Comparative Methods of Computing Maximum Likelihood Estimates
for Non-Linear Econometric Systems

by

Yock Yoon CHONG
B.Sc.(Sci.), M.Sc.(Comp. Sci.)

1981

Thesis submitted for the
Degree of Doctor of Philosophy
at the
London School of Economics
University of London
To

My Mother

and the Fond Memory of

My Dear Father
I wish to thank my supervisor, Professor J.D. Sargan, for his continuous help, advice and encouragement throughout this study. The undertaking of this research would not have been possible without his constant supervision.

I would also like to thank Professor D.F. Hendry for his invaluable comments and suggestions, though his demand for clarity is not always met, and to Dr. H.R. Wills for many helpful discussions and suggestions on the computer programs development.

I am also grateful to the Queen Mary College DAP Support Unit for advice with the computing aspects of the DAP programs, and to the Social Science Research Council for the financial support which made this work possible.
This research is mainly concerned with numerical optimisation techniques applied to general non-linear econometric simultaneous equations systems. The method of estimation used is maximum likelihood. An estimation program which applies gradient-type procedures, specifically the Berndt-Hall-Hall-Hausman and Gill-Murray-Pitfield methods, is developed. This program allows the estimation of a general small-to-medium size model which is non-linear in parameters, variables or both. In the course of program development, a general differentiation program is written which will differentiate a set of econometric equations and thus provide the analytical gradients for the optimisation procedures. A comparative study has been made of the relative efficiency of the two methods by running a set of simulated non-linear models and also using a small macro-economic model of the British Economy specified by David F. Hendry. To improve the efficiency of the estimation program in terms of computing time, the Berndt-Hall-Hall-Hausman method was implemented on the ICL Distributed Array Processor (DAP) which employs parallel computations. The DAP runs show that for a model with a large sample size, the DAP is approximately 30 times faster than the conventional computer CDC 7600, but that for the present algorithm, the latter is a more efficient alternative for small sample sizes.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>3</td>
</tr>
<tr>
<td>Abstract</td>
<td>4</td>
</tr>
<tr>
<td>Chapter 1 Survey and Literature</td>
<td>9</td>
</tr>
<tr>
<td>1.0 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>1.1 Maximum Likelihood Estimator for Non-Linear Econometric Systems</td>
<td>10</td>
</tr>
<tr>
<td>1.2 Numerical Optimisation Applied to Non-Linear Econometric Models</td>
<td>16</td>
</tr>
<tr>
<td>1.3 Specific Application Program</td>
<td>18</td>
</tr>
<tr>
<td>Chapter 2 Numerical Optimisation Techniques For Non-Linear Econometric Models</td>
<td>21</td>
</tr>
<tr>
<td>2.0 Numerical Optimisation</td>
<td>21</td>
</tr>
<tr>
<td>2.1 Optimisation Methods</td>
<td>21</td>
</tr>
<tr>
<td>2.2 Minimisation With Derivatives</td>
<td>23</td>
</tr>
<tr>
<td>2.2.1 The Newton-Raphson Method</td>
<td>24</td>
</tr>
<tr>
<td>2.2.2 Variable-metric Methods</td>
<td>33</td>
</tr>
<tr>
<td>2.2.3 General Gradient Method</td>
<td>35</td>
</tr>
<tr>
<td>2.2.4 The Method of Fletcher and Powell (DFP)</td>
<td>36</td>
</tr>
<tr>
<td>2.2.5 The Complementary DFP Updating Formula (CompDFP)</td>
<td>38</td>
</tr>
<tr>
<td>2.2.6 Gill-Murray-Pitfield Method (GMP)</td>
<td>39</td>
</tr>
<tr>
<td>2.2.7 The BHHH Hessian Approximation</td>
<td>42</td>
</tr>
<tr>
<td>2.3 Minimisation Without Derivatives</td>
<td>44</td>
</tr>
<tr>
<td>2.3.1 Conjugate Direction Methods</td>
<td>44</td>
</tr>
<tr>
<td>2.3.2 Variable-metric Methods</td>
<td>47</td>
</tr>
<tr>
<td>2.3.3 Comments Regarding Minimisation Without Derivatives</td>
<td>51</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Choice of Line Search</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Quadratic Interpolation</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Cubic Interpolation</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Bard Line Search</td>
</tr>
<tr>
<td>2.4.4</td>
<td>The BHHH Line Search</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Efficiency and Termination</td>
</tr>
<tr>
<td>Chapter 2.5</td>
<td>Choice of Stopping Criterion</td>
</tr>
<tr>
<td>2.5.1</td>
<td>The Gradient Stopping Criterion</td>
</tr>
<tr>
<td>2.5.2</td>
<td>The Weighted-Gradient Stopping Criterion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>Characteristics of Econometric Estimation Problems</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>The Model</td>
<td>66</td>
</tr>
<tr>
<td>3.1</td>
<td>The BHHH Method of Estimation and Inference by Maximum Likelihood</td>
<td>70</td>
</tr>
<tr>
<td>3.2</td>
<td>The BHHH Algorithm</td>
<td>74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>A Method of Specification, Differentiation and Computation for Sets of General Functions</th>
<th>77</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>A Differentiation Program</td>
<td>77</td>
</tr>
<tr>
<td>4.1</td>
<td>Organisation of the Differentiation Program</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>General Considerations and Function Specifications</td>
<td>78</td>
</tr>
<tr>
<td>4.3</td>
<td>Sets of Conventions for Defining A Function</td>
<td>82</td>
</tr>
<tr>
<td>4.4</td>
<td>Differentiation</td>
<td>95</td>
</tr>
<tr>
<td>4.5</td>
<td>Simplification</td>
<td>98</td>
</tr>
<tr>
<td>4.6</td>
<td>Numerical Evaluation of Function and Derivative Values</td>
<td>98</td>
</tr>
<tr>
<td>4.7</td>
<td>Example</td>
<td>99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>A Computer Program For The Estimation of General Non-Linear Econometric Models</th>
<th>103</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>An Estimation Program</td>
<td>103</td>
</tr>
<tr>
<td>5.1</td>
<td>Organisation of the Estimation Program</td>
<td>104</td>
</tr>
<tr>
<td>5.2</td>
<td>Functional Definitions</td>
<td>106</td>
</tr>
<tr>
<td>5.3</td>
<td>Estimation Methods</td>
<td>107</td>
</tr>
<tr>
<td>5.4</td>
<td>Program Composition</td>
<td>108</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6</td>
<td>Non-Linear Econometric Modelling on a Parallel Processor</td>
<td>112</td>
</tr>
<tr>
<td>6.1</td>
<td>Potential Role of the Distributed Array Processor for Research in Economics</td>
<td>112</td>
</tr>
<tr>
<td>6.2</td>
<td>The Distributed Array Processor (DAP)</td>
<td>115</td>
</tr>
<tr>
<td>6.3</td>
<td>Architecture</td>
<td>115</td>
</tr>
<tr>
<td>6.4</td>
<td>Programming the DAP</td>
<td>116</td>
</tr>
<tr>
<td>6.5</td>
<td>Examples</td>
<td>118</td>
</tr>
<tr>
<td>6.6</td>
<td>Estimation Procedure and Implementation (Timings and Results in Section 7.7)</td>
<td>121</td>
</tr>
<tr>
<td>7</td>
<td>A Set of Non-Linear Models</td>
<td>125</td>
</tr>
<tr>
<td>7.0</td>
<td>Non-Linear Models Simulations</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Modelling Considerations</td>
<td>126</td>
</tr>
<tr>
<td>7.2</td>
<td>Model Specification</td>
<td>127</td>
</tr>
<tr>
<td>7.3</td>
<td>Data Generation Process</td>
<td>130</td>
</tr>
<tr>
<td>7.4</td>
<td>Sets of Models</td>
<td>132</td>
</tr>
<tr>
<td>7.5</td>
<td>Alternative Values of Parameters</td>
<td>135</td>
</tr>
<tr>
<td>7.6</td>
<td>Estimation Results and Computer Timings</td>
<td>137</td>
</tr>
<tr>
<td>7.7</td>
<td>Results and Timings for Models (iii) on the Distributed Array Processor</td>
<td>163</td>
</tr>
<tr>
<td>8</td>
<td>An Aggregate Demand Model for the United Kingdom, 1957-1967</td>
<td>167</td>
</tr>
<tr>
<td>8.0</td>
<td>A Demand Model</td>
<td>167</td>
</tr>
<tr>
<td>8.1</td>
<td>The Model (Linear Form)</td>
<td>167</td>
</tr>
<tr>
<td>8.2</td>
<td>Transformation of Model (Non-Linear Form)</td>
<td>169</td>
</tr>
<tr>
<td>8.3</td>
<td>Treatment of Coefficients for Equation $u_1$ and Constants for all equations</td>
<td>171</td>
</tr>
<tr>
<td>8.4</td>
<td>Deseasonised Data Series</td>
<td>174</td>
</tr>
<tr>
<td>8.5</td>
<td>NLMLE Specification of Model</td>
<td>176</td>
</tr>
<tr>
<td>8.6</td>
<td>Results</td>
<td>177</td>
</tr>
<tr>
<td>Chapter/Appendix</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>General Conclusion</td>
<td>182</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Cholesky Factorisation</td>
<td>186</td>
</tr>
<tr>
<td>Appendix B</td>
<td>NLMLE Listing</td>
<td>193</td>
</tr>
<tr>
<td>Appendix C</td>
<td>A User's Guide to NLMLE</td>
<td>244</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Program Output - A Typical Run From Model (iii)</td>
<td>246</td>
</tr>
<tr>
<td>Appendix E</td>
<td>A Parallel Inversion Routine</td>
<td>252</td>
</tr>
<tr>
<td>Appendix F</td>
<td>Sets of Non-Linear Models</td>
<td>256</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>259</td>
</tr>
</tbody>
</table>
CHAPTER 1

SURVEY AND LITERATURE

1. Introduction

This research is concerned with the estimation of general non-linear simultaneous equations econometric models by the method of maximum likelihood.

A computer program called Non-Linear Maximum Likelihood Estimation (NLMLE) is specifically developed for this purpose. This program allows the estimation of a quite general non-linear model by the method of Berndt-Hall-Hall-Hausman (BHHH, 1974) or alternatively by the use of the Gill-Murray-Pitfield (GMP, 1972) algorithm (with or without derivatives) from the NAG library. In the course of program development, a differentiation program is developed which will differentiate a set of functions defined by general FORTRAN specifications to any order (Sargan and Chong (1980)). After developing the main estimation program, where much effort has been concentrated on finding an efficient line search algorithm by the use of quadratic interpolation (Powell, 1964), comparison has been made of the relative efficiency of the two methods (BHHH and GMP) by applying these to simulated data from a representative set of models. A more realistic model (Hendry, 1974) has also been studied. At this stage it appeared clear that the BHHH method provides generally a better optimisation algorithm when the number of parameters in the model is greater than 8 and the sample size is greater than 50.

The program is written in FORTRAN IV and the serial version has been implemented on CDC 7600 and ICL 2980 computer systems. For DAP
(Distributed Array Processor) application using parallel processing, a few subroutines from the existing program have been reorganised and reprogrammed in DAPFORTRAN. Due to the storage restrictions of the DAP (2 Mbyte), the program is restricted to the estimation models with no more than 5 equations and 30 parameters but up to 4096 observations. It became clear that the use of DAP will be advantageous when the sample size is large, ideally close to 4096 observations. To increase the degree of parallelism and extend the program size will require further research which is beyond the scope of the present studies.

As the program (the serial version) is written in FORTRAN IV, it is fairly portable; one would expect without much difficulty to implement it on other computer systems.

In the following sections, we will briefly review the literature on maximum likelihood estimators, on numerical methods applied to non-linear econometric models and, lastly, formula manipulation and symbolic differentiation on a computer.

1.1 Maximum Likelihood Estimator for Non-linear Econometric Systems

The usual method of formulating a model which is generally non-linear in both the variables and the parameters in a form suitable for maximum likelihood estimation is that suggested initially by Eisenpress and Greenstadt (1966). For a later discussion, see Chow (1973). The full-information maximum likelihood estimates of the k-vector of unknown parameters $\theta$ can be obtained as the parameter $\hat{\theta}$ that maximises the concentrated log-likelihood function of simultaneous equations systems that are non-linear in the parameters and/or variables.
Each equation of the simultaneous equation system is specified by expressing the error on the equation as a general function of the variables and the parameters, and it is then assumed that the errors are jointly normally distributed.

The Eisenpress and Greenstadt procedure is to estimate each equation in a non-linear system by ordinary least squares, and then to use these results as the initial approximation to a full-information solution. However, the problem of identification in non-linear systems is not treated and provision has to be made ultimately to avoid working with under-identified equations and systems. In Eisenpress and Greenstadt's work, they define a given equation in the form

\[ y_{it} = g_i(y_{lt}, \ldots, y_{i-1,t}, y_{i+1,t}, \ldots, y_{N,t}, z_{lt}, \ldots, z_{Mt}, \theta_1, \ldots, \theta_K) + u_{it} \]  

(1.1)

where one endogenous variable \( y_i \) is an arbitrary function \( g_i \) of the other endogenous variables, the predetermined variables \( z_m \) and the parameters \( \theta_k \) subject to a random disturbance \( u_i \). Then least squares or maximum likelihood estimation (if the \( u_i \) are assumed normal) is applied to the \( T \) observations on the \( y \)'s and \( z \)'s, to estimate the \( \theta \)'s, under the assumption that

\[
E(u_{it}) = 0, \quad t = 1, \ldots, T
\]

\[
\text{Cov}(u_{it}, u_{it'}) = \begin{cases} 
0 & t \neq t' \\
\sigma_i^2 & t = t'
\end{cases}
\]

The estimates of the \( \theta \)'s will, in general, be inconsistent, but this
calculation is able to provide a first guess for use in the techniques that follow.

To obtain the least-squares estimates of the \( \theta \)'s, the function

\[
L' = \sum\limits_t (y_{it} - g_i)^2
\]

is minimised with respect to the \( \theta \)'s, using the modified Newton method which requires the first and second derivatives of \( L' \) with respect to \( \theta \).

On more general assumptions, the full-information maximum likelihood estimates are obtained by defining the set of equations as

\[
f_{it}(y_t', z_t', \theta) \equiv f_{it}(y_{it}', ..., y_{Nt}', z_{it}', ..., z_{Mt}); \\
\theta_1', ..., \theta_K = u_{it}
\]

\[i = 1, ..., N; t = 1, ..., T\]  

where \( u_{it} \) are the random disturbances of these relations. The \( f \)'s are assumed to have derivatives up to third order.

The concentrated log-likelihood function of (1.2) is

\[
L^*(\theta) = \text{const} - \frac{T}{2} \log \det(S) + \sum\limits_t \log |\det(J_t)| \tag{1.3}
\]

where

\( f_t \) is the \( N \times 1 \) vector with elements \( f_{it} \),

\[
S \equiv S(\theta) = \left( \frac{1}{T} \sum\limits_t f_{it}(\theta) f_{jt}(\theta) \right)
\]
and $J_t$ is the $N \times N$ Jacobian matrix,

$$J_t = J_t(\theta) = \left[ \frac{\partial f_t(y_t, z_t; \theta)}{\partial y_t} \right].$$

To maximise $L^*(\theta)$, the gradient method or modified Newton method is used. These both make use of the first and second derivatives of $L^*(\theta)$ with respect to the $\theta$'s. As in many iterative procedures, the first approximation and the conditioning of various matrices (e.g. the Hessian) are important in determining the speed of convergence. Usually any arbitrary first guess for $\theta$ will be accepted by the algorithms, but a good first guess may speed up convergence substantially.

Chow (1973) generalises the modified Newton method for the computation of full-information maximum likelihood estimates of parameters of a system of linear structural equations to the case of a system of non-linear structural equations. The main differences of Chow's approach to that of Eisenpress and Greenstadt are:

1. Eisenpress' and Greenstadt's basic formulation is more general, assuming that all parameters in the system may appear in every equation (see equation 1.2), whereas Chow assumes as the basic set-up that there is a distinct set of parameters belonging to each equation. His basic formulation is as follows:

Let the $g^{th}$ function

$$f_{gt}(y_{1t}, \ldots, y_{Nt}; \beta_g)$$

of the $G$ dependent variables and $K = N - G$ predetermined variables $y_{1t}, \ldots, y_{Nt}$ at period $t$ and of the row vector $\beta_g$ of $N$ unknown parameters to be equated to a residual $u_{gt}$ ($g = 1, \ldots, G$).
Assume $u_{gt}$ is normally distributed with means zero and covariances $E(u_{gt}u_{hs}) = \delta_{ts} \sigma_{gh}$. For $T$ observations, the concentrated log-likelihood function $L$ is proportional to

$$-\frac{T}{2} \log |S| + \sum_{t=1}^{T} \log |B_t^o|,$$

where

$$S \equiv (S_{gh}) = \left( \frac{1}{T} \sum_t u_{gt} u_{ht} \right)$$

and

$$B_t^o \equiv (b_{gh,t}^o) = \left( \frac{\partial u_{gt}}{\partial y_{ht}} \right)$$

(2) Partly because of his formulation, Chow obtains simpler and more explicit expressions for the derivatives of the likelihood functions.

(3) Again, partly because of his formulation, the problem of linear restriction on the parameters in the same equation or in different equations can be conveniently dealt with by Chow.

(4) Chow's paper features the treatments of identities in the system and of residuals which may follow an autoregressive scheme.
An extensive derivation of the estimation equations for non-linear systems is given under the assumptions that each structural equation contains a distinct set of parameters, that the parameters are not subject to any linear restrictions, and that the (additive) residuals are serially uncorrelated. It also provides the treatment of the special case when some equations are linear, and contrasts this case with the non-linear case.

Berndt-Hall-Hall-Hausman (1974) propose an ingenious idea of using the statistical relation that the covariance matrix of the maximum likelihood estimator is equal to the inverse of the covariance matrix of the gradient of the log-likelihood function, which in its turn is equal to minus the (inverse) Hessian matrix of the log-likelihood function. Their algorithm requires much less computation than previous algorithms and unlike previous algorithms is less likely to fail from instability of the iterative procedure. We will concentrate our studies on the BHHH method and will derive the estimation equations in Chapter 3.

Amemiya (1977) proves the consistency and the asymptotic normality of the maximum likelihood estimator in the general non-linear simultaneous equation model. The proof depends on the assumption that the errors are normally distributed which is not necessary for simultaneous equation models which are linear in the variables. It is also proved that the maximum likelihood estimator is asymptotically more efficient than the non-linear three stage least squares estimator providing
the specification is correct. However, the latter has the advantage of being consistent even when the normality assumption is removed.

Hatanaka (1978) proposes a full-information estimation method for macro-economic models which are generally non-linear in variables. The method is shown to be asymptotically efficient and feasible in terms of computer computations, and hopefully it may be applied to the undersized sample case. The idea of the BHHH method may be applied to replace the Hessian of the two-step scoring estimator which is asymptotically equivalent to the maximum likelihood estimator.

1.2 Numerical Optimisation Applied to Non-linear Econometric Models

Bard (1970) investigates several of the best known gradient methods, and the performance of these methods is compared in the solution of some least squares maximum likelihood, and Bayesian estimation problems. He concludes that modifications of the Gauss method (including Marquardt's) performed best, followed by variable metric rank one and Davidon-Fletcher-Powell methods, in that order. There appeared to be no need to locate the optimum precisely in the one-dimensional searches, but the matrix inversion method used with the Gauss algorithm must guarantee a positive definite inverse.

Sargan and Sylwestrowicz (1976) develop a specialised numerical optimisation computer program for the estimation of simultaneous equation econometric models, in the hope that it would be more efficient than the alternative computing methods. The methods are compared by estimating a small macro-economic model of the British economy as specified by Hendry (1974) with five different sets of assumptions
as to the stochastic processes generating the errors of the equations. All the assumptions involve separate single equation autoregressive equations, explaining the current error on each equation in terms of the previous error on the same equation. This was chosen as giving experience of the use of the optimisation program on a representative problem, from the point of view of time and complication in computing the required function, and with a number of parameters to be estimated which is reasonably large as most econometric models involve a large number of parameters.

To maximise the likelihood function appropriate to a simultaneous linear equations model, a subroutine is provided to calculate the likelihood function and use this with a general optimisation routine not requiring analytical derivatives, such as the Powell conjugate directions method. It is also possible to use the Davidon-Fletcher-Powell type of quasi-Newton method, providing it with a subroutine to calculate the likelihood function and its first derivatives; and finally a special generalised Gauss-Newton program was written for use with the non-linear simultaneous equation likelihood functions.

The extensive results presented in the paper indicate that the Gill-Murray-Pitfield optimisation routine, making use of analytic first derivatives, is the most efficient method with most of the models.

Belsley (1980) examines some important elements in calculating the non-linear full-information maximum likelihood estimator which produce substantial reductions in computational cost. The choice of optimiser, method of Hessian approximation, choice of convergence criterion and exploitation of sparsity of matrices are all investigated.
It is concluded that the Newton-Raphson algorithm employing an analytical computed Hessian is computationally much more efficient than Davidon-Fletcher-Powell. The weighted gradient stopping criterion is recommended, that is,

\[-g'H^{-1}g \leq \epsilon\]

where

- \(g\) is the gradient of the likelihood of function \(L(\theta)\),
- \(H\) is the Hessian matrix of \(L(\theta)\),
- \(\epsilon\) is the tolerance level of accuracy.

Exploiting the sparsity of \(J_t\) (the Jacobian) and the efficient calculation of the components that make up the analytic Hessian are also investigated for large models.

1.3 Specific Application Program

In order to implement a general optimisation procedure of the BHHH kind on the computer, we need to be able to specify the general functions

\[f_{it}(y_t, z_t, \theta) = u_{it}\]

in a suitable computer code form so that we can differentiate these functions twice analytically w.r.t.\(\theta\), that is, \(\partial^2 f_{i}(y_{t},z_{t},\theta)/\partial\theta\)

and \(\partial^2 f_{i}(y_{t},z_{t},\theta)/\partial\theta\partial\theta'\).

It is of considerable importance that the resulting form of function specification should lead to efficient evaluation of both
functions and their derivatives. It is not necessary that the
derivatives should be presented in the most logical form from the
point of view of mathematical interpretation, but unnecessary
repetitive computing should be avoided in the course of their evaluation.

Obviously it would be a bad strategy to differentiate the set
of functions explicitly and then input the derivatives to the computer
for evaluation for the following reasons:

1. The functions can be very complex.

2. There exists a high probability of differentiation and
   programming errors.

3. For each model, we need to differentiate and program the
   functions and derivatives. The volume of arithmetic involved
   could be very intense, for example, it takes several function
   evaluations in the line search just to reduce the function value.

Therefore an automatic differentiation program is necessary to compute
the set of econometric functions and their derivatives. Most programs
for automatic differentiation are embedded in general computer
packages for symbol manipulation, for example, systems such as MACSYMA
and REDUCE (see references). These are used for performing symbolic as
well as numerical mathematical manipulations. With such computer
packages, it is possible to differentiate, integrate, take limits,
solve systems of linear or polynomial equations, factor polynomials,
expand functions in Taylor series, solve differential equations and
perform many other operations.

Since it is intended that the differentiations of the set of
econometric functions is to be carried out on the computer in order that
complicated functions are differentiated accurately with results which are easy to understand and compute numerically, and to avoid unnecessary repetitive computation in the process, systems as comprehensive as MACSYMA and REDUCE would not be appropriate for this application. Moreover, it is necessary to have a compact set of derivatives of the concentrated log-likelihood function for our estimation procedures, it would be difficult to extract the necessary subroutines from these packages in order to perform the same task.

However, we are very much influenced by the function specifications and data structures of these systems, and have decided to develop our own differentiation program (see Chapter 4) with the specific application of differentiating and evaluating non-linear econometric functions.

This program is written in FORTRAN IV and is machine portable. It permits natural mathematical notation in FORTRAN definitions and can differentiate a set of functions with respect to a given set of variables up to any order. Only numerical values of the functions and derivatives are printed instead of the symbolic form of the expression in mathematical notation. It is hoped that it could be easily implemented with any non-linear econometric estimation programs written in FORTRAN IV. The program was specifically written so that it could be developed on the CDC 7600 computer, and then transferred with minimal rewriting onto the ICL 2980 system with the Distributed Array Processor (DAP).
CHAPTER 2

NUMERICAL OPTIMISATION TECHNIQUES FOR

NON-LINEAR ECONOMETRIC MODELS

2. Numerical Optimisation

Many issues arise in the practical task of optimising a non-linear function. A general description of these may be found in Goldfeld and Quandt (1972), Jacoby et al. (1972), Murray (1972), and Bard (1970). In this chapter, we examine two issues that are among the more important and interesting in the optimisation of the concentrated log-likelihood function, relevant to the non-linear full-information maximum likelihood (NLFIML) estimator, namely, the choice of the optimisation method and the method of approximating the Hessian.

2.1 Optimisation Methods

Two different optimisation algorithms are compared in separate runs for the various models. The gradient-type method (e.g. BHHH) and the quasi-Newton method (e.g. GMP). Each of these algorithms is a Newton-like procedure in that its step in the parameter space at each iteration could be a Newton-like step,

\[
\Delta \theta^{(k)} = \theta^{(k+1)} - \theta^{(k)} = -\lambda^{(k)} [H^{(k)}(\theta)]^{-1} \cdot g^{(k)}(\theta)
\]

where

\[
\lambda^{(k)} = \text{a scalar, the step-length along the search direction}
\]
\( H^{(k)}(\theta) \) a \( K \times K \) matrix; the Hessian
\[
\frac{\partial^2 L^*(\theta)}{\partial \theta' \partial \theta}|_{\theta=\hat{\theta}(k)}
\]
or approximate Hessian at the \( k^{th} \) iteration.

\( g^{(k)}(\theta) \) = a \( K \)-vector; the gradient of \( L^*(\theta) \) evaluated at the \( k^{th} \) iteration,
\[
\frac{\partial L^*(\theta)}{\partial \theta}|_{\theta=\hat{\theta}(k)}
\]

There are, of course, many ingenious non-Newton optimisation techniques that require no first or second derivatives. Such algorithms have been developed to handle general optimisation problems. However, when a specific functional form for the objective function is known, as is the case here in the concentrated log-likelihood function, it is generally concluded that it is beneficial to exploit this information. Thus, we examine closely methods that use at least gradient information. We also examine methods that use numerical approximation to the gradient to see how relatively inefficient they may be.

The Hessian \( H \) of (2.1) plays two important roles in maximum-likelihood estimation, and the means for its calculation or approximation can affect both. First, the Hessian is used numerically. Either the Hessian or its approximation is used at each iteration of a Newton-like optimisation algorithm to determine the next step in (2.1). Second, the Hessian is used statistically. At the maximum-likelihood solution, the negative of the inverse Hessian provides an estimate of the asymptotic variance-covariance matrix of \( \theta \).
2.2 Minimisation with Derivatives

We use the notation \( \phi(\theta) = -L'(\theta) \) defined as the concentrated log-likelihood function whose least value is to be calculated, where \( \theta \) is a vector whose components are the variables which are to be adjusted automatically by a minimisation algorithm. Let \( n \) be the number of components of \( \theta \). To define \( \phi(\theta) \) for a minimisation algorithm, the user must provide a subroutine that calculates \( \phi(\theta) \) for any \( \theta \). He must also provide a starting vector \( \theta^{(0)} \), say, and perhaps some other information, for example, step-lengths and the accuracy required. Then most algorithms automatically construct a sequence of points \( \theta^{(k)} \) \((k = 1, 2, \ldots)\), which should converge to the required vector of variables.

The algorithms that we consider are iterative, and we let \( \theta^{(0)} \) be the starting point of the \( k \)th iteration. They include safeguards which force the inequality

\[
\phi(\theta^{(k+1)}) \leq \phi(\theta^{(k)})
\]

(2.2)

to be satisfied.

We assume that the function \( \phi(\theta) \) has continuous first and second derivatives. In this case it is almost always best to select the minimisation algorithm from the general methods that are designed to work particularly well when \( \phi(\theta) \) is a quadratic function. The reason is that if the steps \( \{\theta^{(k+1)} - \theta^{(k)}\} \) become small, then usually the local behaviour of \( \phi(\theta) \) is similar to that of a quadratic function, so the algorithm should work well, and if the steps \( \{\theta^{(k+1)} - \theta^{(k)}\} \) are large then usually good progress is being made anyway.
Some very useful algorithms of this type are described in the books by Brent (1973) and Kowalik and Osborne (1968).

2.2.1 The Newton-Raphson Method

When \( G^{(k)}(\theta) = [H^{(k)}(\theta)]^{-1} \), where \( H^{(k)} \) is the exact Hessian matrix at \( \theta^{(k)} \), we have the Newton-Raphson method which is described in the following algorithm.

