436 1440 1456 1512 -- as for D1
D8	1320 1344 1352 1372 1412 -- stepsize in Hessian approximation within cg step
E(	40 1192 1456 1460 1740 -- preconditioning scaling factors

Symbol	Referenced in line(s)
E3	1268 1272 1484 -- linear cg convergence tolerance
E5	1128 1320 1624 -- a value used for scaled comparisons (= 10)
E7	1112 1136 1140 1320 1352 -- epsilon for tests
E8	1116 1136 1140 1656 -- acceptable point test tolerance
E9	1112 1128 1552 1688 1692 1868 1872 -- the machine precision
F	1168 1656 1676 -- the loss function value
F0	1168 1172 1176 1180 1212 1216 1656 1676 1760 1764 -- the lowest value found for the loss function
G(	40 1248 1376 1412 1416 1544 1796 1816 1824 -- the gradient
G9	1220 1224 1256 1272 1480 1760 1764 1780 1824 1832 -- the gradient norm
H(	40 1188 1288 1308 1740 -- preconditioning
I1	1204 1212 1216 1260 1268 1524 1728 1748 -- main loop counter
I3	1152 1652 -- flag for function not computable
I4	1132 1136 1140 1276 1504 1508 -- number of cg iterations in inner loop
I5	1640 -- the number of masked parameters
I6	1608 1632 1640 -- a counter for the number of parameters which are unchanged in a step
I7	1276 1300 1492 1496 1500 1504 1508 -- conjugate gradients loop counter
I8	1212 1216 1388 1772 -- counter for gradient evaluations performed
I9	1144 1212 1216 1644 1844 1848 -- counter for function evaluations performed
J	1184 1188 1192 1196 1200 1232 1236 1240 1244 1248 1252 1284 1288 1292 1304 1308 1312 1324 1328 1332 1340 1344 1348 1356 1360 1364 1368 1372 1376 1380 1400 1404 1408 1412 1416 1420 1424 1428 1432 1448 1452 1456 1460 1464 1468 1472 1476 1536 1540 1544 1548 1564 1568 1572 1576 1580 1592 1604 1612 1616 1620 1624 1628 1636 1680 1684 1688 1692 1700 1704 1708 1712 1716 1720 1736 1740 1744 1784 1788 1792 1796 1800 1804 1808 1816 1824 1828 1860 1868 1872 1876 1880 1884 1888 1908 -- a loop control counter
J7	1124 1844 1848 -- parameter display control index
J8	1120 1840 1844 1848 -- parameters are displayed every J8 function evaluations (no display if J8=0)
K2	1336 1340 1344 -- pointer to upper or lower bound
M9	1100 1104 1136 1140 1204 -- iteration limit
N	1104 1128 1132 1184 1232 1284 1304 1324 1356 1400 1448 1536 1564 1612 1640 1680 1736 1784 1860 -- number of parameters in loss function
O(	1328 1344 1364 1404 1452 1540 1568 1580 1592

```
              1616  1684  1688  1692  1700  1704  1708  1788  1792
              1796  1800  1868  1872  1876 -- bounds and masks
              information storage
   Q$         1864  1868  1872  1876  1880  1884 -- used to hold
              display information regarding masks and active bounds
   R(           40  1248  1288  1308  1424  1456  1468  1472
   R1         1300  1488
   R2         1280  1288  1300  1440  1488
   R9         1444  1472  1480  1484 -- residual norm
   S1         1556  1560  1660 -- the stepsize
   S7         1580  1584  1592  1596  1600 -- distance to bound
   S8         1560  1584  1596  1600  1620  1656  1660 -- maximum
              stepsize allowed by bounds
   T(           40  1236  1464  1540  1544  1572  1576  1580  1592
              1620 -- the search direction
   T2        1532  1544  1552  1656 -- projection of gradient on search
              direction
   T8         1408  1416 -- temporary storage for gradient
   T9         1128  1136  1140  1180  1256 -- gradient norm tolerance
   W(           40  1244  1308  1332  1340  1344  1368  1372  1424
              1428  1464 -- step direction in linear cg iteration
   X(         1196  1240  1372  1420  1580  1592  1620  1624  1628 --
              storage for the "best" parameters found so far
=======================================================================
   LINES: 233      BYTES: 8988     SYMBOLS: 78     REFERENCES: 423
```

9-2. Use of the Truncated Newton Method

Following our established pattern, we first illustrate TN using the Hobbs weed infestation problem. An edited console output is presented in Listing 9-2-1. Once again, it should be noted that different BASIC interpreters may give slightly different results, due to the nonlinearity of this problem (see Chapter 12).

Listing 9-2-2 shows the result of applying TN to the Brown Almost Linear Function (BALF) test problem. This output from a compiled program has been heavily edited to save space.

Table 9-2-1 illustrates the eventual quadratic convergence of the TN approach on the generalized Rosenbrock problem GENROSE (see Chapter 18).

For comparison with the performance of CG, we note that TN applied to the CAL4 test function finds a minimum function value of .1548055 after 511 seconds, 124 gradient and 31 function evaluations (interpreted BASIC on the Maestro PC clone). This compares to 200 seconds, 35 gradient and 85 function evaluations to the same calculated function minimum with CG. Parameters agree to four significant (decimal) digits.

Listing 9-2-1. Solution of the Hobbs problem using the truncated Newton method in interpreted BASIC.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
21:34:22      06-17-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(3)*I))

HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
 are masks or bounds to be set or altered ([cr] = no)
TN - TRUNCATED NEWTON WITH BOUNDS CONSTRAINTS -- 860101
strategy - m9,e7,e8,t9,i4  18  3.45267E-04  .0001  3.576279E-06  2
INITIAL OR RESTART FUNCTION VALUE =  158.2325
 CYCLE 1  GRADIENTS  1  FNS  1  FN =  158.2325  G-NORM =  3073.867
parameters   2    5    3
 1  CONJUGATE GRADIENT STEPS
 CYCLE 2  GRADIENTS  3  FNS  2  FN =  71.54031  G-NORM =  2083.213
parameters   2.01948    4.994733    3.026811
 1  CONJUGATE GRADIENT STEPS
 CYCLE 3  GRADIENTS  5  FNS  3  FN =  2.780728  G-NORM =  91.81672
parameters   2.021125    4.967728    3.105334
                       . . .

 3  CONJUGATE GRADIENT STEPS
^ CYCLE 4  GRADIENTS  25  FNS  9  FN =  2.598165  G-NORM = .8202834
parameters   1.984252    4.93296    3.123025
 2  CONJUGATE GRADIENT STEPS
 CYCLE 5  GRADIENTS  28  FNS  10  FN =  2.598144  G-NORM = .9098134
```

```
parameters   1.984239    4.932903    3.123049
 3  CONJUGATE GRADIENT STEPS
^* CYCLE 6  GRADIENTS 31  FNS 12  FN =  2.593379  G-NORM =  4.156822
parameters   1.978018    4.924927    3.126092
 1  CONJUGATE GRADIENT STEPS
 UT A U = 0 INITIAL OR RESTART FUNCTION VALUE =  2.593379
 CYCLE 1  GRADIENTS  33  FNS  12  FN =  2.593379  G-NORM =  4.156822
 1  CONJUGATE GRADIENT STEPS
 CYCLE 2  GRADIENTS  35  FNS  13  FN =  2.593081  G-NORM =  .6255055
parameters   1.978077    4.924908    3.1262
 3  CONJUGATE GRADIENT STEPS
^ CYCLE 3  GRADIENTS  38  FNS  14  FN =  2.589023  G-NORM =  2.052562
parameters   1.970245    4.920717    3.131377
                         . . .

 1  CONJUGATE GRADIENT STEPS
* CYCLE 4  GRADIENTS  55  FNS  22  FN =  2.587437  G-NORM =  .9388936
parameters   1.964402    4.912879    3.134416
 1  CONJUGATE GRADIENT STEPS
**** CYCLE 1  GRADIENTS 57  FNS 26  FN = 2.587437  G-NORM = .9388936
parameters   1.964402    4.912879    3.134416
 1  CONJUGATE GRADIENT STEPS
***EXITING WITH FUNCTION VALUE =  2.587437  GRADIENT NORM = .9388936
  ELAPSED SECS= 174  AFTER 58  GRAD & 29  FN EVAL
CALCULATED FUNCTION MINIMUM =  2.587437
PARAMETER ESTIMATES
B( 1 ) =  1.964402
B( 2 ) =  4.912879
B( 3 ) =  3.134416
```

Listing 9-2-2. Application of TN to the Brown Almost Linear Function (BALF) test problem.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
22:47:22      08-01-1986
BALF.RES BROWN ALMOST LINEAR FUNCTION 860705
ORDER (N)  10
...
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) n
 are masks or bounds to be set or altered ([cr] = no)
TN - TRUNCATED NEWTON WITH BOUNDS CONSTRAINTS -- 860101
strategy - m9,e7,e8,t9,i4  60  3.45267E-04  .0001  1.192093E-05  6
INITIAL OR RESTART FUNCTION VALUE =  273.2481
 CYCLE 1  GRADIENTS  1  FNS  1  FN =  273.2481  G-NORM =  344.5425
 1  CONJUGATE GRADIENT STEPS
 CYCLE 2  GRADIENTS 3  FNS 2  FN = 8.547947E-03  G-NORM = 2.016328
 1  CONJUGATE GRADIENT STEPS
 CYCLE 3  GRADIENTS 5  FNS 3  FN = 3.118196E-05  G-NORM =
       4.595544E-03
 2  CONJUGATE GRADIENT STEPS
```

```
* CYCLE 4  GRADIENTS 8  FNS 5  FN = 3.114266E-05  G-NORM =  .0017116
 INITIAL OR RESTART FUNCTION VALUE =  3.114266E-05
 CYCLE 1  GRADIENTS  9  FNS  5  FN = 3.114266E-05 G-NORM =  .0017116
EXITING WITH FUNCTION VALUE = 3.114266E-05  GRADIENT NORM =  .0017116
  ELAPSED SECS= 8  AFTER 9  GRAD & 5  FN EVAL
 CALCULATED FUNCTION MINIMUM =  3.114266E-05
 ...
  another set of starting parameters ( [cr] = N ) y
 ...
 ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) n
  are masks or bounds to be set or altered ([cr] = no)
 TN - TRUNCATED NEWTON WITH BOUNDS CONSTRAINTS -- 860101
 strategy - m9,e7,e8,t9,i4  60  3.45267E-04  .0001  1.192093E-05  6
 INITIAL OR RESTART FUNCTION VALUE =  3.114266E-05
  CYCLE 1  GRADIENTS  1  FNS 1  FN = 3.114266E-05  G-NORM =  .0017116
 ...
 EXITING WITH FUNCTION VALUE = 2.735857E-07 GRADIENT NORM =
     1.161126E-04
   ELAPSED SECS= 20  AFTER 26  GRAD & 6  FN EVAL
```

Table 9-2-1. Minimization of the GENROSE test function in 25 parameters using the truncated Newton method. A nonlinear scaling factor Q1=10 was used in the function. The starting parameters were B(i) = i/26.

CYCLE	GRADS	FNS	VALUE	GNORM
1	1	1	131.3281	71.14738
2	15	3	111.9527	107.8159
3	18	5	107.0254	102.103
4	32	8	96.67805	98.60008
5	42	10	82.4187	71.68604
6	56	14	81.68446	71.15248
7	62	15	74.80776	20.04004
8	76	18	74.4247	19.91689
9	86	19	73.40703	9.420466
10	93	21	73.05738	8.92196
11	97	22	72.80037	5.051223
12	101	24	72.31548	14.67333
13	115	25	72.20983	24.93863
14	123	26	70.72276	16.4816
15	129	28	69.96716	36.01245
16	137	30	69.87117	33.81849
17	143	32	68.94146	43.13652
18	148	33	66.31014	22.39637
19	151	34	65.286	44.29461
20	154	35	62.2531	64.40255
21	157	37	59.0879	48.19069
1	159	37	59.0879	48.19069
2	162	38	56.83736	64.65436
3	166	39	54.0301	120.216
4	180	41	31.5215	266.6809
5	184	42	3.319121	51.46633
6	190	43	1.085504	11.18026
7	195	44	1.006397	1.475724
8	209	45	1.000684	.9737218
9	214	46	1.000124	.1823202
1	215	46	1.000124	.1823202

10

THE VARIABLE METRIC METHOD

10-0. Overview -- Variable Metric

The name and original algorithm in this family were suggested by Davidon (1959). However, the implementation of Fletcher and Powell (1963), often known as FLEPOMIN, clearly established the power of the approach.

The essential idea in this class of algorithm, which is known also as the quasi-Newton method, is to build up an approximation to the Hessian matrix H, or its inverse, using only gradient and function value information gleaned from each stage of the minimization process. This leads to an iteration of the following type:

1. Given an initial approximate Hessian H', and an initial set of parameters $\underline{X}$, compute the gradient $\underline{g}(\underline{X})$, and find a search direction $\underline{t}$ by solving

(10-0-1) $H' \underline{t} = -\underline{g}(\underline{X})$

(If H' = H, this is Newton's method.)

2) Using some criteria for an "acceptable" new set of parameters $\underline{B}$, where

(10-0-2) $\underline{B} = \underline{X} + \text{stepsize} * \underline{t}$,

choose a stepsize. Various "acceptable point" or one-dimensional minimization (linear search) methods may be used.

3) Using the gradient at B, that is g(B), update the approximation to the Hessian or its inverse.

The differences between different members of the variable metric / quasi-Newton family of methods lie in the diversity of possibilities for:

- the initial Hessian approximation (or equivalently the initial search direction);
- the linear search strategy;
- the Hessian update formula;
- the choice of the Hessian or its inverse (or some matrix decomposition of either) as the object to update;
- related to the previous point, the manner in which Equation (10-0-1) is solved for the search direction t;
- the particular combination of the above.

Here we shall consider an implementation of a variable metric method which has used the following choices.

- The inverse Hessian approximation is used.
- The initial inverse Hessian is set to the unit matrix, so that the initial search direction is the steepest descent direction (the negative gradient). This initialization is repeated if the update cannot be performed or the linear search fails.
- The inverse Hessian is updated by using a formula due to four workers in the field, the Broyden-Fletcher-Goldfarb-Shanno or BFGS update formula (Gill, Murray and Wright, 1981, page 119). Some workers believe that the numerical stability of this update can compromise the method (Gill et al., 1981, page 122).

While noting the existence of better variants of the variable metric (quasi-Newton) approach than the one presented, we are reasonably satisfied that our code offers a reasonable balance of reliability and efficiency for a wide range of parameter estimation problems.
- The linear search is performed by examining the points generated by the stepsizes

(10-0-3) $\text{stepsize} = 0.2^k$, k=0,1,2....

in sequence until a point B is found where the function value

(10-0-4) $f(\underline{B}) < f(\underline{X}) + 0.001 * \text{stepsize} * \underline{g}(\underline{X})^T\underline{t}$

Note that the inner product between the gradient g and and the search direction t should be negative to ensure a "downhill" search. Thus, the new point is only "acceptable" if it is lower than the current iterate by some finite amount. The tolerance 0.001 is chosen to ensure some progress is made in reducing the function. These choices arose in discussions between one of the authors (JCN) and Dr.Roger Fletcher of Dundee University. They are based on Fletcher's (1970) work and presented in more detail in Nash (1979, chapter 15).

The main strengths of the variable metric method as implemented here, are the following:

1. It shares with Newton's method quadratic convergence to the minimum once "sufficiently close". That is, once in the region of the (local) minimum of a function, so that X is distance d<1 from the minimum, we find the sequence of iterates which follow to be d^2, d^4, d^8, ... from the minimum. This property is very advantageous when we can provide reasonable initial guesses for the parameters.

2. The particular implementation presented uses very little working storage -- only one array n by n for the

inverse Hessian, and 5 vectors of length n (parameters B and X, gradient g, search direction t, and "old" gradient C).

3. The method implemented also has a very short program code.

Weaknesses of this method are few. It may, with badly scaled functions or those with constraints, be unable to find an acceptable point satisfying (10-0-3), so the method may terminate at a point which is not a local minimum of the function. With such functions a more common behaviour is very slow progress towards the minimum, since quadratic convergence is only observed "near" a minimum. Functions which have a Hessian matrix which is singular at the minimum may also prove difficult to minimize. These correspond, for example, to parameter estimation problems where a linear combination of two or more parameters can be estimated, but the individual parameters may take on arbitrary sets of values satisfying the linear combination. Examples are presented in Section 14-3.

10-1. Variable Metric Source-code

Listing 10-1-1 presents the source-code VM.BAS. As in CG and TN we have used the convention that an asterisk (*) is displayed when the line search step length is reduced. Other output is largely self-explanatory, though a brief note about the gradient projection norm is in order. VM displays the inner product between the current gradient g and the current search direction t as the GRADIENT PROJECTION NORM. CG displays the inner product of g with itself, while TN displays the square root of this quantity. For steepest descent steps, where t = - g, VM and CG should therefore give the same value for the measure of the gradient size, while TN will give the square root of this value.

Listing 10-1-1. The Variable Metric Code.

```
                VM.BAS           06-15-1986   09:43:44

40 DIM G(25),T(25),C(25),H(25,25): REM VM DIMs
44 DIM A(25,25): REM POSTVM DIM to save Hessian
1000 PRINT "VM -- VARIABLE METRIC WITH BOUNDS CONSTRAINTS -- 851118"
1004 PRINT #3,"VM -- VARIABLE METRIC WITH BOUNDS CONSTRAINTS -- 851118"
1008 REM CALLS:
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     GRADIENT G(B)  -- line 2500
1020 REM     ENVRON (computing environment) -- line 7120
1024 REM
1028 REM INPUTS TO THE ROUTINE:
1032 REM    B() -- a vector of initial parameter estimates
1036 REM    N   -- the number of parameters in the function F(B)
1040 REM    O( , ) -- masks and bounds
1044 REM    I5  -- the number of masked parameters. Must be zero if
1048 REM           masks are not used.
1052 REM OUTPUT FROM THE ROUTINE:
1056 REM    X() -- a vector of final parameter estimates for the
1060 REM          values of the parameters which minimize the function.
1064 REM    F0  -- value of the function at the minimum
1068 REM    G() -- value of the gradient at the minimum
1072 REM    A() -- value of the Hessian at the minimum
1076 REM    I8  -- the number of gradient evaluations
1080 REM    I9  -- the number of function evaluations
1084 REM
1088 GOSUB 7120: REM computing environment
1092 REM DIM B(N),X(N),G(N),T(N),H(N,N),C(N) needed, A(N,N) in POSTVM
1096 LET J8=1: REM no of fn eval before parameter display
1100 LET J7=1: REM counter for parameter display
1104 GOSUB 1648: REM function in F
1108 IF I3=0 THEN 1124
1112 PRINT "FUNCTION NOT COMPUTABLE AT INITIAL POINT"
1116 PRINT #3,"FUNCTION NOT COMPUTABLE AT INITIAL POINT"
1120 STOP
1124 LET I8=I8+1: REM count gradient evaluation
1128 LET I7=I8: REM records last gradient=steepest descent count
1132 GOSUB 2500: REM gradient in G( )
1136 LET F0=F: REM save lowest value so far
1140 FOR I=1 TO N
1144 FOR J=1 TO N
1148 IF I8>I7 THEN LET A(I,J)=H(I,J): REM save dispersion information
1152 LET H(I,J)=0
1156 NEXT J
1160 LET H(I,I)=1: REM initialize Hessian inverse to unit matrix
```

```
1164 NEXT I
1168 LET I7=I8: REM save itn count at last steepest desc. dirn.
1172 LET I3=0: REM ensure "non-computable function" flag not set
1176 PRINT I8;" GRADIENTS, ";I9;" FNS, LOSS FN=";F0
1180 PRINT #3,I8;" GRADIENTS, ";I9;" FNS, LOSS FN=";F0
1184 GOSUB 1664: REM display parameters
1188 REM top of iteration
1192 FOR I=1 TO N
1196 LET X(I)=B(I): REM save "best" parameters
1200 LET C(I)=G(I): REM save last gradient
1204 NEXT I
1208 FOR J=1 TO N: REM adjust gradient for constraints
1212 IF O(J,3)=0 THEN 1240: REM masked
1216 IF O(J,3)=1 THEN 1244: REM free
1220 IF (O(J,3)+1.5)*G(J)<0 THEN 1240: REM L.M. non-negative
1224 LET O(J,3)=1: REM free parameter
1228 PRINT "freeing parameter ";J
1232 PRINT #3,"freeing parameter ";J
1236 GOTO 1244
1240 LET G(J)=0: REM active constraint
1244 NEXT J
1248 REM form  H * G, and GT * H * G
1252 LET G8=0: REM gradient times step
1256 FOR I=1 TO N
1260 LET S9=0: REM accumulator for H * G row I
1264 FOR J=1 TO N
1268 LET S9=S9-H(I,J)*G(J): REM note negative sign
1272 NEXT J
1276 LET T(I)=S9: REM so that T() solves Newton eqns.
1280 IF O(I,3)=1 THEN 1288
1284 LET T(I)=0: REM constraint forces search component to zero
1288 LET G8=G8+T(I)*G(I): REM gradient projection on search direction
1292 NEXT I
1293 IF I8=I7 THEN PRINT " GRADIENT PROJECTION NORM=";-G8
1294 IF I8=I7 THEN PRINT #3," GRADIENT PROJECTION NORM=";-G8
1296 IF G8<0 THEN 1316
1300 PRINT "UPHILL SEARCH DIRECTION"
1304 PRINT #3,"UPHILL SEARCH DIRECTION"
1308 IF I7=I8 THEN 1644: REM if dirn. is sd, can't continue
1312 GOTO 1140: REM otherwise reset to steepest descent (SD)
1316 LET D1=-G8
1320 LET S1=1: REM step reduction line search
1324 LET S8=S1: REM find max-step and check new constraint
1328 FOR J=1 TO N
1332 IF O(J,3)<=0 THEN 1368: REM masked or active constraint
1336 IF T(J)=0 THEN 1368: REM no sense in checking
1340 IF T(J)>0 THEN 1356: REM check upper bound
1344 LET S7=(O(J,1)-X(J))/T(J): REM distance to lower bd.
1348 IF S7>S8 THEN 1368: REM smaller step in effect
1352 GOTO 1364
```

```
1356 LET S7=(O(J,2)-X(J))/T(J): REM distance to upper bd.
1360 IF S8<=S7 THEN 1368: REM already smaller step in effect (<= not <)
1364 LET S8=S7
1368 NEXT J
1372 LET I6=0: REM counter for parameters unchanged in step
1376 FOR J=1 TO N
1380 IF O(J,3)=0 THEN 1400: REM mask check (count as unchanged)
1384 LET B(J)=X(J)+S8*T(J)
1388 IF E5+X(J)<>E5+B(J) THEN 1400
1392 LET I6=I6+1
1396 LET B(J)=X(J): REM to avoid "drift"
1400 NEXT J: REM end of step
1404 IF I6<N-I5 THEN 1432: REM check if no unchanged parameters
1408 IF I8=I7 THEN 1644: REM done if already steepest descent
1412 PRINT "Steepest Descent ";
1416 PRINT #3,"Steepest Descent ";
1420 GOTO 1140
1424 REM note that this convergence test is sufficient if bounded
1428 REM constraints are released at start of each iteration
1432 GOSUB 1648: REM evaluate function
1436 IF I3=1 THEN 1444: REM check if function computable
1440 IF F<F0-D1*S8*.0001 THEN 1460: REM acceptable point
1444 LET S1=.2*S8: REM reduce stepsize
1448 PRINT "*";: REM symbol * means stepsize is being reduced
1452 PRINT #3,"*";
1456 GOTO 1324
1460 LET F0=F: REM save new lowest value
1464 FOR J=1 TO N: REM check bounds
1468 IF O(J,3)<=0 THEN 1520: REM masked or constrained parameter
1472 IF (B(J)-O(J,1))>E9*(ABS(O(J,1))+1) THEN 1492: REM check lower bound
1476 PRINT "lower ";
1480 PRINT #3,"lower ";
1484 LET O(J,3)=-2: REM lower bound active
1488 GOTO 1512
1492 IF (O(J,2)-B(J))<=E9*(ABS(O(J,2))+1) THEN 1500: REM check upper bound
1496 GOTO 1520: REM no constraint
1500 LET O(J,3)=-1: REM upper bound active
1504 PRINT "upper ";
1508 PRINT #3,"upper ";
1512 PRINT "bound activated on parameter ";J
1516 PRINT #3,"bound activated on parameter ";J
1520 NEXT J
1524 LET I8=I8+1: REM prepare update -- get new gradient
1528 GOSUB 2500
1532 LET I3=0: REM & reset flag to indicate update failure
1536 LET D1=0: REM BFGS update of inverse Hessian approximation
1540 LET D2=0
1544 FOR I=1 TO N
1548 LET T(I)=T(I)*S8
1552 LET C(I)=G(I)-C(I): REM change in gradient
```

```
1553 IF O(I,3)=0 THEN LET C(I)=0: REM FOR MASK
1556 LET D1=D1+T(I)*C(I)
1560 NEXT I
1564 FOR I=1 TO N
1568 LET S9=0
1572 FOR J=1 TO N
1576 LET S9=S9+H(I,J)*C(J)
1580 NEXT J
1584 LET X(I)=S9
1588 LET D2=D2+S9*C(I)
1592 NEXT I
1596 IF D1<=0 THEN 1628: REM test of positive definite-ness of H
1600 LET D2=1+D2/D1
1604 FOR I=1 TO N
1608 FOR J=1 TO N
1612 LET H(I,J)=H(I,J)-(T(I)*X(J)+X(I)*T(J)-D2*T(I)*T(J))/D1
1616 NEXT J: REM above is main update equation
1620 NEXT I
1624 GOTO 1176
1628 PRINT "UPDATE NOT POSSIBLE"
1632 PRINT #3,"UPDATE NOT POSSIBLE"
1636 LET I3=1: REM &
1640 GOTO 1140: REM return to try steepest descent
1644 RETURN: REM end of routine
1648 LET I3=0: REM loss function evaluation -- reset flag
1652 GOSUB 2000: REM user function subroutine
1656 LET I9=I9+1
1660 RETURN
1664 IF J8=0 THEN RETURN: REM no parameter display
1668 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1672 LET J7=INT(I9/J8)+1: REM increment parameter display control
1676 PRINT "parameters";
1680 PRINT #3,"parameters";
1684 FOR J=1 TO N
1688 LET Q$=""
1692 IF B(J)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1696 IF O(J,2)-B(J)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1700 IF O(J,3)=0 THEN LET Q$="M": REM masked
1704 PRINT "  ";B(J);Q$;
1708 PRINT #3,"  ";B(J);Q$;
1712 IF 5*INT(J/5)<>J THEN 1732
1716 PRINT
1720 PRINT "          ";
1724 PRINT #3,
1728 PRINT #3,"          ";
1732 NEXT J
1736 PRINT
1740 PRINT #3,
1744 RETURN
```

```
Line no.    Referenced in line(s)
 1124      1108
 1140      1312  1420  1640
 1176      1624
 1240      1212  1220
 1244      1216  1236
 1288      1280
 1316      1296
 1324      1456
 1356      1340
 1364      1352
 1368      1332  1336  1348  1360
 1400      1380  1388
 1432      1404
 1444      1436
 1460      1440
 1492      1472
 1500      1492
 1512      1488
 1520      1468  1496
 1628      1596
 1644      1308  1408
 1648      1104  1432
 1664      1184
 1732      1712
 2000      1652 -- subroutine to calculate the loss function
 2500      1132  1528 -- subroutine to calculate the gradient
 7120      1088 -- computing environment subroutine

 Symbol     Referenced in line(s)
A(           44  1148 -- value of the Hessian at the minimum
B(         1196  1384  1388  1396  1472  1492  1692  1696  1704
           1708 -- the parameter vector
C(           40  1200  1552  1553  1556  1576  1588 -- change in
           gradient
D1         1316  1440  1536  1556  1596  1600  1612
D2         1540  1588  1600  1612
E5         1388 -- a value used for scaled comparisons (= 10)
E9         1472  1492  1692  1696 -- the machine precision
F          1136  1440  1460 -- the loss function value
F0         1136  1176  1180  1440  1460 -- the lowest value found
           for the loss function
G(           40  1200  1220  1240  1268  1288  1552 -- the gradient
G8         1252  1288  1293  1294  1296  1316 -- gradient
           projection on search direction
H(           40  1148  1152  1160  1268  1576  1612 -- Hessian
           inverse matrix
I          1140  1148  1152  1160  1164  1192  1196  1200  1204
           1256  1268  1276  1280  1284  1288  1292  1544  1548
           1552  1553  1556  1560  1564  1576  1584  1588  1592
```

```
            1604  1612  1620 -- a loop control counter
 I3         1108  1172  1436  1532  1636  1648 -- flag for function
            not computable
 I5         1404 -- the number of masked parameters
 I6         1372  1392  1404 -- a counter for the number of
            parameters which are unchanged in a step
 I7         1128  1148  1168  1293  1294  1308  1408 -- gradient
            iterationn count at last steepest descent direction
 I8         1124  1128  1148  1168  1176  1180  1293  1294  1308
            1408  1524 -- counter for gradient evaluations performed
 I9         1176  1180  1656  1668  1672 -- counter for function
            evaluations performed
 J          1144  1148  1152  1156  1208  1212  1216  1220  1224
            1228  1232  1240  1244  1264  1268  1272  1328  1332
            1336  1340  1344  1356  1368  1376  1380  1384  1388
            1396  1400  1464  1468  1472  1484  1492  1500  1512
            1516  1520  1572  1576  1580  1608  1612  1616  1684
           1692  1696  1700  1704  1708  1712  1732 -- a loop control
            counter
 J7         1100  1668  1672 -- parameter display control index
 J8         1096  1664  1668  1672 -- parameters are displayed
            every J8 function evaluations (no display if J8=0)
 N          1140  1144  1192  1208  1256  1264  1328  1376  1404
            1464  1544  1564  1572  1604  1608  1684 -- number of
            parameters in loss function
 O(         1212  1216  1220  1224  1280  1332  1344  1356  1380
            1468  1472  1484  1492  1500  1553  1692  1696  1700
            -- bounds and masks information storage
 Q$         1688  1692  1696  1700  1704  1708 -- used to hold
            display information regarding masks and active bounds
 S1         1320  1324  1444 -- the stepsize
 S7         1344  1348  1356  1360  1364 -- distance to bound
 S8        1324  1348  1360  1364  1384  1440  1444  1548 -- maximum
            stepsize allowed by bounds
 S9         1260  1268  1276  1568  1576  1584  1588 -- accumulator
            for H * G row I
 T(           40  1276  1284  1288  1336  1340  1344  1356  1384
            1548  1556  1612 -- the search direction
 X(        1196  1344  1356  1384  1388  1396  1584  1612 -- storage
            for the "best" parameters found so far
==================================================================
 LINES: 193     BYTES: 7221     SYMBOLS: 58     REFERENCES: 307
```

10-2. Use of the Variable Metric Method

Once again, we first illustrate the VM code on the Hobbs weed infestation problem, where it performs respectably. The edited output is presented in Listing 10-2-1.

For a second example, we choose to illustrate how bounds constraints may affect how the minimization proceeds. For this purpose, we apply VM to the Hock and Schittkowski (1981) problem number 25. This is listed as the Gulf Research and Development problem by Moré, Garbow and Hillstrom (1981), which has a minor typographical error in the residual expression. Listing 10-2-2 presents an edited output. Note that the bounds constraints only once become active during the minimization process. In this example we use the Moré et al. starting point, since we have found the suggested starting point of Hock and Schittkowski to be such that our methods have great difficulty in making progress to minimize the function (the residuals are extremely large, and we suspect some overflow/underflow problems).

Listing 10-2-1. Minimization of Hobbs problem using variable metric.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
20:23:57      06-24-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(3)*I))

HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
 are masks or bounds to be set or altered ([cr] = no)
VM -- VARIABLE METRIC WITH BOUNDS CONSTRAINTS -- 851118
 1  GRADIENTS,  1  FNS, LOSS FN= 158.2325
```

```
parameters   2    5    3
 GRADIENT PROJECTION NORM= 9448660
**** 2  GRADIENTS,  6  FNS, LOSS FN= 39.92594
parameters   2.029586    4.992    3.040722
** 3  GRADIENTS,  9  FNS, LOSS FN= 39.78583
parameters   1.948403    4.982576    3.099941
* 4  GRADIENTS,  11  FNS, LOSS FN= 29.61903
parameters   1.985641    4.654778    3.113797
*** 5  GRADIENTS,  15  FNS, LOSS FN= 28.32154
parameters   1.992316    4.75184    3.128635
              . . .

 17  GRADIENTS,  27  FNS, LOSS FN= 2.587278
parameters   1.961862    4.909164    3.135698
*Steepest Descent  17  GRADIENTS,  28  FNS, LOSS FN= 2.587278
parameters   1.961862    4.909164    3.135698
 GRADIENT PROJECTION NORM= 8.184759E-05
**** 18  GRADIENTS,  33  FNS, LOSS FN= 2.587266
parameters   1.961853    4.909167    3.135687
** 19  GRADIENTS,  36  FNS, LOSS FN= 2.587259
parameters   1.961855    4.909165    3.135686
*******Steepest Descent  19  GRADIENTS,  43  FNS, LOSS FN= 2.587259
parameters   1.961855    4.909165    3.135686
 GRADIENT PROJECTION NORM= .2152535
**********  ELAPSED SECS= 110  AFTER 19  GRAD & 53  FN EVAL

           {the comparable results for CG and TN are
                CG        121        25         64
                TN        174        58         29 }

CALCULATED FUNCTION MINIMUM =  2.587259
PARAMETER ESTIMATES
B( 1 ) =  1.961855
B( 2 ) =  4.909165
B( 3 ) =  3.135686
```

Listing 10-2-1. Minimization of the test problem HS25 using the variable metric method.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
23:23:25      07-04-1986
HS25.RES -- HOCK AND SCHITTKOWSKI PROB 25
bounds on parameters for problem
 .1  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   25.6
 0  <=   b( 3 )  <=   5
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y )
B( 1 )= 5
```

```
B( 2 )= 2.5
B( 3 )= .15
 are masks or bounds to be set or altered ([cr] = no)
VM -- VARIABLE METRIC WITH BOUNDS CONSTRAINTS -- 851118
 1  GRADIENTS,  1  FNS, LOSS FN= 12.1107
parameters   5    2.5    .15
 GRADIENT PROJECTION NORM= 1578.6
** 2  GRADIENTS,  4  FNS, LOSS FN= 6.719496
parameters   4.989791    2.499831    .344
 3  GRADIENTS,  5  FNS, LOSS FN= 6.647151
parameters   5.214464    2.541513    .4037432
 4  GRADIENTS,  6  FNS, LOSS FN= 6.540985
...
 10  GRADIENTS,  13  FNS, LOSS FN= 4.286698
parameters   20.74522    10.04146    .8618659
upper bound activated on parameter  2
 11  GRADIENTS,  14  FNS, LOSS FN= .2670283
parameters   45.98727    25.6 U   1.422573
freeing parameter  2
* 12  GRADIENTS,  16  FNS, LOSS FN= .1434541
parameters   43.71592    24.1648    1.373266
...
 33  GRADIENTS,  37  FNS, LOSS FN= 6.277097E-13
parameters   50.00004    25    1.5
* 34  GRADIENTS,  39  FNS, LOSS FN= 5.475212E-13
parameters   50.00003    25    1.5
**Steepest Descent  34  GRADIENTS,  41  FNS, LOSS FN= 5.475212E-13
parameters   50.00003    25    1.5
 GRADIENT PROJECTION NORM= 3.311988E-12
*  ELAPSED SECS= 192  AFTER 34  GRAD & 42  FN EVAL
CALCULATED FUNCTION MINIMUM =  5.475212E-13
PARAMETER ESTIMATES
B( 1 ) =  50.00003
B( 2 ) =  25
B( 3 ) =  1.5
```

COVER SHEET

Chapter 11

Chapter title: Methods for sums of squared functions

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

11

METHODS FOR SUMS OF SQUARED FUNCTIONS

11-0. Overview -- Sums of Squared Functions

In this chapter we consider methods designed to minimize functions which can be expressed as the sum of squared terms

$$(11\text{-}0\text{-}1) \qquad f(\underline{B}) = \sum_{i=1}^{m} r_i^2(\underline{B}) = \underline{r}^T \underline{r}$$

The elements $\underline{r}$ are termed residuals, and, as we have noted in Equation (2-6-1)

(11-0-2) r(i,$\underline{B}$) = Z(Y(i,2),Y(1,3),...,$\underline{B}$) - Y(i,1)

where Z( ) is the modelling function.

As we have pointed out in Chapter 7, this was recognized very early (Gauss, 1809) as an important class of estimation problem. The traditional approach, following Gauss' work is to apply Newton's method to find a root of the Equations

$$(11\text{-}0\text{-}3) \qquad g(\underline{B}) = 0 = 2 \sum_{i=1}^{m} J_{ij}\, r_i(\underline{B})$$

where

(11-0-4) J_{ij} = Partial derivative of $r_i(\underline{B})$ with respect to B(j)

The m by n matrix J is the Jacobian of the sum of squares function f($\underline{B}$). To apply Newton's method, we need the Hessian matrix H, which for this problem has the form

(11-0-5) $$H_{kj} = \text{Partial of } g_k(\underline{B}) \text{ with respect to } B(j)$$
$$= \text{Partial of } g_j(\underline{B}) \text{ with respect to } B(k)$$
$$= 2 \sum_{i=1}^{m} (J_{ij}\, J_{ik} + r_i(\underline{B})\, Q_{ijk})$$

where

(11-0-6) Q_{ijk} = Partial derivative of J_{ij} with respect to B(k)
= Second partial derivative of $r_i(\underline{B})$ with respect to B(j) and B(k)

The methods to be discussed in Section 11-3 are all based more or less on the approximation of H by the first term of the sum in (11-0-5),

(11-0-7) $H \simeq 2\ J^T J$

giving the Gauss-Newton Equations

(11-0-8) $J^T J\, \underline{t} = -\, J^T\, \underline{r}$

Gauss was able to justify the use of the "approximate" Hessian by arguing that the terms involving second derivatives were multiplied by residuals, which in his problem (orbit equations of astronomical objects) were expected to be small because of the relatively high precision of the measurements used to gather the data for the problem. When the residuals are not small, this approximation of the Hessian cannot be supported in this way. "Large residual" problems, that is, problems where it is expected the residual sum of squares at the minimum will never be small relative to the size of observations, are frequently used as tests of

nonlinear least squares programs since they are expected to be troublesome to solve. In practice, we have not found such problems to be exceptionally difficult to solve, though convergence may be slow. In some cases, we increased the size of the stabilizing constant phi, presented below in Equation (11-4-5), in order to speed-up convergence.

The many particular variants of the Gauss-Newton method used to solve nonlinear least squares problems differ in the heuristics applied to overcome linear dependence in the columns of the Jacobian, J, or to safeguard the stepsize used in searching along the search direction $\underline{t}$.

11-1. Popularity of Nonlinear Least Squares

The majority of nonlinear parameter estimation problems we have encountered have been formulated using a least squares (L-2) loss function. The reasons why this should be preferred over more general maximum likelihood formulations or other loss functions such as maximum absolute deviation (L-infinity) or sum of absolute deviations (L-1) are several.

1. The loss function f($\underline{B}$) is continuous (in its unconstrained form) and has continuous first and second derivatives with respect to the residuals

(11-1-1) r(i,$\underline{B}$) = Z(i,$\underline{B}$) - Y(i,1)

Such continuity eliminates some difficulties in the computation of gradients of f($\underline{B}$) with respect to the parameters using the chain rule.

2. Much of the history and tradition of estimation has relied on least squares. Even if a worker uses an alternative loss function, comparison with the least squares estimates are likely to be computed.

3. Frequently the calculations are easily and quickly performed by standard or at least available programs. Some

of these have been adapted to a particular area of research in pharmacokinetics (e.g. Duggleby, 1984).

11-2. Conversion of Problems to Sums of Squares

Many problems present themselves directly in the form of a sum of squared terms. That is, we are able to express

(11-2-1) $$f(B) = \underline{r}^T(\underline{B})\ \underline{r}(\underline{B}) = \sum_{i=1}^{m} r^2(i,\underline{B})$$

Many situations do not have such an obviously "least squares" form. In Generalized Least Squares (GLS) modelling (Johnston, 1972), it is assumed that there is a variance - covariance (or, in short, covariance) matrix V such that the expected value of the product of the i-th and j-th residual is

(11-2-2) $$E(r(i,\underline{B}),\ r(j,\underline{B})) = V_{ij}(\underline{B})$$

From the properties of expectations, the matrix V must be non-negative definite. (A number of authors use "positive definite", which is not strictly correct. The consequence of an indefinite covariance matrix V is non-uniqueness of its factorizations below.) The loss function to be minimized is

(11-2-3) $$f(\underline{B}) = \underline{r}^T(\underline{B})\ V\ \underline{r}(\underline{B})$$

which is NOT, as it stands, a sum of squares. However, we can define some new residuals, $\underline{r}'$, which restore the familiar form.

The tool to be used here is a matrix decomposition (Nash, 1979, Section 2.5; Golub and Van Loan, 1983). The Choleski decomposition of a positive definite matrix V is the product of a lower triangular matrix L and its transpose

(Nash, 1979, chapter 7)

(11-2-4) $$V = L\ L^T$$

The elements of L may be developed in several ways, of which we shall discuss only a column-wise variant. Because L is lower triangular, we can write

(11-2-5) $$V_{ij} = \sum_{k=1}^{\min(i,j)} L_{ik}\ L_{jk}$$

Thus

(11-2-6) $$L_{11} = \mathrm{SQRT}\ (V_{11})$$

and

(11-2-7) $$L_{j1} = V_{i1} / L_{11} \qquad i=2,\ 3,\ \ldots n$$

For all other columns we have

(11-2-8) $$L^2_{jj} = V_{jj} - \sum_{k=1}^{j-1} L^2_{jk}$$

In consequence of V being singular, one or more of the sums derived via (11-2-8) should be zero (or even negative computationally). In this case we are free to set our entire j-th column of L to zero and proceed to compute the (j+1)-st column. (See Nash, 1979, Section 7.2 for a more complete explanation.)

Once the Choleski decomposition is available, a simple back-substitution allows us to find

(11-2-9) $$\underline{r}'^T\ \underline{r}'^T = \underline{r}^T\ V\ \underline{r}^T = \underline{r}^T\ L\ L\ \underline{r}$$

by solving

(11-2-10) $$\underline{r}' = L^T\ \underline{r}$$

Other decompositions of V are possible. The <u>square root</u> of V, denoted

$$V^{1/2}$$

is computed by obtaining the <u>eigenvalues</u> E and eigenvectors X of V, so that

(11-2-11) $V X = X E$

(The columns of X are the eigenvectors and the diagonal matrix E has diagonal elements which are the eigenvalues.) Many methods exist for the computation of eigensolutions of real - symmetric matrices. Nash (1979, Chapters 10, 11) gives some compact algorithms and includes a discussion of the current decomposition. Rice (1983, Section 6.7) discusses various sources of software for FORTRAN computing environments. Golub and Van Loan (1983, Chapter 8) give a fairly complete survey of the symmetric matrix eigenvalue problem.

If V is non-negative definite, the diagonal elements of E are positive or zero. Thus we can take their square roots and define a new diagonal matrix D so that

(11-2-12) $D^2 = E$

The square root of V is then

(11-2-13) $X D X^T = V^{1/2}$

since

(11-2-14) $X D X^T X D X^T = X D D X^T = V$

Here we use the fact that the eigenvectors of a symmetric matrix form an orthogonal matrix. Using this square root of V, we can define another set of residuals

(11-2-15) $\underline{r}''(\underline{B}) = V^{1/2} \underline{r}(\underline{B})$

Ingenuity may permit many other problems which are not directly formulated as least squares problems to be solved using nonlinear least squares programs. We will content ourselves with the brief suggestions given above, having provided software for more general problems elsewhere in the book.

11-3. Diverse Approaches

The Gauss-Newton Equations (11-0-8) can be solved by any method for the solution of a set of simultaneous linear equations. However, it is worth noting that the matrix

(11-3-1) $H' = J^T J$

which is an approximation to the Hessian, is non-negative definite. The matrix H' is always positive definite as long as the columns of J are linearly independent. For many practical problems, it is possible to demonstrate that the columns of J are linearly independent, so that H' is positive definite. This allows us to apply the Choleski algorithm (Dahlquist and Bjorck, 1974; Nash, 1979, Chapter 7), which is well-suited to this class of problem. Unfortunately, linear independence in a computational sense is much more difficult to ensure. Linear dependence of the columns of J, or the consequent singularity of H', are equally difficult to detect.

At this point we would like to take up some of our readers' time to point out quite forcefully that the absence of a diagnostic message to the effect that the coefficient matrix is singular, when an attempt is made to solve systems of linear equations, does NOT mean that the solution is satisfactory. As an example, consider the matrix defined by the Equations

(11-3-2) $A(i,i) = i$

$A(i,j) = \min(i,j) - 2$ for $j <> i$

However, this family of matrices has largest and smallest eigenvalues as given in Table (11-3-1), even though the Choleski decomposition of these matrices has all the diagonal elements equal to 1.

Table 11-3-1. Eigenvalues of the Moler matrix for different orders of the matrix

Order	Maximum eigenvalue	Minimum eigenvalue
5	8.64627228E-03	7.4874999
10	8.58280692E-06	31.5898097
15	8.38190280E-09	76.0920128
20	8.18545232E-12	140.8991467
25	7.99360578E-15	225.9864570
30	7.80625564E-18	331.3457707
35	7.62329653E-21	456.9736680
40	7.44462551E-24	602.8684750
45	7.27014210E-27	769.0292779
50	7.09974815E-30	955.4555364

As a safeguard against computational singularity of H', we could calculate its eigenvalues. Jones (1970) uses the eigenvalue / eigenvector decomposition of H' explicitly in his SPIRAL method. We believe a more efficient approach is to use the singular value decomposition (svd) of J and solve the linear least squares problem

(11-3-3) $J\ \underline{t} \sim \underline{r}$

for the search direction $\underline{t}$. The singular value decomposition allows a least squares solution $\underline{t}$ to be developed for (11-3-3) even if J is singular. At the same time, singular values are computed which are the positive square roots of the eigenvalues of H' (Nash, 1979, Chapter 3; Golub and Van Loan, 1983, Section 2.3). Thus we are given the opportunity to alter the solution strategy if the singular values indicate that the columns of J are nearly linearly dependent (Gill, Murray and Wright, 1981, Section 4.7.2).

The singular value decomposition has been incorporated in nonlinear least squares code by one of us (Nash, 1985; Nash, 1984b (LEQB05)). In general, however, we have preferred to use the Choleski algorithm to solve the Gauss-Newton equations, though only in the context of

Marquardt's modification (see below), which generally avoids the difficulty of the singularity of H'. The svd generally requires much more computational effort per iteration than the Choleski decomposition, though it offers considerably greater flexibility in strategy and much more information about the nature of the problem at hand.

Some brief time trials have also suggested that in interpreted BASIC the Choleski decomposition of H' into

(11-3-4) $H' = L\ L^T$

is quicker than using

(11-3-5) $H' = L\ D\ L^T$

(that is, with the diagonal elements of L fixed at 1). However, we are convinced that other computing environments may provide examples where the latter form of the Choleski algorithm is faster. Note that the Choleski decomposition and back-substitution code have been modified to allow the programs to handle masks and bounds on the parameters.

Once the search direction $\underline{t}$ has been found, various linear search techniques can be applied. Hartley (1961) chose to try to minimize the sum of squares function $f(\underline{B})$ along the search direction $\underline{t}$. He evaluated the function at $\underline{B} = \underline{X} + \underline{t}$ and $\underline{B}' = \underline{X} + 0.5\ \underline{t}$ and used $f(\underline{B})$, $f(\underline{X})$ and $f(\underline{B}')$ to provide three pieces of information to fit a quadratic function in the stepsize. We have the situation

(11-3-6)
$$f(\underline{X}) = a$$
$$f(\underline{B}) = a + b + c$$
$$f(\underline{B}') = a + 0.5b + 0.25c$$

The minimum of the parabola is at

(11-3-7) $\text{stepsize} = -\ b\ /\ 2c$

Another scheme uses the projection of the gradient $g(\underline{X})$ on the search direction $\underline{t}$ to provide the third piece of information in (11-3-6), that is

(11-3-8) $\underline{g}^T(\underline{X})\ \underline{t} = b$

In any event, we either perform an inner product and one function evaluation or two function evaluations in order to find an appropriate new test point

(11-3-9) $\underline{B} = \underline{X} + \text{stepsize} * \underline{t}$

The function must be evaluated at this point to ensure

(11-3-10) $f(\underline{B}) < f(\underline{X})$

Thus, Hartley's method is quite costly in function evaluations, and hence in potential computational effort.

Marquardt (1963) suggested an alternative which overcomes problems of singularity of H' and steplength in a single strategy. The fundamental idea is to replace the Gauss-Newton Equations (11-0-8) with

(11-3-11) $(J^T J + \text{lambda} * Q)\ \underline{t} = - J^T\ \underline{r}$

where lambda is a technical parameter used to modify the equations and Q is essentially a scaling matrix. Marquardt (1963) suggested that Q either be a unit matrix or that it have elements equal to the diagonal elements of H', that is, the current Jacobian inner product. Nash (1977) showed how a linear combination of a unit matrix and the diagonal elements of H' could be better than either in that it

- had the scaling properties of the diagonal elements of H' when these were large;
- avoided singularity in the solutions of Equations (11-3-11) when <u>computationally</u> some of the diagonal elements of H' are zero.

In practice, especially in computing environments with limited precision floating-point arithmetic, zeroes on the diagonal of H' are a quite common cause of failure of Marquardt's methods.

There remains the issue of a choice of lambda. Marquardt suggested an initial value of 0.1, which is

decreased by a factor of 0.1 after every successful step (when the function value is lower at $\underline{X} + \underline{t}$) and increased by a factor of 10 on failure. The new point $\underline{B}$ replaces $\underline{X}$ if a new lowest point is found, but $\underline{X}$ is retained on failure, which may include failure because of computational singularity in solving the Equations (11-3-11). Increasing lambda ensures that eventually

- the equations are non-singular;
- the stepsize is small enough that a new lower function value can be found; or
- the stepsize is so small no further progress can be made.

Details of this approach have been provided by Nash (1977) and Nash (1979, Algorithm 23, which has one small and obvious misprint in a looping variable).

In the present work, we have extended this code to include masks and bounds. The extension to encompass masks and bounds adds considerably to the length of the code, particularly in the Choleski decomposition of H'. We now avoid altering parameters which are masked or which are constrained by active bounds by explicitly zeroing appropriate elements of the Choleski factors or the solution vector $\underline{t}$.

11-4. The Levenberg-Marquardt-Nash Approach

Of the various approaches discussed in the previous section, we have found the Nash (1977) variant of Marquardt's (1963) method to be highly efficient over a wide class of nonlinear least squares problems. This approach augments the diagonal elements of the Gauss-Newton coefficient matrix (11-3-1) with both an absolute and a relative size increment. This has merit in a limited precision environment such as Microsoft BASIC single precision.

The overall strengths of the Marquardt method are its general reliability in solving nonlinear least squares problems. Even presented with very poorly scaled functions, it will proceed steadily, if slowly, towards lower sums of squares loss function values. For reasonably scaled problems with good initial estimates of the parameters, it is surprisingly efficient.

The major weaknesses of the Marquardt approach are

- that the user must provide correct program code for the residuals and their first derivatives with respect to the parameters
- that the working storage for the method is at least one n by n array, and may be higher depending on implementation details;
- that small implementation details can be quite important to the success or efficiency of the method.

The particular algorithm we have chosen to include here may not be the most ideally suited for all applications. Our choices, and the justification for them, are as follows.

1. The Levenberg-Marquardt Equations (11-3-11) are formed explicitly. In the best of circumstances, any such accumulation of inner products is best carried out in double-length arithmetic (Golub and Van Loan, 1983, page 37), which not all BASICs provide. Furthermore, Equations (11-3-11) are in essence <u>normal equations</u> for a linear least squares problem. The formation of explicit normal equations can be a source of unnecessary numerical error (Nash, 1979, Chapter 5). Matrix transformation methods

involving either QR decomposition or the singular value decomposition could be applied directly to the linear least squares problem

(11-4-1) $A \, \underline{t} \, \underline{\sim} - \underline{r}'$

where A is the matrix formed by appending the square root of the (diagonal) matrix lambda*Q to the bottom of the Jacobian J, and r' is the residual vector r augmented by n zeros, that is,

(11-4-2) $A = \begin{pmatrix} J \\ \\ E \end{pmatrix} \qquad r' = \begin{pmatrix} r \\ \\ 0 \end{pmatrix}$

where

(11-4-3) $E^2 = \text{lambda} * Q$

We have considered this approach but decided not to use it here because

- in advance we prefer to avoid having to declare working storage dependent on M, the number of observations (order of vector r) as well as the number of parameters N;
- in practice, such methods have appeared slow under the BASIC interpreters at our disposal;
- we have over 10 years' experience with essentially the code included below as applied to unconstrained problems.

We intend to continue to seek more efficient and reliable least squares problems, but at the time of writing have found explicit formation of the normal equations and their solution via the Choleski decomposition to be as efficient as any we have tested. This approach has also not demonstrated any particular weaknesses relative to the use of matrix decomposition methods. There is ample evidence that the solution of the Marquardt Equations (11-3-11) or (11-4-1) would be accomplished via a matrix decomposition with predictable and smaller error bounds on the solution t. However, the diagonal scaling lambda*Q allows us to avoid a singular set of linear equations. Furthermore, t is used as a search direction and is not of itself the result we seek.

2. The strategy we have chosen for controlling the Levenberg-Marquardt parameter, lambda, is as follows:

- before each new iteration, lambda is replaced by 0.4 * lambda
- if the matrix ($J^T J$ + lambda * Q) proves to be computationally singular in that a pivot element is zero during the Choleski decomposition, or if the sum of squares of the residual functions at the new point (b + t) is greater than that at b, then lambda is increased by the assignment

(11-4-4) lambda := 10 * lambda + 1/B9

where B9 is a "large number" determined for the computing environment by subroutine ENVRON (see Appendix E). The term 1/B9 is included in case lambda has been set to zero after many reductions.

3. The initial value of lambda (variable S9) used is 0.0001, so that a value of 0.00004 enters the first iteration. If it is suspected that the problem is "easy" or if all the parameters are linear, then setting an initial value of zero will usually improve convergence. (For linear parameters, this is assured in one iteration.) This change, which converts the iteration to a Gauss-Newton step, may be made by the assignment of variable S9 to zero in the code MRT.BAS in Section 11-5.

4. Convergence is judged on the basis of essentially no

difference between the parameter vectors (b + t) and b. Comparisons are made relative to a scalar constant E5, for which we use the value 10.

5. The matrix Q in Equations (11-3-11) is chosen to be a diagonal matrix whose elements are the diagonal elements of $J^T J$ plus lambda times a scaled unit matrix. Thus, in MRT.BAS the Gauss-Newton coefficient matrix is augmented in the j,j position with

(11-4-5) $(J^T J)_{jj}$ + lambda * phi

where in this work we set phi = A8 = 1 . For problems where the solution has a large residual sum of squares, we have on occasion found that faster convergence may be obtained with a much larger phi, which forces the search direction towards the gradient, or steepest descents, direction.

11-5. Marquardt Source-code

In order to maintain similarity to the other function minimization codes we have already presented, the following code uses the counters I9 and I8 to record the number of times the loss function (sum of squares) and its gradient (the right hand side of the Gauss-Newton equations) are evaluated. However, to compute the sum of squares, M residuals must be calculated by the code at line 3500. Likewise, there are M rows of the Jacobian matrix (each computed by code at line 4000).

Listing 11-5-1. The Marquardt Code.

```
              MRT.BAS           06-24-1986   21:26:05

40 DIM A(325),C(325),T(25),G(25),D(25)
1000 PRINT "MRT -- MARQUARDT/NASH -- 860412"
1004 PRINT #3,"MRT -- MARQUARDT/NASH -- 860412"
1008 REM CALLS:
```

```
1012 REM     RESIDUAL R1    -- line 3500
1016 REM     JACOBIAN D(B)  -- line 4000
1020 REM     ENVRON (computing environment) -- line 7120
1024 REM
1028 REM INPUTS TO THE ROUTINE:
1032 REM    B() -- a vector of initial parameter estimates
1036 REM    N   -- the number of parameters in the sum of squares
1040 REM    M   -- the number of data points
1044 REM    O( , ) -- masks and bounds
1048 REM    I5  -- the number of masked parameters. Must be zero if
1052 REM           masks are not used.
1056 REM OUTPUT FROM THE ROUTINE:
1060 REM    X() -- a vector of final parameter estimates for the
1064 REM           parameter values which minimize the sum of squares
1068 REM    F0  -- value of the sum of squares at the minimum
1072 REM    I8  -- number of Jacobian evaluations
1076 REM    I9  -- number of sum of squares evaluations
1080 REM
1084 GOSUB 7120: REM machine environment
1088 LET J8=1: REM no of fn eval before parameter display
1092 LET J7=1: REM counter for parameter display
1096 LET A7=10: REM lambda increase factor
1100 LET A9=4: REM lambda decrease factor = A9/A7
1104 LET A8=1: REM Nash phi factor
1108 LET S9=.0001: REM initial lambda
1112 LET N2=N*(N+1)/2
1116 REM DIM A(N2),C(N2),T(N),G(N),D(N)
1120 IF M>N THEN 1132
1124 PRINT "WARNING: NO. OF DATA POINTS <= NO. OF VARIABLES."
1128 PRINT #3,"WARNING: NO. OF DATA POINTS <= NO. OF VARIABLES."
1132 LET I3=0: REM non-computability flag
1136 GOSUB 1848: REM compute sum of squares
1140 IF I3=0 THEN 1156
1144 PRINT "ERROR:  FUNCTION NOT COMPUTABLE"
1148 PRINT #3,"ERROR:  FUNCTION NOT COMPUTABLE"
1152 STOP
1156 LET S9=A9*S9/A7: REM decrease lambda -- top of cycle
1160 LET F0=F: REM save lowest value of sum of squares
1164 PRINT "ITN";I8;"  EVAL'N";I9;" SS=";F0
1168 PRINT #3,"ITN";I8;"  EVAL'N";I9;" SS=";F0
1172 GOSUB 1884: REM display parameters
1176 FOR I=1 TO N2: REM form JT * J (sscp matrix) and JT * r (rhs)
1180 LET A(I)=0: REM initialize sscp matrix
1184 NEXT I
1188 FOR I=1 TO N
1192 LET G(I)=0: REM initialize gradient (JT * r)
1196 NEXT I
1200 FOR I=1 TO M: REM loop over residuals
1204 GOSUB 3500: REM & compute I'th residual in R1
1208 REM this call not necessary if residuals have been saved
```

```
1212 GOSUB 4000: REM compute I'th row of Jacobian in D()
1216 FOR J=1 TO N
1220 LET G(J)=G(J)+D(J)*R1: REM accumulate gradient (better in dbl)
1224 LET K2=J*(J-1)/2
1228 FOR K=1 TO J
1232 LET A(K2+K)=A(K2+K)+D(J)*D(K): REM accumulate sscp (better in dbl)
1236 NEXT K
1240 NEXT J
1244 NEXT I: REM end sum of squares and cross-products formation
1248 LET I8=I8+1: REM count evaluation of Hessian
1252 FOR J=1 TO N2
1256 LET C(J)=A(J): REM save sscp matrix
1260 NEXT J
1264 FOR J=1 TO N
1268 LET X(J)=B(J): REM save current best parameters
1272 NEXT J
1276 FOR J=1 TO N: REM adjust gradient for bounds constraints
1280 IF O(J,3)>=0 THEN 1308: REM masked or free
1284 IF (O(J,3)+1.5)*G(J)<=0 THEN 1304: REM Lagrange multiplier
1288 LET O(J,3)=1: REM free parameter
1292 PRINT "freeing parameter ";J
1296 PRINT #3,"freeing parameter ";J
1300 GOTO 1308
1304 LET G(J)=0: REM constraint on parameter j active
1308 NEXT J: REM bounds constraints on G imposed
1312 FOR J=1 TO N: REM augment sscp matrix on diagonal
1316 LET K2=J*(J+1)/2
1320 LET A(K2)=C(K2)*(1+S9)+A8*S9
1324 LET T(J)=-G(J): REM right hand side of equations
1328 IF J=1 THEN 1344
1332 FOR I=1 TO J-1
1336 LET A(K2-I)=C(K2-I): REM off-diagonal elements
1340 NEXT I
1344 NEXT J
1348 PRINT "LAMDA = ";S9
1352 PRINT #3,"LAMDA = ";S9
1356 LET I3=0: REM use non-computability flag for Choleski decomposition
1360 GOSUB 1596: REM Choleski decomposition
1364 IF I3=0 THEN 1380
1368 PRINT "SSCP MATRIX NOT COMPUTATIONALLY POSITIVE DEFINITE"
1372 PRINT #3,"SSCP MATRIX NOT COMPUTATIONALLY POSITIVE DEFINITE"
1376 GOTO 1576: REM so we increase lambda and try again
1380 GOSUB 1696: REM Choleski back-substitution
1384 LET G8=0: REM project solution (before step)
1388 FOR J=1 TO N
1392 IF O(J,3)=1 THEN 1400: REM free parameter
1396 LET T(J)=0: REM bound constraint or mask active
1400 LET G8=G8+G(J)*T(J): REM gradient times step
1404 NEXT J
1408 IF G8>0 THEN 1576: REM uphill direction
```

```
1412 LET S1=1: REM initial step size is always 1.0
1416 LET S8=S1: REM find max-step and check new constraint
1420 FOR J=1 TO N
1424 IF T(J)=0 THEN 1460: REM no sense in checking
1428 REM T(J)=0 when mask or bound is active
1432 IF T(J)>0 THEN 1448: REM check upper bound
1436 LET S7=(O(J,1)-X(J))/T(J): REM distance to lower bd.
1440 IF S7>S8 THEN 1460: REM smaller step in effect
1444 GOTO 1456
1448 LET S7=(O(J,2)-X(J))/T(J): REM distance to upper bd.
1452 IF S8<S7 THEN 1460: REM step smaller than limit (J7 may = 0 if <=)
1456 LET S8=S7
1460 NEXT J
1464 IF S8<S1 THEN PRINT "stepsize reduced to ";S8;" by bounds"
1468 IF S8<S1 THEN PRINT #3,"stepsize reduced to ";S8;" by bounds"
1472 LET I6=0: REM step along search direction
1476 FOR J=1 TO N: REM loop over parameters
1480 IF O(J,3)=0 THEN 1496: REM masked parameter?
1484 LET B(J)=X(J)+S8*T(J): REM use restricted stepsize
1488 IF B(J)+E5<>X(J)+E5 THEN 1496: REM test of difference
1490 LET B(J)=X(J): REM reset to avoid "drift"
1492 LET I6=I6+1
1496 NEXT J
1500 IF I6=N-I5 THEN 1592: REM convergence test -- note masks
1504 GOSUB 1848: REM compute sum of squares
1508 IF I3=0 THEN 1524
1512 PRINT "SUM OF SQUARES NOT COMPUTABLE"
1516 PRINT #3,"SUM OF SQUARES NOT COMPUTABLE"
1520 GOTO 1576: REM so increase lambda and try again
1524 IF F>=F0 THEN 1576
1528 FOR J=1 TO N: REM check bounds on new parameters
1532 IF O(J,3)<=0 THEN 1568: REM masked or constrained
1536 IF B(J)-O(J,1)>E9*(ABS(O(J,1))+E9) THEN 1552
1540 PRINT "lower bound on parameter ";J;" active"
1544 PRINT #3,"lower bound on parameter ";J;" active"
1548 LET O(J,3)=-2
1552 IF O(J,2)-B(J)>E9*(ABS(O(J,2))+E9) THEN 1568
1556 PRINT "upper bound on parameter ";J;" active"
1560 PRINT #3,"upper bound on parameter ";J;" active"
1564 LET O(J,3)=-1
1568 NEXT J
1572 GOTO 1156: REM try another cycle
1576 LET S9=S9*A7: REM increase lambda
1580 IF S9>0 THEN 1312: REM safety check to ensure positive lambda
1584 LET S9=E9: REM set lambda to machine precision (> 0)
1588 GOTO 1312: REM augment sscp matrix and repeat solution
1592 RETURN: REM converged -- finish of MRT
1596 FOR J=1 TO N: REM Choleski decomposition (Alg. 7 Nash, 1979)
1600 LET K2=J*(J+1)/2
1604 IF J=1 THEN 1640
```

```
1608 FOR I=J TO N
1612 LET K1=I*(I-1)/2+J
1616 LET S=A(K1)
1620 FOR K=1 TO J-1
1624 LET S=S-A(K1-K)*A(K2-K)
1628 NEXT K
1632 LET A(K1)=S
1636 NEXT I
1640 IF O(J,3)<=0 THEN 1656: REM mask or active bound
1644 IF A(K2)>0 THEN 1668
1648 LET I3=1: REM not computationally positive definite
1652 GOTO 1692: REM return with flag set
1656 LET S=0: REM constraint active -- null out row
1660 LET A(K2)=0
1664 GOTO 1672
1668 LET S=1/SQR(A(K2))
1672 FOR I=J TO N
1676 LET K1=I*(I-1)/2+J
1680 LET A(K1)=A(K1)*S
1684 NEXT I
1688 NEXT J
1692 RETURN: REM end of Choleski decomposition
1696 IF O(1,3)<=0 THEN 1708: REM mask or active bound
1700 LET S=1/A(1)
1704 GOTO 1712
1708 LET S=0
1712 LET T(1)=T(1)*S: REM Cholback   (Nash, 1979, Alg. 8)
1716 REM Choleski back-substitution -- L * G = X
1720 IF N=1 THEN 1776: REM 1 by 1 matrix
1724 LET K2=1
1728 FOR I=2 TO N
1732 FOR J=1 TO I-1
1736 LET K2=K2+1
1740 LET T(I)=T(I)-A(K2)*T(J)
1744 NEXT J
1748 LET K2=K2+1
1752 IF O(I,3)<=0 THEN 1764: REM mask or active bound
1756 LET S=1/A(K2)
1760 GOTO 1768
1764 LET S=0
1768 LET T(I)=T(I)*S
1772 NEXT I
1776 IF O(N,3)<=0 THEN 1788: REM mask or active bound
1780 LET S=1/A(N2)
1784 GOTO 1792
1788 LET S=0: REM active constraint
1792 LET T(N)=T(N)*S: REM begin LT * X = G solution
1796 IF N=1 THEN 1844
1800 FOR I=N TO 2 STEP -1
1804 LET K2=I*(I-1)/2
```

```
1808 FOR J=1 TO I-1
1812 LET T(J)=T(J)-T(I)*A(K2+J)
1816 NEXT J
1820 IF O(I-1,3)<=0 THEN 1832
1824 LET S=1/A(K2)
1828 GOTO 1836
1832 LET S=0
1836 LET T(I-1)=T(I-1)*S
1840 NEXT I
1844 RETURN: REM end Choleski back-substitution
1848 LET F=0: REM sum of squares calculation
1852 LET I3=0
1856 LET I9=I9+1
1860 FOR I=1 TO M
1864 GOSUB 3500
1868 IF I3=1 THEN 1880: REM is function computable?
1872 LET F=F+R1*R1
1876 NEXT I: REM PRINT "after sumsquares calcn f=";F;" i3=";I3
1880 RETURN
1884 IF J8=0 THEN RETURN: REM no parameter display
1888 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1892 LET J7=INT(I9/J8)+1: REM increment parameter display control
1896 PRINT "parameters";
1900 PRINT #3,"parameters";
1904 FOR J=1 TO N
1908 LET Q$=""
1912 IF B(J)-O(J,1)<=E9*(ABS(O(J,1))+E9) THEN LET Q$="L": REM lower bd
1916 IF O(J,2)-B(J)<=E9*(ABS(O(J,2))+E9) THEN LET Q$="U": REM upper bd
1920 IF O(J,3)=0 THEN LET Q$="M": REM masked
1924 PRINT "  ";B(J);Q$;
1928 PRINT #3,"  ";B(J);Q$;
1932 IF 5*INT(J/5)<>J THEN 1952
1936 PRINT
1940 PRINT "          ";
1944 PRINT #3,
1948 PRINT #3,"          ";
1952 NEXT J
1956 PRINT
1960 PRINT #3,
1964 RETURN
```

```
     Line no.    Referenced in line(s)
      1132      1120
      1156      1140  1572
      1304      1284
      1308      1280  1300
      1312      1580  1588
      1344      1328
      1380      1364
      1400      1392
```

```
      1448      1432
      1456      1444
      1460      1424  1440  1452
      1496      1480  1488
      1524      1508
      1552      1536
      1568      1532  1552
      1576      1376  1408  1520  1524
      1592      1500
      1596      1360
      1640      1604
      1656      1640
      1668      1644
      1672      1664
      1692      1652
      1696      1380
      1708      1696
      1712      1704
      1764      1752
      1768      1760
      1776      1720
      1788      1776
      1792      1784
      1832      1820
      1836      1828
      1844      1796
      1848      1136  1504
      1880      1868
      1884      1172
      1952      1932
      3500      1204  1864 -- subroutine to calculate the residual
      4000      1212 -- subroutine to calculate the Jacobian
      7120      1084 -- computing environment subroutine

 Symbol     Referenced in line(s)
A(            40  1180  1232  1256  1320  1336  1616  1624  1632
            1644  1660  1668  1680  1700  1740  1756  1780  1812
            1824 -- sscp matrix
A7          1096  1156  1576 -- lambda increase factor
A8          1104  1320 -- Nash phi factor
A9          1100  1156 -- lambda decrease factor (A9/A7)
B(          1268  1484  1488  1490  1536  1552  1912  1916  1924
            1928 -- the parameter vector
C(            40  1256  1320  1336 -- storage of sscp matrix
D(            40  1220  1232 -- I-th row of Jacobian
E5          1488 -- a value used for scaled comparisons (= 10)
E9          1536  1552  1584  1912  1916 -- the machine precision
F           1160  1524  1848  1872 -- the loss function value
F0          1160  1164  1168  1524 -- the lowest value found
            for the loss function
```

```
G(          40  1192  1220  1284  1304  1324  1400 -- the gradient
G8        1384  1400  1408 -- projection of solution
I         1176  1180  1184  1188  1192  1196  1200  1244  1332
          1336  1340  1608  1612  1636  1672  1676  1684  1728
          1732  1740  1752  1768  1772  1800  1804  1808  1812
          1820  1836  1840  1860  1876 -- a loop control counter
I3        1132  1140  1356  1364  1508  1648  1852  1868 -- flag
          for function not computable
I5        1500 -- the number of masked parameters
I6        1472  1492  1500 -- a counter for the number of
          parameters which are unchanged in a step
I8        1164  1168  1248 -- counter for gradient evaluations
          performed
I9        1164  1168  1856  1888  1892 -- counter for function
          evaluations performed
J         1216  1220  1224  1228  1232  1240  1252  1256  1260
          1264  1268  1272  1276  1280  1284  1288  1292  1296
          1304  1308  1312  1316  1324  1328  1332  1344  1388
          1392  1396  1400  1404  1420  1424  1432  1436  1448
          1460  1476  1480  1484  1488  1490  1496  1528  1532
          1536  1540  1544  1548  1552  1556  1560  1564  1568
          1596  1600  1604  1608  1612  1620  1640  1672  1676
          1688  1732  1740  1744  1808  1812  1816  1904  1912
          1916  1920  1924  1928  1932  1952 -- a loop control
          counter
J7        1092  1888  1892 -- parameter display control index
J8        1088  1884  1888  1892 -- parameters are displayed
          every J8 function evaluations (no display if J8=0)
K         1228  1232  1236  1620  1624  1628 -- a loop control
          counter
K1        1612  1616  1624  1632  1676  1680 -- triangular array
          element index for Choleski decomposition
K2        1224  1232  1316  1320  1336  1600  1624  1644  1660
          1668  1724  1736  1740  1748  1756  1804  1812  1824
          -- triangular array element index
M         1120  1200  1860 -- the number of data points
N         1112  1120  1188  1216  1264  1276  1312  1388  1420
          1476  1500  1528  1596  1608  1672  1720  1728  1776
          1792  1796  1800  1904 -- number of parameters in loss
          function
N2      1112  1176  1252  1780 -- number of elements in triangular
          array = N*(N+1)/2
O(        1280  1284  1288  1392  1436  1448  1480  1532  1536
          1548  1552  1564  1640  1696  1752  1776  1820  1912
          1916  1920 -- bounds and masks information storage
Q$        1908  1912  1916  1920  1924  1928 -- used to hold
          display information regarding masks and active bounds
R1        1220  1872 -- the residual
S         1616  1624  1632  1656  1668  1680  1700  1708  1712
          1756  1764  1768  1780  1788  1792  1824  1832  1836
          -- temporary accumulator / factor in Choleski algorithm
S1        1412  1416  1464  1468 -- initial step size
```

```
S7        1436  1440  1448  1452  1456 -- distance to bound
S8        1416  1440  1452  1456  1464  1468  1484 -- maximum
          stepsize allowed by bounds
S9       1108  1156  1320  1348  1352  1576  1580  1584 -- lambda
T(          40  1324  1396  1400  1424  1432  1436  1448  1484
         1712  1740  1768  1792  1812  1836 -- the search direction
X(        1268  1436  1448  1484  1488  1490 -- storage for the
          "best" parameters found so far
=================================================================
LINES: 245      BYTES: 8689     SYMBOLS: 79     REFERENCES: 407
```

11-6. Use of the Marquardt Method

As with the other five methods, we first demonstrate MRT using the Hobbs problem. Listing 11-6-1 presents edited output.

As another example, consider a problem which arose in modeling the population of soft-shelled clams (Mya arenasia) on the Connecticut shoreline. This data was collected by Dr. Diane J. Brousseau and analyzed by Dr. Jenny A. Baglivo, both of Fairfield University. Determining the age of the clams is important in any study of the population, yet this is a time-consuming activity, so that it is desired to estimate age of the clams by simply measuring their length and applying a formula or table. Age (in months) and length (in mm) were recorded for different sites. Here we will consider just one site (near Westport CT), where data for 585 clams were gathered. The model proposed is a form of the Von Bertalanffy growth function (Draper and Smith, 1981, page 512ff.), which we will write in the form of a modeling function

(11-6-1) Z($\underline{B}$) = B(1) * (1 - exp(-B(2) * (Y(i,2) - B(3))))

We will scale the parameters in the problem code (BAGLIVO.RES). The functional form is an exponential approach to an asymtotic value B(2). Plots of the data and the final fitted function(s) are instructive, and have been carried out by Brousseau, Baglivo, and their collaborator, Dr. George Lang, whose work we anticipate will be published in the near future.

Here our use of this problem will be to illustrate the use of MRT in fitting the model. First, we select a subset of the 585 data points. Arbitrarily we select the first 41 points, and this particular data set is included on the diskette as BAGLIVO.DTA on the diskette. Dr. Baglivo suggested B(3) = -0.5, so we masked B(3) at this value in our first trial run, using B(1) = B(2) = 1 as initial values for the unmasked parameters. The output parameters were B(1) = 5.500696 and B(2) = 7.47362.

Turning to the full set of data, we first compiled the program to shorten elapsed time for runs, then started MRT at an initial point B = (5.5, 7.5, -0.5), where the residual sum of squares is 72199.23, 201 seconds, 7 gradient and 17 function evaluations were required to obtain a sum of squares of 71831.29 at (5.625306, 7.187534, -0.5). Freeing B(3), but limiting its lower bound to -2 allowed the sum of squares to be reduced to 67080.95 after a further 223 seconds, 8 gradient and 18 function evaluations. The parameter estimates were then B = (5.647857, 6.702821, -2). Later, from approximately this starting point, and with the lower bound on B(3) relaxed to -100, we obtained a residual sum of squares of only 44882.54 at B = (10.38152, 0.4719552, -100). A further relaxation of the bound to -1000 finally allowed an unbounded minimum to be found of 44871.7 at B = (12.61953, 0.3265697, -114.9181). Unfortunately, residual plots show a pattern in the residuals when plotted against age, and particularly when plotted against length, suggesting more work needs to be done, either in removing outliers from the data or in altering the model. Such analyses, which are

important to the overall success of nonlinear modeling efforts, are beyond the needs of our present discussion.

Listing 11-6-1. Minimization of Hobbs problem using the Marquardt method.

```
DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
12:20:43      07-09-1986
HOBBS 3-PARAMETER LOGISTIC FUNCTION SETUP - 851017
 FUNCTION FORM = 100*B(1)/(1+10*B(2)*EXP(-0.1*B(3)*I))

HOBBS.RES 3 parameter logistic fit
bounds on parameters for problem
 0  <=   b( 1 )  <=   100
 0  <=   b( 2 )  <=   100
 0  <=   b( 3 )  <=   30
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) y
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
 are masks or bounds to be set or altered ([cr] = no)
MRT -- MARQUARDT/NASH -- 860412
ITN 0   EVAL'N 1  SS= 158.2325
parameters   2    5    3
LAMDA =  .00004
ITN 1   EVAL'N 2  SS= 2.63402
parameters   1.934047    4.844973    3.144204
LAMDA =  .000016
ITN 2   EVAL'N 3  SS= 2.587501
parameters   1.960944    4.907519    3.135944
LAMDA =  .0000064
ITN 3   EVAL'N 4  SS= 2.587268
parameters   1.961851    4.90915    3.135703
LAMDA =  2.56E-06
LAMDA =  .0000256
LAMDA =  .000256
LAMDA =  .00256
LAMDA =  .0256
LAMDA =  .256
  ELAPSED SECS= 34  AFTER 4  GRAD & 9  FN EVAL
CALCULATED FUNCTION MINIMUM =  2.587268
PARAMETER ESTIMATES
B( 1 ) =  1.961851
B( 2 ) =  4.90915
B( 3 ) =  3.135703
```

11-7. Other Approaches

We have already mentioned that other possibilities exist for implementing a nonlinear least squares code. As an illustration of another approach, we will compare the structure and performance of MRT with that of LEQB05 (Nash, 1984b; Nash, 1985).

In that suite of BASIC program code, the nonlinear least squares problem is but one task among several solved using the singular value decomposition (svd) as a tool. The approach is to solve the linear least squares problem

(11-7-1) $J \underline{t} = - \underline{r}$

using the decomposition

(11-7-2) $J = U S V^T$

where S is a diagonal matrix of non-negative elements and where the M by N matrix U and N by N matrix V have the orthogonality properties

(11-7-3) $U^T U = V^T V = V V^T = 1_N$

which is the identity of order N. The svd allows the solution of (11-7-1) to be formed as

(11-7-4) $\underline{t} = - J^+ r = - V S^+ U^T \underline{r}$

where

(11-7-5) $$S^T_{ii} = 1/S_{ii} \text{ if } S_{ii} > 0$$
$$= 0 \quad \text{if } S_{ii} = 0$$

The matrix J^+ is a generalized inverse of the Jacobian. The particular form of generalized inverse generated in (11-7-4) is the Moore-Penrose inverse (Golub and Van Loan, 1983, page 139), which yields the unique minimum length least squares solution (Dahlquist and Bjorck, 1974, page 203) of (11-7-1). With the search direction $\underline{t}$, a search is made for a new point, such that the sum of squares loss function

(11-7-6) $f(\underline{b} + k^j \underline{t}) < f(\underline{b})$

for j = 0, 1, 2,... and k is some positive number smaller than 1. If the search fails, then the algorithm is halted and $\underline{b}$ is taken as the solution. If a new point with a smaller sum of squares is found, then it becomes a new $\underline{b}$ and the process is repeated. Listing 11-7-1 shows the application of this method to the Hobbs problem.

Listing 11-7-1. Application of the LEQB05 code (Nash, 1984b) to the Hobbs problem.

```
HOBBS FUNCTION - 3 PARAM LOGISTIC
STARTING PARAMETERS
B( 1 )= 2
B( 2 )= 5
B( 3 )= 3
12:37:49
AT ITN 0   SS= 158.2325  AFTER  1
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 3  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.642364
12:37:58
AT ITN 1   SS= 2.642364  AFTER  2
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587704
12:38:06
AT ITN 2   SS= 2.587704  AFTER  3
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587279
12:38:15
AT ITN 3   SS= 2.587279  AFTER  4
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587271
12:38:23
AT ITN 4   SS= 2.587271  AFTER  5
```

```
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587268
12:38:31
AT ITN 5   SS= 2.587268  AFTER  6
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587265
12:38:39
AT ITN 6   SS= 2.587265  AFTER  7
END SWEEP  1 , 3  ROTNS PERFORMED
END SWEEP  2 , 2  ROTNS PERFORMED
END SWEEP  3 , 0  ROTNS PERFORMED
 3  PRINCIPAL COMPONENTS USED. 0  IGNORED
TRIAL SS= 2.587267
TRIAL SS= 2.587265
CONVERGED TO SS= 2.587265  IN  9
12:38:49       <----- elapsed time = 60 seconds
B( 1 )= 1.961861
B( 2 )= 4.909163
B( 3 )= 3.135698
```

In general this approach has the following advantages:

- the singular values inform us of the condition of the Jacobian at each iteration;
- the decomposition allows us to compute various measures of convergence or nonlinearity (see Chapter 13);
- by means of the mechanism indicated in Equation (11-4-2), a Levenberg-Marquardt correction can be applied, if desired.

The disadvantages are:

- each iteration takes several times as much computational effort in finding the search direction t as the Choleski decomposition variant of the Marquardt method;
- because the diagonal elements and the sum of squares and cross-products matrix

$$J^T \ J$$

require column norms to be computed, the scaled Levenberg-Marquardt correction requires extra effort to apply and will probably not be used;
- the entire matrix J may be stored in some implementations of the svd, requiring a relatively large working store for problems having a large number of data points (observations).

In preparing this book, we were very tempted to change to the method discussed here. However, our long experience with MRT led us to keep the more established method for the present collection of BASIC code.

11-8. Numerical Approximation of the Jacobian

Using the same principles as used in Section 7-5, where numerical approximations to the gradient were developed using forward differences, the following program code segment (NUMJAC.BAS, in Listing 11-8-1) can replace the code which the user normally supplies in lines 4000-4499 to compute the I-th row of the Jacobian in vector D(). While this saves human effort, it generally increases the computational time by a factor of 4 or 5 for typical problems. Furthermore, in low precision computing environments, false convergence may be observed. Once again, the function evaluation counter I9 is not altered in this code.

Listing 11-8-1. The NUMJAC Code.

```
           NUMJAC.BAS          06-24-1986   21:27:14

    4000 REM NUMJAC--NUMERICAL CALCULATION OF PARTIAL DERIV
    4001 REM CALLS:
    4002 REM    RESIDUAL R1 -- line 3500
    4003 REM
    4004 REM INPUTS:
    4005 REM    E5   -- a number for relative equality tests
    4006 REM    E9   -- the machine precision
    4007 REM    B()  -- the parameter values
    4008 REM    R1   -- the I'th residual for the given B()
    4009 REM    N    -- the number of parameters
    4010 REM    O(,) -- the parameter bounds and masks (see Sect. 3-7)
    4011 REM
    4012 REM OUTPUTS:
    4013 REM    D()  -- the numerical approximations to the Jacobian
    4014 REM
    4015 LET E4=E5*SQR(E9): REM step size for derivatives
    4018 LET R0=R1: REM store original res
    4020 FOR J9=1 TO N: REM loop over B
    4025 IF O(J9,3)=0 THEN 4080: REM check for masks
    4030 LET D0=B(J9): REM save parameter (don't use X() - see CG)
    4035 LET E3=E4*(ABS(D0)+E4)
    4040 LET B(J9)=B(J9)+E3
    4050 GOSUB 3500: REM res calculation
    4060 LET D(J9)=(R1-R0)/E3
    4070 LET B(J9)=D0: REM reset the parameter
    4080 NEXT J9
    4090 LET R1=R0: REM restore original fn
    4100 RETURN

    Line no.    Referenced in line(s)
     3500      4050 -- subroutine to calculate the residuals
     4080      4025

     Symbol     Referenced in line(s)
    B(         4030  4040  4070 -- the parameter vector
    D(         4060 -- the numerical approximations to the Jacobian
    D0         4030  4035  4070 -- temporary storage of the parameter
    E3         4035  4040  4060 -- the delta
    E4         4015  4035 -- step size for derivatives
    E5         4015 -- a value used for scaled comparisons (= 10)
    E9         4015 -- the machine precision
    J9         4020  4025  4030  4040  4060  4070  4080 -- a loop control
               counter
    N          4020 -- number of parameters in loss function
    O(         4025 -- bounds and masks information storage
    R0         4018  4060  4090 -- original residual
```

```
    R1         4018  4060  4090 -- residual delta away from original
               point
=====================================================================
    LINES: 29      BYTES: 986     SYMBOLS: 14     REFERENCES: 31
```

COVER SHEET

Chapter 12

Chapter title: Choosing an Approach and Method

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

12 CHOOSING AN APPROACH AND METHOD

12-0. The Choices Facing the User

In this chapter we attempt to provide some guidance on the strategic and tactical choices which are used in solving a nonlinear parameter estimation problem. In particular, we will point out which of the methods we have presented, or combination thereof, are likely to be most suitable for solving a given type of problem. We will also suggest ways to speed up the human cost of performing this work.

The most obvious first choice facing the user of our work is which of the six nonlinear estimation programs to use. For a "new" problem, that is, one with which there is little or no user experience, or for which the data is suspected to result in unusual or poorly-defined parameter values, our strategy is to first decide the estimation criterion. Having expressed this criterion as an objective function, a skeleton of the program code to establish the data (the "setup" Section 15-3) and to compute the loss function or residuals (also Section 15-3) is written. However, at this stage, unless the problem is well-defined and understood, we are unlikely to write program code for derivatives. Nor are we likely to impose constraints, except perhaps by indicating that a constraint is violated by setting a flag variable. (In our codes, this flag variable is usually given label I3.)

At this stage, it is possible to test the loss function code. Since analytic expressions for derivative calculations are not available, we do NOT use FGTEST, but invoke either the Hooke and Jeeves (HJ) or Nelder-Mead (NM) codes in a preliminary exploration of the problem. Generally our own preference is for HJ at this stage, since it is easy to visualize the progress, or lack thereof, of this algorithm in exploring the loss function surface. Typically, simple errors in programming the loss function are discovered and corrected in this exercise, for which we recommend a BASIC _interpreter_ for user convenience, or at the very least an incremental compiler which allows the user to avoid long delays during compilation and linkage of programs.

Note that any corrections made must be SAVED in the problem file. The simplest way to do this is by means of the DELETE instruction in most BASICs. That is, the user must delete all of the consolidated program code EXCEPT the DIMensions in lines 30-39 and problem code in lines 2000-4499. In our own work, we have found it rather easy to forget the DIMension statements while performing the deletions. In Microsoft BASIC, and the generally equivalent BASICA of the IBM PC, it is important to SAVE the problem code as ASCII characters rather than in tokenized form. This is accomplished by entering the command

```
SAVE "problem", A
```

In many cases, HJ or NM will find a satisfactory solution point, that is, one for which

1. the loss function is acceptably small;
2. the parameters satisfy any constraints which exist, whether or not they have been explicitly included in our codes;
3. the parameters satisfy user expectations of their likely or appropriate values, or can otherwise be supported in the context of the application.

What has been described is a "minimal effort" strategy. The user only prepares enough program code to try out the estimation problem, and verifies whether constraints are violated after a candidate solution has been developed.

In a case where this succeeds, it is relatively easy to obtain parameter dispersion estimates by invoking the Marquardt code (MRT) for nonlinear least squares problems, or the variable metric method (VM) for more general loss functions. Numerical approximation of gradient or Jacobian information will usually suffice, so that the user need not prepare more program code. Alternatively, the general post-solution analysis program POSTGEN (Section 13-6) may provide sufficient information about the parameter variability.

When a given problem is a member of a family of related estimation tasks, it is usually worthwhile developing program code to compute derivatives from analytic expressions. Furthermore, if many problems are to be solved, it is useful to consider such possibilities as:

1. data entry and checking programs;
2. specialized code to provide initial estimates of the parameters;
3. specialized results-analysis code to verify and report estimates in a manner of immediate importance to end users.

12-1. Indicators for a Given Method

From Section 12-0, it may seem that we will use HJ or NM to solve most problems. In practice, these codes do get quite heavy usage in our own work, with MRT as a follow-up, since a large proportion of problems are formulated using the least squares loss function.

For a great many problems, the variable metric (VM) method is a reliable and efficient work-horse. Its particular virtues shine on problems with a modest number of parameters (<10), since the arrays which are used become quite demanding of both storage space and access time as the number of parameters becomes large. On highly nonlinear problems, the motivations for the method have dubious justification, and we may expect poor performance. Similarly, problems which have a singular Hessian should cause difficulty. Our strategy of always restarting the iteration with a steepest descent search whenever difficulties are encountered does, however, render our code reasonably reliable, though slow, in such situations.

Nevertheless, there are situations where conjugate gradient and truncated-Newton methods have a role. In particular, when there are a large number of parameters, CG and TN take a higher profile. Generally TN should be the more efficient method of the two. However, we do not suggest its use except when analytic derivative expressions are available since numerical approximation is already used in TN for <u>second</u> derivatives. Both CG and TN have theoretical limitations when the objective function to be minimized has a singular Hessian at the solution (see Section 14-3), but CG has fairly extensive heuristics which may allow it to make better progress towards the solution than TN in such cases.

(An alternative may be to switch back to HJ near the solution. To save effort, the parameters should be written to and read from a file for such exercises. Program code to carry out such file operations was considered for inclusion in this book, but is needed rarely enough that we decided to leave the addition of necessary statements to the user.)

Of the two direct search techniques, we generally prefer NM if seeking the minimum of a function, since it is usually -- but not always -- more efficient. HJ is preferred when information about the loss function surface is to be extracted, since it is easier to follow the axial search points. HJ also uses much less working storage for problems in a large number of parameters. It may "converge" more quickly than NM when started near a solution point, since the code does not compute the function if no change is made to a parameter during the axial search. Unfortunately, because all of our methods for function minimization operate in arithmetic which uses a discrete approximation to the continuous world, there will be many cases where a given method demonstrates startlingly "good" or "bad" performance.

For nonlinear least squares problems, MRT has proved extremely reliable and efficient. When a problem can be cast in nonlinear least squares form, we have found that MRT is generally more likely to find a solution and more likely to find it quickly than any other method. However, errors in the derivative calculations severely disrupt MRT. Before it is used, we recomment RJTEST be invoked to ensure derivatives are computed correctly. Similarly, FGTEST should be applied before CG, TN or VM (the gradient methods) are invoked, since these suffer a similar slow-down or failure if derivative information is incorrect.

While analytic derivative expressions are generally preferred over numerical approximations, there is a reasonably wide appreciation among workers in the field that methods using numerical approximation may take less steps to approach a minimum if the initial parameters supplied to begin the iterations are "far" from the minimum. Because of the extra work in the evaluation of the derivatives, it is a debatable point if less overall computational effort is required. Furthermore, the progress of methods in which numerical approximation is used may be sensitive to the precision of floating-point arithmetic used or to small errors in the internal routines used to compute special functions (sin, cos, exp, log, etc.). For the convenience of users, ourselves included, we have provided program code to allow numerical derivative approximations to be used with all the gradient minimization methods. NUMGRAD (Section 7-5) provides for gradient approximation for general loss functions, while NUMJAC (Section 11-8) is for use with least squares problems, that is, where rows of the Jacobian need to be approximated.

When the loss function used to estimate parameters is not a least squares form, and gradient information is available, the gradient methods VM, TN and CG are candidates. For small numbers of parameters, our experience is that VM is the most reliable and efficient of the three. However, TN has demonstrated remarkable performance on a number of problems, though it is somewhat more sensitive to the machine arithmetic and convergence tolerances. As indicated above, CG may have some advantages when the Hessian is singular at the solution point. Since the presence of a singular Hessian is unlikely to be obvious -- we probably do not know the solution point yet -- the user will have to rely on other indications, such as slow convergence. However, slow convergence is also indicative of incorrect derivative information or problems which are highly nonlinear. The former possibility can be detected using FGTEST (if analytic derivative code is present). Problems with extreme

nonlinearity will slow the progress of our methods unless the problem can be changed in some way. Even transforming or re-parametrizing the problem cannot remove intrinsic nonlinearity. Ratkowski (1983, Chapter 2) discusses the assessment of nonlineraity in regression models, that is, for a least squares loss function, using the ideas of Bates and Watts (1980). However, for more general loss functions, tools for discovering the degree of nonlinearity of a problem are yet to be developed.

Our own practice is to stop a program if progress is slow in order to switch to another method (after derivative code, if present, has been checked). While switching method is not guaranteed to give any more rapid approach to a solution, the change from a gradient method to one of the direct search methods may allow us to discover some structure in the problem so that the slow convergence may be better understood. Ultimately, more profound examination of the problem is required if such understanding is important. For example, it is possible to compute the actual Hessian matrix at a given point $\underline{B}$, and to compute its eigenvalues (for example, with LEQB05, Nash, 1984b, or some other program), so that singularity of the Hessian may be confirmed.

12-2. General Performance of Methods

In this section we present primarily tabular information detailing the program size, working storage requirements, and execution times on different problems for each of our methods. Unfortunately, it is not possible to simply state that one method is faster or better than another without considering the way(s) in which they have been implemented and programmed. In particular, our approach has been to include features in the programs which make the parameter estimation problem easier to solve, or which add to information about the problem at hand. However, these features add to the length of the program code and to the time required for its execution. Therefore, in this section, we include measures of length and complexity and of the execution time of stripped-down versions of our codes. These "small" programs have been created by

- removing code to copy console output to a file
- eliminating the calculation of dispersion measures for parameter estimates
- purging all REMark statements
- using a highly simplified computing environment subroutine ENVRON
- not allowing mask and bound constraints to be included
- removing the code to enter constraint information.

In the tables which follow, the stripped-down or "small" codes will be labelled with an S, while the programs with features, or which are "full" programs, will be labelled with an F. Tables 12-2-1 presents measures of length and complexity of the minimization subroutines and the overall programs they generate when applied to a test problem. In these measures, "lines" means number of lines of code. "Bytes" refers to the number of bytes of source code actually read by a cross-reference program, while "storage" is the figure listed by a directory as being taken up by the program code file. "Symbols" are any variables, arrays or line labels identified by the cross-reference program, while "references" are the count of the number of mentions of such items. For the overall programs, we include the file size of the compiled executable (.EXE file) program.

To illustrate performance, we have used test problems which usually have been devised to test function minimization

software rather than to solve real problems. One of these is the generalized Rosenbrock test function (GENROSE.FN or GENROSEJ.RES, Chapter 18). This test problem, we would stress, permits a fairly complicated function to be created with quite minimal program code. Problems arising out of real-world problems will generally involve more extensive programming.

Table 12-2-2 attempts to give some idea of the comparative sizes of the array and vector storage needed by the various methods to minimize a function. Note that no storage is included for the problem data which would commonly accompany real problems.

Table 12-2-3 attempts to illustrate the performance of our methods on the generalized Rosenbrock test problem using a BASIC interpreter. Table 12-2-4 gives similar data for compiled programs. These tables also show the general penalty paid for inclusion of bounds, masks, and parameter dispersion estimates. Here we see that CG is considerably slowed by these "extra" facilities, whereas it does not stand out from the other codes from the point of view of code length and complexity (Tables 12-2-1 and 12-2-2).

Our methods' performance varies with the number of parameters to be estimated. In part, this reflects the mechanism used to carry out the minimization of the loss function. For example, NM, VM, and MRT all use arrays to store information about the loss function. Access to these arrays, as compared to the vectors used in HJ, CG, and TN, may be time consuming. Therefore, we may expect the relative timings of our methods to differ when the number of parameters changes. This is illustrated in Tables 12-2-5a and 12-2-5b, and plotted for the timings only in Figures 12-2-1 and 12-2-2. In Figure 12-2-2, we see the advantage of CG and TN over VM for "many" parameters. From Table 12-2-5b, however, this is not only due to the number of gradient and function evaluations used.

Tables 12-2-5a and 12-2-5b also show results for the Steepest Descents (SD) method. In the present case, the program for this method was created by forcing CG to restart EVERY iteration, that is, to always use the gradient as the search direction. We include these results to point out that while the local descent may be provably "steepest", the overall progress to a minimum by SD may be hopelessly slow. The Hobbs problem further demonstrates this inefficiency (Table 12-2-7)

We bave briefly mentioned in Chapters 8 and 9 that the BASIC language translator used to run our programs may alter the performance. Table 12-2-7 compares the performance of our methods using the Hobbs weed infestation problem which has been chosen as an example problem for all the methods. At the same time the differences in behavior of our programs under two different BASIC interpreters are illustrated. It is widely assumed that since Microsoft is the author of both interpreters, the results should be comparable in their arithmetic. The present results show that this is not the case, which can be verified by running special programs such as PARANOIA (Karpinski, 1985).

Another factor influencing performance is the microprocessor which is used in the computer at hand. The original IBM PC uses an Intel 8088 microprocessor running at a clock speed of 4.77 MHz. Later "compatibles" chose to use other members of the Intel 8086 family of microprocessors, taking advantage of wider data paths, higher clock speeds or both. More recently, NEC has redesigned the operation of the 8088 (though Intel disputes this and legal action is being pursued at time of writing) so that the NEC V20 chip may replace the 8088, run at the same clock speed, yet achieve some speed advantage by completing some instructions more quickly. This behaviour, along with possible operating

system effects, is shown in Table 12-2-6. Our main reason for presenting these tables is to provide the user with some feeling for the variations in timing which arise due to factors outside of the software provided in this book.

Table 12-2-1. Code length for the methods presented

a. for subroutines only

	Method					
Subroutine:	HJ	NM	CG	TN	VM	MRT
lines F	142	218	192	233	193	245
lines S	78	130	102	134	95	152
ratio	1.82	1.68	1.88	1.74	2.03	1.61
bytes F	5688	8360	7560	8988	7221	8689
bytes S	1434	2723	1913	2801	1849	2790
ratio	3.97	3.07	3.95	3.21	3.91	3.11
symbols F	44	56	66	78	58	79
symbols S	35	40	54	62	39	59
ratio	1.26	1.40	1.22	1.26	1.49	1.34
references F	190	378	305	423	307	407
references S	120	279	179	276	179	275
ratio	1.58	1.35	1.70	1.53	1.72	1.48

Table 12-2-1. Code length for the methods presented (cont.)

b. for subroutines with simplified driver, generalized Rosenbrock test problem GENROSE.FN (or GENROSEJ.RES for MRT) and timing statements

	HJ	NM	CG	TN	VM	MRT
lines F	491	567	549	590	569	713
lines S	166	218	198	230	189	256
ratio	2.96	2.60	2.77	2.57	3.01	2.79
bytes F	19111	21783	21196	22624	21578	25402
bytes S	4253	5542	4945	5833	4830	5930
ratio	4.49	3.93	4.29	3.88	4.47	4.28
storage F	20097	23041	22401	23809	22785	26881
storage S	4582	5976	5343	6295	5210	6444
ratio	4.39	3.86	4.19	3.78	4.37	4.17
symbols F	126	138	143	159	140	172
symbols S	73	79	88	99	76	98
ratio	1.73	1.75	1.63	1.61	1.84	1.76
.EXE bytes F	33536	37504	36096	38528	36992	42752
.EXE bytes S	21120	25216	22784	24960	22784	25088
ratio	1.59	1.49	1.58	1.54	1.62	1.70

Table 12-2-2. Working storage in terms of arrays or vectors needed to operate program to estimate n parameters

	HJ	NM	CG	TN	VM	MRT
working store						
elements F	6n	n^2+9n+2	7n	12n	2n^2+8n	n^2+9n
elements S	2n	n^2+5n+2	4n	9n	n^2+5n	n^2+6n
value (n=10)F	60	192	70	120	280	190
value (n=10)S	20	152	40	90	150	160
ratio (n=10)	3.00	1.26	1.75	1.33	1.87	1.19
value (n=30)F	180	1172	210	360	2040	1170
value (n=30)S	60	1052	120	270	1050	1080
ratio (n=30)	3.00	1.11	1.75	1.33	1.94	1.08
value (n=50)F	300	2952	350	600	5400	2950
value (n=50)S	100	2752	200	450	2750	2800
ratio (n=50)	3.00	1.07	1.75	1.33	1.96	1.05

Table 12-2-3. Performance on the generalized Rosenbrock test problem (GENROSE.FN; or GENROSEJ.RES with MRT) under a program language interpreter.

Execution time for n=10 (except NM, n=5, initial stepsize=5) Timings on the Maestro IBM PC compatible using the BASICA interpreter and a NEC V20 micro-processor.

	HJ	NM	CG	TN	VM	MRT
fn evals F	3538	631	295	52	167	30
grads F	0	0	120	287	86	18
seconds F	1363	592	528	801	879	817
seconds S	1090	417	275	611	655	712
ratio	1.25	1.42	1.92	1.31	1.34	1.15

Table 12-2-4. Performance on the generalized Rosenbrock test problem (GENROSE.FN; or GENROSEJ.RES with MRT) using compiled programs.

Execution time for n=10 (except NM, n=5, initial stepsize=5) Timings on the Maestro IBM PC compatible using the Microsoft MS-BASIC compiler version 5.36 (NEC V20 micro-processor)

	HJ	NM	CG	TN	VM	MRT
fn evals F	5346	679	284	90	163	30
grads F	0	0	116	52	167	18
seconds F	281	102	68	117	126	75
seconds S	212	87	42	80	75	65
ratio	1.33	1.17	1.62	1.46	1.68	1.10

Table 12-2-5a. Performance with varying numbers of parameters. GENROSE and GENROSEJ test problems minimized by stripped down versions of the minimization codes. Programs compiled by Microsoft BASCOM v. 5.36 and executed on a Maestro PC clone under PC DOS 3.10 using a NEC V20 processor.

Method	Order	Gradient Evaluations	Function Evaluations	Time (s)	Minimal F value
HJ	2	0	590	20	1
	4	0	460	13	1
	6	0	1680	50	1
	8	0	1623	56	1.000001
	10	0	5346	212	1
NM	2	0	75	8	1.000001
	4	0	359	40	1.000014
0.1 initial	6	0	852	123	1.000003
stepsize	8	0	1812	334	1.000022
	10	0	1683	390	9.492024**
CG	2	27	70	5	1.000003
	4	40	102	10	1.000005
	6	67	171	19	1.000005
	8	86	211	27	1
	10	116	284	42	1
TN	2	28	13	4	1
	4	95	32	12	1.004218##
	6	221	55	32	1
	8	324	66	56	1
	10	391	90	80	1
VM	2	19	36	3	1
	4	25	56	8	1
	6	28	81	14	1
	8	55	97	35	1
	10	85	163	75	1
MRT	2	8	15	3	1
	4	10	20	7	1
	6	13	23	17	1
	8	16	26	35	1
	10	18	30	65	1
Steepest	2	499	1176	132	1.000016
Descents	4	1625	3934	552	1.000039
(SD)	6	1908	4627	817	1.00004
	8	2410	5839	1217	1.00004

** apparent failure, may be a plateau on the function surface
## abnormal end -- "too many iterations"

Table 12-2-5b. Performance with varying numbers of parameters. QUADSN test problem minimized by full versions of the minimization codes. Programs compiled by Microsoft BASCOM v. 5.36 and executed on a Corona PC-21 under MS DOS 2.11 using an Intel 8088 processor. Note that since the test problem is NOT a sum of squares, we do not have test results for MRT.

Method	Order	Gradient Evaluations	Function Evaluations	Time (s)	Minimal F value
HJ	5	0	89	5	-7.5 **
	10	0	176	12	-27.5 **
	15				
	20				
	25	0	439	52	-162.5 **
NM	5	0	294	57	-7.5
	10	0	2275	768	-27.40122 ##
	15				
	20				
	25				
CG	5	7	12	4	-7.5
	10	12	35	10	-27.5
	15	15	43	14	-60
	20	17	48	22	-105
	25	19	53	30	-162.5
TN	5	21	6	7	-7.5
	10	27	16	13	-27.5
	15	32	16	20	-60
	20	39	16	28	-105
	25	36	12	33	-162.5
VM	5	10	23	9	-7.5
	10	15	31	28	-27.5
	15	18	48	60	-60
	20	24	59	121	-105
	25	30	74	215	-162.5
Steepest	5	19	57	7	-7.5
Descents	10	30	74	18	-27.5
(SD)	15	40	104	30	-59.99999 <<
	20	44	116	42	-104.9999 <<
	25	53	134	60	-162.5 <<

** minimum found after one axial search, all other work is expended to recognize that minimum has been found
## failed to find minimum in reasonable time; terminated
<< a lower function value was found in the axial search. For n=25, this implies that binary/decimal conversion in the output display is inexact.

Table 12-2-6. Performance on different PC clones

The performance of the methods may vary according to the particular design of the computer used, operating system and BIOS (Basic Input/Output System). This is illustrated by the following timing figures from four machines which attempt to imitate the IBM PC:

Corona:	A Corona Data Systems PC-21 (Intel 8088 processor) MS-DOS 2.11. Pure Data 1464SP clock.
Noname:	a mongrel IBM PC compatible (Intel 8088 processor) MS-DOS 2.11. Pure Data 1464SP clock.
Noname V20:	Noname with the Intel processor replaced with a NEC V20 micro-processor. Pure Data 1464SP clock.
Maestro:	an IBM PC compatible with NEC V20 processor. IBM PC DOS 3.10. I/O Plus II clock board.

The test programs used the test problem HS25.RES (Appendix A) and the six minimization methods HJ, NM, CG, TN, VM and MRT.

Method	System	Fn	Grad	Time (sec.)	Ratio to Noname V20
HJ	Corona	3701	0	7858	1.33
	Maestro			5872	1.00
	Noname			5890	1.15
	Noname V20			6747	
NM	Corona	136	0	295	1.31
	Maestro			218	0.97
	Noname			5890	1.13
	Noname V20			6747	
CG	Corona	195	87	829	1.34
	Maestro			617	1.00
	Noname			714	1.15
	Noname V20			620	
TN	Corona	33	83	475	1.33
	Maestro			354	0.99
	Noname			410	1.15
	Noname V20			356	
VM	Corona	42	34	258	1.33
	Maestro			192	1.00
	Noname			223	1.15
	Noname V20			194	
MRT	Corona	18	11	58	1.32
	Maestro			44	1.00
	Noname			50	1.14
	Noname V20			44	

Table 12-2-7. Performance of the six methods with the Hobbs 3-parameter logistic function. Calculations run under the BASICA interpreter on the Maestro IBM PC compatible with NEC V20 processor, unless denoted 'gw', in which case the Microsoft GWBASIC interpreter was used, or 'c', in which case Microsoft MS-BASCOM compiler was used.

Method	Time	Grads	Fns	$F(\underline{B})$	B(1)	B(2)	B(3)
HJ	346s	0	445	2.587576	1.960721	4.90423	3.135595
HJ gw	332s	0	455	2.587578	1.960722	4.90423	3.135595
HJ c	51s	0	414	2.587335	1.962612	4.908446	3.134927
NM(0.1)	135s	0	112	2.587285	1.961959	4.908759	3.135553
NM gw	122s	0	110	2.587286	1.961601	4.909224	3.135929
NM c	32s	0	129	2.587251	1.961705	4.909004	3.135799
CG	146s	31	75	2.587745	1.964355	4.915891	3.135098
CG gw	113s	25	64	2.611886	1.992342	4.957549	3.12174
CG c	29s	38	92	2.588032	1.963899	4.917044	3.135696
TN	128s	44	16	2.588178	1.968247	4.915642	3.131997
TN gw	161s	58	29	2.587437	1.964402	4.912879	3.134416
TN c	30s	60	22	2.587258	1.962246	4.909766	3.135517
VM	94s	18	39	2.587269	1.961856	4.909156	3.135701
VM gw	102s	19	53	2.587259	1.961855	4.909165	3.135686
VM c	19s	18	63	2.587256	1.96173	4.909112	3.135785
MRT	40s	5	10	2.587261	1.961861	4.909162	3.135698
MRT gw	32s	4	9	2.587268	1.961851	4.90915	3.135703
MRT c	6s	5	10	2.587255	1.961862	4.909163	3.135698
SD	2366s	497	1193	2.60721	1.989319	4.95257	3.123068
SD gw	1082s	245	582	2.60465	1.986402	4.950441	3.124891
SD c	383s	508	1218	2.611789	1.99253	4.957191	3.121476

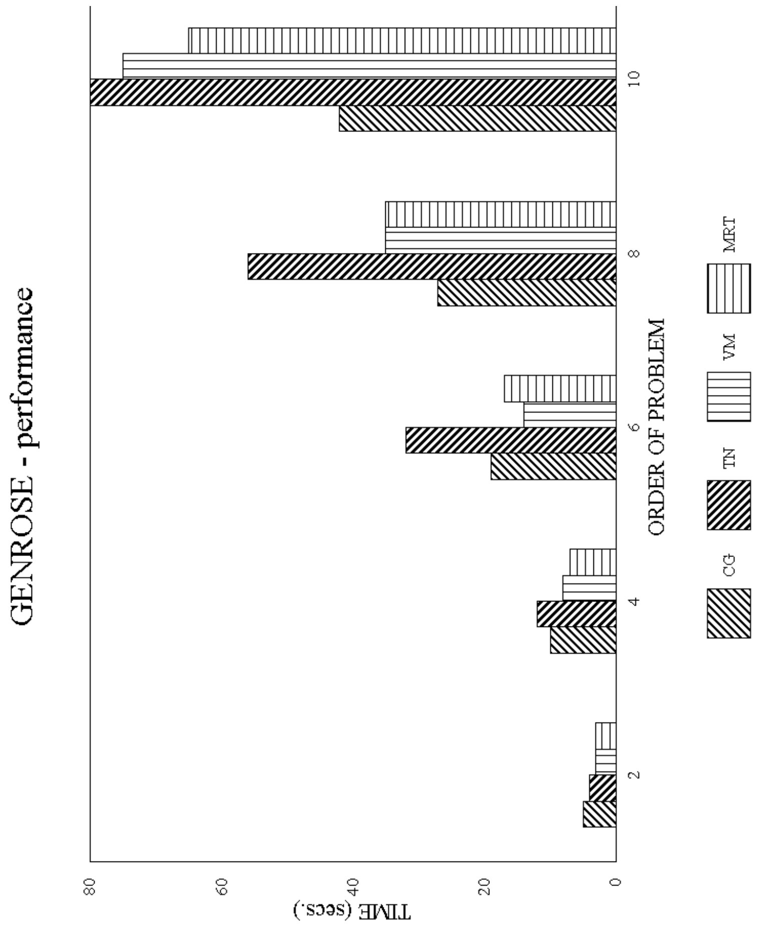


Figure 12-2-1. Performance of gradient minimization methods on the Generalized Rosenbrock problem.

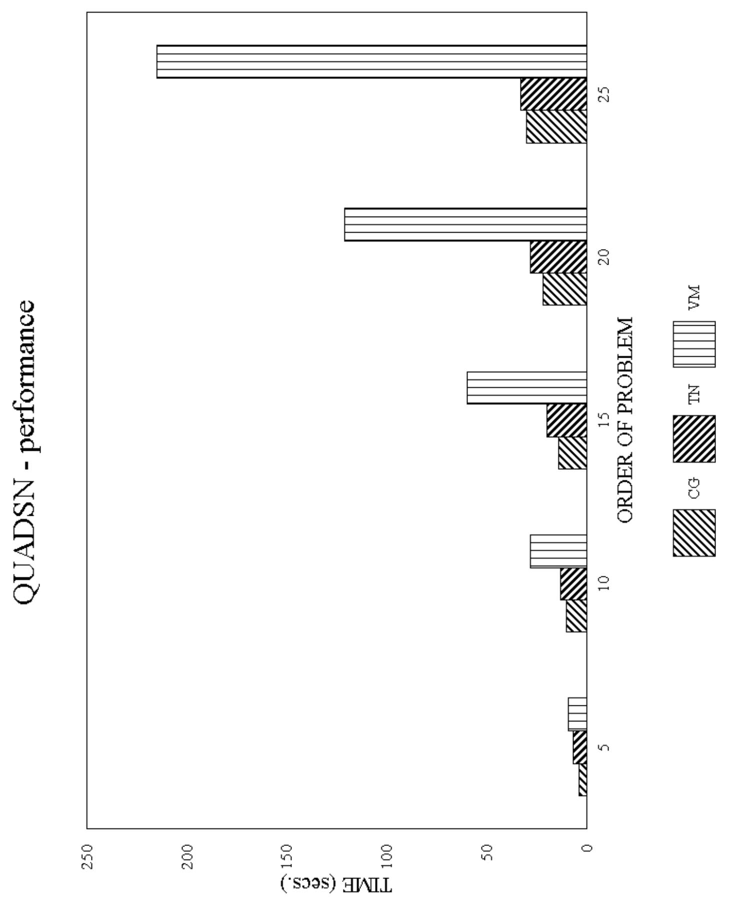


Figure 12-2-2. Performance of gradient minimization methods on the QUADSN test problem.

COVER SHEET

Chapter 13

Chapter title: Measures and consequences of nonlinearity

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

13

MEASURES AND CONSEQUENCES OF NONLINEARITY

13-0. Overview -- Consequences of Nonlinearity

We have already seen how "nonlinearity", as defined by the equations used to estimate parameters in models, generally forces us to use iterative algorithms in the estimation process. In this chapter we explore some of the other implications of nonlinearity, which primarily concern the variability of the parameter estimates with respect to the "true" but unknown parameters.

The variability or uncertainty in parameter estimates is presumed to be caused by errors or perturbations in the observed data for the problem at hand. Even for mathematically exact problems, there are perturbations made in representing floating-point numbers in a given machine representation, so that the sensitivity of the computed parameters to these representation errors may be useful to calculate. In general, we will assume that we can describe the distribution of observational errors or perturbations and provide an estimate of their magnitude. We will then attempt to compute the distribution and magnitude of the possible parameter estimates which may be computed for the problem. Since the computation of such parameter distributions is in general very difficult, we usually resort to approximations.

In linear least squares regression, it is assumed that the residual vector

(13-0-1) $\underline{r}(\underline{B},X,\underline{y}) = X\,\underline{B} - \underline{y}$

arises because the observations $\underline{y}$ have errors in the vector error from the true but unknown values y-true, that is,

(13-0-2) $\underline{y} = \underline{y\text{-true}} + \underline{error}$.

It is further assumed that the errors are uncorrelated and are drawn from the same distribution, a condition summarized by the phrase "independent, identically distributed" or i.i.d. If the distribution of errors is Gaussian (normal), then the parameter estimates themselves are also Gaussian distributed. Moreover, if the true but unknown parameters form a vector beta, the mean of the distribution of the parameter estimates $\underline{B}$ is beta. That is, if E( ) is the expectation operator,

(13-0-3) $E(\underline{B}) = \underline{beta}$

Furthermore, the variance of the parameter estimates is related to the underlying variance of a single element of $\underline{y}$, which we shall label "sigma-squared", via the Equation

(13-0-4) $\text{Variance}(\underline{B}_i) = (X^T X)^{-1}_{ii}\ \text{sigma}^2$

An estimate of sigma-squared is provided by

(13-0-5) $\underline{r}^T(\underline{B},X,y)\ \underline{r}(\underline{B},X,y) \;/\; (m - n)$

where m is the number of observations and n is the number of parameters.

In the context of this book, this is a result of limited utility, since the assumptions, namely

- independence of errors
- identical Gaussian distribution of errors
- error-free X matrix

along with the linear form of model, are not applicable to many of the estimation tasks encountered in this book. For example, many of the loss functions minimized in order to estimate parameters reflect a belief that the distribution of errors in the model is not Gaussian, and/or that they are not independent. Moreover, unlike observed data, constraints have no variability themselves, but limit the possible values of the parameters.

This book concerns software for nonlinear parameter estimation. The consequences of nonlinearity are clearly important to the development of such software. However, as yet the subject is still an active area of research and discussion. Therefore, we make no claims that our software provides the only method by which the consequences of nonlinearity may be assessed. Indeed, we have chosen to implement rather simple approaches to this topic. To complement this, we have tried to provide references to the literature, along with some indication of the major opinions and directions of research in the subject.

13-1. Why Nonlinear Estimation is Difficult

Nonlinear parameter estimation problems are difficult to solve because the scaling cannot be set, once and for all, before a solution is attempted. If we consider first the case of least squares problems, the properties of the problem can be compared with those of the well-known linear least squares regression problem.

Linear regression problems suffer from two main categories of difficulty:

1. failure of the regression assumptions of independence and constant variance (homoskedasticity) of the errors, and of zero correlation between the predictor variables and the errors;

2. linear dependency of the predictor variables, giving rise to poorly determined or non-unique parameter estimates (multicollinearity). A related concern is that the parameter estimates may be highly dependent on just one or two observations or points of high leverage.

These issues are sufficiently important that a number of books have been devoted to their exposition (exemplified by Belsley, Kuh and Welsch, 1980, and Draper and Smith, 1981).

The traditional solution method for finding the parameters $\underline{B}$ which fit a matrix of data (predictor variables) to a vector of dependent or explained values $\underline{y}$ is to solve the normal Equations

(13-1-1) $(X^T X) \underline{B} = X^T \underline{y}$

A more reliable alternative is to form the generalized inverse of X and compute

(13-1-2) $\underline{B} = X^+ \underline{y}$

where X^+ is the generalized inverse (Nash, 1979, Chapter 5; Nash, 1984b).

Equation (13-1-1) becomes the central iteration equation of the Gauss-Newton method (see Sections 7-1, 11-0) if X is replaced by the Jacobian J for a nonlinear least squares problem. Indeed, if the residuals are defined by the equation

(13-1-3) $\underline{r} = X \underline{B} - \underline{y}$

it is clear that J is identical to X when the Gauss-Newton method is applied to a linear regression problem. Since X (i.e. J) does NOT involve $\underline{B}$, the solution is obtained in one iteration. When the residuals are such that the Jacobian elements are functions of the parameters, the solution of the iteration equation is regarded as a modification to current parameter values, and is repeated until the modification does not alter the parameters. As we have seen, other iteration schemes, such as the Newton iteration, may be preferred, and

may be derived from different theoretical motivations. Nevertheless, all suffer from the basic consequence of nonlinearity, that is, the dependency of the iteration equations on the parameters. Thus, besides the difficulties which may be present in linear regression problems, we no longer have the possibility of universally scaling the problem. If we regard the scaling to be described by a diagonal matrix Z, then

(13-1-4) $\underline{B}' = Z \underline{B}$

(13-1-5) $X' = X Z^{-1}$

and the linear regression normal equations are simply (13-1-1) with X and $\underline{B}$ primed. The scaling is usually chosen to improve the condition of the normal equations. In particular, it is common to standardize each predictor variable (column of X) by subtracting its mean and dividing by its standard deviation. (This reduces the dimensionality of the problem by 1, and can only be used if the regression predictor variables include a constant, which disappears in the standardization. Such a standardization should not be applied, for example, to regression through the origin problems, since there is then no constant in the model.) Readers should note that this widespread practice is not without controversy. See, for example, the published debate between Hocking (1983, 1984) and Belsley (1984a), as well as the follow-up paper by Belsley (1984b).

In the nonlinear case, the elements of the Jacobian vary with the parameters so that a scaling which makes the iteration equations "easier" to solve can only be applied locally. Related obstacles to the estimation of nonlinear parameters are the following.

1. For some values of the parameters, the iteration may diverge, that is, the iterates do not tend toward a solution.

2. There may be sets of parameters which give rise to very large or very small values of the elements in the iteration equations. This is especially common when power or exponential functions appear in the model.

3. There may be domains in the parameter space over which the model function takes the same computed value. This is common if there are logarithms or very small powers in the model function.

As an example of some of the difficulties, consider the problem of minimizing the sum of squared residuals

(13-1-6) r(i,B) = exp(B(1)*x(i)) + 3*exp(B(2)*x(i)) - y(i)

where

(13-1-7) y(i) = exp(0.1*x(i)) + 3 * exp(x(i))

and

(13-1-8) x(i) = (i - 1)/5 for i = 1,2,3,4,5

It is relatively easy to compute the quantities needed for the iteration equation of different methods. At the solution point

(13-1-9) $\underline{B}^* = (0.1, 1)^T$

we find that the gradient is null and the Hessian (to 5 figures) is

(13-1-10) $H = \begin{pmatrix} 1.3719 & 7.6133 \\ 7.6133 & 43.028 \end{pmatrix} = J^T J$

and has eigenvalues 44.376 and 2.4070E-2. However, for the parameter set (1,2), where the sum of squares is 122.617, the gradient is

(13-1-11) $\underline{g} = (24.176, 150.24)^T$

and the Hessian is

(13-1-12) $H = \begin{pmatrix} 22.192 & 29.510 \\ 29.510 & 295.49 \end{pmatrix}$

which has eigenvalues 298.64 and 19.042. This matrix is no longer equal to the Jacobian inner product matrix. At the intermediate point (0.55,1.5), at which the sum of squares is 20.773,

(13-1-13) $\underline{g} = (7.2365, 42.836)^T$

(13-1-14) $H = \begin{pmatrix} 7.6498 & 14.864 \\ 14.864 & 119.53 \end{pmatrix}$

with eigenvalues 121.476 and 5.7089. These examples illustrate the nonlinearity of this problem, for which the Newton method converges in 25 iterations from (1,2) and 28 iterations from (0.55,1.5). The Gauss-Newton iteration is slightly faster, using 17 and 22 iterations respectively. These examples were run in single precision Microsoft GWBASIC. A fixed-strategy Marquardt algorithm was also tried (i.e. with fixed Levenberg Marquardt lambda factor), but this took a great many iterations, most of which were circling the solution. Applying our methods to this problem was also illuminating. While MRT and TN converged quite rapidly to the expected solution from both starting points, CG and VM converged slowly to the point

(13-1-15) $\underline{B} = (1.351285, 0.5908801)^T$

which has a sum of squares of 9.854645E-6. At this point, which is hard to distinguish from the desired solution on computed properties alone, we find

(13-1-16) $\underline{g} = (-2.11E-6, -3.34E-6)^T$

(13-1-10) $H = \begin{pmatrix} 7.9234 & 13.764 \\ 13.764 & 24.175 \end{pmatrix} \simeq J^T J$

which has eigenvalues 32.033 and 0.065503.

13-2. Measures of Nonlinearity

In order to assess the level of difficulty created by nonlinearity in an estimation problem, various authors have attempted to define measures of nonlinearity. For nonlinear least squares situations, one of the most useful approaches to date has been that of Bates and Watts (1980). Ratkowsky

(1983, Chapter 2) presents a readable discussion of this and some other measures of nonlinearity, as well as a FORTRAN program for computing the Bates/Watts intrinsic and parameter-effects nonlinearity measures. Here we will content ourselves with an outline of the spirit of these measures.

As we have seen above, the primary consequence of nonlinearity is that the gradient of the loss function being used as a criterion for parameter estimation is not a linear function of the parameters. If we consider a Taylor expansion of the model functions $\underline{Z}(\underline{B}, X)$ in Equation (2-3-1) about a point (parameter set) $\underline{b}$, then we have to second order, and ignoring the data matrix X,

(13-2-1) $\underline{Z}(\underline{B}) = \underline{Z}(\underline{b}) + J\ (\underline{B} - \underline{b}) + (\underline{B} - \underline{b})^T\ A\ (\underline{B} - \underline{b})$

where J is the Jacobian at $\underline{b}$ and A is the acceleration matrix of m by n by n second partial derivatives, evaluated at $\underline{b}$

(13-2-2) A_{ijk} = Partial Z(i,$\underline{b}$) w.r.t b_j and b_k

If the model function Z (i,$\underline{B}$) is linear in the parameters $\underline{B}$, then all elements of A are zero. When the model is nonlinear, this is no longer the case. Moreover, different parametrizations of the same essential model (for example, exp(- i * B) versus 10 ^ (- i * B) will give different values to the elements of A. Bates and Watts (1980) observed that the tangent plane to the model response surface (that is, the plot of Z(i,$\underline{B}$) versus $\underline{B}$, requiring n + 1 dimensions to represent) must be the same for all equivalent models. Thus all parametrizations have the same tangent plane at the point on the surface currently referred to as $\underline{b}$. The acceleration normal to the tangent plane must therefore be the same for all parametrizations, and is referred to as the intrinsic acceleration. This is converted to a curvature and scaled to allow comparison across a class of problems. The

acceleration parallel to the tangent plane yields a parameter effects curvature which may be reduced by appropriate reparametrization of the model. A number of workers suggest this approach; however, we feel that our clientele may have difficulty enough with familiar functional forms. Therefore, despite the advantages of reparametrization, we have not explicitly provided tools in this book to carry them out.