(a) Algorithm 2.1

It is assumed that an initial estimate \( \theta^{(0)} \) of the optimum point \( \theta^* \) is known.

Step 0: Set \( k = 0 \)

Step 1: Compute \( q^{(k)}(\theta) \), and \( H^{(k)}(\theta) \) from

\[
q^{(k)}(\theta) = \left. \frac{\partial^2 \phi(\theta)}{\partial \theta \partial \theta'} \right|_{\theta = \theta^{(k)}}
\]

\[
H^{(k)}(\theta) = \left. \frac{\partial^2 \phi(\theta)}{\partial \theta \partial \theta'} \right|_{\theta = \theta^{(k)}}
\]

Step 2: Compute \( \xi^{(k)} \) by solving the system of linear equations

\[
H^{(k)}(\theta) \xi^{(k)} = -q^{(k)}(\theta)
\]

Step 3: Compute \( \theta^{(k+1)} \) from

\[
\theta^{(k+1)} = \theta^{(k)} + \xi^{(k)}
\]

Step 4: If convergence has been attained, stop, else set \( k = k + 1 \) and go to Step 1.
(b) Safeguarding the Method of Newton-Raphson

The sequence \{\theta^{(k)}\} generated from Algorithm 2.1 will converge to a critical point \theta^* of \phi which is a strong local minimiser of \phi if \theta^{(o)} is sufficiently close to \theta^*, and the order of convergence is quadratic. Unfortunately, however, a sufficiently good initial estimate \theta^{(o)} of \theta^* is often not available. In order to make Newton's method more satisfactory for practical use, devices must be incorporated into Algorithm 2.1 which reduce the probability of divergence. We shall consider the principal causes of failure in Newton's method.

We have the descent direction \( p^{(k)} \) defined by

\[
p^{(k)} = - [H^{(k)}(\theta)]^{-1} g^{(k)}(\theta) \tag{2.3}
\]

where \( H^{(k)}(\theta) \) and \( g^{(k)}(\theta) \) are the Hessian matrix and gradient vector respectively for the function \( \phi \) at \( \theta^{(k)} \),

If \( [H^{(k)}(\theta)]^{-1} \) exists, then \( G^{(k)}(\theta) = [H^{(k)}(\theta)]^{-1} \) is positive definite if and only if \( H^{(k)}(\theta) \) is positive definite. If \( G^{(k)}(\theta) \) is positive definite then

\[
g^{(k)'}(\theta)p^{(k)} = -g^{(k)'}(\theta)[G^{(k)}(\theta)]g^{(k)}(\theta) < 0
\]

so that \( p^{(k)} \) is downhill for \( \phi \) at \( \theta^{(k)} \). If, however, \( \phi \) is not well approximated by a quadratic function, in the neighbourhood of \( \theta^{(k)} \), then the step \( p^{(k)} \) to \( \theta^{(k+1)} \) may be too large, in that

\[
\phi(\theta^{(k+1)}) > \phi(\theta^{(k)})
\]

and the minimisation has not progressed smoothly.
If \( [H^{(k)}(\theta)]^{-1} \) exists but is not positive definite it may be that although the Newton step \( p^{(k)} \) is well defined, we have \( g^{(k)'}(\theta)p^{(k)} = 0 \). In this case although \( p^{(k)} \) is not actually uphill for \( \phi \) at \( \theta^{(k)} \), we cannot be sure that \( \phi(\theta^{(k+1)}) < \phi(\theta^{(k)}) \).

If \( g^{(k)'}(\theta)p^{(k)} > 0 \), then \( p^{(k)} \) is uphill for \( \phi \) at \( \theta^{(k)} \), and no step in the direction \( p^{(k)} \) can help the minimisation.

If \( [H^{(k)}(\theta)]^{-1} \) does not exist then \( p^{(k)} \) is not even defined, so that if further progress is to be made we need an alternative method for constructing \( p^{(k)} \) when \( G^{(k)}(\theta) \) is singular.

It is clear that the Newton sequence \{\( \theta^{(k)} \)\}, if it converges at all, it converges to a critical point of \( \phi \). But the sequence \{\( \theta^{(k)} \)\} may converge to a saddle point or to a maximiser of \( \phi \) instead of a minimiser, although this does not usually happen if \( \theta^{(0)} \) is sufficiently close to a minimiser of \( \phi \).

We may conclude from the above discussion that Newton's method is subject to the following causes of failure during the \((k + 1)\)th iteration.

1. \( G^{(k)}(\theta) \) exists and is positive definite but \( p^{(k)} \) is too large and \( \phi(\theta^{(k+1)}) > \phi(\theta^{(k)}) \).

2. The direction \( p^{(k)} \) is orthogonal to \( g^{(k)'}(\theta) \).

3. \( G^{(k)}(\theta) \) exists but is not positive definite.

4. \( G^{(k)}(\theta) \) does not exist.

To reduce the probability of failure due to the causes above, we consider the following strategies.
1. If \( G^{(k)}(\theta) \) is positive definite then \( p^{(k)} \) is downhill for \( \phi \) at \( \hat{\theta}^{(k)} \) and there exists \( \lambda > 0 \) sufficiently small such that

\[
\phi(\hat{\theta}^{(k)} + \lambda p^{(k)}) < \phi(\hat{\theta}^{(k)}). \tag{2.4}
\]

If the length of \( p^{(k)} \) is so small that \( \phi \) is well approximated by a quadratic function throughout the range of \( \hat{\theta}^{(k)} \) to \( \hat{\theta}^{(k+1)} \) then it is likely that

\[
\phi(\hat{\theta}^{(k)} + p^{(k)}) < \phi(\hat{\theta}^{(k)}). \tag{2.4}
\]

If, however,

\[
\phi(\hat{\theta}^{(k)} + p^{(k)}) > \phi(\hat{\theta}^{(k)})
\]

then a value of \( \lambda^{(k)} \in (0,1) \) must be found such that

\[
\phi(\hat{\theta}^{(k)} + \lambda^{(k)} p^{(k)}) < \phi(\hat{\theta}^{(k)}). \tag{2.5}
\]

A simple strategy for computing a value of \( \lambda^{(k)} \) which satisfies (2.5) is given in the following algorithm.

Notice that if \( \lambda^{(k)} \) is becoming very small and (2.5) is not satisfied, the algorithm then terminates.
Algorithm 2.2

1. Set $\lambda = 1$.

2. Compute $\phi$ from $\phi = \phi(\theta^{(k)} + \lambda p^{(k)})$.

3. If $\phi < \phi(\theta^{(k)})$ go to 6.

4. If $\lambda < 10^{-5}$ go to 7.

5. Set $\lambda = \frac{\lambda}{2}$ and go to 2.

6. Set $\lambda^{(k)} = \lambda$, and $\phi(\theta^{(k+1)}) = \phi$.

Return to Newton-Raphson routine.

7. Stop.

An alternative, and perhaps more natural strategy is to compute $\lambda^{(k)}$ by performing a line search along $p^{(k)}$ if $p^{(k)}$ is known to be downhill for $\phi$ at $\theta^{(k)}$. The introduction of the parameter $\lambda^{(k)}$ computed according to Algorithm 2.2 or by performing a line search (see section 2.4) safeguards Newton's method against cause 1 of failure.

2. Consider now if $p^{(k)}$ is orthogonal to $q^{(k)}(\theta)$ and no progress can be made by performing a line search along $p^{(k)}$. This illustrates cause 2 of failure. In practice, owing to rounding error, effective orthogonality of $p^{(k)}$ and $q^{(k)}(\theta)$ is decided by determining whether

$$|g^{(k)}(\theta)p^{(k)}| < \epsilon \|q^{(k)}(\theta)\|_2\|p^{(k)}\|_2,$$

where $\epsilon > 0$ is a given small number relative to unity. One strategy for safeguarding Newton's method against cause 2 of failure is to replace $p^{(k)}$ with $-q^{(k)}(\theta)$ if

$$|g^{(k)}(\theta)p^{(k)}| < \epsilon \|q^{(k)}(\theta)\|_2\|p^{(k)}\|_2$$

and perform a line search
along the new $p^{(k)}$. Thus if $q^{(k)}(\theta)$ and $[G^{(k)}(\theta)] q^{(k)}(\theta)$ are effectively orthogonal, we take a steepest descent step.

3. If $G^{(k)}(\theta)$ is not positive definite, then $p^{(k)}$ is not necessarily downhill and the existence of $\lambda^{(k)} > 0$ which satisfies (2.5) cannot be guaranteed.

If $q^{(k)} p^{(k)} > 0$, then $p^{(k)}$ is uphill at $\theta^{(k)}$ and $-p^{(k)}$ therefore downhill. In this case we replace $p^{(k)}$ defined by (2.3) with $-p^{(k)}$ and the existence of $\lambda > 0$ such that $\phi(\lambda) < \phi(0)$ is guaranteed. It is a better strategy to replace $p^{(k)}$ with $-p^{(k)}$ if $q^{(k)} p^{(k)} > 0$ than rejecting $p^{(k)}$ altogether and taking a steepest descent step. This strategy therefore safeguards Newton's method against cause 3 of failure.

4. If $H^{(k)}(\theta)$ is singular, then $p^{(k)}$ given by (2.4) is not defined. A simple strategy for overcoming this difficulty is to replace $p^{(k)}$ with $-q^{(k)}(\theta)$ and take a steepest descent step,
thus safeguarding Newton's method against cause 4 of failure.

From the preceding discussion, we obtain the following algorithm.

**Algorithm 2.3**

It is assumed that an estimate \( \theta^{(0)} \) of a minimiser \( \theta^* \) of \( \phi \) and \( \epsilon > 0 \) are given.

**Step 0:** Set \( k = 0 \).

**Step 1:** Compute \( q^{(k)}(\theta) \), and \( H^{(k)}(\theta) \) from

\[
q^{(k)}(\theta) = \left. \frac{\partial \phi(\theta)}{\partial \theta} \right|_{\theta = \theta^{(k)}}
\]

\[
H^{(k)}(\theta) = \left. \frac{\partial^2 \phi(\theta)}{\partial \theta \partial \theta^t} \right|_{\theta = \theta^{(k)}}
\]

**Step 2:** If \( H^{(k)}(\theta) \) is singular, go to Step 10.

**Step 3:** Compute \( p^{(k)} \) from

\[
p^{(k)} = - [H^{(k)}(\theta)]^{-1} q^{(k)}(\theta)
\]

**Step 4:** If \(|q^{(k)}(\theta) p^{(k)}| \leq \epsilon \|q^{(k)}(\theta)\|_2 \|p^{(k)}\|_2\) \), then go to Step 10.

**Step 5:** If \(|q^{(k)}(\theta) p^{(k)}| > \epsilon \|q^{(k)}(\theta)\|_2 \|p^{(k)}\|_2\) \), then go to Step 12.
Step 6: Compute $\lambda^{(k)}$ by repeated bisection (Algorithm 2.2) or by performing a line search (section 2.4).

Step 7: Set $\theta^{(k+1)} = \theta^{(k)} + \lambda^{(k)} p^{(k)}$.

Step 8: If convergence is attained, go to Step 13.

Step 9: Set $k = k + 1$, go to Step 1.

Step 10: Set $p^{(k)} = - g^{(k)}(\theta)$.

Step 11: Perform a line search along $p^{(k)}$ to obtain $\lambda^{(k)}$, and go to Step 7.

Step 12: Set $p^{(k)} = - p^{(k)}$ and go to Step 6.

Step 13: Set $\theta^* = \theta^{(k+1)}$.

Step 14: Stop.

(c) Objections to Newton-Raphson Method

There are a number of objections to Newton's method as a computational procedure, the most important of which are as follows.

1. In order to evaluate $H^{(k)}(\theta)$, we must compute $n(n+1)/2$ function values $\frac{\partial^2 \phi(\theta)}{\partial \theta \partial \theta}$ $\bigg|_{\theta = \theta^{(k)}}$. This means that $n(n+1)/2$ partial derivatives must be calculated analytically and programmed, with consequent probability of analytical and programming error. Also storage space must be allocated in
the computer for \( n(n+1)/2 \) numbers, and for a subprogram which is required in order to compute them. And the time required for these calculations may be very large (especially when estimating econometric models).

2. At each iteration, \( [H^{(k)}(\theta)]^{-1} \) must be computed. This requires \( O(n^3) \) arithmetic operations. Also a subprogram for inverting matrices is required.

3. It may be that \( H^{(k)}(\theta) \) is singular for some \( k \) so that the method breaks down.

4. The method is not guaranteed to converge unless \( \theta^{(0)} \) is sufficiently close to \( \theta^* \).

Note: We have in fact safeguarded objections 3 and 4.

These objections frequently make Newton's method unsuitable for numerical calculation, but if a sufficiently good initial estimate \( \theta^{(0)} \) of \( \theta^* \) is known, and if the first and second partial derivatives of \( \phi \) are easy to program and to compute, then the method is among the best which is available. On the other hand, if \( \theta^{(0)} \) is far from \( \theta^* \), then there is no reason to believe that in the first few iterations the method has any advantages over a comparatively crude algorithm which takes much less time than the Newton-Raphson.

Objection 1 may to some extent be overcome by using a numerical differentiation formula in order to estimate the Hessian matrix, but \( n \) additional evaluations of \( g^{(k)}(\theta) \) are required for the computation of \( H^{(k)}(\theta) \). The use of a numerical differentiation formula may be inefficient if \( n \) is large or if the computational labour required for the evaluation of \( g^{(k)}(\theta) \) is large and also \( g^{(k)}(\theta) \) could be inaccurate.

The preceding considerations provide a motive for constructing methods for minimising \( \phi \) in which it is not necessary to compute
or invert \( H^{(k)}(\theta) \), but which have superlinear convergence, since they ultimately find a value of \( H^{(k)}(\theta) \) which approximates the exact Hessian when \( \theta \) is close to \( \theta^* \).

### 2.2.2 Variable-Metric Methods

In these methods, the matrix \( [H^{(k)}(\theta)]^{-1} \) is replaced by a positive definite symmetric matrix \( G^{(k)}(\theta) \) calculated from currently available quantities such as \( \theta^{(k)} \), \( \theta^{(k-1)} \), and \( \theta^{(k-1)}(\theta) \), to compute \( \theta^{(k+1)} \) from

\[
\theta^{(k+1)} = \theta^{(k)} - \lambda^{(k)} G^{(k)}(\theta) q^{(k)}(\theta)
\]  

(2.6)

If \( G^{(k)}(\theta) \) is positive definite (\( \forall k \geq 0 \)), then \( -G^{(k)}(\theta)q^{(k)}(\theta) \) is downhill for \( \phi \) at \( \theta^{(k)} \), because \( q^{(k)}(\theta)G^{(k)}(\theta)q^{(k)}(\theta) < 0 \) if \( q^{(k)}(\theta) \neq 0 \). Therefore if \( G^{(k)}(\theta) \) is positive definite and \( q^{(k)}(\theta) \neq 0 \), there exists \( \lambda^{(k)} > 0 \) such that

\[
\phi(\theta^{(k)} - \lambda^{(k)} G^{(k)}(\theta) q^{(k)}(\theta)) < \phi(\theta^{(k)})
\]

A method for overcoming to some extent objection 4 to Newton's method therefore consists of generating \( \{\theta^{(k)}\} \) from (2.6).

The use of (2.6) also overcomes objection 3 to Newton's method because even though the Hessian is singular, \( G^{(k+1)}(\theta) \) can be defined so that it not only exists but is positive definite.

The following algorithm is a general variable metric method for minimizing \( \phi \).

---

1 It may be possible to make statements about the rate of convergence that occurs. Let \( \theta^{*} \) to be the local minimum point. Define the error

\[
e^{(k)} = \theta^{(k)} - \theta^{*}.
\]

If \( e^{(k)} + 0 \), that means convergence. For instance, if

\[
||e^{(k+1)}|| = \alpha \|e^{(k)}\| + a,
\]

then the rate of convergence is said to be linear or first order if \( \alpha > 0 \), and Superlinear if \( \alpha = 0 \). Clearly it is desirable to have \( a \) as small as possible. In some cases it is possible to show that

\[
\|e^{(k+1)}\|/\|e^{(k)}\|^2 + a
\]

in which case the rate is said to be quadratic or second order. This is even more rapid since the error decreases as the square of the previous error.
Algorithm 2.4

It is assumed that $\varphi^{(0)}$ and $G^{(0)}$ are given.

Step 0: Set $k = 0$.

Step 1: Compute $\phi^{(k)}(\varphi)$ and $g^{(k)}(\varphi)$ from

$$
\phi^{(k)}(\varphi) = \phi(\varphi^{(k)})
$$

$$
g^{(k)}(\varphi) = g(\varphi^{(k)})
$$

Step 2: Compute $p^{(k)} = -g^{(k)}(\varphi)g^{(k)}(\varphi)$.

Step 3: Compute $\lambda^{(k)}$ such that

$$
\phi(\varphi^{(k)}) + \lambda^{(k)}p^{(k)} = \min_{\lambda} \phi(\varphi^{(k)}) + \lambda p^{(k)}
$$

Step 4: Compute $\varphi^{(k+1)}$ from

$$
\varphi^{(k+1)} = \varphi^{(k)} + \lambda^{(k)}p^{(k)}
$$

Step 5: Compute $q^{(k+1)}(\varphi)$, $s^{(k)}$ and $y^{(k)}$ from

$$
q^{(k+1)}(\varphi) = q(\varphi^{(k+1)})
$$

$$
s^{(k)} = \varphi^{(k+1)} - \varphi^{(k)}
$$

and

$$
y^{(k)} = q^{(k+1)}(\varphi) - q^{(k)}(\varphi)
$$
Step 6: Compute \( G^{(k+1)}(\theta) \) from

\[
G^{(k+1)}(\theta) = G^{(k)}(\theta) + \Pi_1 z^{(k)} z^{(k)}' + \Pi_2 \omega^{(k)} \omega^{(k)}',
\]

where \( z^{(k)} \), \( \omega^{(k)} \) are \( n \times 1 \) vectors and \( \Pi_1 \), \( \Pi_2 \) are scalars. The exact values of these variables, which are functions of \( q^{(k)}(\theta) \), \( p^{(k)} \) and \( q^{(k+1)}(\theta) \), will depend upon the modification rule used.

Therefore Algorithm 2.4 contains a class of methods rather than a single variable-metric method. Many updating formulae of the type in Step 6 have been proposed since Davidon (1959) described the first variable-metric method. In this chapter some of the most successful variable-metric methods will be described.

2.2.3 General Gradient Method

At the beginning of the \( k \) iteration, we possess a current value of \( \theta^{(k)} \) and we seek a new \( \theta^{(k+1)} \) using the formula

\[
\theta^{(k+1)} = \theta^{(k)} - \lambda^{(k)} G^{(k)}(\theta) q^{(k)}(\theta)
\]

where

\( q^{(k)}(\theta) \) is the gradient vector of \( \phi(\theta) \) at \( \theta = \theta^{(k)} \)

\( \lambda^{(k)} \) is a scalar that minimises \( \phi(\theta^{(k)}) - \lambda G^{(k)}(\theta) q^{(k)}(\theta) \),

and \( G^{(k)}(\theta) \) is a positive definite matrix which guarantees that if \( q^{(k)}(\theta) \neq 0 \), then for sufficiently small positive \( \lambda^{(k)} \), we have satisfied the condition (2.2).

\( G^{(k)}(\theta) \) should be some approximation to \( [H^{(k)}(\theta)]^{-1} \), where \( H^{(k)}(\theta) \) is the Hessian matrix of \( \phi \) at \( \theta = \theta^{(k)} \).
2.2.4 The Method of Fletcher and Powell (DFP)

The method of Fletcher and Powell (1963) is an improved version of a method due to Davidon (1959). It is still one of the best methods for unconstrained minimisation in which only the gradient vector of the objective function is required.

The matrix $G^{(k+1)}(\theta)$ is given by the updating formula

$$G^{(k+1)}(\theta) = G^{(k)}(\theta) + \frac{\lambda^{(k)} P^{(k)} P^{(k)'} - G^{(k)}(\theta) y^{(k)} y^{(k)'} G^{(k)}(\theta)}{y^{(k)'} G^{(k)}(\theta) y^{(k)}}$$

(2.8)

We now show that (2.8) does correspond to the general algorithm as defined above.

If Householder's (1964) rank-one modification rule is applied first to

$$G^{(k)}(\theta) = G^{(k)}(\theta) + \frac{\lambda^{(k)} P^{(k)} P^{(k)'}}{P^{(k)'} Y^{(k)}}$$

and then to

$$G^{(k+1)}(\theta) = G^{(k)}(\theta) - \frac{G^{(k)}(\theta) y^{(k)} y^{(k)'} G^{(k)}(\theta)}{y^{(k)'} G^{(k)}(\theta) y^{(k)}}$$

the corresponding recurrence relation for the approximate Hessian matrix is obtained

$$H^{(k+1)}(\theta) = H^{(k)}(\theta) + \left[ \frac{1}{\delta \lambda} - \frac{\eta}{\delta^2} \right] y^{(k)} y^{(k)'} + \frac{1}{\delta} \left( q^{(k)}(\theta) y^{(k)} + y^{(k)} q^{(k)'}(\theta) \right)$$

(2.9)

where

$$\eta = \frac{q^{(k)'}(\theta) P^{(k)}}{y^{(k)'} G^{(k)}(\theta) y^{(k)}}$$
and
\[ \delta = \chi^{(k)} p^{(k)} \]

If an exact line search is made then \( q^{(k+1)'}(\theta)p^{(k)} = 0 \), but since this cannot be guaranteed in practice, \( \delta \) is taken as

\[ \delta = q^{(k+1)'}(\theta)p^{(k)} - \eta \]

To show how equation (2.9) can be written as a particular form of the equation defined in Step 6, define

\[ t = \mu q^{(k)}(\theta) + \frac{1}{\mu \delta} \chi^{(k)} \]

where \( \mu \) is an undetermined constant. Then

\[ tt' = \frac{1}{\delta} \left( q^{(k)}(\theta)\chi^{(k)} + \chi^{(k)} q^{(k)}(\theta) + \mu^2 q^{(k)}(\theta) q^{(k)}(\theta)' \right) + \frac{1}{\mu \delta^2} \chi^{(k)} \chi^{(k)'} \]

This formula can be used with (2.9) to give

\[ H^{(k+1)}(\theta) = H^{(k)}(\theta) + \left( \frac{1}{\delta \lambda} - \frac{\eta}{\delta^2} - \frac{1}{\mu^2 \delta^2} \right) \chi^{(k)} \chi^{(k)'} + \]

\[ tt' - \mu^2 q^{(k)}(\theta) q^{(k)'}(\theta) \]

Now the coefficient of \( \chi^{(k)} \chi^{(k)'} \) can be made equal to zero by choosing \( \mu \) to satisfy

\[ \mu^2 = \frac{\lambda}{\delta - \lambda \eta} \]

or
Thus we get the recurrence relation

\[ H^{(k+1)} = H^{(k)} + \frac{1}{\mu^2 \delta^2} \tilde{t} t' - \mu^2 g^{(k)}(\theta) g^{(k)}(\theta)' \]  

where

\[ \tilde{t} = (\mu^2 \delta - 1) g^{(k)}(\theta) + g^{(k+1)}(\theta). \]

### 2.2.5 The Complementary DFP Updating Formula (CompDFP)

This updating scheme is given by Broyden (1970) and Fletcher (1970), where the formula for the approximate inverse Hessian is given by

\[ G^{(k+1)}(\theta) = G^{(k)}(\theta) + \frac{1}{\rho} \{ P^{(k)}(\theta)' P^{(k)}(\theta) - P^{(k)}(\theta)' Y^{(k)} G^{(k)}(\theta) \} \]

where

\[ \rho = \lambda^{(k)} + \frac{Y^{(k)} G^{(k)}(\theta) Y^{(k)}}{P^{(k)} Y^{(k)}}. \]

This formula is the complementary DFP formula, that is, when \( Y^{(k)} \) and \( \lambda^{(k)} P^{(k)} \) are interchanged, equation (2.8) corresponds to the above \( G^{(k+1)}(\theta) \), that is,

\[ G^{(k+1)}(\theta) = G^{(k)}(\theta) + \frac{Y^{(k)} Y^{(k)'}}{\lambda^{(k)} P^{(k)' Y^{(k)}}} + \frac{Q^{(k)} Q^{(k)'}}{P^{(k)' Q^{(k)}}}. \]

The complementary DFP formula in particular has been found to work well in practice, perhaps even better than the DFP formula. It has usually been implemented in conjunction with low accuracy line searches.
In the variable-metric methods which we described in the preceding sections, the approximation $G^{(k+1)}(\theta)$ to the inverse Hessian of the objective function $\phi$ at $\theta^{(k+1)}$ is obtained by adding either a matrix of rank 1 or a matrix of rank 2 to $G^{(k)}(\theta)$. But for some $k$, $G^{(k)}(\theta)$ may not be positive definite (due to rounding error), so a special method must be employed to ensure that the matrix $G^{(k)}(\theta)$ is positive definite for all values of $k$. By resetting $G^{(k)}(\theta)$ to the unit matrix whenever $\phi$ cannot be decreased by searching along $p^{(k)}$ is not a wholly desirable strategy, because in discarding $G^{(k)}(\theta)$ we throw away the only knowledge about the curvature of $\phi$ which is available for use in the algorithm.

However, Gill, Murray and Pitfield (1972) have described an implementation of variable-metric methods which has several advantages over the traditional implementations. In this method, the current estimate of the Hessian matrix is updated, rather than the current estimate $G^{(k)}(\theta)$ of the inverse Hessian.

(a) The Basic Iteration of GMP

Algorithm 2.5

Step 0: Given $\theta^{(k)}$ and $q^{(k)}(\theta)$, calculate $p^{(k)}$ by solving the set of equations

$$H^{(k)}p^{(k)} = -q^{(k)}(\theta).$$

The matrix $H^{(k)}$ is recurred in the form

$$H^{(k)} = L^{(k)}D^{(k)}L^{(k)^T}.$$
where $L^{(k)}$ is a unit-lower triangular matrix, and $D^{(k)}$ a diagonal matrix. The vector $p^{(k)}$ can be found by solving successively

$$L^{(k)}v = -q^{(k)}$$

and

$$L^{(k)}p^{(k)} = D^{(k)}^{-1}v$$

More explicitly, we have

$$v_i = -q_i^{(k)} - \sum_{j=1}^{i-1} \frac{L_{ij}^{(k)}}{d_i^{(k)}} v_j$$

and

$$p_i^{(k)} = \frac{v_i^{(k)}}{d_i^{(k)}} - \sum_{j=i+1}^{n} \frac{L_{ij}^{(k)}}{d_i^{(k)}} p_j^{(k)}$$

which require $n^2$ multiplications and $n$ divisions.

**Step 1:** Set $\theta^{(k+1)} = \theta^{(k)} + \lambda^{(k)} p^{(k)}$

and

$$q^{(k+1)}(\theta) = q(\theta^{(k+1)})$$

and $\lambda^{(k)}$ is a scalar such that

$$\phi(\theta^{(k)} + \lambda^{(k)} p^{(k)}) = \min_{\lambda} \phi(\theta^{(k)} + \lambda p^{(k)})$$

**Step 2:** Modify the triangular factors of $H^{(k)}$ so that

$$H^{(k+1)} = H^{(k)} + \Pi_1 z^{(k)} z^{(k)} + \Pi_2 \omega^{(k)} \omega^{(k)}$$
where \( z^{(k)} \), \( w^{(k)} \) are \( n \times 1 \) vectors and \( \Pi_1 \), \( \Pi_2 \) are scalars.

Consequently, this revised algorithm uses a formula similar to the recurrence formula (2.10).

(b) **Maintenance of Positive Definiteness**

We consider the matrix

\[
H^{(k+1)} = H^{(k)} + \sigma z^{(k)} z^{(k)\prime}
\]

where \( \sigma \) is a scalar and \( z^{(k)} \) an \( n \times 1 \) column vector. The modification to the Cholesky factorisation (Appendix A) is performed as follows:

Rewrite the above equation as

\[
H^{(k+1)} = L^{(k)} D^{(k)} L^{(k)\prime} (I + W W') D^{(k)\prime} L^{(k)}
\]  

(2.12)

where

\[
L^{(k)} D^{(k)} L^{(k)\prime} = z^{(k)}
\]

Equation (2.12) is then factorised into the form

\[
H^{(k+1)} = L^{(k)} D^{(k)} L^{(k)\prime} (I - \sigma^{(1)} W W') (I - \sigma^{(1)} W W') D^{(k)\prime} L^{(k)}
\]  

(2.13)

by writing

\[
\sigma^{(1)} = - \frac{\sigma}{1 + (1 + W W')^{\frac{1}{2}}}
\]
The matrix \( L^{(k)}D^{(k)}(I - \sigma^{(1)}VV') \) is factorised into the product of a lower triangular and an orthogonal matrix. If \( H^{(k+1)} \) is indefinite, \( \sigma^{(1)} \) is not real, and \( H^{(k+1)} \) must be replaced by a positive definite matrix \( \bar{H} \) to guarantee a downhill direction of search. \( \bar{H} \) is obtained by redefining \( \sigma^{(1)} \) as

\[
\sigma^{(1)} = \frac{-\sigma}{1 + (1 + |\sigma| V'V)^{1/2}}
\]

by the nature of factorisation (2.13) (see Appendix A), \( \bar{H} \) will be positive definite and this property cannot be affected by cumulative rounding errors (as happened with earlier algorithms).