Because we envisage our work as being employed with loss functions more general than least squares, we do not present program code for the Bates/Watts nonlinearity measures. At the time of writing, we are aware of nonlinearity measures only for nonlinear least squares, further restricted to the unconstrained situation. Ratkowsky (1983, Sections 2.5 and 2.9) discusses a measure of bias in nonlinear estimation as well as the somewhat more complicated measure due to Box (1971). Given the current research activity in this area, it seems probable that more measures of nonlinearity will be developed, hopefully of wide applicability.

13-3. Convergence Tests and Restarts

For each of the methods we have presented, all of which are iterative, convergence tests have been provided to stop the program when a "satisfactory" point has been found. In Sections 4-3 and 7-6 we have already presented some ideas on this matter, which will now be summarized and extended.

The criteria for convergence of iterative function minimization methods include the following.

1. Parameter change criterion: no change in parameters is observed during an iteration.

2. Function change criterion: no change is observed in the loss function during an iteration.

3. Gradient conditions: the gradient is essentially zero, indicating that a stationary point has been found. Since a stationary point may be a local maximum, a local minimum or a saddle point, this criterion must usually be strengthened by the subsidiary condition that the Hessian matrix (that is, the matrix of second derivatives of the loss function) is positive semi-definite.

When constraints are imposed, we must also include conditions which test if all constraints are satisfied (Gill, Murray and Wright, 1981, pp. 73-82). Since our methods only deal with bounds on parameters, it is relatively straightforward to test whether a bound constraint is active, since all components of the gradients $\underline{g}$ must point "downhill towards the active bounds". This can be stated more scientifically. The function of f($\underline{B}$) must increase away from each active lower bound constraint. Thus, if

(13-3-1) $l_j \leq B(j)$

is active, then the gradient component

(13-3-2) $g_j(\underline{B}) > 0$

Similarly, if the upper-bound

(13-3-3) $B(k) \leq u_k$

is active,

(13-3-4) $g_k(\underline{B}) < 0$

so that the function increases as we move away from the constraint.

In the gradient-method codes which implement bounds constraints, the convergence tests include these tests on the contraints. For unconstrained minimization, our principal strategy has been to continue the iterative procedures until no new set of parameters is found with a strictly lower computed loss function value. "New set of paramters" is taken to imply that the parameter sets B-old and B-new are the "same", which for practical purposes (Equation 4-3-2)

requires either a tolerance or a scaling to avoid comparisons of rounding-error noise when small parameter values are encountered.

This strategy has been modified in certain instances:

1. In NM, the polytope may "flatten" so that the "highest" and "lowest" points have essentially the same function value. In such cases, we feel the iteration should be halted. Thus the convergence test used halts the minimization if the function values at the "highest" and "lowest" vertices of the current polytope differ by no more than

E9 * E9 * fscale

where E9 is the machine precision (see Appendix E, Computing Environment) and fscale is a quantity used to adjust the test value to the magnitude of the loss function for the problem. We use one plus the absolute value of the function at the initial set of parameters, that is,

(13-3-5) $\text{fscale} = 1 + \text{ABS}(f(\underline{B}^{(0)})$

The absolute value allows functions which are negative to be minimized. The constrant 1 avoids difficulties with arithmetic underflow - when the function value is small. (This may easily arise, for example, when using NM to test a function at the supposed minimum, if the function has zero-value at that point.)

2. In TN and CG, a "small" gradient norm may make it impossible to compute the next iterate, so a gradient norm test is included in these codes. In our experience, the gradient norm is quite sensitive to the scale of the loss function, and the gradient norm test is infrequently the mechanism which terminates our programs.

3. In CG and VM, the final test for lack of progress is

made with the steepest descent direction. This avoids the possibility that the current search direction could be "uphill" as a result of imperfectly computed updates to the search direction.

This last point introduces restarts, that is, re-initiation of the minimization method. Our experience is that it is worthwhile re-starting gradient methods with the steepest descent direction. The general post-solution analysis code POSTGEN performs an axial search around the supposed minimum. If a lower point is found, it is clear that the minimization should be restarted, and the code segment DRIVER allows the user to do this. By choosing NOT to enter a new set of initial paramter values, the programs will restart at the lowest point found, i.e. that found in the axial search. We have observed (Nash, 1979, p. 148) that this may be useful in attempting to minimize certain loss functions with NM. More generally, as we will mention in Chapter 14, it may be useful to restart the minimization from several points around a supposed minimum, especially if multiple minima are present or if convergence to a local minimum is suspected. We provide for manual (keyboard) entry of new initial parameters, but leave automation of this task to the interested user, since the choice of approach will depend greatly on the particular application and its context.

So far in this section, we have considered mechanisms for halting iterative function minimization methods. As pointed out by Bard (1974) and Bates and Watts (1981a), these should be called termination criteria rather than convergence criteria. Unfortunately, a true convergence test requires a fairly comprehensive understanding of the loss function. For unconstrained nonlinear least squares problems, Bates and Watts (1981a) have developed a criterion which halts the minimization of the sums of squares minimization if the current point is some small fraction of

the confidence region radius away from the least squares point. They indicate both theoretically and algorithmically how this test may be applied. In our own limited trials, based on the program LEQBO5 (Nash, 1984b; see also Section 11-7), we have observed this test criterion to significantly reduce the number of iterations taken before the iterations halt, while the parameter estimates are virtually identical to those obtained with the most stringent parameter-change criterion. That is, the termination criteria cause our algorithms to expend a great deal of computational effort deciding that the current point is a minimum of the loss function. This is illustrated in Table 13-3-1, which should be compared with Figure 11-7-1 illustrating the use of the singular value decomposition for nonlinear least squares.

Table 13-3-1. Comparison of convergence under parameter change termination criteria and Bates/Watts convergence test. The Hobbs test problem (Section 3-3) is used

Method	Starting point	Time (seconds)	Evaluations function	gradient
MRT	(2, 5, 3)	31	9	4
	(1, 1, 1)	71	19	10
LEQB05	(2, 5, 3)	60	9	7
	(1, 1, 1)	236	56	26
LEQB05 &	(2, 5, 3)	34	4	4
Bates/Watts	(1, 1, 1)	227	53	25

The Bates-Watts criterion is recommended if the application task involves unconstrained nonlinear least squares problems. In the more general case, similar criteria remain to be developed. As far as we are aware, a convergence criterion which halts a minimization algorithm using similar "close enough" concepts has yet to be developed for general loss functions or problems involving constraints. Therefore, we have not used the Bates-Watts convergence test in this book. Having so decided, there was less motivation to use a Jacobian matrix decomposition approach to nonlinear least squares, and the MRT code was chosen. This code is poorly suited to inclusion of the Bates/Watts test, but appears overall to be efficient and reliable. However, we anticipate that decomposition approaches will see improvements in speed shortly, at which point MRT should be replaced.

13-4. Estimates of Parameter Dispersion

Having minimized a loss function to determine estimates for a set of parameters $\underline{B}$, we may wish to know how "good" the estimates are. In particular, the dispersion of the parameter estimates may be very important if the parameter estimates are to be used in decision-making. If there is a statistical distribution of "errors" made in observing the data, that is, if the "true" residuals have an unexpected distribution of values, we would ideally like to know the distribution of the parameters $\underline{B}$. For a linear model, where

(13-4-1) $\underline{r}(\underline{B}) = X\,\underline{B} - \underline{y}$

with the assumption that the errors in observation are concentrated in the response vector $\underline{y}$ and are independent of each other, then variances of the parameters $\underline{B}$ form a matrix

(13-4-2) $V(\underline{B}) = s^2\ (X^T X)^{-1}$

where

(13-4-3) $s^2 = (\underline{r}^T\,\underline{r})/(m - n)$

because the estimate for $\underline{B}$ is

(13-4-4) $\underline{B} = (X^T X)^{-1}\ X^T\,\underline{y}$

which is <u>linear</u> in $\underline{y}$, the relationship can be discovered

which relates the variance of the "errors" in observation with the variances of the estimated parameters. Thus, for linear models, under the regression assumptions, estimates of the dispersion of the parameters can be computed.

In the nonlinear least squares case, similar ideas can be used. Since the locally linearized model gives rise to the Gauss-Newton Equations (11-0-8), it is temptiong to suggest dispersion estimates

(13-4-5) $V(\underline{B}) = s^2\ (J^T J)^{-1}$

where J is the Jacobian and s^2 is defined as in (13-4-3) but with the nonlinear residuals (2-6-1).

Traditionally, the matrix $V(\underline{B})$ is not computed, but only the square roots of its diagonal elements, which are often referred to as the <u>standard errors</u> of the corresponding parameters. That is, the j,j element of V gives the variance of B(j). Some implementations of Gauss-Newton methods compute an inverse of the matrix V (or a scaled version thereof), and can proceed to compute estimates of the <u>correlation</u> matrix of the parameters

(13-4-6) $CORR(\underline{B}) = Q(\underline{B})^T\ V(\underline{B})\ Q(\underline{B})$

where $Q(\underline{B})$ is the diagonal matrix whose non-zero elements are the reciprocals of the square roots of the corresponding diagonal elements of $V(\underline{B})$. An alternative view of the operations to compute $CORR(\underline{B})$ is that the i,j element of $V(\underline{B})$ is divided by the square root of the product of the i,i and j,j elements. Active bounds and masks complicate the programming in detail, though not in concept. In POSTMRT we build up the correlation estimates piecemeal, since we do not have the entire matrix $V(\underline{B})$ available at any one stage, but only a single row (or column).

Another linearization which could be used is

(13-4-7) $V'(\underline{B}) = H(\underline{B})^{-1}$

which is the inverse Hessian at the solution point. These and other approaches to computing the dispersion are used to estimate <u>confidence regions</u>, that is, regions in the parameter space wherein the parameters are found with a certain probability. Linearization methods give dispersion measures, from which confidence intervals on the parameters may be calculated using conventional statistical formulas. The <u>lack-of-fit</u> method and <u>likelihood</u> method produce confidence regions more or less directly, though with considerable computational effort. Donaldson and Schnabel (1985) provide a readable account of all of these methods, along with the results of a Monte Carlo study of the <u>coverage</u> of five different methods. That is, they study whether the confidence regions produced by different methods contain the true intervals, i.e., <u>cover</u> them.

The conclusions of the Monte Carlo study were that Equation (13-4-5) performed better than (13-4-7) and one other linearization technique, particularly when numerical approximations to derivatives were used. However, none were as good as the more expensive and difficult lack-of-fit and likelihood methods. Donaldson and Schnabel conclude that "it appears that there is a need for new methods for estimating confidence regions that are both reliable and easy to report". We concur with this opinion, and regret that we are able to offer the reader only a partial response to the need for estimates of parameter dispersion.

It is also possible to obtain parameter dispersion estimates from the information in the final polytope of the Nelder-Mead method (see Spendley, 1969 for details). As this adds to the length of the code required, particularly when bounds and masks are allowed, we have not included this approach in our software. By using the Marquardt method with numerical derivatives, we can obtain the parameter dispersion estimates, even if the parameter estimates have been found by

one of the direct search methods. Spendley's ideas are directed to nonlinear least squares problems, while the approach in the next section deals with general loss functions.

13-5. A Genereal Post-Solution Anlaysis

For general loss functions, possibly involving constraints, we have chosen to use very simple indicators of the parameter dispersion. In order to test the sensitivity of the loss function to small changes in each parameter, we perform a simple axial search, evaluating

(13-5-1) $f(\underline{B} - \underline{step}(j)),\ f(\underline{B} + \underline{step}(j))$

for each j, where <u>step</u>(j) is a null vector except in its j-th element, which has the value

(13-5-2) $step(j)_j = E8 * (ABS(B(J)) + E8$

where E8 is a small multiple of the square root of the machine precision. We report the <u>change</u> in F for each such step, providing two primary pieces of information about the loss function at the supposed solution:

1. the relative sensitivity of the loss function along each parameter axis;

2. the symmetry of the loss function surface about the supposed solution. (If the solution is on an active bound constraint, a large value is assigned to the computed function.)

A quadratic model for the surface at the solution point gives rise to a parabola along each parameter axis. Labelling the function values for negative and positive steps as f(-) and f(+), with the function value at the supposed solution as f(0), we have

(13-5-3) $f(0) = a$

$f(-) = a - b * stepj + c * stepj * stepj$

$f(+) = a + b * stepj + c * stepj * stepj$

using stepj from (13-5-2). The coefficients a, b, and c are thus

(13-5-4) $a = f(0)$

$b = (f(+) - f(-)) / (2 * stepj)$

$c = (f(+) + f(-) - 2 * f(0)) / (2 * stepj * stepj)$

Since c is the second derivative of the parabolic approximation, we easily convert it to a radius of curvature (Protter and Morrey, 1964, page 394)

(13-5-5) $\text{r. of curvature} = (1 + b * b)^{3/2} / c$

(The conventional radius of curvature uses the absolute value of this quantity. However, it should always be positive for the present application, and a negative number is useful as an indicator that a local minimum has not been found.)

For a loss function which is symmetric about the solution, we will have b = 0. We therefore convert this parameter to the tilt by computing its arctangent in degrees. Upper and lower bounds will give tilts near -90 and +90 degrees respectively.

Other mechanisms for the calculation of measures of dispersion for the parameter estimates include the jackknife and the bootstrap (Efron, 1982). These techniques are called resampling plans because one or more of the observations are omitted from the data and the parameter estimates recomputed. If the i-th observation is omitted, we compute a set of estimated parameters $\underline{B(i)}$. The m sets of estimated paramters, together with the set of parameters $\underline{B}$ calculated with the entire sample then give an estimate for the variances of the n parameters (in vector form)

(13-5-6) $$\underline{VAR} = (m - 1) \sum_{i=1}^{m} (\underline{B(i)} - \underline{B})^2 / m$$

According to Efron (1982), this dispersion estimate is due to John Tukey. Note that the square roots of these numbers should be compared to the standard errors or radii of curvature computed in our codes. We do not provide code here for the jackknife or bootstrap, as these techniques are computationally very demanding. Nevertheless, our programs could easily be modified to allow such computations. Furthermore, personal computers such as the IBM PC and clones provide a relatively cheap computing resource for such calculations, especially if set up to run unattended at night or over weekends.

We thank Dr. Morven Gentleman for reminding us of the use of the jackknife as a possible dispersion estimate method.

13-6. POSTGEN - General Post-Solution Analysis

POSTGEN is a program code which can be added to any of the function minimization algorithms. Indeed, we include it in POSTMRT and POSTVM (though with minor changes in line numbers). Because we do not know the structure of the loss function used in the estimation, POSTGEN can only provide crude estimates of parameter dispersion based on a search along each parameter axis to develop the radius of curvature and tilt measures discussed above.

With the Nelder - Mead code (NM), POSTGEN is an integral part of the minimization (Nash, 1979, p. 189), since we require the axial search to escape from situations where the polytope may have "flattened" to such an extent that progress cannot be made.

The output from POSTGEN has already been included in the examples given in Listings 5-2-1, 5-2-2, and 6-2-1, and is a part of the POSTVM and POSTMRT examples below.

Listing 13-6-1. The POSTGEN Code.

```
              POSTGEN.BAS         08-07-1986   13:36:04

6000 PRINT "POSTGEN.BAS 851118 -- GENERAL POST SOLUTION ANALYSIS"
6005 PRINT #3,"POSTGEN.BAS 851118 -- GENERAL POST SOLUTION ANALYSIS"
6010 REM CALLS
6015 REM     FUNCTION F  -- line 2000
6020 REM     RESIDUAL R1 -- line 3500 (only if J6=1)
6025 REM
6030 REM INPUTS TO THE ROUTINE
6035 REM    X() -- the vector of final parameter estimates
6040 REM    N   -- the number of parameters in the function F(B)
6045 REM    M   -- the number of data points (0 for a .FN)
6050 REM    O( , ) -- masks and bounds
6055 REM    Y( , ) -- the data array for the problem
6060 REM    I5  -- the number of masked (fixed) parameters
6065 REM    J6  -- flag which is 1 if residuals can be computed
6070 REM    F0  -- value of the function at the supposed minimum
6075 REM
6080 REM    B9, E5, E9 -- constants provided by ENVRON
6085 REM
6090 REM OUTPUT FROM THE ROUTINE
6095 REM    F1 -- lower function value than F0 (if found, else a
6100 REM          higher value to prevent a restart in the DRIVER)
6105 REM    X() -- new parameter values if a lower point is found
6110 REM
6115 REM need array T(N) to store tilt angles
6120 LET T5=1/E9: REM used for active bounds ('large' F(B))
6125 LET I6=0: REM flag for active bounds (-2=l.b.,-1=u.b.,0=inactive)
6130 IF M<=N-I5 THEN 6155
6135 LET F9=F0/(M-N+I5): REM PSEUDO-SIGMA^2
6140 PRINT "LOSS FUNCTION PER DEGREE OF FREEDOM (SIGMA^2) =";F9
6145 PRINT #3,"LOSS FUNCTION PER DEGREE OF FREEDOM (SIGMA^2) =";F9
6150 GOTO 6160
6155 LET F9=-1
6160 FOR J=1 TO N
6165 LET E8=E5*SQR(E9): REM now compute points left and right
6170 LET E8=E8*(ABS(X(J))+E8): REM stepsize
6175 PRINT "B(";J;")=";
6180 PRINT #3,"B(";J;")=";
6185 IF O(J,3)=0 THEN 6400: REM masked
6190 LET B(J)=X(J)-E8
6195 IF B(J)<O(J,1) THEN 6225
```

```
6200 LET I3=0
6205 GOSUB 2000: REM function value
6210 IF I3=1 THEN 6230
6215 IF F<F0 THEN 6570: REM lower function value
6220 GOTO 6235
6225 LET I6=-2: REM active lower bound
6230 LET F=T5: REM failure
6235 LET F3=F
6240 LET B(J)=X(J)+E8
6245 IF B(J)>O(J,2) THEN 6275
6250 LET I3=0
6255 GOSUB 2000: REM function value
6260 IF I3=1 THEN 6280
6265 IF F<F0 THEN 6570: REM lower function value
6270 GOTO 6285
6275 LET I6=-1: REM active upper bound
6280 LET F=T5: REM failure
6285 LET F4=F
6290 LET B(J)=X(J): REM reset parameter value
6295 LET C1=.5*(F4-F3)/E8: REM linear term -- should be zero
6300 LET C2=(F4+F3-2*F0)/(2*E8*E8): REM quadratic term
6305 IF I6=-2 THEN LET C2=(F4-F0)/(E8*E8): REM symmetrized curvature
6310 IF I6=-1 THEN LET C2=(F3-F0)/(E8*E8): REM when bounds active
6315 LET C0=1+C1*C1: REM denominator for curvature
6320 LET C2=C2/(C0*SQR(C0))
6325 LET C0=B9: REM set large in case of singularity
6330 IF C2<>0 THEN LET C0=1/C2: REM radius of curvature for problem
6335 LET T(J)=-45*ATN(C1)/ATN(1): REM tilt -- DIMension T in HJ, NM
6340 PRINT USING "##.######^^^^";B(J);: REM !!
6345 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6350 PRINT "  step=";
6355 PRINT #3,"  step=";
6360 PRINT USING "##.##^^^^";E8;: REM !!
6365 PRINT #3,USING "##.##^^^^";E8;: REM !!
6370 PRINT "  f-, f+";
6375 PRINT #3,"  f-, f+";
6380 PRINT USING " ##.####^^^^";F3;F4: REM !!
6385 PRINT #3,USING " ##.####^^^^";F3;F4: REM !!
6390 LET X(J)=C0: REM save radius of curvature in x()
6395 GOTO 6420
6400 PRINT USING "##.######^^^^";B(J);: REM !!
6405 PRINT " MASKED"
6410 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6415 PRINT #3," MASKED"
6420 NEXT J
6425 PRINT "        best function value found is            ";
6430 PRINT #3,"        best function value found is            ";
6435 PRINT USING " ##.####^^^^";F0: REM !!
6440 PRINT #3,USING " ##.####^^^^";F0: REM !!
6445 PRINT
```

```
6450 PRINT #3,
6455 PRINT "radii of curvature for surface along axial directions"
6460 PRINT #3,"radii of curvature for surface along axial directions"
6465 PRINT "  & tilt angle in degrees"
6470 PRINT #3,"  & tilt angle in degrees"
6475 PRINT
6480 PRINT #3,
6485 FOR J=1 TO N
6490 IF O(J,3)=0 THEN 6535
6495 PRINT "for B(";J;")  R. OF CURV. = ";
6500 PRINT #3,"for B(";J;")  R. OF CURV. = ";
6505 PRINT USING "##.###^^^^";X(J);: REM !!
6510 PRINT #3,USING "##.###^^^^";X(J);: REM !!
6515 PRINT "   tilt = ";
6520 PRINT #3,"   tilt = ";
6525 PRINT USING "####.#####";T(J): REM !!
6530 PRINT #3,USING "####.#####";T(J): REM !!
6535 NEXT J
6540 IF J6=1 THEN GOSUB 6900: REM print residuals
6545 FOR J=1 TO N
6550 LET X(J)=B(J): REM reset parameter for restart
6555 NEXT J
6560 LET F1=F0+ABS(F0)+1: REM to ensure no restart
6565 RETURN: REM to DRIVER
6570 PRINT
6575 PRINT #3 ,
6580 PRINT "lower function value found"
6585 PRINT #3,"lower function value found"
6590 IF J6=1 THEN GOSUB 6900: REM print residuals
6595 FOR K=1 TO J
6600 LET X(K)=B(K): REM reset parameter values -- note, j included
6605 NEXT K
6610 LET F1=F: REM to indicate lower function value
6615 RETURN

     Line no.    Referenced in line(s)
      2000      6205  6255 -- the function subroutine
      6155      6130
      6160      6150
      6225      6195
      6230      6210
      6235      6220
      6275      6245
      6280      6260
      6285      6270
      6400      6185
      6420      6395
      6535      6490
      6570      6215  6265
      6900      6540  6590 -- residual display
```

```
 Symbol     Referenced in line(s)
 B(         6190  6195  6240  6245  6290  6340  6345  6400  6410
            6550  6600 -- the parameter vector
 B9         6325 -- a "large" number
 C0         6315  6320  6325  6330  6390 -- denominator for curvature
 C1         6295  6315  6335 -- linear term
 C2         6300  6305  6310  6320  6330 -- quadratic term
 E5         6165 -- a value used for scaled comparisons (= 10)
 E8         6165  6170  6190  6240  6295  6300  6305  6310  6360
            6365 -- stepsize
 E9         6120  6165 -- the machine precision
 F          6215  6230  6235  6265  6280  6285  6610 -- function
            value
 F0         6135  6215  6265  6300  6305  6310  6435  6440  6560
            -- value of the function at the supposed minimum
 F1         6560  6610 -- lower function value than F0 if found, else
            a higher value to prevent a restart in the DRIVER
 F3         6235  6295  6300  6310  6380  6385 -- function value minus
            stepsize from supposed minimum
 F4         6285  6295  6300  6305  6380  6385 -- function value plus
            stepsize from supposed minimum
 F9         6135  6140  6145  6155 -- loss function per degree of
            freedom (sigma^2)
 I3         6200  6210  6250  6260 -- flag for computable function
            (=1 for failure)
 I5         6130  6135 -- the number of masked (fixed) parameters
 I6         6125  6225  6275  6305  6310 -- flag for active bounds
            (-2=lower bound, -1=upper bound, 0=inactive)
 J          6160  6170  6175  6180  6185  6190  6195  6240  6245
            6290  6335  6340  6345  6390  6400  6410  6420  6485
            6490  6495  6500  6505  6510  6525  6530  6535  6545
            6550  6555  6595 -- a loop control counter
 J6         6540  6590 -- flag which is 1 if residuals can be computed
 K          6595  6600  6605 -- a loop control counter
 M          6130  6135 -- the number of data points
 N          6130  6135  6160  6485  6545 -- number of parameters in
            loss function
 O(         6185  6195  6245  6490 -- bound/masks information storage
 T(         6335  6525  6530 -- tilt angles
 T5         6120  6230  6280 -- a "large" number assigned to the
           function value when constraints are violated. If T5 is too
           large, the calculation of dispersion estimates gives rise
            to "overflows".
 X(         6170  6190  6240  6290  6390  6505  6510  6550  6600
            -- the vector of final parameter estimates
=======================================================================
    LINES: 124      BYTES: 4660     SYMBOLS: 40     REFERENCES: 161
```

13-7. POSTMRT - a Jacobian Dispersion Estimate

Below in Listing 13-7-1 we present code which computes the dispersion measures given by a linearization involving the Jacobian as in Equation (13-4-5). The general post-solution analysis is also performed for comparison purposes. In displaying the results, standard errors for the parameters are undefined if the estimate, SIGMA^2, of the variance of the "errors" in the data cannot be computed. One such situation occurs when MRT is used to solve nonlinear equations and the number of "data points" M is equal to the number of parameters N. A singular Jacobian provides another cause. We arbitrarily display the number -9999 in such cases.

Listing 13-7-2 presents the output of POSTMRT applied to the Hobbs weed infestation problem. This may be thought of as a continuation of the output presented in Listing 11-6-1.

Listing 13-7-1. The POSTMRT Code.

```
              POSTMRT.BAS         08-07-1986   13:36:43

     6000 PRINT "POST-SOLUTION ANALYSIS FOR MARQUARDT-NASH NLLS"
6005 PRINT #3,"POST-SOLUTION ANALYSIS FOR MARQUARDT-NASH NLLS"
6010 REM 860504
6015 REM CALLS
6020 REM     RESIDUAL R1 -- line 3500
6025 REM     CHOLESKY DECOMPOSITION -- line 1596 (MRT)
6030 REM     CHOLESKY BACKSUBSTITUTION -- line 1696 (MRT)
6035 REM     SUM OF SQUARES -- line 1848 (MRT)
6040 REM
6045 REM INPUTS TO THE ROUTINE
6050 REM    X() -- the vector of final parameter estimates
6055 REM    C() -- the Jacobian matrix stored as a vector
6060 REM    G() -- the gradient of the sum of squares loss function
6065 REM    N   -- the number of parameters in the function F(B)
6070 REM    M   -- the number of data points (0 for a .FN)
6075 REM    O( , ) -- masks and bounds
6080 REM    Y( , ) -- the data array for the problem
6085 REM    I5  -- the number of masked (fixed) parameters
6090 REM    F0  -- value of the function at the supposed minimum
```

```
6095 REM
6100 REM    B9, E5, E9 -- constants provided by ENVRON
6105 REM
6110 REM OUTPUT FROM THE ROUTINE
6115 REM    F1 -- lower function value than F0 (if found, else a
6120 REM          higher value to prevent a restart in the DRIVER)
6125 REM    X() -- new parameter values if a lower point is found
6130 REM
6135 REM need array T(N) to store tilt angles
6140 FOR J=1 TO N2
6145 LET A(J)=C(J): REM use unweighted matrix
6150 NEXT J
6155 IF M>N-I5 THEN GOTO  6170
6160 LET F9=-1
6165 GOTO 6200: REM standard errors cannot be computed
6170 LET F9=F0/(M-N+I5)
6175 PRINT "SIGMA^2 = LOSS FUNCTION PER DEGREE OF FREEDOM =";F9
6180 PRINT #3,"SIGMA^2 = LOSS FUNCTION PER DEGREE OF FREEDOM =";F9
6185 LET I3=0
6190 GOSUB 1596: REM Choleski decomposition (unweighted matrix)
6195 IF I3=0 THEN GOTO 6220
6200 FOR J=1 TO N
6205 LET D(J)=-9999: REM flag singular sscp matrix or too few data points
6210 NEXT J
6215 GOTO 6360
6220 PRINT
6225 PRINT #3,
6230 PRINT "PARAMETER CORRELATION ESTIMATES"
6235 PRINT #3,"PARAMETER CORRELATION ESTIMATES"
6240 FOR K9=1 TO N
6245 PRINT "ROW ";K9;" : ";
6250 PRINT #3,"ROW ";K9;" : ";
6255 FOR J=1 TO N
6260 LET T(J)=0
6265 NEXT J
6270 LET T(K9)=1
6275 GOSUB 1696: REM Choleski backsubstitution
6280 LET D(K9)=SQR(T(K9)*ABS(F9))
6285 FOR J=1 TO K9
6290 LET T1=D(K9)*D(J)
6295 IF T1=0 THEN LET T1=1: REM to account for masks
6300 REM note no explicit test for masks here
6305 PRINT USING "###.#####";ABS(F9)*T(J)/T1;
6310 PRINT #3,USING "###.#####";ABS(F9)*T(J)/T1;
6315 IF 4*INT(J/4)<J THEN 6340
6320 PRINT
6325 PRINT "          ";
6330 PRINT #3,
6335 PRINT #3,"          ";
6340 NEXT J
```

```
6345 PRINT
6350 PRINT #3,
6355 NEXT K9
6360 PRINT
6365 PRINT #3,
6370 PRINT "SOLUTION WITH ERROR MEASURES AND GRADIENT OF SUM OF SQUARES"
6375 PRINT #3,"SOLUTION WITH ERROR MEASURES AND GRADIENT OF SUM OF SQUARES
6380 PRINT
6385 PRINT #3,
6390 FOR J=1 TO N
6395 PRINT "B(";J;")=";B(J);"  STD ERR=";D(J);"  GRAD(";J;")=";G(J)*2
6400 PRINT #3,"B(";J;")=";B(J);"  STD ERR=";D(J);"  GRAD(";J;")=";G(J)*2
6405 NEXT J
6410 INPUT "HIT [CR] TO CONTINUE";X$
6415 PRINT: REM start of POSTGEN.BAS
6420 PRINT #3,
6425 LET T5=1/E9: REM used for active bounds ('large' F(B))
6430 LET I6=0: REM flag for active bounds (-2=l.b.,-1=u.b.,0=inactive)
6435 FOR J=1 TO N
6440 LET E8=E5*SQR(E9): REM now compute points left and right
6445 LET E8=E8*(ABS(X(J))+E8): REM stepsize
6450 PRINT "B(";J;")=";
6455 PRINT #3,"B(";J;")=";
6460 IF O(J,3)=0 THEN 6675: REM masked
6465 LET B(J)=X(J)-E8
6470 IF B(J)<O(J,1) THEN 6500
6475 LET I3=0
6480 GOSUB 1848: REM function value
6485 IF I3=1 THEN 6505
6490 IF F<F0 THEN 6845: REM lower function value
6495 GOTO 6510
6500 LET I6=-2: REM active lower bound
6505 LET F=T5: REM failure
6510 LET F3=F
6515 LET B(J)=X(J)+E8
6520 IF B(J)>O(J,2) THEN 6550
6525 LET I3=0
6530 GOSUB 1848: REM function value
6535 IF I3=1 THEN 6555
6540 IF F<F0 THEN 6845: REM lower function value
6545 GOTO 6560
6550 LET I6=-1: REM active upper bound
6555 LET F=T5: REM failure
6560 LET F4=F
6565 LET B(J)=X(J): REM reset parameter value
6570 LET C1=.5*(F4-F3)/E8: REM linear term -- should be zero
6575 LET C2=(F4+F3-2*F0)/(2*E8*E8): REM quadratic term
6580 IF I6=-2 THEN LET C2=(F4-F0)/(E8*E8): REM symmetrized curvature
6585 IF I6=-1 THEN LET C2=(F3-F0)/(E8*E8): REM when bounds active
6590 LET C0=1+C1*C1: REM denominator for curvature
```

```
6595 LET C2=C2/(C0*SQR(C0))
6600 LET C0=B9: REM set large in case of singularity
6605 IF C2<>0 THEN LET C0=1/C2: REM radius of curvature for problem
6610 LET T(J)=-45*ATN(C1)/ATN(1): REM tilt -- DIMension T in HJ, NM
6615 PRINT USING "##.######^^^^";B(J);: REM !!
6620 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6625 PRINT "  step=";
6630 PRINT #3,"  step=";
6635 PRINT USING "##.##^^^^";E8;: REM !!
6640 PRINT #3,USING "##.##^^^^";E8;: REM !!
6645 PRINT "  f-, f+";
6650 PRINT #3,"  f-, f+";
6655 PRINT USING " ##.####^^^^";F3;F4: REM !!
6660 PRINT #3,USING " ##.####^^^^";F3;F4: REM !!
6665 LET X(J)=C0: REM save radius of curvature in x()
6670 GOTO 6695
6675 PRINT USING "##.######^^^^";B(J);: REM !!
6680 PRINT " MASKED"
6685 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6690 PRINT #3," MASKED"
6695 NEXT J
6700 PRINT "        best function value found is            ";
6705 PRINT #3,"        best function value found is            ";
6710 PRINT USING " ##.####^^^^";F0: REM !!
6715 PRINT #3,USING " ##.####^^^^";F0: REM !!
6720 PRINT
6725 PRINT #3,
6730 PRINT "radii of curvature for surface along axial directions"
6735 PRINT #3,"radii of curvature for surface along axial directions"
6740 PRINT "  & tilt angle in degrees"
6745 PRINT #3,"  & tilt angle in degrees"
6750 PRINT
6755 PRINT #3,
6760 FOR J=1 TO N
6765 IF O(J,3)=0 THEN 6810
6770 PRINT "for B(";J;")  R. OF CURV. = ";
6775 PRINT #3,"for B(";J;")  R. OF CURV. = ";
6780 PRINT USING "##.###^^^^";X(J);: REM !!
6785 PRINT #3,USING "##.###^^^^";X(J);: REM !!
6790 PRINT "   tilt = ";
6795 PRINT #3,"   tilt = ";
6800 PRINT USING "####.#####";T(J): REM !!
6805 PRINT #3,USING "####.#####";T(J): REM !!
6810 NEXT J
6815 GOSUB 6900: REM print residuals
6820 FOR J=1 TO N
6825 LET X(J)=B(J): REM reset parameter for restart
6830 NEXT J
6835 LET F1=F0+ABS(F0)+1: REM to ensure no restart
6840 RETURN: REM to DRIVER
```

```
6845 PRINT
6850 PRINT #3 ,
6855 PRINT "lower function value found"
6860 PRINT #3,"lower function value found"
6865 GOSUB 6900: REM print residuals
6870 FOR K=1 TO J
6875 LET X(K)=B(K): REM reset parameter values -- note, j included
6880 NEXT K
6885 LET F1=F: REM to indicate lower function value
6890 RETURN

     Line no.    Referenced in line(s)
      1596      6190 -- Cholesky decomposition in MRT
      1696      6275 -- Cholesky backsubstitution in MRT
      1848      6480  6530 -- sum of sqares in MRT
      6170      6155
      6200      6165
      6220      6195
      6340      6315
      6360      6215
      6500      6470
      6505      6485
      6510      6495
      6550      6520
      6555      6535
      6560      6545
      6675      6460
      6695      6670
      6810      6765
      6845      6490  6540
      6900      6815  6865 -- residual display

      Symbol     Referenced in line(s)
     A(         6145 -- the Jacobian matrix stored as a vector
     B(         6395  6400  6465  6470  6515  6520  6565  6615  6620
                6675  6685  6825  6875 -- the parameter vector
     B9         6600 -- a "large" number
     C(         6145 -- the Jacobian matrix stored as a vector
     C0        6590  6595  6600  6605  6665 -- denominator for curvature
     C1         6570  6590  6610 -- linear term
     C2         6575  6580  6585  6595  6605 -- quadratic term
     D(         6205  6280  6290  6395  6400 -- standard error
     E5         6440 -- a value used for scaled comparisons (= 10)
     E8         6440  6445  6465  6515  6570  6575  6580  6585  6635
                6640 -- stepsize
     E9         6425  6440 -- the machine precision
     F          6490  6505  6510  6540  6555  6560  6885 -- function
                value
     F0         6170  6490  6540  6575  6580  6585  6710  6715  6835
                -- value of the function at the supposed minimum
```

```
     F1         6835  6885 -- lower function value than F0 if found, else
                 a higher value to prevent a restart in the DRIVER
     F3        6510  6570  6575  6585  6655  6660 -- function value minus
                 stepsize from supposed minimum
     F4         6560  6570  6575  6580  6655  6660 -- function value plus
                 stepsize from supposed minimum
     F9         6160  6170  6175  6180  6280  6305  6310 -- loss function
                 per degree of freedom (sigma^2)
     G(          6395  6400 -- the gradient at X()
     I3         6185  6195  6475  6485  6525  6535 -- flag for computable
                 function (=1 for failure)
     I5          6155  6170 -- the number of masked (fixed) parameters
     I6          6430  6500  6550  6580  6585 -- flag for active bounds
                 (-2=lower bound, -1=upper bound, 0=inactive)
     J           6140  6145  6150  6200  6205  6210  6255  6260  6265
                 6285  6290  6305  6310  6315  6340  6390  6395  6400
                 6405  6435  6445  6450  6455  6460  6465  6470  6515
                 6520  6565  6610  6615  6620  6665  6675  6685  6695
                 6760  6765  6770  6775  6780  6785  6800  6805  6810
                 6820  6825  6830  6870 -- a loop control counter
     K           6870  6875  6880 -- a loop control counter
     K9         6240  6245  6250  6270  6280  6285  6290  6355 -- a loop
                 control counter
     M           6155  6170 -- the number of data points
     N           6155  6170  6200  6240  6255  6390  6435  6760  6820
                 -- number of parameters in loss function
     N2          6140 -- number of elements in upper triangular matrix
     O(         6460  6470  6520  6765 -- bound/mask information storage
     T(          6260  6270  6280  6305  6310  6610  6800  6805 -- tilts
     T1          6290  6295  6305  6310 -- denominator for parameter
                 correlation estimate
     T5          6425  6505  6555 -- a "large" number assigned to the
               function value when constraints are violated. If T5 is too
                large, the calculation of dispersion estimates gives rise
                 to "overflows".
     X$          6410 -- temporary variable for pause
     X(          6445  6465  6515  6565  6665  6780  6785  6825  6875
                 -- the vector of final parameter estimates

=======================================================================
     LINES: 180      BYTES: 6289     SYMBOLS: 52     REFERENCES: 222
```

Listing 13-7-2. Output of POSTMRT.BAS for the Hobbs problem

```
     POST-SOLUTION ANALYSIS FOR MARQUARDT-NASH NLLS
     SIGMA^2 = LOSS FUNCTION PER DEGREE OF FREEDOM = .2874743

     PARAMETER CORRELATION ESTIMATES
     ROW  1  :   1.00000
     ROW  2  :   0.72030  1.00000
     ROW  3  :  -0.93651 -0.43731  1.00000
```

```
SOLUTION WITH ERROR MEASURES AND GRADIENT OF SUM OF SQUARES

B( 1 )= 1.961851   STD ERR= .1130657   GRAD( 1 )=-3.011704E-03
B( 2 )= 4.90915   STD ERR= .1688401   GRAD( 2 )= 6.756783E-04
B( 3 )= 3.135703   STD ERR= 6.863148E-02   GRAD( 3 )=-3.582001E-03

B( 1 )= 1.961851E+00  step= 6.79E-03  f-, f+  2.8787E+00  2.8786E+00
B( 2 )= 4.909150E+00  step= 1.70E-02  f-, f+  2.7119E+00  2.7108E+00
B( 3 )= 3.135703E+00  step= 1.08E-02  f-, f+  3.7843E+00  3.7908E+00
        best function value found is             2.5873E+00

radii of curvature for surface along axial directions
  & tilt angle in degrees

for B( 1 )  R. OF CURV. =  1.580E-04   tilt =    0.37142
for B( 2 )  R. OF CURV. =  2.322E-03   tilt =    1.88952
for B( 3 )  R. OF CURV. =  1.114E-04   tilt =  -16.69814

RESIDUALS
 1.188707E-02 -3.276968E-02  9.201622E-02  .2087689  .3926239
-5.760002E-02 -1.105726  .7157936 -.1076355 -.3483925
 .6525726 -.2876206
```

13-8. POSTVM - a Hessian Dispersion Estimate

Equation (13-4-7) gives another measure for the parameter dispersions. In Listing 13-8-1, we present code to allow this measure to be used if VM has been employed to minimize the loss function for the parameter estimates. Note that code has been added to VM to save the latest inverse Hessian approximation, since our strategy of terminating with a steepest descent search will reset the inverse Hessian to a unit matrix. Listing 13-8-2 presents the output of POSTVM when applied to the Hobbs problem. This may be thought of as a continuation of Listing 10-2-1.

Listing 13-8-1. The POSTVM Code.

```
              POSTVM.BAS          08-07-1986   13:38:33

6000 PRINT "POST-SOLUTION ANALYSIS FOR VARIABLE METRIC"
6005 PRINT #3,"POST-SOLUTION ANALYSIS FOR VARIABLE METRIC"
6010 REM POSTVM.BAS 860405
6015 REM CALLS
6020 REM     FUNCTION F  -- line 2000
6025 REM     RESIDUAL R1 -- line 3500 (only if J6=1)
6030 REM
6035 REM INPUTS TO THE ROUTINE
6040 REM    X() -- the vector of final parameter estimates
6045 REM    N   -- the number of parameters in the function F(B)
6050 REM    M   -- the number of data points (0 for a .FN)
6055 REM    O( , ) -- masks and bounds
6060 REM    Y( , ) -- the data array for the problem
6065 REM    A( , ) -- the Hessian inverse approximation (from VM)
6070 REM    G() -- the gradient at X() (from VM)
6075 REM    I5  -- the number of masked (fixed) parameters
6080 REM    J6  -- flag which is 1 if residuals can be computed
6085 REM    F0  -- value of the function at the supposed minimum
6090 REM
6095 REM    B9, E5, E9 -- constants provided by ENVRON
6100 REM
6105 REM OUTPUT FROM THE ROUTINE
6110 REM    F1 -- lower function value than F0 (if found, else a
6115 REM          higher value to prevent a restart in the DRIVER)
6120 REM    X() -- new parameter values if a lower point is found
6125 REM
6130 REM need array T(N) to store tilt angles
6135 IF M>N-I5 THEN 6150
6140 LET F9=-1
6145 GOTO 6155
6150 LET F9=F0/(M-N+I5)
6155 PRINT "LOSS FUNCTION PER DATA POINT (SIGMA^2) =";F9
6160 PRINT #3,"LOSS FUNCTION PER DATA POINT (SIGMA^2) =";F9
6165 PRINT
6170 PRINT #3,
6175 PRINT "SOLUTION, DISPERSION MEASURES & GRADIENT OF LOSS FUNCTION"
6180 PRINT #3,"SOLUTION, DISPERSION MEASURES & GRADIENT OF LOSS FUNCTION"
6185 PRINT
6190 PRINT #3,
6195 FOR J=1 TO N: REM includes masked parameters
6200 LET T(J)=SQR(A(J,J)*F9)
6205 PRINT "B(";J;")=";B(J);" DISPERSION =";T(J);"  GRAD(";J;")=";G(J)
6210 PRINT #3,"B(";J;")=";B(J);" DISPERSION = ";T(J);" GRAD(";J;")=";G(J)
6215 NEXT J
6220 PRINT
6225 PRINT #3,
6230 LET T5=1/E9: REM used for active bounds ('large' F(B))
6235 LET I6=0: REM flag for active bounds (-2=l.b.,-1=u.b.,0=inactive)
6240 FOR J=1 TO N
6245 LET E8=E5*SQR(E9): REM now compute points left and right
6250 LET E8=E8*(ABS(X(J))+E8): REM stepsize
6255 PRINT "B(";J;")=";
6260 PRINT #3,"B(";J;")=";
6265 IF O(J,3)=0 THEN 6480: REM masked
6270 LET B(J)=X(J)-E8
6275 IF B(J)<O(J,1) THEN 6305
6280 LET I3=0
6285 GOSUB 2000: REM function value
6290 IF I3=1 THEN 6310
6295 IF F<F0 THEN 6650: REM lower function value
6300 GOTO 6315
6305 LET I6=-2: REM active lower bound
6310 LET F=T5: REM failure
6315 LET F3=F
6320 LET B(J)=X(J)+E8
6325 IF B(J)>O(J,2) THEN 6355
6330 LET I3=0
6335 GOSUB 2000: REM function value
6340 IF I3=1 THEN 6360
6345 IF F<F0 THEN 6650: REM lower function value
6350 GOTO 6365
6355 LET I6=-1: REM active upper bound
6360 LET F=T5: REM failure
6365 LET F4=F
6370 LET B(J)=X(J): REM reset parameter value
6375 LET C1=.5*(F4-F3)/E8: REM linear term -- should be zero
6380 LET C2=(F4+F3-2*F0)/(2*E8*E8): REM quadratic term
6385 IF I6=-2 THEN LET C2=(F4-F0)/(E8*E8): REM symmetrized curvature
6390 IF I6=-1 THEN LET C2=(F3-F0)/(E8*E8): REM when bounds active
6395 LET C0=1+C1*C1: REM denominator for curvature
6400 LET C2=C2/(C0*SQR(C0))
6405 LET C0=B9: REM set large in case of singularity
6410 IF C2<>0 THEN LET C0=1/C2: REM radius of curvature for problem
6415 LET T(J)=-45*ATN(C1)/ATN(1): REM tilt -- DIMension T in HJ, NM
6420 PRINT USING "##.######^^^^";B(J);: REM !!
6425 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6430 PRINT "  step=";
6435 PRINT #3,"  step=";
6440 PRINT USING "##.##^^^^";E8;: REM !!
6445 PRINT #3,USING "##.##^^^^";E8;: REM !!
6450 PRINT "  f-, f+";
6455 PRINT #3,"  f-, f+";
6460 PRINT USING " ##.####^^^^";F3;F4: REM !!
6465 PRINT #3,USING " ##.####^^^^";F3;F4: REM !!
6470 LET X(J)=C0: REM save radius of curvature in x()
6475 GOTO 6500
```

```
6480 PRINT USING "##.######^^^^";B(J);: REM !!
6485 PRINT " MASKED"
6490 PRINT #3,USING "##.######^^^^";B(J);: REM !!
6495 PRINT #3," MASKED"
6500 NEXT J
6505 PRINT "        best function value found is            ";
6510 PRINT #3,"        best function value found is            ";
6515 PRINT USING " ##.####^^^^";F0: REM !!
6520 PRINT #3,USING " ##.####^^^^";F0: REM !!
6525 PRINT
6530 PRINT #3,
6535 PRINT "radii of curvature for surface along axial directions"
6540 PRINT #3,"radii of curvature for surface along axial directions"
6545 PRINT "  & tilt angle in degrees"
6550 PRINT #3,"  & tilt angle in degrees"
6555 PRINT
6560 PRINT #3,
6565 FOR J=1 TO N
6570 IF O(J,3)=0 THEN 6615
6575 PRINT "for B(";J;")  R. OF CURV. = ";
6580 PRINT #3,"for B(";J;")  R. OF CURV. = ";
6585 PRINT USING "##.###^^^^";X(J);: REM !!
6590 PRINT #3,USING "##.###^^^^";X(J);: REM !!
6595 PRINT "   tilt = ";
6600 PRINT #3,"   tilt = ";
6605 PRINT USING "####.#####";T(J): REM !!
6610 PRINT #3,USING "####.#####";T(J): REM !!
6615 NEXT J
6620 IF J6=1 THEN GOSUB 6900: REM print residuals
6625 FOR J=1 TO N
6630 LET X(J)=B(J): REM reset parameter for restart
6635 NEXT J
6640 LET F1=F0+ABS(F0)+1: REM to ensure no restart
6645 RETURN: REM to DRIVER
6650 PRINT
6655 PRINT #3 ,
6660 PRINT "lower function value found"
6665 PRINT #3,"lower function value found"
6670 IF J6=1 THEN GOSUB 6900: REM print residuals
6675 FOR K=1 TO J
6680 LET X(K)=B(K): REM reset parameter values -- note, j included
6685 NEXT K
6690 LET F1=F: REM to indicate lower function value
6695 RETURN

     Line no.    Referenced in line(s)
      2000      6285  6335 -- the function subroutine
      6150      6135
      6155      6145
      6305      6275
```

```
 6310      6290
 6315      6300
 6355      6325
 6360      6340
 6365      6350
 6480      6265
 6500      6475
 6615      6570
 6650      6295  6345
 6900      6620  6670 -- residual display

 Symbol     Referenced in line(s)
A(          6200 -- the Hessian inverse approximation
B(          6205  6210  6270  6275  6320  6325  6370  6420  6425
            6480  6490  6630  6680 -- the parameter vector
B9          6405 -- a "large" number
C0          6395  6400  6405  6410  6470 -- denominator for curvature
C1          6375  6395  6415 -- linear term
C2          6380  6385  6390  6400  6410 -- quadratic term
E5          6245 -- a value used for scaled comparisons (= 10)
E8          6245  6250  6270  6320  6375  6380  6385  6390  6440
            6445 -- stepsize
E9          6230  6245 -- the machine precision
F           6295  6310  6315  6345  6360  6365  6690 -- function
            value
F0          6150  6295  6345  6380  6385  6390  6515  6520  6640
            -- value of the function at the supposed minimum
F1          6640  6690 -- lower function value than F0 if found, else
            a higher value to prevent a restart in the DRIVER
F3          6315  6375  6380  6390  6460  6465 -- function value minus
            stepsize from supposed minimum
F4          6365  6375  6380  6385  6460  6465 -- function value plus
            stepsize from supposed minimum
F9          6140  6150  6155  6160  6200 -- loss function per degree
            of freedom (sigma^2)
G(          6205  6210 -- the gradient at X()
I3          6280  6290  6330  6340 -- flag for computable function
            (=1 for failure)
I5          6135  6150 -- the number of masked (fixed) parameters
I6          6235  6305  6355  6385  6390 -- flag for active bounds
            (-2=lower bound, -1=upper bound, 0=inactive)
J           6195  6200  6205  6210  6215  6240  6250  6255  6260
            6265  6270  6275  6320  6325  6370  6415  6420  6425
            6470  6480  6490  6500  6565  6570  6575  6580  6585
            6590  6605  6610  6615  6625  6630  6635  6675 -- a loop
            control counter
J6          6620  6670 -- flag which is 1 if residuals can be computed
K           6675  6680  6685 -- a loop control counter
M           6135  6150 -- the number of data points
N           6135  6150  6195  6240  6565  6625 -- number of parameters
```

```
                in loss function
O(              6265  6275  6325  6570 -- bound/mask information storage
T(              6200  6205  6210  6415  6605  6610 -- tilt angles
T5              6230  6310  6360 -- a "large" number assigned to the
             function value when constraints are violated. If T5 is too
             large, the calculation of dispersion estimates gives rise
                to "overflows".
X(              6250  6270  6320  6370  6470  6585  6590  6630  6680
                -- the vector of final parameter estimates
================================================================
LINES: 141      BYTES: 5206     SYMBOLS: 42     REFERENCES: 176
```

Listing 13-8-2. Output of POSTVM.BAS for the Hobbs problem

```
POST-SOLUTION ANALYSIS FOR VARIABLE METRIC
LOSS FUNCTION PER DATA POINT (SIGMA^2) = .2874733

SOLUTION WITH DISPERSION MEASURES AND GRADIENT OF LOSS FUNCTION

B( 1 )= 1.961855  DISPERSION =  .1746728   GRAD( 1 )=-.2832031
B( 2 )= 4.909165  DISPERSION =  .2768867   GRAD( 2 )= 7.261706E-02
B( 3 )= 3.135686  DISPERSION =  .1116119   GRAD( 3 )=-.3602448

B( 1 )= 1.961855E+00  step= 6.79E-03  f-, f+  2.8805E+00  2.8767E+00
B( 2 )= 4.909165E+00  step= 1.70E-02  f-, f+  2.7107E+00  2.7120E+00
B( 3 )= 3.135686E+00  step= 1.08E-02  f-, f+  3.7880E+00  3.7869E+00
        best function value found is            2.5873E+00

radii of curvature for surface along axial directions
  & tilt angle in degrees

for B( 1 )  R. OF CURV. =  1.774E-04   tilt =   15.80547
for B( 2 )  R. OF CURV. =  2.323E-03   tilt =   -2.17667
for B( 3 )  R. OF CURV. =  9.828E-05   tilt =    3.00071

RESIDUALS
 1.187134E-02 -3.280115E-02  9.195804E-02  .2086706  .3924713
-5.782891E-02 -1.106058  .7153359 -.1082306 -.3491364
 .6516876 -.2885971
```

13-9. RESSAVE - Display of Residuals

For the majority of nonlinear parameter estimation problems, the loss function will be defined in terms of residuals. The problem file will have a filename extension '.RES' in such cases (see Section 15-3). An important step in analyzing the results of parameter estimation, whether linear or nonlinear, is to examine the values or plots of the residuals. Various tools have been devised by statisticians to do this (Belsley, Kuh and Welsch, 1980). To make the data available for such tools, and for our own simple plotting program PLOT.BAS, we provide the program code RESSAVE.BAS which is listed below. This also displays the residuals on the screen as part of the post-solution analysis. RESSAVE is automatically consolidated into the parameter estimation program if the problem file has the filename extension '.RES', but is omitted if the extension is '.FN', since residuals may not be defined in the latter case. RESSAVE creates files of data suitable for PLOT.BAS. The structure of these files, which have filename extension '.PLD', is described in Figure 15-7-1.

Listing 13-9-1. The RESSAVE Code.

```
              RESSAVE.BAS       08-14-1986   21:59:44

6900 PRINT
6902 PRINT #3,
6904 PRINT "RESIDUALS"
6906 PRINT #3,"RESIDUALS"
6908 FOR I=1 TO M
6910 GOSUB 3500: REM no check for computable point
6912 PRINT R1;
6914 PRINT #3,R1;
6916 IF 5*INT(I/5)=I THEN PRINT
6918 IF 5*INT(I/5)=I THEN PRINT #3,
6920 NEXT I
6922 PRINT
6924 PRINT #3,
6926 INPUT "FILENAME TO SAVE DATA ([CR]=NO SAVE):";F$
6928 PRINT #3,"FILENAME TO SAVE DATA ([CR]=NO SAVE):";F$
6930 IF F$="" THEN RETURN
6932 IF INSTR(F$,".")=0 THEN LET F$=F$+".PLD": REM !!
6934 OPEN F$ FOR OUTPUT AS #2: REM !!
6936 INPUT "TITLE:";T$
6938 LET T$=T$+"   "+TIME$+" "+DATE$: REM !! add time and date stamp
6940 PRINT #2,T$
```

```
6942 PRINT "VARIABLES STORED IN MATRIX Y MAY BE IDENTIFIED BY COLUMN
            NUMBER"
6944 PRINT "RESIDUALS ARE IDENTIFIED BY A 0 INDEX (VERTICAL AXIS ONLY)
6946 PRINT "A 'TIME' SERIES 1,2,3,... IS IDENTIFIED BY 0 (HORIZONTAL AXIS)
6948 INPUT "IDENTIFY HORIZONTAL AXIS VARIABLE (INDEX OF Y(M,?) OR 0):";T1
6950 INPUT "HORIZONTAL AXIS VARIABLE NAME:";T$
6952 PRINT #2,T$
6954 INPUT "IDENTIFY VERTICAL AXIS VARIABLE (INDEX OF Y(M,?) OR 0):";T2
6956 INPUT "VERTICAL AXIS VARIABLE NAME:";T$
6958 PRINT #2,T$
6960 FOR I=1 TO M
6962 IF T2=0 THEN 6974
6964 IF T1=0 THEN 6970
6966 PRINT #2,Y(I,T1),Y(I,T2)
6968 GOTO 6984
6970 PRINT #2,I,Y(I,T2)
6972 GOTO 6984
6974 GOSUB 3500: REM determine residual
6976 IF T1=0 THEN 6982
6978 PRINT #2,Y(I,T1),R1
6980 GOTO 6984
6982 PRINT #2,I,R1
6984 NEXT I
6986 CLOSE #2
6988 GOTO 6922

     Line no.    Referenced in line(s)
      3500      6910  6974 -- residual subroutine
      6922      6988
      6970      6964
      6974      6962
      6982      6976
      6984      6968  6972  6980

      Symbol     Referenced in line(s)
     DATE$      6938 -- date stamp (machine function)
     F$         6926  6928  6930  6932  6934 -- name of file
     I          6908  6916  6918  6920  6960  6966  6970  6978  6982
                6984 -- a loop control counter
     M          6908  6960 -- the number of data points
     R1         6912  6914  6978  6982 -- the value of the residual
     T$         6936  6938  6940  6950  6952  6956  6958 -- temporary
                string variable for file information
     T1         6948  6964  6966  6976  6978 -- temporary variable for
                file information
     T2         6954  6962  6966  6970 -- temporary variable for file
                information
     TIME$      6938 -- time stamp (machine function)
     Y(         6966  6970  6978 -- the data array for the problem
=======================================================================
     LINES: 45      BYTES: 1384     SYMBOLS: 16     REFERENCES: 51
```

COVER SHEET

Chapter 14

Chapter title: Difficult Problems

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

14

DIFFICULT PROBLEMS

14-0. Some Difficult Problems

In Section 13-1 we considered why nonlinear parameter estimation problems are difficult _in general_. In this chapter, we look at some especially difficult problems -- nonlinear parameter estimation tasks which are particularly hard to solve.

14-1. Mixtures of Exponentials

One of the most common nonlinear estimation problems is also one of the most difficult. This is the multiple exponential problem defined by the residual function

$$r(i,\underline{B}) = \sum_{j=1}^{n/2} B(2j-1)\exp(B(2j) * Y_{i2}) - Y_{i1} \qquad (14\text{-}1\text{-}1)$$

The problem arises frequently in situations where we desire to determine the rate constants of two or more simultaneous processes which cause the production or disappearance of some measurable property. Examples are

- chemical reactions (where two or more mechanisms may account for the disappearance of a reagent;
- radioactive decay, where the production of measurable radiation may occur by spontaneous breakup of isotopes having different half-lives;
- flows into and out of an underground aquifer.

This problem has several features which cause trouble.

First, many combinations of parameters will give similar values of the residual (14-1-1). Clearly, because each term in the sum in (14-1-1) has the same form, the residual will have the same value for any permutation of these terms.

However, even sets of terms which are not equivalent may yield similar calculated values of the residual. For example, using our codes it is relatively easy to solve the following linear fitting problem:

Find the parameters B(j), j=1,2,...,n which minimize the sum of squared residuals

$$r(i,\underline{B}) = \sum_{j=1}^{n} B(j) \exp(-i * 2^j) - y_{i1} \qquad (14\text{-}1\text{-}2)$$

for i = 1,2,...,10 where

$$y_{i1} = \exp(-i) + 5 \exp(-5i) \qquad (14\text{-}1\text{-}3)$$

Table 14-1-1 gives the results of such a fitting exercise. Note that the sum of squares is explained by only a few of the residual elements. If data does not exist at these points (i.e. these values of i) which dominate the sum of squares, we could easily obtain results which deviate markedly from the "true" values.

Because of its practical importance, the multiple

exponential problem has received a lot of attention over the years (Lanczos, 1956; Osborne, 1975; Holt and Antill, 1977; Ruhe and Wedin, 1980; Ruhe, 1980). Here we will present two quite simple approaches. The first, which is coded in Listing 14-1-1, attempts to implement Equation (14-1-1) directly, and is called LANCZOS.RES. The second works only with the nonlinear (i.e. exponential) parameters in this equation. The linear coefficients are found by a linear regression calculation inside the function evaluation. The code for this problem is called LANCLIN.FN and is presented in Listing 14-1-2. Note that in this listing the linear regression is performed via the normal equations and the Gauss-Jordan inversion, which have been chosen solely to keep the code short. Better methods exist for solving the linear regression problem (Nash, 1979, Chapters 3, 4 and 5). Furthermore, the derivatives of the function with respect to the exponential parameters are messy to compute (see Section 14-5 and references therein), so we use numerical approximation of the gradient.

Some results from the VM and MRT methods applied to these problem files are given in Table 14-1-2. Here the test problem is generated by calculating a sum of known exponential factors with known linear coefficients, then rounding these to a specified number of decimal places (Lanczos, 1956, page 279). In all cases we noted quite slow convergence. Worse, there appear to be alternate sets of parameters which minimize the loss function, and even some reasonable approximations to the generated data in fewer exponentials than were used to create the problem. Where possible, we recommend that users NOT use our methods to solve problems such as these, where there is such an inherent instability of the parameters.

Table 14-1-1. Fits of sums of fixed exponential functions to a particular sum of two exponential functions

```
fit  1*exp(-i)+5*exp(-5*i) to
sum (0,1,..,n-1) of b(j)*exp(-(2^j) * i)  for i=1,2,...,10

number of fitting terms =  1
SUM OF SQUARES= 1.570065E-04,  SIGMA^2= 1.744516E-05
B( 1 ) =  1.079336   STD ERR =  1.055737E-02   GRAD( 1 ) = -1.575907E-08
RESIDUALS
-4.503727E-03  1.050992E-02  3.948365E-03  1.453076E-03  5.345606E-04
 2.675407E-03  9.842267E-04  3.620769E-04  1.332007E-04  4.900177E-05

number of fitting terms =  2
SUM OF SQUARES= 8.983323E-06, SIGMA^2= 1.122915E-06
B( 1 ) =  .9574934   STD ERR =  1.094476E-02   GRAD( 1 ) = -6.988816E-09
B( 2 ) =  .3639704   STD ERR =  3.170027E-02   GRAD( 2 ) =  7.791571E-09
RESIDUALS
-6.901846E-05  6.86707E-04  -1.215617E-03  -6.564481E-04  -2.698826E-04
 2.375626E-03  8.734235E-04  3.212443E-04  1.181697E-04  4.347088E-05

number of fitting terms =  3
SUM OF SQUARES= 7.082429E-06, SIGMA^2= 1.011775E-06
B( 1 ) =  .9953701     STD ERR =  2.951649E-02 GRAD( 1 ) = -8.179242E-09
B( 2 ) =  1.480138E-02 STD ERR =  .2564747     GRAD( 2 ) = -2.641008E-09
B( 3 ) =  1.822997     STD ERR =  1.329834     GRAD( 3 ) = -3.344345E-10
RESIDUALS
-4.656613E-07  2.906087E-05  -1.841473E-04  -7.96395E-05  -3.051984E-05
 2.467368E-03  9.076722E-04  3.339112E-04  1.228387E-04  4.518976E-05

number of fitting terms =  4
SUM OF SQUARES= 6.816415E-06, SIGMA^2= 1.136069E-06
B( 1 ) =  .9571055   STD ERR =  .0730019   GRAD( 1 ) =  2.24537E-08
B( 2 ) =  .9793414   STD ERR =  1.689179   GRAD( 2 ) =  8.077948E-09
B( 3 ) = -36.06955   STD ERR =  65.59652   GRAD( 3 ) =  1.091344E-09
B( 4 ) =  1721.703   STD ERR =  2980.376   GRAD( 4 ) =  1.98732E-11
RESIDUALS
 0  -1.112116E-06  6.887024E-05  -4.611773E-04  -2.446329E-04
 2.378444E-03  8.735814E-04  3.211834E-04  1.181312E-04  4.345454E-05

number of fitting terms =  5
SUM OF SQUARES= 6.816415E-06, SIGMA^2= 1.363283E-06
B( 1 ) =  .9570874   STD ERR = -9999   GRAD( 1 ) =  1.296875E-09
B( 2 ) =  .9797934   STD ERR = -9999   GRAD( 2 ) =  4.94307E-10
B( 3 ) = -36.0872    STD ERR = -9999   GRAD( 3 ) =  7.024854E-11
B( 4 ) =  1722.504   STD ERR = -9999   GRAD( 4 ) =  1.250441E-12
B( 5 ) =  .5660512   STD ERR = -9999   GRAD( 5 ) =  4.609342E-16
RESIDUALS
 4.096037E-09  -1.121647E-06  6.898071E-05  -4.613592E-04  -2.447349E-04
 2.378402E-03  8.735653E-04  3.211774E-04  1.181289E-04  4.345372E-05
```

Table 14-1-2. The Lanczos multiple exponential problem

a. generated problem, data rounded to 10 decimal places (full precision)

result of	function value	B(1)	B(2)	B(3)	B(4)	B(5)	B(6)	Performance
		parameters						
initial point (a)	2.058388E-06	.0951	1	.8607	3	1.5576	5	
MRT + LANCZOS	2.357684E-14	.095040	.999701	.860512	2.99959	1.55695	4.99985	C54/12/17
VM + LANCZOS	9.929036E-12	.095103	1.00001	.860444	2.99981	1.55695	4.99975	M22/9/15

b. generated problem, data rounded to 2 decimal places (Lanczos problem)

(i) from Lanczos start, initial point (a)

result of	function value	B(1)	B(2)	B(3)	B(4)	B(5)	B(6)	Performance
		parameters						
initial point (a)	1.377043E-04	.0951	1	.8607	3	1.5576	5	
VM + LANCZOS	1.071565E-04	.095135	.976958	.857374	3.04174	1.55644	4.93946	M49/20/30
MRT + LANCZOS	1.046967E-04	.328311	1.65265	2.16720	4.45669	.014141	U 20	C833/184/259

(ii) from a "typical" initial point

result of	function value	B(1)	B(2)	B(3)	B(4)	B(5)	B(6)	Performance
		parameters						
initial point (b)	12.08321	1	1	1	2	1	3	
VM + LANCZOS	1.114132E-04	.305933	1.65455	.392956	3.40809	1.80959	4.70645	M98/39/67
MRT + LANCZOS	1.046983E-04	2.16720	4.45669	.328311	1.65265	.014141	U 20	C758/168/234
initial point (c)	6.777668E-03	*	1	*	2	*	3	
VM + LANCLIN	1.083339E-04<	$\underline{.119081}$	1.14765	$\underline{.662266}$	2.91525	$\underline{1.72733}$	4.80981	M166/32/42
initial point (d)	.367506	*	1	*	2	-	-	
VM + LANCLIN	1.136305E-04	$\underline{.411118}$	1.81417	$\underline{2.09829}$	4.57487	-	-	M88/23/50
initial point (e)	5.384428	1	1	1	2	-	-	
VM + LANCZOS	1.136179E-04	.403236	1.80885	2.10510	4.57195	-	-	M43/28/33

Notes: * or underlined parameter is calculated from linear regression
< implies the function value is not optimal and POSTGEN has found a lower value
Performance is given as
(Machine)(time in seconds)/(gradient evaluations)/(function evaluations)
where Machine = C for Corona PC 21, M for Maestro PC (V20 processor)

Listing 14-1-1. LANCZOS.RES problem file.

```
30 DIM O(10,3),B(10),X(10),Y(100,2)
3000 PRINT "LANCZOS.RES -- multiple exponentials -- 851128"
3010 PRINT #3,"LANCZOS.RES -- multiple exponentials -- 851128"
3020 LET P$="LANCZOS.RES -- multiple exponentials -- 851128"
3030 PRINT " *** note that this is a very difficult problem *** "
3040 PRINT #3," *** note that this is a very difficult problem *** "
3050 INPUT "number of observations (e.g. 24)=";M
3060 PRINT #3,"number of observations (e.g. 24)=";M
3070 INPUT "number of exponential terms=";N1
3080 PRINT #3,"number of exponential terms=";N1
3090 LET N=2*N1
3100 PRINT "model is sum j=1,2,...,";N/2
3110 PRINT
3120 PRINT "  B(2*j-1) * exp( - B(2*j) * Y(i,1))"
3130 PRINT
3140 PRINT "   for i=1,2,...,";M
3150 PRINT #3, "model is sum j=1,2,...,";N/2
3160 PRINT #3,
3170 PRINT #3, "  B(2*j-1) * exp( - B(2*j) * Y(i,1))"
3180 PRINT #3,
3190 PRINT #3, "   for i=1,2,...,";M
3200 INPUT "round to how many decimals ";L2
3210 PRINT #3,"round to how many decimals ";L2
3220 LET Z2=1
3225 IF L2>15 THEN 3260
3230 FOR L3=1 TO L2
3240 LET Z2=Z2*10
3250 NEXT L3
3260 PRINT "Lanczos data"
3265 PRINT "  x","  y" Y"
3270 PRINT #3,"Lanczos data"
3275 PRINT #3,"  x","  y"
3280 FOR I=1 TO M
3290 LET Z1=.05*(I-1)
3300 LET Y(I,2)=.0951*EXP(-Z1)+.8607*EXP(-3*Z1)+1.5567*EXP(-5*Z1)
3310 LET Y(I,2)=INT(Y(I,2)*Z2+.5)/Z2
3320 LET Y(I,1)=Z1
3330 PRINT Z1,Y(I,2)
3340 PRINT #3,Z1,Y(I,2)
3350 NEXT I
3360 FOR J=1 TO N/2
3370 LET B(2*J-1)=1
3380 LET B(2*J)=J: REM note starting values
3390 LET O(2*J-1,1)=0
3400 LET O(2*J-1,2)=1000
3410 LET O(2*J-1,3)=1
3420 LET O(2*J,1)=0
3430 LET O(2*J,2)=20
```

```
3440 LET O(2*J,3)=1
3450 NEXT J: REM bounds on linear and exponential coefficients
3460 RETURN
3500 LET R1=-Y(I,2)
3505 LET Z1=Y(I,1)
3510 FOR L=1 TO N/2
3520 LET R1=R1+B(2*L-1)*EXP(-B(2*L)*Z1)
3530 NEXT L
3540 RETURN
4000 LET Z1=Y(I,1)
4010 FOR L=1 TO N/2
4020 LET D(2*L-1)=EXP(-B(2*L)*Z1)
4030 LET D(2*L)=-D(2*L-1)*Z1*B(2*L-1)
4040 NEXT L
4050 RETURN
```

Listing 14-1-2. LANCLIN.FN problem file.

```
30 DIM O(10,3),B(10),X(10),Y(100,2),Z(110,11)
2000 LET I3=0: REM lanclin.fn -- removes linear parameters
2015 REM build data matrix
2020 FOR L1=1 TO M
2025 FOR L2=1 TO N
2030 LET Z(L1+N,L2)=EXP(-B(L2)*Y(L1,1)): REM will change with fns
2035 NEXT L2
2040 LET Z(L1+N,N+1)=Y(L1,2): REM will change with situation
2045 NEXT L1
2050 REM now compute normal equations
2055 FOR L1=1 TO N
2060 FOR L2=1 TO L1
2065 LET Z0=0
2070 FOR L3=1 TO M
2075 LET Z0=Z0+Z(L3+N,L1)*Z(L3+N,L2)
2080 NEXT L3
2085 LET Z(L1,L2)=Z0
2090 LET Z(L2,L1)=Z0
2095 NEXT L2
2100 NEXT L1
2110 FOR L1=1 TO N
2115 LET Z0=0
2120 FOR L3=1 TO M
2125 LET Z0=Z0+Z(N+L3,L1)*Z(N+L3,N+1)
2130 NEXT L3
2135 LET Z(L1,N+1)=Z0
2140 NEXT L1: REM done accumulation of normal equations
2150 FOR L1=1 TO N
2155 IF Z(L1,L1)=0 THEN 2470
```

```
2160 LET Z0=Z(L1,L1)
2165 FOR L2=L1+1 TO N+1
2170 LET Z(L1,L2)=Z(L1,L2)/Z0
2175 NEXT L2
2180 LET Z(L1,L1)=1
2185 FOR L2=1 TO N
2190 IF L2=L1 THEN 2215: REM skip pivot row
2195 FOR L3=L1+1 TO N+1
2200 LET Z(L2,L3)=Z(L2,L3)-Z(L2,L1)*Z(L1,L3)
2205 NEXT L3
2210 LET Z(L2,L1)=0: REM elimination
2215 NEXT L2
2220 NEXT L1: REM done
2260 LET F=0
2265 FOR L1=1 TO M
2270 LET R1=-Z(L1+N,N+1)
2275 FOR L2=1 TO N
2280 LET R1=R1+Z(L1+N,L2)*Z(L2,N+1)
2285 NEXT L2
2290 LET F=F+R1*R1
2295 NEXT L1
2300 RETURN
2470 PRINT "zero pivot in normal equations matrix"
2475 PRINT #3,"zero pivot in normal equations matrix"
2480 LET I3=1
2490 RETURN
2500 PRINT "Numerical gradient must be used. Program stopped."
2510 PRINT #3,"Numerical gradient must be used. Program stopped."
2520 STOP
3000 PRINT "LANCLIN.FN -- multiple exponentials -- 851128"
3005 INPUT "number of observations (e.g. 24)=";M
3010 INPUT "number of exponential terms=";N
3015 PRINT "no check on positivity of linear coefficients"
3020 PRINT "model is sum j=1,2,...,";N
3025 PRINT
3030 PRINT "  a(j) * exp( - B(j) * Y(i,1))"
3035 PRINT
3040 PRINT "   for i=1,2,...,";M
3045 INPUT "round to how many decimals ";L2
3050 PRINT #3,"LANCLIN.FN -- multiple exponentials -- 851128"
3055 PRINT #3,"number of observations (e.g. 24)=";M
3060 PRINT #3,"number of exponential terms=";N
3065 PRINT #3,"no check on positivity of linear coefficients"
3070 PRINT #3,"model is sum j=1,2,...,";N
3075 PRINT #3,
3080 PRINT #3,"  a(j) * exp( - B(j) * Y(i,1))"
3085 PRINT #3,
3090 PRINT #3,"   for i=1,2,...,";M
3095 PRINT #3,"round to how many decimals ";L2
3100 LET Z2=1
```

```
3105 IF L2>15 THEN 3125
3110 FOR L3=1 TO L2
3115 LET Z2=Z2*10
3120 NEXT L3
3125 PRINT "Lanczos data"
3130 PRINT "  x","  y"
3135 PRINT #3,"Lanczos data"
3140 PRINT #3,"  x","  y"
3145 FOR I=1 TO M
3150 LET Z1=.05*(I-1)
3155 LET Y(I,2)=.0951*EXP(-Z1)+.8607*EXP(-3*Z1)+1.5567*EXP(-5*Z1)
3160 LET Y(I,2)=INT(Y(I,2)*Z2+.5)/Z2
3165 LET Y(I,1)=Z1
3170 PRINT Z1,Y(I,2)
3175 PRINT #3,Z1,Y(I,2)
3180 NEXT I
3185 FOR J=1 TO N
3190 LET O(J,1)=0
3195 LET O(J,2)=1000
3200 LET O(J,3)=1
3205 NEXT J: REM bounds on linear and exponential coefficients
3210 LET I5=0
3215 LET P$="LANCLIN.FN -- multiple exponentials -- 851128"
3220 RETURN
4500 PRINT "linear parameters corresponding to exponential terms"
4505 PRINT #3,"linear parameters corresponding to exponential terms"
4510 FOR L1=1 TO N
4515 PRINT Z(L1,N+1);" ";
4520 PRINT #3,Z(L1,N+1);" ";
4525 IF 5*INT(L1/5)=L1 THEN PRINT
4530 IF 5*INT(L1/5)=L1 THEN PRINT #3,
4535 NEXT L1
4540 PRINT
4545 PRINT #3,
4550 RETURN
```

14-2. Constrained Problems with Awkward Feasible Regions

Problems which are otherwise straightforward may be made difficult because the parameters must obey certain constraints. In certain cases, these constraints may limit the feasible region in such a way that our methods -- which attempt to minimize a loss function -- may be unable to make progress from certain starting points. In other situations, constraints may lead to poor scaling of the loss function, or may result in loss functions which are awkward to compute.

Our examples in this section will be mainly artificial, as real problems often arise in highly technical ways which may obscure our exposition.

As an example of a set of constraints which prevents our methods from finding the solution, consider the minimization of the function

(14-2-1) $f(B) = (B+1)^2$

subject to the pair of constraints

(14-2-2) $B \leq -1$ or $B \geq +2$

Clearly, if we start with an initial point to the right of 2, a minimization method should converge to a "solution" at B = 2, which is not the minimum, even though it is locally minimal. The constraint $B \geq 2$ has introduced a local minimum which is not the desired solution point. Indeed, this type of difficulty is related to that of multiple minima discussed in the next section.

In Section 2-5, we presented an estimation problem in which a linear model was nonlinearly constrained. In practice, we prefer to explicitly solve equality constraints and thereby reduce the dimensionality of the problem. However, it is not always obvious how to proceed. We will use $\underline{q}$ to denote the full set of parameters to avoid confusion with the ultimate set of parameters $\underline{B}$ which arises once all the equality constraints have been removed by substitutions. Consider as an example a model in which we wish to impose a constraint

(14-2-3) $q_1 * q_3 = q_2 * q_4$

(see Nash, 1979, page 184). There is an obvious substitution

(14-2-4) $q_1 = q_2 * q_4 / q_3$

but one can also eliminate any of the four parameters by similar substitutions. Which substitution should be used is

by no means obvious.

A slightly more awkward problem arises if we wish to minimize some function along a path or trajectory. For example, in planning energy consumption for a satellite, the orientation for a satellite will be important for radiative heating and cooling and for solar panel electricity generation. However, the orbit is also important. If we suppose that the minimization is to be performed along the orbit, then the spatial position coordinates x, y and z are related by the functions which describe the orbit. In a highly simplified example, consider the orbit to be a circle of radius R. Then the equation of the orbit is

(14-2-5) $x^2 + y^2 = R^2$

and one of the orbital positions is required as a parameter in the loss function. We cannot immediately solve (14-2-5) for the other parameter because, given x, the sign of y is not determined from Equation (14-2-5). A better approach uses polar coordinates to describe the circular orbit, but this may introduce more complicated expressions into the loss function or its derivatives.

14-3. Multiple Minima

Both the exponential fitting and awkward constraints difficulties are in part due to the existence of more than one local minimum of the loss function which describes the estimation criteria. It is our experience that many nonlinear estimation problems suffer from such extraneous "solutions", but that these cause no more than a nuisance in most cases. In this section we will look at some examples where the multiple minima may hinder the estimation of parameters in some way.

The first possibility is that our minimization techniques converge on a local minimum rather than a global minimum. For certain models, it may be mathematically probable that there is only one minimum -- both local and global -- for the loss function. This information, if available, may be useful. However, the loss function, as computed by the floating-point arithmetic of the machine at hand may represent the function surface (over the domain of the parameters) as a sequence of very shallow steps or plateaus. Algorithms may have difficulty in deciding if these plateaus are not "flat", thereby signifying that a (local) minimum has been found. Gradient methods which use analytic derivative information may be able to use gradient information and avoid premature convergence. Direct search methods or gradient methods using numerical approximation of derivatives are less likely to continue.

A second possibility is that there are several equally good solutions. For example, in the multiple exponentials problems, the parameter pairs

$$B(2j - 1),\ B(2j) \quad \text{for } j = 1,2,\dots n/2$$

are interchangeable in the model function (14-1-1). In fact we will have

$$(n/2)! = (n/2) * (n/2 - 1) * \dots * 2 * 1$$

sets of parameters which will yield identical residuals. Similarly, a model function in which a parameter has been squared to ensure non-negativity will give the same residual for each sign. These cases may give algorithms difficulties, especially if parameter values coalesce in situations such as multiple exponentials.

A third possibility is that there are infinitely many

solutions to the minimization problem. As an example, consider minimizing the function

$$(14\text{-}3\text{-}1) \qquad f(\underline{B}) = (B(1) - B(2))^2$$

This is clearly minimized for B(1) = B(2) and has a value of zero anywhere along the line B(1) = B(2). Note that the gradient is

$$(14\text{-}3\text{-}2) \qquad g(\underline{B}) = 2\ (B(1) - B(2))\ (1,-1)^T$$

and the Hessian is

$$(14\text{-}3\text{-}3) \qquad H(\underline{B}) = 2 \begin{Bmatrix} 1 & -1 \\ -1 & 1 \end{Bmatrix}$$

which is singular. Indeed, Newton's method cannot be used for this problem, and the direct search methods are preferred. Another problem of this type is the variable order test problem NASHHARD given below, for which the Hessian is singular at the solution (see Table 14-3-2). Another problem with a singular Hessian is the Extended Powell Singular problem (POWSING.RES, Section 18-8), for which the results of our methods are given in Table 14-3-1.

Another minimization / parameter estimation problem which gives rise to a singular Hessian is that of minimizing the Rayleigh quotient to find the extreme eigensolutions of a symmetric matrix (Golub and Van Loan, 1983, page 308). The program code EIGV.FN, given in Section 15-3, allows us to compute the eigensolutions of a variety of tridiagonal matrices to illustrate this problem, for which we would caution that our minimization methods are far from optimal. On the other hand, there are occasions when (nonlinear) eigenvalue problems arise, and in such situations the function minimization approach may be useful in obtaining a solution, even if the calculations are inefficient in terms of computer time.

Following Fletcher and Bradbury (1966), but changing notation, we wish to minimize the function

(14-3-4) $f(\underline{B}) = (\underline{B}^T A \underline{B}) / (\underline{B}^T \underline{B})$

where A is a symmetric matrix. In EIGV.FN, we limit our attention to two special classes of tridiagonal matrices, both with super- and sub- diagonal elements which are all of value 1. The diagonal elements are either of constant value (provided by the user), or take on integer values in a pattern to give the Wilkinson (1965) W+ and W- series of matrices (Nash, 1979, p. 210-211). Clearly, in (14-3-4), the function f($\underline{B}$) has the same value if we replace $\underline{B}$ with k*$\underline{B}$, where k is any non-zero constant. This indeterminacy in $\underline{B}$ implies that the Hessian of f($\underline{B}$) is singular. To avoid the indeterminacy, it is common to apply a constraint to the elements of $\underline{B}$, which eventually become the eigenvector corresponding to the smallest eigenvalue. (We can also multiply the expression on the right hand side of (14-3-4) by -1 to find the most positive eigenvalue.) One constraint is to fix one of the parameters at 1, which we could accomplish with a mask. Unfortunately, it is possible for the chosen parameter to be zero in the solution, which could mean that the rest of the parameters would have to grow very large. A more desirable constraint is to have the denominator of (14-3-4) constrained to be 1, which corresponds to the most common normalization of the eigenvector. Our software is poorly adapted to such constraints. A compromise is to bound all the parameters so that growth is limited to some pre-chosen maximum absolute value.

The results of applying the gradient methods CG, TN and VM to the problem of finding the eigensolution of a very simple tridiagonal matrix of order 5 are given in Table 14-3-3. It appears from these examples, and is supported by others that we have tried, that the use of a mask on one parameter gives inferior performance to moderate bounds. In fact, relatively loose bounds appear to result in faster

convergence. Only occasionally do we see large parameter values resulting. Generally, we prefer to use CG and VM for such problems on account of their use of the steepest descent direction whenever thay encounter difficulties. By so doing, they appear to escape some of the false convergence problems experienced in TN due to the singularity of the Hessian. Note, however, that more assiduous attention to detail in designing either the minimization method or the mechanism for constraining the loss function could result in a truncated Newton approach being superior for these eigenvalue problems. Once again, we remind the reader that this example is NOT a desirable way to solve common matrix eigenvalue problems, and that such problems are used as a test of our methods and as a signpost to their application to more general parameter estimation problems of the eigenvalue type.

Overcoming the multiple minima difficulty often requires judicious choice of starting values for iterations. Several authors, among them Bremermann (1970), use pseudo-random numbers to generate a sequence of test points in the parameter space. These may be used in a direct search fashion or as the initial points for descent methods for function minimization, such as those presented in this book. We recommend the use of randomly generated initial points only as a last resort. Wherever possible, good preliminary estimates of parameters should be used as starting points for iterative methods to avoid encountering local minima and to increase the probability of converging to minima which are in accord with our expectations.

Table 14-3-1. Comparison of performance of gradient methods on the Extended Powell Singular Function. Since CG achieves lower function values than TN, data for the CG iteration at which CG first attains a lower function value than TN is listed. All results obtained on the Maestro PC (NEC V20 processor) using Microsoft BASCOM compiler.

Method	Order	Evaluations Grads	Fns	Function Value	Gradient Norm	Time (s)
CG	4	59	148	3.069549E-6	1.377785E-4	...
		78	190	1.186588E-7	2.205398E-5	31
	8	24	61	4.936417E-6	2.751944E-4	...
		65	165	1.441078E-7	2.093374E-5	43
	12	36	90	3.739169E-5	2.573315E-3	...
		74	181	4.350753E-8	1.103766E-5	69
	16	28	72	2.258942E-5	2.624864E-4	...
		56	137	5.672281E-10	1.793615E-5	73
	20	29	74	2.091766E-4	3.487494E-2	...
		63	152	7.313906E-7	5.927326E-5	106
	24	29	74	2.001750E-4	0.023132	...
		75	183	1.236600E-9	6.329030E-5	160
TN	4	89*	41*	5.564290E-6	4.151230E-3	22
	8	28	9	2.739735E-5	2.530778E-3	14
	12	34	8	4.301756E-5	4.831269E-3	23
	16	34	9	5.012974E-5	2.293328E-3	33
	20	21	7	3.638404E-4	1.471556E-2	29
	24	20	7	4.355455E-4	6.094569E-3	37
VM	4	14	21	7.589277E-8	...	
		58	117	1.462422E-17	1.337764E-12	32
	8	17	29	1.126289E-7	...	
		30	46	1.176533E-13	4.211352E-7	35
	12	22	39	4.322589E-08	...	
		37	61	1.094024E-13	4.883697E-6	76
	16	28	51	1.790522E-10	...	
		52	158	1.354156E-13	6.777129E-6	172
	20	40	78	5.708740E-11	4.927131E-6	182
	24	43	81	1.441267E-13	4.991723E-6	261
MRT	4	18	20	3.090550E-15		13
	8	18	20	6.181100E-15		34
	12	18	20	9.271650E-15		71
	16	18	20	1.236220E-14		134
	20	18	20	1.545275E-14		228
	24	18	20	1.854330E-14		358

* exceeded the iteration limit imposed for the program

Table 14-3-2. Results minimizing the NASHHARD test function with gradient minimization methods. All results obtained on the Maestro PC (NEC V20 processor) using Microsoft BASCOM compiler.

Method	Order	Evaluations Grads	Fns	Function Value	Gradient Norm	Time (s)
CG	2	5	10	0	0	2
	10	12	23	1.049846E-9	1.624888E-5	6 <
		18	32	2.134698E-12	7.489684E-7	+4 <
	20	50	115	9.880319E-6	8.092292E-4	49 <
TN	2	12	6	4.035367E-10	1.23095E-4	3 <
		22	9	2.119672E-12	8.89055E-6	+3 <
	10	19	8	1.063265E-5	1.303466E-2	10 <
		27	10	1.876464E-10	4.415456E-5	+4 <
		29	11	1.875446E-10	4.413037E-5	+2 <
	20	27	8	0.1567797	0.9256471	19 <
		120	22	1.198232E-8	1.033852E-3	+50 <
VM	2	9	12	3.065856E-14	2.571826E-7	3 <
	10	16	29	3.276573E-12	1.096896E-5	23 <
	20	26	53	2.666087E-11	3.120621E-5	99 <

< indicates a lower function value was found by POSTGEN
+ in front of a timing indicates additional time to reach the result indicated from the termination point immediately above

14-4. Global Minima

The technique of random starting points is one of the approaches to finding the global minimum of a function (Dixon and Szegö, 1978). Another approach is to use grid search, though clearly the computational effect in such exercises is bound to be high. For certain classes of functions, other techniques may make possible the determination of a set of parameters which yield the global minimum of these functions (for example, Mladineo, 1986)

Table 14-3-3. Results of applying gradient methods to finding the smallest eigenvalue and associated eigenvector of a symmetric tridiagonal matrix of order 5 having diagonal elements equal to 5 and superdiagonal and subdiagonal elements equal to 1. Parameters were bounded to have absolute value less than or equal to 1.

Method	Initial	Mask	Time	Evaluations		Eigenvalue	Largest
	B	B(?)	(s)	Grads	Fns	found	Residual
CG	a	1	98	25	63	3.267949192431123	1.72D-08
	b	1	overflows encountered - method failed				
	a	-	59	13	37	3.267949192431123	8.68D-11
	b	-	63	14	38	3.267949192431123	3.14D-11
TN	a	1	195	77	22	3.267949192431123	1.14D-08
	b	1	10	2	1	5.572010346551897<	
			+10	4	2	5.572010341465459<	
	manually stopped		+ 9	6	3	5.572010336379021<	1.17
	a	-	45	16	5	3.291318779015936<	
			+129	56	37	3.267949192431123	8.10D-10
	b	-	10	2	1	5.572010346551897<	
			+10	4	2	5.572010342671636<	
			+10	6	3	5.572010338791375<	
	manually stopped		+10	8	4	5.572010334911113<	1.17
VM	a	1	136	23	46	3.267949192431123	3.18D-10
	b	1	235	38	105	3.416291843543349	4.22D-01
	a	-	161	26	49	3.267949192431123	1.46D-11
	b	-	165	26	50	3.267949192431123	3.67D-12

a initial B = (1, 0, 0, 0, 0)
b initial B by pseudo random generator, and is approximately (.21510, .18311, -1.78576E-2, .49929, .14651)
< implies a lower function value was found by POSTGEN

We have not found it necessary to be overly concerned with global versus local minima. Most estimation problems which arise in real situations have fairly well-defined conditions for "acceptability" of the parameters. While it may happen that one or more parameters is poorly defined, which implies that the loss function changes very slowly with respect to these parameters or some combination of them, it is relatively rare that a local minimum is close enough to the global minimum in the parameter space to cause us grave

difficulties. That is, while alternative minima may arise, all giving the same value for the loss function, local minima which have unsatisfactory values of the loss function, and which are sufficiently near the global minimum giving the desired parameter estimates, have not arisen in our experience. Dixon and Szegö (1978) present some test problems having local and global minima. Some of these are not dissimilar to parameter estimation loss functions. Therefore, the possibility that a method has converged to a local minimum must be kept in mind. As a strategy for determining whether a solution point is a local minimum, we recommend restarting the minimization some distance away from the convergence point to see if the second attempt yields the same set of parameters. To this end, our DRIVER code is set up to allow for easy restarts. We also avoid exiting from the BASIC interpreter on termination, in part for this reason, in part because the SYSTEM command in Microsoft BASIC is not standard.

The restart strategy we recommend and use is clearly lacking in any form of guarantee of success that the appropriate global minimum will be found. Nevertheless, we are of the opinion that situations where the issue of local versus global minimum are important in nonlinear parameter estimation are relatively uncommon. We would welcome correspondence with readers whose experience includes problems which do involve questions of global minima in order that our methods may be appropriately extended should such matters become more prominent.

14-5. The Variable Projection Method

The multiple exponentials problem of Section 14-1 is but one member of a class of separable nonlinear least squares problems. These have the property that, if some of the parameters are given, the rest are determined as the solution of a linear sub-problem. If we define our residuals as

$$(14\text{-}5\text{-}1)\qquad r_i(\underline{B},\underline{a},y_i,t_i) = \sum_{j=1}^{N} a_j \exp(-B(j) * t_i) - y_i$$

for i = 1, 2, ..., M, then given $\underline{B}$, we can compute $\underline{a}$ as the least squares solution which minimizes the norm of

$$(14\text{-}5\text{-}2)\qquad \underline{r} = X\,\underline{a} - \underline{y}$$

where X is the matrix defined by the elements

$$(14\text{-}5\text{-}3)\qquad X_{ij} = \exp(-B(j) * t_i)$$

To solve the nonlinear least squares problem, we note that the residuals (14-5-2) are the same as those in (14-5-1) if the linear parameters $\underline{a}$ solve the linear least squares sub-problem. This is easily arranged using a generalized inverse of X, which we will label Xminus. This could be the Moore-Penrose inverse discussed in Section 11-7, but other possibilities exist. The solution of the linear least squares problem is

$$(14\text{-}5\text{-}4)\qquad \underline{a} = \underline{a}(\underline{B}) = \text{Xminus}\ \underline{y}$$

where we include the functional dependence of $\underline{a}$ on $\underline{B}$ explicitly to remind us that nonlinear least squares methods need the Jacobian to set up the Gauss-Newton or Marquardt Equations (11-3-11). Substitution of (14-5-4) in (14-5-2) gives the residuals (as functions only of $\underline{B}$) as

$$(14\text{-}5\text{-}4)\qquad \underline{r} = X\ \text{Xminus}\ \underline{y} - \underline{y} = (X\ \text{Xminus} - 1)\ \underline{y}$$

Note that $\underline{y}$ does not depend on $\underline{B}$, and that the dependence on $\underline{B}$ arises only through the matrix

$$(14\text{-}5\text{-}6)\qquad (X\ \text{Xminus} - 1)$$

so that the Jacobian requires derivatives of this matrix product. Such derivatives are relatively troublesome to compute, but the work can be simplified by appropriate choices for the algorithm to generate the variable projection matrix (14-5-6). In this book, we do not propose the include a variable projection method because

- our experience with such methods is limited;
- the various steps involved in computing the projection matrix (14-5-6), either explicitly or implicitly, give rise to programs which are somewhat more complex than those included in the book;
- inclusion of constraints on the linear parameters $\underline{a}$ requires different mechanisms than those so far discussed.

A further consideration for this particular book is that BASIC does not lend itself well to the type of program which the variable projection method requires. The construction of the generalized inverse, the projection matrix (14-5-6), its derivatives and hence the Jacobian, and the solution to the Gauss-Newton-Marquardt Equations (11-3-11) all may be accomplished using similar matrix operations which are most easily invoked as true subroutine calls which allow local variables within the subroutine. Therefore, with some reluctance, we must refer the reader to published algorithms on the variable projection method, namely, Golub and Pereyra (1973), Krogh (1974), and Kaufman (1975). An interesting review was produced by Ruhe and Wedin (1974), unfortunately only published in a technical report as far as we are aware.

COVER SHEET

Chapter 15

Chapter title: Implementation details

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

15

IMPLEMENTATION DETAILS

15-0. Implementation Details

In this chapter we examine some of the decisions we have made in building the software described in this book. These decisions color the use of the software. They concern such matters as

- the relationships between methods
- how problems are converted to program code
- how data is provided to programs
- how users should write program code which interfaces to the software
- tools for putting together the pieces of program code which solve problems.

Most of these topics have already been mentioned at least once, and our examples have illustrated the use of the different codes. However, we shall attempt here to focus on the overall mechanisms used to implement the particular package of methods we have put together. We shall repeat important implementation information so that it is collected into a single chapter.

Section 15-1. Block Structure

The block structure of the program segments in this book is outlined in Figure 15-1-1.

Names which are capitalized are segments which have been included in this book. Code segments such as GENSTEP (Section 7-4), which have been included within other programs, are not explicitly mentioned here. Programs to the left-hand side of the figure call those which are to their right.

Table 15-1-1 gives a listing of the functions of the program code segments.

Table 15-1-2 presents the names of some of the example problem files.

```
DRIVER   | HJ      | ENVRON
         | NM      | function
         |         | SUMSQR      | residual

         | VM      | ENVRON
         | CG      | function
         | TN      | gradient
                   | NUMGRAD
                   | SUMSQR       | residual
                                  | jacobian
                                  | NUMJAC

         | MRT     | ENVRON
         |         | residual
                   | jacobian
                   | NUMJAC

         | setup

         | POSTMRT |
         | POSTGEN |RESSAVE
         | POSTVM  |

         | result-analysis


FGTEST   | setup
         | ENVRON
         | function
         | gradient
         | SUMSQR  | residual
                   | jacobian


RJTEST   | setup
         | ENVRON
         | residual
         | jacobian
```

Figure 15-1-1. Relationship between program code segments. Names in lower case are generic (that is, functional) segments which must be supplied for a particular estimation problem. Program segments call programs listed to their right and below.

Table 15-1-1. Purpose of program code segments presented

I. Named segments

CG	- a conjugate gradients function minimizer (Section 8-1)
DRIVER	- to carry out the setup, solution, reporting and analysis of parameter estimation problems (Section 15-2)
ENVRON	- to determine the properties of the computing environment (Appendix E)
FGTEST	- to compute the loss function at given points in the parameter space and to compare the computed gradient with finite derivative approximations (Section 3-7)
HJ	- the Hooke and Jeeves direct search function minimizer (Section 5-1)
MRT	- the Marquardt-Nash nonlinear least squares solver (Section 11-5)
NM	- the Nelder-Mead polytope direct search function minimizer (Section 6-1)
NUMGRAD	- to compute numerical approximations to the gradient elements of a loss function (Section 7-5)
NUMJAC	- to compute numerical approximations to the derivatives of the I-th residual function in a nonlinear least squares problem, i.e. the I-th row of the Jacobian for this problem (Section 11-8)
POSTGEN POSTMRT POSTVM	- to provide for reporting of results of parameter estimation. Examples are POSTGEN.BAS (Section 13-6), POSTVM.BAS (Section 13-8) and POSTMRT.BAS (Section 13-7). These code segments start at line 6000.
RESSAVE	- to display residuals and to write data to files suitable for PLOT.BAS in Section 15-7 (Section 13-9)
RJTEST	- to compute the residuals for a nonlinear least squares problem at given points in the parameter space and to compare the computed Jacobian elements with finite difference approximations (Section 3-8)
SUMSQR	- to convert function minimization to sum of squares form (Section 2-6)
TN	- the Truncated-Newton function minimizer (Section 9-1)
VM	- the Fletcher (1970) variable metric function minimizer (Section 10-1)

Table 15-1-1. Purpose of program code segments presented (continuation)

II.	Generic code segments which must be supplied by the user. Examples of such code segments are given in Table 15-1-2.
setup	- to provide data, bounds, masks and other information about a parameter estimation problem to our programs. Examples are contained in the problem code segments starting at line 3000.
function	- to compute the loss function at a given point B in the parameter space. Examples are contained in the code segments having filename extension ".fn" starting at line 2000.
gradient	- to compute the gradient of the loss function at a given point B in the parameter space. Examples are contained in the code segments having filename extension ".fn" starting at line 2500.
residual	- for nonlinear least squares problems (or absolute deviation criterion problems), compute the I-th residual at point B in the parameter space. Examples are found in the code segments having filename extension ".res" starting at line 3500.
jacobian	- for nonlinear least squares problems, compute the I-th row of the Jacobian matrix at a given point B in the parameter space. Examples are contained in the program code segments having filename extension ".res" starting at line 4000.
post-analysis	- to provide for reporting of results of parameter estimation. Examples are POSTGEN.BAS (Section 13-6), POSTVM.BAS (Section 13-8) and POSTMRT.BAS (Section 13-7) but the user may wish to supply different or additional measures of dispersion of or relationship between the parameters. These code segments start at line 6000.
results-analysis	- to perform calculations or display of the results of an estimation task in the context of a particular example problem. Always starts at line 4500. Examples are noted in Table 15-1-2.

Table 15-1-2. Example problem files and discussion thereof

Name		Sections	Notes
ABS	FN	4-3,18-1	sum of absolute values
CAL4	FN	8-2,9-2,18-2	contrived test problem
EIGV	FN	14-3,15-3	eigenproblem of a class of tridiagonal symmetric matrices
GENROSE	FN	9-2,12-2,18-2	generalized Rosenbrock problem
LANCLIN	FN	14-1	multiple exponentials -- linear parameters separated
NASHEASY	FN	18-4	large test problem
NASHHARD	FN	14-3,18-5	large test problem
QUADSN	FN	12-2,18-1	quadratic test function
RS	FN	18-7	problem with general inequality constraints
WOOD	FN	18-3	classic test problem
BAGLIVO	RES	11-6,18-9	model of population of soft-shelled clams
BALF	RES	8-2,9-2,18-8	Brown Almost Linear Function
BARD	RES	2-1,16-3	first-order chemical reaction
BMDP	RES	16-4	radioimmunoassay calibration
BT	RES	18-1	test for bounds and masks
CHEM	RES	5-2,16-1,16-2	chemical kinetic problems
COBB	RES	3-4,17-3	Cobb-Douglas production function
GENROSEJ	RES	12-2,18-2	generalized Rosenbrock residuals
HOBBS	RES	2-1,3-3,3-7,3-9,5-2, 6-2,8-2,9-2,10-2,11-6, 11-7,12-2,13-3,13-7,13-8	weed infestation
HS25	RES	10-2,12-2,18-8	Hock & Schittkowski problem 25
KAPESTB	RES	17-1	Kapsalis Canadian macroeconomic model estimation
KMENTALP	RES	17-3	a nonlinear production function
LANCZOS	RES	14-1	multiple exponentials problem
LINTEST	RES	18-1	linear least squares regression
LONION	RES	1-3,18-6	yield/density model (onions)
MARKMOD	RES	17-4	market share Markov model
MHKINX	RES	1-3,16-5	methyl iodide decomposition in aqueous alkaline solution
POWSING	RES	14-3,18-8	extended Powell singular function
WOODR	RES	18-3	classic test problem

15-2. Main Driver Code

The structure of our software is such that we are able to use one driver code to carry out all the estimation calculations in this book. Of course, we need a way to bring together various pieces of program code and to provide data to the resulting program. The tools to accomplish this are discussed in Section 15-5.

Below is a listing of the program segment DRIVER.BAS, which is the first segment in any program we run.

The REMarks within this code explain the functioning of DRIVER in executing all the different routines needed to perform a nonlinear estimation task. However, because the working storage requirements of various methods for estimation and of the diverse problems may be different, we have included DIMension statements in the method and problem code segments rather than in the driver program code.

Most of the BASIC interpreters used on personal computers allow for dynamic DIMensioning of working storage providing the DIM statement is "executed" before the variables for which it allocates storage are used. That is, within the program segments which need the working storage or within the setup routine, we could include lines of the form

```
     3020 LET M = 24: REM 24 DATA POINTS
     3030 LET N = 12: REM 6 PARAMETERS
     3040 DIM B(N), X(N), Y(M,3)
```

However, compilers generally require fixed dimensions. which generally must be established before any statement is executed involving variables, strings or arrays. Therefore, we have reserved the following line numbers for the DIMensioning of working storage.

line 30 for arrays required by the problem
line 40 for arrays required by the estimation method

(If more arrays are needed, lines 31, 32, etc. or 41, 42, etc. may be used.)

A remaining annoyance is that a choice is needed for the amount of working storage to be allowed in our estimation methods. While it would be relatively straightforward to create a program to recognize specially coded statements in the user-supplied problem -- for instance via some REMark statements with special character sequences -- to find and adjust the DIMensions appropriately, this would add another step, with its attendant delays, to the consolidation process. Therefore, we have chosen the crude alternative of fixing the size of the working storage for the estimation methods by setting a maximum value of N = 25 for all methods. Generally, we are satisfied that users will find these allocations adequate, but they may be altered within the code if necessary. Note that some alternative implementations of the Marquardt method may require working storage dependent on both the order (N) of the problem as well as the number of data points (M) in the problem. The strategy we have chosen for working storage allocation will not suffice in this situation. (Note that the programs NM, VM and MRT use rectangular arrays, so require more storage space.)

DRIVER is set up to allow users of the IBM PC and similar computers to record the time taken for the estimation routines to carry out the minimization of the loss function. Only the time within the minimization routine is counted, and if the post-solution analysis of the parameters (for which no timing is made) finds a lower value of the loss function at one of the axial search positions, then DRIVER will allow the user to restart the minimization from this new point. The counts of function and gradient evaluations are cumulative in such restarts (but do not include any evaluations used in the post-solution analysis). However, the "stop-watch" is reset to zero on re-entering the function minimization. Users may easily change the program if these choices prove unsuitable.

DRIVER allows the user to set initial parameters and also to alter masks and bounds. Before such entry, existing bounds and masks are reported to the user. After entry, parameter values are checked. Note that masked parameters may be assigned values outside of the given upper or lower bounds. This is useful in situations where one wishes to examine the loss function surface at points outside the usual range of interest. Any non-masked parameter which is "out-of-bounds" is reset to the nearest bound. In the case where a lower bound has a value in excess of the corresponding upper bound (there are no explicit checks for this possibility), we mask the parameter and set its value to the given upper bound.

A final feature of the main driver code is that it allows the user to open a file (always channel number 3) to which copies of all program-generated output lines are "printed". Thus users should avoid using file #3 for any other purpose (see Section 15-4).

Listing 15-2-1. The main driver code for nonlinear estimation.

```
         DRIVER.BAS        08-28-1986   19:47:47

5 PRINT "DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128"
10 INPUT "FILE FOR CONSOLE IMAGE = ";G$
15 IF G$="" THEN LET G$="NUL"
20 OPEN G$ FOR APPEND AS 3: REM !! may be non-standard
25 PRINT #3,"DRIVER -- GENERAL PARAMETER ESTIMATION DRIVER - 851017,851128
30 REM line 30 reserved for dimensions from problem (B,X,Y,O)
35 REM in case of extra line
40 REM line 40 reserved for dimensions from minimizer
45 REM in case of extra line
50 PRINT #3,TIME$,DATE$: REM !!
55 PRINT TIME$,DATE$: REM !!
60 REM CALLS:
```

```
65 REM    FUNCTION MINIMIZER  -- line 1000
70 REM    SETUP               -- line 3000
75 REM    POST-SOL'N ANALYSIS -- line 6000
80 REM    RESULT ANALYSIS     -- line 4500
85 REM
90 REM X$, T$ and U$ are used for responses and time
95 GOSUB 3000: REM setup of user-specified problem
100 LET I8=0: REM count of iterations or gradient evaluations
105 LET I9=0: REM count of function or sum of squares evaluations
110 PRINT P$: REM display problem title (user supplied)
115 PRINT #3,P$
120 GOSUB 500: REM display bounds
125 PRINT "ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";
130 INPUT X$
135 PRINT #3,"ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) ";X$
140 IF X$="N" THEN 175
145 IF X$="n" THEN 175
150 FOR J=1 TO N
155 PRINT "B(";J;")=";
160 INPUT B(J)
165 PRINT #3,"B(";J;")=";B(J)
170 NEXT J
175 INPUT " are masks or bounds to be set or altered ([cr] = no) ";X$
180 PRINT #3," are masks or bounds to be set or altered ([cr] = no) ";X$
185 IF X$="y" OR X$="Y" THEN 195
190 GOTO 285: REM note that already masked parameters stay masked
195 FOR J=1 TO N
200 PRINT "b(";J;")=";B(J);"  mask ([cr] = no) ";
205 INPUT X$
210 PRINT #3,"b(";J;")=";B(J);"  mask ([cr] = no) ";X$
215 LET O(J,3)=0: REM masked parameter
220 IF X$="y" OR X$="Y" THEN 280: REM if masked, no need for bounds
225 LET O(J,3)=1: REM free parameter
230 PRINT "lower bound =";O(J,1);"  ([cr] = no change) ";
235 INPUT X$
240 PRINT #3,"lower bound =";O(J,1);"  ([cr] = no change) ";X$
245 IF X$="" THEN GOTO 255: REM no change
250 LET O(J,1)=VAL(X$): REM new lower bound
255 PRINT "upper bound =";O(J,2);"  ([cr] = no change) ";
260 INPUT X$
265 PRINT #3,"upper bound =";O(J,2);"  ([cr] = no change) ";X$
270 IF X$="" THEN GOTO 280: REM no change
275 LET O(J,2)=VAL(X$): REM new upper bound
280 NEXT J
285 LET S1=0: REM set steplength parameters within minimizer
290 LET S2=0: REM or change these lines of code
295 GOSUB 560: REM mask and bound check
300 LET T$=TIME$: REM !! SYSTEM DEPENDENT
305 GOSUB 1000: REM function minimization routine
310 LET U$=TIME$: REM !! system dependent
```

```
315 GOSUB 435: REM !! system dependent time display routine
320 PRINT "CALCULATED FUNCTION MINIMUM = ";F0
325 PRINT #3,"CALCULATED FUNCTION MINIMUM = ";F0
330 PRINT "PARAMETER ESTIMATES"
335 PRINT #3,"PARAMETER ESTIMATES"
340 FOR J=1 TO N
345 PRINT "B(";J;") = ";X(J)
350 PRINT #3,"B(";J;") = ";X(J)
355 NEXT J
360 GOSUB 6000: REM post-solution analysis
365 IF F0<=F1 THEN 375: REM no lower point found
367 INPUT "continue from lower point found ([cr] = Y) ";X$
369 PRINT #3,"continue from lower point found ([cr] = Y) ";X$
371 IF X$<>"N" AND X$<>"n" THEN 110
373 REM finished
375 INPUT " call problem-dependent results-analysis ( [cr] = N ) ";X$
385 PRINT #3," call problem-dependent results-analysis ( [cr] = N ) ";X$
390 IF X$="Y" THEN GOSUB 4500
395 IF X$="y" THEN GOSUB 4500
400 PRINT " another set of starting parameters ( [cr] = N ) ";
405 INPUT X$
410 PRINT #3," another set of starting parameters ( [cr] = N ) ";X$
415 IF X$="Y" THEN 100
420 IF X$="y" THEN 100
425 CLOSE #3
430 STOP
435 LET T1=VAL(LEFT$(T$,2)): REM !!
440 LET T2=VAL(MID$(T$,4,2)): REM !!
445 LET T3=VAL(RIGHT$(T$,2)): REM !!
450 LET U1=VAL(LEFT$(U$,2)): REM !!
455 LET U2=VAL(MID$(U$,4,2)): REM !!
460 LET U3=VAL(RIGHT$(U$,2)): REM !!
465 LET T4=T3+60*T2+3600*T1: REM !!
470 LET U4=U3+60*U2+3600*U1: REM !!
475 LET T5=U4-T4: REM !!
480 IF T5<0 THEN LET T5=T5+24*3600!: REM !! ASSUME NEVER OVER 24 HRS
485 PRINT " ELAPSED SECS=";T5;" AFTER";I8;" GRAD &";I9;" FN EVAL": REM!!
490 PRINT #3," ELAPSED SECS=";T5;" AFTER";I8;" GRAD &";I9;" FN EVAL"
495 RETURN: REM !!
500 PRINT "bounds on parameters for problem"
505 PRINT #3,"bounds on parameters for problem"
510 FOR J=1 TO N
515 IF O(J,3)=0 THEN 535
520 PRINT O(J,1);" <=   b(";J;")  <=  ";O(J,2)
525 PRINT #3,O(J,1);" <=   b(";J;")  <=  ";O(J,2)
530 GOTO 545
535 PRINT " ****  b(";J;") WILL BE MASKED (FIXED) ****"
540 PRINT #3," ****  b(";J;") WILL BE MASKED (FIXED) ****"
545 NEXT J
550 RETURN
```

```
555 REM check bounds and set up indicator
560 LET I5=N: REM count masks set
565 FOR J=1 TO N
570 IF O(J,3)=0 THEN  665
575 LET O(J,3)=1: REM reset free or active bound (below)
580 LET I5=I5-1: REM not a masked parameter
585 IF B(J)>O(J,1) THEN  610
590 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
595 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,1)
600 LET B(J)=O(J,1)
605 LET O(J,3)=-2 : REM lower bound active
610 IF B(J)<O(J,2) THEN  665
615 PRINT "parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
620 PRINT #3,"parameter B(";J;") reset from ";B(J);"  to  ";O(J,2)
625 IF O(J,3)>-2 THEN 655:REM check if lower, upper bounds equal (masked)
630 LET O(J,3)=0: REM set mask on
635 LET I5=I5+1: REM adjust mask count
640 PRINT " mask imposed on parameter B(";J;") at upper bound ";O(J,2)
645 PRINT #3," mask imposed on parameter B(";J;") at upper bound ";O(J,2)
650 GOTO 665
655 LET B(J)=O(J,2)
660 LET O(J,3)=-1: REM upper bound active
665 LET X(J)=B(J): REM to ensure "best" parameters are saved
670 NEXT J
675 RETURN


Line no.    Referenced in line(s)
  100       415   420
  110       371
  175       140   145
  195       185
  255       245
  280       220   270
  285       190
  375       365
  435       315
  500       120
  535       515
  545       530
  560       295
  610       585
  655       625
  665       570   610   650
 1000       305 -- the function minimizer subroutine
 3000        95 -- the problem setup code (user supplied)
 4500       390   395 -- results-analysis code (user supplied)
 6000       360 -- post-solution analysis subroutine
 Symbol     Referenced in line(s)
B(          160   165   200   210   585   590   595   600   610
            615   620   655   665 -- the parameter vector
DATE$        50    55 -- the system date function (non-standard)
F0          320   325   365 -- the minimum value of the loss function
            found by the function minimization routine
F1          365 -- lower value of loss function, if any, found by
            post-solution analysis
G$           10    15    20 -- console image file
I5          560   580   635 -- number of masked parameters
I8          100   485   490 -- number of gradient evaluations in
            minimization
I9          105   485   490 -- number of function evaluations in
            minimization
J           150   155   160   165   170   195   200   210   215
            225   230   240   250   255   265   275   280   340
            345   350   355   510   515   520   525   535   540
            545   565   570   575   585   590   595   600   605
            610   615   620   625   630   640   645   655   660
            665   670 -- a loop counter
N           150   195   340   510   560   565 -- the number of parameters
O(          215   225   230   240   250   255   265   275   515
            520   525   570   575   585   590   595   600   605
            610   615   620   625   630   640   645   655   660
            -- bounds and masks information storage
P$          110   115 -- the name of the problem
S1          285 -- initial stepsize for minimization routines
S2          290 -- additional initial stepsize information
T$          300   435   440   445 -- used to store time-stamp
T1          435   465 -- used in computing elapsed time
T2          440   465 -- used in computing elapsed time
T3          445   465 -- used in computing elapsed time
T4          465   475 -- used in computing elapsed time
T5          475   480   485   490 -- elapsed seconds in minimization
TIME$        50    55   300   310 -- system time of day function
U$          310   450   455   460 -- used to store time-stamp
U1          450   470 -- used in computing elapsed time
U2          455   470 -- used in computing elapsed time
U3          460   470 -- used in computing elapsed time
U4          470   475 -- used in computing elapsed time
X$          130   135   140   145   175   180   185   205   210
            220   235   240   245   250   260   265   270   275
            367   369   371   375   385   390   395   405   410
            415   420 -- input response string
X(          345   350   665 -- to store 'best' parameters found
=================================================================
LINES: 138      BYTES: 5428     SYMBOLS: 49     REFERENCES: 206
```

15-3. The Problem File

The problem file provides all the information which is specific to the user's problem. This includes identifying the problem; entering required data; declaring storage for intermediate calculations; specifying the model and loss function and any adjustable weights, bounds, or other constraints, as well as associated derivatives; and describing any special analysis or display of the results.

The user must supply information and program code to prepare for the estimation problem. It is possible to initialize a problem in DRIVER, except for data entry. For convenience, though, it is generally preferable to provide default settings of bounds, masks, and even initial parameter values via the setup portion of the problem file. The problem file also contains either function and gradient or residual and Jacobian information to describe the model and the loss function. Also, for some situations, it is desirable to perform calculations with the estimated parameters. We call this results-analysis, and the user must supply code for such calculations also, if required. Table 15-3-1 presents the structure of the program code which the user must supply. In the setup code, we may enter data via the BASIC statement READ using DATA statements or we may INPUT information from the keyboard or from a file (Section 15-4).

When the loss function or residual is computed it is possible that inadmissible or undesirable arguments to functions are derived from the current parameters. For example, we should not attempt the square root of negative numbers or division by zero. Very large or very small results of function evaluation may, in some cases, lead to overflow or underflow difficulties, or simply to very poorly determined function or derivative values. When such situations arise, we recommend that the code set the variable I3 to 1 and return to the minimization method, since our codes are designed to handle such exceptions in a more or less graceful manner. An example of a problem where this mechanism is quite critical to success is the Brown Almost Linear Function (BALF.RES, Section 18-8).

Table 15-3-1. Contents of the user-supplied problem file

For general function minimization

	File name extension .FN
	line 30 DIMensions for B( ), X( ), O( ), and optionally Y( )
(function)	lines 2000-2499 compute the function f($\underline{B}$) in variable F. If F cannot be computed, set I3=1 and return.
(gradient)	lines 2500-2999 compute the gradient g($\underline{B}$) in vector $\underline{G}$. This code segment is optional when a direct-search minimization method is used (HJ or NM), or when NUMGRAD is to be included in the consolidated program
(setup)	lines 3000-3499 set up the problem. N, the order of the problem, is set or input. If appropriate, M, the number of observations is also provided. P$, a string of character information describing the problem, must be provided. The number of variables (columns of the data matrix Y, which has M rows) is provided. In Y( ) are placed the data required to compute the function f($\underline{B}$) or residual R1($\underline{B}$,I) for the problem at hand. Y( ) need not be completely defined, that is, some elements may not have a value assigned. Bounds and masks are set in the array O(,) $O(J,1) \le B(J) \le O(J,2)$ B(J) is fixed, or masked, if O(J,3) = 0, and is free, or unconstrained, if O(j,3) = 1. I5 is set to the number of masked parameters (it must be set to zero if there are no masks). If desired, initial values for the parameters are set in B( ).

Table 15-3-1. Contents of the user-supplied problem file (cont'd)

(results-analysis)	lines 4500-5999 provide problem-specific post-solution calculations. For example, it may be desired to convert the units in which the parameters are displayed, or use the results to derive further information.
For nonlinear least squares	
	File name extension .RES
(setup)	lines 3000-3499 setup, as above.
(residual)	lines 3500-3999 compute the I-th residual r($\underline{B}$,I) in R1. If we cannot do this, set I3=1 and return.
(jacobian)	lines 4000-4499 compute the partial derivatives of r($\underline{B}$,I) with respect to the parameters $\underline{B}$ and store the results in the vector D. D is therefore the I-th row of the Jacobian matrix. Our programs have been written so that this code is not called unless the residual can be evaluated at $\underline{B}$. Thus no check is made on the "computability" of the Jacobian elements.
(results-analysis)	lines 4500-4599 provide problem-specific post-solution calculations.

Listing 3-3-1 has already presented the HOBBS.RES program code for the Hobbs weed infestation problem. Other examples are discussed in the chapters on the various methods for parameter estimation, in Chapters 16 and 17 which present applications in the areas of economic modeling and chemical kinetics, and in Chapter 18, where a number of problems are presented and discussed. To provide an example which has all features of a problem file, including a results-analysis, Listing 15-3-1 illustrates code to solve the eigenproblem of a class of tridiagonal symmetric matrices.

Listing 15-3-1. Program code EIGV.FN to solve the eigenproblem of a class of tridiagonal, symmetric matrices.

```
1 DEFDBL A-H, O-Z: REM !! double precision recommended
30 DIM Y(25,3),B(25),X(25),O(25,3)

2000 LET Z4=0: REM EIGV.FN - Rayleigh quotient for matrix eigenproblem
2010 LET Z9=0: REM for denominator
2020 FOR L1=1 TO N
2030 LET Z9=Z9+B(L1)*B(L1)
2040 LET Z4=Z4+Y(L1,1)*B(L1)*B(L1)
2050 IF L1>1 THEN LET Z4=Z4+B(L1)*B(L1-1)*Y(L1,2)
2060 IF L1<N THEN LET Z4=Z4+B(L1)*B(L1+1)*Y(L1,2)
2070 NEXT L1
2080 LET F=Z8*Z4/Z9: REM Z8 negative allows for maximal eigenvalue
2090 RETURN

2500 GOSUB 2000: REM derivatives for EIGV; ensure Z4, Z9 defined
2505 FOR L1=1 TO N
2510 LET Z6=2*Y(L1,1)*B(L1): REM diagonal term of numerator
2520 LET Z5=2*B(L1): REM for deriv. of denominator
2525 IF L1>1 THEN LET Z6=Z6+2*B(L1-1)*Y(L1,2)
2530 IF L1<N THEN LET Z6=Z6+2*B(L1+1)*Y(L1,2)
2540 LET G(L1)=Z8*(Z6*Z9-Z5*Z4)/(Z9*Z9)
2550 NEXT L1
2560 RETURN

3000 PRINT "EIGV.FN - Rayleigh Quotient for tridiagonal matrix 860623"
3010 PRINT #3,"EIGV.FN - Rayleigh Quotient for tridiagonal matrix 860623"
3020 LET P$="EIGV - TRIDIAGONAL MATRIX EIGENPROBLEM"
3030 INPUT "INPUT ORDER OF MATRIX (N) = ";N
3040 PRINT #3,"INPUT ORDER OF MATRIX (N) = ";N
3050 REM Y(I,1) stores diagonal of matrix, off diagonals in Y(i,2)
3060 LET L2=INT(N/2)
3070 LET L1=0: REM signal extra element on diagonal
3080 IF 2*L2=N THEN LET L1=1
3090 PRINT "ENTER +1 FOR WILKINSON W+ MATRIX, -1 FOR W- "
3092 INPUT "ENTER ANY NUMBER >+1 FOR A CONSTANT DIAGONAL ";Z9
3100 PRINT #3,"ENTER +1 FOR WILKINSON W+ MATRIX, -1 FOR W- "
3101 PRINT #3,"ENTER ANY NUMBER >+1 FOR A CONSTANT DIAGONAL ";Z9
3102 INPUT "ENTER +1 FOR SMALLEST EIGENVALUE, -1 FOR LARGEST ";Z8
3104 PRINT #3,"ENTER +1 FOR SMALLEST EIGENVALUE, -1 FOR LARGEST ";Z8
3106 REM Z8=-1 allows maximal eigenvalue to be found
3110 FOR J=1 TO L2
3112 IF ABS(Z9)=1 THEN 3120
3114 LET Y(J,1)=Z9: REM constant diagonal
3115 LET Y(N-J+1,1)=Z9
3116 GOTO 3140
3120 LET Y(J,1)=L2-J+1: REM Wilkinson W+ or W- matrix
3130 LET Y(N-J+1,1)=Z9*(L2-J+1)
```

```
3140 LET Y(J,2)=1: REM off diagonals all 1
3142 LET Y(N-J+1,2)=1
3144 NEXT J
3150 IF L1=1 THEN LET Y(L2+1,1)=0
3155 IF Z9>1 THEN LET Y(L2+1,1)=Z9
3156 LET Y(L2+1,2)=1
3160 PRINT "DIAGONAL ELEMENTS OF MATRIX"
3165 PRINT #3,"DIAGONAL ELEMENTS OF MATRIX"
3170 FOR J=1 TO N
3180 PRINT Y(J,1);
3185 PRINT #3,Y(J,1);
3190 NEXT J
3192 PRINT
3194 PRINT #3,
3200 INPUT "limit to absolute value of any parameter ";Z2
3205 PRINT #3,"limit to absolute value of any parameter ";Z2
3210 FOR I=1 TO N
3220 LET O(I,1)=-Z2: REM bounds set via Z2
3230 LET O(I,2)=Z2
3240 LET O(I,3)=1: REM free parameters
3250 NEXT I
3251 INPUT "choose Fletcher and Bradbury (1) or pseudo-random start(2)";L4
3253 IF L4=1 THEN GOSUB 3400: REM Fletcher and Bradbury start
3255 IF L4=2 THEN GOSUB 3325: REM pseudo-random start
3260 INPUT "ONE PARAMETER MAY BE FIXED (MASKED). WHICH ONE OR 0 ?";J
3270 PRINT #3,"ONE PARAMETER MAY BE FIXED (MASKED). WHICH ONE OR 0?";J
3280 LET O(J,3)=0: REM note no check on validity of choice here
3310 PRINT
3320 PRINT #3,
3322 RETURN: REM end of setup
3325 LET Z0=.123454: REM random number seed
3330 PRINT "GENERATED PSEUDO-RANDOM STARTING PARAMETERS"
3335 PRINT #3,"GENERATED PSEUDO-RANDOM STARTING PARAMETERS"
3340 FOR I=1 TO N
3345 LET Z0=RND(Z0)
3350 LET B(I)=Z0-.5
3360 PRINT B(I);
3365 PRINT #3,B(I);
3370 NEXT I
3380 PRINT
3385 PRINT #3,
3390 RETURN
3400 LET L3=1: REM Fletcher & Bradbury start
3410 LET Z3=Z8*Y(1,1)
3415 LET B(1)=0
3420 FOR L1=2 TO N: REM seek smallest/largest diagonal element
3430 IF Z8*Y(L1,1)<=Z3 THEN 3440
3432 LET L3=L1
3434 LET Z3=Z8*Y(L1,1)
3440 NEXT L1
```

```
3450 LET B(L3)=1
3455 PRINT "Fletcher/Bradbury suggestion for start is e(";L3;")"
3457 PRINT #3,"Fletcher/Bradbury suggestion for start is e(";L3;")"
3460 RETURN

4500 PRINT "TEST EIGENVALUE AND VECTOR": REM results-analysis
4510 PRINT #3,"TEST EIGENVALUE AND VECTOR"
4520 PRINT "NORMALIZED EIGENVECTOR"
4530 PRINT #3,"NORMALIZED EIGENVECTOR"
4540 REM normalize eigenvector
4550 LET Z9=0
4560 FOR L1=1 TO N
4570 LET Z9=Z9+B(L1)*B(L1)
4580 NEXT L1
4590 IF Z9>0 THEN 4630
4600 PRINT "NULL VECTOR -- CANNOT CONTINUE"
4610 PRINT #3,"NULL VECTOR -- CANNOT CONTINUE"
4620 STOP
4630 LET Z9=1/SQR(Z9)
4640 FOR L1=1 TO N
4650 LET B(L1)=B(L1)*Z9
4660 PRINT "B(";L1;")=";B(L1)
4670 PRINT #3,"B(";L1;")=";B(L1)
4680 NEXT L1
4690 PRINT
4695 PRINT #3,
4700 FOR L1=1 TO N
4710 LET Z7=-Z8*B(L1)*F: REM diagonal element
4720 LET Z7=Z7+Y(L1,1)*B(L1)
4730 IF L1>1 THEN LET Z7=Z7+B(L1-1)
4740 IF L1<N THEN LET Z7=Z7+B(L1+1)
4750 PRINT "RESIDUAL ";L1;" = ";Z7
4760 PRINT #3,"RESIDUAL ";L1;" = ";Z7
4770 NEXT L1
4780 PRINT
4790 PRINT #3,
4800 RETURN
```

15-4. File Input / Output

Where reasonable, we have avoided using data files in this book. This has the advantages that the number of files included in the package supplied is reduced, and that the translation of programs and problems to BASIC computing environments other than Microsoft's GWBASIC or BASICA is easier. Nevertheless, many users will wish to use data file input. To this end, we have suggested a file structure in the examples presented in Chapter 16. In PC-DOS and MS-DOS, it is possible to assign the file CON as the input file, in which case the keyboard is used as the input source.