2.2.7 The BHHH Hessian Approximation

The BHHH (1974) Hessian approximation is based on the fact that for correctly specified models, the Hessian matrix of the likelihood function at the minimising value of \( \theta \) is equal to the variance-covariance matrix of the gradient of the likelihood function. The result can be used to give a computationally efficient approximation based on the information needed to calculate the gradient, avoiding both the third derivatives required by the analytic Hessian and the repeated function evaluations required by numerical approximation. The approximation used is positive definite almost always, and so should not suffer from the errors associated with the inversion of an ill-conditioned matrix. Its drawbacks are:

(i) that its approximation need not be very good in small samples or for misspecified models, and
(ii) that it provides a consistent estimate of the Hessian only at the true value of $\theta$, but in so far as the maximum likelihood estimator is consistent, it can be expected for large samples to provide a good approximation in some neighbourhood of the maximum likelihood estimate.

The BHHH method is an example of a general class (the Gauss-Newton class) of optimisation methods which make use of the statistical properties of the likelihood function and its derivatives. Briefly, at the $k^{th}$ iteration, the BHHH Hessian matrix is approximated by

$$H^{(k)}(\theta) = \frac{1}{T} \left( \frac{\partial \phi}{\partial \theta} \right) \left( \frac{\partial \phi}{\partial \theta} \right)^\prime \bigg|_{\hat{\theta}(k)}$$

where $T$ is the sample size and $\hat{\theta}(k)$ is the current estimate of the true value $\theta$. Let $g^{(k)}(\theta) = \left( \frac{\partial \phi}{\partial \theta} \right) \bigg|_{\hat{\theta}(k)}$, then the iterative algorithm is:

$$\frac{1}{T} \left[ \left( \frac{\partial \phi}{\partial \theta} \right) \left( \frac{\partial \phi}{\partial \theta} \right)^\prime \right] \Delta \hat{\theta}^{(k)} - \lambda^{(k)} \left( \frac{\partial \phi}{\partial \theta} \right) = 0$$

and the basic Newton step becomes:

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \lambda^{(k)} \left[ \left( H^{(k)}(\theta) \right)^{-1} g^{(k)}(\theta) \right]_{\theta = \hat{\theta}(k)}$$

$$= \hat{\theta}^{(k)} + \lambda^{(k)} P^{(k)}$$

A detailed description of the BHHH method and algorithm is given in Chapter 3.
2.3 Minimisation Without Derivatives

We now describe the two most common optimisation techniques for minimisation without derivatives, and also discuss the disadvantages of such methods in our optimisation problem.

2.3.1 Conjugate Direction Methods

In order to define conjugate directions clearly, we begin by supposing that \( \phi(\theta) \) is a homogeneous positive definite quadratic function, whose second derivative matrix is \( H(\theta) \). Then the \( n \) non-zero directions \( p_i \) (\( i = 1, 2, \ldots, n \)) are mutually conjugate if and only if the equations

\[
p_i' H_\theta p_j = 0, \quad i \neq j
\]

hold.

Conjugate directions are important to minimisation algorithms, because, in the above quadratic case, the following construction calculates one vector of variables that minimises \( \phi(\theta) \). Let \( \theta_0 \) be any starting vector. For \( i = 1, 2, \ldots, n \), we let \( \theta_i \) be the vector

\[
\theta_i = \theta_{i-1} + \lambda_i p_i,
\]

where \( \lambda_i \) is the value of \( \lambda \) that minimises the function of that variable

\[
F_i(\lambda) = \phi(\theta_{i-1} + \lambda p_i).
\]

Then \( \theta_n \) is the point at which \( \phi(\theta) \) is least.
In a conjugate direction method for minimising a general function without calculating derivatives, we begin the \( k \)th iteration at the point \( \theta^{(k)} \), with search directions \( p_i^{(k)} \) \((i = 1, 2, \ldots, n)\). Initially these directions are the co-ordinate directions, but they are modified on each iteration by some method that should tend to make them mutually conjugate with respect to the Hessian matrix at the solution, \( \theta^* \) say. The main operation of the \( k \)th iteration is to let \( \theta^{(k+1)} = \theta^{(k)} \), and for \( i = 1, 2, \ldots, n \) to define \( \theta_i^{(k)} \) to be the point

\[
\theta_i^{(k)} = \theta_i^{(k-1)} + \lambda_i^{(k)} p_i^{(k)} , \quad (2.17)
\]

where again \( \lambda_i^{(k)} \) is determined by a line search to minimise \( \phi(\theta_i^{(k)} + \lambda_i^{(k)} p_i^{(k)} ) \) with respect to \( \lambda_i \). We then set \( \theta^{(k+1)} = \theta_n^{(k)} \).

Thus if \( \theta^{(k)} \) is close to \( \theta^* \), and if the search directions are almost mutually conjugate, we expect \( \theta^{(k+1)} \) to be much better than \( \theta^{(k)} \) as an estimate of \( \theta^* \). However, this description omits the steps required to modify the \( p_i^{(k)} \). To do this the \( k \)th iteration obtains the directions \( p_i^{(k+1)} \) \((i = 1, 2, \ldots, n)\), which may involve some more values of the objective function. Then a few extra function values may be needed to fix \( \theta^{(k+1)} \). Usually the value of \( \phi(\theta^{(k+1)} ) \) is the least calculated value of the objective function, and always satisfies the inequality \( \phi(\theta^{(k+1)} ) \leq \phi(\theta^{(k)} ) \).

For example, most versions of Powell's (1964) algorithm use the formulae

\[
p_i^{(k+1)} = p_i^{(k)} , \quad i = 1, 2, \ldots, n-1
\]

\[
p_n^{(k+1)} = \sum_{i=1}^{n} \lambda_i^{(k)} p_i^{(k)} , \quad (2.18)
\]
and $\theta_{(k+1)}$ is obtained by a line search from $\theta_{n(k)}$ in the direction $p_{n(k+1)}$. Provided $\lambda_{1(k)}$ is non-zero for all values of $k$, it may be proved that this method obtains the least value of a quadratic function in at most $n$ iterations.

The conjugate direction methods avoid the two main drawbacks of the variable-metric methods, for they do not require values of $\theta_{k}(\theta)$ $(k = 1, 2, \ldots)$, and most function values are applied to reducing the objective function. However, they too have some disadvantages.

One is that it is sometimes awkward to ensure that for all $k$ the directions $p_{1(k)}$ $(i = 1, 2, \ldots, n)$ have good linear independence properties. For example, if $\lambda_{1(k)}$ is small in comparison with $\lambda_{1(k)}$ $(i = 2, 3, \ldots, n)$, then equation (2.18) requires modification. In this case Powell's (1964) algorithm makes the search direction $p_{n(k+1)}$ equal to $p_{1(k)}$, although this change weakens the quadratic termination properties of the method, which often loses efficiency, particularly when $n$ is greater than about ten.

To avoid this difficulty, Brent (1973) suggests a different modification to Powell's algorithm, which requires the eigenvalues and eigenvectors of an $n \times n$ symmetric matrix to be calculated after every $n$ iterations. The extra work of the eigen problems can cause the total computing time to be greater than before, if each evaluation of $\phi(\theta)$ requires comparatively little time. However, it usually gives a worthwhile reduction in the number of function values needed for the whole minimisation calculation, so Brent's method is recommended for serious problems, where the calculation of $\phi(\theta)$ is quite long.

Another disadvantage is that conjugate directions may not be well-determined for certain non-quadraic functions. For such ill-conditioned functions, if the second derivative matrix of $\phi(\theta)$ at
certain points is almost singular, it is usual to have a line of such points near the bottom of a curved valley. Therefore it is calamitous that minimisation algorithms often generate sequences of points \( \theta^{(k)} \) \((k = 1, 2, \ldots) \) that follow curved valleys. Thus the aim of trying to obtain linearly independent conjugate directions, including search directions which allow moves along the floors of any valley, makes the criterion for the choice of new conjugate directions ambiguous. These remarks make the justification for conjugate direction methods with such difficult functions rather uncertain, except in regions of \( \theta \)-space where \( \phi(\theta) \) satisfies a strict convexity condition.

2.3.2 Variable-Metric Methods

The other optimisation technique for minimisation without derivatives is that developed by Gill-Murray-Pitfield (GMP), this method is essentially the same as that of 2.2.6 except for the estimation of the gradient vector. At the beginning of the \( k \)th iteration \((k = 1, 2, \ldots)\) of a variable-metric method, we require a starting point \( \theta^{(k)} \), a vector \( g^{(k)}(\theta) \) and a symmetric matrix \( H^{(k)}(\theta) \). The vector \( g^{(k)}(\theta) \) is an estimate of the gradient of \( \phi(\theta) \) at \( \theta^{(k)} \), and the matrix \( H^{(k)}(\theta) \) is an estimate of the Hessian matrix of \( \phi(\theta) \) at \( \theta^{(k)} \).

Sometimes the errors in \( H^{(k)}(\theta) \) are quite large. For example, in many useful algorithms it is advantageous to force \( H^{(k)}(\theta) \) to be positive definite, even though the true second derivatives may have negative eigenvalues at \( \theta^{(k)} \). To simplify the description, we suppose in this section that \( H^{(k)}(\theta) \) is positive definite on every iteration.

The derivative estimates provide the quadratic approximation

\[
\phi(\theta^{(k)} + \delta) = \phi(\theta^{(k)}) + \delta^T g^{(k)}(\theta) + \frac{1}{2} \delta^T H^{(k)}(\theta) \delta \tag{2.19}
\]
The value of $\delta$ that minimises the right-hand side of (2.19) satisfies the equation

$$g_k(\delta) + H_k(\delta)\delta = 0$$  \hspace{1cm} (2.20)

Therefore some variable-metric methods define $\theta^{(k+1)}$ by the equation

$$\theta^{(k+1)} = \theta^{(k)} - [H^{(k)}(\delta)]^{-1}g^{(k)}(\delta).$$  \hspace{1cm} (2.21)

However, because this choice of $\theta^{(k+1)}$ may conflict with inequality $\phi^{(k+1)}(\delta) \leq \phi^{(k)}(\delta)$, it is usual to let $\theta^{(k+1)}$ be the vector

$$\theta^{(k+1)} = \theta^{(k)} - \lambda^{(k)} [H^{(k)}(\delta)]^{-1}g^{(k)}(\delta)$$  \hspace{1cm} (2.22)

where $\lambda^{(k)}$ is a scalar which is chosen to enforce the above condition, and possibly another condition also to ensure that $H^{(k+1)}(\delta)$ is positive definite. To determine the value of $\lambda^{(k)}$, we seek a good estimate of the least value of the function

$$F(\lambda) = \phi(\theta^{(k)} - \lambda[H^{(k)}(\delta)]^{-1}g^{(k)}(\delta))$$  \hspace{1cm} (2.23)

by calculating only a few actual values of $F(\lambda)$. A description of a suitable method for adjusting $\lambda$ is given in section 2.4.

Next the gradient of $\phi(\delta)$ at the point $\theta^{(k+1)}$ is estimated. Usually in non-derivative algorithms, either forward or central differences are employed, the $i$th component of $g^{(k+1)}(\delta)$ ($i = 1, 2, \ldots$) being defined by the equation

$$g_{1^{(k+1)}}(\delta) = \{\phi(\theta^{(k+1)} + h_1e_i) - \phi(\theta^{(k+1)})\}/h_1$$  \hspace{1cm} (2.24)

or by the equation
\[ g_{i}^{(k+1)}(\theta) = \frac{\phi(\theta^{(k+1)} + h_i e_i) - \phi(\theta^{(k+1)} - h_i e_i)}{2h_i}, \]  

(2.25)

where \( e_i \) is the \( i^{th} \) co-ordinate vector.

An important and valuable feature of the methods used to define \( H^{(k+1)}(\theta) \) is that they require no more values of the objective function. The successful choices of \( H^{(k+1)}(\theta) \) satisfy the equation

\[ H^{(k+1)}(\theta) [\theta^{(k+1)} - \theta^{(k)}] = [g^{(k+1)}(\theta) - g^{(k)}(\theta)] \]  

(2.26)

because, when \( \phi(\theta) \) is a quadratic function, this equation is also satisfied by the true Hessian matrix. One of the most useful choices of \( H^{(k+1)}(\theta) \) is that given by the Broyden-Fletcher-Shanno formula

\[ H^{(k+1)}(\theta) = H^{(k)}(\theta) - \frac{H^{(k)}(\theta) \delta^{(k)}H^{(k)}(\theta)'}{\delta^{(k)}H^{(k)}(\theta)\delta^{(k)}} + \frac{Y^{(k)}}{\delta^{(k)}H^{(k)}(\theta)^{\delta^{(k)}}} \]  

(2.27)

where \( \delta^{(k)} \) and \( Y^{(k)} \) are the differences

\[ \delta^{(k)} = \theta^{(k+1)} - \theta^{(k)} \]  

(2.28)

\[ Y^{(k)} = g^{(k+1)}(\theta) - g^{(k)}(\theta). \]

Thus the data that is needed to begin the next iteration is already calculated.

The extensive numerical results given by Gill-Murray-Pitfield (1972) indicate that the class of variable-metric methods contains the best of the available algorithms for minimisation without derivatives. However, each iteration of a variable-metric method uses at least \( n \).
function values to estimate first derivatives, but it uses only about three or four function values in the line search that seeks the minimum of the function (2.23). Thus in large problems only a small proportion of the function evaluations are applied directly to the main problem of reducing the objective function. This is a poor strategy unless \( \phi(\theta) \) is almost quadratic.

Another deficiency of variable-metric methods is that usually the search direction \(-[H^{(k)}(\theta)]^{-1}q^{(k)}(\theta)\) in expression (2.22) gives fast convergence only if the direction of \( q^{(k)}(\theta) \) is a good approximation to the direction of the true gradient of \( \phi(\theta) \) at \( \theta^{(k)} \). However, the true gradient should tend to zero as \( k \) increases, so the difficulties of calculating a suitable vector \( q^{(k)}(\theta) \) become more and more severe. Therefore many algorithms switch from formula (2.24) to formula (2.25) when \( q^{(k)}(\theta) \) becomes small, in order to obtain higher accuracy at the cost of almost doubling the number of function values per iteration. Thus the precision of the calculated values of \( \phi(\theta) \) is very important. To avoid these extra function values, Cullum (1972) suggest the formula

\[
g^{(k+1)}_i = \frac{[\phi(\theta^{(k+1)} + h_i e_i) - \phi(\theta^{(k+1)}) - \frac{1}{2} h_i^2 H^{(k)}_{i\bar{i}}(\theta)]/h_i}{h_i} (2.29)
\]

instead of equation (2.25), where \( H^{(k)}_{i\bar{i}}(\theta) \) is the \( i \)th diagonal element of \( H^{(k)}(\theta) \).

Another way of obtaining better accuracy in the differences (2.24) and (2.25) is to avoid the use of adaptive methods in the calculation of \( \phi(\theta) \). For example, if \( \phi(\theta) \) is a definite integral which is calculated by a numerical quadrature formula, and if the weights of the quadrature formula are held constant, then the leading error term of the quadrature formula usually cancels out when the difference (2.24) or (2.25) is formed.
The choice of the step-length $h_i$ in equation (2.24) and (2.25) also causes problems. The earliest variable-metric method (Stewart, 1967) includes a technique that chooses $h_i$ automatically, and numerical results show that it works quite well. However, Gill and Murray (1972) suggest that it is better to keep $h_i$ ($i = 1, 2, \ldots, n$) constant throughout the calculation, in order that the leading error terms in $g(k)(\Theta)$ and $g^{(k+1)}(\Theta)$ cancel when $y^{(k)}$ is calculated from expression (2.28). In our opinion, it is preferable if the step-lengths are adjusted automatically, so that people who do not understand the difficulties of numerical differentiation can apply the minimisation subroutines successfully, without expert advice on the choice of $h_i$.

2.3.3 Comments Regarding Minimisation Without Derivatives

Although the most successful algorithms now for minimisation without derivatives are variable-metric and conjugate direction methods, we have noted major disadvantages in both these classes of methods. Difficulties occur in variable-metric methods because of the strong dependence on accurate first derivatives, and in conjugate direction methods the revision of the conjugate directions can be a very poorly defined problem. However, the estimation of second derivatives in a variable-metric method seldom impairs efficiency, and the fact that conjugate direction methods usually search along $n$ independent directions on every iteration helps to avoid jamming away from the solution. But finding a good algorithm which may retain the advantages and lose the disadvantages of current algorithms may take a long time, particularly because comparisons should be made with current methods that have been designed and programmed carefully. Therefore, in this
study, we concentrate our effort on minimisation methods with analytic derivatives, and we will implement the BHHH and CMP methods in our computer program to estimate the parameters of the concentrated log-likelihood function.

2.4 Choice of Line Search

An important part of all these minimisation algorithms is the choice of the step-length \( \lambda^{(k)} \) along the direction \( p^{(k)} \).

Although some algorithms have been suggested which generally accept \( \lambda^{(k)} = 1 \), it is usual to require that \( \lambda^{(k)} \) is chosen to ensure that \( \phi(\hat{\theta}^{(k+1)}) \leq \phi(\hat{\theta}^{(k)}) \), which gives a minimal stability in the iteration. Although it may cost relatively little in computing time to ensure that \( \lambda^{(k)} \) is chosen so as to minimise

\[
\phi(\hat{\theta}^{(k)} + \lambda p^{(k)}),
\]

this may be relatively wasteful of computer time when the cost of computing \( \phi(\hat{\theta}) \) is high, or if \( n \) is so large that in the early iterations, when the direction \( p^{(k)} \) is relatively arbitrary, there is no great advantage in searching along the direction \( p^{(k)} \). It is necessary to balance the time taken in searching along the direction \( p^{(k)} \) with the time taken to choose a more suitable direction \( p^{(k+1)} \). This balance is clearly dependent on the properties of the function \( \phi(\theta) \), and is usually decided on the basis of experience with a variety of functions.
2.4.1 Quadratic Interpolation

Powell (1964) published a simple algorithm for determining the minimising value of \( \lambda \), using quadratic interpolation. This algorithm forms part of Powell's more general method of finding the minimum value of a function \( \phi(\theta) \) without calculating derivatives. However, it may also be used in conjunction with any gradient method, or more generally, with any optimisation technique which requires a one-dimensional search.

To find the minimum on a line, we must provide the following:

(i) a set of points (or a point) on the line, \( \theta \),
(ii) the direction of the line \( p \),
(iii) an upper bound to the length of step along the line, \( m \),
(iv) an order of magnitude of the length of step along the line, \( h \), assumed to be less than \( m \), and
(v) the accuracy to which the minimum is required, \( \epsilon \).

The method of minimisation must be such as to find the minimum of a quadratic form, so it is primarily based on the quadratic defined by three function values.

Initially \( \phi(\theta) \) and \( \phi(\theta + hp) \) are calculated, and then either \( \phi(\theta - hp) \) or \( \phi(\theta + 2hp) \) is worked out depending on whether \( \phi(\theta) \) is less than or greater than \( \phi(\theta + hp) \). These three function values are now used in the general formula which predicts the turning value of the quadratic defined by \( \{a, \phi(\theta + ap)\}, \{b, \phi(\theta + bp)\}, \) and \( \{c, \phi(\theta + cp)\} \) to be at \( \theta + \lambda p \), where
\[
\lambda = \frac{1}{2} \cdot \frac{(b^2 - c^2)\phi_a + (c^2 - a^2)\phi_b + (a^2 - b^2)\phi_c}{(b - c)\phi_a + (c - a)\phi_b + (a - b)\phi_c}.
\]

(2.30)

It is a minimum if

\[
\frac{(b - c)\phi_a + (c - a)\phi_b + (a - b)\phi_c}{(a - b)(b - c)(c - a)} < 0.
\]

(2.31)

If the turning value is predicted to be a maximum, or if the value of \( \lambda \) is such that to calculate \( \phi(\theta + \lambda \rho) \) a step greater than \( m \) must be taken, the maximum allowed step is taken in the direction of decreasing \( \phi \), and the function value at the point which is furthest from \( (\phi + \lambda \rho) \) is discarded, so the prediction may be repeated.

Otherwise \( \lambda \) is compared with \( a, b, c \), and, if it is within the required accuracy of one of them, that point is chosen as the minimum. If it is not, \( \phi(\theta + \lambda \rho) \) is calculated so that the quadratic prediction may be repeated; the function value which is thrown away out of \( \phi(\theta + \rho \rho), \phi(\theta + \rho p) \) and \( \phi(\theta + \rho p) \) is normally the greatest, but it is not if rejecting a smaller one can yield a definite bracket on a minimum, which would not be obtained otherwise.

In order to reduce the number of times \( \phi(\theta_1, \theta_2, \ldots, \theta_n) \) has to be calculated, advantage may be taken of the fact that three function values are sufficient to predict

\[
\frac{\partial^2}{\partial \lambda^2} \{\phi(\theta + \lambda \rho)\}.
\]

(2.32)
The prediction of the second derivative is

\[ \xi = -2 \cdot \frac{(b - c) \phi_a + (c - a) \phi_b + (a - b) \phi_c}{(a - b)(b - c)(c - a)} \]  (2.33)

So, if after finding the minimum in the direction \( p \) the components of \( p \) are scaled by \( 1/\sqrt{\xi} \), the next time a minimum is sought in the same direction the unit second derivative may be used. In this case just \( \phi(0) \) and \( \phi(\theta + h\lambda) \) are sufficient to predict the minimum to be at \( (\theta + \lambda \eta) \),

\[ \lambda = \frac{1}{2}h - \frac{\phi(\theta + h\lambda) - \phi(\theta)}{h} \]  (2.34)

Choice of \( h \)

It is important to have a method of adjusting the step-length \( h \) before entering the line search procedure to ensure that a definite bracket on a minimum is located without too many function evaluations. Assume we have the initial step \( \lambda = 1 \), and during each iteration a new step \( \lambda^* \) is obtained, we can then set

\[ D = ||p||_2 \cdot \lambda^* \]

and define

\[ \lambda = \min \{ \max \left(\frac{\lambda}{2}, 2D\right), 2\lambda \} \]

that is,
if $\lambda > D \geq \lambda/4$, reset $\lambda = 2D$

if $\lambda/4 > D$, reset $\lambda = \lambda/2$.

This will ensure that $\lambda/2 \leq \lambda \leq 2\lambda$.

We then set

$$h = \min\left(1, \frac{\lambda}{\|p\|_2}\right)$$

as our starting step-length in the line search algorithm.

A simple method for choosing $h$ is described in the following steps:

1. Compute $d = \|p\|_2$.
2. Set $h = \min(1, \lambda/d)$.
3. Call line search to locate a new step-length $\lambda^*$.
4. Compute $D = d \times \lambda^*$.
5. Reset $\lambda = \min\{\max(\lambda/2, 2D), 2\lambda\}$.
6. Return to the optimisation routine.

Consequently, our estimation program for the non-linear econometric system uses the above procedures for the line search and adjusting the step-length $h$. 
2.4.2 Cubic Interpolation

The line search given in the GMP procedure is that suggested by Davidon (1959). Given two points \( \lambda_1 \) and \( \lambda_2 \) with function values \( \phi_1 \) and \( \phi_2 \) and derivatives \( g_1 = g(\theta^{(1)}) \) and \( g_2 = g(\theta^{(2)}) \), a stationary point \( \lambda^* \) of the third order polynomial passing through these two points and having the specified derivative values is given by

\[
\lambda^* = (\lambda_2 - \lambda_1)(1 - (g_2 - \gamma - \eta)/(g_2 - g_1 + 2\gamma)) \quad (2.35)
\]

where

\[
\gamma = (\eta^2 - g_1 g_2)^{1/2}
\]

and

\[
\eta = 3(\phi_1 - \phi_2)/(\lambda_2 - \lambda_1) + g_1 + g_2.
\]

The stationary point defined above is the one which lies in the interval \((\lambda_1, \lambda_2)\) if the minimum of \( \phi(\theta) \) along \( \theta^{(k)} \) lies in this interval. Assuming \( \lambda_1 < \lambda_2 \) then the minimum lies in the interval \((\lambda_1, \lambda_2)\) if \( g_1 < 0 \) and \( g_2 > 0 \).

In the non-derivative case, quadratic interpolation is applied. The stationary point \( \lambda^* \) of the second order polynomial passing through three points is given by equation (2.30).
2.4.3 Bard Line Search

An alternative method of choosing $\lambda^{(k)}$ is given by Bard (1970).

Define

$$F_k(\lambda) \equiv \phi(\theta^{(k)}) + \lambda[H^{(k)}(\theta)]^{-1}g^{(k)}(\theta),$$

where $g^{(k)}(\theta)$ is the gradient vector and $H^{(k)}(\theta)$ is the approximate Hessian matrix.

$$\lambda_{\text{max}} \equiv \text{upper bound on } \lambda_i.$$

Bard considers the case where there are inequality constraints and in this case, $\lambda_{\text{max}}$ is determined as the minimum positive $\lambda$ such that $\theta + \lambda[H(\theta)]^{-1}g(\theta)$ is on a constraint.

When there is no inequality, $\lambda_{\text{max}}$ is set to an arbitrary large number.

At the start of the $k^{\text{th}}$ iteration, we possess the value

$$F_k(\theta) = \phi(\theta^{(k)})$$

and

$$F_k'(\lambda) \equiv \frac{dF_k}{d\lambda} \bigg|_{\lambda=0} = g^{(k)}(\theta)[H^{(k)}(\theta)]^{-1}g^{(k)}(\theta).$$

$\lambda^{(o)}$ is assumed given at the start of the $k^{\text{th}}$ iteration and define $F^{(o)} = F_k(\lambda^{(o)})$. 
The Basic Algorithm

Step 0: Compute $F^{(0)}$. If $F^{(0)} < F_k^{(0)}$ accept $\lambda^{(k)} = \lambda^{(0)}$, otherwise continue.

Step 1: Determine the second degree polynomial in $\lambda$ which agrees with $F_k(\lambda)$ at $\lambda = 0$ and $\lambda = \lambda^{(0)}$, and whose slope at $\lambda = 0$ agrees with $F_k'(0)$. Let $\lambda^{(1)}$ be the point at which this polynomial is stationary, that is, define

$$F_k^{*}(\lambda) = F_k^{(0)} + F_k'(0)\lambda + a\lambda^2$$

where $a$ is chosen so that

$$F_k^{(1)}(\lambda^{(0)}) = F_k^{(0)} + F_k'(0)\lambda^{(0)} + a\lambda^{(0)^2}.$$ 

Then we have

$$a = \frac{F_k^{(1)}(\lambda^{(0)}) - F_k^{(0)} - F_k'(0)\lambda^{(0)}}{\lambda^{(0)^2}}.$$ 

The stationary value is given by

$$F_k'(0) + 2a\lambda = 0,$$ 

and

$$\lambda^{(1)} = -\frac{F_k'(0)}{2a} = -\frac{F_k'(0)\lambda^{(0)^2}}{2\{F_k(\lambda^{(0)}) - F_k^{(0)} - F_k'(0)\lambda^{(0)}\}}.$$
Step 2: If this is the first \( \lambda^{(1)} \) calculated for the \( k \)th iteration go to Step 3, otherwise define

\[
\lambda^{(2)} = \max\{0.25\lambda^{(0)}, \min(0.75\lambda^{(0)}, \lambda^{(1)})\},
\]

that is, if \( 0.25\lambda^{(0)} < \lambda^{(1)} < 0.75\lambda^{(0)} \), set \( \lambda^{(2)} = \lambda^{(1)} \)

if \( \lambda^{(1)} > 0.75\lambda^{(0)} \), set \( \lambda^{(2)} = 0.75\lambda^{(0)} \)

if \( \lambda^{(1)} < 0.25\lambda^{(0)} \), set \( \lambda^{(2)} = 0.25\lambda^{(0)} \)

Replace \( \lambda^{(0)} \) with \( \lambda^{(2)} \) and return to Step 0.

Step 3: Define \( \lambda^{(3)} = \min(\lambda^{(1)}, 0.75\lambda^{\max}) \).