A related concern is that of recording the program output in the file. While there are operating system patches which allow for this possibility, we have simply copied all "PRINT" statements as "PRINT #3," where file #3 is OPENed as a disk file. Output to file NUL will avoid the creation of such a file, and DRIVER is set up to use "NUL" if a carriage return is the response to the request for the name of a file to store the console output.

When saving data for later analysis or plotting, our programs use file #2 temporarily. That is, a file assigned to channel #2 is OPENed, data is written, then the file is CLOSEd. The structure of these files is presented in Figure 15-7-1. Users should avoid using file numbers 2 and 3 unless they are aware of the uses to which these channels have already been put in our codes.

While it is not directly a question of file output, control of when and how current parameter estimates are displayed is a detail some users may like to change. Parameter display is controlled within the method program code, that is, within HJ, NM, CG, TN, VM, or MRT. We use the variable J8 to denote how often, in terms of function evaluations, the parameter values are to be displayed.

Whenever the function evaluation counter I9 passes an integral multiple of J8, the parameters are displayed. Variable J7 is used as a temporary storage to hold the integer which is the multiple of J8 which I9 is expected to exceed.

Since the methods each have a different structure, we have chosen particular values for J8 as the default for each method. HJ and NM display the parameters at intervals of approximately 40 function evaluations, CG after 10 evaluations, and TN, VM, and MRT at intervals of 1 evaluation. In the last instance, the call to display parameters is made after each iteration, so that more than one function evaluation may actually take place before the display occurs. While the default choices have proven quite appropriate for our own work, other users may wish to use different values for J8. In particular, the value J8 = 0 suppresses parameter display.

A related matter is the format for displaying parameters. We have chosen to print the parameters, 5 to a line, with each parameter followed by a letter U, L or M if it happens to be at an upper bound, at a lower bound, or masked. This results in a generally compact display on the 25 row by 80 column screen of the IBM PC. Should there be several "small" or "large" parameters, however, an exponential form of display is generated by BASICA or GWBASIC unless the Microsoft BASIC statement PRINT USING is employed. Since we have stressed the importance of scaled parameters at several points in our work, and in particular in Chapter 3, we feel more comfortable in avoiding the use of PRINT USING, which is not part of standards agreed at the time of writing. (We have, however, used this construction in other program codes where the tabular layout of displays is more important.)

15-5. Consolidation of Program Segments

Under PC-DOS or MS-DOS, the main environment considered in developing the material in this book, it is possible to execute pre-recorded sequences of commands from a so-called BATch file. Users of other machines may find useful ideas in this section, but will not be able to use the procedures directly. A batch file must have the filename extension .BAT and may include symbolic parameters %1, %2,..., %9. We have used this device to simplify the task of running nonlinear estimation calculations.

To carry out a particular nonlinear estimation calculation, the user must already have prepared a file problem.RES or problem.FN which accomplishes the setup, results-analysis, and residual/jacobian or function/gradient calculations. Here "problem" is a name to be replaced by the filename given to the particular problem at hand. The appropriate program code is then consolidated and execution is started (possibly with a delay of some seconds) by typing

```
     NL method problem
```

where "method" is one of HJ, NM, VM, CG, TN, or MRT. If the method is MRT, then a function having the name problem.RES must be present on either drive C: or drive B:. If numerical approximations to gradient or jacobian elements are to be used (NUMGRAD or NUMJAC is to be included), then the command to be issued is

```
     NL method problem N
```

A fourth parameter allows the consolidated code to be compiled before execution. Thus the command

```
     NL method problem N C
```

compiles the code to solve "problem" by "method" using numerically approximated derivatives. If analytic derivatives are to be used then the command is

```
     NL method problem X C
```

(any characters other than n or N may replace X).

When numerical derivative approximations are used it is necessary to remove any program code which may be present in the problem file for calculating derivatives. This is done by the program DELETER, which is provided on the program disk in both source and executable form.

When the C(ompile) option is used, it is necessary to properly sequence the BASIC statements according to their line numbers, since out-of-sequence statements will cause a compiler error. The Microsoft interpreters are capable of automatically sequencing the codes, and users may wish to simply SAVE the resulting code using the ASCII option of the SAVE command, that is

```
     SAVE"progname",A
```

However, to allow the NL.BAT procedure to be used, we have included a program SORTER.BAS to sequence consolidated programs, as well as its compiled form SORTER.EXE. We have noted that some text and program editors store files in a form which results in errors when the Microsoft BASCOM compiler is invoked. These may be overcome by using the BASICA or GWBASIC interpreters and loading and saving the program files using the ",A" option when saving. In these commands, if no filename extension is provided, then an extension of the form ".BAS" is automatically appended. However, file with name extensions other than ".BAS" may be specified, as long as they contain BASIC program code. Simply terminating the program name with a period will allow files whose names have no extension (i.e. have a null extension) to be loaded and saved.

NL is a BATch files which is listed in Figure 15-5-1. This ASSUMES that the computing environment at hand has the program code disk in drive B:. The file "problem.RES" or "problem.FN" may be stored on the disk in drive B: or drive C:. Drive C: is assumed to contain a scratch disk (we use a RAMdisk, that is, a simulation of flexible disk by random access memory). The logged-in drive is assumed to contain the BASIC interpreter, which in our case is called GWBASIC, but for many IBM PC's is BASICA. (When the logged-in drive is C:, the machine usually displays the prompt "C>".) Users will have to change the BATch files to accommodate the configuration at hand. We generally use drive C: as the logged-in disk, and ensure that the batch file NL.BAT has been transferred to this "disk". Readers with experience of MS-DOS will be aware of a number of other mechanisms by which required operations may be accomplished.

There are two minor warnings about the use of NL.BAT. First, when the numerical derivative option is used, lines of code from the problem file are actually DELETED on the version of the problem file which is stored on the scratch disk. If the problem file is not present on the scratch disk, a copy is made from the "software" drive (B: in the listing). A danger is that the user modifies the problem file as stored on the scratch disk, then copies it over to the software disk after running the problem with numerically approximated derivatives. (As an aside, we remind the reader that NUMJAC and NUMGRAD alter the parameters without checking whether bounds are violated.) This copy, however, has NUMGRAD or NUMJAC in place of code which may have taken some effort to create.

The second danger concerns the naming of the problem file. Since NL.BAT will work with a file problem.FN or problem.RES, users should avoid having a name used with both filename extensions. As presently established, NL.BAT will first seek the '.FN' file on the scratch disk, then the '.RES' file on this disk. Then it searches the software disk in the same order. In our own work we have on occasion encountered both these nuisances, which are a result of an attempt to render the overall software system as convenient to use as possible.

Depending on the configuration of their computer, users may wish to alter NL.BAT so that different disk drives are used or a different BASIC interpreter or compiler is invoked. We will assume that users are able to make such changes, and include just one version of NL.BAT in the software.

Under the BASICA (or GWBASIC) interpreter, NL.BAT and DRIVER.BAS have been written so that the user may carry out several sets of calculations with the same program. That is, DRIVER is designed to simply STOP rather than returning to the operating system (MS-DOS). Also, the same file may be used to save a copy of the console output for all these "runs", since we have chosen to open such files in APPEND mode. When the user has finished with a particular program, a return to the operating system is accomplished by typing the command

```
SYSTEM
```

followed by a carriage return.

[NOTE ADDED 1995-8-18: The Examples and Extensions to Nonlinear Parameter Estimation contain updated examples of the batch files presented here as well as a BASIC program that does the program consolidation. Some users have reported that the codes can be used under Microsoft Visual BASIC. We have ourselves used them with QBASIC that accompanies Microsoft DOS 5 and 6.]

Listing 15-5-1. Command file NL.BAT to consolidate and run program code to perform nonlinear estimation with user-supplied Jacobian or gradient calculation code.

```
rem nl.bat -- build a file to run a nonlinear estimation problem
rem first parameter is the method code (HJ, NM, CG, TN, VM, MRT)
rem second parameter is problem code
rem third parameter is N for numerical derivatives
rem fourth parameter is C for compiler to be used
rem programs disk in drive b:
rem scratch drive c: set to logged-in drive
c:
if NOT EXIST b:%1.bas goto errorm
if EXIST %2.fn goto onc
if EXIST %2.res goto oncr
if EXIST b:%2.fn goto onb
if EXIST b:%2.res goto onbr
rem error -- cannot find problem file
goto end
:onb
copy b:%2.fn/a
:onc
if %1==mrt goto errmrt
if %1==MRT goto errmrt
if %3==N goto delete
if %3==n goto delete
goto build
:delete
copy %2.fn t1/a
b:deleter
copy t2+b:numgrad.bas %2.fn/a
:build
copy b:driver.bas t1/a
copy t1+b:%1.bas t2/a
copy t2+%2.fn t1/a
goto getpost
:onbr
copy b:%2.res /a
:oncr
if %3==N goto deleter
if %3==n goto deleter
goto buildr
:deleter
copy %2.res t1/a
b:deleter
copy t2+b:numjac.bas %2.res/a
:buildr
copy b:driver.bas t1/a
copy t1+b:%1.bas t2/a
if %1==mrt goto rescopy
if %1==MRT goto rescopy
copy t2+b:sumsqr.bas t1/a
copy t1 t2/a
:rescopy
copy t2+%2.res t1/a
:getpost
if NOT EXIST b:post%1.bas goto genpost
copy c:t1+b:post%1.bas c:t2/a
goto resave
:genpost
copy c:t1+b:postgen.bas c:t2/a
:resave
if EXIST %2.fn goto envron
copy c:t2+b:ressave.bas c:t1/a
copy c:t1 c:t2/a
:envron
copy c:t2+b:envron.bas c:prog.bas/a
erase t2
erase t1
if %4==C goto compile
if %4==c goto compile
:interp
gwbasic c:prog
goto endchk
:compile
if %1==nm goto delhjnm
if %1==NM goto delhjnm
if %1==hj goto delhjnm
if %1==HJ goto delhjnm
goto nowcomp
:delhjnm
copy c:prog.bas c:t1
b:deleter
copy c:t2 c:prog.bas
:nowcomp
copy prog.bas t1/a
b:sorter
copy t2 prog.bas/a
erase t2
erase t1
rem insert Microsoft BASCOM disk in B:
pause
B:BASCOM c:prog,c:prog.OBJ,NUL/O/X
B:LINK c:prog,c:prog.EXE,NUL,B:BASCOM.LIB
ERASE c:prog.OBJ
c:prog
goto endchk
:errorm
rem error -- method file not found
goto end
```

```
:errmrt
rem error -- cannot use mrt for general minimization
goto end
:endchk
if %3==n goto tellfix
if %3==N goto tellfix
goto end
:tellfix
rem *** WARNING *** function or residual code altered on drive c:
rem                 delete if necessary
pause
:end
rem finished !!
```

15-6. Testing Functions and Derivatives

We have recommended that code for functions and residuals and their gradients or derivatives be tested before the minimization codes are invoked. Figure 15-6-1 presents a BATch file to allow for such testing by issuing the command

```
test problem
```

Once again, we assume that the problem code exists on drive C: or drive B: with the filename extension .FN or .RES, and that the batch file test.bat and the BASIC interpreter GWBASIC.EXE are on drive C:, which is assumed to be logged-in.

Listing 15-6-1. Command file TEST.BAT to consolidate and run program code to test analytic derivative calculations in user-supplied Jacobian or gradient calculation code.

```
rem test.bat -- command to test .res and .fn code segments
rem first argument is problem name (.res or .fn)
rem second argument is C for compile, anything else for interpreter
rem using GWBASIC as name of interpreter
rem programs on b:, code segment on c: or b:
rem output in prog.bas on c:
if exist C:%1.res goto oncres
if exist C:%1.fn  goto oncfn
if exist B:%1.res goto onbres
if exist B:%1.fn  goto onbfn
rem problem file not found
goto end
:oncfn
copy b:fgtest.bas+c:%1.fn+b:envron.bas c:prog.bas/a
goto doit
:oncres
copy b:rjtest.bas+c:%1.res+b:envron.bas c:prog.bas/a
goto doit
:onbfn
copy b:fgtest.bas+b:%1.fn+b:envron.bas c:prog.bas/a
goto doit
:onbres
copy b:rjtest.bas+b:%1.res+b:envron.bas c:prog.bas/a
:doit
if %2==C goto compgo
if %2==c goto compgo
gwbasic c:prog
goto end
:compgo
rem insert Microsoft BASCOM disk in B:
pause
copy prog.bas t1
b:sorter
copy t2 prog.bas
erase t2
erase t1
B:BASCOM c:prog,c:prog.OBJ,NUL/O/X
B:LINK c:prog,c:prog.EXE,NUL,B:BASCOM.LIB
ERASE c:prog.OBJ
c:prog
:end
rem finished !!
```

Section 15-7. Plotting

To make our software collection more complete, a program, PLOT.BAS, has been included to provide two dimensional line graphs. Unfortunately as the graphics capabilities are machine-dependent, this code is not ISO standard and therefore not portable. The program was designed to run on an IBM-PC with a color graphics card using "high resolution" of 640 by 200 points. Other environments may support this routine sufficiently to provide a display albeit of different size than the original design. For example, it will not run on machines with other graphics displays unless the BASIC interpreter or compiler has been modified to allow the SCREEN and LINE commands to operate in the same manner as on the color graphics display.

In order to print out the displays made by PLOT.BAS, the user can use the screen print facility available in most IBM PC compatible computers. This is to some extent dependent on the printer available, though a wide range of dot-matrix printers support monochrome printing of the 640 by 200 graphics. Before running the basic interpreter, the command

```
   GRAPHICS
```

should be issued. This executes a program provided with the operating system (DOS) which causes the screen to be printed as a series of dots, rather than a matrix of characters, whenever the key pair 'Shift' and 'Prt Sc' is pressed.

PLOT.BAS is designed to plot data files created by the post-analysis routines i.e. those with PLD as an extension. The required structure is given in Figure 15-7-1. The data points may be in any order as the program sorts them in ascending order by the x value. At present, there is a limit of 600 points which can easily be modified by changing the dimensions of X and Y in the program in Figure 15-7-2.

```
line 1:   title for plot
line 2:   X variable name
line 3:   Y variable name (limit of 15 characters)
line 4:   X(1),Y(1)
  :
line n+4: X(N),Y(N)
```

Figure 15-7-1 Data file structure for PLOT.BAS.

First, PLOT.BAS displays the names of the PLD files on the user-specified drive. As no error checking is provided, there must be at least one PLD file on the specified drive. This can be changed by modifying the FILES statement in the code given in Listing 15-7-1. To select one of the displayed files, only the name must be typed as the program adds the drive and extension. However other data files may be chosen by typing in the complete name and extension (i.e. the "." is necessary) and the drive, if required. PLOT.BAS does not check that the data file structure is appropriate.

Next, PLOT.BAS lists the data points in sorted order. Once any key is pressed, the program displays the graph. The user then has two options which are chosen by pressing any of the following keys:

S or s -- to stop the program
C or c -- to continue i.e. run the program again

A third possibility is indicated in REMark statements:

P or p -- to print the graph (This is identical to using the PrintScreen key.)

This PrintScreen initiation code is totally dependent on some peculiarities of the MS-DOS operating system, and NOT a recommended practice for general software development (Malloy, 1986). For example, it does not work (and causes some very strange results) if PLOT.BAS is compiled with the 'REM' taken out of these statements.

Obviously several enhancements can be made to PLOT.BAS such as the error checking, revised data file structure, or additional options to best suit the user's needs. A point graph can be obtained by substituting the appropriate DRAW commands for LINE. To facilitate this, the complete listing of PLOT.BAS is given in Listing 15-7-1.

Listing 15-7-1. PLOT.BAS -- a graphing program.

```
         PLOT.BAS          08-23-1986   14:27:14

10 DIM X(50),Y(50): REM PLOT -- a m/c specific plotter 860823
20 LET Z1=400: REM x-axis length
30 LET Z2=150: REM y-axis length
40 LET Z3=99: REM left margin (for labels)
50 LET Z4=19: REM top margin (for title)
60 SCREEN 0: REM set screen to normal
70 PRINT "PLOT -- 2-D PLOT OF DATA FILE 860109"
80 INPUT "drive with plot files";X$
90 IF X$<>"" THEN LET X$=X$+":"
100 LET Y$=X$+"*.PLD"
110 FILES Y$: REM list plot files
120 INPUT "name of data file:";F$
130 LET F$=X$+F$: REM add drive specification
140 IF INSTR(F$,".")=0 THEN LET F$=F$+".PLD": REM PLD ext used by POSTXXX
150 OPEN F$ FOR INPUT AS #3
160 INPUT #3,T$: REM title on first line
170 INPUT #3,X$: REM variable names
180 INPUT #3,Y$
190 LET X0=INT((80-LEN(T$))/2): REM to centre title
200 LET T$=SPACE$(X0)+T$
210 LET M=0: REM counter for no. of data points
220 LET Y0=9999999!: REM to store smallest y
230 LET Y9=-999999!: REM to store largest y
240 IF EOF(3) THEN 400: REM check for end of file
250 LET M=M+1
```

```
260 INPUT #3,X(M),Y(M): REM data must be given in (x,y) pairs
270 IF Y(M)<Y0 THEN LET Y0=Y(M): REM find smallest y
280 IF Y(M)>Y9 THEN LET Y9=Y(M): REM find largest y
290 LET I=M-1
300 IF I=0 THEN 240
310 IF X(I+1)>X(I) THEN 240: REM sort pairs by x
320 LET X0=X(I+1)
330 LET X(I+1)=X(I)
340 LET X(I)=X0
350 LET X0=Y(I+1)
360 LET Y(I+1)=Y(I)
370 LET Y(I)=X0
380 LET I=I-1
390 GOTO 300
400 CLOSE #3: REM close file
410 GOSUB 860: REM print out x,y
420 SCREEN 2: REM set hi-res graphics
430 KEY OFF: REM turn off fn key display
440 PRINT T$;: REM print title
450 LET X5=Z1/(X(M)-X(1)): REM x scaling factor
460 LET Y5=Z2/(Y9-Y0): REM y scaling factor
470 LET X0=X(1)
480 LET X9=X(M)
490 FOR I=1 TO M
500 LET X(I)=INT((X(I)-X0)*X5)+Z3: REM scaled values
510 LET Y(I)=Z2+Z4-INT((Y(I)-Y0)*Y5)
520 NEXT I
530 LINE (Z3-17,Z4)-(Z3-17,Z2+Z4): REM y-axis
540 LINE (Z3-19,Z4)-(Z3-17,Z4): REM top y-tick
550 LOCATE 1+INT((Z4+4)/8),1
560 PRINT USING "##.#^^^^";Y9;: REM largest Y
570 LET Z0=INT((Z4+4)/8+((Z2/8)-LEN(Y$))/2)
580 FOR I=1 TO LEN(Y$): REM print Y-axis label
590 LOCATE Z0+I,INT((Z3-17)/16)
600 PRINT MID$(Y$,I,1);
610 NEXT I
620 LINE (Z3-19,Z2+Z4)-(Z3-17,Z2+Z4): REM bottom y-tick
630 LOCATE 1+INT((Z2+Z4+4)/8),1
640 PRINT USING "##.#^^^^";Y0;: REM smallest Y
650 LINE (Z3,Z2+Z4+7)-(Z3+Z1,Z2+Z4+7): REM x-axis
660 LOCATE 3+INT((Z2+Z4+4)/8),INT(Z3/8)-4
670 PRINT USING "##.#^^^^";X0;: REM smallest X
680 LOCATE 3+INT((Z2+Z4+4)/8),INT(Z3/8+4+((Z1/8)-LEN(X$))/2)
690 PRINT X$;: REM x-axis label
700 LINE (Z3,Z2+Z4+7)-(Z3,Z2+Z4+9): REM x-axis tick left
710 LOCATE 3+INT((Z2+Z4+4)/8),INT((Z3+Z1)/8)-4
720 PRINT USING "##.#^^^^";X9;: REM largest X
730 LINE (Z3+Z1,Z2+Z4+7)-(Z3+Z1,Z2+Z4+9): REM x-axis tick right
740 FOR I=1 TO M
745 LET X6=X(I): LET Y6=Y(I): PSET (X6,Y6),1: PSET (X6-1,Y6),1
```

```
750 PSET (X6+1,Y6),1: PSET (X6,Y6-1),1: PSET (X6,Y6+1),1
760 NEXT I: REM plot a + at each point
770 LOCATE 25,1,0: REM bottom left of screen
780 LET Q$=INKEY$
790 IF Q$="S" THEN 960
800 IF Q$="s" THEN 960
810 IF Q$="C" THEN 60: REM do another plot
820 IF Q$="c" THEN 60
830 REM IF Q$="P" THEN GOSUB 710: REM print screen
840 REM IF Q$="p" THEN GOSUB 710: REM print screen
850 GOTO 780: REM ignore other characters
860 FOR I= 1 TO M
870 PRINT X(I),Y(I)
880 NEXT I
890 INPUT "cont";Q$
900 RETURN
910 REM LET A!=-51973.8: REM setup for PRTSCRN
920 REM LET B=VARPTR(A)
930 REM CALL B
940 REM LPRINT CHR$(12)
950 RETURN
960 SCREEN 0: REM set screen to normal
970 STOP

Line no.    Referenced in line(s)
   60       810   820
  240       300   310
  300       390
  400       240
  780       850
  860       410
  960       790   800

 Symbol     Referenced in line(s)
F$          120   130   140   150 -- .PLD plot data file
I           290   300   310   320   330   340   350   360   370
            380   490   500   510   520   580   590   600   610
            740   750   760   860   870   880 -- loop counter
INKEY$      780 -- individual key input function
M           210   250   260   270   280   290   450   480   490
            740   860 -- number of points to plot
Q$          780   790   800   810   820   890 -- user response
T$          160   190   200   440 -- title of plot
X$           80    90   100   130   170   680   690 -- x-axis label
X(           10   260   310   320   330   340   450   470   480
            500   745   870 -- vector of x values to plot
X0          190   200   320   340   350   370   470   500   670
            -- smallest x value
X5          450   500 -- x value scaling factor
X6          745   750 -- temporary x value
```

```
X9          480   720 -- largest x value
Y$          100   110   180   570   580   600 -- y-axis label
Y(           10   260   270   280   350   360   370   510   745
            870 -- vector of y values to plot
Y0          220   270   460   510   640 -- smallest y value
Y5          460   510 -- y value scaling factor
Y6          745   750 -- temporary y value
Y9          230   280   460   560 -- largest y value
Z0          570   590 -- plotting position in pixels
Z1           20   450   650   680   710   730 -- screen x axis in pixels
Z2           30   460   510   530   570   620   630   650   660
            680   700   710   730 -- screen y height in pixels
Z3           40   500   530   540   590   620   650   660   680
            700   710   730 -- screen left margin in pixels
Z4           50   510   530   540   550   570   620   630   650
            660   680   700   710   730 -- screen top line in pixels
=======================================================================
LINES: 97      BYTES: 3232     SYMBOLS: 35     REFERENCES: 176
```

16

EXAMPLES FROM CHEMICAL KINETICS

16-0. Chemical Kinetic Parameter Estimation Problems

This chapter presents several nonlinear estimation problems from the field of chemical kinetics to illustrate the use of the software and ideas of the previous chapters. In Section 2-1, chemical kinetics was mentioned as a source of growth and decay models. The underlying processes concern the actions of chemical compounds resulting in the disappearance of those compounds and the appearance of others. The rate(s) at which compounds are produced or are destroyed are the main focus of attention, although there may be other parameters of interest related to the thermodynamics of the reaction (see Eggers et al., 1964, Chapter 14, as well as various sections of Moellwyn-Hughes, 1961).

16-1. The Michelis-Menten Curve

Caceci and Cacheris (1964) present a simple problem involving the classic Michelis-Menten model for the kinetics of enzyme reactions (see also Watts, 1981). This model relates the velocity (v) of an enzyme reaction, that is the rate at which a reaction product is produced, to the concentration (c) of the substrate undergoring an enzyme catalyzed reaction. The Michelis-Menten model is

(16-1-1) v = B(1) c / [B(2) + c]

where the parameters B(1) and B(2) represent, respectively, the maximum velocity and the half-velocity concentration. That is, when c = B(2) the velocity of the reaction will be half the maximum, B(1). Table 16-1-1 presents two sets of data for which parameter estimates are desired. For the purposes of our programs, we store the observed substrate concentrations in the first column of the array Y(,) and the observed reaction velocities in column 2 of this array.

Since the model function is straightforward, we can compute derivatives of the model function, and hence of the residual

(16-1-2) r(i,B) = B(2)*Y(I,1)/[B(1) + Y(I,1)] - Y(I,2)

These derivatives are

(16-1-3) D(1) = - B(2) * D(2) * D(2)/Y(I,1)

(16-1-4) D(2) = Y(I,1) / (B(1) + Y(I,1))

[NOTE: D(2) in (16-1-4) must be evaluated before D(1) in (16-1-3).]

Here we have chosen to use multiplication in place of squaring to point out that many BASIC interpreters (and some compilers) use the power function to compute squares. That is, X^2 is evaluated by means of an approximation designed to compute X^Y. This may result in unnecessary rounding error and computational effort, which is easily avoided by use of the explicit multiplication.

Caceci and Cacheris used a Nelder-Mead polytope method for finding the parameter estimates. As we have noted, the direct-search methods do not provide readily available estimates of the dispersion of the parameters. Therefore, it will probably be preferable to use a nonlinear least squares program if a least squares loss function is used. As a check on the stability of the parameter estimates, we have

estimated them using least squares, sum of absolute deviations (L-1) and maximum absolute deviation (L-infinity) loss functions. Table 16-1-2 presents the results.

Table 16-1-1. Data for Michelis-Menten enzyme kinetics problems

(Note: no units were given for reaction velocity or substrate concentration in the data sources.)

Set 1:	Caceci & Cacheris (1984) (BYTE.CHM)						
	Substrate concentration						
	c = Y(.,1)	1.68	3.33	5	6.67	10	20
	Reaction velocity						
	v = Y(.,2)	0.172	0.25	0.286	0.303	0.334	0.384
Set 2:	Watts (1981) (WATTS.CHM)						
	Substrate concentration						
	c = Y(.,1)	2.0	2.0	0.667	0.667	0.4	0.4
		2.0	2.0	0.667	0.667	0.4	0.4
	Reaction velocity						
	v = Y(.,2)	0.0615	0.0527	0.0334	0.0258	0.0138	0.0258
		0.0129	0.0183	0.0083	0.0169	0.0129	0.0087

Table 16-1-2. Results of estimation of the Michelis-Menten method using different loss functions. The Hooke and Jeeves method (HJ) was used. The number in brackets under each parameter result is the computed radius of curvature for the parameter.

Problem	Loss function	Value of loss function	B(1)	B(2)
BYTE	Root mean square error (i.e. L-2)	5.2849E-3	2.4538 (140.3)	0.4239 (0.343)
	Mean absolute error (i.e. L-1)	4.3172E-4	2.5231 (0.111)	0.4303 (0.001)
	Max. absolute error (i.e. L-infinity)	1.7986E-2	1.8571 (0.182)	0.4000 (0.002)*
WATTS	Root mean square error	4.0933E-3	1.7016 (807.1)	0.1056 (1.105)
	Mean absolute error	3.8833E-3	1.6941 (4611.)	0.1000 (large)
	Max. absolute error	6.0997E-3	1.6101 (0.441)	0.1000 (0.001)

* The tilt for this parameter was calculated as 12 degrees.

From Table 16-1-2 it is clear that the choice of loss function alters the parameter estimates obtained. Moreover, the radius of curvature dispersion estimates are quite sensitive to the altered loss function. In the computations, we actually used the least squares, sum of absolute residuals, and minimum maximum residual loss functions. The minimum loss function value has been transformed in the table where appropriate. That is, the minimal sum of squared residuals is divided by the number of residuals and the square root taken to give the root mean square "error". Similarly, the mean absolute error is computed. These measures are then more or less comparable with each other and the maximum absolute residual.

16-2. More Complicated Kinetic Models

Dr. Frank Perrella of the Wistar Institute in Philadelphia (who is now working for Du Pont) has provided us with some more complicated problems in enzyme kinetics. Of these, we have selected several to illustrate the possible extensions of the Michaelis-Menten model.

1. The "one-site plus NSB" model

(16-2-1) $v = B(2) * c / (B(1) + c) + B(3) * c$

2. The "one-site sigmoid" model

(16-2-2) $v = B(2) * c^{B(3)} / (B(1)^{B(3)} + c^{B(3)})$

3. The "one-site sigmoid + NSB" model

(16-2-3) $v = B(2) * c^{B(3)} / (B(1)^{B(3)} + c^{B(3)}) + B(4) * c$

4. The "two-site" model

(16-2-4) $v = B(2) * c / (B(1) + c) + B(4) * c / (B(3) + c)$

We can subsume all these models into the general form

(16-2-5) $v = B(2) * c^{B(3)}/ (B(1)^{B(3)} + c^{B(3)}) + B(4) * c + B(5) * c / (B(6) + c)$

since each of the other forms is then derived from the others by forcing appropriate parameters to zero through the use of masks. In all these models, positive parameters are desired, so that appropriate bounds should be imposed. Note that it is now worthwhile computing analytic derivatives for this generalized model.

Because we are using just one functional form for several problems, it is worth our while to compute derivatives of the model function, and hence of the residuals, analytically. We would point out, however, that Dr. Perrella has generally been satisfied to use numerical approximations. Listing 16-2-1 presents program code for these problems. Listing 16-2-2 gives the contents of the data files (to which commentary has been added) for the problems of both this and the previous section. The results of estimation using program MRT are given in Table 16-2-1.

Listing 16-2-1. Program code CHEM.RES for chemical kinetics problems.

```
30 DIM Y(100,6),B(6),X(6),O(6,3)
3000 PRINT "CHEM - GENERAL KINETIC PROBLEM -- 860102"
3002 PRINT #3,"CHEM - GENERAL KINETIC PROBLEM -- 860102"
3005 LET P$="CHEM.RES kinetic problem 860102"
3010 LET N=6: REM 6 parameters in model used
3015 FOR I=1 TO 6: REM different functional forms by masking parameters
3020 LET B(I)=0: REM parameters initialized to zero
3025 NEXT I
3030 LET B(3)=1: REM except for exponent
3035 INPUT "DISK DRIVE ON WHICH DATA FILES TO BE FOUND? ";X$
3040 FILES X$+":*.CHM": REM display data file names
3045 INPUT "CHOOSE DATA FILE FROM ABOVE:",Z$
3050 LET Z$=X$+":"+Z$
3052 PRINT #3,"CHOOSE DATA FILE FROM ABOVE:",Z$
3055 IF INSTR(1,Z$,".")=0 THEN LET Z$=Z$+".CHM":REM check for extension
3060 LET P$=P$+" + DATAFILE "+Z$
3065 FOR I=1 TO N: REM initialize bounds
3070 LET O(I,1)=0: REM lower bound
3075 LET O(I,2)=100: REM loose upper bound
3080 LET O(I,3)=1: REM not masked
3085 NEXT I
3090 OPEN Z$ FOR INPUT AS #1
3095 LINE INPUT #1,X$: REM !! name  --  NOTE LINE INPUT
3100 PRINT X$
3105 PRINT #3,X$: REM !! may want to omit all prints to channel #3
3110 INPUT #1,X$: REM date
3115 PRINT X$
3120 PRINT #3,X$
3125 INPUT #1,M: REM number of data points
3130 PRINT M;" data points"
3135 PRINT #3,M;" data points"
3140 INPUT #1,N: REM number of parameters, possibly < 6
3145 PRINT N;," parameters"
3150 PRINT #3,N;" parameters"
3155 IF N>6 THEN STOP: REM safety check
3160 REM now read the data series
3165 INPUT #1,N3: REM number of data series
3170 PRINT N3," data series"
```

```
3175 PRINT #3,N3," data series"
3180 FOR J=1 TO N3
3185 PRINT "data series ";J;" : ";
3190 INPUT #1,X$
3195 PRINT X$
3200 PRINT #3,"data series ";J;" : ";
3205 PRINT #3,X$
3210 FOR I=1 TO M: REM loop over data points
3215 INPUT #1,Y(I,J)
3220 PRINT Y(I,J);
3225 PRINT #3,Y(I,J);
3230 IF 5*INT(I/5)=I THEN PRINT
3235 IF 5*INT(I/5)=I THEN PRINT #3,
3240 NEXT I
3245 PRINT
3250 PRINT #3,
3255 NEXT J
3260 IF EOF(1) THEN 3360
3265 INPUT #1,X$: REM label for bounds
3270 PRINT X$
3275 PRINT #3,X$
3280 IF LEFT$(X$,3)="END" OR LEFT$(X$,3)="end" THEN 3360
3285 FOR I=1 TO N: REM read bounds
3290 INPUT #1,O(I,1),O(I,2),O(I,3)
3295 PRINT O(I,1),O(I,2),O(I,3)
3300 PRINT #3,O(I,1),O(I,2),O(I,3)
3305 NEXT I
3310 IF EOF(1) THEN 3360
3315 INPUT #1,X$: REM label for parameters
3320 PRINT X$
3325 PRINT #3,X$
3330 IF LEFT$(X$,3)="END" OR LEFT$(X$,3)="end" THEN 3360
3335 FOR I=1 TO N: REM read starting parameters
3340 INPUT #1,B(I)
3345 PRINT B(I)
3350 PRINT #3,B(I)
3355 NEXT I
3360 CLOSE #1: REM end of data input
3365 PRINT " FUNCTION FORM = B(2)*Y(I,1)^B(3)/(B(1)^B(3)+Y(I,1)^B(3))"
3370 PRINT "           + B(4)*Y(I,1) + B(5)*Y(I,1)/(B(6)+Y(I,1))-Y(I,2)"
3375 PRINT
3380 PRINT #3," FUNCTION FORM = B(2)*Y(I,1)^B(3)/(B(1)^B(3)+Y(I,1)^B(3))"
3385 PRINT #3,"        + B(4)*Y(I,1) + B(5)*Y(I,1)/(B(6)+Y(I,1))-Y(I,2)"
3390 PRINT #3,
3395 RETURN
3500 LET I3=0: REM set computability flag
3510 IF B(1)^B(3)+Y(I,1)^B(3)=0 THEN 3540
3515 IF B(6)+Y(I,1)=0 THEN 3540
3520 LET R1=B(2)*Y(I,1)^B(3)/(B(1)^B(3)+Y(I,1)^B(3))+B(4)*Y(I,1)
3525 LET R1=R1+B(5)*Y(I,1)/(B(6)+Y(I,1))-Y(I,2): REM gen chem kinetic eqn
```

```
3530 RETURN
3540 LET I3=1: REM not computable
3550 RETURN
4000 LET Z1=B(1)^B(3)
4010 LET Z2=Y(I,1)^B(3)
4020 LET Z=Z1+Z2
4030 LET D(2)=Z2/Z
4040 LET D(1)=-D(2)*B(2)*B(3)*B(1)^(B(3)-1)/Z
4050 LET D(4)=Y(I,1)
4060 LET D(5)=Y(I,1)/(B(6)+Y(I,1))
4070 LET D(6)=-D(5)*B(5)/(B(6)+Y(I,1))
4080 LET D(3)=D(2)*B(2)*LOG(Y(I,1)): REM
          +D(1)*(LOG(B(1))*Z1+LOG(Y(I,1))*Z2)
4085 LET D(3)=D(3)-B(2)*Z2*(Z1*LOG(B(1))+Z2*LOG(Y(I,1)))/(Z*Z)
4090 RETURN
```

Listing 16-2-2. Commented data files for chemical kinetics problems. Comments, which are not part of the files, are enclosed within braces { }. Other data files on the disk suitable for use with the CHEM.RES problem are WATTS.CHM and all other files Perrell?.CHM, where ? is a digit.

```
BYTE problem (Caceci & Cacheris, 1984)
860406
6    {6 data points}
2    {2 parameters}
2    {2 data series or variables}
Substrate concentration
1.68,3.33,5,6.67,10,20
Reaction velocity
.172,.25,.286,.303,.334,.384
bounds
0,100,1
0,100,1
parameters
1
1
end

Perrell8 two site model
860406 [3H]PDBu binding to rat brain cytosol receptor
12   {12 data points}
6    {6 parameters}
2    {2 data series or variables}
Substrate concentration
.2,.39,.59,.81,1.62,3.29,6.7,8.43,12.9,22.6,30.8,44.7
Reaction velocity
.064,.108,.14,.184,.322,.581,.981,1.14,1.46,1.99,2.59,2.94
bounds
0,100,1
0,100,1
0,100,0  {note that parameters 3 and 4 are masked}
0,100,0
0,100,1
0,100,1
parameters
2.909
0.4824
1        {parameter 3 fixed at 1}
0
5.45
51.85
end
```

Table 16-2-1. Estimation of various chemical kinetic models of the Michaelis-Menten type. The data sets and initial parameter estimates are as in the files illustrated in Listing 16-2-1. All computations performed with compiled programs based on MRT.

Data Set	Analytic Derivatives					Numeric Derivatives			
	Order	Grad	Fn	Time	Fmin	Grad	Fn	Time	Fmin
Byte	2	7	7	5s	1.6757E-4	7	7	6s	1.6757E-4
Watts	2	8	11	9s	2.0106E-4	7	11	9s	2.0106E-4
Perrell1	3	6	11	14s	1.5759E-3	6	12	20s	1.5759E-3
from (1,1,1)		13	20	27s	1.5759E-3				
from (1,1,1M)		6	10	7s	1.6639E-3				
then free B(3)		+5	+9	+11s	1.5759E-3				
Perrell2	3	3	8	7s	3.9695E-3	3	9	10s	3.9695E-3
from (1,1,1)		11	17	19s	3.9695E-3				
from (1,1,1M)		9	18	10s	4.3293E-1				
then free B(3)		+9	+13	+15s	3.9695E-3				
Perrell3	4	5	10	14s	1.5759E-3	4	11	18s	1.5759E-3
from (1,1,1,1)		6	6	10s	6.7074E-2*	6	6	15s	6.6798E-2*
from (1,1,1M,0M)		6	10	9s	1.6639E-3				
then free B(3)		+5	+9	+13s	1.5759E-3				
then free B(4)		+1	+4	+4s	1.5759E-3				
Perrell4	3	5	10	11s	6.6248E-3	2	9	8s	6.6248E-3
from (1,1,1)		24	34	44s	6.6248E-3	22	35	57s	6.6248E-3
from (1,1,1M)		8	12	9s	1.7651E-2				
then free B(3)		+6	+15	+14s	6.6248E-3				
Perrell5	3	3	7	8s	5.2255E-3	4	10	13s	5.2255E-3
from (1,1,1)		7	11	15s	5.2255E-3	6	14	18s	5.2255E-3
from (1,1,1M)		8	17	11s	1.9983E-2				
then free B(3)		+14	+20	+28s	5.2255E-3				
Perrell6	4	2	7	6s	4.3218E-3	4	11	12s	4.3218E-3
from (1,1,1,1)		8	12	12s	4.4857E-3	13	23	32s	4.3217E-3
Perrell7	4	2	6	6s	3.1904E-4	3	8	10s	3.1904E-4
from (1,1,1,1)		8	14	14s	9.5798E-4	17	29	49s	3.1904E-4
Perrell8	4	2	7	7s	2.5423E-2	3	10	11s	2.5427E-2
from (1,1,1,1)		11	18	20s	3.5256E-2	5	7	13s	1.4856E-1*

Table 16-2-1. Estimation of various chemical kinetic models of the Michaelis-Menten type (continued).

Notes: * == the program indicates a lower point exists when performing the axial search in POSTMRT

Perrell6, Perrell7 and Perrell8 use a nominal set of 6 parameters with parameters 3 and 4 masked to 1 and 0 respectively (see Equation 16-2-5).

An M after a parameter means that it was masked to the stated value during this run. Subsequent runs are presented with a '+' in front of performance measures to indicate that these counts are additional to those for the run immediately above.

From Table 16-2-1, and the work which went into its creation, several points concerning the performance of nonlinear estimation programs may be observed.

1. On the measure of execution time, the program which used analytic expressions for derivatives was faster in 14 of 16 cases, tied in one, and slower in one. Our general experience is that if analytic derivatives are available, they are worth using, partly for speed, and partly because they allow the dispersion estimates to be more reliably computed.

2. Despite this last recommendation, it is very easy to make errors in working out and coding the required derivatives. In the present example, we made an error in one of the derivative expressions (for B(3)). Since this expression was not always evaluated, our programs appeared to operate correctly, but we noted that convergence of the program with analytic derivatives was marginally slower for most problems, and markedly slower for two of them (Perrell3 and Perrell5). Worse, the program RJTEST did not discover the error, since our expression for the derivative had a term in log(B(3)) and our test points used B(3)=1 or had other values which rendered the overall derivative approximately

correct. We can only warn that slow convergence is frequently a sign that derivative expressions are incorrect.

3. The use of masks to first estimate a subset of the parameters and then the rest gives mixed results in terms of performance. In some cases, such a strategy results in much quicker overall convergence, while in others a slower approach to the solution is observed. What the use of masks does gain us, however, is some appreciation of the value of the "extra" parameters in modeling the phenomenon under consideration. Furthermore, we have some control over the process of estimation and can restrict the parameter values more easily.

16-3. A Simple Chemical Kinetics Example

Bard (1974, page 123ff.) presents a nonlinear parameter estimation problem arising from the simplest of chemical reactions, the first order transformation of compound A into compound B. If t represents the time since the commencement of the reaction (or of our observation of it) and y is the fraction of A remaining, then this situation is modeled by the differential equation

(16-3-1) $dy/dt = - k y$

where k is the rate constant of the reaction. The solution for this differential equation, given y = 1 at t = 0, is

(16-3-2) $y = \exp(- k t)$

However, k varies with the absolute temperature T according to the model

(16-3-3) $k = B(1) \exp(- B(2) / T)$

This problem was introduced briefly in Section 2-1. Let us put the time t in column 2 of the data array Y(,), the corresponding absolute temperature in column 3 of the array, and the fraction of reagent left at time t in column 1. The residual function is then

(16-3-4) $r(i, \underline{B}) = \exp(Z2) - Y(i, 1)$

where

(16-3-5) $Z2 = -B(1) * Y(i,2) * \exp((-B(2) / Y(i,3))$

The analytic derivatives of (16-3-4) with respect to the parameters $\underline{B}$ are straightforward to evaluate and expressions are given in the code segment BARD.RES which appears as Listing 16-3-1. The results of estimating the parameters under a simple least squares loss function via method MRT using initial estimates (750, 1200) are

```
     B(1) = 813.9229  with standard error estimate 246.2581
     B(2) = 961.0168  with standard error estimate 68.5346.
```

The residual sum of squares is 0.03980605 after 12 iterations and 17 sum of squared evaluations requiring 84 seconds on the Corona PC-21 under GWBASIC. The gradient components with respect to both parameters are of the order of 1E-9 and the tilts are less than 0.01 degrees. Bard reports parameter values (813.4583, 960.9063) and a sum of squares of 0.03980559. He indicates that the starting point (100, 2000) is "a much poorer initial guess". Nevertheless, MRT converges to (813.8326, 960.9914) after 13 iterations and 19 sum of squares evaluations in 92 seconds under GWBASIC. This illustrates that methods may individually be sensitive to particular starting points. There are, however, problems for which certain starting points give difficulty to a wide class of minimization methods.

Listing 16-3-1. BARD.RES problem file.

```
30 DIM Y(15,3),B(2),X(2),O(2,3)
3000 PRINT "BARD 2-PARAMETER CHEMICAL REACTION - 851215"
3010 PRINT #3,"BARD 2-PARAMETER CHEMICAL REACTION - 851215"
3020 LET P$="BARD.RES Bard kinetic problem pp.123-131"
3030 LET M=15: REM 15 data points
3040 LET N=2: REM 2 parameters
```

```
3050 RESTORE 3200: REM reset data pointer
3060 FOR I=1 TO M
3070 READ Y(I,2),Y(I,3),Y(I,1): REM time, temp, reagent (note order)
3080 NEXT I
3090 REM now set the bounds on the parameters
3100 LET O(1,1)=0: REM positive asymptote
3110 LET O(1,2)=10000: REM loose upper bound
3120 LET O(1,3)=1: REM not masked
3130 LET O(2,1)=-100
3140 LET O(2,2)=100000!: REM loose bounds on second parameter
3150 LET O(2,3)=1: REM not masked
3160 LET I5=0: REM count of number of masked parameters
3170 PRINT " FUNCTION FORM = exp[-B(1)*Y(i,2)*exp(-B(2)/Y(i,3))]"
3175 PRINT
3180 PRINT #3," FUNCTION FORM = exp[-B(1)*Y(i,2)*exp(-B(2)/Y(i,3))]"
3185 PRINT #3,
3190 RETURN
3200 DATA 0.1, 100, 0.980
3201 DATA 0.2, 100, 0.983
3202 DATA 0.3, 100, 0.955
3203 DATA 0.4, 100, 0.979
3204 DATA 0.5, 100, 0.993
3205 DATA 0.05, 200, 0.626
3206 DATA 0.1, 200, 0.544
3207 DATA 0.15, 200, 0.455
3208 DATA 0.2, 200, 0.225
3209 DATA 0.25, 200, 0.167
3210 DATA 0.02, 300, 0.566
3211 DATA 0.04, 300, 0.317
3212 DATA 0.06, 300, 0.034
3213 DATA 0.08, 300, 0.016
3214 DATA 0.1, 300, 0.066
3500 LET I3=0: REM Bard chemical reaction
3520 LET Z1=-B(2)/Y(I,3)
3530 IF ABS(Z1)>40 THEN 3580: REM out of bounds
3540 LET Z2=-B(1)*Y(I,2)*EXP(Z1)
3550 IF ABS(Z2)>40 THEN 3580
3560 LET R1=EXP(Z2)-Y(I,1)
3570 RETURN
3580 LET I3=1
3590 RETURN
4000 LET Z1=-B(2)/Y(I,3)
4010 LET D(1)=-Y(I,2)*EXP(Z1)*EXP(-B(1)*Y(I,2)*EXP(Z1))
4020 LET D(2)=-B(1)*D(1)/Y(I,3)
4030 RETURN
```

16-4. The BMDP and Related Examples

A problem closely related to the above ones is used as an example for the nonlinear estimation program in the BMDP (1981, page 297) suite of programs. The problem is to determine the three parameters in the model

(16-4-1) Z(B,x) = B(3) + 1 / (x * B(1) + B(2))

which is believed to describe the counts of radiation emitted from a sample having concentration x of insulin after treatment with a radioactive agent in a radioimmunoassay. The values of x are "activity" corresponding to levels of insulin in some standard samples. In this example the units were not given. The radiation counts, y, were scaled by dividing by 1000.

Listing 16-4-1 gives the problem file for this particular example. As the data required is limited, we have opted to use DATA statements rather than a separate file. Table 16-4-1 gives the results of our methods on this problem.

Listing 16-4-1. BMDP.RES problem file.

```
30 DIM B(3),X(3),Y(14,2),O(3,3)
3000 PRINT
3001 LET P$="BMDP.RES Diabetologia problem"
3002 PRINT "BMDP example (Brown,Diabetologia,10,23-25 (1974))"
3003 PRINT
3004 PRINT "  model:  Z( B, X) =  B(3) + 1 / (X * B(1) + B(2))"
3005 PRINT
3006 PRINT #3,"BMDP example (Brown,Diabetologia,10,23-25 (1974))"
3007 PRINT #3,
3008 PRINT #3,"  model:  Z( B, X) =  B(3) + 1 / (X * B(1) + B(2))"
3009 PRINT #3,
3010 LET N=3: REM parameters
3020 LET M=14: REM observations
3030 RESTORE 3200
3031 PRINT "  data"
3032 PRINT #3,"  data"
3033 PRINT "activity","counts"
3034 PRINT #3,"activity","counts"
```

```
3040 FOR I=1 TO M
3050 READ Y(I,1),Y(I,2): REM X, Z-observed
3051 PRINT Y(I,1),Y(I,2)
3052 PRINT #3,Y(I,1),Y(I,2)
3060 NEXT I
3070 FOR J=1 TO 3
3080 LET O(J,1)=-100: REM loose lower bound
3081 LET O(J,2)=100: REM loose upper bound
3082 LET O(J,3)=1: REM not masked
3090 NEXT J
3095 LET I5=0: REM no of masked variable
3100 RETURN: REM end of setup
3200 DATA 0,9.274
3201 DATA 0,9.522
3202 DATA 5,8.082
3203 DATA 5,8.354
3204 DATA 10,7.296
3205 DATA 10,7.518
3206 DATA 25,5.864
3207 DATA 25,5.974
3208 DATA 50,4.396
3209 DATA 50,4.110
3210 DATA 100,2.830
3211 DATA 100,2.674
3212 DATA 200,1.798
3213 DATA 200,1.566
3220 REM end of data
3500 IF (.001*B(1)*Y(I,1)+.1*B(2))=0 THEN 3600
3505 LET I3=0
3510 LET R1=1/(.001*B(1)*Y(I,1)+.1*B(2))+.1*B(3)-Y(I,2)
3520 RETURN: REM end of BMDP residual
3600 LET I3=1: REM not computable
3610 RETURN
4000 LET D(2)=1/(.001*B(1)*Y(I,1)+.1*B(2))
4010 LET D(2)=-D(2)*D(2)*.1
4020 LET D(1)=Y(I,1)*D(2)*.01
4030 LET D(3)=.1
4040 RETURN: REM end of BMDP Jacobian
```

Table 16-4-1. Results of our methods on the BMDP radioimmunoassay problem. Calculations performed under GWBASIC on the Corona PC-21 computer (Intel 8088 processor).

Method	Time (s)	Function Evaluations	Gradient Evaluations	Function Value	Parameters B(1)	B(2)	B(3)
HJ	517	0	575	.249146	2.694894	1.09	1.395142
NM 1	198	0	143	.2491481	2.69208	1.089564	1.362476
CG	184	33	84	.2505029 <			
	+29	37	99	.2504854 <			
	+53	45	125	.2504595 <			
	+41	51	145	.2504398 <#	2.644163	1.085396	.9280808
TN	145	42	22	.2491448	2.692519	1.089711	1.370697
VM	116	20	39	.2491441	2.693598	1.089807	1.380485
MRT	63	7	12	.2491444	2.693599	1.089808	1.380495
BMDP software manual (scaled numbers)		7	n/a	.249144	2.6941	1.08981	1.38048

< implies a lower function value was found by POSTGEN
# program manually stopped at this point

Parameter dispersion estimates

Origin	for B(1)	for B(2)	for B(3)
NM & POSTGEN			
radii of curvature	2.024E-1	3.159E-3	7.108
tilts	-0.1123	1.787	1.349E-2
POSTVM -- inverse Hessian approximation	2.264E-2	1.157E-4	2.265E-2

Note: 1995-8-17 -- these should be replaced by their square roots

Origin	for B(1)	for B(2)	for B(3)
POSTMRT -- inverse Jacobian inner product	.2203	2.071E-2	1.864
BMDP software manual (scaled numbers)	0.220	0.0207	1.864

Parameter correlation estimates from POSTMRT

		B(1)	B(2)	B(3)
B(1)	:	1.00000		
B(2)	:	0.74460	1.00000	
B(3)	:	0.93570	0.88301	1.00000

16-5. Concurrent Chemical Reactions

In Section 1-3 we have already presented data for the decomposition of methyl iodide in aqueous alkaline solution (Moelwyn-Hughes, 1961, page 1252ff.). This situation involves two mechanisms by which methyl iodide may be converted to methanol and a free iodide ion.

(A) $CH_3I + H_2O \text{ ---> } CH_3OH + H^+ + I^-$

(B) $CH_3I + OH^- \text{ ---> } CH_3OH + I^-$

We assign a rate constant B(1) to the first reaction and a rate constant B(2) to the second. Suppose we start with an initial concentration, a, of methyl iodide, and an initial concentration, b, of hydroxyl ion. If (a-x) is the concentration of methyl iodide remaining, then (b-x) is the concentration of hydroxyl ion left, since hydroxyl ions are either neutralized by the methyl iodide reaction (B) or by the hydrogen ion in reaction (A). The concentration of solvent (water) is considered large and constant. Rather than putting this concentration into the rate law explicitly, we subsume it in the rate constant B(1). The overall rate of change of the iodide concentration x is given by the differential equation

(16-5-1)   dx/dt = B(1)*(a-x) + B(2)*(a-x)*(b-x)

This can be integrated analytically, following Moelwyn-Hughes exposition, to

(16-5-2)   t = log[{a/(a-x)}{B(1)+B(2)*(b-x)}/{B(1)+B(2)*b}]/q

where

(16-5-3)   q = B(1) + B(2) * (b - a)

Defining

(16-5-4)   v = B(2)*a/{B(1) + B(2) * b}

we rearrange (16-5-2) to provide the fractional completion of

the overall reaction as a function of time as

(16-5-5)   (x/a) = [exp(q*t) - 1]/[exp(q*t)-v]

The data in Table 1-3-2 can now be used to estimate the parameters B(1) and B(2) which appear in Equation (16-5-5) and the subsidiary expressions (16-5-3) and (16-5-4). While this now appears straightforward, the limiting condition, that 100*x/a tends to 100% as time t becomes large, will in practice cause computational difficulties. This is because the evaluation of exp(q*t) will overflow for large arguments. Re-writing (16-5-5) as

(16-5-6)   (x/a) = [1 - exp(-q*t)]/[1-v*exp(-q*t)]

avoids this difficulty, since exp(-q*t) can be replaced by zero for q*t "large". In some computing environments, this "underflow to zero" is automatic; in others, we may have to take special steps to avoid underflow or "argument out of range" type errors causing premature termination of our program.

Listing 16-5-1 presents the program code for the Moelwyn-Hughes problem. The derivative expressions are relatively tedious to develop and test. They took us well over an hour to develop, code and test. In practice, with only two parameters to estimate, we would generally use numerical derivatives and NOT prepare the analytic code. However, the results here show the importance of analytic derivatives in some cases. The relatively poor performance of the numerical derivatives is probably related to the fact that Microsoft's BASIC interpreters generally evaluate special functions (exp, sin, cos, log, etc.) in SINGLE precision only, even if all variables are double precision. The compiler, on the other hand, does claim double precision evaluation (Microsoft, 1983, page 79). Table 16-5-1 compares results generated in different ways. Note that the use of numerical Jacobian with VM is carried out by MERGEing code within the BASIC interpreter and not via the batch procedure

NL of Section 15-5.

Two minor points should be noted about the code and the problem. First, we have not bothered to incorporate a scaling into the parameters, preferring to "see how things look" before undertaking extra work. Since the final parameter estimates are of roughly similar magnitudes, we have decided not to carry out the scaling, since it is differences in magnitude which cause the most difficulty for algorithms, in particular for HJ and NM. Second, we note that Moelwyn-Hughes did not attach units to his data, which we would prefer to see as an aid to judging the scale of the parameters we are trying to estimate.

Table 16-5-1. Various results for the methyl iodide decomposition problem. All calculations performed with the BASICA interpreter on a Maestro PC (NEC V20 processor).

Method	Time	Evaluations		Residual	B(1)	B(2)
	(s)	Gradient	Function	Sumsquares		
HJ	64	0	72	.3329566	9.999001E-4	-.00127
NM	72	0	61	.2119008	5.832319E-4	2.907586E-3
VM	112	11	94	.2122634	5.868147E-4	2.875544E-3
VM(DP)	76	9	46	.2118925	5.858285E-4	2.881887E-3
VM+NUMGRAD	77	7	75	.372517<	1.343352E-4	7.385634E-3
	+35	9	113	.358401	1.322872E-4	7.423055E-3
VM+NUMGRAD(DP)	>640**	33	417	.229871	7.452619E-4	1.282232E-3
VM+NUMJAC	106	10	72	.2121343	6.041701E-4	2.698163E-3
VM+NUMJAC(DP)	>1150**	61	642	.2287890	7.405557E-4	1.330142E-3
MRT	32	5	9	.211897	5.856642E-4	2.883530E-3
MRT+NUMJAC	39	5	9	.2121404	6.039180E-4	2.699840E-3

** manually stopped

< implies that POSTGEN found a lower function value

Listing 16-5-1. MHKINX.RES code - Methyl Iodide Decomposition

```
30 DIM B(2), X(2), O(2,3), Y(8,3)
3000 PRINT "MHKINX.RES -- Moelwyn-Hughes methyl iodide"
3005 PRINT "   decomposition in alkaline aqueous solution"
3010 PRINT "    851105 -- see p. 1252 Moelwyn-Hughes,"
3015 PRINT "    Physical Chemistry, 2nd edition, Pergamon, 1961"
3020 PRINT
3025 PRINT #3,"MHKINX.RES -- Moelwyn-Hughes methyl iodide"
3030 PRINT #3,"   decomposition in alkaline aqueous solution"
3035 PRINT #3,"    851105 -- see p. 1252 Moelwyn-Hughes,"
3040 PRINT #3,"    Physical Chemistry, 2nd edition, Pergamon, 1961"
3045 PRINT #3,
3050 LET P$="MHKINX.RES 851229 -- Moelwyn-Hughes methyl iodide p.1252"
3060 LET N=2: REM 2 parameters
3070 LET M=8: REM 8 time observations
3080 RESTORE 3200
3090 PRINT " time (secs)"," % reaction complete"
3095 PRINT #3," time (secs)"," % reaction complete"
3100 FOR I=1 TO M
3105 READ Y(I,1),Y(I,2): REM time, % completion of reaction
3110 LET Y(I,1)=Y(I,1)*60: REM convert to seconds
3115 LET Y(I,3)=1/(1+.01*Y(I,2)):REM to approximate relative error
3120 PRINT Y(I,1),Y(I,2)
3125 PRINT #3,Y(I,1),Y(I,2)
3130 NEXT I
3135 LET Z4=.01176: REM  a = moles / litre (CH3)X to start
3140 LET Z5=.1036: REM  b = moles / litre (OH-) to start
3145 FOR J=1 TO 2
3150 LET O(J,1)=-100: REM no bounds
3155 LET O(J,2)=100
3160 LET O(J,3)=1: REM not masked
3165 NEXT J
3170 LET I5=0: REM no masks
3175 PRINT "use small values for parameters in this problem"
3180 PRINT "  reported answers at 2.45E-5, 8.55E-3"
3185 PRINT #3,"use small values for parameters in this problem"
3190 PRINT #3,"  reported answers at 2.45E-5, 8.55E-3"
3195 RETURN
3200  DATA  0 , 0
3201  DATA  3.5 , 16.8
3202  DATA  8.5 , 36
3203  DATA  13.5 , 50.7
3204  DATA  19.5 , 64.3
3205  DATA  26.5 , 75.2
3206  DATA  37.5 , 85.1
3207  DATA  999999 , 100: REM this data point cannot be satisfied
3500 LET I3=0
3505 IF B(1)+B(2)*Z5 = 0 THEN 3600
3510 LET Z1=B(2)*Z4/(B(1)+B(2)*Z5): REM beta in mhkin residual
3515 LET Z2=B(1)+B(2)*(Z5-Z4): REM alpha
3520 LET Z3=Y(I,1): REM time t
```

```
3525 IF -Z2*Z3>40 THEN 3600
3540 LET Z8=(1-EXP(-Z2*Z3))/(1-Z1*EXP(-Z2*Z3)): REM x/a
3550 LET R1=100*Z8-Y(I,2): REM residual
3555 LET R1=R1*Y(I,3): REM scale to get relative error
3560 RETURN
3600 LET I3=1: REM avoid zero divide
3610 RETURN
4000 LET Z1=B(2)*Z4/(B(1)+B(2)*Z5): REM beta in mhkin residual
4010 LET Z2=B(1)+B(2)*(Z5-Z4): REM alpha
4020 LET Z3=Y(I,1): REM time t
4025 LET Z0=EXP(-Z2*Z3)
4030 LET Z8=(1-Z0)
4040 LET Z9=(1-Z1*Z0)
4050 LET Z7=B(1)+B(2)*Z5: REM denominator of beta
4060 LET D(1)=Z3*Z0/Z9: REM first term of derivative
4070 LET D(1)=D(1)-Z8*Z0*(B(2)*Z4/(Z7*Z7)+Z1*Z3)/(Z9*Z9)
4080 LET D(2)=Z3*Z0*(Z5-Z4)/Z9
4090 LET D(2)=D(2)-Z8*Z0*(Z1*Z3*(Z5-Z4)+(Z5*B(2)/Z7-1)*Z4/Z7)/(Z9*Z9)
4100 LET D(1)=100*D(1)*Y(I,3)
4110 LET D(2)=100*D(2)*Y(I,3)
4120 RETURN
```

In dealing with the above problem, we used the solution to a differential equation (16-5-1). In the general form of concurrent reactions, we may have several differential equations, since there may be several reactions progressing simultaneously which interact with each other in certain ways. A particularly important class of problems has the name _compartment models_. Here we think of several reagents each occupying a "compartment" with reactions taking place to create or consume these reagents much like pipes into or out of the compartments. Each "pipe" or reaction has its own rate law and parameters. Generally we will want to estimate some or all of these parameters based on observations of one or more of the reagents or "compartments".