Step 4: If \( |\lambda^{(3)} - \lambda^{(0)}| \leq 0.1\lambda^{(0)} \) or \( \lambda^{(1)} \leq 0.25\lambda^{(0)} \), accept \( \lambda^{(k)} = \lambda^{(o)} \), otherwise continue.

Step 5: Compute \( F^{(3)} = F_k(\lambda^{(3)}) \). Take \( \lambda^{(k)} = \lambda^{(3)} \) or \( \lambda^{(k)} = \lambda^{(o)} \) depending on whether \( F^{(3)} \) or \( F^{(0)} \) is the smaller.

Bard line search is different from other methods because it considers the problem whether \( \phi(\delta^{(k)} + \lambda\rho^{(k)}) \) is on a constraint.

2.4.4 The BHHH Line Search

To choose \( \lambda^{(k)} \), Berndt-Hall-Hall-Hausman (1974) suggest that an arbitrary \( \delta \) is chosen, \( 0 < \delta < 1 \).

The BHHH procedure is then to take \( \lambda = 1 \) if

\[
\phi(\delta^{(k)} + \rho^{(k)}) - \phi(\delta^{(k)}) \geq \delta\rho^{(k)}'g^{(k)}(\delta)
\]

and otherwise choose \( \lambda^{(k)} \) such that
\[ \delta \lambda^{(k)} p^{(k)} q^{(k)}(\theta) \leq \phi(\theta^{(k)} + \lambda^{(k)} p^{(k)}) - \phi(\theta^{(k)}) \]
\[ \leq (1 - \delta) \lambda^{(k)} p^{(k)} q^{(k)}(\theta). \]  

Now if condition (2.36) is not satisfied,
\[ \phi(\theta^{(k)} + \lambda p^{(k)}) - \phi(\theta^{(k)}) < \delta \lambda p^{(k)} q^{(k)}(\theta) \]
for \( \lambda \) just less than 1, and
\[ \left( \frac{\phi(\theta^{(k)} + \lambda p^{(k)}) - \phi(\theta^{(k)})}{\lambda p^{(k)} q^{(k)}(\theta)} \right) \rightarrow 1 \text{ as } \lambda \rightarrow 0. \]

Thus by reducing \( \lambda \) from 1, we can find a \( \lambda^{(k)} \) satisfying equation (2.37). Unfortunately this may be a time-wasting procedure, since we often find that it is necessary to consider several values of \( \lambda \) before a suitable \( \lambda^{(k)} \) is found.

2.4.5 Efficiency and Termination

The four possible line search procedures have been programmed. Each of these has been tested in separate runs on a set of simulated non-linear simultaneous models.

From the results, it became clear that procedures Bard and BHHH are relatively time-wasting compared with quadratic and cubic interpolations because we have to compute several \( \lambda^{(k)} \) before we can locate a suitable \( \lambda^{(k)} \). It is expensive in terms of computing time to compute many function evaluations as the objective function \( \phi \) can be very complex. These extra function values required in each line search
are not compensated for by finding a lower value of $F(\lambda)$, nor do the theoretically better convergence properties show up in our comparisons. Hence more iterations are needed in the iterative procedures for the model to achieve convergence.

Cubic interpolation again has a drawback because it is relatively expensive to compute the gradient of $\phi$ at $\lambda_2$, but it is a more efficient method compared with Bard and BHHH.

We would recommend the use of quadratic interpolation as a line search procedure. Since the step-length $h$ is adjusted during each iteration, we have a good projection of $\lambda^{(k)}$ for a start and hence reduce the possibility of searching too many $\lambda^k$ on the line $\phi(\delta + \lambda^k)$. The average number of function evaluations in this procedure is between 1 to 2. Since we can locate a good estimate of $\lambda^{(k)}$ with a smaller number of function evaluations, we are not only reducing the computing time for function evaluations, but also the time taken to optimise the model.

2.5. **Choice of Stopping Criterion**

Determining when to stop the iterations that lead to a minimum of $\phi$ is a problem of great practical interest: stopping short of the mark has its obvious costs in the quality of the estimates; going too far involves unnecessary costs in computer time.

2.5.1 **The Gradient Stopping Criterion**

In principle there seems little problem determining when to stop: at the minimum the gradient is zero. Thus it is common practice to
choose some arbitrarily small $\varepsilon$, such as $10^{-4}$ or $10^{-5}$, and to stop when the largest gradient (in absolute value) is less than $\varepsilon$.

This stopping criterion, called the gradient criterion, can also be effected by stopping when the square length of $g$ (where $g = g^{(k)}(\theta)$) is small, that is, when

$$\|g\|_2 \leq \varepsilon.$$ 

The gradient criterion has two major weaknesses. First, it is scale-sensitive. Changes in the units in which the data are measured can cause the scale of specific parameters and their gradients to be made arbitrarily large or small. In econometric problems, parameters that are naturally small will tend to have relatively large gradients that can keep the optimiser seeking a minimum long after it is close enough for practical purposes. Similarly, large coefficients (some constant terms) may have relatively small gradients that can be ignored by this criterion even when they should not be. In practice, the gradient stopping criterion is very conservative, tending to drive the optimiser beyond the point of diminishing returns in terms of parameter estimates. It tends, therefore, to be a good criterion when we wish to be sure to go far enough.

A second weakness of the gradient criterion is that it ignores the statistical context of likelihood estimation and treats all parameters alike - whether they are significant or not. It is quite possible for a large gradient in the direction of a wholly insignificant parameter estimate to force the continuation of the optimisation process even though those parameters that are estimated with significance are changing little.
2.5.2 The Weighted-Gradient Stopping Criterion

Here the weighted-gradient stopping criterion is introduced, that is,

\[- g' H^{-1} g \leq \epsilon, \quad \text{where} \quad H = H^{(k)}(\theta)\]

This criterion equals zero (assuming $H^{-1}$ is negative definite) if and only if $g = 0$, and it is scale-invariant. If $H$ is ill-conditioned $- g' H^{-1} g$ could be large even if $g$ is small. This characteristic is in fact an advantage of the weighted-gradient criterion in the NLFIML context, for, near the maximum likelihood solution, the negative of the Hessian estimates the variance-covariance matrix of $g$. An ill-conditioned Hessian occurs when some element of $\theta$ has a high variance and the corresponding element of $g$ has a small variance. Thus, this criterion incorporates a weighting scheme that, near the solution, takes into account the precision with which the gradient components are known: gradients with large variances are appropriately downweighted or conversely. Therefore the weighted-gradient criterion would seem to have value as a stopping criterion. However, in practice we found criteria of this kind tended to stop earlier than other criteria, and we preferred to make use of criteria which were not so directly related to the statistical properties of our estimators, but rather to the numerical properties of the errors in the parameters or derivatives.

For our stopping criteria in the estimation program, we would use:
where $\varepsilon_1$, $\varepsilon_2$ and $\varepsilon_3$ are prescribed tolerance levels. If any two of the above stopping criteria are satisfied, we will then terminate the iterative procedure.
CHAPTER 3

CHARACTERISTICS OF ECONOMETRIC ESTIMATION PROBLEMS

3. The Model

We are concerned with estimation in the multivariate model

\[ f_i(y_{it}; z_{it}; \theta) \equiv f_i(y_{1t}, \ldots, y_{nt}, z_{1t}, \ldots, z_{mt}; \theta_1, \ldots, \theta_K) \]

\[ = u_{it} \quad i = 1, \ldots, n \]

\[ t = 1, \ldots, T \quad (3.1) \]

where \( u_{it} = (u_{1t}, \ldots, u_{nt})' \) is the vector of normally distributed, serially independent disturbances with mean zero and a symmetric positive definite variance covariance matrix \( \Omega \), \( \theta \) is the vector of \( K \) unknown parameters, \( f_i(\cdot; \cdot; \cdot) \) is a twice continuously differentiable function, \( y_t \) is an \((n \times 1)\) vector of jointly dependent variables and \( z_t \) is an \((m \times 1)\) vector of predetermined variables.

We now set up a likelihood function based on a multivariate normal distribution for the \( u_{it} \). Noting that the joint probability density function for \( u_t \) is

\[ (2\pi)^{-\frac{n}{2}} |\Omega|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} u_t' \Omega^{-1} u_t \right\}, \]

the joint probability for the \( T \) observations \((y_t; z_t; t = 1, \ldots, T)\) is:
\[ d^n_p = (2\pi)^{-\frac{nT}{2}} (\det \Omega)^{-\frac{T}{2}} \left( \prod_{t=1}^{T} |\det J_t| \right) \exp \left\{ -\frac{1}{2} \sum_{ij} f_{it} \Omega^{-1}_{ij} f_{jt} \right\} dv \]

where

\[ dv = \prod_{t=1}^{T} (dy_{1t}, dy_{2t}, ..., dy_{nt}) \]

\[ f_{it} = f_i(y_t; z_t; \theta) \]

\[ \det J_t \text{ is the Jacobian determinant (i.e. the determinant of first derivatives w.r.t. } y_t) \text{ with} \]

\[ J_t = \begin{bmatrix} \frac{\partial f_{it}(y_t; z_t; \theta)}{\partial y_{1t}} \\ \vdots \\ \frac{\partial f_{it}(y_t; z_t; \theta)}{\partial y_{nt}} \end{bmatrix} \]

\[ \Omega_{ij} \text{ is the } ij^{th} \text{ element of } \Omega^{-1}. \text{ The logarithm of the likelihood is then} \]

\[ L(\theta, \Omega) = -\frac{nT}{2} \log 2\pi + \frac{T}{2} \log (\det \Omega^{-1}) + \sum_{t} \log |\det J_t| \]

\[ -\frac{1}{2} \sum_{i} \sum_{j} f_{it} \Omega^{-1}_{ij} f_{jt} \]

The Maximum Likelihood (ML) estimator of \( \theta \) and \( \Omega \) is implicitly defined as a solution to the following necessary first order conditions for the maximum of the log-likelihood function \( L(\theta, \Omega) \):

\[ \frac{\partial L(\theta, \Omega)}{\partial \Omega_{ij}} = 0 \quad \text{for } i = 1, ..., n \]

\[ \frac{\partial L(\theta, \Omega)}{\partial \Omega_{ij}} = 0 \quad \text{for } j = 1, ..., i \]

(3.5)
Finding the ML estimator involves solving these equations for $K + n(n+1)/2$ unknown parameters. The dimensionality (and quite possibly the computational expense) of this problem can be reduced considerably by noting that the elements of $\Omega$ are unrestricted (except that $\Omega$ is symmetric and positive definite). Hence $\Omega$ can be eliminated from the log-likelihood function by concentrating it out, that is, by solving the ML estimator of $\Omega$ in (3.5) analytically and substituting $\Omega$ by its ML estimator in (3.4).

Because $\Omega^{-1}$ is symmetric,

$$
\frac{\partial \log (\det \Omega^{-1})}{\partial \Omega_{ij}} = \begin{cases} 
\Omega_{ii} & \text{if } i = j \\
2\Omega_{ij} & \text{if } i \neq j
\end{cases}
$$

(3.7a)

and so

$$
\frac{\partial L(\theta, \Omega)}{\partial \Omega_{ij}} = \begin{cases} 
\frac{T}{2} \Omega_{ii} - \frac{1}{2} \sum_{t} f_{it} f_{jt} \\
T\Omega_{ij} - \sum_{t} f_{it} f_{jt}
\end{cases}
$$

(3.7b)

Setting the derivatives in (3.7b) equal to zero as in (3.5) gives the ML estimator of $\Omega_{ij}$ $(i, j = 1, \ldots, n)$:

$$
\frac{1}{T} \sum_{t} f_{it} f_{jt}.
$$

(3.8)

Upon substituting this into the log-likelihood function in (3.4), the last term of that equation is
Hence the last term in equation (3.4) is a constant. The non-constant part of \( L(\theta) \) is

\[
\frac{T}{2} \log (\det \Omega^{-1}) + \sum_t \log |\det J_t| .
\]  

(3.10)

Thus the concentrated log-likelihood function is

\[
L^*(\theta) = c + \sum_t \log |\det J_t| - \frac{T}{2} \log \left( \det \left( \sum_t \frac{f_t f^*_t}{T} \right) \right)
\]  

(3.11)

where \( c \) is the constant \( \frac{nT}{2} \log T - \frac{nT}{2} \log (2\pi) - \frac{nT}{2} \). \( L^*(\theta) \) is a function of \( \theta \) only (and not \( \Omega \)). Further the ML estimator for \( \theta \) and \( \Omega \) obtained from (3.5) and (3.6) is identical to that obtained by solving for \( \partial L^*(\theta)/\partial \theta_\alpha = 0 \), \( \alpha = 1, \ldots, K \), and using (3.8) as the ML estimator of \( \Omega_{ij} \). Working with the concentrated log-likelihood for \( L^*(\theta) \) gives a simple analytic expression for the ML estimator of \( \Omega \) and reduces the number of parameters which need to be estimated by iterative techniques from \( K + n(n+1)/2 \) to \( K \), thereby saving on computational costs.
3.1 The BHHH Method of Estimation and Inference by Maximum Likelihood

Maximum likelihood estimates are assumed to be generally statistically efficient in large samples. Berndt-Hall-Hall-Hausman (BHHH, 1974) have developed a practical approach to maximum likelihood estimation within the framework of gradient methods. Their approach has two advantages over the application of Newton's method (Eisenpress and Greenstadt (1966), Chow (1973)). First, its convergence is more likely since unlike Newton's method which uses a Hessian matrix that may not be positive definite, it confines the direction vector to the gradient halfspace. Second, the BHHH method requires the evaluation of derivatives of the functions $f_{it}$ up to second only, while Newton's method requires certain third derivatives of functions $f_{it}$.

We need to maximise $L^*(0)$ the concentrated log-likelihood function defined in equation (3.11).

Differentiating (3.11) w.r.t. $\theta$, the gradient of the log-likelihood function is:

$$\frac{\partial L^*}{\partial \theta} = \sum_t \frac{\partial}{\partial \theta} \log |\det J_t| - \frac{T}{2} \frac{\partial^2}{\partial \theta \partial \theta} \log \left[ \det \left( \frac{\sum_t f_{it}^2}{T} \right) \right]$$

or

$$\sum_{t=1}^T \frac{\partial L^*}{\partial \theta} = \sum_{t=1}^T \left( p_t - q_t \right).$$

$$\sum_{t=1}^T \frac{\partial L^*}{\partial \theta} = \sum_{t=1}^T \left( p_t - q_t \right).$$
The variance-covariance matrix of the gradient is given by:

\[ E \left[ \left( \frac{\partial L^*}{\partial \theta} \right) \left( \frac{\partial L^*}{\partial \theta} \right)' \right] = E \left[ (p - q) (p - q)' \right]. \]  \hspace{1cm} (3.13)

Define

\[ p_{kt} = \frac{3}{\theta_k} \log |\det J_t| \]
\[ \quad = \sum (J_t)_{i,j}^{-1} \frac{\partial J_t}_{t,i,j} \frac{\partial \theta_k}{\theta_k} \]  \hspace{1cm} (3.14)

where \( \frac{\partial^2 f}{\partial \theta_k \partial \theta_j} \) is the square matrix with typical element \( \frac{\partial^2 f}{\partial \theta_k \partial \theta_j} \), \( i = 1, \ldots, n \), \( j = 1, \ldots, n \), and

\[ q_{ti} = \left( \frac{\partial^2 f}{\partial \theta_t} \left[ \sum_{t=1}^{T} \frac{f_t}{\partial \theta_t} \right]^{-1} \right)_{t=1}^{T} f_t. \]  \hspace{1cm} (3.15)

Let \( Q_t = \frac{\partial^2 L^*}{\partial \theta \partial \theta} \) and

\[ Q = \sum_{t=1}^{T} Q_t, \quad \bar{Q} = T^{-1} Q. \]

Then

\[ \bar{Q}_0 = \frac{1}{T} \left( \frac{\partial^2 L^*}{\partial \theta \partial \theta} \right)_{\theta = \bar{\theta}} \]

where \( \bar{\theta} \) is the true value of \( \theta \), has the property

\[ E \left( \bar{Q}_0 \right) = -E \left( \frac{1}{T} \left( \frac{\partial L^*}{\partial \theta} \right) \left( \frac{\partial L^*}{\partial \theta} \right)' \right)_{\theta = \bar{\theta}} \].
\( E(\tilde{Q}_g) \) is the information matrix which indicates the amount of information from the data on the parameters which we are estimating. Also the inverse of the information matrix is the variance of the estimator. The proof is omitted since it involves detailed manipulation of the derivatives of the concentrated log-likelihood function, and the use of the identity for the information matrix (see, e.g., Kendall and Stuart (1961, Chapter 18) and Theil (1973, Section 8.4)).

Let

\[ H = E(\tilde{Q}_g) \]

and define

\[ R = \sum_{t=1}^{T} (p_t - q_t)(p_t - q_t)' \] (3.16)

Then \( \text{plim} \frac{1}{T} R = \lim_{T \to \infty} E \left( \frac{1}{T} (p - q)(p - q)' \right) \),

thus \( R \) provides a consistent positive definite estimator of \( -H \), which can be used in a quasi-Newton optimisation algorithm if and only if \( u_t \) is independently identically normally distributed.

The basic algorithm is:

\[ \sum_{t=1}^{T} (p_t - q_t)(p_t - q_t)' \Delta \theta^{(r)} = \lambda^{(r)} (p - q) = 0 \]

that is

\[ \theta^{(r+1)} = \theta^{(r)} + \lambda^{(r)} \left[ R^{-1} (p - q) \right|_{\theta^{(r)}} \]

\[ = \theta^{(r)} + \lambda^{(r)} d^{(r)} \] (3.17)

where \( \lambda^{(r)} \) is chosen to \( \max_{\lambda} L^* \left( \theta^{(r)} + \lambda d^{(r)} (\theta^{(r)}) \right) \) by a line search.
R must be positive definite, but there is a possibility that R may approach a singular matrix as the process iterates. Thus, we need to restrict R.

We let \( \alpha \) be a prescribed positive constant less than one. Then at each iteration, we require

\[
\rho = \frac{d'g}{d'd} > \alpha, \quad 0 < \alpha < 1 \tag{3.18}
\]

where \( g = p - q \),

this will ensure the algorithm moving downhill. If \( \rho \) drops below \( \alpha \) on a particular iteration, we should replace \( R \) by a matrix with larger diagonal elements.

All gradient methods require a line search to determine the scalar \( \lambda \) after calculating the direction, \( d \). Given that \( \lambda \) is chosen by such a line search algorithm (Section 2.4), together with the restriction on \( R \), and given that \( L^*(\theta) \) is twice continuously differentiable, we can now state the convergence theorem.

Consider the sequence

\[
\theta(1), \theta(2), \ldots
\]

where

\[
\theta(r+1) = \theta(r) + \lambda(r)d(r)
\]

and

\[
d(r) = (R(r))^{-1}g(r)
\]
If $R^{(r)}$ obeys the restriction (3.18) and $\lambda^{(r)}$ is chosen to satisfy
$L^*(\theta^{(r)} + \lambda^{(r)}d(r)) > L^*(\theta^{(r)})$, then $\lim_{r \to \infty} g^{(r)} = 0$.

Not every critical point of $L^*(\theta)$ is a local maximum since saddle points can occur. If the iterative process chooses a value of $\theta$ where $L^*(\theta)$ has a saddle point, the iterative process will stall, as $g = 0$ at such points. However it is more likely that the process will find a local maximum which is not a global maximum. To safeguard against the possibility of accepting convergence to a local maximum that is not a global maximum, we choose several initial values of $\theta$. If they do not all lead to convergence to the same point, then we might investigate the actual shape of the function with more care until the global maximum is located.

3.2 The BHHH Algorithm

The basic iteration is:

Step 0: For each $i, j, k$ and $t$; $i, j = 1, \ldots, n$
$k = 1, \ldots, K$
$t = 1, \ldots, T$

Compute: $f_{it}' \quad J_{ijt}$

$$S = (S_{ij}) = \left[ \sum_{t}^{f_{it}' f_{jt}} \right]$$

$$S^{-1} = (S^{-1}_{ij})$$

$$y_{it} = \sum_{j} S^{-1}_{ij} f_{jt}$$

$$H_{ijt} = (J_{ijt})^{-1}$$
Step 1:

Compute: \( \frac{\partial f_i}{\partial \theta_k}, \frac{\partial J_{ij}}{\partial \theta_k} \) for each \( t \)

\[
p_{kt} = \sum_{ij} H_{ijt} \left( \frac{\partial J_{ij}}{\partial \theta_k} \right)_t
\]

\[
q_{kt} = \sum_i \left( \frac{\partial f_{it}}{\partial \theta_k} \right) y_{it}
\]

\[
\nu_{kt} = p_{kt} - q_{kt}
\]

Repeat Steps 0 and 1 for all \( t \) and \( k \).

Step 2:

Compute: \( g = \sum_{t=1}^{T} \nu_{kt} \), \( k = 1, \ldots, K \)

\[
R_{ij} = \sum_{t=1}^{T} \nu_{it}' \nu_{jt}' \quad i, j = 1, \ldots, n
\]

Step 3: Compute new direction

\[
d^{(r)}(x) = (R_{ij})^{-1} g
\]

Step 4: Check for convergence:

\[
(i) \quad \max_i \frac{|d_{i}^{(r)}|}{\max \{1, |\theta^{(r)}|\}} \leq \epsilon_1
\]
(ii) \( ||g||_2 \leq \epsilon_2 \)

(iii) \( \frac{|\theta_i^{(r)} - \theta_i^{(r-1)}|}{\theta_i^{(r-1)}} \leq \epsilon_3 \), for all \( i \)

(iv) \( \frac{|L^*(\theta^{(r)}) - L^*(\theta^{(r-1)})|}{L^*(\theta^{(r)})} \leq \epsilon_4 \)

If any of the conditions is satisfied, go to Step 6.

Step 5:

(a) Search for \( \lambda^{(r)} \) using a line search procedure to ensure that

\[
L^*(\theta^{(r)} + \lambda^{(r)}d^{(r)}) > L^*(\theta^{(r)})
\]

(b) Update \( \theta^{(r)} \) by setting

\[
\theta^{(r+1)} = \theta^{(r)} + \lambda^{(r)}d^{(r)}
\]

(c) Return to Step 0.

Step 6: If convergence is achieved, report parameter estimates \( \hat{\theta} \) and its estimated variance-covariance matrix \( (R_{ij})^{-1} \).

The BHHH algorithm is implemented in the computer programs described in Chapter 5.
CHAPTER 4

A METHOD OF SPECIFICATION, DIFFERENTIATION AND COMPUTATION
FOR SETS OF GENERAL FUNCTIONS

4. A Differentiation Program

To estimate non-linear simultaneous equations systems by the method of maximum likelihood, it is necessary to compute the gradient of the log-likelihood function either analytically or by the use of numerical approximation. It was decided to implement the BHHH method and the Gill-Murray-Pitfield algorithm in our estimation program; both methods employ analytical gradient, hence a specific differentiation program is written for such purpose (Sargan and Chong (1980)).

4.1 Organisation of the Differentiation Program

The differentiation program may be divided into three parts:

1. To read in a specification of a set of functions and code it in a form easily translated and implemented in computer memory.

2. To differentiate such a set of functions, and to hold the specifications of the derivatives in computer memory.

3. To calculate the values of the functions and derivatives for given values of the variables.
The organisation of the differentiation program is shown in Figure 4.1.

The set of non-linear functions is first read into the computer memory in FORTRAN IV definitions. The definitions are then encoded into a list of integers representing the appropriate input symbols. A formula processor subroutine is called to process the list of integers into function specifications in a machine internal code and these are then stored permanently. When all input functions have been processed, differentiation can begin. Each of the components of function specifications are considered in turn and applied the appropriate differentiation rule. The resultant derivatives are defined as functions in the same internal code as function specifications. When all functions have been differentiated, an evaluation routine is called to decode the internal code and then compute the functions and derivatives values. The output from the differentiation program prints only the numerical values, not the functions in normal FORTRAN expressions.

4.2 General Considerations and Function Specifications

If we follow the usual FORTRAN conventions as to the definitions of a function with "*" and "/" having priority over "+" and "-", we will find our more complicated expressions having brackets whenever a factor in a product is the sum of a series of terms. Thus we will simplify our computing and differentiating by introducing a NEW function NFUNC corresponding to the contents of the brackets, whenever brackets are used in the definition of the function.
Figure 4.1

Transformed Functions in Internal Code

Function Specification

Consider Definition Functions

Variables

+, -

Increase No. of Factors

Increase No. of Terms

Exponentiation

/, **

Define New Function (i.e. Function of a Function)

{,}

Define Special Function

LOG, EXP, SIN, COS, ATAN

Constant

Store Value in Constant List

Differentiation

Over Each Term of Function

Consider Component Functions

Constant

Special Function

Factor

Function of a Function

Exponentiation

Apply Appropriate Rule

Decode Internal Code and Compute Numerical Values
In addition to "**", "/", "+", "-", we wish to introduce "**", "LOG", "EXP", "SIN", "COS" and "ATAN". These will be
called **SPECIAL FUNCTIONS** and denoted by \(F_k\).

We then define a set of "defined functions" as follows:

\[
f_s = \sum_{i=1}^{I} \sum_{j=1}^{J_i} c_i \prod_{i=1}^{I} \phi_{ij}
\]  

(4.1)

where the \(\phi_{ij}\) are described as **FACTORS** and the products of the
factors are defined as **TERMS**. (Of course, more correctly, every \(I, J_i, c_i, \phi_{ij}\) should have a suffix \(s\).)

The \(c_i\) are real constants, and should be chosen from a store
or list of constants created as the definitions are read into memory.

The factors \(\phi_{ij}\) are of three types:

(i) \(\phi_{ij} = (x_k)^p\)

where \(x_k\) is some variable in the set of variables;

(ii) \(\phi_{ij} = (f_k)^p\)

where \(f_k\) is some function previously defined;

(iii) \(\phi_{ij} = (F_k(x_t))^p\) or \((F_k(f_t))^p\).

In all three cases, \(p\) is an exponent (which corresponds to a
"**" in the original input).
The specifications of the functions are held as a set of integers, one after the other, so that space is not wasted. Also the definitions follow one another in the order in which the definitions are read in, not in the order in which they will need to be computed. To compute any function in the correct entry, it would be necessary to have an array of pointers for the order of computation. This array need only list the functions in priority of computations or perhaps contain the addresses of the start of each function specification.

Suppose we discriminate between indices relating to variables by saying that:

(a) integers $\leq 1000$ refer to variables.
(b) integer $i > 1000$ that $i-1000$ is the index of the corresponding function.
(c) However, each factor may be a special function, so we allocate the first five integers (1 to 5) to indicate special functions:

that is, $\log = 1$
$\exp = 2$
$\sin = 3$
$\cos = 4$
$\atan = 5$

Note that we do not allow a special function of a special function unless this is done by using a defined function.
4.3 Sets of Conventions for Defining a Function

(i) Constant Indices

We have set up a set of conventions for constants as follows:

(a) An integer one means there is no constant for the current term or function, that is, \( c_1 = 1.0 \), for example, \( \text{Exp } x_1 \) becomes \( 1.0 \times \text{Exp } x_1 \).

(b) Positive sign means plus, with an INDEX corresponding to the order in a vector of constant CONS. The value of the constant can be found in element \((\text{INDEX}-1)\) of vector CONS. For example, value of \( c_5 \) can be found in CONS \((5-1)\).

(c) Negative sign means minus with the constant held as (b) above.

(d) If the number of factors is zero, then the term only consists of a constant. This constant is defined in the same way as above.

(ii) Specifications of a Function

As we have suggested earlier, we use integers to represent a set of functional symbols. For all integers less than or equal to 1000, they represent variables and any integer \( i > 1000 \) then \( i - 1000 \) is the index of the corresponding function. We have allowed integers 1 to 5 to represent special functions such as LOG, EXP, SIN, COS and ATAN. Thus if \( i < 1000 \), we take \( i - 5 \) to be the index of a variable.
Now we suggest that the actual list of integers required for a given specification be as follows:

1. Number of function (i.e. defining order in list of functions).

2. Number of terms in function. In term of formula 4.1, this is I.

Then for each term we need to define:

(2a) Number of factors in the current term (which is $J_i$ for each $i$).

(2b) Index of constant at the start of each term (i.e. index of $c_i$ and the constants are stored in a list).

(2c) For each factor, an integer defining whether it is a variable, a special function, or a defined function.

(2d) For each special function a following index saying whether the function depends on a variable or a defined function.

(2e) A negative integer which occurs only if an exponent is used.

(iii) Function of a Function

A function can refer to another function with a separate definition provided that whenever this happens, we can arrange that the second function is computed earlier. There will be incorrect results if it is not possible to arrange a consistent order of computation.