The functional form of our model is then a _system_ of differential equations, which may or may not be solvable analytically. If the differential equations are linear, then a formal solution can usually be written down in terms of the

matrix exponential (Moler and Van Loan, 1978). This is unfortunately computed only via numerically unstable techniques, which involve in their most benign form the computation of matrix eigenvalues. Clearly, the necessity of invoking a routine to compute matrix eigenvalues at each iteration of our parameter estimation routines is a grave nuisance. Indeed, given the lack of a true subroutine in BASIC, and also the choice we have made to direct our work to programs of relatively modest size, problems requiring the use of either numerical integration of differential equations or numerical linear algebra are considered beyond the scope of this book. This is not to say that there are not important problems and methods in this area, but the interested reader must be referred elsewhere (Bates and Watts, 1985; Bates, Wolf and Watts, 1985; Jennrich and Bright, 1976).

COVER SHEET

Chapter 17

Chapter title: Applications in economic modeling and forecasting

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

17

APPLICATIONS IN ECONOMIC MODELING AND FORECASTING

17-0. Economic Modeling and Forecasting

In this chapter we examine a number of problems in economic modeling and forecasting which give rise to nonlinear parameter estimation tasks.

Econometric models attempt to provide a mathematical description of an economy or a sector thereof which can be used to study the evolution of trends or responses to policy changes or external shocks. In this exposition we will concentrate on the aspects of this subject which give rise to parameter estimation problems. For discussions of the broader issues, we refer the reader to works on the subject of econometric modeling (Johnston, 1972; Intrilligator, 1978).

17-1. Estimation of an Econometric Model

Dr. Constantine Kapsalis has developed a macroeconomic model of the Canadian economy intended to illustrate econometric techniques on a microcomputer. Kapsalis (1985) follows a conventional path in building his model:

1. equations describing the behavior of the economy of interest are specified;
2. data is collected for the relevant variables;
3. the parameters in the equations specified are estimated one at a time using Ordinary Least Squares (OLS) as the estimation criterion.

A fourth step, simulation of the model over a trial period for which data is available, should be carried out to validate the model. Here we shall focus on the third step, and alternatives to the conventional OLS estimation methods.

Kapsalis' model uses the following equations

(17-1-1) $CONS(t) = B(1) + B(2) * GNP(t)$

(17-1-2) $INVT(t) = B(3) + B(4) * [GNP(t) + GNP(t-1) + GNP(t-2)] / 3$

(17-1-3) $EXPT(t) = B(5) + B(6) * USGNP(t)$

(17-1-4) $\log(IMPT(t)) = B(7) + B(8) * \log(GNP(t))$

(17-1-5) $GNP(t) = CONS(t) + EXPY(t) - IMPT(t) + INVT(t) + GOVT(t)$

where t indicates the time period, and the variables are described in Table 17-1-1.

Table 17-1-1. Variables in the Kapsalis model

Y(.,1) = CONS	=	consumption
Y(.,2) = INVT	=	investment
Y(.,3) = GNP	=	gross national product
Y(.,4) = EXPT	=	exports
Y(.,5) = IMPT	=	imports
Y(.,6) = USGNP	=	United States gross national product
Y(.,7) = GOVT	=	government expenditure on goods and services

Data for all the variables (in millions of Canadian dollars) are included within the program code segment in Listing 17-1-1. These have been drawn directly from Kapsalis (1985) for each year from 1970 to 1982.

Equation (17-1-5) is an identity relating variables in the model. It contains no parameters to be estimated. Equation (17-1-4) has been presented in a form by which the parameters can be estimated by linear regression methods. This implies, however, that we believe that errors in this equation are of the form

(17-1-6) $Error1(t) = B(7) + B(8)*\log(GNP(t)) - \log(IMPT(t))$

This means that the model of imports can be made exact by the multiplicative error term exp(Error1(t)), that is, by writing equation (17-1-4) as

(17-1-7) $IMPT(t) = \exp\{B(7) + B(8)*\log(GNP(t)) - Error1(t)\}$
$= GNP(t)^{B(8)} * \exp\{B(7) - Error1(t)\}$

Use of an additive error term leads to the form

(17-1-8) $IMPT(t) = B'(7) * GNP(t)^{B(8)} - Error2(t)$

where

(17-1-9) $B'(7) = \exp(B(7))$

The methods of this book can be used to estimate equation (17-1-8) by the Ordinary (though not Linear) Least Squares criterion. However, we shall extend the estimation process to demonstrate a more general procedure which allows the entire set of model parameters in a single calculation.

First, we define a generalized residual vector as the concatenation of the residual vectors of the individual equations. That is,

(17-1-10) $\underline{r}^T = (\underline{r1}^T, \underline{r2}^T, \underline{r3}^T, \underline{r4}^T)$

where

(17-1-11) $\underline{r1} = B(1) + B(2) * \underline{GNP} - \underline{CONS}$

(17-1-12) $\underline{r2} = B(3) + B(4) * (\underline{GNP} + \underline{GNP'} + \underline{GNP''}) / 3 - \underline{INVT}$

(17-1-13) $\underline{r3} = B(5) + B(6) * \underline{USGNP} - \underline{EXPT}$

(17-1-14) $\underline{r4} = B'(7) * \underline{GNP}^{B(8)} - \underline{IMPT}$

We have used GNP' and GNP" to indicate the series GNP lagged one and two periods, respectively. Note that this requires us to decide whether or not r2 is to have two less elements than r1, r3 and r4. If r2 is to have full length, data elements GNP(1968) and GNP(1969) are needed.

Further possibilities for choice concern weightings for the residuals. Note that we have combined residuals equation by equation. We could also combine residuals by time period, and indeed weightings could be applied by time period to diminish the influence of time periods considered less "important" in describing the behavior of the economy being modeled. This flexibility is useful in allowing the economist to incorporate knowledge and understanding of the economy not present in the numbers and equations. However, it introduces another source of subjectivity into an exercise where we may be trying to move to more quantitative and objective models and descriptions of economic phenomena.

In Listing 17-1-1 we present the program code needed to estimate the Kapsalis model under a least squares loss function applied to the residuals (17-1-10). Here we have applied a weighting in case the original data does not perfectly satisfy the identity (17-1-5). The weighting we have used for time period t is

(17-1-15) weight(t) = 1E-4/{1+abs(deviation(t))}

where deviation(t) is the difference between the left and right hand sides of Equation (17-1-5) for time period t. In the present example, the deviations are all zero for the given data. The factor of 1E-4 may be important in avoiding overly large numbers in the function and derivative calculations which may lead to overflow or underflow in the calculations within our estimation methods. We have also scaled the parameters, using as a guide the results obtained by Kapsalis from Ordinary Least Squares. Note that the particular form of the loss function forces a modification to RESSAVE (Section 13-9) if we wish to save data for

plotting.

Table 17-1-2 presents some results from estimation of the Kapsalis model using our methods. Here we would like to suggest the methods TN or CG since most econometric models involve many more parameters than the present example, and the majority of the equations are linear or "nearly linear" in the parameters. Unfortunately, the one nonlinear modeling equation, that is, Equation (17-1-4) and its transformation, Equation (17-1-8), has a variety of parameters which give similar contributions to the loss function. TN and CG both converge quite slowly for this problem. Some advantage may be made of masks to fix all parameters but B(7) and B(8) so that they may be estimated approximately first. CG was not allowed to execute to termination, even in a double precision compiled version. TN seems to perform better in double precision here, but is still slow compared to MRT and VM. Even NM does quite well by comparison.

A plot of Imports versus Income implies an almost linear relationship, which may explain why the exponential form causes some convergence difficulties. In a large scale model, diagnostic tools to help the user to find such situations before they cause a waste of computational effort would be very desirable. However, to date, the approach presented here of estimating the entire model in one calculation does not appear to have been seriously considered by econometricians, though it was proposed some time ago (Nash, 1980). For large scale econometric modeling, we feel that a program code generator is advisable to convert expressions of model equations familiar to economists into the program statements needed by our estimation methods.

As a final comment, we note that the Kapsalis model is separable in the sense that the four equations may each be estimated separately. In the present example, this may be accomplished by appropriate use of masks, though readers should be aware that our methods all work by trying to find point where the loss function value is less than that at the current "best" point. The sensitivity of the parameters is such that when the overall loss function is quite large in absolute value the function value comparison does not detect a smaller loss function because it has been evaluated in finite length arithmetic. That is, large residuals from one equation hide the relatively smaller residuals from another, with the outcome that the equation with smaller residuals is only approximately estimated. The deviation of the parameters from their "best" values is most pronounced for the troublesome Equation (17-1-4) discussed above. Indeed, if we estimate the Kapsalis model one equation at a time using masks, the order in which the equations are estimated will influence the parameter values obtained unless at some point in the overall process all parameters are freed.

Listing 17-1-1. KAPESTW.RES -- Estimation of the Kapsalis Econometric Model

```
 30 DIM Y(20,11),B(8),X(8),O(8,3),Z(13)
 3000 PRINT "KAPESTW - KAPSALIS MODEL ESTIMATION - 851221"
 3001 PRINT #3,"KAPESTW - KAPSALIS MODEL ESTIMATION - 851221"
 3005 LET P$="KAPEST - KAPSALIS MODEL ESTIMATION - 851221"
 3010 LET N=8: REM 8 parameters
 3020 LET L0=13: REM 13 time periods
 3030 LET M=L0*4: REM 4 equations to estimate
 3035 LET L2=4: REM 4 equations
 3040 PRINT "data for estimation"
 3041 PRINT #3, "data for estimation"
 3045 LET Z1=7: REM 7 data series + 4 storage column for the residuals
 3050 RESTORE 3300: REM pointer to data area
 3055 FOR J=1 TO Z1
 3060 READ X$
 3065 PRINT X$
 3066 PRINT #3, X$
 3070 FOR I=1 TO L0
 3075 READ Y(I,J)
 3080 PRINT Y(I,J);
 3081 PRINT #3, Y(I,J);
 3085 IF 5*INT(I/5)=I THEN PRINT
 3086 IF 5*INT(I/5)=I THEN PRINT #3,
```

```
 3090 NEXT I
 3095 PRINT
 3096 PRINT #3,
 3100 NEXT J
 3105 PRINT "weightings of observations by time period"
 3106 PRINT #3, "weightings of observations by time period"
 3110 FOR I=1 TO 13
 3115 LET Z(I)=Y(I,1)+Y(I,3)-Y(I,4)+Y(I,2)+Y(I,7)-Y(I,5):
       REM identity residual
 3120 LET Z(I)=1E-4/(1+ABS(Z(I))):
       REM downweight period if identity violated
 3125 PRINT Z(I);
 3126 PRINT #3, Z(I);
 3130 IF 5*INT(I/5)=I THEN PRINT
 3131 IF 5*INT(I/5)=I THEN PRINT #3,
 3140 NEXT I
 3145 PRINT
 3146 PRINT #3,
 3150 FOR J=1 TO N
 3160 LET O(J,1)=-1000
 3170 LET O(J,2)=1000
 3180 LET O(J,3)=1
 3190 NEXT J
 3192 LET O(7,1)=.01: REM lower bound for exponent
 3194 LET O(7,2)=4: REM maximum fourth power
 3200 RETURN
 3300 DATA "consumption - column Y(.,1)"
 3302 DATA 51526,55616,59841,63879,67160,70645,75180
 3304 DATA 77009,79038,80607,81431,82961,81206
 3306 DATA "investment - column Y(.,2)"
 3308 DATA 18994,21192,22470,25731,28337,26409,29129
 3310 DATA 27966,27714,31214,29938,33233,25558
 3312 DATA "exports - column Y(.,3)"
 3314 DATA 21223,22181,23655,26156,25620,23993,26304
 3316 DATA 28233,31207,32141,32753,33685,33152
 3318 DATA "imports - column Y(.,4)"
 3320 DATA 20588,22016,24489,27824,30538,29684,32274
 3322 DATA 32798,34291,36662,35915,37286,33072
 3324 DATA "GNP - column Y(.,5)"
 3326 DATA 88805,95342,100407,107737,111163,112762,119937
 3328 DATA 122709,126339,130050,131139,135646,130019
 3330 DATA "US GNP - column Y(.,6)"
 3332 DATA 1042243,1077573,1138537,1204205,1196525,1182412,1246352
 3334 DATA 1314996,1381144,1420315,1416091,1453341,1426075
 3336 DATA "government spending - column Y(.,7)"
 3338 DATA 17650,18369,18930,19795,20584,21399,21598,22299
 3340 DATA 22671,22750,22932,23053,23175
 3500 REM kapest residuals
 3505 LET I3=0
 3510 LET R1=0: REM start with zero
 3520 LET L3=1+INT((I-1)/L0): REM index of equation in use
 3530 LET L4=I-L0*(L3-1): REM index of time point
```

```
3535 IF L4<3 THEN 3800: REM to avoid lags
3540 ON L3 GOTO 3550,3600,3650,3700
3550 LET R1=.1*B(1)*Y(L4,5)-10000*B(2)-Y(L4,1)
3560 GOTO 3800: REM end equation 1
3600 LET R1=1000*B(3)+.1*B(4)*(Y(L4,5)+Y(L4-1,5)+Y(L4-2,5))/3
3605 LET R1=R1-Y(L4,2)
3610 GOTO 3800: REM end equation 2
3650 LET R1=-10000*B(5)+.01*B(6)*Y(L4,6)-Y(L4,3)
3660 GOTO 3800:REM end equation 3
3700 LET R1=(Y(L4,5)^B(7))/EXP(B(8))-Y(L4,4)
3710 GOTO 3800: REM end equation 4
3800 LET R1=R1*Z(L4): rem for weighting if desired
3805 LET Y(L4,L3+7)=R1: REM save residual
3810 RETURN
4000 REM kapsalis estimation derivatives
4010 LET L3=1+INT((I-1)/L0): REM index of equation in use
4020 LET L4=I-L0*(L3-1): REM index of time point
4030 FOR Z5=1 TO 8
4040 LET D(Z5)=0
4060 NEXT Z5
4070 IF L4<3 THEN RETURN
4080 ON L3 GOTO 4100,4200,4300,4400
4100 LET D(1)=.1*Y(L4,5)*Z(L4)
4110 LET D(2)=-10000*Z(L4)
4120 RETURN
4200 LET D(3)=1000*Z(L4)
4210 LET D(4)=.1*(Y(L4,5)+Y(L4-1,5)+Y(L4-2,5))*Z(L4)/3
4220 RETURN
4300 LET D(5)=-10000*Z(L4)
4320 LET D(6)=.01*Y(L4,6)*Z(L4)
4330 RETURN
4400 LET Z6=(Y(L4,5)^B(7))
4410 LET D(7)=LOG(Y(L4,5))*Z6*Z(L4)/EXP(B(8))
4420 LET D(8)=-Z6*Z(L4)/EXP(B(8))
4430 RETURN
4500 PRINT "RESIDUALS BY EQUATION"
4510 PRINT #3,"RESIDUALS BY EQUATION"
4520 PRINT "Equation:    1 ";TAB(27);"2";TAB(41);"3";TAB(55);"4"
4530 PRINT #3,"Equation:     1 ";TAB(27);"2";TAB(41);"3";TAB(55);"4"
4540 PRINT "Time   Consumption";TAB(23);"Investment";TAB(37);"Exports";
4541 PRINT TAB(51);"Imports"
4550 PRINT #3, "Time   Consumption";TAB(23);"Investment";
4551 PRINT #3, TAB(37);"Exports";TAB(51);"Imports"
4560 PRINT
4570 PRINT #3,
4580 FOR L4=1 TO L0: REM time periods
4582 PRINT L4;
4583 PRINT #3,L4;
4590 FOR L3=1 TO L2: REM loop over equations
4610 PRINT TAB(14*L3-6);Y(L4,L3+7);
```

```
4620 PRINT #3,TAB(14*L3-6);Y(L4,L3+7);
4630 NEXT L3
4640 PRINT
4650 PRINT #3,
4660 NEXT L4
4670 PRINT
4680 PRINT #3,
4690 RETURN
```

Table 17-1-2. Results of estimating the Kapsalis model

	VM	MRT	Kapsalis	NM	TN	TNDP
	6.925533	6.925534	7.108	6.922131	6.925323	6.92500046915343
	.9153634	.9153651	1.143687	.9112113	.9166269	.9156323649775526
	9.60504	9.605069	9.605066	9.60746	9.597612	9.565672524346011
	1.56723	1.567227	1.567	1.567076	1.567659	1.570566033173139
	1.395444	1.395441	1.160820	1.396167	1.395736	1.393990297552926
	3.271196	3.271194	3.1	3.271722	3.271157	3.272129415546971
	1.250913	1.250914	1.414	1.206034	.9796238	1.226846014948644
	4.2569	4.256922	6.174	3.73109	1.078828	3.974892547557808
loss function value corresponding to the above parameters						
	.7339538	.7339541	.7705057	.7356609	.7972375	.7344364663534258
secs	110	78		998	442	706
grads	36	12		0	254	272
fns	60	21		1687	72**	49**

** too many iterations - abnormal end

17-2. Simulation of Econometric Models

The second major phase of econometric modeling uses the equations and parameters estimated as Section 17-1 to simulate the behavior of the economy. This allows the model to be checked if data are available for the variables to be predicted - - the endogenous variables. Alternatively, the future values of these variables may be forecast. In order to make such forecasts, the values of some external, or exogenous, variables must be supplied. For the Kapsalis model of the previous section, the exogenous variables are

USGNP and GOVT. The endogenous variables are then CONS, INVT, EXPT, IMPT and GNP. Equations (17-1-1) through (17-1-5) provide five (nonlinear) equations in five unknowns. Since the methods in this book all use the vector $\underline{B}$ to denote the "unknowns" or values to be estimated, there is a serious potential for confusion with the coefficients $\underline{B}$ in the equations given in the previous section. Those coefficients must now be transferred to the data array Y(,) in our programs.

In order to determine the simulated values of the endogenous variables for time period t, we define

(17-2-1) $\underline{B}^T$ = (CONS(t), INVT(t), EXPT(t), IMPT(t), GNP(t))

If we can supply values for GNP(t-1), GNP(t-2) and the exogenous values GOVT(t) and USGNP(t), $\underline{B}$ may be found by solving the simultaneous Equations (17-1-1) through (17-1-5), if such a solution exists.

The traditional approach to such solutions is a functional iteration. Initial guesses to the endogenous variables, that is, $\underline{B}$, are provided. From GNP(t) = B(5) and Equation (17-1-1), a new value of CONS(t) = B(1) is found and immediately substituted in vector $\underline{B}$. The second equation is then used to get a new value for B(2) = INVT(t), vector $\underline{B}$ is revised, then Equation (17-1-3) used to update B(3) = EXPT(t) and so on. If all the equations are <u>linear</u>, this is precisely the Gauss-Seidel iteration (Dahlquist and Bjorck, 1974, p.189). An alternative scheme is to use one set of parameters $\underline{B}$ in all the equations to find a new set $\underline{B}$', rather than using the new values as soon as they are available. This gives rise to the Jacobi iteration for the solution of linear systems of equations (Dahlquist and Bjorck, 1974, p. 189). However, in econometrics, the Jacobi " and Gauss-Seidel processes are applied to nonlinear sets of equations. Clearly, we can use the techniques presented in

earlier chapters to minimize the residuals of such sets of equations under some loss function or other. The most obvious is the least squares criterion, which allows us to apply all six methods available, and also allows for relatively straightforward evaluation of the necessary derivatives for the gradient methods.

The code and results for this example are not included here. Readers interested in another example are referred to a recent paper by the authors where the Klein economic model is both estimated and simulated using the methods of the last two sections (Nash and Walker-Smith, 1986).

17-3. Production Functions

One of the more common nonlinear models in economics is the class of models called <u>production functions</u> (see Intrilligator, 1983, pages 251-302). One of these was the troublesome Equation (17-1-4) in the Kapsalis model. A typical form is the Cobb-Douglas production function which attempts to predict the production Q of a commodity using the capital, K, and labor, L, used in making it.

(17-3-1) $Q = B(1)\ K^{B(2)}\ L^{B(3)}$

This is commonly estimated by a logarithmic transformation to

(17-3-2) $\log(Q) = \log(B(1)) + B(2)\log(K) + B(3)\log(L)$

but this implies a multiplicative error rather than an additive one. A simple example, which was used as a classroom example and communicated to us by Prof. Pedro Arroja of the University of Ottawa, serves to illustrate the importance the choice of estimation approach. Listing 17-3-1 is an edited output which shows first the logarithmic estimation, which could have been accomplished in one iteration had the damping factor LAMBDA been zero, followed by the nonlinear estimation using the same data. To allow for easy comparison, we have used exp(B(1)) in place of B(1) to simplify the expressions slightly in the code.

In this example, we find that the parameter estimates change to the extent that one actually changes sign. However, the standard error estimate for this parameter, B(3) is large enough via either estimation approach that it may be dropped from the model as not being statistically significant. Listing 17-3-2 presents the code and data for this problem.

Listing 17-3-1. Estimation of a Cobb-Douglas production function.

```
MRT -- MARQUARDT/NASH -- 860412
ITN 0   EVAL'N 1  SS= 14.59573
parameters   1    1    1
LAMDA =  .00004
ITN 1   EVAL'N 2  SS= 2.989008E-02
parameters   2.261876    .8385177    .5455544
...  ELAPSED SECS= 69  AFTER 9  GRAD & 14  FN EVAL
CALCULATED FUNCTION MINIMUM =  2.597076E-02
PARAMETER CORRELATION ESTIMATES
ROW  1  :   1.00000
ROW  2  :  -0.88569  1.00000
ROW  3  :  -0.98840  0.80501  1.00000

SOLUTION WITH ERROR MEASURES AND GRADIENT OF SUM OF SQUARES
B( 1 ) =  5.416473   STD ERR =  3.071983   GRAD( 1 ) =  4.768372E-07
B( 2 ) =  .5634718   STD ERR =  .312824   GRAD( 2 ) =  1.326203E-06
B( 3 ) =  2.794833E-03   STD ERR =  .5295704   GRAD( 3 ) =  2.145767E-06

RESIDUALS
 8.928824E-02  1.027632E-02 -8.865738E-02 -3.110886E-02 -5.911732E-02
 2.732182E-02 -1.143503E-02 -1.075745E-03 -3.877163E-03  .0683856

calculate loss function for original un-logged data
*** changing to exponential form for residual ***
unlogged residuals
 72.4751  8.862915 -82.46564 -28.94568 -57.34674
 25.95355 -11.31384 -1.06781 -3.869995  66.3899
sum of squares =  21483.6
 another set of starting parameters ( [cr] = N ) Y
ENTER INITIAL VALUES FOR PARAMETERS ( [cr] = Y ) N
MRT -- MARQUARDT/NASH -- 860412
```

```
ITN 0   EVAL'N 1  SS= 21483.6
parameters   5.416473    .5634718    2.794833E-03
...  ELAPSED SECS= 49  AFTER 6  GRAD & 12  FN EVAL
CALCULATED FUNCTION MINIMUM =  21249.06
PARAMETER CORRELATION ESTIMATES
ROW  1  :   1.00000
ROW  2  :  -0.88092  1.00000
ROW  3  :  -0.98753  0.79554  1.00000

SOLUTION WITH ERROR MEASURES AND GRADIENT OF SUM OF SQUARES
B( 1 ) =  6.010979   STD ERR =  2.939307   GRAD( 1 ) = -5.539063
B( 2 ) =  .4847078   STD ERR =  .3024644   GRAD( 2 ) = -12.54688
B( 3 ) = -8.427461E-02   STD ERR =  .5063433   GRAD( 3 ) = -20.125

B( 1 )= 6.010979E+00  step= 2.08E-02  f-, f+  2.5003E+04  2.5162E+04
B( 2 )= 4.847078E-01  step= 1.69E-03  f-, f+  2.1409E+04  2.1411E+04
B( 3 )=-8.427461E-02  step= 3.03E-04  f-, f+  2.1266E+04  2.1266E+04
        best function value found is              2.1249E+04

radii of curvature for surface along axial directions
  & tilt angle in degrees

for B( 1 )  R. OF CURV. =  6.274E+03   tilt =  -89.98499
for B( 2 )  R. OF CURV. =  9.826E-01   tilt =  -89.84993
for B( 3 )  R. OF CURV. =  5.021E-06   tilt =  -84.09729

RESIDUALS
 80.87122  15.58179 -81.1084 -30.52716 -51.33667
 26.2713 -10.08472 -.9866943 -7.297974  58.94123
```

Listing 17-3-2. COBB.RES problem file.

```
30 DIM Y(10,4),B(3),X(3),O(3,3)
3000 PRINT "COBB DOUGLAS - Y=B(1)*K^B(2)*L^B(3) 860906"
3002 PRINT #3,"COBB DOUGLAS - Y=B(1)*K^B(2)*L^B(3) 860906"
3003 LET P$="COBB DOUGLAS -- COBB.RES 860906"
3010 LET N=3
3015 LET M=10
3018 LET O(1,1)=-1000: REM MULTIPLIER HAS LOOSE BOUNDS
3019 LET O(1,2)=1000
3020 LET O(1,3)=1
3022 FOR J=2 TO N
3023 LET O(J,3)=1: REM free parameters
3024 LET O(J,1)=-10: REM tight lower bound
3025 LET O(J,2)=10: REM tight upper bound
3026 NEXT J
```

```
3028 RESTORE 3300
3030 FOR I=1 TO M
3032 READ Y(I,1),Y(I,2),Y(I,3)
3033 PRINT Y(I,1),Y(I,2),Y(I,3)
3034 PRINT #3,Y(I,1),Y(I,2),Y(I,3)
3035 NEXT I
3038 FOR J=1 TO N
3039 LET B(J)=1: REM STARTING VALUES
3040 NEXT J
3050 INPUT "use log-log transformation ([cr] = exponential form) ";X$
3055 PRINT #3,"use log-log transformation ([cr] = exponential form) ";X$
3060 LET L9=0
3065 IF X$="Y" OR X$="y" THEN LET L9=1
3090 RETURN
3300 DATA 776,10.3,100
3310 DATA 858,10.7,99
3320 DATA 972,11.2,102
3330 DATA 945,11.8,101
3340 DATA 999,12.4,88
3350 DATA 937,12.9,91
3360 DATA 995,13.4,87
3370 DATA 993,13.6,87
3380 DATA 1000,13.7,90
3390 DATA 938,13.9,93
3500 LET I3=0:REM COBB DOUGLAS
3502 REM no safety check included
3505 IF L9=1 THEN 3530
3510 LET R1=EXP(B(1))*(Y(I,2)^B(2))*(Y(I,3)^B(3))-Y(I,1)
3520 RETURN
3530 LET R1=B(1)+B(2)*LOG(Y(I,2))+B(3)*LOG(Y(I,3))-LOG(Y(I,1))
3540 LET Y(I,4)=EXP(B(1))*(Y(I,2)^B(2))*(Y(I,3)^B(3))-Y(I,1)
3600 RETURN: REM note correction of above line
4000 LET I3=0: REM COBB DOUGLAS DERIVATIVES
4010 IF L9=1 THEN 4100
4020 LET D(1)=(Y(I,2)^B(2))*(Y(I,3)^B(3))*EXP(B(1))
4030 LET D(2)=D(1)*LOG(Y(I,2))
4040 LET D(3)=D(1)*LOG(Y(I,3))
4060 RETURN
4100 LET D(1)=1
4110 LET D(2)=LOG(Y(I,2))
4120 LET D(3)=LOG(Y(I,3))
4130 RETURN
4500 PRINT "calculate loss function for original un-logged data"
4504 PRINT #3, "calculate loss function for original un-logged data"
4520 LET Z9=0: REM for sum of squares
4530 LET L9=0: REM switch to unlogged form
4532 PRINT "*** changing to exponential form for residual ***"
4534 PRINT #3,"*** changing to exponential form for residual ***"
4540 PRINT "unlogged residuals"
4550 PRINT #3, "unlogged residuals"
```

```
4570 FOR L1=1 TO M
4600 LET Z9=Z9+Y(L1,4)*Y(L1,4)
4610 PRINT Y(L1,4);
4612 PRINT #3,Y(L1,4);
4620 IF 5*INT(L1/5)=L1 THEN PRINT
4622 IF 5*INT(L1/5)=L1 THEN PRINT #3,
4640 NEXT L1
4650 PRINT
4652 PRINT #3,
4660 PRINT "sum of squares = ";Z9
4662 PRINT #3, "sum of squares = ";Z9
4680 RETURN
```

Another model of a similar form is that suggested by Kmenta (1971), which Talpaz (1976) used as an illustration of a nonlinear estimation problem. The functional form of the model is

(17-3-3) $Q = B(1) \{B(2)\ K^{-B(4)} + (1-B(2)) * L^{-B(4)}\}^{(-B(3)/B(4))}$

where Q is the output, K the capital input, and L the labor input required in producing a commodity. This model can be estimated in this form once data is provided, though the derivative expressions are rather messy. Following Talpaz (1976), we could use a variable metric method and employ numerical approximation of derivatives. However, analytic derivatives may offer some advantage in cases such as this where exponential expressions are present and we expect rapid changes in the value of Q with relatively small changes in the parameters. To simplify our work, we replace B(3)/B(4) with B(3), so that our new model is

(17-3-4) $Q = B(1) \{B(2)\ K^{-B(4)} + (1-B(2)) * L^{-B(4)}\}^{-B(3)}$

Results of estimation based on these two forms is given in Table 17-3-1. Note that Kmenta (1971) used a linearization based on a Taylor expansion of model (17-3-3) to allow for ordinary (linear) least squares methods to be employed.

Table 17-3-1. Results of estimating the Kmenta-Talpaz model.
All calculations using our methods were run under GWBASIC on a Corona PC 21.

	Kmenta linear	Talpaz	Kmenta ML	VM+KTA	VM+KTB	MRT+KTB
B(1)	17.2624	11.2202	11.3868	11.21282	11.21311	11.21346
B(2)	.4723	.4052	.4064	.40560	.40486	.40531
B(3)**	.8245	.8270	.8222	.82729	.82708	.82719
B(4)	.4334	.5964	.6042	.59558	.59716	.59624
SumSquares	46437.18	979.96	984.15	979.956	979.956	979.956
Time (secs)		20(IBM 360/65)		1782	1138	242
Grad Evals		12		62	71	9
Fn Evals		n.a.		112	114	19

KTA = Model (17-3-3) with numerically approximated derivatives
KTB = Model (17-3-4)
ML = maximum likelihood (not specified as to form)

** parameter B(3) has been adjusted to be consistent with Model (17-3-3) if appropriate

17-4. Forecasting Problems

The examples in this chapter presented above all relate to economic models. These models are used in some cases for forecasting purposes, that is, for predicting the values of economic variables at time periods in the future. However, they also have another important use, which is to examine the consequences of economic policies or situations, sometimes called "shocks".

For purely forecasting purposes, a number of generally simpler techniques have been devised (Makridakis, Wheelwright and McGee, 1983). Here we shall not attempt to provide any systematic survey of these techniques, but only furnish a brief skeleton of some possible applications of our paremeter estimation methods to forecasting techniques. We have already published one study of function minimization methods

applied to such problems (Nash and Walker-Smith, 1986). Briefly, this considered applications to the following techniques in addition to the Klein economic model estimation and simulation already mentioned:

1. the estimation of Auto-Regressive/Moving Average (ARMA) models of the type generally approached via the Box-Jenkins methodology, which itself commonly uses a Marquardt method as the core algorithm;

2. the determination of the parameters in an exponential smoothing model for a time series, using as a particular example the Brown double exponential smoothing;

3. the estimation of the trend and seasonal factors in an additive seasonal decomposition model having either a linear or nonlinear trend component.

To complete this chapter, we present a problem in forecasting market share via a Markov model. The situation, which we have greatly simplified, considers that the market for a class of products is divided into those manufactured by company X and those of all other suppliers, which we will label A. Customers either buy from X or A. If we call x the fraction of the total customers buying from X, and consider the number of customers to be a fixed number N, then the quantity of product X must supply is determined by changes in x. One mechanism for change is to consider that purchasing decisions are made just once per period of interest, for example, once per year or once per month. This is not totally unrealistic for many products. Then we look at the probability that a customer moves from products of A to those of X, and vice-versa. These probabilities are called transition probabilities. For the present example, we need just two independent values,

```
(17-4-1)    P(X --> A) = B(1)
            P(A --> X) = B(2)
```

Clearly, we also have the probabilities

(17-4-2) P(X --> X) = 1 - B(1)

P(A --> A) = 1 - B(2)

for customers remaining loyal to their current supplier.

The proportion of customers with company X in time period t is x(t), so the market share for A must be (1 - x(t)). We get the market shares in the next time period by applying the transition probabilities to the current market shares. We will only look at the market share for X, which for time period (t+1) must be

```
(17-4-3)    x(t+1) = x(t) - x(t) * B(1) + (1 - x(t)) * B(2)
                   =   market share in time period t
                     - loss of market share to A
                     + gain in market share from A
```

Equation (17-4-3) simplifies to

(17-4-4) x(t+1) = B(2) + x(t) * (1 - B(1) - B(2))

In the general case where there are many companies, the market shares can be written as a vector $\underline{x}$, the transition probabilities can be written as a matrix T, and the model can be written in vector form as

(17-4-5) $\underline{x}(t+1) = T\ \underline{x}(t)$

which is subject to constraints on the elements of $\underline{x}$ (they must add up to 1) and on the elements of T (columns must add to 1, since the total probability of change of any one market share must be unity). If the elements of T do not change over time, then we say that the Markov model is <u>stationary</u>. This allows us to write

(17-4-6) $\underline{x}(t) = T^t\ \underline{x}(0)$

The initial values of $\underline{x}(0)$ must, of course, be provided to us in order that the values of the market shares in time t can be computed.

Returning to our simple model, which is stationary by virtue of the transition probabilities being parameters which

are fixed over time, we would like to find appropriate values of these parameters which fit a given set of market share data. Once fitted, the model can be easily extrapolated to provide forecasts. Note that we need just one more parameter,

(17-4-7) x(0) = B(3)

to complete the model.

Listing 17-4-1 presents the code for this model, including some sample data. A test data set which has been generated to have a very small residual sum of squares is included in REMark statements. MRT is able to solve this latter problem very rapidly from a variety of starting points, though there appears to be a local minimum near the point (.984047, 0.07987, 1). For the sample data, the solution has a residual sum of squares of approximately 5.3E-4 at (0, 0.0182, 0), that is, having two active bounds constraints and only one free parameter. In such situations MRT will often perform poorly, since it has no linear search sub-algorithm, and is ill-suited to minimizing a function of one parameter. VM, however, with a step-reduction and test at each major iteration, is able to make rapid progress when only one parameter is free.

Direct search methods are also reasonably effective for this problem. Moreover, we may easily alter the loss function to change the criterion of fit for our forecasting model. Table 17-4-1 shows some results obtained with NM and the sample data. In particular, the last set of results has been obtained with an asymmetric loss function. This adds three times the absolute deviation when the model exceeds the data value, but only one times the absolute deviation when the model (i.e. prediction) is less than the observed value. Such a loss function could be motivated in situations where a supplier does not wish to hold inventory, so that forecasts in excess of demand would have negative consequences.

Asymmetric loss functions have appeared with relative rarity in the statistical literature (see, however, Zellner, 1986, and references therein).

Listing 17-4-1. MARKET.RES problem file.

```
30 DIM Y(10,3),B(3),X(3),O(3,3)
3000 PRINT #3,"MARKET.RES - MARKOV CUSTOMER SIMULATION - 860303"
3005 LET P$="MARKET.RES 3 parameter Markov model"
3010 LET M=7: REM 7 data points
3020 LET N=3: REM 3 parameters
3050 RESTORE: REM reset data pointer
3052 PRINT "market share data"
3054 PRINT #3,"market share data"
3060 FOR I=1 TO M
3070 READ Y(I,1): REM market share at time point I
3075 LET Y(I,3)=I: REM define a time period index for plotting
3076 PRINT I,Y(I,1)
3077 PRINT #3,I,Y(I,1)
3080 NEXT I
3090 REM now set the bounds on the parameters
3095 FOR I=1 TO 3
3100 LET O(I,1)=0: REM positive transition probability or proportion
3102 LET O(I,2)=1: REM upper bound
3104 LET O(I,3)=1: REM not masked
3106 LET B(I)=0.02: REM reasonable initial values
3110 NEXT I
3120 PRINT " FUNCTION FORM   x(t+1) =  B(2) + x(t) (1 - B(1) - B(2))"
3122 PRINT
3124 PRINT #3, " FUNCTION FORM   x(t+1) =  B(2) + x(t) (1 - B(1) - B(2))"
3126 PRINT #3,
3130 PRINT "      x(0) = B(3)"
3132 PRINT #3,"      x(0) = B(3)"
3190 RETURN
3200 DATA 0.01,0.03,0.04,0.06,0.09,0.11,0.13
3202 REM following data is set up to have small residuals
3205 REM DATA 0.0590,0.1031,0.1428,0.1785,0.2107
3210 REM DATA 0.2396,0.2656
3220 REM parameters for this data set approximately (0.05,0.05,0.01)
3500 LET I3=0: REM start with computable function
3510 LET Z3=1-B(1)-B(2)
3520 LET Z9=B(3): REM y(i,2) contains x(i) model value
3530 FOR L=1 TO I: REM loop up to current time point
3540 LET Z9=B(2)+Z9*Z3
3550 LET Y(L,2)=Z9: REM to save for plotting
3560 NEXT L
3570 LET R1=Z9-Y(I,1)
3610 RETURN
```

```
4000 LET Z3=1-B(1)-B(2): REM MARKET.RES derivatives
4010 LET Z9=B(3): REM y(i,2) contains x(i-1) model value
4012 LET D(1)=0
4014 LET D(2)=0
4016 LET D(3)=1
4018 FOR L=1 TO I
4030 LET D(1)=D(1)*Z3-Z9
4050 LET D(2)=1+D(2)*Z3-Z9
4060 LET D(3)=D(3)*Z3
4120 LET Z9=B(2)+Z9*Z3
4190 NEXT L
4230 RETURN
```

Table 17-4-1. Estimation of the market share model under various loss functions using the Nelder-Mead code NM

Loss function	B(1)	B(2)	B(3)	Loss Function	Maximum Deviation
Sum of squared deviations	0	1.819439E-2	0	5.311885E-4	1.359608E-2
Sum of absolute deviations	2.68E-7	1.868546E-2	0	5.270168E-2	1.501545E-2
Maximum absolute deviation	1.27E-7	1.773517E-2	0	1.226748E-2	1.226748E-2
-8.19695E-3 -1.226728E-2					
Asymmetric sum ** of deviations	8.35E-7	1.534985E-2	0	9.779023E-2	2.737466E-2

** Sum is made up of 3 * (positive deviations) - 1 * (negative deviations)

COVER SHEET

Chapter 18

Chapter title: Test problems and other examples

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

18

TEST PROBLEMS AND OTHER EXAMPLES

18-0. Test Problems and Other Examples

In this chapter we collect together a number of test problems which we have used throughout the development of the ideas and software within the book. Some of the problems presented are truly test problems. That is, they are established specifically to provide a mechanism to exercise particular features of algorithms. Others are genuine parameter estimation problems which are used as "tests" because they are in some sense representative of the types of problems frequently encountered. To avoid overloading the reader with program code, all the listings for this chapter are presented in Appendix A in alphabetic order by the name of the problem file.

18-1. Simple Tests

In many situations it is useful to have quite simple tests which allow for rapid verification that a method functions acceptably on straightforward problems. After all, not all problems are difficult to solve.

LINTEST.RES

This problem asks the minimization method to solve a simple linear least squares problem. It is designed to allow us to determine how much extra work a nonlinear optimization code carries out to solve a problem for which linear algebraic calculations would suffice. Table 18-1-1 presents a summary of some results of applying the six parameter estimation methods to this problem. Note the importance of scaling the parameters to the success and efficiency of the calculations.

Table 18-1-1. Performance of the parameter estimation methods on the LINTEST.RES linear regression problem. Results are also given for the case where the problem code has no scaling factors applied. All calculations were performed with the BASICA interpreter on the Maestro PC (NEC V20 processor). Starting values have been adjusted so all problems were started from the same point (B(i)=1 in the unscaled form).

Method	Time (s)	Gradient Evaluations	Function Evaluations	Function Value
HJ	1325	0	1422	29.43522
NM	210	0	158	29.44211
CG	72	13	34	29.42403
TN	69	22	8	29.43691
VM	47	7	21	29.42323
MRT	55	6	11	29.42326

Table 18-1-1 (continued) Performance of the parameter estimation methods on the LINTEST.RES problem.

On unscaled problem, we obtain the results

Method	Time (s)	Gradient Evaluations	Function Evaluations	Function Value	
HJ	>2650	0	3027	>30	

stopped manually due to excessive run time

NM .1	153	0	120	734.0398	<
.1	+176	0	257	29.42335	
CG	2267	470	1089	37.74041	<
	+126	494	1152	37.24775	<
	+79	508	1194	36.7742	<

not continued from this point due to excessive run time

In double precision, however,

CG(DP)		4	14	29.42328944945385	
	3829	678	1567	29.42328938971049	
TN	6	1	1	553636.4	<

(The Hessian appeared to be singular in this case. Starting from the lower point found by POSTGEN caused repetition of the termination with the message 'UT A U = 0'.)

In double precision, however,

TN(DP)	192	53	19	29.42339246577917	

(Stop message:'ABNORMAL END -- too many major cycles')

VM	80	11	41	29.42322	
MRT	38	5	5	29.42326	

< implies a lower function value is found by POSTGEN

- -

QUADSN.FN

This problem presents a method with a simple quadratic loss function, similar in nature to that in LINTEST, but having a negative value at the minimum. The problem was communicated to us by Stephen Nash (private communication).

The results of applying the five function minimization methods (MRT cannot be applied because this is not a least squares problem) to QUADSN have been presented in Chapter 12 in Table 12-2-5b and Figure 12-2-2.

ABS.FN

This problem minimizes the sum of the absolute values of the parameters. It is designed to allow us to examine the effect of discontinuity in the gradient at the minimum. Furthermore, since the Hessian is a null matrix in this case (the gradient is constant except for the discontinuities whenever one of the parameters is zero), we would expect the Newton-like methods to do poorly. Indeed, TN is unable to make any progress. Results are presented in Table 18-1-2. In all cases the starting values for the parameters are set to 3.333.

BT.RES

This test problem is designed to allow the bounds and masks facilities in our programs to be tested. The loss function involved is simply the sum of squares of the parameters, that is,

$$(18\text{-}1\text{-}1) \qquad f(\underline{B}) = \sum_{j=1}^{n} (Y(j,1) * (B(j) - Y(J,2)))^2$$

Table 18-1-2. Performance of the five function minimization methods on the ABS.FN test problem of order 5. All calculations were performed with the BASICA interpreter on the Maestro PC (NEC V20 processor). The starting stepsize for NM is given following the 'NM'.

Method		Time (s)	Gradient Evaluations	Function Evaluations	Function Value	
HJ		71	0	383	2.125744E-07	
NM	.1	482	0	517	9.88846	<
	.1	+416	0	955	0.4758012	<
	.1	+264	0	1233	5.271324E-04	<
	.1	+324	0	1571	5.52028E-06	
CG		32	15	32	9.576638E-08	
TN		5	2	1	16.665	<
		+5	4	2	16.65348	<
		+4	6	3	16.642	<
		(abandoned at this point -- all function value reduction due to POSTGEN axial search)				
VM		107	19	81	1.552806E-06	

< implies a lower function value is found by POSTGEN

However, we apply a set of lower and upper bounds to the parameters and a mask (i.e. fixed value) to one of them. Table 18-1-3 presents the results of applying the six parameter estimation methods to this problem. The bounds are chosen so that the starting stepsize of the Hooke and Jeeves code does not "accidentally" stumble upon the minimum. Even so, the efficiency of HJ is surprisingly good on this problem. As expected, due to the simple mechanism used to satisfy bounds, NM does poorly. We do not recommend NM when many of these constraints are expected to be active at the minimum. There are nine parameters in the standard problem suggested, and the methods which must build and access arrays (VM and MRT) are slower than the methods using vectors only (CG and TN).

Table 18-1-3. Performance of the six minimization methods on the bounds test problem BT.RES. All calculations were performed with the BASICA interpreter on the Maestro PC (NEC V20 processor). The starting stepsize for NM is given following the 'NM'.

Method	Time (s)	Gradient Evaluations	Function Evaluations	Function Value
HJ	45	0	101	152.5432
NM 0.1	342	0	164	201.9753
lower function value found during axial search				
0.1	+386	0	350	152.5432
CG	25	5	4	152.5432
TN	39	8	4	152.5432
VM	45	4	4	152.5432
MRT	91	4	4	152.5432

18-2. Problems from S. G. Nash

The following two problems were communicated to us by S.G.Nash of the Mathematical Sciences Department of Johns Hopkins University.

GENROSE.FN and GENROSEJ.RES

The banana-shaped valley of Rosenbrock (1960) is defined as

$$(18\text{-}2\text{-}1) \quad f(\underline{B}) = 100\ (B(2) - B(1)^2) + (1 - B(1))^2$$

S.G. Nash (1982) reports the extension of this to an arbitrary number of parameters. This function is

$$(18\text{-}2\text{-}2) \quad f(\underline{B}) = 1 + \sum_{j=2}^{n} (100 * (B(j) - B(j-1)^2)^2 + (1 - B(j)^2)$$

Note that the minimum of this function is 1 rather than 0, which ensures that it cannot be computed with particular accuracy at the solution.

The function in Equation (18-2-2) can also be written as a nonlinear least squares problem. Rather than detail the residual functions, we shall refer the reader directly to the program code GENROSEJ.RES in Appendix A, which allows this generalized Rosenbrock test problem to be set-up and solved by method MRT.

Performance of our methods on these test problems has been presented in Table 12-2-5a and Figure 12-2-1.

CAL4.FN

This function is a sum of various powers of the parameters or the parameters minus a constant. A parameter may be varied in the function, and the number of parameters is also variable. We refer the reader to the code in Appendix A for details.

18-3. WOODR.RES and WOOD.FN - A Traditional Test

This problem is an extension of the well-known banana-shaped valley test function of Rosenbrock (1960) and has been widely used (Dixon, 1973; Jacoby, 1972; Nash, 1976). There are four parameters and Jacoby (1972) gives a bounds-constrained variant of the problem where each parameter must lie in the closed interval [-10,10]. In our experience, however, these bounds are rarely encountered. In particular, from the traditional starting point (-3,-1,-3,-1), our methods do not follow trajectories in the parameter space which touch the bounds.

The function is usually written

(18-3-1) $$f(\underline{B}) = 100\,(B(2) - B(1)^2)^2 + (1-B(1))^2 + 90\,(B(4) - B(3)^2)^2 + (1-B(3))^2 + 10.1\,\{(B(2) - 1)^2 + (B(4) -1)^2\} + 19.8\,(B(2) - 1)\,(B(4) -1)$$

which has the value 19192 at the traditional starting point and value 0 at (1,1,1,1).

Another view of the WOOD function is as a nonlinear least squares problem (Nash,1976,Problem WD4). This has residuals

(18-3-2)
$$\begin{aligned} r(1,\underline{B}) &= 10 * (B(2) - B(1) * B(1)) \\ r(2,\underline{B}) &= 1 - B(1) \\ r(3,\underline{B}) &= (B(4) - B(3) * B(3)) * \mathrm{SQRT}\ (90) \\ r(4,\underline{B}) &= 1 - B(3) \\ r(5,\underline{B}) &= (B(2) + B(4) - 2) * \mathrm{SQRT}(10) \\ r(6,\underline{B}) &= (B(2) - B(4)) * \mathrm{SQRT}(0.1) \end{aligned}$$

This latter form allows the problem to be attempted with all our methods, including MRT. In some cases, it may be preferable to evaluate the function as in Equation (18-3-1).

This problem is "difficult" primarily because the Hessian is computationally singular at the solution. This is easily demonstrated by actually computing the Hessian matrix at the solution point (1, 1, 1, 1), which is

(18-3-3) $$H = \begin{pmatrix} 802 & -400 & 0 & 0 \\ -400 & 220.2 & 0 & 19.8 \\ 0 & 0 & 720 & -360 \\ 0 & 19.8 & -360 & 200.2 \end{pmatrix}$$

for which the computed eigenvalues are (using LEQB05, Nash, 1984b) 1005.925, 905.6054, 32.15015 and 0.7195693. The ratio of largest to smallest eigenvalues exceeds 1000, and we may expect that gradient methods, particularly if operated in single precision, will compute imprecise search directions.

18-4. NASHEASY.FN - A Large but "Easy" Problem

To illustrate the effectiveness of conjugate gradient methods in minimizing certain nonlinear functions, one of us devised the following nonlinear function

(18-4-1) $$f(\underline{B}) = \sum_{j=1}^{n} z(j)^2 + \prod_{j=1}^{n} [\exp(-z(j)^2) - 1]^2$$

where

(18-4-2) $z(j) = B(j) - j$

Despite its appearance, this test problem is relatively "easy" for our methods to solve.

18-5. NASHHARD.FN - A Large but Awkward Problem

The previous example is relatively straightforward to solve because the parameters have quite limited interactions with each other in contributing to the function value. A much more difficult problem is the function that is evaluated as

(18-5-1) $$f(\underline{B}) = \sum_{j=1}^{n} (B(j) - j + 1)^2\ (B(j+1) - j)^2$$

where we define

(18-5-2) $B(N+1) = B(1)$ and $B(0) = B(N)$

for simplicity of our expressions. The gradient component for parameter B(k) is therefore

(18-5-3) $$G(k) = 2 * (B(k) - k+1) * [(B(k+1) - k)^2 + (B(k-1) - (k+2)^2]$$

where the definitions (18-4-2) are again employed.

This problem is more difficult than that presented in Section 18-4 because the Hessian matrix is singular at the solution. From (18-4-3) we can compute the elements of the Hessian matrix as

(18-5-4) $H(k,k) = 2 [(B(k+1) - k)^2 + (B(k-1) - k+2)^2]$

$H(k,k+1) = 4 (B(k) - k+1) (B(k+1) - k)$

which gives H(N,1) and H(1,N) also.

By symmetry, (18-5-5) gives the subdiagonal elements of the Hessian. All other elements of the Hessian are zero. However, as the parameters approach the solution values, that is,

(18-5-6) $B(j) = j - 1$

the gradient and Hessian become null. This causes particular difficulties for methods which make use of the Hessian, namely TN and VM.

18-6. LONION.RES - A yield-density model problem

In Section 1-3 we introduced data (Ratkowski, 1983) for yield/density modelling of White Imperial Spanish Onions at Purnong Landing (Table 1-3-1). If we denote yield (g/plant) by y and density of plants (plants/metre squared) by x, then three potential models for the relationship between yield and density are:

(18-6-1) $y = (B(1) + B(2) * x)^{-1/B(3)}$
(Bleasdale and Nelder, 1960)

(18-6-2) $y = (B(1) + B(2) * x + B(3) * x^2)^{-1}$
(Holliday, 1960)

(18-6-3) $y = (B(1) + B(2) * x^{B(3)})^{-1}$
(Farazdaghi and Harris, 1968)

Note that since the density figures are of the order of 100, we should scale the parameters to avoid having estimates with widely differing magnitudes. The lack of scaling, plus the presence of parameters as exponents, makes these estimation problems difficult to solve. Indeed, under the most obvious loss function, the sum of squared residuals, they become extremely difficult. However, according to Ratkowsky (1983, page 51), the variance of log(y) is to be considered constant for this problem, suggesting a multiplicative stochastic error about the idealized model, or a loss function which attempts to minimize the sum of squares of the deviations.

(18-6-4) $r(i,\underline{B}) = \log(z(i,\underline{B})) - \log(y_i)$

Dropping the time-period index, i, for simplification, this gives the functional forms

(18-6-5) $r(\underline{B}) = -(\log(B(1) + B(2) * x))/B(3) - \log(y)$
(Bleasdale and Nelder)

(18-6-6) $r(\underline{B}) = - \log(B(1) + B(2) * x + B(3) * x^2) - \log(y)$
(Holliday)

(18-6-7) $r(\underline{B}) = - \log(B(1) + B(2) * x^{B(3)}) - \log(y)$
(Farazdaghi and Harris)

Table 18-6-1 gives various results for these models when fitting the data in using different approaches and methods. The Holliday model appears to be quite difficult to estimate due to the very rapid change in the magnitude of the model function value with quite small changes in the parameters. Our results resemble closely those of Ratkowsky (1983, p. 59). Using the models directly, rather than in the logarithmic form, may make them extremely difficult to estimate if the precision for evaluation of the model is inadequate.

Table 18-6-1. Results for yield/density models using the Purnong Landing data of Table 1-3-1.

Model	Origin	B(1)	B(2)	B(3)	Sum of squares
Bleasdale	Ratkowsky	3.842E-3	1.302E-4	9.055E-1	0.283335
-Nelder	MRT	3.864E-3	1.306E-4	9.047E-1	0.2833202
	VM	3.219E-3	1.191E-4	9.296E-1	0.2835228
Holliday	Ratkowsky	2.054E-3	8.571E-5	3.808E-8	0.284193
	MRT**	1.883E-3	9.159E-5	0	0.2861092
	VM	0.9799	0.1118	-6.334E-4	1666.787<
Farazdaghi	Ratkowsky	2.294E-3	6.032E-5	1.080	0.282867
-Harris	MRT	2.294E-3	6.033E-5	1.080	0.282856
	VM	2.276E-3	6.151E-5	1.076	0.2828634

** initial parameters provided from asymptotic approximation (Ratkowsky, 1983, page 59)

< lower function value found by POSTGEN

- -

18-7. RS.FN - general inequality constraints

In general, we have not included methods for equality or inequality constraints other than simple masks or bounds. Problems involving such constraints are important, but in our experience do not constitute a major portion of the estimation tasks we have encountered. Furthermore, we believe the cost of both the generation and operational use of programs which incorporate such constraints to outweigh the benefits in the environments and applications where we envisage our work being put to work. However, it is worth illustrating how the tools provided in this book can be used to approach, albeit in a relatively inefficient manner, the solution of a problem involving inequality constraints. The function to be minimized is (Jacoby, 1972, page 25)

(18-7-1) $$f(\underline{B}) = B(1)^2 + B(2)^2 + 2B(3)^2 + B(4)^2 - 5B(1) - 5B(2) - 21B(3) + 7B(4)$$

subject to the constraints

(18-7-2) $$-B(1)^2 - B(2)^2 - B(3)^2 - B(4)^2 - B(1) + B(2) - B(3) + B(4) + 8 \geq 0$$

(18-7-3) $$-B(1)^2 - 2B(2)^2 - B(3)^2 - 2B(4)^2 + B(1) + B(4) + 1 \geq 0$$

(18-7-4) $$-2B(1)^2 - B(2)^2 - B(3)^2 - 2B(1)^2 + B(2) + B(4) + 5 \geq 0$$

Note that these constraints are nonlinear in the parameters. Among the classes of constrained minimization problems, linear inequality constraints are fairly straightforward, while nonlinear constraints are "considerably more complicated" (Gill, Murray and Wright, 1981, Chapter 5).

We use a simple penalty function approach to this constrained problem. For each of the constraints (18-7-2), (18-7-3) and (18-7-4) we compute a "violation", that is, the value of the left hand side of each constraint expression if this value is negative, and zero if it is non-negative. The sum of squares of the violations is then multiplied by a user-supplied constraint penalty factor and added to the objective function (18-7-1). The resulting function can then be minimized by our minimization methods. The code for this problem is given in Appendix A.

Table 18-7-1 presents the results of applying our methods to this problem. Because this is an artificial problem, it is possible to obtain apparent rapid convergence if a method such as HJ is started with initial parameter estimates which are are a whole number of stepsize units from the optimal values.

Notes to Table 18-7-1.

* HJ always uses an initial stepsize of 1.0 and a stepsize reduction factor of 0.1 for these runs

NM has the initial stepsize given with the algorithm

$$ The parameter values at the minimum were imput as initial values.

All runs made on 'Noname' IBM PC clone under MS-DOS 2.11 and GWBASIC with NEC V20 processor.

< implies that the axial search in POSTGEN discovered a lower function value

Table 18-7-1. Performance of minimization methods on the general constrained test problem RS.FN.

Method	Penalty	Function	value	Time	Grad	Funct	Unscaled	Constraint values			B(1)	B(2)	B(3)	B(4)
	scaling	Initial	Final	(s)	Evaluations		Function	1	2	3				
HJ *	10	0	-44.12002	108	0	441	-44.23619	5.4E-2	0E+0	9.3E-2	1.899992E-2	1.007799	2.016198	-0.9816996
	100	-43.07446	-44.01234	174	0	717	-44.02442	5.2E-3	0E+0	9.7E-3	4.999971E-3	1.000207	1.999754	-1.000405
	1000	-43.90361	-44.00117	41	0	163	-44.00247	5.0E-4	0E+0	1.0E-3	-9.100283E-4	1.004407	1.999754	-0.9994047
	10000	-43.98946	-44.00002	41	0	164	-44.00015	6.4E-5	0E+0	9.3E-5	-1.520029E-3	1.004847	1.999723	-0.9993648
	1000000	-43.98742	-43.99954	33	0	129	-43.99955	1.9E-6	0E+0	0E+0	-2.010028E-3	1.005367	1.999824	-0.9991647
	1E+7	-43.99951	-43.99986	37	0	144	-43.99986	0E+0	0E+0	0E+0	-1.840027E-3	1.005188	1.999824	-0.9991647
HJ $$	1E+7	-44	-44	15	0	57	-4E+1	0E+0	0E+0	0E+0	0	1	2	-1
NM 1	10	0	-44.12006<	185	0	251	...							
NM 0.001	10	-44.12006	-44.12006	23	0	280	-44.23564	5.4E-2	0E+0	9.3E-2	1.693887E-2	1.007544	2.017653	-0.9800386
NM 0.001	100	-43.07982	-44.01233	139	0	184	-44.02487	5.0E-3	0E+0	1.0E-2	-1.537669E-3	1.002994	2.00382	-0.9947969
CG	10	0	-44.12004	44	19	55	-44.23604	5.4E-2	0E+0	9.3E-2	1.775243E-2	1.007008	2.017134	-0.9809648
	100	-43.0761	-44.01226	34	13	41	-44.02465	4.9E-3	0E+0	1.0E-2	-2.374176E-3	1.001572	2.004842	-0.9937918
	1000	-43.90076	-44.00098	37	15	47	-44.00217	6.5E-4	0E+0	8.8E-4	-4.856950E-3	1.000909	2.003476	-0.9957204
	10000	-43.99022	-43.99987	32	13	39	-43.99996	6.3E-5	0E+0	6.7E-5	-5.023001E-3	1.000858	2.003344	-0.9958187
TN	10	0	-44.12006	69	50	26	-44.23581	5.4E-2	0E+0	9.3E-2	1.691801E-2	1.007767	2.01775	-0.9797119
	100	-43.0783	-44.01245	44	30	21	-44.02461	5.0E-3	0E+0	9.8E-3	1.801350E-3	1.000673	2.00203	-0.9974418
	1000	-43.90296	-44.00125	58	35	39	-44.00249	5.0E-4	0E+0	1.0E-3	9.343142E-4	1.000025	1.999682	-1.000376
	10000	-43.99006	-44.00008<	43	28	23	...							
		-44.0001	-44.00011	22	41	35	-44.00028	3.7E-5	0E+0	1.2E-4	7.740972E-4	1.000309	1.999406	-1.000641
VM	10	0	-44.12006	77	22	66	-44.23628	5.4E-2	0E+0	9.4E-2	1.698375E-2	1.007771	2.017802	-0.9796037
	100	-43.07409	-44.01245	50	14	46	-44.02482	5.0E-3	0E+0	9.9E-3	1.861390E-3	1.00085	2.001957	-0.9975076
	1000	-43.9011	-44.00125	54	14	55	-44.00248	5.1E-4	0E+0	9.8E-4	1.592855E-4	0.9999021	2.000264	-0.9997096
	10000	-43.9902	-44.00013	60	12	78	-44.00026	5.8E-5	0E+0	1.0E-4	6.893097E-5	1.000094	1.999959	-1.000034

18-8. The Moré, Garbow and Hillstrom test problems

Moré, Garbow and Hillstrom (1981) have described a number of widely used test problems, all of which can be cast in a nonlinear least squares form. Of this set, we have used a number of examples in this book. As we intend to prepare BASIC code for the entire set of functions for publication elsewhere, we shall only summarize here the problems we have used within the present work.

POWSING.RES

This problem is a generalization of a problem with a nearly singular Hessian due to Powell (1962).

BALF.RES

This is Brown's (1969) almost-linear function of variable order. The traditional starting point has B(i) = 0.5 for all i. The solution has a zero function value at

$$(18\text{-}8\text{-}1)\quad B(j) = q \quad \text{for } q = 1, 2, \ldots, n-1$$
$$B(n) = q^{(1-n)}$$

where q satisfies

$$(18\text{-}8\text{-}2)\quad n\, q^{n} - (n+1)\, q^{(n-1)} + 1 = 0$$

HS25.RES

This three-parameter bounds-constrained minimization is called the Gulf research problem by Moré et al. However, it is presented by Hock and Schittkowski (1981) as problem 25 of their collection. The minimum is at (50, 25, 1.5) and has

zero sum of squares. The Hock and Schittkowski suggested initial point is (100, 12.5, 3), which has such large residuals that our methods have difficulty making progress towards the minimum. Moré et al. suggest (5, 2.5, 0.15) as a starting point, which has been used in our examples.

Codes for these problems are included on the program disk.

18-9. BAGLIVO.RES - a exponential growth curve

This problem has already been discussed in Secion 11-6. In particular, the modelling function is

$$(11\text{-}6\text{-}1)\quad Z(\underline{B}) = B(1) * (1 - \exp(-B(2) * (Y(i,2) - B(3))))$$

where the variable in column 2 of matrix Y records the time at which the size or weight of a growing organism reaches a value recorded in column 1 of Y. The suggested minimization is a simple sum of squared deviations. Data for this problem is provided in the file BAGLIVO.INP, which contains a sub-sample of just 36 data points. Data is given in the very simple form

(sample element number),(size of clams in mm.),(age in months)

For this data set, the minimum sum of squares is 1979.447 and the parameter values (5.074584, 9.781973, -363.5333).

COVER SHEET

Appendices

Chapter title: Appendices, Bibliography, and Index

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

APPENDIX A

PROGRAM CODE FOR TEST AND EXAMPLE PROBLEMS

Program codes are listed in alphabetic order of problem file name. References to codes appearing elsewhere in the book are listed in Table 15-1-2.