(a) Functions with Bracket-Contents

We treat any bracket-contents of a function as a new function (NFUNC). When an open bracket is encountered on read in, a new function is immediately defined and the specification of this new function is stored in the definition list. The advantage of doing this is that it enables a function specification including brackets to be interpreted
consecutively without providing temporary storage for bracket-contents
definition, and has the advantage that in differentiation and
numerical evaluation the contents of the bracket are differentiated
or computed only once, rather than computed each time it occurs, in
the definition of the various order derivatives. A lower priority
index is allocated to this new function in the order of computation,
that is, a new function is placed in the front of the priority queue.
We choose to use this convention so as to allow definitions of
original functions to be non-consecutive (this corresponds to an
unconditional jump in an ordinary program). We need to jump over
the new function specification list and continue from there as we
compute or differentiate the original function. As an example we
have the sequence,

1010, -16, 477

which would mean function $f_{10}$ then jumps to address 477 in the
specification list and continues from address 477.

We use the integer -16 to mean this UNCONDITIONAL jump whenever
a bracket-contents is defined as a new function and the following
integer gives the address of the NEXT integer in the specification.

On reading in a specification, when brackets have been encountered,
$f_{10}$ would be the contents of the brackets, and in fact the speci-
ification $f_{10}$ would immediately follow in the specification list.
In implementation, it would be arranged that $f_{10}$ was computed and
differentiated first. The "unconditional jump" would be used, so
that in then computing the original function the program could jump
to the next factor or term.
We also note that the advantage of having the number of terms, and factors of each term specified, is that we can use a simple "DO" loop on implementation.

(b) Treatment of NFUNC

We need to have markers for the new functions that we defined as bracket-contents as we go along. But also the definitions are going to define a set of functions, which are numbered by the expression

"f_i = ......." where i = function number

as the start of the definition. Note also that we are going to define functions by taking derivatives. Suppose that we assume our largest model is written in the form

\[ u_{it} = f_i(y_t, z_t, \theta) \]

\[ t = 1, \ldots, T \]

and the maximum value of \( n \) is 20, and \( y_t \) is an \( 1 \times n \) vector of endogenous variables (also of maximum dimension 20). Suppose also that in setting up the definitions of \( f_i \), we use \( n^* \) intermediate or defined functions \( f_i^* \) (bracket-contents), so that the total functions defined in this way is \( n + n^* \). Suppose that \( n^* < 20 \). It is suggested that we should permanently store the definition of

\[ f_i \]

\[ i = 1, \ldots, n \]

\[ f_i^* \]

\[ i = 1, \ldots, n^* \]

and also the definitions of
Now if we wish to compute its derivatives with respect to $\theta$ (suppose $\theta$ was of dimension $m \leq 50$), we need to compute

\[ \frac{\partial f_i}{\partial y_j} \quad i = 1, \ldots, n \]
\[ \frac{\partial f^*_i}{\partial y_j} \quad j = 1, \ldots, n^* \]

Thus at each iteration (of the optimisation program), we need to compute $(n + n^*)(n + 1)(m + 1)$ functions. On this basis, if we store all functional specifications, we would have up to $40 \times 21 \times 51 = 40,000$ specifications, and if we allow up to 20 indices per specification, this means that the specification list (NLIST) would require 800,000 words which seems to be very unreasonable. So it is suggested that we should modify this.

Suppose we store the specification of $f^*_i$, $f^*_i / \partial y_j$, $\partial f^*_i / \partial \alpha_k$, permanently, and then store $\partial f_i / \partial y_j$, $\partial f_i / \partial \alpha_k$, $\partial^2 f_i / \partial y_j \partial \alpha_k$, $\partial^2 f^*_i / \partial y_j \partial \alpha_k$ only temporarily, so for each $k$, we work out the specifications of these derivatives, then compute their numerical values for all $t$. Then move on to $(k+1)$ writing the specifications for the derivatives with respect to $\theta_{k+1}$ over those for $\theta_k$. 
In this way it is only necessary to store the specifications for $2(n+n^*)(l+n)$ functions at the time, that is 1680 specifications (say up to 30,000 words). We might reduce this to 25,000 words on the grounds that not all $n, n^*$ and the number of words per specification would take their maxima in any one model.

Now what this means directly for NFUNC is that we reserve the function values 1 to $n$ (where $n$ is the number of equations in the model) for the explicitly defined functions, and then start the implicitly defined functions (the bracket-contents) consecutively from $n+1$ to $2(n+n^*)(n+1)$. Each time we define a new function either as the contents of the bracket, or as a derivative of one existing function, we increase NFUNC by one. At the end of the differentiation w.r.t. $y_\ell$, NFUNC should have reached $n+n^*+n(n+n^*)$, and at the end of that round of differentiations we take note of the corresponding NFUNC, that is set MAXF = NFUNC. Then after differentiating w.r.t. $\theta_k$ and computing the resulting derivatives, we reset NFUNC = MAXF, the address for storing derivative specifications back to the address corresponding to function NFUNC, and we then write the new specifications over the old specifications.

(c) Priority Ordering

The most difficult aspect of this procedure is to ensure that functions are defined in an appropriate order of computation. We choose the usual convention that the lower index function in a priority list is computed earlier. To compute the function and to obtain the form of derivative functions it is necessary to ensure that we order the computations so that the value of function "B" is computed before function "A" which depends on function "B". Thus if we have an index of computing priority, then we must ensure that function "B" has a lower index than function "A" if "A" depends on "B".
We find it convenient to keep two lists:

(i) NPRIOR (j) shows the order in which \( f_j \) is computed:
NPRIOR (1) = j shows \( f_j \) is computed first, NPRIOR (2) = k shows that \( f_k \) is computed second and so on.
(ii) We also invert this ordering by also listing NPRS (j) which shows for each j, the number of the function which is j in the priority ordering.

Each time a function is defined, it is taken next in the priority ordering. Now each time the definition of a function i refers to function j, it is checked that j occurs first in the ordering by comparing NPRIOR (i) and NPRIOR (j). The test should be passed if j is always defined before i. But if there is a misorder then first j is checked, by considering the NPRIOR for each function occurring in its definition, so that NPRIOR (j) is set at the greatest of these plus one. Then NPRIOR (i) is set at NPRIOR (j) + 1. Note that in order to do this it will be necessary to increase the priorities attached to all functions with priorities between the earlier NPRIOR (i) and NPRIOR (j) by one, but these can be located easily by using NPRS. NPRS must then be reallocated by using NPRS (NPRIOR (i)) = i. This procedure of checking the order of functions should be repeated if any reordering has been done.

(iv) **Exponent**

To introduce "**" operation, we treat "/" as "**-1". It seems worthwhile from the computing point of view to treat separately the cases where the exponent is a small positive or negative integer from other exponents, and to ensure that we do not waste space by adding unnecessary indices to the specification list when generally no exponent is necessary. In particular we need \( x^2, x^3, x^4 \) etc.
to be in a form when the computer can easily recognise that, for example,
\[
\frac{d^2}{dx^2}(x^3) = 6x \text{ with no exponent on the } x, \text{ or } \frac{d}{dx}(x^2) = 2x \text{ and so on.}
\]

Thus it is suggested that the exponent be held immediately following the factor to which it refers as a negative number. Thus in computing the value of the function on differentiating, we test after each factor to see whether the next integer in the specification list is negative. If so we realise there is an exponent, and we can then interpret the index to get the exponent. Care must be taken here as we come to functions defined implicitly by brackets, we need to test whether this negative integer "belongs" to the inner function or the outer function by means of the address of the unconditional jump.

From the computing point of view, there is no point in distinguishing large integers, and it is suggested that we take \( p \) the exponent only if

(i) it is an integer

and

(ii) \(-15 \leq p \leq 10\) this allows, for the fact that if we differentiate five times \( x^{-10} \), we end up with \( x^{-15} \).

It is suggested that if the integer in the specification is \(-j\), and \( j \leq 26 \), then we take \( p = j - 16 \) as the exponent, but if \( j > 26 \), then we take \( p = c_{(j-26)} \).

Note that in the case \( j > 26 \), we would be in fact computing

\[
|x|^p
\]

or

\[
|f|^p
\]

since we would use \( \exp(p \log |x|) \).
An Example

Consider the following sequence of integers:

24, 3, 2, 5, 10, 1036, 3, -3, 1, 10, 6, 1010, 2, 1, 2, 6, 1005

Here

"24" shows that this is \( f_{24} \) specification.

"3" shows that it has three terms.

"2" shows that the first term has two factors.

"5" shows that it starts with constant \( c_4 \).

"10" corresponds to \( x_5 \).

"1036" corresponds to \( f_{36} \).

"3" shows that the second term has three factors.

"-3" shows a minus sign at the start of the term, that is, the term is \(-c_2 \cdot (\cdot)(\cdot)\).

"1" shows special function \( \log \).

"10" corresponds to \( x_5 \).

"6" corresponds to \( x_1 \).

"1010" corresponds to \( f_{10} \).

"2" shows the third term has two factors.

"1" shows that there is no constant for this term.

"2" shows the special function \( \exp \).

"6" corresponds to \( x_1 \).

"1005" corresponds to \( f_5 \).

Thus we can translate this sequence into

\[
f_{24} = c_4 \cdot x_5 \cdot f_{36} - c_2 \cdot \log x_5 \cdot x_1 \cdot f_{10} + \exp x_1 \cdot f_5
\]

For expressions having exponents, we consider the example below:

\[
f_{20} = c_4 \cdot x_7 \cdot f_{10/x_9} + c_6 \cdot f_{11} \cdot 2 \cdot x_1
\]
which is translated as

20, 2, 3, 5, 12, 1010, 14, -15, 2, 7, 1011, -18, 6,

where -15 indicates a "**-1" that is "/" and -18 is an
exponent of **2 (i.e. \( p = -j-16 \)). If

\[ f_{10} = x_j ** c_6, \]

then the sequence of integers would be

10, 1, 1, 1, 12, -32,

where -32 corresponds to \( c_{-j-26} \).

Note that when the exponent is not an integer, we are in fact
taking the modulus of the variable, function or defined function.

(v) **Reading in a Specification**

The difference between our computer specification and reading
a general FORTRAN function is that we allowed no brackets (except
for the brackets that are always used in connection with special
functions). All that is needed is to read in a more complicated
function with brackets to define a new function when brackets occur.
Thus, as soon as opening brackets "(" occur on the read in, a
new function NFUNC is defined, and a jump (-16) is inserted over
the definition of this function.
As an example, we have

(a) \[ f_{16} = c_{10} + c_{11} \times (c_{12} + c_{13} \times x_{10}^2) \times \exp (c_{14} \times f_{12}) \]

Then this would be translated as

\[ f_{16} = c_{10} + c_{11} \times f_{105} \times \exp f_{106} \]

with

\[ f_{105} = c_{12} + c_{13} \times x_{10}^2 \]

\[ f_{106} = c_{14} \times f_{12} \]

or another example

(b) \[ f_{16} = \exp (c_{17} + c_{18} \times (c_{19} + f_{12})^2) \]

then

\[ f_{16} = \exp f_{21} \]

\[ f_{21} = c_{17} + c_{18} \times f_{22}^2 \]

and

\[ f_{22} = c_{19} + f_{12} \]

Note that when a bracket has been opened, when the close bracket ")" is reached it would be checked whether the name of a variable or function or special function following variable or function is all that is in the bracket, if so the bracket is ignored (unless two consecutive special functions would be created by removing the bracket).
Example (b) is the one where brackets are nested. There is no problem with this provided enough information is stored to resume after the bracket.

The most important information is the address of the jump address, so that when the close bracket ")" is reached, the address of the next instruction entry can be found and stored after the integer -16 in the specification list.

Thus in reading, we need an index say NDEPTH of the level of nested brackets:

\[
NDEPTH = 0 \quad \text{meaning we have no brackets} \\
= 1 \quad \text{we have a function arising from a bracket} \\
= 2 \quad \text{we have a function arising from 2 nested brackets.} \\
\vdots \\
\vdots 
\]

Then we need an array JUMPAD(I,J) for the jump address connected with brackets of depth J with function I.

The important points to be remembered are:

(a) the number of terms that have already been read in the functions outside the bracket;
(b) the number of factors that have already been read in the current term;
(c) whether the previous operator was "/" (meaning that it will be necessary to insert **-1 later as exponent).

Thus we need three temporary lists to keep the intermediate values of (a) - (c) generated on reading in the function.

As a function is read in, two indices are held: NTERM and NFACT to indicate the number of terms and number of factors.
respectively. NTERM starts as 1 and is increased by 1 after reading "+" or "-". NFACT is stored in the appropriate address of the specification list when "+" or "-" is read, it is then set to zero, and increased by one whenever "*" or "/" is read.

It is necessary to remember which function is being defined outside the bracket). We use NFUNC as the number of functions currently being processed and increase NFUNC by one for each bracket J. Similarly we need to remember the address in the specification list of the number of factors in the current term, this is the address where NFACT will be stored at the next "+" or "-". Note that each time a closing bracket is found, NDEPTH is decreased by one, the definition of the function in the bracket is completed, and the definition of the lower level function is resumed.

(vi) Characters for Input

The following is a list of characters which can only be used to construct a set of equations:

<table>
<thead>
<tr>
<th>List</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>V</td>
<td>F</td>
<td>*</td>
<td>/</td>
<td>+</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>List</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>Character</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>*</td>
<td>)</td>
<td>=</td>
<td>L</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>List</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>Character</td>
<td>C</td>
<td>A</td>
<td>N</td>
<td>G</td>
<td>X</td>
<td>P</td>
<td>I</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>(</td>
</tr>
</tbody>
</table>

To input any equation, users may construct any functional form of the equation from these 33 characters.
Some Remarks

1. A variable is represented by the character V, for example,
   \[ V_{10} \]

2. Equation name is represented by F, for example,
   \[ F_1 \text{ is equation 1} \]
   \[ F_2 \text{ is equation 2} \]

3. Operators are +, -, *, /, **.

4. Digits are 0 to 9.

5. Decimal is ".".

6. Brackets are used to define intermediate functions, for example,
   \[ V_{10}/((V_1 * V_2 + V_3) * \text{Exp} V_6) \]

7. "=" enables an expression to be read more easily, for example,
   \[ F_2 = V_1 * V_2 + 1/2 * V_3 \]

8. Characters are
   These enable us to set up special functions such as LOG, EXP, SIN, COS, ATAN. Also these may be used to construct the key word "NEXT" for the continuation of an equation on a second card.

4.4 Differentiation

When differentiating it is assumed that we wish to store a similar specification for the derivative. This may not be necessary if we require the derivative to be calculated for various values of the variables. In this case it is assumed that the specification of an appropriate set of derivatives is stored temporarily in public storage, and a calculation subroutine then interprets these specifications to get the numerical values for given \( X \) (the data matrix).
In cases where it is necessary to calculate second or higher order derivatives, it will be necessary to be able to trace which specification gives the first derivative of which function w.r.t. which variable. Thus, we prepare a one-dimensional array NADR(I) which stores addresses at the start of function I and a two-dimensional array NDER(I, J) which stores the function NFUNC of the derivative of function I w.r.t. variable J.

Thus to find the address of \( \frac{\partial^2 f_i}{\partial x_j \partial x_k} \) we would first locate NFUNC of \( \frac{\partial f_i}{\partial x_j} \) in array NDER(I, J), then look up NFUNC of NDER(I, J) differentiated w.r.t. variable \( x_k \), and the corresponding new NFUNC and its address will lead to the derivative function \( \frac{\partial^2 f_i}{\partial x_j \partial x_k} \).

The specification of derivative is based on the formula:

\[
\frac{\partial \phi}{\partial x_s} = \sum_i \sum_k c_{ij} \prod_{j \neq k} (P_j)^{P_{ij}} \frac{\partial f_k}{\partial a} \cdot \frac{\partial f_a}{\partial x_s} P_k^{P_k} (P_k-1)
\]  

(4.2)

for a function of the form:

\[
\phi = \sum_i c_{ij} \prod_k \left[ P_{k(f)} \right]^{P_k}
\]

Note that the total number of terms may be equal to the total number of factors. (It will be less whenever a variable occurs rather than a defined function.)

In the most general case, in each term of the derivative \( (J_i - 1) \) of the factors are the same as \( \sum(J_i - 1) \) factors of the corresponding term of the function. The simplest treatment is to copy the specification of the term until we reach factor \( k \), and then insert the terms that correspond to the derivatives. We then copy the remaining factors.
in the original term. Note that if the factor does not contain a special function we can omit the factor \( \frac{3F}{k/3f} \). If a special function is a function of a variable then we check whether this is the variable with respect to which we are differentiating. If not we omit this term from the derivative (by not increasing our running count of the number of terms in the derivatives and increase \( k \) by one immediately, that is, go on to the next value of \( k \)). If the variables are the same, then we omit this factor from the term, that is, set \( \frac{3f_a}{3x_s} = 1 \). Finally if there is no exponent, then we can omit the last factor of equation (4.2) from the derivative function.

\( \frac{3f_a}{3x_s} \) is a defined function, \( \frac{3f}{3f} \) is another special function of \( f \), except in two cases:

(i) for \( \log f \), we insert \( \frac{1}{f} \), that is, we have a factor \( f \) followed by the exponent "**-1":

(ii) for \( \tan^{-1} f \), we need to insert \( \frac{1}{1+f^2} \). The way to do this is to define a new function \( 1+f^2 \), while ensuring that \( 1+f^2 \) is computed before the derivative function.

As each term in the derivative function is generated, the computer must remember the address where the number of factors is stored. Then as it looks at each factor it keeps a count of the number of factors so that at the end of the specification of that term it can insert the number of factors into the right place in the specification list. Similarly it keeps a running count of the number of terms in the derivatives, which at the end of generating the total specification can be inserted into the second place in the specification list of that function. Notice that using these rules a term which is linear in a variable, may end up with a constant, and there is no need to take special heed, provided that we agree that a constant can occur as a term in any order in a function.
4.5 **Simplification**

Ultimately it must be stressed that it is worthwhile simplifying the functions and derivatives to avoid repetitive calculation.

(i) **Eliminating Surplus Functions**

As a new function is defined either on reaching outside specifications or by differentiation, the definitions of the existing function are scanned, and if an existing function is found to have exactly the same definition, then the new function is cancelled.

(ii) **Cancelling Repeated Factors in Differentiating**

In differentiating a special function factor such as EXP, LOG, SIN, COS, or ATAN; in each case there is the possibility that one of the other factors in this term is the same as the derivative of this factor. So each time we differentiate a factor of this type, we run through all the other factors in the term and if they are the same as the derivative factor, we increase the exponent of this factor by one.

(iii) **Replacement of Simple Functions**

The basic idea is that in the case where the derivative of a function is a single term, and where that function is used as a factor in defining a second function (as the contents of a bracket), then in differentiating the second function, we replace the derivative of the factor by its definition as a product of factors.

4.6 **Numerical Evaluation of Function and Derivative Values**

To evaluate function values, we need to compute all functions with lower indices in the priority list first. Intermediate functions
such as bracket-contents and derivative functions generated by the process of differentiation can be picked up easily from the specification lists as the indices of the head address of all function specifications are stored in an array of pointers.

To calculate the function and derivative values for a set of given values, \( X \), the computer interprets each function specification as a mathematical expression and then evaluates the function value. All function specifications must have their priorities checked before any function evaluation to get the right order of computation.

Intermediate function values are held in a temporary list pointed by the function number \( \text{NFUNC} \). When an original function depends on the intermediate function \( \text{NFUNC} \), it only needs to compute the original function value, then pick up the numerical value of \( \text{NFUNC} \) from temporary storage and then update the original function value with the value of \( \text{NFUNC} \). Hence it does not have to recompute the same \( \text{NFUNC} \) value whenever any function refers to it.

4.7 Example

Consider the following example with two equations

(a) \((\log (x_1 + x_2 * x_3))^2 = f_1\)
(b) \(x_1 + x_2 * \log (x_1 + x_3) = f_2\)

If we call equation (a) \( f_1 \) and equation (b) \( f_2 \), then on reading in the first equation, the program will translate equation (a) as follows:
\[ f_1 = f_{100} ** 2 \quad \text{(we take 100 as an arbitrary number)} \]

\[ f_{100} = \log f_{101} \]

\[ f_{101} = x_1 + x_2 * x_3 \]

where \( f_{100} \) is the outer bracket-contents and \( f_{101} \) is the inner bracket-contents. \( f_{100} \) and \( f_{101} \) are defined as intermediate functions.

Since \( f_1 \) depends on \( f_{100} \) and \( f_{100} \) depends on \( f_{101} \), the priority ordering of function \( f_1 \) in the priority list would be:

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Where \( f_{101} \) has a lower index than \( f_{100} \) and \( f_1 \).

Therefore we need to compute \( f_{101}' \) then \( f_{100} \) and finally \( f_1 \) in this ordering.

For the treatment of a new function number \( \text{NFUNC} \), whenever a new function such as bracket-contents is defined implicitly, we increase the value of \( \text{NFUNC} \) by one. Also derivative functions are considered as new functions and hence \( \text{NFUNC} \) would have to be increased by one each time when a new function is generated by the differentiation.

In equation (a)

\[ \text{NFUNC} = 100 \quad \text{means} \quad \log f_{101} \]

\[ \text{NFUNC} = 101 \quad \text{means} \quad x_1 + x_2 * x_3 \]
Note that if there is only one structural equation in this example, then \texttt{NFUNC} = 102 would be the derivative function differentiated w.r.t. any of the variables. Since we have two structural equations here, we need to let the following consecutive \texttt{NFUNC} value denote the bracket-contents of the second equation unless all the following structural equations are independent (i.e., non-nested) functions.

For example, in equation (b), we have

\[
\begin{align*}
  f_2 &= x_1 + x_2 \times \log f_{102} \\
  f_{102} &= x_1 + x_3
\end{align*}
\]

Now $f_{102}$ has a lower index in priority list than $f_2$ which means $f_{102}$ must be computed or differentiated before $f_2$.

The priority list for $f_1$ and $f_2$ would be

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

that is, \texttt{NPRIOR (1)} = 101
\texttt{NPRIOR (2)} = 100
\texttt{NPRIOR (3)} = 1
\texttt{NPRIOR (4)} = 102
\texttt{NPRIOR (5)} = 2
and the reverse ordering of all functions would be

\[
\begin{align*}
\text{NPRS (101) } &= 1 \\
\text{NPRS (100) } &= 2 \\
\text{NPRS (1) } &= 3 \\
\text{NPRS (102) } &= 4 \\
\text{NPRS (2) } &= 5
\end{align*}
\]

that is, function 101 is the first function to be differentiated and computed, then function 100 and so on.

Now for the derivative function \( \text{NFUNC} \) (from 103 onwards) would be increased by one each time when a new derivative function is generated, therefore each value of \( \text{NFUNC} \) would represent an equation for the derivative function.

To compute function values, we have to pick up \( \text{NFUNC} \) from the priority list to get the correct order of function to be computed. But to compute the derivative values, we compute each derivative function \( \text{NFUNC} \) by setting

\[
\text{NFUNC} = \text{NDER (I,J)}
\]

where \( \text{NDER} \) is a two-dimensional array to store the derivative of function \( I \) differentiated w.r.t. variable \( J \). Again \( \text{NFUNC} \) must have its priority checked before any numerical evaluation.
5. An Estimation Program

In this chapter, we describe a computer program called NLMLE (Non-Linear Maximum Likelihood Estimation) which estimates a small to medium size non-linear econometric model by the method of maximum likelihood.

The numerical techniques applied to the program are:

(i) The BHHH method.

(ii) The variable-metric method of Gill-Murray-Pitfield.

Both methods employ analytical derivatives for the computation of the gradient of the concentrated log-likelihood function.

Generally, non-linear econometric models are either non-linear in parameters, variables or both. When non-linear in the variables \( y_t \), that is, when \( J_t = \frac{\partial^2 f_t}{\partial y_t} \) varies over \( t \), there will be substantial computation to calculate \( \frac{\partial f_t}{\partial y_t} \).

NLMLE is designed to tackle this kind of highly non-linear model with complex econometric functions. It enables users to define the set of simultaneous equations in functional form. The equations are input to the computer together with the attached data, the choice of optimisation technique and line search procedure, a tolerance level for the accuracy of the estimates and a maximum number of iterations for the model to run.
The output from the program comprises the computed parameter estimates, their standard errors and the T-ratios, the residual sum of squares matrix and the asymptotic variance-covariance matrix of the parameter estimates.

5.1 Organisation of the Estimation Program

The estimation program is divided into three major parts:

(i) The differentiation program.

(ii) The optimisation procedures and line search.

(iii) Some supporting routines for a convergence test, the initialisation of a new step-size for each iteration and for calculation of output statistics.

Figure 5.1 illustrates the flow of the estimation program.

After input, the differentiation program is loaded to differentiate the set of equations. The analytic gradient and Hessian matrix are then set up. (The Hessian matrix depends on the choice of the optimisation procedure.) The optimisation method then maximises the likelihood function of the set of equations.

If convergence has been achieved, the procedure terminates and the supporting routines will print out relevant statistics. If convergence has not been achieved, then the program updates the current value of parameter estimates with the new step-size calculated from a line search procedure. It then repeats the process until it has satisfied the convergence criterion or it has reached the maximum number of iterations.
Figure 5.1
5.2 Functional Definitions

We define the set of equations according to the function specifications described in Chapter 4. It is important that all equations must be specified according to the conventions set up in the previous chapter.

Usually, users need not be concerned with the internal workings of the differentiation program, but some knowledge of the representation of expressions and the way they are defined should be acquired in order to use NLMLE more easily, efficiently, and effectively.

We differentiate the set of equations in the following steps:

(i) differentiate with respect to the endogenous variables $y_t$ to get the Jacobian, that is $\frac{\partial f}{\partial y_t}$;

(ii) differentiate with respect to the parameter $\theta_k$, that is $\frac{\partial f}{\partial \theta_k}$;

(iii) differentiate the Jacobian with respect to parameter $\theta_k$, that is $\frac{\partial^2 f}{\partial \theta_k \partial y_t}$.

Repeat (ii) and (iii) for all $k$, $k=1, \ldots, K$.

Following the functions being differentiated, an evaluation routine is called to evaluate the numerical values of the derivative functions. When the evaluator sees a factor, it checks to see whether the factor has a value assigned to it, if there is a value, it updates the function value. If it sees a function as an argument, again it checks the function and updates the current function value. If there is a special function attached to the factor or function, it applies the special function to the evaluated argument. If it is division
or exponentiation, the same treatment is applied as with the special function.

The evaluator scans each function term by term and factor by factor. It returns the final value of each function.

A packing routine is then loaded to pick up all the derivative values and put them in a compact form as the gradient of the concentrated log-likelihood function, that is,

\[
g = \frac{\partial \log L^*(\theta)}{\partial \theta} = \sum_i \sum_j \sum_t \left( J_{ij}^{-1} \frac{\partial J_{ij}^{t+1}}{\partial \theta} \right) - \frac{T}{2} \sum_t \left( \frac{f_t f_t^T}{\partial \theta} \right) \left( \sum_t \frac{f_t f_t^T}{T} \right)^{-1} f_t
\]

(5.1)

where \( (J_t) \), \( \frac{\partial J_{ij}^{t+1}}{\partial \theta} \) and \( \sum_t \frac{f_t f_t^T}{T} \) are all \( n \times n \) matrices.

5.3 Estimation Methods

A gradient method and a variable metric method by Gill-Murray-Pitfield are provided. The two methods offer different choices for the Hessian matrix. The first (METHOD = BHHH) is that described in Chapter 3. The second method offered (METHOD = GMP with analytical derivatives) differs from the first only in the calculation of the Hessian matrix. For this method, the updating rule is described in Chapter 2.
The BHHH routine is specially programmed and implemented in NLMLE. For GMP, NLMLE uses a routine from the NAG library for the optimization.

5.4 Program Composition

The program is written in FORTRAN IV and was developed on both CDC 7600 and ICL 2980 computers. The DAP versions applying parallel processing will be described in Chapter 6.

The two serial versions comprise a main section and 18 subroutines.

- **FIMLX** the main section.
- **INPUT** reads in data decks and sets up any lags required.
- **DATALT** allows a variety of data transformations to be performed.
- **RDCARD** reads in equations.
- **FRML** formula processor to process the input equations into machine internal code.
- **NUMBER** reads in constants, variable and function indices and exponentiation of the input equations.
- **DIFF** differentiates equations.
- **BHHH** Brendt-Hall-Hall-Hausman estimation procedure.
- **DIFIML** sets up equations for differentiation.
- **DIEVAL** packing routine for derivative functions.
- **EVAL** evaluation routines for original and derivative functions.
- **GCHECK** gradient check routine using finite differences.
FUNML routine to calculate log-likelihood function.
GSTEP line search.
BARD line search by BARD.
INVERT matrix inversion routine using Gauss-Jordan full pivoting.
PRIOR checks priority ordering of function.
PQEVAL evaluation of gradient.

Input to NLMLE

\[ \text{IMETH} = \begin{cases} 
0 & \text{method} = \text{BHHH} \\
1 & \text{method} = \text{GMP} \ \text{(analytic derivatives)} \\
2 & \text{method} = \text{GMP} \ \text{(finite differences)} 
\end{cases} \]

\[ \text{ISTEP} = \begin{cases} 
0 & \text{linear search} = \text{GSTEP} \\
1 & \text{linear search} = \text{BARD} 
\end{cases} \]

IMAX maximum number of iterations.
X data matrix.
N the number of parameters \( \theta \).
V a \( 1 \times n \) array containing an estimate of the position of the best available initial value \( L^*(\theta) \).
TOLB tolerance level for the termination criterion.
NSQZ number of iterations in linear search.
NB number of stochastic equations in the system.
NINTF number of intermediate functions.
NVAR number of variables \( (y_t \) and \( z_t \)).
NT number of observations.
NL number of lags
NI number of identities
NY number of endogenous variables \( (y_t) \)
NZ number of predetermined variables \( (z_t) \)

The overall input is terminated by four dollar signs, that is, $$ $$.

A user's guide to NLMLE is given in Appendix C.
Output of NLMLE

(a) At each iteration, option to print:

(i) old and new function values, new step-size and number of
function calls for that iteration;
(ii) gradient and gradient norm; weighted-gradient, i.e. \( g' \mathbf{H}^{-1} g \);
(iii) direction vector \( d^{(i)} \);
(iv) parameter estimates \( \hat{\theta}^{(i)} \);

(b) At the end of the iterative procedure, information regarding:

(i) whether the program converged and the number of iterations
used;
(ii) maximum number of function calls;
(iii) the log-likelihood function value;
(iv) the final parameter estimates;
(v) the estimated asymptotic variance-covariance matrix of the
parameters \( \hat{\theta} \);
(vi) the standard errors and T-ratios;
(vii) the residual sum of squares matrix.

Limitations

The following apply to the CDC 7600 version of the program:

(i) A maximum of 20 equations.
(ii) A maximum of 30 parameters.
(iii) A maximum of 100 observations.
(iv) A maximum of 50 variables.

The version on the ICL 2980 can estimate a larger model of up
to 100 parameters and \( \gg 100 \) observations.
The three parallel DAP versions are restricted to five-equation models with a maximum of 20 parameters and 30 variables but up to 4096 observations.

The program listing is given in Appendix B, and an example of the output in Appendix D.
6.1 Potential Role of the Distributed Array Processor for Research in Economics

The ICL Distributed Array Processor (DAP) is a 64 x 64 array of microprocessors embedded in the store of a host computer, each processor being associated with 4K bits of semi-conductor store (which can be accessed by the host if the DAP is not in use). Clearly, taking the processor to the data (rather than vice-versa) avoids the time usually required to route information.

Conventional computers use sequential operating procedures and upper limits exist for the speed of calculation possible using such an approach. Miniaturisation and micro-circuits are part of an effort to resolve such problems.

However, a DAP presents a radically different potential solution, using parallel computation; moreover, while the DAP is operating, the host is free to carry out other tasks. Thus to add two 64 x 64 matrices on a 64 x 64 DAP takes the same time as adding two scalars and any task which can be tackled in 64 parallel streams on a 64 cell array takes $64^{-1}$ of the time for 64 sequential operations.

Users of the DAP will have to learn new ways of conceptualising their objectives. The main idea can be seen by considering the multiplication of two $N \times N$ matrices $A$ and $B$. In conventional FORTRAN the algorithm would be a programmed version of:
\( C = A B \Rightarrow C_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \)

which can be rewritten as the inner product

\[ C_{ij} = a'_{i} b_{j}. \]

where \( A' = (a'_{1}, \ldots, a'_{N}), B = (b'_{1}, \ldots, b'_{N}) \).

In the DAP, parallel computation would exploit the outer product from:

\[ C = \sum_{k=1}^{N} a_{k} b'_{k} \]

Each micro-processor does \( N \) multiplications in sequence and cumulates the total. Similar reformulations apply to matrix inversion, etc.

Research in Economics is invariably multivariate and hence is intensive in the use of matrix operations. For example, econometric estimation usually entails maximising a scalar function of matrices, with prolific use of inversion and multiplication of large dimensional matrices. Similarly, recent advances in computing economic equilibria require massive array calculations which consume a considerable amount of cpu time. Monte-Carlo simulation is intensively used by econometricians both to study the properties of econometric estimators and to model the behaviour of economic systems with a large number of participants. Finally, investigating the finite sample distributions of procedures for system estimation does create major demands for time on available computer systems.
Large-scale econometric models are generally non-linear in both the variables used in estimation and the parameters of the likelihood function. Few systems of this kind have been appropriately estimated because the computational time required is very large. With the introduction of array processors that are capable of executing a large number of instructions simultaneously, the computational time can be substantially reduced.

In order to estimate the system of equations, $L^*(0)$ (3.11) has to be maximised with respect to $\theta$, which is a formidable task for large values of $n$ and $k$, where $n$ is the number of equations in the system and $k$ is the number of parameters. Also if $T$ is large, it is equally difficult to compute $\sum_{t=1}^{T} \log |J_t|$ due to the excessive amount of cpu time needed.

Many special cases of equation (3.11) have been investigated, and efficient methods for optimising the relevant likelihood function have been extensively programmed, for example, Hendry (1976), Hendry and Srba (1980), Hendry and Tremayne (1976). Many of the numerical optimisation methods are strongly oriented towards implementation on serial computers.

However, the Distributed Array Processor presents the possibility of a different solution. The power of the DAP is based on its high degree of parallel operation; hence a specially designed algorithm is essential so that the DAP can be fully exploited.
6.2 The Distributed Array Processor (DAP)

The basic concept of a parallel processor which can execute the same instruction on many data items has been known for many years. There have been several processors of this type built, most notably, STARAN and ILLIAC IV (Thurber and Wald (1975)). Although the DAP is similar in concept to these machines, it has two important differences:

(i) the number of processing elements (PEs): the DAP has 4096 as opposed to 256 and 64 for the STARAN and ILLIAC IV, respectively;

(ii) the simplicity of the processing elements (PEs): the PEs in the DAP are one bit processors which means that all operations other than bit manipulations are done in software.

6.3 Architecture

The DAP is a 64 x 64 two-dimensional grid of PEs each with 4096 bits of local memory (the fact that there are 4096 PEs and 4096 bits of local memory is only coincidence). Each PE can perform two basic operations: one bit addition and one bit broadcast of data to one of its four neighbouring PEs (Gostick 1979, 1981; Parkinson 1976 (Nov.), 1977 (Nov.), 1980).

For the purpose of computation, we can describe the DAP as consisting of:

(i) 4096 store planes containing 64 x 64 bits.

(ii) The activity plane (A plane) of 64 x 64 bits. The setting of a particular bit in the A plane to 1 (that is, .TRUE.) allows the corresponding PE to perform a given instruction; that is, the A plane acts as a 'MASK' as to whether an instruction is executed in a particular PE or not.
(iii) A $64 \times 64$ array of PEs that are each connected to their four nearest neighbours.

6.4 Programming the DAP

The DAP has two programming languages: APAL and DAPFORTRAN. APAL is a low level assembly language (ICL (1979)). DAPFORTRAN is an extension of standard FORTRAN. The DAP will execute all standard FORTRAN statements except for formatted READ/ WRITE commands (ICL (1981)). The additional facilities of DAPFORTRAN are basically two extra variable modes: vector and matrix together with a generalised indexing syntax to allow efficient use to be made of them. The three modes can be declared as follows:

- `INTEGER SCALAR_INTEGER, SCALAR_INTEGER_ARRAY(100)`
- `INTEGER VECTOR( ), SET_OF_INTEGER_VECTORS( , 10)`
- `INTEGER MATRIX( ), SET_OF_INTEGER_MATRICES( , 15)`

Note that ICL FORTRAN has always regarded the limitation of the length of variable names to 6 characters as being unnecessarily restrictive, and permits up to 32 characters for all names in a program. DAPFORTRAN follows this convention, thus permitting a much more sensible naming of variables, routines, etc., and hence giving more readable programs. Since spaces are not permitted within DAPFORTRAN names, it is desirable to have some alternative method for breaking up long names. For this purpose the underscore character _ may be used (as in 2900 system Control Language). This character is ignored by the compiler.

By the declaration `VECTOR( )` we mean a vector with 64 components and, by `MATRIX( )`, a matrix of $64 \times 64$ components. Also sets of vectors
and matrices can be declared as shown above. Similar declarations can be made for REAL, LOGICAL and CHARACTER variables.

All the normal arithmetic and logical operations are defined element by element for vector and matrix modes. For example, if A, B and C are REAL matrices, then

\[
A = B + C
\]

means that A is the element by element sum of B and C. In standard serial FORTRAN the above statement is:

\[
\begin{align*}
&\text{DO 10 I = 1, 64} \\
&\text{DO 10 J = 1, 64} \\
&10 \ A(I, J) = B(I, J) + C(I, J)
\end{align*}
\]

Moreover, arithmetic and logical operations can be performed on variables of different modes if there is no ambiguity. For example, if A and B are real matrices and C is a REAL scalar, then

\[
A = B \ast C
\]

means that A is the element by element product of the matrix B and a matrix consisting of 4096 components all with the same value C.

The other significant feature of DAPFORTRAN is masked (or logical) assignment; that is, the assignment of one matrix to another can be 'masked' with a LOGICAL matrix. For example, if A and B are INTEGER matrices and MASK is the LOGICAL matrix defined by
MASK = A .GT. B

(i.e. an element of MASK is .TRUE. when the corresponding element of A is greater than the corresponding element of B).

Then

A(MASK) = B

means that the element by element assignment only takes place when the corresponding element of MASK is .TRUE.

The equivalent FORTRAN code for this statement is

```
DO 10 I = 1, 64
  DO 10 J = 1, 64
    IF (A(I,J) .GT. B(I,J)) A(I,J) = B(I,J)
10  CONTINUE
```

which is very inefficient on a serial processor whereas there is no difference in efficiency between A = B and A(MASK) = B on the DAP.

6.5 Examples

Clearly DAPFORTRAN is an ideal programming language when considering 64*64 matrices or matrices that can be partitioned into 64*64 submatrices. This does not mean that it is inflexible and cannot be used on problems of different dimensions; for example, we consider the problem of multiplying two N*N matrices A and B where N < 64. The following method is used:
\[
\begin{pmatrix}
ad & ae \\
bd & be
\end{pmatrix}
\begin{pmatrix}
b1 & b2 \\
b3 & b4
\end{pmatrix}
= \begin{pmatrix}
ad & ae \\
bd & be
\end{pmatrix}
\begin{pmatrix}
b1 & b2 \\
b3 & b4
\end{pmatrix}
+ \begin{pmatrix}
ad & ae \\
bd & be
\end{pmatrix}
\begin{pmatrix}
b1 & b2 \\
b3 & b4
\end{pmatrix}
\]

(the operations \(*\) and \(+\) are element by element multiplication and addition, respectively).

The DAPFORTRAN code is:

```
SUBROUTINE MATRIX_MULTIPLY (C,A,B,N)

C
C This is a subroutine to multiply two N x N matrices A and B
C where N \in \{1, \ldots, 64\} and place the result in C.
C We assume that the contents of A and B are undefined
C except for the N x N submatrix of values in the top left corner
C of A and B.
C
REAL A(,), B(,), C(,)
LOGICAL MASK(,)

MASK = ROWS (N+1, 64) .OR. COLS (N+1, 64)

C
C ROWS (N+1, 64) is a LOGICAL MATRIX FUNCTION that creates a
```
LOGICAL matrix which has its first N rows set to .FALSE. and the remaining rows set to .TRUE. COLS (N+1, 64) has the same definition, mutatis mutandis, with respect to columns.

C

C = 0.0
A (MASK) = 0.0
B (MASK) = 0.0
DO 10 K = 1, N
10 C = C + MATC (A(:,K)) * MATR (B(K,:))

MATC (REAL VECTOR) is a REAL MATRIX FUNCTION that creates a REAL matrix all of whose columns are equal to REAL VECTOR.
MATR (REAL VECTOR) has the same definition, mutatis mutandis, with respect to rows.

RETURN
END

A second example is the calculation of $C_i = A_i \ast B_i$, $i = 1, \ldots, 4096$, where $A_i$, $B_i$ and $C_i$ are all 5x5 (say) matrices. The appropriate segment of DAPFORTRAN code is:

REAL A(:, 5,5), B(:, 5,5), C(:, 5,5)
DO 10 I = 1, 5
DO 10 J = 1, 5
C(:,I,J) = 0.0
DO 10 K = 1, 5

This is identical to the equivalent code for a serial processor, except
that, in every reference to a vector or array, the first subscript is preceded by two commas to indicate that the procedure is to be carried out in every processor simultaneously.

The corresponding FORTRAN code would be:

```fortran
REAL A(4096,5,5), B(4096,5,5), C(4096,5,5)
DO 10 L = 1, 4096
  DO 20 I = 1, 5
    DO 20 J = 1, 5
      C(L,I,J) = 0.0
    DO 20 K = 1, 5
10 CONTINUE
```

6.6 Estimation Procedure and Implementation

The BHHH method is of a form suitable for parallel computation (see Chapter 3).

Three versions of the program were implemented:

(A) A parallel version on the DAP for models of up to 4096 observations.

(B) A parallel version on the DAP for between 65 and 128 observations.

(C) A parallel version on the DAP for models of up to 64 observations.

In the serial version (NLMLE, Chapter 5), we evaluate all the functions and derivatives at \( \theta^{(1)} \) for each of the observations. Clearly, this is not the most efficient method. In version (A), the architecture of the DAP allows us to evaluate the functions of up to 4096 observations simultaneously.
In versions (B) and (C), we evaluate simultaneously $L^*(\theta^{(i)} + \lambda_d^{(i)})$ for 32 or 64 values of $\lambda_d$, respectively. This is because we are able to 'partition' the DAP into 32 or 64 'parallel processors' according to the number of observations. This allows us to find the optimal value of $L^*(\theta^{(i)} + \lambda_d^{(i)})$ in our test models with a grid search procedure in only one step.

To evaluate the log-likelihood function requires the following calculations:

(a) $\sum_{t=1}^{T} f_i f_j$, for $i, j = 1, \ldots, n$

(b) $(J_t)^{-1}$, for $t = 1, \ldots, T$

(c) $\sum_{t=1}^{T} \log|\det J_t|$

On the DAP these calculations are performed very efficiently. The inner products (a) can be evaluated for a given $i$ and $j$ in two steps: firstly, we calculate $f_i f_j$, $t = 1, \ldots, T$ simultaneously and, secondly, we find the summation in one operation.

The DAPFORTRAN code is:

```fortran
REAL F(,,N), INNER_PRODUCTS (N,N)
C
DO 10 I = 1, N
  DO 10 J = 1, N
  10 INNER_PRODUCTS (I,J) = SUM (F(,,I) * F(,,J))
```

(SUM is an in-built DAPFORTRAN function that computes $\sum_{i,j} A_{ij}$ for $i, j$.)
a $64 \times 64$ matrix $A_{ij}$. 

Similarly for (b), the inversion of up to 4096 $n \times n$ matrices $J_t$ can be performed in parallel on the DAP using Gaussian elimination and column pivoting. At the same time we obtain the determinant of $J_t$.

The DAPFORTRAN code for the inversion routine is shown in Appendix H.

Lastly, (c) can be written as one line of DAPFORTRAN

\begin{verbatim}
REAL DET_JT(,), SUM_LOG_DETJT
C
SUM_LOG_DETJT = SUM (LOG(DET_JT))
\end{verbatim}

Versions (B) and (C) are similar to the above except that a separate summation in (a) and (c) is required for each $\lambda_d$. For example, the DAPFORTRAN code for (a) is:

\begin{verbatim}
REAL F(,,N), INNER_PRODUCTS (N,N,L)
LOGICAL ALPHA_MASKS (,,L)
C
C L IS THE NUMBER OF GRID POINTS ON THE LINE
DO 10 I = 1, N
  DO 10 J = 1, N
    DO 10 K = 1, L
      10 INNER_PRODUCTS (I,J,K) = SUM (MERGE (F(,,I) * F(,,J), 0.0, +ALPHA_MASKS (,,K)))
\end{verbatim}
(MERGE (REAL_MATRIX_A, REAL_MATRIX_B, LOGICAL_MATRIX_MASK) is an in-built DAPFORTRAN function that produces a REAL matrix whose elements are the same as REAL_MATRIX_A if the corresponding element of LOGICAL_MATRIX_MASK is .TRUE., and equal to the corresponding element of REAL_MATRIX_B otherwise.)

In versions (B) and (C), we can now determine the optimum $\lambda_\alpha$ by evaluating $L^t(\theta^{(1)} + \lambda_\alpha \tilde{d}^{(1)})$ for all $\lambda_\alpha$ simultaneously.

To compute the gradient $\frac{\partial L^*}{\partial \theta}$ consists essentially of matrix multiplication and taking the trace of a matrix. The DAP can do both of these operations very efficiently. Finally we evaluate the Hessian matrix which is again a matrix product.

We have chosen a set of test models (Model (iii), Chapter 7) for our DAP programs. The timings and results are described in section 7.7.
7. **Non-linear Models Simulation**

To test the NLMLE estimation program, we need to define sets of models which are simple to specify and graded by:

(i) size of the model with respect to the number of equations, \( n \); variables, \( m \); unknown parameters, \( k \); and observations, \( T \).

(ii) non-linearity (i.e. high, little, none) with respect to the unknown parameters, variables/or both

(iii) properties of data (i.e. inter-correlation)

(iv) white noise (i.e. random errors).

It is of great difficulty to obtain a realistic model with the above representation and structure because model building on such a system takes a long time to construct and collect the data. Thus we have decided to derive our non-linear system by Monte-Carlo simulation. An example of such typical non-linear model is a cross-section production model with large sample size, large parameter set, high non-linearity in the variables, moderate correlation of the data set and automatically white noise if we are sampling the data from a distribution.

It was decided to only generate the data of the model approximately to the true data by a Data Generation Process and then applied a Newton-type iterative solution to the system. We can vary the number of parameters by using the same model, but prespecify some of the parameters at fixed values to reduce the number of parameters requiring estimation.
Notice that by sampling a population $X_1, \ldots, X_T$ with the sample parameter $\theta$, the error of $\theta$ and its estimate $\hat{\theta}$ is of order $T^{-1}$, i.e.

we have the distribution of the sample population

$$D(X_1, \ldots, X_T|\theta),$$

and if we apply the maximum likelihood estimation to the sample values

$$\max_\theta L(\theta|x_1, \ldots, x_T) = \delta$$

then

$$(\hat{\theta} - \theta) = O_p\left(\frac{1}{\sqrt{T}}\right),$$

where

$O_p$ is the order in probability.

Hence when $T$ is small, the error could be very large, but as long as the estimator is consistent, we would expect $\hat{\theta}$ converge asymptotically to the true value of the sample $\theta$.

7.1 Modelling Considerations

Suppose we take $n = 2$ and $n = 5$ as the two different non-linear simultaneous systems, and that we use a combination of linear functions and arctangent to introduce variable non-linearity with respect to the endogenous variables. Notice that arctangent has the advantage that it is increasing for all values of $X$, and combined with a linear term is not likely to introduce multiple solutions. It would be appropriate to construct the model so that the function is a quadratic in the parameters and the variables raised to the power.
7.2 Model Specification

It was decided to specify:

for the \( n = 2 \) system:

\[
\Omega_u = \sigma^2 \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}
\]

and for the \( n = 5 \) system:

\[
\Omega_u = \sigma^2 \begin{bmatrix} 1.0 & 0.5 & 0.0 & 0.0 & 0.5 \\ 0.5 & 1.0 & -0.5 & 0.0 & 0.0 \\ 0.0 & -0.5 & 1.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & -0.5 \\ 0.5 & 0.0 & 0.0 & -0.5 & 1.0 \end{bmatrix}
\]

The model was chosen to be of a manageable size and generality. It was appropriate that the model be fully non-linear in the variables, and particularly that the Jacobians in each time period should be functions of both variables and parameters. It was decided to select the variables, so that the equations of the model could be easily solved for \( y_t \) as a function of \( u_t \) (so yielding the path of \( y_t \) for given \( u_t \)) by a Newton iterative solution method, starting from the path \( y_{ot} \) defined as the path corresponding to \( u_t = 0 \), or such that

\[
f(y_{ot}, z_t, \theta) = 0.
\]
Since the $z_t$ are determined once and for all, and are fixed in repeated simulations, and since it is usual for there to be at least as many variables $z_t$ as equations in the model (i.e. $m > n$), it was decided to introduce a set of $z_{it}'$ such that $z_{it}'$ occurs only in $f_i(y_{ot}', z_{t}', \theta) = 0$, $i = 1, \ldots, n$. It was then possible to first determine $y_{oit}'$, $i = 1, \ldots, n$, and $z_{it}'$, $i = n+1, \ldots, m$, using any simple procedure, and then to solve the above equation to obtain $z_{it}'$, $i = 1, \ldots, n$. This provides an initial solution path $y_{ot}'$ which serves as the starting value for a Newton iteration to solve the more general equations. The model is of the form:

$$u_{it} = f_{it}(y_{t}', z_{t}', \theta), \quad i = 1, \ldots, n \quad t = 1, \ldots, T$$

We define some intermediate functions,

$$f_{it}(y_t) = y_t \tan^{-1}(\omega_t y_{it}) + \sum_j B_{ij} y_{jt} \quad (7.1)$$

In (7.1), all or most of the coefficients can be fixed a priori, for example, we could take $\gamma_i = 0.1$ and $B_{ij} = 0$ except $B_{il} = 1$. A better alternative, which introduces a further non-linearity in $f_{it}(y_t', z_t', \theta)$ as a function of $\theta$, is to write

$$B_{ij} = B_{oij} + B_{lij} \theta_i + B_{2ij} \theta_i^2 \quad (7.2)$$

where the model contains only one or two parameters $\theta_k$, and each $B_{ij}$ is written as a quadratic in $\theta_1$, or $\theta_2$, with $B_{oij}$, $B_{lij}$ and $B_{2ij}$ fixed a priori.
Now we write

$$u_{it} = (\eta_{ii} z_{it} + (z_{(k_i t)}^2)^{\delta_i}) + f_i^*(y_{it}), \quad i = 1, \ldots, n \quad (7.3)$$

where $k_i$ depends on $i$, so that each equation contains two exogenous variables. For example, when $n = 2$, we take $k_1 = 3$ and $k_2 = 4$.

We generate the data with negligible error by using a Newton-type iterative solution method to solve the equations for a random $u_{it}$, starting the iteration from values of $y_{jt}$ that correspond to $u_{it} = 0$. If $\eta_{ii} \neq 0$, we can set the $y_{it}$ at some values $y_{oit}$ and calculate $g_{it}^* = f_i^*(y_{oit})$ and $u_{it} = 0$, and then set $z_{(k_i t)}^*$ at some equally arbitrary values, and then solve

$$z_{it} = -\frac{1}{\eta_{ii}} \{ (z_{(k_i t)}^2)^{\delta_i} + g_{it}^* \}. \quad (7.4)$$

At this stage the $y_{oit}$ are the solutions of the equations (7.3) when $u_{it} = 0$, for all $i$ and $t$. We can compute $z_{it}$ once and for all for a given model.

To generate $N$ replications for a given model, we solve equation (7.3) only approximately. We generate $u_{it}$ as jointly normal, and then add this into

$$y_{it}^* = -\eta_{ii} z_{it} + (z_{(k_i t)}^2)^{\delta_i}. \quad (7.5)$$

Now we write $y_{t}^{(r)}$ for the $r$th iterate with

$$y_{t}^{(o)} = y_{ot}$$
and
\[ \Delta y_t^{(r)} = y_t^{(r)} - y_t^{(r-1)} \]

Then we use a Newton-type iteration for the solution of
\[ f_1^* (y_t^{(r)}) = 0 \]
and
\[ y_t^{(r)} = y_t^{(r-1)} - f_1^* (y_t^{(r-1)}) / f_1'' (y_t^{(r-1)}) \]
\[ \frac{3f_1^*}{(y_t^{(r-1)})} \Delta y_t^{(r)} = y_t^{(r-1)} \delta y_t^{(r-1)} u_{it} \]
\[ (7.6) \]

where \( u_{it} \) and \( y_{it}^* \) are computed with \( r = 0 \). We repeat the iteration, until \( ||y_t^{(r)} - y_t^{(r-1)}|| \) is sufficiently small.

7.3 Data Generation Process

In order to consider a set of different models we take a basic model for \( n = 2 \) and \( n = 5 \) and then vary one parameter at a time.

Suppose we generate each \( z_{(k_1)t} \) for \( k_1 > n \) independently (once for all) using the following equation:
\[ z_{k_1t} = \lambda_{k_1} z_{k_1(t-1)} + \mu_k + U_{kt} \]
\[ (7.7) \]

where the \( U_{kt} \) are independently distributed as \( \sim N(0, 1) \).

For the \( n = 2 \) case, we take only one \( k \), and set \( k_1 = 3 \), \( \lambda_3 = 0.5, \mu_k = 1.0. \)

For the \( n = 5 \) case, we take \( k_1 = 6 \) and \( k_2 = 7 \), and set \( \lambda_6 = 0.5, \lambda_7 = 0.7, \mu_6 = \mu_7 = 1.0. \)

We then generate the values of \( y_{oit} \) using the same form of equations (7.7) by using \( y \) instead of \( z \).
Now for \( n = 2 \), we generate:

\[ y_{01t} \text{ using } \lambda = 0.5, \mu = 2.0; \]
\[ y_{02t} \text{ using } \lambda = 0.7, \mu = 2.0. \]

In addition when \( n = 5 \), we generate:

\[ y_{03t} \text{ using } \lambda = 0.5, \mu = -1.0; \]
\[ y_{04t} \text{ using } \lambda = 0.7, \mu = -1.0; \]
\[ y_{05t} \text{ using } \lambda = 0.5, \mu = 0.0. \]

Now for the values of \( B_{ij} \) in equation (7.2), we take \( B_{ij} = 0 \) for all models.

For \( n = 2 \), we take one parameter \( \theta_k = \theta \), and set

\[
[B_{1ij}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},
\]

\[
[B_{2ij}] = \varepsilon \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}
\]

where \( \varepsilon = 0.5 \).

For \( n = 5 \), we set

\[
[B_{1ij}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}
\]
where $\varepsilon = 0.1$, and we also set $a_i = 1$ for all $i$.

Postulation

For the two systems and for all $i$, we fix the values of the parameters to be:

\[ \eta_{ii} = 5 \]
\[ \delta_i = 1 \]
\[ \gamma_i = 0.3 \]
\[ \alpha_i = 1 \]
\[ \theta_i = 1 \] .

We also fix $\sigma^2 = 0.5$ and $T = 20$.

With these quantities, we can proceed the Data Generation Process according to the specification of our models.

7.4 Sets of Models

It is proposed to generate one set of data for $n = 5$ models, and consider three values for the number of parameters $p$.