ABS.FN

```
30 DIM B(25),X(25),O(25,3),Y(25,1)
2000 LET I3=0: REM ABS.RES
2004 LET F=0
2005 FOR L=1 TO N
2010 LET F=F+ABS(B(L))
2015 NEXT L
2020 RETURN
2500 FOR L=1 TO N
2540 LET G(L)=1
2550 IF B(L)<0 THEN LET G(L)=-1: REM FOR ABSOLUTE VALUE
2560 IF B(L)=0 THEN LET G(L)=0
2570 NEXT L
2580 RETURN
3000 PRINT "ABS.RES -- MINIMIZE SUM OF ABSOLUTE PARAMETERS - 860119"
3005 PRINT #3,"ABS.RES -- MINIMIZE SUM OF ABSOLUTE PARAMETERS - 860119"
3010 LET P$="ABS.RES - SUM OF ABSOLUTE PARAMETERS"
3020 INPUT "NUMBER OF PARAMETERS = ";N
3030 PRINT #3,"NUMBER OF PARAMETERS = ";N
3040 LET M=N: REM FOR MRT
3050 FOR J=1 TO N
3060 LET O(J,1)=-100
3070 LET O(J,2)=100
3080 LET O(J,3)=1: REM ALL FREE
3090 LET B(J)=3.333: REM STARTING VALUES
3100 NEXT J
3110 RETURN
```

BAGLIVO.RES

```
30 DIM Y(400,2),B(3),X(3),O(3,3)
3000 PRINT "J. BAGLIVO VERSION OF VON BERTALANFFY GROWTH CURVE - 860805"
3010 PRINT #3,"J. BAGLIVO VERSION OF VON BERTALANFFY GROWTH CURVE -
     860805"
3020 LET P$="BAGLIVO.RES GROWTH CURVE - 860805"
3030 LET M=0: REM initially 0 data points
3040 LET N=3: REM 3 parameters
3045 INPUT "file for data = ";F$
3050 OPEN F$ FOR INPUT AS 1
3055 IF EOF(1) THEN 3080
3060 LET M=M+1
3065 INPUT #1,Z3,Y(M,1),Y(M,2)
3070 PRINT Z3,Y(M,1),Y(M,2)
3072 PRINT #3, Z3,Y(M,1),Y(M,2)
3075 GOTO 3055
3080 PRINT M;" data points found"
3090 PRINT #3,M;" data points found"
3100 REM now set the bounds on the parameters
3110 LET O(1,1)=0: REM positive asymptote
3120 LET O(1,2)=100: REM loose upper bound
3130 LET O(1,3)=1: REM not masked
3140 LET O(2,1)=0: REM positive to avoid zero divide
3150 LET O(2,2)=100: REM loose bounds on second parameter
3160 LET O(2,3)=1: REM not masked
3170 LET O(3,1)=-2: REM allow slightly negative shift
3180 LET O(3,2)=30: REM upper bound -- note test on exponential argument
3190 LET O(3,3)=1: REM not masked
3200 PRINT " FUNCTION FORM = 10*B(1)*(1-EXP(-0.01*B(2)*(Y(I,2)-B(3)))"
3210 PRINT #3," FUNCTION FORM = 10*B(1)*(1-EXP(-0.01*B(2)*(Y(I,2)-B(3)))"
3220 PRINT
3230 PRINT #3,
3240 RETURN
3500 LET I3=0: REM start with computable function
3510 IF ABS(0.01*B(2)*(Y(I,2)-B(3)))>50 THEN 3550:
     REM not computable if exponent big
3520 LET R1=10*B(1)*(1-EXP(-0.01*B(2)*(Y(I,2)-B(3))))-Y(I,1)
3540 RETURN
3550 LET I3=1: REM cannot compute the residual
3560 RETURN
4000 LET Z9=EXP(-0.01*B(2)*(Y(I,2)-B(3))): REM Baglivo growth curve
4010 LET D(1)=10*(1-Z9)
4020 LET D(2)=0.1*B(1)*Z9*(Y(I,2)-B(3))
4030 LET D(3)=-0.1*B(1)*B(2)*Z9
4040 RETURN
```

BT.RES

```
30 DIM B(25),X(25),O(25,3),Y(25,1)
3000 PRINT "BT.RES -- TEST BOUNDS IN NLLS PROBLEMS -- 851117"
3005 PRINT #3,"BT.RES -- TEST BOUNDS IN NLLS PROBLEMS -- 851117"
3010 LET P$="BT.RES -- TEST BOUNDS IN NLLS PROBLEMS -- 851117"
3020 PRINT "  simple quadratic with bounds"
3025 PRINT #3,"  simple quadratic with bounds"
3030 REM dim b(n),x(n)
3040 INPUT " number of parameters (50 max) ";N
3045 PRINT #3," number of parameters (50 max) ";N
3050 LET M=N
3060 IF N>50 OR N<2 THEN 3040
3070 FOR J=1 TO N
3080 LET B(J)=-N+2.2*J: REM initial parameter values
3090 LET O(J,1)=(J-1)*(N-1)/N: REM use this for test
3100 LET O(J,2)=J*(N+1)/N
3110 LET O(J,3)=1
3120 NEXT J
3130 LET O(INT(N/2)+1,3)=0: REM one parameter masked
3140 RETURN
3500 LET R1=B(I): REM bt.res
3510 RETURN
4000 FOR L=1 TO N: REM bt.res
4020 LET D(L)=0
4030 NEXT L
4040 LET D(I)=1
4050 RETURN
```

CAL4.FN

```
30 DIM B(25),X(25),O(25,3)
2000 LET Z1=ABS(B(N)): REM CAL4.FN
2005 LET F=Z1*Z1*Z1/3+0.5*Z1*Z1+0.5*(B(1)-5*Z9)*(B(1)-5*Z9)
2006 REM Z9 = alpha in SGN notation
2010 FOR L1=1 TO N-1
2015 LET Z2=B(L1+1)-Z9*B(L1)
2020 LET F=F+0.5*(1+0.1*L1)*Z2*Z2
2060 NEXT L1
2070 RETURN
2500 IF N<3 THEN 2550: REM CAL4.FN
2505 FOR L1=2 TO N-1
2510 REM -- DELETED
2515 LET Z2=B(L1)-Z9*B(L1-1)
2517 LET Z3=Z9*B(L1)-B(L1+1)
2520 LET G(L1)=(1+.1*L1)*Z9*Z3+(1+0.1*(L1-1))*Z2
2540 NEXT L1
2550 LET G(1)=1.1*Z9*(Z9*B(1)-B(2))+(B(1)-5*Z9)
2560 LET G(N)=B(N)+(1+0.1*(N-1))*(B(N)-Z9*B(N-1))
```

```
2570 IF B(N)>=0 THEN LET G(N)=G(N)+B(N)*B(N)
2580 IF B(N)<0 THEN LET G(N)=G(N)-B(N)*B(N)
2590 RETURN
3000 PRINT "CAL4.FN SGN FUNCTION 860629"
3010 PRINT #3,"CAL4.FN SGN FUNCTION 860629"
3020 LET P$="CAL4.FN SGN FUNCTION 860629"
3030 INPUT "ORDER OF PROBLEM (2<=N<=25) ";N
3035 PRINT #3,"ORDER OF PROBLEM (2<=N<=25) ";N
3040 INPUT "PROBLEM PARAMETER Z9=alpha (0.9 suggested) ";Z9
3045 PRINT #3,"PROBLEM PARAMETER Z9=alpha (0.9 suggested) ";Z9
3050 REM now set the bounds on the parameters
3090 LET B(1)=5*Z9: REM suggested starting point
3100 FOR I=1 TO N
3105 LET O(I,1)=-100
3110 LET O(I,2)=100
3115 LET O(I,3)=1: REM free
3117 IF I>1 THEN LET B(I)=Z9*B(I-1): REM initial guess
3120 NEXT I
3240 RETURN
```

GENROSE.FN

```
30 DIM B(25),X(25),O(25,3)
2000 LET F=1: REM Generalized Rosenbrock function 8510311
2020 FOR L=2 TO N
2030 LET Z1=B(L)-B(L-1)*B(L-1)
2040 LET Z2=1-B(L)
2050 LET F=Q1*Z1*Z1+Z2*Z2+F
2060 NEXT L
2070 RETURN
2500 LET G(1)=0: REM Generalized Rosenbrock partial derivatives
2510 FOR L=2 TO N
2520 LET Z1=B(L)-B(L-1)*B(L-1)
2530 LET Z2=1-B(L)
2540 LET G(L)=2*(Q1*Z1-Z2)
2550 LET G(L-1)=G(L-1)-4*Q1*B(L-1)*Z1
2560 NEXT L
2570 RETURN
3000 PRINT "GENERALIZED ROSENBROCK FUNCTION SETUP -- 851031"
3005 PRINT #3,"GENERALIZED ROSENBROCK FUNCTION SETUP -- 851031"
3010 LET P$="GENROSE.FN -- SGN GENERALIZED ROSENBROCK Q1=10"
3020 INPUT "NO. OF PARAMETERS (2<=N<=25):";N
3025 PRINT #3,"NO. OF PARAMETERS (2<=N<=25):";N
3030 IF N>25 THEN 3020: REM dimensions set for 25
3040 IF N<2 THEN 3020: REM minimum of 2 parameters
3050 LET Q1=100: REM to allow changes in the function form
3060 PRINT "SCALING FACTOR IN GENROSE =";Q1
3070 PRINT #3,"SCALING FACTOR IN GENROSE =";Q1
3090 REM set the bounds on the parameters
```

```
3100 FOR I=1 TO N
3106 LET O(I,1)=-100: REM loose lower bound
3108 LET O(I,2)=100: REM loose upper bound
3110 LET O(I,3)=1: REM not masked
3112 NEXT I
3118 LET I5=0: REM no. of masked parameters
3120 PRINT " FUNCTION FORM = 1 + SUMMATION of Z(I) for I=2 TO N"
3125 PRINT "    WHERE Z(I) = 100*(B(I)-B(I-1)**2)**2 + (1-B(I))**2"
3130 PRINT
3135 PRINT #3," FUNCTION FORM = 1 + SUMMATION of Z(I) for I=2 TO N"
3140 PRINT #3,"    WHERE Z(I) = 100*(B(I)-B(I-1)**2)**2 + (1-B(I))**2"
3145 PRINT #3,
3150 GOSUB 3450: REM select initial point
3190 RETURN
3450 PRINT "CHOOSE INITIAL POINT"
3451 PRINT #3,"CHOOSE INITIAL POINT"
3452 PRINT "1. ALL ZEROS"
3453 PRINT #3,"1. ALL ZEROS"
3454 PRINT "2. B(I)=I/(N+1)"
3455 PRINT #3,"2. B(I)=I/(N+1)"
3456 PRINT "3. B(I)=(-1)^I"
3457 PRINT #3,"3. B(I)=(-1)^I"
3458 PRINT "4. B(I)=-1"
3459 PRINT #3,"4. B(I)=-1"
3460 INPUT "   CHOICE  (1,2,3,4) ";L1
3461 PRINT #3,"   CHOICE  (1,2,3,4) ";L1
3462 IF L1=0 THEN RETURN
3463 FOR I=1 TO N
3464 ON L1 GOTO 3465,3470,3475,3480
3465 LET B(I)=0
3466 GOTO 3485
3470 LET B(I)=I/(N+1)
3471 GOTO 3485
3475 LET B(I)=1
3476 IF 2*INT(I/2)=I THEN LET B(I)=-1
3477 GOTO 3485
3480 LET B(I)=-1
3485 NEXT I
3490 RETURN
```

GENROSEJ.RES

```
30 DIM B(25),X(25),O(25,3),Y(25,2)
3000 PRINT "GENROSEJ.RES 851031 SGN generalized Rosenbrock"
3005 PRINT #3,"GENROSEJ.RES 851031 SGN generalized Rosenbrock"
3010 LET P$="GENROSEJ.RES 851031 SGN generalized Rosenbrock"
3020 INPUT "NO. OF PARAMETERS (2<=N<=25):";N
3025 PRINT #3,"NO. OF PARAMETERS (2<=N<=25):";N
3030 IF N>25 THEN 3020: REM dimensions set for 25
3040 IF N<2 THEN 3020: REM minimum of 2 parameters
3045 LET M=2*N-1
3050 LET Q1=100: REM to allow changes in the function form
3052 PRINT "SCALING FACTOR IN GENROSE =";Q1
3054 PRINT #3,"SCALING FACTOR IN GENROSE =";Q1
3060 LET Q2=SQR(Q1)
3090 REM set the bounds on the parameters
3100 FOR I=1 TO N
3106 LET O(I,1)=-100: REM loose lower bound
3108 LET O(I,2)=100: REM loose upper bound
3110 LET O(I,3)=1: REM not masked
3112 NEXT I
3120 PRINT " FUNCTION FORM = 1 + SUMMATION of Z(I) for I=2 TO N"
3125 PRINT "    WHERE Z(I) = 10*(B(I)-B(I-1)**2)**2 + (1-B(I))**2"
3130 PRINT
3140 PRINT #3," FUNCTION FORM = 1 + SUMMATION of Z(I) for I=2 TO N"
3150 PRINT #3,"    WHERE Z(I) = 10*(B(I)-B(I-1)**2)**2 + (1-B(I))**2"
3160 PRINT #3,
3170 GOSUB 3450: REM initial point
3190 RETURN
3450 PRINT "CHOOSE INITIAL POINT"
3451 PRINT #3,"CHOOSE INITIAL POINT"
3452 PRINT "1. ALL ZEROS"
3453 PRINT #3,"1. ALL ZEROS"
3454 PRINT "2. B(I)=I/(N+1)"
3455 PRINT #3,"2. B(I)=I/(N+1)"
3456 PRINT "3. B(I)=(-1)^I"
3457 PRINT #3,"3. B(I)=(-1)^I"
3458 PRINT "4. B(I)=-1"
3459 PRINT #3,"4. B(I)=-1"
3460 INPUT "   CHOICE  (1,2,3,4) ";L1
3461 PRINT #3,"   CHOICE  (1,2,3,4) ";L1
3462 IF L1=0 THEN RETURN
3463 FOR I=1 TO N
3464 ON L1 GOTO 3465,3470,3475,3480
3465 LET B(I)=0
3466 GOTO 3485
3470 LET B(I)=I/(N+1)
3471 GOTO 3485
3475 LET B(I)=1
3476 IF 2*INT(I/2)=I THEN LET B(I)=-1
```

```
3477 GOTO 3485
3480 LET B(I)=-1
3485 NEXT I
3490 RETURN
3500 LET I3=0: REM genrosej -- rosenbrock least squares form
3530 IF I>1 THEN 3560
3540 LET R1=1
3550 RETURN
3560 LET L=INT(I/2)+1: REM index of summation
3570 IF 2*INT(I/2)=I THEN 3610
3580 REM odd i
3590 LET R1=1-B(L)
3600 RETURN
3610 LET R1=Q2*(B(L)-B(L-1)*B(L-1))
3620 RETURN
4000 FOR L=1 TO N: REM genrosej derivatives
4010 LET D(L)=0
4020 NEXT L
4030 IF I=1 THEN 4120
4040 LET L=INT(I/2)+1
4050 IF 2*INT(I/2)=I THEN 4100
4060 LET D(L)=-1
4070 RETURN
4100 LET D(L)=Q2
4110 LET D(L-1)=-2*Q2*B(L-1)
4120 RETURN
```

KMENTALP.RES

```
30 DIM Y(25,4),B(4),X(4),O(4,3)
3000 PRINT "KMENTA-TALPAZ CES PRODUCTION FUNCTION -- 860526"
3010 PRINT #3,"KMENTA-TALPAZ CES PRODUCTION FUNCTION -- 860526"
3020 LET P$="KMENTALP.RES Kmenta Talpaz CES production function"
3030 LET M=25: REM 12 data points
3040 LET N=4: REM 3 parameters
3050 REM note that we DIM Y(25,4) to allow residuals to be saved
3060 RESTORE: REM reset data pointer
3065 PRINT "output";TAB(15);"capital input";TAB(30);"labor input"
3066 PRINT #3,"output";TAB(15);"capital input";TAB(30);"labor input"
3070 FOR I=1 TO M
3080 READ Y(I,1), Y(I,2), Y(I,3): REM output, capital input, labor input
3082 PRINT Y(I,1);TAB(15);Y(I,2);TAB(30);Y(I,3)
3083 PRINT #3,Y(I,1);TAB(15);Y(I,2);TAB(30);Y(I,3)
3090 NEXT I
3092 PRINT
3093 PRINT #3,
3100 REM now set the bounds on the parameters
3110 LET O(1,1)=0: REM positive asymptote
3120 LET O(1,2)=100: REM loose upper bound
```

```
3130 LET O(1,3)=1: REM not masked
3140 LET O(2,1)=0: REM positive to avoid zero divide
3150 LET O(2,2)=100: REM loose bounds on second parameter
3160 LET O(2,3)=1: REM not masked
3170 LET O(3,1)=0: REM positive exponential parameter
3180 LET O(3,2)=30: REM upper bound -- note test on exponential argument
3190 LET O(3,3)=1: REM not masked
3192 LET O(4,1)=0: REM positive exponent
3194 LET O(4,2)=100: REM loose upper bound
3196 LET O(4,3)=1: REM not masked
3200 PRINT " Q = B(1)*((B(2)*K^(-B(4))+(1-B(2))*L^(-B(4)))^(-B(3)/B(4)))"
3210 PRINT #3," Q =
     B(1)*((B(2)*K^(-B(4))+(1-B(2))*L^(-B(4)))^(-B(3)/B(4)))"
3212 PRINT "  where Q = output, K=capital input, L=labor input"
3214 PRINT #3,"  where Q = output, K=capital input, L=labor input"
3220 PRINT
3230 PRINT #3,
3240 RETURN
3250 DATA 106, 8, 23
3252 DATA 81.08, 9, 14
3254 DATA 72.8, 4, 38
3256 DATA 57.34, 2, 97
3258 DATA 66.79, 6, 11
3260 DATA 98.23, 6, 43
3262 DATA 82.68, 3, 93
3264 DATA 99.77, 6, 49
3266 DATA 110, 8, 36
3268 DATA 118.93, 8, 43
3270 DATA 95.05, 4, 61
3272 DATA 112.83, 8, 31
3274 DATA 64.54, 3, 57
3276 DATA 137.22, 6, 97
3278 DATA 86.17, 4, 93
3280 DATA 56.25, 2, 72
3282 DATA 81.1, 3, 61
3284 DATA 65.23, 3, 97
3286 DATA 149.56, 9, 89
3288 DATA 65.43, 3, 25
3290 DATA 36.06, 1, 81
3292 DATA 56.92, 4, 11
3294 DATA 49.59, 2, 64
3296 DATA 43.21, 3, 10
3298 DATA 121.24, 6, 71
3500 LET I3=0: REM start with computable function
3510 REM KMENTALP.RES
3520 LET Z7=Y(I,2)^(-B(4))
3530 LET Z8=Y(I,3)^(-B(4))
3540 LET Z9=B(2)*Z7+(1-B(2))*Z8
3550 IF Z9<=0 THEN 3600: REM function not computable
3560 LET Z6=B(1)*(Z9^(-B(3)/B(4)))
3570 LET R1=Z6-Y(I,1)
```

```
3580 RETURN
3600 LET I3=1: REM failure
3610 RETURN
4000 PRINT "derivative expressions not included in problem file"
4010 PRINT "     --- use numerical derivatives"
4020 STOP
```

LINTEST.RES

```
30 DIM Y(19,4),B(4),X(4),O(4,3)
3000 PRINT "LINTEST 3-PARAMETER LINEAR REGRESSION - 860514"
3010 PRINT #3,"LINTEST 3-PARAMETER LINEAR REGRESSION - 860514"
3020 LET P$="LINTEST 3-PARAMETER LINEAR REGRESSION - 860514"
3030 LET M=19: REM 19 data points
3040 LET N=3: REM 3 parameters
3050 REM note that we DIM Y(19,4) to allow residuals to be saved
3060 RESTORE: REM reset data pointer
3065 FOR J=1 TO 3: REM LOOP OVER VARIABLES
3066 READ X$: REM NAME OF VARIABLE
3067 PRINT X$
3068 PRINT #3,X$
3070 FOR I=1 TO M
3080 READ Y(I,J): REM weed growth at time point I
3081 PRINT Y(I,J);
3082 PRINT #3,Y(I,J);
3083 IF 5*INT(I/5)<>I THEN 3090
3084 PRINT
3085 PRINT #3,
3090 NEXT I
3091 PRINT
3092 PRINT #3,
3095 NEXT J
3100 REM now set the bounds on the parameters
3110 FOR J=1 TO 3
3140 LET O(J,1)=-100
3150 LET O(J,2)=100: REM loose bounds
3160 LET O(J,3)=1: REM not masked
3190 NEXT J
3192 LET B(1)=.1: REM parameter initial values
3194 LET B(2)=10
3196 LET B(3)=10
3200 PRINT " FUNCTION FORM = 10*B(1)+0.1*B(2)*var(2)+0.1*B(3)*var(3)"
3210 PRINT #3," FUNCTION FORM = 10*B(1)+0.1*B(2)*var(2)+0.1*B(3)*var(3)"
3220 PRINT
3230 PRINT #3,
3240 RETURN
3250 DATA "QPORK",45.3, 49.1, 49, 44.3, 49.2, 56.5, 52.5, 50.3
3260 DATA 50.1, 50.7, 51.8, 47.9, 46.9, 53.8, 53.7, 52, 57.2, 66.2, 61
3270 DATA "PBEEF", 80.2, 81.6, 82.3, 85.7, 98.6, 105.1, 100.7, 100, 109.6
```

```
3280 DATA 106.5,103,107.3,118.1,124.2,126.3,136.7,140.7,144.3,157.6
3290 DATA "PPORK",98.7,86.9,87.5,103.4,100.1,92.8,91.8,100,103.2
3300 DATA 103.3,101,112.7,130.3,117.8,116.8,130.6,128.1,109.6,132.4
3500 LET I3=0: REM start with computable function
3520 LET R1=10*B(1)+.1*B(2)*Y(I,2)+.1*B(3)*Y(I,3)-Y(I,1)
3530 LET Y(I,4)=R1: REM save the residual value
3540 RETURN
4000 LET D(1)=10: REM LINTEST
4010 LET D(2)=.1*Y(I,2)
4020 LET D(3)=.1*Y(I,3)
4030 RETURN
```

LONION.RES

```
30 DIM B(3), X(3), O(3,3), Y(42,2)
3000 PRINT "LONION.RES -- Ratkowski (1983) yield/density models"
3002 PRINT "  for White Imperial Spanish Onions at Purnong Landing"
3004 PRINT "    851105"
3006 LET N=3
3008 PRINT "using log of deviations"
3010 PRINT "models: 1. y=(b(1)+b(2)*x)^(-1/b(3))"
3012 PRINT "           Bleasdale and Nelder"
3014 PRINT "        2. y=1/(b(1)+b(2)*x+b(3)*x*x)"
3016 PRINT "           Holliday"
3018 PRINT "    3. y=1/(b(1)+b(2)*x^b(3))"
3020 PRINT "           Farazdaghi and Harris"
3022 PRINT
3024 INPUT "which model is to be used ";L9
3030 PRINT #3, "LONION.RES -- Ratkowski (1983) yield/density models"
3032 PRINT #3, "  for White Imperial Spanish Onions at Purnong Landing"
3034 PRINT #3, "    851105"
3036 LET P$="LONION.RES Ratkowsky Yield/Density - 851105"
3038 PRINT #3, "using log of deviations"
3040 PRINT #3, "models: 1. y=(b(1)+b(2)*x)^(-1/b(3))"
3042 PRINT #3, "           Bleasdale and Nelder"
3044 PRINT #3, "        2. y=1/(b(1)+b(2)*x+b(3)*x*x)"
3046 PRINT #3, "           Holliday"
3048 PRINT #3, "    3. y=1/(b(1)+b(2)*x^b(3))"
3050 PRINT #3, "           Farazdaghi and Harris"
3052 PRINT #3,
3054 PRINT #3,"which model is to be used ";L9
3120 IF L9<1 OR L9>3 THEN 3008
3130 PRINT "model number ";L9;"  in use"
3140 PRINT #3,"model number ";L9;"  in use"
3150 LET M=42: REM number of data points
3155 RESTORE 3400
3160 FOR I=1 TO M
3180 READ Y(I,2),Y(I,1): REM note reverse order
3182 REM note that the data is not played back
```

```
3190 REM that is, read yield in y(.,2), density in y(.,1)
3192 NEXT I
3195 LET I5=0: REM no of masked parameters
3200 FOR J=1 TO 3: REM set bounds and masks
3210 LET O(J,1)=-100: REM loose lower bound
3220 LET O(J,2)=1000: REM loose upper bound
3230 PRINT "mask parameter ";J;" ";
3240 INPUT " ([cr] = no )";X$
3242 PRINT #3,"mask parameter ";J;" ";X$
3250 IF X$="Y" OR X$="y" THEN 3280
3260 LET O(J,3)=1
3270 GOTO 3290
3280 LET O(J,3)=0: REM masked
3285 LET I5=I5+1
3290 NEXT J
3300 RETURN
3400  DATA  23.48, 223.02
3401  DATA  26.22, 234.24
3402  DATA  27.79, 221.68
3403  DATA  32.88, 221.94
3404  DATA  33.27, 197.45
3405  DATA  36.79, 189.64
3406  DATA  37.58, 211.2
3407  DATA  37.58, 191.36
3408  DATA  41.49, 156.62
3409  DATA  42.66, 168.12
3410  DATA  44.23, 197.89
3411  DATA  44.23, 154.14
3412  DATA  51.67, 153.26
3413  DATA  55.58, 142.79
3414  DATA  55.58, 126.17
3415  DATA  57.93, 167.95
3416  DATA  58.71, 144.54
3417  DATA  59.5, 151.3
3418  DATA  60.67, 130.52
3419  DATA  62.63, 125.3
3420  DATA  67.71, 114.05
3421  DATA  70.06, 116.31
3422  DATA  70.45, 120.71
3423  DATA  73.98, 134.16
3424  DATA  73.98, 114.48
3425  DATA  78.67, 91.17
3426  DATA  95.9, 101.27
3427  DATA  96.68, 97.33
3428  DATA  96.68, 101.37
3429  DATA  101.38, 97.2
3430  DATA  103.72, 87.12
3431  DATA  104.51, 81.71
3432  DATA  105.68, 76.44
3433  DATA  108.03, 87.1
```

```
3434  DATA  117.82, 84.54
3435  DATA  127.21, 69.09
3436  DATA  134.26, 64.4
3437  DATA  137.39, 66.81
3438  DATA  151.87, 63.01
3439  DATA  163.61, 55.45
3440  DATA  166.35, 62.54
3441  DATA  184.75, 54.68
3500 LET I3=0: REM computability flag
3505 ON L9 GOTO 3510,3600,3700: REM Ratkowski yield/density models
3510 IF B(3)=0 THEN 3800: REM FAILURE
3511 IF B(1)+B(2)*Y(I,2)<E9 THEN 3800
3512 LET R1=(-1/B(3))*log(B(1)+B(2)*Y(I,2))-log(Y(I,1))
3513 REM Bleasdale Nelder
3520 RETURN
3600 LET Z1=Y(I,2): REM Holliday
3605 IF B(1)+B(2)*Z1+B(3)*Z1*Z1<E9 THEN 3800
3610 LET R1=-log(B(1)+B(2)*Z1+B(3)*Z1*Z1)-log(Y(I,1))
3620 RETURN
3700 IF B(1)+B(2)*(Y(I,2)^B(3))<E9 THEN 3800
3705 LET R1=-log(B(1)+B(2)*(Y(I,2)^B(3)))-log(Y(I,1)):
          REM Farazdaghi & Harris
3710 RETURN
3800 LET I3=1: REM function not computable
3810 RETURN
4000 ON L9 GOTO 4010,4100,4200: REM yield/density derivatives
4010 LET Z2=(B(1)+B(2)*Y(I,2)): REM Bleasdale Nelder
4020 LET Z3=-1/B(3)
4030 LET D(1)=Z3/Z2
4040 LET D(2)=Y(I,2)*D(1)
4050 LET D(3)=LOG(Z2)/(B(3)*B(3))
4060 RETURN
4100 LET Z1=Y(I,2): REM Holliday
4110 LET Z2=B(1)+(B(2)+B(3)*Z1)*Z1
4120 LET D(1)=-1/Z2
4130 LET D(2)=D(1)*Z1
4140 LET D(3)=D(2)*Z1
4150 RETURN
4200 LET Z1=Y(I,2): REM Farazdaghi and Harris
4210 LET Z2=B(1)+B(2)*Z1^B(3)
4220 LET D(1)=-1/Z2
4230 LET D(2)=D(1)*(Z1^B(3))
4240 LET D(3)=D(2)*B(2)*LOG(Z1)
4250 RETURN
```

NASHEASY.FN

```
30 DIM B(25),X(25),O(25,3)
2000 LET F=0: REM new Nash function
2010 LET Z5=1
2020 FOR L=1 TO N
2030 LET Z8=B(L)-L
2040 LET Z7=EXP(-Z8*Z8)-1
2050 LET Z5=Z5*Z7
2060 LET F=F+Z8*Z8
2070 NEXT L
2080 LET F=F+Z5*Z5
2090 RETURN
2500 LET Z5=1: REM new Nash function partial derivatives
2510 FOR L=1 TO N
2520 LET Z8=B(L)-L
2530 LET G(L)=2*Z8
2540 LET Z5=Z5*(EXP(-Z8*Z8)-1)
2550 NEXT L
2560 FOR L=1 TO N
2570 LET Z8=B(L)-L
2580 LET Z7=EXP(-Z8*Z8)
2590 IF Z7=1 THEN 2610
2600 LET G(l)=G(L)-4*Z5*Z5*Z8*ZY/(Z7-1)
2610 NEXT L
2620 RETURN
3000 PRINT "NASH FUNCTION NASHEASY.FN SETUP - 851119"
3010 PRINT #3,"NASH FUNCTION NASHEASY.FN SETUP - 851119"
3020 INPUT "NO. OF PARAMETERS (1<=N<=25):";N
3025 PRINT #3,"NO. OF PARAMETERS (1<=N<=25):";N
3030 IF N>25 THEN 3020: REM dimensions set for 25
3040 IF N<1 THEN 3020: REM minimum of 1 parameter
3090 REM set the bounds on the parameters
3100 FOR I=1 TO N
3106 LET O(I,1)=-100: REM loose lower bound
3108 LET O(I,2)=100: REM loose upper bound
3110 LET O(I,3)=1: REM not masked
3112 NEXT I
3118 LET I5=0: REM no. of masked parameters
3120 PRINT " FUNCTION FORM = SUMMATION of Z(I)**2 + PRODUCT of Z1(I)"
3122 PRINT "                                            for I=1 TO N"
3125 PRINT "    WHERE Z(I) = B(I) - I
3127 PRINT "    WHERE Z1(I)= ( EXP( -Z(I)*Z(I)) - 1 )**2"
3130 PRINT
3140 PRINT #3," FUNCTION FORM = SUMMATION of Z(I)**2 + PRODUCT of Z1(I)"
3145 PRINT #3,"                                            for I=1 TO N"
3150 PRINT #3,"    WHERE Z(I) = B(I) - I
3155 PRINT #3,"    WHERE Z1(I)= ( EXP( -Z(I)*Z(I)) - 1 )**2"
3160 PRINT #3,
3170 LET P$="NASHEASY.FN"
```

```
3190 RETURN
```

NASHHARD.FN

```
30 DIM B(25),X(25),O(25,3)
2000 LET F=0: REM Nash large function NASHHARD.FN
2010 REM f=sum (x(i)*x(i)*x(i+1)*x(1+1)) for i=1 to n
2020 REM take indices mod(n)
2030 FOR L=1 TO N
2040 LET L1=L+1
2050 IF L=N THEN LET L1=1
2060 LET Z8=B(L)-L+1
2070 LET Z9=B(L1)-L1+1
2080 LET F=F+Z8*Z8*Z9*Z9
2090 NEXT L
2100 RETURN
2500 FOR L=1 TO N: REM Nash large fn NASHHARD.FN partial derivatives
2510 LET L1=L+1
2520 IF L=N THEN LET L1=1
2530 LET L0=L-1
2540 IF L=1 THEN LET L0=N
2550 LET Z9=B(L)-L+1
2560 LET Z8=B(L1)-L1+1
2570 LET Z7=B(L0)-L0+1
2580 LET G(L)=2*Z9*(Z8*Z8+Z7*Z7)
2590 NEXT L
2600 RETURN
3000 PRINT "NASH LARGE FUNCTION NASHHARD.FN SETUP - 851117"
3005 PRINT #3,"NASH LARGE FUNCTION NASHHARD.FN SETUP - 851117"
3020 INPUT "NO. OF PARAMETERS (1<=N<=25):";N
3025 PRINT #3,"NO. OF PARAMETERS (1<=N<=25):";N
3030 IF N>25 THEN 3020: REM dimensions set for 25
3040 IF N<1 THEN 3020: REM minimum of 1 parameter
3090 REM set the bounds on the parameters
3100 FOR I=1 TO N
3106 LET O(I,1)=-100: REM loose lower bound
3108 LET O(I,2)=100: REM loose upper bound
3110 LET O(I,3)=1: REM not masked
3112 NEXT I
3118 LET I5=0: REM no. of masked parameters
3120 PRINT " FUNCTION FORM = SUMMATION of Z(I) for I=1 TO N"
3125 PRINT "    WHERE Z(I) = (B(I)-I+1)**2)*(B(I+1)-(I+1)+1)**2"
3130 PRINT
3140 LET P$="NASHHARD.FN"
3150 PRINT #3," FUNCTION FORM = SUMMATION of Z(I) for I=1 TO N"
3160 PRINT #3,"    WHERE Z(I) = (B(I)-I+1)**2)*(B(I+1)-(I+1)+1)**2"
3170 PRINT #3,
3190 RETURN
```

QUADSN.FN

```
30 DIM B(25),X(25),O(25,3)
2000 LET F=0: REM QUADSN.FN
2010 FOR L1=1 TO N
2020 LET F=F+0.5*B(L1)*B(L1)*L1
2030 LET Z1=L1
2040 IF L1>N/2 THEN LET Z1=-Z1
2050 LET F=F-B(L1)*Z1
2060 NEXT L1
2070 RETURN
2500 FOR L1=1 TO N
2510 LET Z1=L1
2520 IF L1>N/2 THEN LET Z1=-Z1
2530 LET G(L1)=L1*B(L1)-Z1
2540 NEXT L1
2550 RETURN
3000 PRINT "QUADSN.FN SGN QUADRATIC FUNCTION 860629"
3010 PRINT #3,"QUADSN.FN SGN QUADRATIC FUNCTION 860629"
3020 LET P$="QUADSN.FN SGN QUADRATIC FUNCTION 860629"
3030 INPUT "ORDER OF PROBLEM (N) ";N
3035 PRINT #3,"ORDER OF PROBLEM (N) ";N
3040 LET M=N
3050 REM now set the bounds on the parameters
3100 FOR I=1 TO N
3105 LET O(I,1)=-100
3110 LET O(I,2)=100
3115 LET O(I,3)=1: REM free
3117 LET B(I)=0: REM initial guess
3120 NEXT I
3200 PRINT " FUNCTION FORM = SUM (I=1 TO N) (0.5*I*B(I)^2-Q(I)*B(I)"
3205 PRINT "      WHERE Q(I)=I FOR I<=N/2, -I OTHERWISE"
3210 PRINT #3," FUNCTION FORM = SUM (I=1 TO N) (0.5*I*B(I)^2-Q(I)*B(I)"
3215 PRINT #3,"      WHERE Q(I)=I FOR I<=N/2, -I OTHERWISE"
3220 PRINT: REM optimal F=-N*(N+1)/4
3230 PRINT #3,
3240 RETURN
```

RS.FN

```
2000 LET Z1=-B(1)*B(1)-B(2)*B(2)-B(3)*B(3)-B(4)*B(4): REM constraint 1
2010 LET Z1=Z1-B(1)+B(2)-B(3)+B(4)+8
2020 IF Z1>0 THEN LET Z1=0
2030 LET Z2=-B(1)*B(1)-2*B(2)*B(2)-B(3)*B(3)-2*B(4)*B(4)
2040 LET Z2=Z2+B(1)+B(4)+10: REM constraint 2
2050 IF Z2>0 THEN LET Z2=0
2060 LET Z3=-2*B(1)*B(1)-B(2)*B(2)-B(3)*B(3)-2*B(1)+B(2)+B(4)+5
2070 IF Z3>0 THEN LET Z3=0: REM constraint 3
2080 LET F=B(1)*B(1)+B(2)*B(2)+2*B(3)*B(3)+B(4)*B(4)-5*(B(1)+B(2))
2090 LET F=F-21*B(3)+7*B(4): REM Rosen and Suzuki (1965) function
2095 LET F=F+Z9*(Z1*Z1+Z2*Z2+Z3*Z3): REM penalty for breaking constraint
2100 RETURN
2500 LET G(1)=2*B(1)-5: REM Rosen and Suzuki derivatives
2510 LET G(2)=2*B(2)-5
2520 LET G(3)=4*B(3)-21
2530 LET G(4)=2*B(4)+7
2540 LET Z1=-B(1)*B(1)-B(2)*B(2)-B(3)*B(3)-B(4)*B(4): REM constraint 1
2550 LET Z1=Z1-B(1)+B(2)-B(3)+B(4)+8
2560 IF Z1>0 THEN LET Z1=0
2570 LET Z2=-B(1)*B(1)-2*B(2)*B(2)-B(3)*B(3)-2*B(4)*B(4)
2580 LET Z2=Z2+B(1)+B(4)+10: REM constraint 2
2590 IF Z2>0 THEN LET Z2=0
2600 LET Z3=-2*B(1)*B(1)-B(2)*B(2)-B(3)*B(3)-2*B(1)+B(2)+B(4)+5
2610 IF Z3>0 THEN LET Z3=0: REM constraint 3
2620 LET G(1)=G(1)+2*Z9*(Z1*(-2*B(1)-1)+Z2*(-2*B(1)+1)+Z3*(-4*B(1)-2))
2630 LET G(2)=G(2)+2*Z9*(Z1*(-2*B(2)+1)+Z2*(-4*B(2))+Z3*(-2*B(2)+1))
2640 LET G(3)=G(3)+2*Z9*(Z1*(-2*B(3)-1)+Z2*(-2*B(3))+Z3*(-2*B(3)))
2650 LET G(4)=G(4)+2*Z9*(Z1*(-2*B(4)+1)+Z2*(-4*B(4)+1)+Z3)
2660 RETURN
3000 PRINT "RS.FN -- NONLINEAR INEQUALITY CONSTRAINTS -- 860120"
3010 PRINT #3,"RS.FN -- NONLINEAR INEQUALITY CONSTRAINTS -- 860120"
3020 LET P$="RS.FN -- 860120"
3030 LET N=4
3040 FOR I=1 TO N
3050 LET O(I,1)=-100: REM loose bounds
3060 LET O(I,2)=100
3070 LET O(I,3)=1: REM free
3080 NEXT I
3090 REM traditional start
3100 LET B(1)=0
3110 LET B(2)=0
3120 LET B(3)=0
3130 LET B(4)=0
3140 INPUT "penalty function scaling factor=";Z9
3150 PRINT #3,"penalty function scaling factor=";Z9
3200 RETURN
4500 PRINT "current penalty function scaling factor =";Z9
4510 PRINT #3,"current penalty function scaling factor =";Z9
```

```
4520 LET Z8=Z9: REM save value of penalty scaling
4530 LET Z9=0
4540 GOSUB 2000: REM get unpenalized function
4550 PRINT "unpenalized function = ";F
4560 PRINT "  constraint violations (in order)";Z1;Z2;Z3
4570 PRINT
4580 PRINT #3, "unpenalized function = ";F
4590 PRINT #3, "  constraint violations (in order)";Z1;Z2;Z3
4600 PRINT #3,
4610 INPUT "new penalty function scaling factor =";Z9
4620 PRINT #3,"new penalty function scaling factor =";Z9
4630 RETURN
```

WOOD.FN

```
2000 LET Z1=1-B(1): REM wood fn
2010 LET Z2=B(2)-1
2020 LET Z3=1-B(3)
2030 LET Z4=B(4)-1
2040 LET Z5=B(2)-B(1)*B(1)
2050 LET Z6=B(4)-B(3)*B(3)
2060 LET F=100*Z5*Z5+Z1*Z1+90*Z6*Z6
2070 LET F=F+Z3*Z3+10.1*(Z2*Z2+Z4*Z4)+19.8*Z2*Z4
2080 RETURN
2500 LET Z1=1-B(1): REM wood gradient
2510 LET Z2=B(2)-1
2520 LET Z3=1-B(3)
2530 LET Z4=B(4)-1
2540 LET Z5=B(2)-B(1)*B(1)
2550 LET Z6=B(4)-B(3)*B(3)
2560 LET G(1)=-400*Z5*B(1)-2*Z1
2570 LET G(2)=200*Z5+20.2*Z2+19.8*Z4
2580 LET G(3)=-360*Z6*B(3)-2*Z3
2590 LET G(4)=180*Z6+20.2*Z4+19.8*Z2
2600 RETURN
3000 PRINT "WOOD.FN - test problem - 860103"
3005 PRINT #3,"WOOD.FN - test problem - 860103"
3010 LET P$="WOOD.FN - test problem - 860103"
3020 LET N=4
3030 FOR L=1 TO N
3040 LET O(L,1)=-1000
3050 LET O(L,2)=1000
3060 LET O(L,3)=1: REM free
3070 NEXT L
3080 INPUT "impose bounds ? ([cr] = no) ";X$
3085 PRINT #3,"impose bounds ? ([cr] = no) ";X$
3090 IF X$="y" OR X$="Y" THEN 3110
3100 GOTO 3150
3110 FOR L=1 TO N
```

```
3120 LET O(L,1)=-10
3130 LET O(L,2)=10
3140 NEXT L
3150 LET B(1)=-3
3160 LET B(2)=-1
3170 LET B(3)=-3
3180 LET B(4)=-1
3190 PRINT "traditional start=";B(1);B(2);B(3);B(4)
3195 PRINT #3,"traditional start=";B(1);B(2);B(3);B(4)
3200 RETURN
```

WOODR.RES

```
3000 PRINT "WOODR.RES - test problem - 860103"
3005 PRINT #3,"WOODR.RES - test problem - 860103"
3010 LET P$="WOODR.RES - test problem - 860103"
3020 LET N=4
3025 LET M=6
3030 FOR L=1 TO N
3040 LET O(L,1)=-1000
3050 LET O(L,2)=1000
3060 LET O(L,3)=1: REM free
3070 NEXT L
3080 INPUT "impose bounds ? ([cr] = no) ";X$
3085 PRINT #3,"impose bounds ? ([cr] = no) ";X$
3090 IF X$="y" OR X$="Y" THEN 3110
3100 GOTO 3150
3110 FOR L=1 TO N
3120 LET O(L,1)=-10
3130 LET O(L,2)=10
3140 NEXT L
3150 LET B(1)=-3
3160 LET B(2)=-1
3170 LET B(3)=-3
3180 LET B(4)=-1
3190 PRINT "traditional start=";B(1);B(2);B(3);B(4)
3195 PRINT #3,"traditional start=";B(1);B(2);B(3);B(4)
3200 RETURN
3500 IF I>6 THEN 3900: REM WOODR.RES residuals
3520 ON I GOTO 3600,3610,3620,3630,3640,3650
3600 LET R1= 10 * (B(2) - B(1) * B(1))
3605 GOTO 3680
3610 LET R1=1 - B(1)
3615 GOTO 3680
3620 LET R1= (B(4) - B(3) * B(3)) * SQR (90)
3625 GOTO 3680
3630 LET R1= 1 - B(3)
3635 GOTO 3680
3640 LET R1= (B(2) + B(4) - 2) * SQR(10)
```

```
3645 GOTO 3680
3650 LET R1= (B(2) - B(4)) * SQR(.1)
3680 RETURN
3900 LET I3=1
3905 PRINT "error in index variable I"
3907 PRINT #3,"error in index variable I"
3910 STOP
4000 IF I>6 THEN 3900: REM WOODR.RES Jacobian
4010 ON I GOTO 4050,4100,4150,4200,4250,4300
4050 LET D(3)=0
4060 LET D(4)=0
4070 LET D(2)=10
4080 LET D(1)=-20*B(1)
4090 RETURN
4100 LET D(2)=0
4110 LET D(3)=0
4120 LET D(4)=0
4130 LET D(1)=-1
4140 RETURN
4150 LET D(1)=0
4160 LET D(2)=0
4170 LET D(4)=SQR(90)
4180 LET D(3)=-D(4)*B(3)*2
4190 RETURN
4200 LET D(1)=0
4210 LET D(2)=0
4220 LET D(4)=0
4230 LET D(3)=-1
4240 RETURN
4250 LET D(1)=0
4260 LET D(3)=0
4270 LET D(2)=SQR(10)
4280 LET D(4)=D(2)
4290 RETURN
4300 LET D(1)=0
4310 LET D(3)=0
4320 LET D(2)=SQR(.1)
4330 LET D(4)=-D(2)
4340 RETURN
```

APPENDIX B

DATA DICTIONARY

The following variables have been used in our parameter estimation program codes. The ancillary programs PLOT.BAS and DUBLTEST.BAS, and any problem files are excluded. Table 15-1-1 gives the locations of the codes.

A(	MRT,POSTMRT	the Jacobian inner product matrix stored as a vector
A(	POSTVM,VM	the Hessian inverse approximation (matrix)
A(	TN	product of Hessian approximation and search direction
A2	TN	residual update parameter in cg step
A7	MRT	lambda increase factor
A8	MRT	Nash phi factor
A9	MRT	lambda decrease factor (A9/A7)
B(	CG,DRIVER,FGTEST, HJ,MRT,NM,NUMGRAD, NUMJAC,POSTGEN, POSTMRT,POSTVM, RJTEST,TN,VM	the parameter vector

B2	TN	cg update parameter
B9	CG,ENVRON,HJ,NM, POSTGEN,POSTMRT, POSTVM	a "large" number
C(	MRT,POSTMRT	Jacobian inner product matrix stored as a vector
C(	VM	change in gradient
C0	POSTGEN,POSTMRT, POSTVM	denominator for curvature
C1	POSTGEN,POSTMRT, POSTVM	linear term
C2	POSTGEN,POSTMRT, POSTVM	quadratic term
C3	CG	stepsize reduction factor
C4	CG	stepsize increase factor
D(	MRT,RJTEST,SUMSQR	vector for I-th row of Jacobian
D(	NUMJAC	the numerical approximations to the I-th row of the Jacobian
D(	POSTMRT	standard error
D0	ENVRON,NUMGRAD, NUMJAC	a temporary floating-point scalar
D1	ENVRON	a temporary floating-point scalar
D1	VM	used in Hessian inverse update
D1	TN	used to update preconditioning
D2	VM	used in Hessian inverse update
D2	TN	used in preconditioning update
D8	TN	stepsize in Hessian approximation within cg step
E(	TN	preconditioning scaling factors
E3	NUMGRAD,NUMJAC	stepsize in derivative approximation
E3	TN	linear cg convergence tolerance
E4	NUMGRAD	common stepsize factor
E4	NUMJAC	step size for derivatives

E5	CG,ENVRON,FGTEST, HJ,MRT,NUMGRAD, NUMJAC,POSTGEN, POSTMRT,POSTVM, RJTEST,TN,VM	a value used for scaled comparisons (= 10)
E6	ENVRON	the radix of the floating-point arithmetic
E7	TN	epsilon for tests
E8	CG	tolerance scaling
E8	FGTEST,RJTEST, POSTGEN,POSTMRT, POSTVM	a stepsize
E8	TN	acceptable point test tolerance
E9	CG,ENVRON,FGTEST, HJ,MRT,NM,NUMGRAD, NUMJAC,POSTGEN, POSTMRT,POSTVM, RJTEST,TN,VM	the machine precision
F	CG,HJ,MRT,NM, NUMGRAD,POSTGEN, POSTMRT,POSTVM, SUMSQR,TN,VM	the loss function value or sum of squares
F	FGTEST	a temporary function value
F$	FGTEST,RJTEST	the file for console image
F$	RESSAVE	name of file
F0	CG,DRIVER,FGTEST, HJ,MRT,NM,POSTGEN, POSTMRT,POSTVM, TN,VM	value of the function at the supposed minimum
F1	CG,HJ	a temporary function value
F1	DRIVER,POSTGEN, POSTMRT,POSTVM	lower value of loss function, if any, found by post-solution analysis
F3	NM	highest function value in current polytope
F3	POSTGEN,POSTMRT, POSTVM	function value minus stepsize from supposed minimum
F4	NUMGRAD	to save function value on entry
F4	POSTGEN,POSTMRT, POSTVM	function value plus stepsize from supposed minimum

F9	NM	adjusted initial function value
F9	POSTGEN,POSTMRT, POSTVM	loss function per degree of freedom (sigma^2)
G$	DRIVER	console image file
G(	CG,FGTEST,MRT, NUMGRAD,POSTMRT, POSTVM,SUMSQR, TN,VM	the gradient vector
G1	CG	cg update parameter
G2	CG	"old" gradient norm
G8	MRT	projection of search direction into constraint space
G8	VM	gradient projection on search direction
G9	CG,TN	the gradient norm
H	FGTEST,RJTEST	the stepsize for the numerical approximation to the gradient or Jacobian
H(	TN	preconditioning vector
H(	VM	Hessian inverse matrix
I	CG,FGTEST,MRT, RESSAVE,RJTEST,VM	a loop control counter
I	SUMSQR	index for residuals
I1	CG	counter for steepest descent loop
I1	FGTEST	a loop control counter
I1	TN	main loop counter
I2	CG	number of cycles before restart
I3	CG,FGTEST,HJ,MRT, NM,POSTGEN,POSTMRT, POSTVM,RJTEST, SUMSQR,TN,VM	flag for function not computable (set to 1 on failure)
I4	TN	number of cg iterations in inner loop
I5	CG,DRIVER,HJ,MRT, NM,POSTGEN,POSTMRT, POSTVM,TN,VM	the number of masked (fixed) parameters

I6	POSTGEN,POSTMRT, POSTVM	flag for active bounds (-2=lower bound, -1=upper bound, 0=inactive)
I6	CG,HJ,MRT,TN,VM	a counter for the number of parameters which are unchanged in a step
I7	TN	conjugate gradients loop counter
I7	VM	gradient iteration count at last steepest descent direction
I8	CG,DRIVER,MRT,TN, VM	counter for gradient evaluations performed
I9	CG,DRIVER,HJ,MRT, NM.TN,VM	counter for function evaluations performed
J	CG,DRIVER,FGTEST, HJ,MRT,NM,POSTGEN, POSTMRT,POSTVM, RJTEST,SUMSQR,TN, VM	a loop control counter
J1	ENVRON	the number of radix digits in the mantissa of a floating point number
J1	NM,RJTEST	a loop control counter
J2	RJTEST	a loop control counter
J5	CG	marker to ensure a step is taken
J6	POSTGEN,POSTVM, SUMSQR	flag which is 1 if residuals can be computed
J7	CG,HJ,NM,MRT,TN,VM	parameter display control index
J8	CG,HJ,NM,MRT,TN,VM	parameters are displayed every J8 function evaluations (no display if J8=0)
J9	NUMGRAD,NUMJAC	a loop control counter
K	CG,MRT,NM,POSTGEN, POSTMRT,POSTVM	a loop control counter
K$	NM	indicator string for type of polytope operation

K1	MRT	triangular array element index for Choleski decomposition
K1	NM	an index variable (often refers to the lowest vertex in the polytope)
K2	MRT	triangular array element index
K2	TN	pointer to upper or lower bound
K3	NM	an index variable (often refers to the highest vertex in the polytope)
K9	CG	pointer to upper or lower bound
K9	POSTMRT	a loop control counter
M	MRT,POSTGEN,POSTMRT, POSTVM,RESSAVE, RJTEST,SUMSQR	the number of data points or residuals
M9	TN	iteration limit
N	CG,DRIVER,FGTEST,HJ, MRT,NM,NUMGRAD, NUMJAC,POSTGEN, POSTMRT,POSTVM, RJTEST,SUMSQR,TN,VM	the number of parameters in loss function
N2	MRT,POSTMRT	number of elements in upper triangular matrix holding Jacobian inner product or its Choleski decomposition
N2	NM	number of dimensions in polytope
O(	CG,DRIVER,FGTEST,HJ, MRT,NM,NUMGRAD, NUMJAC,POSTGEN, POSTMRT,POSTVM, RJTEST,TN,VM	bounds and masks information storage
P$	DRIVER,FGTEST	the name of the problem
Q$	CG,HJ,MRT,NM,TN,VM	used to hold display information regarding masks and active bounds
R(	TN	linear cg residuals
R0	NUMJAC,RJTEST	residual value at initial parameters

R1	MRT,RESSAVE,SUMSQR	the value of the residual
R1	NUMJAC	residual at (original point + delta)
R1	RJTEST	temporary storage of the residual
R1	TN	used in linear cg step
R2	TN	used in linear cg step
R9	TN	residual norm
S	MRT	temporary accumulator / factor in Choleski algorithm
S(	RJTEST	a flag that the relative deviation of the derivative is unacceptable
S1	CG,TN,VM	the stepsize
S1	DRIVER,MRT	initial step size
S1	HJ	the current stepsize for axial search
S1	NM	stepsize used to build the initial or restart polytope
S2	CG	initial step length
S2	DRIVER	additional initial stepsize information
S2	HJ	the stepsize reduction factor
S3	CG	best stepsize in line search
S3	HJ	a temporary variable
S4	CG	stepsize on entry to inverse quadratic interpolation
S5	NM	polytope reflection factor
S6	NM	polytope reduction and shrinkage factor
S7	MRT,TN,VM	distance to bound
S7	NM	polytope extension factor
S8	CG,MRT,TN,VM	maximum stepsize allowed by bounds
S8	NM	"size" of polytope before shrink operation
S9	MRT	lambda (damping factor)
S9	NM	"size" of polytope after shrink operation
S9	RJTEST	the sum of the squared residuals
S9	VM	accumulator for H * G row I

T$	DRIVER	used to store time-stamp
T$	RESSAVE	temporary string variable for file information
T(	CG,MRT,TN,VM	the search direction
T(	FGTEST,RJTEST	temporary storage for parameters
T(	HJ,NM	a temporary vector needed by POSTGEN
T(	POSTGEN,POSTMRT, POSTVM	tilt angles
T0	RJTEST	interpolated error of "best" fit
T1	DRIVER	used in computing elapsed time
T1	POSTMRT	denominator for parameter correlation estimate
T1	RESSAVE	temporary variable for file information
T1	RJTEST	stabilized denominator for derivative estimate comparisons
T2	CG,TN	projection of gradient on search direction
T2	DRIVER	used in computing elapsed time
T2	RESSAVE	temporary variable for file information
T3	DRIVER	used in computing elapsed time
T4	DRIVER	used in computing elapsed time
T5	DRIVER	elapsed seconds in minimization
T5	POSTGEN,POSTMRT, POSTVM	a "large" number assigned to the function value when constraints are violated
T7	RJTEST	user-specified tolerance for acceptable relative deviation
T8	TN	temporary storage for gradient
T9	CG,TN	convergence tolerance
U$	DRIVER	used to store time-stamp
U1	DRIVER	used in computing elapsed time
U2	DRIVER	used in computing elapsed time
U3	DRIVER	used in computing elapsed time

U4	DRIVER	used in computing elapsed time
W(	FGTEST,RJTEST	the numerical approximations to the gradient or Jacobian
W(	NM	an array to store the polytope and function values
W(	TN	step direction in linear cg iteration
X$	DRIVER,FGTEST POSTMRT,RJTEST	temporary user input
X(	CG,DRIVER,HJ,MRT, NM,POSTGEN,POSTMRT, POSTVM,TN,VM	the vector of final parameter estimates
Y(	RESSAVE	the data array for the problem

Users may wish to note that the following letters of the alphabet have NOT been used in programs.

L - could be reserved for integers within the user problem

P, Q, U, V, Z - can be used for REALs i.e. floating-point (U1, U2, U3, and U4 are used to compute the elapsed time in DRIVER after the parameters have been estimated.)

In Microsoft BASIC, all our programs can be operated in double precision by prefacing code with the statement

```
1 DEFDBL A-H,O-Z
```

This is placed at line 1, which will be the first line in the program.

APPENDIX C

LINE NUMBER CONVENTIONS

For greater detail, the reader is referred to Chapter 15 and the discussion and tables therein.

1- 999	Driver program
1000-1999	Function or sum of squares minimization routine
2000-2499	Function: F=f($\underline{B}$); increment I9
2500-2999	Gradient: $\underline{g}$($\underline{B}$); increment I8
3000-3499	Setup i.e. get data in Y(,), possibly from data files; set bounds and masks; initialize $\underline{B}$
3500-3999	Residual I of nonlinear least squares problem in R1
4000-4499	Partial derivatives of Ith residual (R1) with respect to $\underline{B}$ in $\underline{D}$ (Row I of Jacobian J)
4500-5999	Results-analysis (problem dependent)

6000-6899	Post-analysis (minimization algorithm dependent)
6900-6999	Save residuals
7000-7330	Environment for calculations (called at line 7120)

APPENDIX D

EXCEPTIONS FROM MINIMAL BASIC

In the program code presented in this book we have tried to keep the style close to the International Standard Minimal BASIC (ISO 6373/1984). However, Minimal BASIC does not lend itself to easily readable listings, nor does it allow for data files. Thus we have allowed ourselves the liberty of the following extensions to Minimal BASIC.

1. We allow the use of trailing REMark statements, but retain the principle of only one executable statement per line. Trailing REMarks may indicate machine specific constructs for timing or documentation as discussed below.
2. Lower-case characters may be found in REMark and PRINT statements
3. Input/Output of data to files follows the Microsoft conventions, that is,

```
     OPEN <filename> FOR <INPUT/OUTPUT> AS <file number>
     INPUT #<file number>,<variable list>
     PRINT #<file number>,<print list>
```

We do not use READ #<file number> or WRITE #<file number> constructs in our programs, since these use binary files which cannot be printed or edited directly. However, READ statements which acquire information from DATA statements are used in example problems. Apart from statements which begin "PRINT #3,..." to allow console output to be saved in a file, data file operations are confined to a few subroutines to limit the work required to make modifications.

4. We allow the use of

```
IF <logical condition> THEN <executable statement>
```

rather than the Minimal BASIC form

```
IF <logical condition> THEN <line number>
```

since this yields programs which are more obviously readable. Users who are restricted to Minimal BASIC can modify our programs trivially as in the following example.

```
100 IF K<N THEN LET X=2*LOG(B)
```

becomes

```
100 IF K>=N THEN 110
101 LET X=2*LOG(B)
110 ...
```

5. In programs which produce tabular output, and in particular the derivative testing programs FGTEST and RJTEST and the post-solution analysis routines POSTGEN, POSTMRT and POSTVM, we have used the PRINT USING facility of Microsoft BASIC rather than invoke rather complicated program code to provide scaled output.

6. Where necessary for the calculation processes, and again in the derivative testing programs FGTEST and RJTEST, we allow the use of double precision variables via the statement

```
DEFDBL A-H,O-Z
```

This is consistent with our variable naming conventions which have been presented in Appendix B. Users may find it useful to insert such a statement at the beginning of consolidated programs if the loss function used for nonlinear parameter estimation is believed to have a small variation (i.e. relative or percentage change) over the domain of the parameters of interest.

7. To indicate how programs are timed and how certain convenient run-time documentation may be incorporated into programs, in particular into the DRIVER program code, we use some programming language constructs which may be specific to MS-DOS versions of Microsoft BASIC, or at least to Microsoft BASIC character handling. Such statements are marked with the trailing REMark

```
REM !! <optional text>
```

Some of the Minimal BASIC conventions which ARE used are

- LET for assignment statements
- we avoid using character string operations apart for assignment of text to a character string variable
- scalar variables have names made up of a single letter followed by a single digit
- array variables have names which are a single letter.

Thus we are fairly confident that users will have no trouble using the code presented in almost any computer which runs a BASIC having floating-point arithmetic and at least the Minimal BASIC variable naming conventions.

Users may note that no "subroutine" starts with a REMark statement, since on occasion it may be desirable to take out such statements to shorten overall code length. Furthermore, we have noted that the Microsoft BASIC compilers may not generate correct object code if an attempt is made to branch to a REMark.

APPENDIX E

COMPUTING ENVIRONMENT

In several sections of this book we have indicated that the properties of the computing system used are important to the operation of the methods used. The main concerns (apart from programming language features and syntax dialects) are:

- the radix of floating-point arithmetic available
- the machine precision (i.e. number of radix digits) of this arithmetic
- the precision to which special functions are computed.

An especially ingenious and rigorous approach to these (and other) computing environment features has been devised by Professor W. Kahan of the University of California-Berkeley (Karpinski, 1985). This is the collection of computer programs known as PARANOIA which befit their name in the extent of the report on computational properties provided.

Simpler approaches are provided by other authors (Malcolm,1972; Gentleman and Marovich, 1974; Nash, 1981; Cody and Waite, 1980).

The machine precision E9 is the smallest positive number such that

$$1.0 + E9 > 1.0$$

If the arithmetic is to base E6, also called the radix. and J1 digits are available to store a floating-point number, this number will be <u>represented</u> as a polynomial in powers of E6 times some exponent and some sign. The exponent is simply a power of E6 which scales the number. Just as we can write 13.467 as 1.3467 * 10 * 10 = 1.3467E+1, we can put the <u>radix point</u> (which is the decimal point for E6=10) just after the first digit. Thus we write the general representation of a floating-point number as

$$\text{(AE-1)} \qquad \text{number} \underline{\sim} (\text{sign}) * E6^{X9} * \sum_{j=1}^{J1} C(j) * E6^{(1-j)}$$

where the C(j), j = 1,2,...J1 are the digits of the number.

Clearly, 1.0 has

```
C(1) = 1
C(j) = 0  if j>1
X9   = 0
```

The smallest positive number which, when added to 1.0, gives a number which is represented as a number greater than 1.0 is thus the number having (un-normalized) representation

```
C(j)  = 0  if j<J1
C(J1) = 1
X9    = 0
```

which has value $E9 = E6^{(1-J1)}$

This is the <u>machine precision</u>. The program ENVRON below

finds E6 and J1 for most computing environments.