$n = 5$

(1) $p = 18$

We assume $a_i = a$ is one unknown, that $B_{ij}$, $i = 1, 2, 3$ depends on the unknown $\theta_1$, and $B_{ij}$, $i = 4, 5$ depends on the unknown $\theta_2$, and take $\gamma_i$, $\eta_{ii}$, $\delta_i$ all unknown for $i = 1, \ldots, 5$. This gives a total of 18 unknown parameters.
Now the model becomes:

\[
\begin{align*}
    f_1^*(y_t) &= Y_1 \tan^{-1}(a_{y1t}) + \theta_1 y_{1t} + 0.1 \cdot \theta_2^2 y_{2t} \\
    f_2^*(y_t) &= Y_2 \tan^{-1}(a_{y2t}) + 0.1 \cdot \theta_2^2 y_{1t} + \theta_1 y_{2t} \\
    f_3^*(y_t) &= Y_3 \tan^{-1}(a_{y3t}) - 0.1 \cdot \theta_2^2 y_{2t} + (\theta_1 + 0.1 \cdot \theta_2^2) y_{3t} \\
    f_4^*(y_t) &= Y_4 \tan^{-1}(a_{y4t}) + 0.1 \cdot \theta_2^2 y_{3t} + (\theta_2 - 0.1 \cdot \theta_2^2) y_{4t} \\
    f_5^*(y_t) &= Y_5 \tan^{-1}(a_{y5t}) + 0.1 \cdot \theta_2^2 y_{2t} + 0.1 \cdot \theta_2^2 y_{4t} + \theta_2 y_{5t} \\
\end{align*}
\]

and

\[
\begin{align*}
    u_{1t} &= (n_{11} z_{1t} + (z_{6t})^2 \delta_1) + f_1^*(y_t) \\
    u_{2t} &= (n_{22} z_{2t} + (z_{6t})^2 \delta_2) + f_2^*(y_t) \\
    u_{3t} &= (n_{33} z_{3t} + (z_{7t})^2 \delta_3) + f_3^*(y_t) \\
    u_{4t} &= (n_{44} z_{4t} + (z_{7t})^2 \delta_4) + f_4^*(y_t) \\
    u_{5t} &= (n_{55} z_{5t} + (z_{7t})^2 \delta_5) + f_5^*(y_t) \\
\end{align*}
\]

The unknown parameters are

\[
Y_1, \ n_{1i}, \ \delta_i, \ i = 1, \ldots, 5; \text{ one } \alpha; \text{ one } \theta_1 \text{ and one } \theta_2.
\]

with 5 endogenous variables \(y_{1t}, \ldots, y_{5t}\) and 7 exogenous variables \(z_{1t}, \ldots, z_{7t}\).
(ii) $p = 12$

We assume the same model as in (i) but set $\theta_1 = \theta_2$ and $\delta_1 = 1$. This reduces the number of parameters by 6, that is, we have $\gamma_i$, $\eta_{ii}$, $i = 1, \ldots, 5$, one $\alpha$ and one $\theta$.

The model is in Appendix F.

(iii) $p = 6$

In addition to (ii), assume $\gamma_i = \gamma$ for all $i$ and $\eta_{11} = \eta_{33} = \eta_{55} = \eta$, this reduces the number of parameters by another 6, that is, we have $\gamma$, $\alpha$, $\theta$, $\eta$, $\eta_{22}$ and $\eta_{44}$.

The model is shown in Appendix F.

Thus from the set of generated data, we get three different optimisation problems.

(iv) $p = 9$

Assuming only one $\theta$, we have a maximum of 9 parameters, that is, 2 each of $\alpha_i$, $\gamma_i$, $\eta_{ii}$, $\delta_i$ and one $\theta$.

Thus we have the following model:

\[
\begin{align*}
    f_1^*(y_t) &= \gamma_1 \tan^{-1}(\alpha_1 y_{1t}) + (\theta + \theta^2)y_{1t} + \theta^2 y_{2t} \\
    f_2^*(y_t) &= \gamma_2 \tan^{-1}(\alpha_2 y_{2t}) - \theta^2 y_{1t} + (\theta + \theta^2)y_{2t}
\end{align*}
\]
and

\[ u_{1t} = \left( \eta_{11} z_{1t} + \left( z_{3t}^2 \right) \delta_1 \right) + f_1^*(y_t) \]

\[ u_{2t} = \left( \eta_{22} z_{2t} + \left( z_{3t}^2 \right) \delta_2 \right) + f_2^*(y_t) \]

(v) \( p = 6 \)

We constrain \( a_i = a, \gamma_i = \gamma, \delta_i = \delta, \) and so leave 6 parameters, that is, \( a, \gamma, \delta, n_{11}, n_{22} \) and \( \theta \).

The model is shown in Appendix F.

(vi) \( p = 4 \)

Finally, in addition to (v), if we take \( n_{11} = n \) and \( \delta = 1 \), we get a 4-parameter model, that is, \( \gamma, a, \theta \) and \( n \).

Again the model is shown in Appendix F.

If we take the 6-parameter model (for \( n = 2 \)) as standard, we can then vary the level of the parameters one at a time from their correct values.

7.5 Alternative Values of Parameters

We considered the following alternative values for the sets of non-linear models:

(i) \( T = 50 \)

(ii) \( \sigma^2 = 0.1 \)

(iii) \( \gamma_i = 0.5 \)
(iv) $e = 1.0$

(v) $\delta_1 = 0.7$

Each of these values has a describable general tendency:

(i) increased sample size;

(ii) reduced error variance;

(iii) reduced non-linearity of $f$ as a function of endogenous variables;

(iv) increased non-linearity with respect to parameters affecting the determination of the endogenous variables;

(v) reduced non-linearity with respect to exogenous variables' parameters.

We start by generating data for standard $n = 2$ model with $T = 50$, where the first 20 observations give us the sample for $T = 20$. We then take this data for $T = 20$, and the standard 6-parameter model and try $NPOINT = 10$ starting points. The first are all the parameters at their correct values, and the remainder are chosen so that for any parameter, for example,

$$\gamma = \gamma(1 + h),$$

where $h$ is chosen at random from the interval $(-0.5, +0.5)$. After we have seen what computing time is required for this experiment, we can then decide whether to use $NPOINT = 10$ for the other models.

Note that we have:

3 models of $n = 5$ with different values of $p$.

3 models of $n = 2$ with different values of $p$.

5 non-standard models of $n = 2$ with $p = 6$. 
Using 10 starting points for all models, yields a total of 11 models x 10 = 110 models.

Note that for each run, not only is a new starting point chosen but new values of \( u_{it} \) are generated. But we can calculate \( z_{it} \) once and for all, and then for a given model calculate \( g_{it}^{*} \) and keep these values unchanged for different runs. The actual \( y_t \) used can be retained constant, and the same data used for models differing only by \( p \).

7.6 Estimation Results and Computer Timings

We present some numerical results for particular runs of models (i) to (vi) estimated by the methods of BHHH and GMP with analytical derivatives. GMP (with numerical approximation to the derivatives) was used in various models and model (iii). Since there were only six unknown parameters in model (iii) it was worth trying this method for the \( n = 5 \) system because the number of function evaluations to approximate the gradient was relatively smaller than for models (i) and (ii). Model (iii) was also used to test the DAP program and the results from the DAP runs will be discussed in section 7.7.

Since the run time for each model starting with the true values was rather long, we were unable to test all the models with 10 different starting points except for model (v) where we varied the parameter \( \gamma \) (Table 11). However, for other models, we estimated with the true starting values and also shifted 0.05 unit away from the true values except for model (i), in this case we only shifted 0.025 unit away. Also \( \sigma^2 \) was chosen to be small (although previously
we suggested $\sigma^2 = 0.5$ so that the models might converge more quickly. To test the sensitivity of the methods to $\sigma^2$, we tested models (iv) to (vi) by varying $\sigma^2$ from 0.1 to 0.5, and the results of these runs are shown in Tables 4-6. Tables 10a and 10b show the efficiency of the methods when $T$ is small.

We now present the results from models (i) to (iii), with $n = 5$, $T = 50$, $\sigma^2 = 0.01$, $\epsilon = 0.1$, $\gamma_i = 0.3$, $\alpha = 1.0$ and $\delta_i = 1.0$.

(a-i) Model (i), $p = 18$

Initial values = True values of parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>BHHH (1) (Old Line Search)</th>
<th>BHHH (2) (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_1$</td>
<td>0.3</td>
<td>0.3035</td>
<td>0.3035</td>
<td>0.3045</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>0.3</td>
<td>0.3410</td>
<td>0.3410</td>
<td>0.3410</td>
</tr>
<tr>
<td>$Y_3$</td>
<td>0.3</td>
<td>0.3081</td>
<td>0.3081</td>
<td>0.3080</td>
</tr>
<tr>
<td>$Y_4$</td>
<td>0.3</td>
<td>0.3031</td>
<td>0.3031</td>
<td>0.3032</td>
</tr>
<tr>
<td>$Y_5$</td>
<td>0.3</td>
<td>0.2993</td>
<td>0.2993</td>
<td>0.2993</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.0</td>
<td>0.9725</td>
<td>0.9725</td>
<td>0.9726</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>1.0</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>$\theta_2$</td>
<td>1.0</td>
<td>0.9992</td>
<td>0.9992</td>
<td>0.9992</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>5.0</td>
<td>5.0132</td>
<td>5.0132</td>
<td>5.0133</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>5.0</td>
<td>5.0287</td>
<td>5.0287</td>
<td>5.0288</td>
</tr>
<tr>
<td>$\eta_{33}$</td>
<td>5.0</td>
<td>5.0012</td>
<td>5.0012</td>
<td>5.0012</td>
</tr>
<tr>
<td>$\eta_{44}$</td>
<td>5.0</td>
<td>5.0020</td>
<td>5.0020</td>
<td>5.0020</td>
</tr>
<tr>
<td>$\eta_{55}$</td>
<td>5.0</td>
<td>4.9994</td>
<td>4.9994</td>
<td>4.9994</td>
</tr>
<tr>
<td>$\delta_1$</td>
<td>1.0</td>
<td>1.0021</td>
<td>1.0021</td>
<td>1.0021</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>1.0</td>
<td>1.0023</td>
<td>1.0023</td>
<td>1.0023</td>
</tr>
<tr>
<td>$\delta_3$</td>
<td>1.0</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>$\delta_4$</td>
<td>1.0</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>$\delta_5$</td>
<td>1.0</td>
<td>0.9990</td>
<td>0.9990</td>
<td>0.9990</td>
</tr>
</tbody>
</table>
### Table 1b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of ( \dagger ) Function Evaluations</th>
<th>Initial Value of Log L(^*)</th>
<th>Final Value of Log L(^*)</th>
<th>CPU Time (CDC 7600) seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH(^{(1)})</td>
<td>58</td>
<td>108</td>
<td>-940.897488</td>
<td>-950.924755</td>
<td>434.6</td>
</tr>
<tr>
<td>BHHH(^{(2)})</td>
<td>38</td>
<td>61</td>
<td>-940.897488</td>
<td>-950.924748</td>
<td>229.0</td>
</tr>
<tr>
<td>GMP</td>
<td>25</td>
<td>68</td>
<td>-940.897488</td>
<td>-950.924769</td>
<td>601.9</td>
</tr>
</tbody>
</table>

**Note:**

(1) Old line search where the initial step-size is fixed to one.

(2) Modified line search where the initial step-size is adjusted using algorithm 2.4.1 of Chapter 2.

\( \dagger \) Number of function calls is the total number of function evaluations in the iterative procedure.

The log-likelihood function values are set to 6 decimal places due to the flatness of L\(^*\).
(a-ii) Model (i),  \( p = 18 \)

Initial values = Shifted true values of parameters

Table lc Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Shifted Initial Values</th>
<th>BHHH(^{(1)}) (Old Line Search)</th>
<th>BHHH(^{(2)}) (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_1 )</td>
<td>0.275</td>
<td>0.3035</td>
<td>0.3035</td>
<td>0.3035</td>
</tr>
<tr>
<td>( \gamma_2 )</td>
<td>0.275</td>
<td>0.3410</td>
<td>0.3410</td>
<td>0.3410</td>
</tr>
<tr>
<td>( \gamma_3 )</td>
<td>0.275</td>
<td>0.3081</td>
<td>0.3081</td>
<td>0.3080</td>
</tr>
<tr>
<td>( \gamma_4 )</td>
<td>0.275</td>
<td>0.3031</td>
<td>0.3031</td>
<td>0.3031</td>
</tr>
<tr>
<td>( \gamma_5 )</td>
<td>0.275</td>
<td>0.2993</td>
<td>0.2993</td>
<td>0.2993</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>0.975</td>
<td>0.9726</td>
<td>0.9725</td>
<td>0.9725</td>
</tr>
<tr>
<td>( \theta_1 )</td>
<td>0.975</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>( \theta_2 )</td>
<td>0.975</td>
<td>0.9992</td>
<td>0.9992</td>
<td>0.9992</td>
</tr>
<tr>
<td>( \eta_{11} )</td>
<td>4.975</td>
<td>5.0132</td>
<td>5.0132</td>
<td>5.0133</td>
</tr>
<tr>
<td>( \eta_{22} )</td>
<td>4.975</td>
<td>5.0287</td>
<td>5.0287</td>
<td>5.0288</td>
</tr>
<tr>
<td>( \eta_{33} )</td>
<td>4.975</td>
<td>5.0012</td>
<td>5.0012</td>
<td>5.0012</td>
</tr>
<tr>
<td>( \eta_{44} )</td>
<td>4.975</td>
<td>5.0020</td>
<td>5.0020</td>
<td>5.0020</td>
</tr>
<tr>
<td>( \eta_{55} )</td>
<td>4.975</td>
<td>4.9994</td>
<td>4.9994</td>
<td>4.9994</td>
</tr>
<tr>
<td>( \delta_1 )</td>
<td>0.975</td>
<td>1.0021</td>
<td>1.0021</td>
<td>1.0021</td>
</tr>
<tr>
<td>( \delta_2 )</td>
<td>0.975</td>
<td>1.0023</td>
<td>1.0023</td>
<td>1.0023</td>
</tr>
<tr>
<td>( \delta_3 )</td>
<td>0.975</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>( \delta_4 )</td>
<td>0.975</td>
<td>1.0001</td>
<td>1.0001</td>
<td>1.0001</td>
</tr>
<tr>
<td>( \delta_5 )</td>
<td>0.975</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>Method</td>
<td>No. of Iterations</td>
<td>No. of Function Evaluations</td>
<td>Initial Value of Log L*</td>
<td>Final Value of Log L*</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>BHHH (1)</td>
<td>111</td>
<td>235</td>
<td>-447.747959</td>
<td>-950.924767</td>
</tr>
<tr>
<td>BHHH (2)</td>
<td>96</td>
<td>186</td>
<td>-447.747957</td>
<td>-950.924762</td>
</tr>
<tr>
<td>GMP</td>
<td>57</td>
<td>108</td>
<td>-447.747957</td>
<td>-950.924769</td>
</tr>
</tbody>
</table>

Note:

Model (a-i) was also estimated with the initial values shifted 0.05 unit away from the true values. None of the above methods converged to the final estimates of the parameters. It was realised that the model was badly identified with the $\gamma$'s. So a set of linear restrictions was introduced, that is, by setting

$$Y_4 = 0.6 - Y_3$$

and

$$Y_5 = Y_1 + Y_2 - Y_3$$

and the model was re-estimated with its true values and the shifted values (0.05 away). From these two runs, both the GMP and BHHH failed to converge after a substantial amount of CPU times (800 seconds and 1200 seconds respectively). The GMP method failed to achieve convergence for this model with starting values of parameters shifted away by 0.05 because the initial setting of the function value was
unreliable. And for the BHHH, because the starting values were badly approximated to the final estimates, the method would not converge at all.

To ensure that model (a-i) converged to the same optimum points apart from starting from the true values, we decided to re-shift the parameters 0.025 unit away from the true values and estimated the model again. The results from these runs are shown in Tables lc and ld.

\[
\text{Figure 7.1}
\]
Tables 1 (a and b) illustrate the estimated parameters and CPU times for model (i). The BHHH procedure with the modified line search seems to be a more efficient method in terms of CPU time. It is faster than the GMP (with analytical derivatives) by a factor of 2\(\frac{1}{2}\) and converged to the same set of optimum values.

The modified line search has reduced the number of iterations and function evaluations substantially (Table 1b), in this respect, the CPU time was reduced by half \(BHHH(2)\) due to the more accurate line search during the iterative procedure.

Figure 7.1 shows the iteration numbers against the CPU times for the three estimation procedures for model (a-i) to achieve convergence. Clearly \(BHHH(2)\) is the most satisfactory optimisation technique for this application.

GMP (with numerical approximation to the derivatives) method was also used but the model failed to converge after a substantial amount of CPU time (1200 seconds). This procedure is not recommended for such models unless the analytical derivatives of the likelihood function cannot be obtained easily.
(b-i) Model (ii), \( p = 12 \)

Initial values = True values of parameters

Table 2a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>BHHH (^{(1)}) (Old Line Search)</th>
<th>BHHH (^{(2)}) (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_1 )</td>
<td>0.3</td>
<td>0.2914</td>
<td>0.2914</td>
<td>0.2913</td>
</tr>
<tr>
<td>( \gamma_2 )</td>
<td>0.3</td>
<td>0.3201</td>
<td>0.3201</td>
<td>0.3201</td>
</tr>
<tr>
<td>( \gamma_3 )</td>
<td>0.3</td>
<td>0.3189</td>
<td>0.3189</td>
<td>0.3189</td>
</tr>
<tr>
<td>( \gamma_4 )</td>
<td>0.3</td>
<td>0.3066</td>
<td>0.3066</td>
<td>0.3066</td>
</tr>
<tr>
<td>( \gamma_5 )</td>
<td>0.3</td>
<td>0.3058</td>
<td>0.3058</td>
<td>0.3058</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1.0</td>
<td>0.9592</td>
<td>0.9592</td>
<td>0.9593</td>
</tr>
<tr>
<td>( \theta )</td>
<td>1.0</td>
<td>0.9975</td>
<td>0.9975</td>
<td>0.9975</td>
</tr>
<tr>
<td>( \eta_{11} )</td>
<td>5.0</td>
<td>4.9884</td>
<td>4.9884</td>
<td>4.9884</td>
</tr>
<tr>
<td>( \eta_{22} )</td>
<td>5.0</td>
<td>4.9996</td>
<td>4.9996</td>
<td>4.9996</td>
</tr>
<tr>
<td>( \eta_{33} )</td>
<td>5.0</td>
<td>4.9999</td>
<td>4.9999</td>
<td>4.9999</td>
</tr>
<tr>
<td>( \eta_{44} )</td>
<td>5.0</td>
<td>5.0002</td>
<td>5.0002</td>
<td>5.0002</td>
</tr>
<tr>
<td>( \eta_{55} )</td>
<td>5.0</td>
<td>5.0009</td>
<td>5.0009</td>
<td>5.0009</td>
</tr>
</tbody>
</table>

Table 2b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH (^{(1)})</td>
<td>17</td>
<td>25</td>
<td>-940.897488</td>
<td>-948.419153</td>
<td>59.8</td>
</tr>
<tr>
<td>BHHH (^{(2)})</td>
<td>18</td>
<td>26</td>
<td>-940.897488</td>
<td>-948.419157</td>
<td>63.0</td>
</tr>
<tr>
<td>GMP</td>
<td>19</td>
<td>45</td>
<td>-940.897488</td>
<td>-948.419164</td>
<td>249.4</td>
</tr>
</tbody>
</table>
(b-ii) Model (ii), $p = 12$

Initial values = Shifted true values of parameters

Table 2c Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Shifted Initial Values</th>
<th>BHHH$^{(1)}$ (Old Line Search)</th>
<th>BHHH$^{(2)}$ (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_1$</td>
<td>0.25</td>
<td>0.2913</td>
<td>0.2913</td>
<td>0.2913</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>0.25</td>
<td>0.3201</td>
<td>0.3201</td>
<td>0.3201</td>
</tr>
<tr>
<td>$\gamma_3$</td>
<td>0.25</td>
<td>0.3189</td>
<td>0.3189</td>
<td>0.3189</td>
</tr>
<tr>
<td>$\gamma_4$</td>
<td>0.25</td>
<td>0.3066</td>
<td>0.3066</td>
<td>0.3066</td>
</tr>
<tr>
<td>$\gamma_5$</td>
<td>0.25</td>
<td>0.3058</td>
<td>0.3058</td>
<td>0.3058</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.95</td>
<td>0.9593</td>
<td>0.9593</td>
<td>0.9593</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0.95</td>
<td>4.9975</td>
<td>4.9975</td>
<td>4.9975</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>4.95</td>
<td>4.9884</td>
<td>4.9884</td>
<td>4.9884</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>4.95</td>
<td>4.9996</td>
<td>4.9996</td>
<td>4.9996</td>
</tr>
<tr>
<td>$\eta_{33}$</td>
<td>4.95</td>
<td>4.9999</td>
<td>4.9999</td>
<td>4.9999</td>
</tr>
<tr>
<td>$\eta_{44}$</td>
<td>4.95</td>
<td>5.0002</td>
<td>5.0002</td>
<td>5.0002</td>
</tr>
<tr>
<td>$\eta_{55}$</td>
<td>4.95</td>
<td>5.0009</td>
<td>5.0009</td>
<td>5.0009</td>
</tr>
</tbody>
</table>

Table 2d Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log $L^*$</th>
<th>Final Value of Log $L^*$</th>
<th>CPU Time (CDC 7600) seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH$^{(1)}$</td>
<td>46</td>
<td>233</td>
<td>-524.748891</td>
<td>-948.419160</td>
<td>553.8</td>
</tr>
<tr>
<td>BHHH$^{(2)}$</td>
<td>36</td>
<td>105</td>
<td>-524.748891</td>
<td>-948.419161</td>
<td>253.3</td>
</tr>
<tr>
<td>GMP</td>
<td>41</td>
<td>84</td>
<td>-524.748891</td>
<td>-948.419164</td>
<td>430.4</td>
</tr>
</tbody>
</table>
In Table 2b, $\text{BHSH}^{(1)}$ performs best, this is the only case where the old line search with step-size fixed to one works better than the modified line search. GMP seems to be slow for this model.

Tables 2c and 2d present very similar results except the CPU times for each method; notice that they have increased substantially.

Figure 7.2 shows the differences in terms of CPU times for the three methods with the true initial values.
(c) Model (iii), p = 6

Table 3a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>BHHH (^{(2)}) (Modified Line Search)</th>
<th>GMP (^{(1)}) (Analytic Derivatives)</th>
<th>GMP (^{(2)}) (Finite Differences)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\gamma)</td>
<td>0.3</td>
<td>0.2973</td>
<td>0.2972</td>
<td>0.2972</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>1.0</td>
<td>0.9726</td>
<td>0.9728</td>
<td>0.9728</td>
</tr>
<tr>
<td>(\theta)</td>
<td>1.0</td>
<td>1.0010</td>
<td>1.0010</td>
<td>1.0010</td>
</tr>
<tr>
<td>(\eta)</td>
<td>5.0</td>
<td>5.0012</td>
<td>5.0012</td>
<td>5.0012</td>
</tr>
<tr>
<td>(\eta_{22})</td>
<td>5.0</td>
<td>4.9896</td>
<td>4.9896</td>
<td>4.9896</td>
</tr>
<tr>
<td>(\eta_{44})</td>
<td>5.0</td>
<td>5.0000</td>
<td>4.9999</td>
<td>4.9999</td>
</tr>
</tbody>
</table>

Table 3b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH (^{(2)})</td>
<td>10</td>
<td>12</td>
<td>-798.450121</td>
<td>-805.136759</td>
<td>16.8</td>
</tr>
<tr>
<td>GMP (^{(1)})</td>
<td>12</td>
<td>26</td>
<td>-798.450121</td>
<td>-805.136993</td>
<td>90.7</td>
</tr>
<tr>
<td>GMP (^{(2)})</td>
<td>18</td>
<td>229</td>
<td>-798.450121</td>
<td>-805.136993</td>
<td>227.0</td>
</tr>
</tbody>
</table>
In Tables 3a and 3b, the estimated parameters and log $L^*$ seemed to agree with each other. But there are vast differences in CPU times. Again BHHH\(^{(2)}\) is the best procedure and is faster than GMP\(^{(1)}\) by a factor of 2 to 3. Although GMP\(^{(2)}\) converged to the same optimum, it is a highly inefficient method.

Figure 7.3 gives the CPU times against iteration numbers for the three estimation methods.
Note:

The above models were also tested with $\sigma^2 = 0.5$, the CPU time for each model was sufficiently large (> 600 seconds), yet the models did not seem to converge. Because of the huge CPU time further experimentation with large values of $\sigma^2$ for these three models is not feasible.

$n = 2$

For the $n = 2$ system, we present results for models (iv) to (vi) with $T = 50$, $\epsilon = 1.0$, $\gamma_1 = 0.5$, $\delta_1 = 0.7$ and varying $\sigma^2$ from 0.1 to 0.5. Only BHHH with modified line search and GMP with analytical derivatives were considered for such experiments.

(d) Model (iv), $p = 9$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>$\sigma^2 = 0.1$</th>
<th>$\sigma^2 = 0.2$</th>
<th>$\sigma^2 = 0.3$</th>
<th>$\sigma^2 = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BHHH</td>
<td>GMP</td>
<td>BHHH</td>
<td>GMP</td>
</tr>
<tr>
<td>$\gamma_1$</td>
<td>0.5</td>
<td>0.9628</td>
<td>0.9628</td>
<td>1.2953</td>
<td>1.2952</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>0.5</td>
<td>1.2352</td>
<td>1.2350</td>
<td>1.6225</td>
<td>1.6220</td>
</tr>
<tr>
<td>$\gamma_1$</td>
<td>1.0</td>
<td>0.4627</td>
<td>0.4628</td>
<td>0.3919</td>
<td>0.3919</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>1.0</td>
<td>0.3794</td>
<td>0.3794</td>
<td>0.3298</td>
<td>0.3299</td>
</tr>
<tr>
<td>$\theta$</td>
<td>1.0</td>
<td>0.9412</td>
<td>0.9412</td>
<td>0.9141</td>
<td>0.9140</td>
</tr>
<tr>
<td>$\delta_1$</td>
<td>0.7</td>
<td>0.6737</td>
<td>0.6737</td>
<td>0.6665</td>
<td>0.6664</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>0.7</td>
<td>0.6816</td>
<td>0.6816</td>
<td>0.6763</td>
<td>0.6763</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>5.0</td>
<td>4.6762</td>
<td>4.6760</td>
<td>4.5643</td>
<td>4.5639</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>5.0</td>
<td>5.0389</td>
<td>5.0387</td>
<td>5.0725</td>
<td>5.0721</td>
</tr>
<tr>
<td>Method</td>
<td>$\sigma^2$</td>
<td>No. of Iterations</td>
<td>No. of Function Evaluations</td>
<td>Initial Value of Log L*</td>
<td>Final Value of Log L*</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>--------------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.1</td>
<td>51</td>
<td>112$^\Delta$</td>
<td>-210.671923</td>
<td>-218.982182</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>55</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BHHH</td>
<td>0.2</td>
<td>89</td>
<td>155$^\Delta$</td>
<td>-176.010432</td>
<td>-184.440556</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>54</td>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BHHH$^+$</td>
<td>0.3</td>
<td>87</td>
<td>166$^\Delta$</td>
<td>-155.734175</td>
<td>-164.257193</td>
</tr>
<tr>
<td>GMP$^+$</td>
<td></td>
<td>52</td>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BHHH$^{++}$</td>
<td>0.5</td>
<td>92</td>
<td>204$^\Delta$</td>
<td>-130.188433</td>
<td>-138.861748</td>
</tr>
<tr>
<td>GMP$^{++}$</td>
<td></td>
<td>59</td>
<td>82</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: Notations used for models (iv) to (vi)

$^{++}$ non-convergence after 100 iterations.

$^+$ non-convergence after 120 seconds.