Unfortunately, some environments may store numbers to a higher precision than that to which functions may be computed. Microsoft BASIC, for example, allows variables and arrays to be defined as double precision (Microsoft, 1983). However, many variants of this popular programming language processor will only compute special functions such as square root, sin, cos, tan, arctan, log or exp to single precision. Clearly this may compromise our work if high accuracy functions are needed, for example, in surveying or navigation applications.

The user can test functions to see if they are double precision using the short program DUBLTEST below. The program uses simple McLaurin series approximations computed in double precision near points where these approximations converge quickly. We use the following tests for x "small":

log(1-x), sin(x)/x, exp(x), arctan(x), sqrt(1+x*(2+x))

The absolute deviations and relative deviations between the function values returned by invoking the desired function in BASIC and the series-computed function value should be of the order of the double precision value for the machine precision. If larger deviations are observed, then it is probable that functions are intended to return results only to single precision. When the precision of such special functions is critical, the user will have to compute their values by other means, such as the series approach shown. However, we warn that this is NOT a good idea unless such series are known to converge rapidly for the arguments supplied.

```
          ENVRON.BAS         09-10-1985   21:32:24

7000 REM ENVRON -- SUBROUTINE TO DETERMINE MACHINE PRECISION
7010 REM                AND SET SOME STANDARD VARIABLES
7020 REM INPUTS:  NONE
7030 REM
7040 REM OUTPUTS:
7050 REM    B9 -- A LARGE NUMBER (CURRENTLY 1E+35)
7060 REM    E5 -- A NUMBER FOR RELATIVE EQUALITY TESTS (CURRENTLY 10)
7070 REM    E9 -- THE MACHINE PRECISION,  E9=MIN (X : 1+X>1)
7080 REM    E6 -- THE RADIX OF ARITHMETIC
7090 REM    J1 -- THE NUMBER OF RADIX DIGITS IN MANTISSA OF
7100 REM          FLOATING-POINT NUMBERS
7110 REM
7120 LET D1=1: REM USE A VARIABLE TO AVOID CONSTANTS WHEN DOUBLE
7130 REM       PRECISION IS INVOKED
7140 LET E5=10: REM ARBITRARY SCALING FOR ADDITIVE EQUALITY TESTS
7150 LET B9=1E+35: REM BIG NUMBER, NOT NECESSARILY BIGGEST POSSIBLE
7160 LET E6=1: REM INITIAL VALUE FOR RADIX
7170 LET E9=1: REM INITIAL VALUE FOR MACHINE PRECISION
7180 LET E9=E9/2: REM START OF LOOP TO DECREASE EST. MACHINE PRECISION
7190 LET D0=E6+E9: REM FORCE STORAGE OF SUM INTO A FLOATING-POINT SCALAR
7200 IF D0>E6 THEN 7180: REM REPEAT REDUCTION WHILE (1+E9) > 1
7210 LET E9=E9*2: REM RESTORE SMALLEST E9 WHICH GIVES (1+E9) > 1
7220 LET E6=E6+1: REM TRY DIFFERENT RADIX VALUES
7230 LET D0=E6+E9
7240 IF D0>E6 THEN 7220: REM UNTIL A SHIFT IS OBSERVED
7250 LET J1=1: REM INITIAL COUNT OF RADIX DIGITS IN MANTISSA
7260 LET E9=1: REM USE RADIX FOR EXACT MACHINE PRECISION
7270 LET E9=E9/E6: REM LOOP WHILE DIVIDING BY RADIX
7280 LET J1=J1+1: REM INCREMENT COUNTER
7290 LET D0=D1+E9: REM ADD TO 1
7300 IF D0>D1 THEN 7270: REM TEST AND REPEAT UNTIL EQUALITY
7310 LET E9=E9*E6: REM RECOVER LAST VALUE OF MACHINE PRECISION
7320 LET J1=J1-1: REM AND ADJUST THE NUMBER OF DIGITS
7330 RETURN

    Line no.    Referenced in line(s)
     7180      7200
     7220      7240
     7270      7300

     Symbol     Referenced in line(s)
    B9         7150 -- a big number
    D0         7190  7200  7230  7240  7290  7300 -- a temporary
               floating-point scalar
    D1         7120  7290  7300 -- a temporary floating-point scalar
    E5         7140 -- a scaling factor for additive equality tests
    E6         7160  7190  7200  7220  7230  7240  7270  7310
               -- the radix of the floating-point arithmetic
```

```
    E9         7170  7180  7190  7210  7230  7260  7270  7290  7310
               -- the machine precision (smallest positive number
               which gives (1+E9) > 1 in the available arithmetic
    J1         7250  7280  7320 -- the number of radix digits in
               the mantissa of a floating point number
    ==============================================================
    LINES: 34      BYTES: 1632     SYMBOLS: 11     REFERENCES: 35


             DUBLTEST.BAS      11-24-1985   11:20:57

10 PRINT "DUBLTEST - 851116 - to test precision of functions"
20 PRINT
30 PRINT "Computes functional approximations using double"
40 PRINT "precision using McLaurin series and compares results"
50 PRINT "with BASIC internal function called with double"
60 PRINT "precision argument.  Absolute and relative deviations"
70 PRINT "are reported which SHOULD be of the order of the"
80 PRINT "machine precision for double precision arguments if the"
90 PRINT "internal functions are computed to double precision."
100 PRINT
110 DEFSNG A-C
120 DEFDBL D-H,O-Z
130 LET A=1!
140 LET D=1#
150 LET B=.1
160 LET E=.1#
170 PRINT "functions available"
180 PRINT "    1)   log(1-x)"
190 PRINT "    2)   sin(x)/x"
200 PRINT "3)   exp(x)"
210 PRINT "    4)   arctan(x)"
220 PRINT "    5)   sqrt(1+2*x+x*x) - 1"
230 PRINT
240 INPUT "function to be used (1-5) ?";K8
250 PRINT
260 ON K8 GOTO 270,290,310,330,350
270 PRINT "log(1-x)"
280 GOTO 360
290 PRINT "sin(x)/x"
300 GOTO 360
310 PRINT "exp(x)"
320 GOTO 360
330 PRINT "arctan(x)"
340 GOTO 360
350 PRINT "sqrt(1+x*(2+x)) - 1"
360 PRINT
370 PRINT "   x";TAB(24);"   function (series)";
380 PRINT TAB(48);" abs. & rel. deviations"
390 LET C=A
```

```
400 LET F=D
410 FOR K=1 TO 16
420 LET C=C*B
430 LET F=F*E
440 ON K8 GOTO 450,470,490,510,530
450 GOSUB 640: REM log
460 GOTO 510
470 GOSUB 760: REM sin(x)/x
480 GOTO 540
490 GOSUB 870: REM exp(x)
500 GOTO 540
510 GOSUB 980: REM arctan(x)
520 GOTO 540
530 GOSUB 1090: REM sqrt(  )
540 PRINT  X;TAB(24);Z;TAB(48);
550 PRINT USING "##.###^^^^";ABS(Y-Z);
560 PRINT "   ";
570 IF Z=0# THEN 590
580 PRINT USING "##.###^^^^";(Y-Z)/Z;
590 PRINT
600 NEXT K
610 PRINT
620 PRINT
630 STOP
640 REM subroutine to evaluate fns in dbl
650 REM log(1-x)
660 LET X=F
670 LET Z=-X
680 LET T=X
690 LET J=1
700 LET J=J+1
710 LET T=T*X*(J-1)/J
720 LET Z=Z-T
730 IF Z<>Z-T THEN 700
740 LET Y=LOG(1#-X)
750 RETURN
760 REM sin(x)/x
770 LET X=F
780 LET Z=1#
790 LET T=Z
800 LET J=1
810 LET J=J+2
820 LET T=-T*X*X/(J*(J-1))
830 LET Z=Z+T
840 IF Z<>Z-T THEN 810
850 LET Y=SIN(X)/X
860 RETURN
870 REM exp(x)
880 LET X=F
890 LET Z=1#
```

```
900 LET J=0
910 LET T=1#
920 LET J=J+1
930 LET T=T*X/J
940 LET Z=Z+T
950 IF Z<>Z-T THEN 920
960 LET Y=EXP(X)
970 RETURN
980 REM arctan
990 LET X=F
1000 LET Z=X
1010 LET T=X
1020 LET J=1
1030 LET J=J+2
1040 LET T=-T*X*X*(J-2)/J
1050 LET Z=Z+T
1060 IF Z<>Z-T THEN 1030
1070 LET Y=ATN(X)
1080 RETURN
1090 REM sqrt(1+x*(2+x))-1
1100 LET X=F
1110 LET Z=2#+X
1120 LET Z=1#+X*Z
1130 LET Y=SQR(Z)-1#
1140 LET Z=X
1150 RETURN
```

COVER SHEET

Bibliography

Chapter title: Bibliography

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc.

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

BIBLIOGRAPHY

The following abbreviations are used in the bibliographic citations.

ACM Trans Math Softw	= ACM Transactions on Mathematical Software
Amer J Phys	= American Journal of Physics
Amer Stat	= American Statistician
Ann Math Stat	= Annals of Mathematical Statistics
Ann NY Acad Sc	= Annals of the New York Academy of Science
Ann Stat	= Annals of Statistics
Appl Stat	= Applied Statistics
Austral Comput J	= Australian Computer Journal
Biomc	= Biometrics
Biomk	= Biometrika
BIT	= Bit Nordisk Tidskrift for Informationsbehandling
Brit J Appl Phys	= British Journal of Applied Physics
Bull IIS	= Bulletin de L'Institut International de Statistique

Can Farm Econ	= Canadian Farm Economics
Can J Ag Econ	= Canadian Journal of Agricultural Economics
Can J Chem Eng	= Canadian Journal of Chemical Engineering
Can J Fish	= Canadian Journal of Fisheries and Aquatic Sciences
Chem Eng Prog	= Chemical Engineering Progress
Clin Pharmacol Ther	= Clinical Pharmacology and Therapeutics
Comm ACM	= Communications of the ACM (Association for Computing Machinery)
Comm Stat - Theory and Methods	= Communications in Statistics -- Theory and Methods
Comp Chem	= Computers and Chemistry
Comput Biom	= Computers and Biomedical Research
Comput J	= Computer Journal (British Computer Society)
Comput Phys Commun	= Computer Physics Communications
Comput Prog Biom	= Computer Programs in Biomedicine
CRC Crit Rev Bioeng	= Chemical Rubber Company Critical Reviews in Bioengineering
Duke Math J	= Duke Mathematical Journal
IEEE Power	= IEEE Transactions on Power Apparatus and Systems
Ind Eng Chem	= Industrial and Engineering Chemistry
Isr J Math	= Israel Journal of Mathematics
J ACM	= Journal of the Association for Computing Machinery

JASA	= Journal of the American Statistical Association
J Austral Math Soc	= Journal of the Australian Mathematics Society
J Biol Chem	= Journal of Biological Chemistry
J Econ	= Journal of Econometrics
J Forecasting	= Journal of Forecasting
J Franklin Inst	= Journal of the Franklin Institute
J Inst Math Applic	= Journal of the Institute of Mathematics and its Applications
J Oper Res Soc	= Journal of the Operational Research Society
J Optim Theory Applic	= Journal of Optimization Theory and Applications
J Mol Spectr	= Journal of Molecular Spectroscopy
J Phar Biop	= Journal of Pharmacokinetics and Biopharmaceutics
J Pharm Sci	= Journal of Pharmaceutical Sciences
J Res Nat Bur Stand	= Journal of Research (US) National Bureau of Standards
J Roy Stat Soc B	= Journal of the Royal Statistical Society, Series B
J SIAM	= Journal of the Society of Industrial and Applied Mathematics
J Stat Comput Simul	= Journal of Statistical Computing and Simulation
Math Biosci	= Mathematical Biosciences
Math Comput	= Mathematics of Computation
Math Prog	= Mathematical Programming
Mgmt Info	= Management Informatics

Mgmt Sci	= Management Science
Numer Math	= Numerische Mathematik
Q Appl Math	= Quarterly of Applied Mathematics
Rev Econ Stat	= Review of Economics and Statistics
R.I.R.O.	= Revue Française d'Informatique et Recherches Opérationelles
SIAM J Appl Math	= SIAM Journal of Applied Mathematics (Society for Industrial and Applied Mathematics)
SIAM J Numer Anal	= SIAM Journal on Numerical Analysis
SIAM J Sci Stat Comp	= SIAM Journal on Scientific and Statistical Computing
SIAM Rev	= SIAM Review
Texas J Sci	= Texas Journal of Science

- -

Allen, D.M. (?). Fitting Pharmacokinetic Models to Data. (Unpublished).

Anderson, D.H. & Osborne, M.R. (1977). Discrete, nonlinear approximation problems in polyhedral norms. Numer Math 28: 143-156.

Bard, Y. (1967). Nonlinear parameter estimation and programming. IBM New York Scientific Center: New York.

Bard, Y. (1970). Comparison of gradient methods for the solution of nonlinear parameter estimation problems. SIAM J Numer Anal 7: 157-186.

Bard, Y. (1974). Nonlinear Parameter Estimation. Academic Press: New York/London.

Barham, R.H. & Drane, W. (1972). An algorithm for least squares estimation of nonlinear parameters when some of the parameters are linear. Technometrics 14: 757-766.

Bartels, R.H. & Conn, A.R. (1981). An Approach to Nonlinear L-1 Data Fitting, C8-81-17, Computer Science Dept., Univ. of Waterloo: Waterloo, Ontario.

Bates, D.M. & Watts, D.G. (1980). Relative curvature measures of nonlinearity. J Roy Stat Soc B 42: 1-25.

Bates, D.M. & Watts, D.G. (1981a). A relative offset orthogonality convergence criterion for nonlinear least squares. Technometrics 23: 179-183.

Bates, D.M. & Watts, D.G. (1981b). Parameter transformations for improved approximate confidence regions in nonlinear least squares. Ann Stat 9: 1152-1167.

Bates, D.M. & Watts, D.G. (1985). Multiresponse estimation with special application to linear systems of differential equations. Technometrics 27: 329-339. Discussion 340-360.

Bates, D.M., & Watts, D.G. (1986+). Nonlinear Least Squares. To be published by John Wiley & Sons: New York.

Bates, D.M., Wolf, D.A. & Watts, D.G. (1986). Nonlinear Least Squares and First-Order Kinetics. Computer Science and Statistics: 17th Symposium on the Interface, D.M.Allen (ed.), Springer: New York, pp. 71-81.

Beale, E.M.L. (1960). Confidence regions in non-linear estimation. J Roy Stat Soc B 22: 41-76.

Beale, E.M.L. (1972). A derivation of conjugate gradients. In Lootsma (1972), pp. 39-43.

Beckman, R.J. & Cook, R.D. (1983). Outlier.....s. Technometrics 25: 119-149.

Beer, F.P. & Johnston, E.R. Jr. (1972). Vector Mechanics for Engineers: Statics and Dynamics, Second Edition. McGraw-Hill: New York.

Belsley, D.A. (1984a). Demeaning conditioning through centering. Amer Stat 38: 73-93.

Belsley, D.A. (1984b). Eigenvector weaknesses and other topics for assessing conditioning diagnostics. Technometrics 26: 297-299.

Belsley, D.A., Kuh, E. & Welsch, R.E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons: New York/Toronto.

Ben-Israel, A. (1967). On iterative methods for solving nonlinear least squares problems over convex sets. Isr J Math 5: 211-224.

Bera, A.K. (?). The Use of Linear Approximation to Nonlinear Regression Analysis. (Unpublished).

Bera, A.K. (?). Linearised Estimation of Nonlinear Simultaneous Equations System. (Unpublished).

Berkson, J. (1957). Tables for the maximum likelihood estimate of the logistic function. Biomc 13: 28-34.

Bernengo, J.C., Ronziere, M.C., Bezot, P., Bezot, C., Herbage, D. & Veis, A. (1983). A hydrodynamic study of collagen fibrillogenesis and quasielestic light scattering. J Biol Chem 258: 1001-1006.

Biggs, M.C. (1975). Some Recent Matrix Updating Methods for Minimising Sums of Squared Terms, Tech. Report No. 67. The Hatfield Polytechnic: Hatfield, UK.

Bleasdale, J.K.A. & Nelder, J.A. (1960). Plant population and crop yield. Nature 188: 342.

Bliss, C.I. & James, A.T. (1966). Fitting the rectangular hyperbola. Biomc 22: 573-602.

Bloomfield, P. & Steiger, W. (1980). Least absolute deviations curve-fitting. SIAM J Sci Stat Comp 1: 290-301.

BMDP Statistical Software, 1981 Edition (1981). Dept. of Biomathematics, Univ. of California, Univ. of California Press: Los Angeles.

Bodkin, R.G. & Klein, C.R. (1967). Nonlinear estimation of aggregated production functions. Rev Econ Stat 49: 28-44.

Boggs, P.T., Byrd, R.H. & Schnabel, R.B.(eds.) (1985). Numerical Optimization 1984. SIAM: Philadelphia.

Box, G.E.P. (1957). Evolutionary operation: a method for increasing industrial productivity. Appl Stat 6: 81-101.

Box, G.E.P. (1958). Use of statistical methods in the elucidation of basic mechanisms. Bull IIS (Stockholm) 36: 215-225.

Box, G.E.P. (1960). Fitting empirical data. Ann NY Acad Sc 86 (Art 3): 792-816.

Box, M.J. (1965). A new method of constrained optimization and a comparison with other methods. Comput J 8: 42-52.

Box, M.J. (1966). A comparison of several current optimization methods, and the use of transformers in constrained problems. Comput J 9: 67-77.

Box, M.J. (1971). Bias in nonlinear estimation. J Roy Stat Soc B 33: 171-201.

Box, M.J., Davies, D. & Swann, W.H. (1971). Techniques d'optimisation non linéaire, Monographie No. 5. Entreprise Moderne D'Edition: Paris. (Original English edition by Oliver & Boyd: London.)

Bradbury, W.W. & Fletcher, R. (1966). New iterative methods for solution of the eigenproblem. Numer Math 9: 259-267.

Breiman, L.& Friedman, J.H. (1985). Estimating optimal transformations for multiple regression and correlation. JASA 80: 580-619.

Bremermann, H. (1970). A method of unconstrained global optimization. Math Biosci 9:1-15.

Brent, R.P. (1973). Algorithms for Minimization without Derivatives. Prentice-Hall: Englewood Cliffs, NJ.

Broekhoven, L.H. & Watts, D.G. (1972). Improved Convergence Using Partitioned Least Squares (PLEAS), Mathematical Preprint No. 1972-26. Queen's Univ.: Kingston, Ontario.

Brown, K.M. (1969). A quadratically convergent Newton-like method based on Gaussian elimination. SIAM J Numer Anal 6: 560-569.

Brown, K.M. & Dennis, J.E. Jr. (1971). A new algorithm for nonlinear least-squares curve fitting. In Mathematical Software, John Rice (ed.). Academic Press: New York/ London, pp. 391-396.

Brown, K.M. & Dennis, J.E. Jr. (1972). Derivative free analogues of the Levenberg-Marquardt and Gauss algorithms for nonlinear least squares approximation. Numer Math 18: 289-297.

Brown, R.D. & Manno, J.E. (1978). ESTRIP, a BASIC computer program for obtaining initial polyexponential parameter estimates. J Pharm Sci 67: 1687-1691.

Byrd, R.H. (1981). Algorithms for robust regression. In Powell (1981) pp. 79-84.

Byron, R.P. & Bera, A.K. (?). Linearised Estimation of Nonlinear Single Equation Functions. (Unpublished).

Caceci, M.S. & Cacheris, W.P. (1984). Fitting curves to data (the Simplex algorithm is the answer). Byte 9 (5): 340-362 (May).

Cantor, D.G. & Evans, J.W. (1970). On approximation by positive sums of powers. SIAM J Numer Anal 18: 380-388.

Carr, N.L. (1960). Kinetics of catalytic isomerization of n-Pentane. Ind Eng Chem 52: 391-396.

Celmins, A. (1981). Least squares model fitting with transformations of variables. J Stat Comput Simul 14: 17-39.

Cauchy, A. (1848). Méthode générale pour la résolution des systèmes d'équations simultanées, Comptes Rendus de l'Académie des Sciences de Paris 27: 536-538.

Chambers, J.M. (1969). A computer system for fitting models to data. Appl Stat 18: 249-263.

Chambers, J.M. (1973). Fitting nonlinear models: numerical techniques. Biomk 60: 1-13.

Cody, W.J. & Waite, W. (1980). Software manual for the elementary functions. Prentice-Hall: Englewood Cliffs, NJ.

Coleman, T.F. & Conn, A.R. (1982). On the Local Convergence of a Quasi-Newton Method for the Nonlinear Programming Problem, TR 82-509. Dept. of Computer Science, Cornell Univ.: Ithaca, NY.

Coleman, T.F. & Moré, J.J. (1982). Software for Estimating Sparse Jacobian Matrices, TR 82-502. Dept. of Computer Science, Cornell Univ.: Ithaca, NY.

Conn, A.R. (1985). Nonlinear programming, exact penalty functions and projection techniques for non-smooth functions. In Boggs, Byrd & Schnabel (1985) pp. 3-25.

Cook, R.D. & Goldberg, M.L. (1986). Curvature for parameter subsets in nonlinear regression. Computer Science and Statistics: 17th Symposium on the Interface, D.M.Allen (ed.), Springer: New York, pp. 95-111.

Cook, R.D. & Witmer, J.A. (1985). A note on parameter-effects curvature. JASA 80: 872-878.

Craig, R.J. & Evans, J.W. (?). A Comparison of Nelder-Mead Type Simplex Search Procedures, TR No. 146. Dept. of Statistics, Univ. of Kentucky: Lexington, KY.

Craig, R.J., Evans, J.W. & Allen, D.M. (1980). The Simplex-Search in Non-linear Estimation, TR No. 155. Dept. of Statistics, Univ. of Kentucky: Lexington, KY.

Curry, H.B. (1944). The method of steepest descent for non-linear minimization problems. Q Appl Math 2: 258-261.

Dahlquist, G. & Bjőrck, A. (1974). Numerical Methods, (translated by N. Anderson). Prentice-Hall: Englewood Cliffs, NJ.

Dantzig, G.B. (1979). Comments on Khachian's Algorithm for Linear Programming, TR SOL 79-22. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Davidon, W.C. (1959). Variable Metric Method for Minimization, ANL-5990 Physics & Mathematics, AEC Research & Development Report, Argonne National Laboratory: Lemont, IL.

Davidon, W.C. (1976). New least-square algorithms. J Optim Theory Applic 18: 187-197.

Davidon, W.C. (1977). Fast least squares algorithms. Amer J Phys 45: 260-262.

Day, Achiya (1985). Successive refinement of large multicell models. SIAM J Numer Anal 22: 865-887.

DeMaine, P.A.D. (1978). Automatic curve-fitting I. Comp Chem 2: 1-6.

DeMaine, P.A.D. & Springer, G.K. (1974). A non-statistical program for automatic curve-fitting and non-linear equations. Mgmt Info 3: 233-250.

Dembo, R.S. (1976). A set of geometric programming test problems and their solutions. Math Prog 10: 192-214.

Dembo, R.S., Eisenstat, S.C. & Steihaug, T. (1982). Inexact Newton methods, SIAM J Numer Anal 19: 400-408.

Dembo, R.S. & Steihaug, T. (1983). Truncated-Newton algorithms for large-scale unconstrained optimization. Math Prog 26: 190-212.

Dembo, R.S. & Steihaug, T. (1983). A test problem generator for large-scale unconstrained optimization. ACM Trans Math Softw 11: 97-102.

Dempster, M.A.H. (1972). Optimization Theory. Mathematical Institute: Oxford, UK.

Dennis, J.E. Jr. (1982). Algorithms for nonlinear fitting. In Powell (1981) pp. 67-78.

Dennis, J.E. Jr., Gay, D.M. & Welsch, R.E. (1981). An Adaptive Nonlinear Least-Squares Algorithm. ACM Trans Math Softw 7: 348-368.

Dennis, J.E. Jr., & Schnabel, R. (1983). Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall: Englewood Cliffs, NJ.

Dixon, L.C.W. (1972). Nonlinear Optimisation. The English Universities Press: London.

Dixon, L.C.W. (1973). Conjugate directions without linear searches. J Inst Math Applic 11: 317-328.

Dixon, L.C.W. (1974). A survey of the state of the art. In Evans (1974), pp. 193-218.

Dixon, L.C.W. (1975). Conjugate gradient algorithms: quadratic termination without linear searches. J Inst Math Applic 15: 9-18.

Dixon, L.C.W. & Szegö, G.P.(eds.) (1975). Toward Global Optimization, North-Holland: Amsterdam/Oxford and American Elsevier: New York.

Dixon, L.C.W. & Szegö, G.P.(eds.) (1978). Toward Global Optimization 2, North-Holland: Amsterdam/Oxford and American Elsevier: New York.

Donaldson, J.R. & Schnabel, R.B. (1986). Computational Experience with Confidence Regions and Confidence Intervals for Nonlinear Least Squares. Computer Science and Statistics: 17th Symposium on the Interface, D.M.Allen (ed.), Springer: New York, pp. 83-93.

Draper, N.R. & Smith, H. (1981). Applied Regression Analysis, Second Edition. John Wiley & Sons: New York/Toronto.

Duggleby, R. (1984a). DNRP 53 User's Manual. Dept. of Biochemistry, Univ. of Queensland: Queensland, Australia.

Duggleby, R. (1984b). Regression analysis of nonlinear Arrhenius plots -- an empirical model and a computer program. Computers in Biology and Medicine, 14: 447-455.

Duncan, G.T. (1978). An empirical study of jackknife-constructed confidence regions in nonlinear regression. Technometrics 20: 123-129.

Eason, E.D. & Fenton, R.G. (1972). Testing and Evaluation of Numerical Methods for Design Optimization, UTME-TP7204. Department of Mechanical Engineering, Univ. of Toronto: Toronto, Ontario.

Eason, E.D. & Fenton, R.G. (1973). A comparison of numerical optimization methods for engineering design. Journal of Engineering for Industry, Transactions of the American Society for Mechanical Engineering, Paper no. 73-DET-17, pp. 1-5.

Efron, B. (1982). The Jackknife, the Bootstrap and Other Resampling Plans, CBMS-NSF, Regional Conference Series in Applied Mathematics. SIAM: Phildelphia.

Eggers, D.F., Gregory, N.W., Halsey, G.D. & Rabinovitch, B.S. (1964). Physical Chemistry. John Wiley & Sons: New York/London.

Endrenyi, L. (ed.) (1981). Kinetic Data Analysis: Design and Analysis of Enzyme & Pharmacokinetic Experiments. Plenum Press: New York/London.

Evans, D.J. (ed.) (1974). Software for Numerical Mathematics. Academic Press: London.

Evans, J.W. & Craig, R.J. (1979). Function Minimization Using a Modified Nelder-Mead Simplex Search Procedure, TR No. 144. Dept. of Statistics, Univ. of Kentucky: Lexington, KY.

Farazdaghi, H. & Harris, P.M. (1968). Plant competition and crop yield. Nature 217: 289-290.

Fiacco, A.V. & McCormick, G.P. (1964). Computational algorithm for the sequential unconstrained minimization technique for nonlinear programming. Mgmt Sci 10: 601-617.

Fiacco, A.V. & McCormick, G.P. (1966). Extensions of SUMT for nonlinear programming:equality constraints and extrapolation, Mgmt Sci 12: 816-828.

Fleck, R. & Bailey, J. (1975). Algorithm 87: Minimum of a nonlinear function by the application of the geometric programming technique. Comput J 18: 86-89.

Fletcher, R. (1969). Optimization (Proceedings of a Symposium of the Institute of Mathematics and its Applications, Univ. of Keele, 1968), Academic Press: London.

Fletcher, R. (1970). A new approach to variable metric algorithms. Comput J 13: 317-322.

Fletcher, R. (1971). A Modified Marquardt Subroutine for Nonlinear Least Squares, AERE - R 6799, Mathematics Branch, Theoretical Physics Division, Atomic Energy Research Establishment: Harwell, UK.

Fletcher, R. (1972). A FORTRAN Subroutine for Minimization by the Method of Conjugate Gradients, AERE - R 7073, Theoretical Physics Division, Atomic Energy Research Establishment: Harwell, UK.

Fletcher, R. (1980a). Practical Methods of Optimization, Vol. 1: Unconstrained Optimization. John Wiley & Sons: New York/Toronto.

Fletcher, R. (1980b). Practical Methods of Optimization, Vol. 2: Constrained Optimization. John Wiley & Sons: New York/Toronto.

Fletcher, R. & Powell, M.J.D. (1963). A rapidly convergent descent method for minimization. Comput J 6: 163-168.

Fletcher, R. & Reeves, C.M. (1964). Function minimization by conjugate gradients. Comput J 7: 149-154.

Fox, T., Hinkley, D. & Larntz, K. (1980). Jackknifing in nonlinear regression. Technometrics 22: 29-33.

Gallant, A.R. (1975). Nonlinear regression. Amer Stat 29: 74-81.

Gallant, A.R. (1977). Three-stage least-squares estimation for a system of simultaneous, nonlinear, implicit equations. J Econ 5: 71-88.

Garfinkel, L., Kohn, M.C., Garfinkel, D. & Endrenyi, L. (1977). Systems analysis in enzyme kinetics: Review. CRC Crit Rev Bioeng 2: 329-361.

Gass, S.I. (1964). Linear Programming, Second Edition. McGraw-Hill: New York/Toronto.

Gauss, K.F. (1809). Theoria Motus Corporum Coelestiam. Werke, Bd. 7: 240-254.

Gay, D.M. (1983). Remark on algorithm 573 (NL2SOL:an adaptive nonlinear least squares algorithm). ACM Trans Math Softw 9: 139.

Gentleman, W.M. & Marovich, S.B. (1974). More on algorithms that reveal properties of floating point arithmetic units. Comm ACM 17: 276-277.

Gerald, K.B. & Matis, J.H. (?). Comparison of Weighted and Unweighted Nonlinear Least Squares in Compartmental Modeling. (Unpublished).

Gerald, K.B. & Matis, J.H. (?). The Effect of Incorrect Initial Values on Least Squares Estimates in a Two Compartment Model. (Unpublished).

Gill, P.E. & Murray, W. (1978). Algorithms for the solution of the nonlinear least squares problem. SIAM J Numer Anal 15: 977-992.

Gill, P.E. & Murray, W. (1979). Conjugate-Gradient Methods for Large-Scale Nonlinear Optimization, Tech. Report SOL 79-16. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Gill. P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1979). Two Step-Length Algorithms for Numerical Optimization, Tech. Report SOL 79-25. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1980a). Computing Finite-Difference Approximations to Derivatives for Numerical Optimization, Tech. Report SOL 80-6. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1980b). Aspects of Mathematical Modeling Related to Optimization, Tech. Report SOL 80-7. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1980c). Methods for Large-Scale Nonlinear Optimization, Tech. Report SOL 80-8. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1982). A note on a sufficient-decrease criterion for a non-derivative step-length procedure. Math Prog 23: 349-352.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1983). Computing forward-difference intervals for numerical optimization. SIAM J Sci Stat Comp 4: 310-321.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1984a). Procedures for optimization problems with a mixture of bounds and general linear constraints. ACM Trans Math Softw 10: 282-298.

Gill, P.E., Murray, W., Saunders, M.A. & Wright, M.H. (1984b). Sparse matrix methods in optimization. SIAM J Sci Stat Comp 5: 562-589.

Gill, P.E., Murray, W. & Wright, M.H. (1981). Practical Optimization. Academic Press: London.

Gillis, P.R. & Ratkowsky, D.A. (1978). The behaviour of estimators of the parameters of various yield-density relationships. Biomc 34: 191-198.

Goldberg, M.L., Bates, D. & Watts, D.G. (1983). Simplified methods of assessing nonlinearity. 1983 Proceedings of the Business & Economics Section. American Statistical Association: Washington, DC.

Goldfarb, D. & Lapidus, L. (1968). Conjugate gradient method for nonlinear programming problems with linear constraints. Ind Eng Chem 7: 142-151.

Golub, G.H. & LeVeque, R.J. (1979). Extensions and Uses of the Variable Projection Algorithm for Solving Nonlinear Least Squares Problems, ARO Report 79-3. Proceedings of the 1979 Army Numerical Analysis & Computers Conference: Stanford, CA.

Golub, G.H. & Pereyra, V. (?). Differentiation of Pseudoinverses, Separable Nonlinear Least Squares Problems, and Other Tales. Computer Science Dept., Stanford Univ.: Stanford, CA.

Golub, G.H. & Pereyra, V. (1973). The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. SIAM J Numer Anal 10: 413-432.

Golub, G.H. & Van Loan, C.F. (1983). Matrix computations. Johns Hopkins University Press: Baltimore MD.

Goodall, C. (1983). M-Estimators of Location: an Outline of the Theory. In Hoaglin et al.(1983), Chapter 11.

Guttman, I. & Meeter, D.A. (1965). On Beale's measures of nonlinearity. Technometrics 7: 623-637.

Hamilton, D. (1984a). Confidence Regions for Parameter Subsets in Nonlinear Regression, TR 84-2. Mathematics, Statistics & Computing Science, Dalhousie Univ.: Halifax, Nova Scotia.

Hamilton, D. (1984b). On an Alternate Exact Confidence Region for Certain Nonlinear Regression Models, TR 84-11. Mathematics, Statistics & Computing Science, Dalhousie Univ.: Halifax, Nova Scotia.

Hamilton, D.C. & Watts, D.G. (1985). A quadratic design criterion for precise estimation in nonlinear regression models. Technometrics 27: 241-250.

Hamilton, D.C., Watts, D.G. & Bates, D.M. (1982). Accounting for intrinsic nonlinearity in nonlinear regression parameter inference regions. Ann Stat 10: 386-393.

Hartley, H.O. (1948). The estimation of nonlinear parameters by 'internal least squares'. Biomk 35: 32-45.

Hartley, H.O. (1961). The modified Gauss-Newton method for the fitting of non-linear regression functions by least squares. Technometrics 3: 269-280.

Hartley, H.O. & Booker, A. (1965). Nonlinear least squares estimation. Ann Math Stat 36: 638-650.

Hathaway, R.J. (1983). Constrained maximum likelihood estimation for normal mixtures. In Computer Science and Statistics: 15th Symposium on the Interface, J.E. Gentle (ed.), North-Holland: Amsterdam. pp. 263-267.

Havriliak, S. Jr. & Watts, D.G. (?). Analytical Representation of Dielectric Constants: a Complex Multiresponse Problem. Dept. of Mathematics and Statistics, Queen's Univ.: Kingston, Ontario. (Unpublished).

Herning, R.I., Jones, R.T., Benowitz, N.L. & Mines, A.H. (1983). How a cigarette is smoked determines blood nicotine levels. Clin Pharmacol Ther 33: 84-90.

Hestenes, M.R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. J Res Nat Bur Stand 49: 409-436.

Hill, G.W. (1981a). Remark on Algorithm 395. ACM Trans Math Softw 7: 247-249.

Hill, G.W. (1981b). Remark on algorithm 396. ACM Trans Math Softw 7: 250-251.

Hinkley, D. (1977). Jackknifing in unbalanced situations. Technometrics 19: 285-292.

Hoaglin, D.C., Mosteller, F. & Tukey, J.W. (eds.) (1983). Understanding Robust and Exploratory Data Analysis. John Wiley & Sons: New York/Toronto.

Hock, W. & Schittkowski, K. (1981). Test examples for nonlinear programming codes., Lecture notes in Economics and Mathematical Systems, 187, Springer: Berlin.

Hocking, R.R. (1983). Developments in linear regression methodology: 1959-1982. Technometrics 25: 219-250.

Hocking, R.R. (1984). Response to David A. Belsley, Technometrics 26: 299-301.

Holliday, R. (1960). Plant population and crop yield. Field Crop Abstracts 13: 159-167, 247-254.

Holt, J.N. & Antill, R.J. (1977). Determining the number of terms in a Prony algorithm exponential fit. Math Biosci, 36: 319-332.

Holt, J.N. & Jupp, D.L.B. (1978). Free-knot spline inversion of a Fredholm integral equation from astrophysics. J Inst Math Applic. 21: 429-443.

Holt, J.N. (1978). Nonlinear least squares inversion of an integral equation using free knot cubic splines. Proc. 8th IFIP Conference on Optimization Techniques. Part 2, Sept. 5-7, 1977, J. Stoer (ed.) Springer: Berlin, pp. 51-58.

Holt, J.N. & Fletcher, R. (1979). An algorithm for constrained nonlinear least squares. J Inst Math Applic 23: 449-463.

Holt, J.N. & Bracken, A.J. (1980). First kind Fredholm integral equations of liver kinetics: numerical solutions by constrained least squares. Math Biosci 51: 11-24.

Hooke, R. & Jeeves, T.A.,(1961). "Direct Search" solution of numerical and statistical problems. J ACM 8: 212-229.

Hougaard, P. (1982). Parameterizations of nonlinear models. J Roy Stat Soc B, 44: 244-252.

IMSL STAT/LIBRARY User's Manual (1984). International Mathematical and Statistical Libraries: Houston, TX.

Intriligator, M.D. (1978). Econometric Models, Techniques, and Applications. Prentice-Hall: Englewood Cliffs, NJ.

Jacoby, S.L.S., Kowalik, J.S. & Pizzo, J.T. (1972). Iterative Methods for Nonlinear Optimization Problems. Prentice-Hall: Englewood Cliffs, NJ.

James, F. & Roos, M. (1975). MINUIT - a system for function minimization and analysis of the parameter errors and correlations. Comput Phys Comm 10: 343-367.

Jennrich, R.I. (1969). Asymptotic properties of nonlinear least squares estimators. Ann Math Stat 40: 633-643.

Jennrich, R.I. & Bright, (1976). Fitting systems of linear differential equations using computer generated exact derivatives. Technometrics 18: 385-392, with Discussion by A. Wiggins, 393-399.

Jennrich, R.I. & Sampson, P.F. (1968). Application of stepwise regression to non-linear estimation. Technometrics 10: 63-72.

Jennrich, R.I. & Sampson, P.F. (1976). Newton-Raphson and related algorithms for maximum likelihood variance component estimation. Technometrics 18: 11-17.

Johnston, J. (1972). Econometric Methods, Second Edition. McGraw-Hill: New York.

Jones, A. (1970). Spiral - a new algorithm for non-linear parameter estimation using least squares. Comput J 13: 301-308.

Kammler, D.W. (1979). Least squares approximation of completely monotonic functions by sums of exponentials. SIAM J Numer Anal 16: 801-818.

Kapsalis, C. (1984). Economic sumulation: a demonstration of the simultaneous equation technique. Unpublished.

Karpinski, R. (1985). PARANOIA: a floating-point benchmark. Byte 10 (2): 223-235 (February).

Kaufman, L. (1975). A variable projection method for solving separable nonlinear least squares problems. BIT 15: 49-57.

Kendall, M.G. & Stuart, A. (1973). The Advanced Theory of Statistics. Griffin: London.

Kennedy, W.J. Jr. & Gentle, J.E. (1980). Statistical Computing. Marcel Dekker: New York.

Khorsani, F. & Milliken, G.A. (1982). Simultaneous confidence bands for nonlinear regression. Comm Stat - Theory and Methods 11: 1241-1253.

Kimeldorf, G., May, J.H. & Sampson, A.R. (1980). Concordant and Discordant Monotone Correlations and Their Evaluation by Nonlinear Optimization, TR 80-20. Inst. for Statistics & Applications, Dept. of Mathematics & Statistics., Univ. of Pittsburgh: Pittsburgh.

Kmenta, J. (1971). Elements of Econometrics. Macmillan: New York.

Kohn, M.C., Menten, L.E. & Garfinkel, D. (1979). Convenient computer program for fitting automatic rate laws to steady state data. Comput Biom 12: 461-469.

Kowalik, J. & Osborne, M.R. (1968). Methods for Unconstrained Optimization Problems. American Elsevier: New York.

Krogh, F.T. (1974). Efficient implementation of a variable projection algorithm for nonlinear least squares problems. Comm ACM 17: 167-169.

Kuester, J.L. & Mize H.H. (1973). Optimization Techniques with FORTRAN. McGraw-Hill: New York/London/Toronto.

Laidler, K.J. (1965). Chemical Kinetics, Second Edition. McGraw-Hill: New York/Toronto.

Lanczos, C. (1956). Applied Analysis. Prentice-Hall: Englewood Cliffs, NJ.

Landriani, G.S., Guardabasso, V. & Rocchetti, M. (1983). NL-FIT: a microcomputer program for non-linear fitting. Comput Prog Biom 16: 35-42.

Lang, J. & Muller, R. (1971). A procedure for nonlinear least squares refinement in adverse practical conditions. Comput Phys Comm 2: 79-86.

Lawton, W.H. & Sylvestre, E.A. (1971). Elimination of linear parameters in nonlinear regression. Technometrics 13: 461-467.

Lehmer, D.H. (1943). Ramanujan's function. Duke Math J 10: 483-492.

Lemaréchal, C. (1981). Nondifferentiable optimization. In Powell (1981) pp. 85-89.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. Q Appl Math 2: 164-168.

Lill, S.A. (1976). A survey of methods for minimizing sums of squares of nonlinear functions. In Optimization in Action, Proc. of the Conference on Optimization in Action, Univ. of Bristol, L.C.W. Dixon (ed.), Academic Press:London, pp. 1-26.

Linssen, H.N. (1975). Nonlinearity measures: a case study. Statistica Neerlandica 29: 93-99.

Lo, K.L., Ong, P.S., McColl, R.D., Moffatt, A.M., Sulley, J. & Moyes, I. (1982). Power system state estimation: a method for metering reinforcement, IEEE Power, PAS-101: 3493-3501.

Looney, S.W. & Bergmann, R.E. (?). Procedures for Fitting a Mixture of Weibull Distributions to an Empirical Growth Function. (Unpublished).

Lootsma, F.A.(ed.) (1972). Numerical Methods for Non-Linear Optimization. Academic Press:London/New York.

Maddison, R.N. (1961). A procedure for nonlinear least squares refinement in adverse practical conditions. J ACM 13: 124-134.

Makridakis, S., Wheelwright, S.C., and McGee, V.E. (1983). Forecasting: methods and applications. John Wiley & Sons: New York.

Malcolm, M.A. (1972). Algorithms to reveal properties of floating-point arithmetic. Comm ACM 15: 949-951.

Malloy, R. (1986). Print screen in BASIC. Byte 11 (2): 384 (February).

Marquardt, D.W. (1959). Solution of nonlinear chemical engineering models. Chem Eng Prog 55 (6): 65-70 (June).

Marquardt, D.W. (1963). An algorithm for least-squares estimation of nonlinear parameters. J SIAM 11: 431-441.

Marquardt, D.W. (1964). Least-Squares Estimation of Nonlinear Parameters, IBM Share Library, Distribution #3094, NLIN, Engineering Department, E.I.DuPont de Nemours & Co., Wilmington, DE. See Marquardt (1966) for revised version.

Marquardt, D.W. (1966). Least-Squares Estimation of Nonlinear Parameters, IBM Share Library, Distribution #309401, NLIN 2, Engineering Department, E.I.DuPont de Nemours & Co., Wilmington, DE.

Marquardt, D.W. (1970). Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation. Technometrics 12: 591-612.

McCullagh, P. & Nelder, J.A. (1983). Generalized Linear Models. Chapman & Hall: London/New York.

McKeown, J.J. (1973). A comparison of methods for solving nonlinear parameter estimation problems. In Identification & System Parameter Estimation, Proc. of the 3rd IFAC Symposium, P. Eykhoff (ed.): The Hague/Delft pp. 12-15.

McKeown, J.J. (1974). Specialised versus General-Purpose Algorithms for Minimising Functions that are Sums of Squares, Tech. Report 50, Issue 2, The Hatfield Polytechnic, Hatfield, UK.

McKeown, J.J. (1978). On Algorithms for Sums of Squares Problems. In Dixon & Szegö (1978), pp. 313-327.

McKeown, J.J. (1979). Experiments in Implementing a Nonlinear Least Squares Algorithm on a Dual-Processor Computer, TR # 102. Numerical Optimisation Centre, The Hatfield Polytechnic: Hatfield, UK.

McLean, D.D., Bacon, D.W. & Downie, John (1980). Statistical identification of a reaction network using an integral plug flow reactor. Can J Chem Eng 58: 608-619.

Meade, N. (1984). The use of growth curves in forecasting market development -- a review and appraisal. J Forecasting 3: 429-451.

Meade, N. (1985). Forecasting using growth curves -- an adaptive approach. J Oper Res Soc 36: 1103-1115.

Meeter, D.A. (1966). On a theorem used in nonlinear least squares. SIAM J Appl Math 14: 1176-1179.

Metzler, C.H. (1981). Statistical properties of kinetic estimates. In Endrenyi (1981), pp. 25-37.

Meyer, R.R. (1970). Theoretical and computational aspects of nonlinear regression. In Nonlinear Programming, J.B.Rosen, O.L.Mangasarian, K.Ritter (eds.). Academic Press: New York/London pp.465-486.

Meyer, R.R. & Roth, P.M. (1972). Modified damped least squares: an algorithm for non-linear estimation. J Inst Math Applic 9: 218-233.

Microsoft Corporation (1983). Microsoft BASIC Reference Manual. Microsoft Corporation: Bellevue, WA.

Miller, A.J. (1979). Problems in non-linear least squares estimation. In Interactive Statistics, D. McNeil (ed.), North-Holland: Sydney, Australia, pp. 39-51.

Mladineo, R.H. (1986). An algorithm for finding the global maximum of a multimodal, multivariate function. Math Prog 34: 188-200.

Moelwyn-Hughes, E.A. (1961). Physical Chemistry. Pergamon Press: Oxford.

Moler, C.M., and Van Loan, C.F. (1978). Nineteen dubious ways to compute the exponential of a matrix. SIAM Rev 20: 801-836.

Moré, J.J., Garbow, B.S. & Hillstrom, K.E. (1981). Testing unconstrained optimization software. ACM Trans Math Softw 7: 17-41.

Murray,W. & Wright, M.H. (1980). Computation of the Search Direction in Constrained Optimization Algorithms, Tech. Report SOL 80-2. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Nash, J.C. (1976). An annotated bibliography on methods for nonlinear least squares problems including test problems. Nash Information Services: Ottawa. (microfiche)

Nash, J.C. (1977). Minimizing a nonlinear sum of squares function on a small computer. J Inst Math Applic 19: 231-237.

Nash, J.C. (1979). Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation. Adam Hilger:Bristol & Halsted Press:New York.

Nash, J.C. (1980). Problèmes mathématiques soulevés par les modèles économiques. Can J Ag Econ, 28: 51-57.

Nash, J.C. (1981). Nonlinear estimation using a microcomputer, Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, (ed. W.F. Eddy), Spinger, New York, pp. 363-366.

Nash, J.C. (1984a). Effective scientific problem solving with small computers. Reston Publishing: Reston, Virginia. (all rights now held by J.C.Nash)

Nash, J.C. (1984b). LEQB05: User Guide - A Very Small Linear Algorithm Package. Nash Information Services Inc.: Ottawa.

Nash, J.C. (1985). Design and implementation of a very small linear algebra program package. Comm ACM, 28: 89-94.

Nash, J.C. (1986a). Review: IMSL MATH/PC-LIBRARY, to appear in Amer Stat, 40 (4) (November).

Nash, J.C. (1986b). Review: IMSL STAT/PC-LIBRARY, to appear in Amer Stat, 40 (4) (November).

Nash, J.C. (1986c). Microcomputers, standards, and engineering calculations. Proceedings of the 5th Canadian Conference on Engineering Education, U. of Western Ontario, May 12-13, 1986, pp. 302-316.

Nash, J.C. & Teeter, N.J. (1975). Building models: an example from the Canadian dairy industry. Can Farm Econ 10 (2): 17-24 (April).

Nash, J.C. & Walker-Smith, M. (1986). Using Compact and Portable Function Minimization Codes in Forecasting Applications. INFOR 24: 158-168.

Nash, S.G. (1982). Truncated-Newton Methods, Report STAN-CS-82-906. Dept. of Computer Science, Stanford Univ.: Stanford, CA.

Nash, S.G. (1983). Truncated-Newton methods for large-scale function minimization. In Applications of nonlinear programming to optimization and control, H.E.Rauch (ed.), Pergamon Press: Oxford, pp. 91-100.

Nash, S.G. (1984). Newton-type minimization via the Lanczos method. SIAM J Numer Anal 21: 770-788.

Nash, S.G. (1985a). Preconditioning of truncated-Newton methods. SIAM J Sci Stat Comp 6: 599-616.

Nash, S. G. (1985b). Solving nonlinear programming problems using truncated-Newton techniques. In Boggs, Byrd & Schnabel (1985) pp. 119-136.

Nazareth, L. (1976). Some recent approaches to solving large residual non-linear least squares problems. Computer Science & Statistics: 9th Annual Symposium on the Interface; Harvard Univ.: Cambridge, MA. (pages unavailable)

Nazareth, L. (1978). Software for Optimization, Tech. Report SOL 78-32. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Nazareth, L. (?). A Hybrid Least Squares Method. (Unpublished).

Nazareth, L. & Nocedal, J. (1978a). Properties of Conjugate Gradient Methods with Inexact Linear Searches, Tech. Report SOL 78-1. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Nazareth, L. & Nocedal, J. (1978b). A Study of Conjugate Gradient Methods, Tech. Report SOL 78-29. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Nelder, J.A. (1961). The fitting of a generalization of the logistic curve. Biomc 17: 89-110.

Nelder, J.A. (1962). An alternative form of a generalized logistic equation. Biomc 18: 614-616.

Nelder, J.A. & Mead, R. (1965). A simplex method for function minimization, Comput J 7: 308-313.

Neter, J. & Wasserman, W. (1974). Applied Linear Statistical Models. Richard D. Irwin Inc: Homewood, IL.

Nocedal, J. & Overton, M.L. (1985). Projected Hessian updating algorithms for nonlinearly constrained optimization. SIAM J Numer Anal 22: 821-850.

Oliver, F.R. (1964). Methods of estimating the logistic growth function. Appl Stat 13: 57-66.

Oliver, F.R. (1966). Aspects of maximum likelihood estimation of the logistic growth function. JASA 61: 697-705.

Olsson, D.M. & Nelson, L.S. (1975). The Nelder-Mead simplex procedure for function minimization. Technometrics 17: 45-51; Letters to the Editor: 393-394.

O'Neill, R. (1971). Algorithm AS 47: function minimization using a simplex procedure. Appl Stat 20: 338-345.

Osborne, M.R. (1972a). A class of methods for minimising a sum of squares. Austral Comput J 4 (4): 164-169.

Osborne, M.R. (1972b). An algorithm for discrete, nonlinear, best approximation problems. In Numerische Methoden der Approximations Theorie, Band 1, L. Collatz & G. Meinardus (eds.), Birkhauser Verlag: Basel/Stuttgart, pp. 117-126.

Osborne, M.R. (1975). Some special nonlinear least squares problems. SIAM J Numer Anal 12: 571-591.

Overton, M.L. (1981). Algorithms for nonlinear L1 and L-infinity fitting. In Powell (1981) pp. 91-101.

Parkinson, J.M. & Hutchinson, D. (1972a). A consideration of non-gradient algorithms for the unconstrained optimization of functions of high dimensionality. In Lootsma (1972) pp. 99-113.

Parkinson, J.M. & Hutchinson, D. (1972b). An investigation into the efficiency of variants on the simplex method. In Lootsma (1972) pp. 115-135.

Parson, L.A. (1983). A lower bound for the norm of the theta operator. Math Comput 41: 680-685.

Peck, C.C. & Barrett, B.B. (1979). Nonlinear least-squares regression programs for micro-computers. J Phar Biop 7: 537-541.

Peckham,G. (1970). A new method for minimising a sum of squares without calculating gradients. Comput J 13: 418-420.

Pedersen, P.V. (1977). Curve fitting and modeling in pharmacokinetics and some practical experiences with NONLIN and a new program FUNFIT. J Phar Biop 5: 513-553.

Pereyra, V. (1967). Iterative methods for solving least squares problems. SIAM J Numer Anal 4: 27-36.

Pfeffer, M. (1973). COMPT, a time-sharing program for nonlinear regression analysis of compartmental models of drug distribution. J Phar Biop 1: 137-163.

Phillips, D.A. (1974). A preliminary investigation of function optimisation by a combination of methods. Comput J 17: 75-79.

Polak, E. & Ribiere, G. (1969). Note sur la convergence de méthodes de directions conjugées. R.I.R.O. 3 (16): 35-43.

Powell, D.R. & Macdonald, J.R. (1972). A rapidly convergent iterative method for the solution of the generalised nonlinear least squares problem. Comput J 15: 148-155.

Powell, M.J.D. (1962). An iterative method for stationary values of a function of several variables, Computer J 5: 147-151.

Powell, M.J.D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. Comput J 7: 155-162.

Powell, M.J.D. (1975a). Some Convergence Properties of the Conjugate Gradient Method, C.S.S. 23. Computer Science and Systems Division, Atomic Energy Research Establishment: Harwell, UK.

Powell, M.J.D. (1975b). Restart Procedures for the Conjugate Gradient Method, C.S.S. 24. Computer Science and Systems Division, Atomic Energy Research Establishment: Harwell, UK.

Powell, M.J.D. (1981). Nonlinear Optimization 1981. Academic Press: London.

Pritchard, D.J. & Bacon, D.W. (1977). Accounting for heteroscedasticity in experimental design. Technometrics 19: 109-115.

Pritchard, D.J. Downie, J. & Bacon, D.W. (1977). Further consideration of heteroscedasticity in fitting kinetic models. Technometrics 19: 227-236.

Protter, M.H. & Morrey, C.B.,Jr. (1964). College Calculus with Analytic Geometry. Addison-Wesley: Reading, MA.

Ralston, M.L. & Jennrich, R.I. (1978). DUD, a derivative-free algorithm for nonlinear least squares. Technometrics 20: 7-14.

Ramsin, H. & Wedin, P-A. (1977). A comparison of some algorithms for the nonlinear least squares problem. BIT 17: 72-90.

Rasor, E.A. (1949). The fitting of logistic curves by means of a nomograph. JASA 44: 548-553.

Ratkowsky, David A. (1983). Nonlinear Regression Modeling. Marcel Dekker: New York.

Reich, J.G. (1981). On parameter redundancy in curve fitting of kinetic data. In Endrenyi (1981) pp. 39-50.

Rheinboldt, W.C. (1974). Methods for Solving Systems of Nonlinear Equations. SIAM: Philadelphia.

Rice, J. (1983). Numerical methods, software and analysis. McGraw-Hill: New York.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. Comput J 3: 175-184.

Ross, G.J.S. (1971). The efficient use of function minimization in non-linear maximum-likelihood estimation. Appl Stat 19: 205-221.

Ross, G.J.S. (1975). Simple non-linear modelling for the general user. 40th Session of the International Statistical Institute: Warsaw. 1-9 September 1975, ISI/BS Invited Paper 81 pp. 1-8.

Ruhe, A. (1979). Accelerated Gauss-Newton algorithms for nonlinear least squares problems. BIT 19: 356-367.

Ruhe, A. (1980). Fitting empirical data by positive sums of exponentials. SIAM J Sci Stat Comp 1: 481-497.

Ruhe, A. & Wedin, P-A. (1974). Algorithms for separable nonlinear least squares problems. Report STAN-CS-74-434, Stanford Univ.: Stanford, CA.

Ruhe, A. & Wedin, P-A. (1980). Algorithms for separable nonlinear least squares problems. SIAM Rev 22: 318-336.

SAS User's Guide: Statistics, 1982 edition (1982). SAS Institute: Cary, NC.

Saunders, M.A. (1979). Sparse Least Squares by Conjugate Gradients: A Comparison of Preconditioning Methods, Tech. Report SOL 79-5. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Schey, J.A. (1977). Introduction to Manufacturing Processes. McGraw-Hill: New York.

Schiff, L.I. (1955). Quantum Mechanics. McGraw-Hill: New York.

Schnabel, R.B., Koontz, J.E., & Weiss, B.E. (1985). A modular system of algorithms for unconstrained minimization, ACM Trans Math Softw 11: 419-440.

Schnute, J. (1981). A versatile growth model with statistically stable parameters. Can J Fish 38: 1128-1140.

Schnute, J. (1982). A Manual for Easy Nonlinear Parameter Estimation in Fishery Research with Interactive Microcomputer Programs, Canadian Tech. Report of Fisheries & Aquatic Sciences 1140. Dept. of Fisheries and Oceans, Government of Canada.

Schnute, J. & Fournier, D. (1980). A new approach to length-frequency analysis: growth structure. Can J Fish 37: 1337-1351.

Schnute, J. & Sibert, J. (1983). The salmon terminal fishery: a practical, comprehensive model. Can J Fish 40: 835-853.

Schwenke, J.R. & Milliken, G.A. (1986). On the calibration problem extended to nonlinear models. To be published in Technometrics.

Schwetlick, H. (1979). Numerische Losung Nichtlinearer Gleichungen. VEB Deutscher Verlag der Wissenschaften: Berlin.

Sedman, A.J. & Wagner, J.G. (1976). CSTRIP, a FORTRAN IV computer program for obtaining initial polyexponential parameter estimates. J Pharm Sci 65: 1006-1010.

Shanno, D.F. & Rocke, D.M. (1986). Numerical methods for robust regression:linear models. SIAM J Sci Stat Comp 7: 86-97.

Shearer, J.M. & Wolfe, M.A. (1985). Alglib, a simple symbol-manipulation package. Comm ACM 28: 820-825.

Simonoff, J.S. & Tsai, C-L. (1986). Jackknife-based estimators and confidence regions in nonlinear regression. Technometrics 28: 103-112.

Smith, F.B. Jr. & Shanno, D.F. (1971). An improved Marquardt procedure for nonlinear regressions. Technometrics 13: 63-74.

Sommerfeld, A. (1964). Mechanics. Academic Press: London.

Sorenson, H.W. (1969). Comparison of some conjugate direction procedures for function minimization. J Franklin Inst 288: 421-441.

Spang, H.A. (1962). A review of minimization techniques for nonlinear functions. SIAM Rev 4: 343-365.

Spendley, W. (1969). Nonlinear least squares fitting using a modified Simplex minimization method. In Fletcher (1969) pp. 259-270.

Spendley, W., Hext, G.R. & Himsworth, F.R. (1962). Sequential application of simplex designs in optimisation and evolutionary operation. Technometrics 4: 441-461.

SPSS Statistical Package for the Social Sciences, Second Edition (1975). McGraw Hill: New York/Toronto.

Stevens, W.L. (1951). Asymptotic regression. Biomc 7: 247-267.

Swann, W. H. (1964). Report on the Development of a New Direct Search Method of Optimization. Imperial Chemical Industries, Central Instrument Research Laboratory Research Note 64/3.

Swann, W. H. (1974). Direct search methods. In Numerical Methods for Unconstrained Optimization, W. Murray (ed.), Academic Press: London/New York.

Tabata, T. & Ito, R. (1975). Effective treatment of the interpolation factor in Marquardt's nonlinear least-squares fit algorithm. Comput J 18: 250-251.

Talpaz, H. (1976). Nonlinear estimation by an efficient numerical search method. Rev Econ Stat 58: 501-504.

Teller, H. & Teller, U. (1981). A simple numerical treatment of the RKR potential integrals and its application to 12C 16O (X' sigma+). J Mol Spectr, 85: 248-252.

Toint, Ph.L. (1978). Some numerical results using a sparse matrix updating formula in unconstrained optimization. Math Comput 32: 839-851.

Valliant, R. (1985). Nonlinear prediction theory and the estimation of proportions in a finite population. JASA 80: 631-641.

Van Loan, C. (1976). Lectures in Least Squares, TR 76-279. Department of Computer Science, Cornell Univ.: Ithaca, NY.

Varah, J.M. (1985). On fitting exponentials by nonlinear least squares. SIAM J Sci Stat Comp 6: 30-44.

Vieira, S. & Hoffmann, R. (1977). Comparison of the logistic and the Gompertz growth functions considering additive and multiplicative error terms. Appl Stat 26: 143-148.

Von Meerwall, E. (1976). A flexible all-purpose curve-fitting program. Comput Phys Comm 11: 211-219.

Walling, D. (1968). Nonlinear least squares curve fitting when some parameters are linear. Texas J Sci 20: 119-124.

Wampler, Roy H. (1977). Software for Nonlinear and Generalized Least Squares, Report W-77-1. National Bureau of Standards, Statistical Engineering Laboratory: Washington, DC.

Ward, L., Nag, A. & Dixon, L.C.W. (1969). Hill-climbing techniques as a method of calculating the optical constants and the thickness of a thin metallic film. Brit J Appl Phys Ser 2, 2: 301-304.

Watts, D.G. (1981). An introduction to nonlinear least squares. In Endrenyi (1981) pp. 1-24.

Watts, D.G. & Broekhoven, L.H. (1973). An Algorithm for Partitioned Least Squares (PLEAS), Preprint 1973-12, Dept. of Mathematics, Queen's Univ.: Kingston, Ontario.

Wilk, M.B. (1958). An identity of use in nonlinear least squares. Ann Math Stat 29: 618.

Wilkinson, J.H. (1965). The algebraic eigenvalue problem.

Clarendon Press: Oxford.

Williams, E.J. (1962). Exact fiducial limits in non-linear estimation. J Roy Stat Soc B, 24: 125-139.

Wold, H. (1966). Nonlinear estimation by iterative procedures. In Research Papers in Statistics, (F.N. David, ed.). John Wiley & Sons: London/New York.

Wolfe, M.A. (1978). Numerical Methods for Unconstrained Optimization, an Introduction. Van Nostrand Reinhold: Wokingham, MA.

Wright, Margaret H. (1978). A Survey of Available Software for Nonlinearly Constrained Optimization, TR SOL 78-4. Systems Optimization Laboratory, Stanford Univ.: Stanford, CA.

Wright, S.J. & Holt, J.N. (1985a). An inexact Levenberg-Marquardt method for large, sparse nonlinear least squares J Austral Math Soc B 26: 387-403.

Wright, S.J. & Holt, J.N. (1985b). Algorithms for nonlinear least squares with linear inequality constraints. SIAM J Sci Stat Comp 6: 1033-1048.

Zellner, A. (1986). Bayesian estimation and prediction using asymmetric loss functions. JASA 81: 446-451.

COVER SHEET

Indices

Chapter title: Indices

John C. Nash
Faculty of Administration
University of Ottawa

Mary Walker-Smith
General Manager
Nash Information Services Inc

Nonlinear Parameter Estimation Methods
An Integrated System in BASIC

AUTHOR INDEX

SUBJECT INDEX