$^\Delta$ number of function calls including the number of function evaluations in gradient check by numerical approximation.
(e) Model (v), p = 6

Table 5a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>$\sigma^2 = 0.1$</th>
<th>$\sigma^2 = 0.2$</th>
<th>$\sigma^2 = 0.3$</th>
<th>$\sigma^2 = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>5.0</td>
<td>1.0534</td>
<td>1.3346</td>
<td>1.5578</td>
<td>1.9148</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.0</td>
<td>0.4528</td>
<td>0.3957</td>
<td>0.3692</td>
<td>0.3693</td>
</tr>
<tr>
<td>$\theta$</td>
<td>1.0</td>
<td>0.9361</td>
<td>0.9101</td>
<td>0.8896</td>
<td>0.8894</td>
</tr>
<tr>
<td>$\delta$</td>
<td>0.7</td>
<td>0.6755</td>
<td>0.6680</td>
<td>0.6629</td>
<td>0.6629</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>5.0</td>
<td>4.6689</td>
<td>4.5529</td>
<td>4.4660</td>
<td>4.3304</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>5.0</td>
<td>4.9437</td>
<td>4.9449</td>
<td>4.9479</td>
<td>4.9527</td>
</tr>
</tbody>
</table>

Table 5b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>$\sigma^2$</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log $L^*$</th>
<th>Final Value of Log $L^*$</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>0.1</td>
<td>45</td>
<td>70\Delta</td>
<td>-210.671923</td>
<td>-218.947553</td>
<td>34.0</td>
</tr>
<tr>
<td>GMP</td>
<td>26</td>
<td></td>
<td>38</td>
<td></td>
<td></td>
<td>44.0</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.2</td>
<td>100</td>
<td>121\Delta</td>
<td>-176.010432</td>
<td>-184.409121</td>
<td>59.4</td>
</tr>
<tr>
<td>GMP</td>
<td>27</td>
<td></td>
<td>40</td>
<td></td>
<td></td>
<td>46.5</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.3</td>
<td>52</td>
<td>77\Delta</td>
<td>-155.734175</td>
<td>-164.223673</td>
<td>37.9</td>
</tr>
<tr>
<td>GMP</td>
<td>26</td>
<td></td>
<td>39</td>
<td></td>
<td></td>
<td>47.5</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.5</td>
<td>61</td>
<td>96\Delta</td>
<td>-130.188433</td>
<td>-138.820730</td>
<td>47.1</td>
</tr>
<tr>
<td>GMP</td>
<td>29</td>
<td></td>
<td>43</td>
<td></td>
<td></td>
<td>51.4</td>
</tr>
</tbody>
</table>
(f) Model (vi), \( p = 4 \)

### Table 6a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>( \sigma^2 = 0.1 )</th>
<th>( \sigma^2 = 0.2 )</th>
<th>( \sigma^2 = 0.3 )</th>
<th>( \sigma^2 = 0.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma )</td>
<td>0.5</td>
<td>0.9036</td>
<td>1.0702</td>
<td>1.1924</td>
<td>1.3886</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.9036</td>
<td>1.0702</td>
<td>1.1923</td>
<td>1.3887</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1.0</td>
<td>0.3694</td>
<td>0.3282</td>
<td>0.3128</td>
<td>0.2997</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3694</td>
<td>0.3282</td>
<td>0.3128</td>
<td>0.2996</td>
</tr>
<tr>
<td>( \theta )</td>
<td>1.0</td>
<td>1.0471</td>
<td>1.0651</td>
<td>1.0784</td>
<td>1.0996</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0471</td>
<td>1.0651</td>
<td>1.0784</td>
<td>1.0996</td>
</tr>
<tr>
<td>( n )</td>
<td>5.0</td>
<td>5.4046</td>
<td>5.5691</td>
<td>5.6950</td>
<td>5.9013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.4046</td>
<td>5.5691</td>
<td>5.6950</td>
<td>5.9014</td>
</tr>
</tbody>
</table>

### Table 6b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>( \sigma^2 )</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log ( L^* )</th>
<th>Final Value of Log ( L^* )</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>0.1</td>
<td>20</td>
<td>34\textsuperscript{A}</td>
<td>-210.671923</td>
<td>-217.132402</td>
<td>11.6</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>18</td>
<td>27</td>
<td></td>
<td></td>
<td>23.9</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.2</td>
<td>25</td>
<td>76\textsuperscript{A}</td>
<td>-176.010432</td>
<td>-182.350609</td>
<td>25.6</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>19</td>
<td>34</td>
<td></td>
<td></td>
<td>30.0</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.3</td>
<td>21</td>
<td>58\textsuperscript{A}</td>
<td>-155.734175</td>
<td>-161.997569</td>
<td>19.6</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>18</td>
<td>25</td>
<td></td>
<td></td>
<td>23.6</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.5</td>
<td>23</td>
<td>69\textsuperscript{A}</td>
<td>-130.188433</td>
<td>-136.368487</td>
<td>23.3</td>
</tr>
<tr>
<td>GMP</td>
<td></td>
<td>20</td>
<td>39</td>
<td></td>
<td></td>
<td>30.6</td>
</tr>
</tbody>
</table>
Tables 4-6 show that the BHHH method is a more efficient procedure for the above models. When $\sigma^2$ is small (Tables 4a and 4b), the method works well but when $\sigma^2$ is increased to 0.5, then the GMP procedure performs better. In the case where $\sigma^2 = 0.3$, both methods fail to converge because the model is badly identified.

In Tables 5a and 5b, again BHHH is a better procedure for this set of models. Again in Tables 6a and 6b, BHHH performs best.

From the above experiments, the evidence is that BHHH is a better procedure for complicated models providing $\sigma^2$ is small. If we have a large $\sigma^2$, the procedure tends to be slow to locate the maximum of the likelihood function. For simple models (Tables 6a and 6b), both methods perform well but BHHH seems to be more efficient.

We now present some results of estimating models (iv) to (vi) by shifting the starting values of parameters and we take $\sigma^2 = 0.1$. 
Table 7a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Shifted Initial Values</th>
<th>BHHH$^+$ (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_1$</td>
<td>0.45</td>
<td>12.8520</td>
<td>0.9628</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>0.45</td>
<td>57.5570</td>
<td>1.2350</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.95</td>
<td>0.0794</td>
<td>0.4628</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.95</td>
<td>0.0506</td>
<td>0.3794</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0.95</td>
<td>1.4448</td>
<td>0.9412</td>
</tr>
<tr>
<td>$\delta_1$</td>
<td>0.65</td>
<td>0.8379</td>
<td>0.6737</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>0.65</td>
<td>0.9728</td>
<td>0.6816</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>4.95</td>
<td>9.9421</td>
<td>4.6760</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>4.95</td>
<td>15.3090</td>
<td>5.0387</td>
</tr>
</tbody>
</table>

Table 7b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH$^+$</td>
<td>81</td>
<td>163$^A$</td>
<td>-133.414671</td>
<td>-197.331094</td>
<td>120.8$^+$</td>
</tr>
<tr>
<td>GMP</td>
<td>49</td>
<td>68</td>
<td>-133.414671</td>
<td>-218.982182</td>
<td>101.8</td>
</tr>
</tbody>
</table>

Note: Compare with Table 4a for $\sigma^2 = 0.1$, GMP tends to work well although the starting values are not close to the true values, in fact the run time is less than the case where we started the model with its true values. For BHHH it
tends to locate a local maximum rather than the global maximum if the starting values are not close to the final estimate $\theta^*$ for this particular model.

(h) $n = 2, \ p = 6, \ T = 50$, shifted initial values from true values.

Table 8a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Shifted Initial Values</th>
<th>BHHH (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.45</td>
<td>1.0535</td>
<td>1.0534</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.95</td>
<td>0.4530</td>
<td>0.4528</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0.95</td>
<td>0.9361</td>
<td>0.9361</td>
</tr>
<tr>
<td>$\delta$</td>
<td>0.65</td>
<td>0.6756</td>
<td>0.6755</td>
</tr>
<tr>
<td>$n_{11}$</td>
<td>4.95</td>
<td>4.6689</td>
<td>4.6689</td>
</tr>
<tr>
<td>$n_{22}$</td>
<td>4.95</td>
<td>4.9438</td>
<td>4.9437</td>
</tr>
</tbody>
</table>

Table 8b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>52</td>
<td>$140$^\Delta</td>
<td>-133.414671</td>
<td>-218.947550</td>
<td>62.6</td>
</tr>
<tr>
<td>GMP</td>
<td>23</td>
<td>36</td>
<td>-133.414671</td>
<td>-218.947553</td>
<td>42.6</td>
</tr>
</tbody>
</table>

Note: Compare with Tables 5a and 5b for $\sigma^2 = 0.1$, GMP is more efficient in locating the maximum of the likelihood function.
although the starting values are shifted away from the true values.

On the other hand, BHHH takes twice as long to converge compared with its previous CPU time.

(i) \( n = 2, \ p = 4, \ T = 50, \) shifted initial values from true values.

Table 9a Final Estimates of Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Shifted Initial Values</th>
<th>BHHH (Modified Line Search)</th>
<th>GMP (Analytical Derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma )</td>
<td>0.45</td>
<td>0.9036</td>
<td>0.9036</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>0.95</td>
<td>0.3694</td>
<td>0.3694</td>
</tr>
<tr>
<td>( \theta )</td>
<td>0.95</td>
<td>1.0471</td>
<td>1.0471</td>
</tr>
<tr>
<td>( \eta )</td>
<td>4.95</td>
<td>5.4046</td>
<td>5.4046</td>
</tr>
</tbody>
</table>

Table 9b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>27</td>
<td>76( \Delta )</td>
<td>-135.783150</td>
<td>-217.132402</td>
<td>25.6</td>
</tr>
<tr>
<td>GMP</td>
<td>20</td>
<td>31</td>
<td>-135.783150</td>
<td>-217.132402</td>
<td>25.9</td>
</tr>
</tbody>
</table>

Note: Compare Table 9b with Table 6b for \( \sigma^2 = 0.1 \), the BHHH procedure takes twice as long to locate the same maximum when the starting values of the parameters are not close to the true values, but GMP works well although we have shifted the parameters.
The above experiments have provided us some evidence that the BHHH is sensitive over the starting values of the parameters. If the starting values of any models that are not close to the final estimates, it would be appropriate to use GMP and then go to BHHH.

To test the effect of sample size, we estimate model (ii) with $T = 20$. Tables 10a and 10b present the result from these runs.

### Table 10a Final estimates of the Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial Values</th>
<th>BHHH (Modified line Search)</th>
<th>GMP (Analytical derivatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_1$</td>
<td>0.3</td>
<td>0.3041</td>
<td>0.3042</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>0.3</td>
<td>0.3516</td>
<td>0.3516</td>
</tr>
<tr>
<td>$Y_3$</td>
<td>0.3</td>
<td>0.3334</td>
<td>0.3334</td>
</tr>
<tr>
<td>$Y_4$</td>
<td>0.3</td>
<td>0.3153</td>
<td>0.3153</td>
</tr>
<tr>
<td>$Y_5$</td>
<td>0.3</td>
<td>0.3223</td>
<td>0.3223</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.0</td>
<td>0.9055</td>
<td>0.9054</td>
</tr>
<tr>
<td>$\theta$</td>
<td>1.0</td>
<td>0.9942</td>
<td>0.9942</td>
</tr>
<tr>
<td>$\eta_{11}$</td>
<td>5.0</td>
<td>4.9816</td>
<td>4.9816</td>
</tr>
<tr>
<td>$\eta_{22}$</td>
<td>5.0</td>
<td>5.0018</td>
<td>5.0018</td>
</tr>
<tr>
<td>$\eta_{33}$</td>
<td>5.0</td>
<td>5.0053</td>
<td>5.0053</td>
</tr>
<tr>
<td>$\eta_{44}$</td>
<td>5.0</td>
<td>5.0028</td>
<td>5.0028</td>
</tr>
<tr>
<td>$\eta_{55}$</td>
<td>5.0</td>
<td>5.0015</td>
<td>5.0015</td>
</tr>
</tbody>
</table>

### Table 10b Details of Iterative Convergence

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of Iterations</th>
<th>No. of Function Evaluations</th>
<th>Initial Value of Log L*</th>
<th>Final Value of Log L*</th>
<th>CPU Time (CDC 7600) Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>44</td>
<td>69</td>
<td>-373.153883</td>
<td>-378.556694</td>
<td>66.5</td>
</tr>
<tr>
<td>GMP</td>
<td>25</td>
<td>53</td>
<td>-373.153883</td>
<td>-378.556694</td>
<td>112.1</td>
</tr>
</tbody>
</table>
If we compare Tables 10a and 10b with Tables 2a and 2b, the CPU time for BHHH method does not change much although we have reduced the sample size \( T \) from 50 to 20. But for GMP, the CPU time for \( T = 20 \) model has reduced substantially, this may indicate that the BHHH method will tend only to gain an advantage in time over the GMP when \( T \) is large.

Figures 7.4 and 7.5 show the time taken for each method against the number of parameters for models (i) to (iii) and (iv) to (vi) respectively. In all these cases (except model (ii)), BHHH with modified line search performs best, it is clearly more efficient when \( p > 12 \) (Figure 7.4). GMP (with numerical approximation to the derivatives) is not recommended when \( p > 6 \) (Figure 7.5).

Generally, BHHH seems to work well for this class of models, the drawback is that if \( u_t \) is not independently identically normally distributed, or if the form of the model is misspecified, which implies \( R \) (equation 3.16) is not a consistent positive definite estimator of the Hessian matrix \( H \), then the method may not be efficient. But we need to stress that all methods are model dependent, and that it is difficult to predict their relative efficiencies. Both methods can be expected to obtain a relative maximum of the likelihood function if they converge. As a safeguard against lack of convergence it is suggested that the BHHH algorithm is tried with a limit on the number of iterations. If the method fails to converge, then the GMP algorithm is started from the final values achieved by the BHHH algorithm.
Figure 7.4.

- 159 -
TOTAL TIME (SECONDS)

NUMBER OF PARAMETERS $P$

- $p = 9$
- $p = 6$
- $p = 4$

BHHH $\rightarrow$ MODIFIED LINE SEARCH
GMP$^{(1)}$ $\rightarrow$ ANALYTICAL DERIVATIVES
GMP$^{(2)}$ $\rightarrow$ NUMERICAL APPROXIMATION

Figure 7.5
Non-Standard Models

The following table presents the results for model (v) by varying the parameter \( \gamma, \sigma^2 = 0.005 \).

We first choose \( \gamma' = \gamma(1 + h) \) where \( h = (-0.5, +0.5) \) and set \( \gamma = 0.5 \). We then generate the data series by using the true value of \( \gamma = 0.5 \) and set the initial values of \( \gamma' \) as defined above.

For 10 different starting points, we now have

\[ \gamma' = \{0.25, 0.30, 0.35, 0.40, 0.45, 0.55, 0.60, 0.65, 0.70, 0.75\} \]

and the parameter list becomes:

\[ \theta^o = \{\gamma, \alpha, \theta, \delta, \eta_{11}, \eta_{22}\} \]

Thus we have 10 models with \( n = 2, p = 6 \) and \( T = 50 \), each of these has a different starting value of \( \gamma' \). The starting values of \( \{\alpha, \theta, \delta, \eta_{11}, \eta_{22}\} \) are fixed at their true values, that is, \( \{1.0, 1.0, 1.0, 5.0, 5.0\} \) respectively.

From Table 11, it can be seen that the CPU time for each run is large and hence we do not carry out the experiment further by varying the rest of the parameters.

Again we have found that BHHH works better except in the case \( \gamma' = 0.45 \).
Table 11
Final Estimates of Parameters and Details of Iterative Convergence

<table>
<thead>
<tr>
<th>METHODS</th>
<th>$\gamma^*$</th>
<th>$\alpha$</th>
<th>$\theta$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$\log L^*$</th>
<th>ITN NO.</th>
<th>FUN EVALS.</th>
<th>CPU TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHH</td>
<td>0.4026</td>
<td>0.5660</td>
<td>0.9789</td>
<td>0.6909</td>
<td>4.8791</td>
<td>4.9665</td>
<td>-333.196186</td>
<td>21</td>
<td>52</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.4025</td>
<td>0.5660</td>
<td>0.9789</td>
<td>0.6909</td>
<td>4.8791</td>
<td>4.9665</td>
<td>-333.196220</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.4025</td>
<td>0.5660</td>
<td>0.9789</td>
<td>0.6909</td>
<td>4.8791</td>
<td>4.9665</td>
<td>-333.196220</td>
<td>22</td>
<td>216</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.4501</td>
<td>0.6029</td>
<td>0.9786</td>
<td>0.6908</td>
<td>4.8772</td>
<td>4.9645</td>
<td>-333.326965</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.4501</td>
<td>0.6027</td>
<td>0.9786</td>
<td>0.6908</td>
<td>4.8771</td>
<td>4.9645</td>
<td>-333.326993</td>
<td>18</td>
<td>29</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.4501</td>
<td>0.6027</td>
<td>0.9786</td>
<td>0.6908</td>
<td>4.8771</td>
<td>4.9645</td>
<td>-333.326993</td>
<td>19</td>
<td>183</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.4984</td>
<td>0.6334</td>
<td>0.9784</td>
<td>0.6908</td>
<td>4.8754</td>
<td>4.9629</td>
<td>-333.326993</td>
<td>18</td>
<td>44</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.4984</td>
<td>0.6334</td>
<td>0.9783</td>
<td>0.6907</td>
<td>4.8754</td>
<td>4.9629</td>
<td>-333.326993</td>
<td>22</td>
<td>225</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.5470</td>
<td>0.6591</td>
<td>0.9781</td>
<td>0.6907</td>
<td>4.8742</td>
<td>4.9615</td>
<td>-333.326993</td>
<td>18</td>
<td>174</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.5471</td>
<td>0.6594</td>
<td>0.9781</td>
<td>0.6907</td>
<td>4.8740</td>
<td>4.9615</td>
<td>-333.326993</td>
<td>18</td>
<td>31</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.5471</td>
<td>0.6594</td>
<td>0.9781</td>
<td>0.6907</td>
<td>4.8740</td>
<td>4.9615</td>
<td>-333.326993</td>
<td>18</td>
<td>174</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.5962</td>
<td>0.6818</td>
<td>0.9784</td>
<td>0.6908</td>
<td>4.8756</td>
<td>4.9630</td>
<td>-333.326993</td>
<td>18</td>
<td>45</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.5962</td>
<td>0.6817</td>
<td>0.9784</td>
<td>0.6908</td>
<td>4.8754</td>
<td>4.9629</td>
<td>-333.326993</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.5962</td>
<td>0.6817</td>
<td>0.9784</td>
<td>0.6908</td>
<td>4.8754</td>
<td>4.9629</td>
<td>-333.326993</td>
<td>23</td>
<td>225</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.6949</td>
<td>0.7175</td>
<td>0.9775</td>
<td>0.6906</td>
<td>4.8708</td>
<td>4.9585</td>
<td>-333.326993</td>
<td>18</td>
<td>41</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.6948</td>
<td>0.7176</td>
<td>0.9775</td>
<td>0.6906</td>
<td>4.8708</td>
<td>4.9585</td>
<td>-333.326993</td>
<td>18</td>
<td>29</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.6948</td>
<td>0.7176</td>
<td>0.9775</td>
<td>0.6906</td>
<td>4.8708</td>
<td>4.9585</td>
<td>-333.326993</td>
<td>18</td>
<td>188</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.7443</td>
<td>0.7453</td>
<td>0.9773</td>
<td>0.6905</td>
<td>4.8699</td>
<td>4.9576</td>
<td>-334.079223</td>
<td>23</td>
<td>31</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.7443</td>
<td>0.7454</td>
<td>0.9773</td>
<td>0.6905</td>
<td>4.8701</td>
<td>4.9578</td>
<td>-334.079223</td>
<td>17</td>
<td>29</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.7443</td>
<td>0.7454</td>
<td>0.9773</td>
<td>0.6905</td>
<td>4.8701</td>
<td>4.9578</td>
<td>-334.079223</td>
<td>21</td>
<td>211</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.7940</td>
<td>0.7973</td>
<td>0.9772</td>
<td>0.6905</td>
<td>4.8693</td>
<td>4.9572</td>
<td>-334.048734</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.7940</td>
<td>0.7973</td>
<td>0.9772</td>
<td>0.6905</td>
<td>4.8693</td>
<td>4.9572</td>
<td>-334.048734</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.7940</td>
<td>0.7973</td>
<td>0.9772</td>
<td>0.6905</td>
<td>4.8693</td>
<td>4.9572</td>
<td>-334.048734</td>
<td>18</td>
<td>177</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.8440</td>
<td>0.8571</td>
<td>0.9770</td>
<td>0.6905</td>
<td>4.8662</td>
<td>4.9563</td>
<td>-333.323967</td>
<td>14</td>
<td>35</td>
</tr>
<tr>
<td>BHHH</td>
<td>0.8437</td>
<td>0.8570</td>
<td>0.9771</td>
<td>0.6905</td>
<td>4.8662</td>
<td>4.9563</td>
<td>-333.323967</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>GMP(1)</td>
<td>0.8437</td>
<td>0.8570</td>
<td>0.9771</td>
<td>0.6905</td>
<td>4.8662</td>
<td>4.9563</td>
<td>-333.323967</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>GMP(2)</td>
<td>0.8534</td>
<td>0.8575</td>
<td>0.9770</td>
<td>0.6905</td>
<td>4.8666</td>
<td>4.9562</td>
<td>-334.445057</td>
<td>19</td>
<td>188</td>
</tr>
</tbody>
</table>

*Note: BHHH with old line search
GMP(1) with analytical derivatives
GMP(2) with finitie differences
CPU time is CDC 7600 time*
7.7 Results and Timings for Model (iii) on the Distributed Array Processor

Five sets of model (iii) were tested. The model had 4096 observations. Due to its size it was not possible to run it on the CDC 7600 and ICL 2980 computers.

To provide a comparison with the DAP program, version (A), we ran the serial version (NLMLE) for 64, 128, 256, 512, 1024 and 2048 observations, respectively, and based on these timings we estimated the time for NLMLE on the 4096 observations.

Table 12 presents the timings of one function call for the above model with different sample sizes.

Table 12

<table>
<thead>
<tr>
<th>T</th>
<th>ICL 2980 (Seconds)</th>
<th>CDC 7600 (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>5.444</td>
<td>1.196</td>
</tr>
<tr>
<td>128</td>
<td>11.248</td>
<td>2.370</td>
</tr>
<tr>
<td>256</td>
<td>19.737</td>
<td>4.698</td>
</tr>
<tr>
<td>512</td>
<td>38.384</td>
<td>9.311</td>
</tr>
<tr>
<td>1024</td>
<td>78.327</td>
<td>19.010</td>
</tr>
<tr>
<td>2048</td>
<td>174.691</td>
<td>42.401</td>
</tr>
<tr>
<td>4096</td>
<td>389.609*</td>
<td>93.691*</td>
</tr>
</tbody>
</table>

* Estimated Times
For each of version (B) and (C), we ran two models, the first having the starting values of the parameters as the true values, and the second having the starting values scaled by 2%. Table 13 presents the timings for all the models.

Table 13

<table>
<thead>
<tr>
<th>Model</th>
<th>T</th>
<th>SYSTEM</th>
<th>NO. OF. ITERATIONS</th>
<th>NO. OF. FUNCTION CALLS</th>
<th>TIME PER ITERATION (SECONDS)</th>
<th>TOTAL CPU TIME (SECONDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4096</td>
<td>DAP</td>
<td>2</td>
<td>2</td>
<td>3.067</td>
<td>6.134</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICL 2980</td>
<td>-</td>
<td>-</td>
<td>389.609*</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDC 7600</td>
<td>-</td>
<td>-</td>
<td>93.691*</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>DAP</td>
<td>2</td>
<td>2</td>
<td>3.981</td>
<td>7.962</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICL 2980</td>
<td>4</td>
<td>4</td>
<td>11.248</td>
<td>44.992</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDC 7600</td>
<td>4</td>
<td>4</td>
<td>2.370</td>
<td>9.480</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>DAP</td>
<td>4</td>
<td>4</td>
<td>3.999</td>
<td>15.996</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICL 2980</td>
<td>8</td>
<td>10</td>
<td>11.747</td>
<td>117.470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDC 7600</td>
<td>8</td>
<td>10</td>
<td>2.851</td>
<td>28.510</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>DAP</td>
<td>6</td>
<td>6</td>
<td>4.245</td>
<td>25.470</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICL 2980</td>
<td>9</td>
<td>10</td>
<td>5.444</td>
<td>55.440</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDC 7600</td>
<td>9</td>
<td>10</td>
<td>1.196</td>
<td>11.960</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>DAP</td>
<td>5</td>
<td>5</td>
<td>4.251</td>
<td>21.255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICL 2980</td>
<td>17</td>
<td>20Δ</td>
<td>6.255</td>
<td>125.100Δ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDC 7600</td>
<td>17</td>
<td>20Δ</td>
<td>1.518</td>
<td>30.360Δ</td>
</tr>
</tbody>
</table>

+ Scaled starting values of Parameters
* Estimates Times
Δ Non-convergence
If the timing is expressed graphically in Figure 7.6, we see that the DAP version (A) is faster than the serial version, on the CDC 7600, when \( T > 179 \), and, on the ICL 2980, when \( T > 49 \). Also in Figure 7.6, DAP versions (B) and (C) are slower than all the other versions for a single function call. However, during each function call in version (B) and (C), we are doing 64 and 32 simultaneous function evaluations respectively in the grid search procedure. In fact, from Table 13, the DAP versions (B) and (C) are more efficient at finding the optimum values of the parameters when the starting values are furthest from the true values.

Our conclusions based on the DAP programs are limited by the fact that the data for our comparisons was generated artificially. However, we would expect that the results for real data to be very similar. Figure 7.6 shows that DAP version (A) is very efficient for large sample models. Consequently, we would expect that the DAP would be appropriate for cross-section data models.

The application of a grid-search procedure in DAP versions (B) and (C) produced significant improvements for scaled starting values of the parameters. Thus we would expect similar improvements for real data sets.

Lastly, we would like to make some general remarks concerning the DAP. Clearly, the DAP should only be considered for a particular computation when the ability to do that computation on a serial computer is limited by the size of the data sets and the basic speed of the serial processor. For any particular computation there is no certainty that the DAP can provide a significant speed-up over a serial processor because the computation could be highly non-parallel.
In this section of the research, we have shown that non-linear optimisation does contain a significant amount of parallel computation and hence the DAP should be seriously considered for this type of computation. To use the DAP rather than a serial computer requires the user to learn DAPFORTRAN. Since DAPFORTRAN is an extension of standard FORTRAN, this was a lot easier than learning a completely new language. Moreover, DAPFORTRAN was found to be more powerful and concise than standard FORTRAN. In fact, the conciseness of DAPFORTRAN enhanced the comprehension of the code and shortened the development time.

Figure 7.6
8. A Demand Model

A more realistic model used was a small macroeconomic model of the British Economy specified by David F. Hendry (1974). The model consists of eight equations, two of which are identities, and 18 unknown coefficients in its structural equations.

8.1 The Model (linear form)

In Hendry's specification, only equations linear in both variables and parameters are considered. The behavioural equations explain consumers' expenditure on durable goods (Cd) and all other goods and services (Cn), gross domestic fixed capital formation (I), inventory investment (Iv), and imports of goods and services (M). Gross domestic product (Y) is determined by the usual accounting identity, and the model is closed by an empirical relation to determine disposable income (Yd). G is real current government expenditure, X is real exports, T is real net indirect taxes, Pm is an index of relative import prices and N is a dummy variable related to the change in the timing of automobile licensing. Dummy variables for a constant term and three seasonal shift factors are included in every equation.
The FIML estimates are as follows:

(i) \( Cd = 0.0816 Yd + 57.0921 N + 0.6530 Cd_1 - 0.1898 Cd_2 + u_1 \)

(ii) \( Cn = 0.1458 Yd + 0.7733 Cn_1 + u_2 \)

(iii) \( I = -0.0690 \Delta Y + 0.7439 I_1 + 0.1042 \Delta Y_1 + 0.2584 I_2 + u_3 \)

(iv) \( Iv = 0.1299 \Delta Y + 0.4417 Iv_1 + 0.2644 Iv_2 + u_4 \)

(v) \( M = 0.5004 Iv - 4.8623 Pm + 0.2338 Y_1 + u_5 \)

(vi) \( Yd = 0.2294 Y + 0.7127 Yd_1 + u_6 \)

\[ Y = Cd + Cn + I + Iv + G + X - T - M \]

\[ \Delta Y = Y - Y_1 \]

The suffix is used to denote a corresponding lag, for example, \( Cd_2 = Cd_{(t-2)} \).

Briefly, equation (i) is derived from a stock-adjustment, expected (or permanent) income model, and includes a dummy variable \( N \) for the annual vehicle registration letter. Equation (ii) is a transformed permanent income equation. Equations (iii) and (iv) are derived from flexible accelerator-capital stock adjustment models. Equation (v) assumes linear price, income, and stock building effects, and (vi) is a transformed distributed lag relationship.
It was decided as an exercise in estimating non-linear models to change only the first equation so that it is linear in the logarithms of the corresponding variables. In order to ensure that the model has, in fact, a predominantly non-linear form it was decided to use as its basic variables the logarithms of the economic variables.

8.2 Transformation of Model (non-linear form)

To transform the set of equations into non-linear form, we first transform the variable to logarithmic time series (except those variables with negative data) and then apply the exponential function to the transformed variables again. Hence we have a new set of equations that would correspond to equations (i) and (vi) but non-linear in both parameters and variables. To eliminate the two identities, we substitute both the variables $Y$ and $\Delta Y$ in the stochastic equations (i) to (vi).

Let the set of endogenous variables be:

\[
\begin{align*}
Y_1 &= \log Cd \\
Y_2 &= \log Yd \\
Y_3 &= \log Cn \\
Y_4 &= \log I \\
Y_5 &= Iv \\
Y_6 &= \log M
\end{align*}
\]

and the set of exogenous variables be:
\[ z_1 = \log C_{d1} \]
\[ z_2 = \log C_{d2} \]
\[ z_3 = N \]
\[ z_4 = \log C_n \]
\[ z_5 = \log I_1 \]
\[ z_6 = \log I_2 \]
\[ z_7 = I_{\nu 1} \]
\[ z_8 = \log P_m \]
\[ z_9 = \log Y_{d1} \]
\[ z_{10} = G + X - T \]
\[ z_{11} = Y_1 \]
\[ z_{12} = Y_1 - Y_2 \]
\[ z_{13} = I_{\nu 2} \]

Now the model becomes:

(i) \[ u_1 = y_1 - \beta_{12}y_2 - \gamma_{11}z_1 - \gamma_{12}z_2 - \gamma_{13}z_3 - \gamma_{10} \]

(ii) \[ u_2 = \exp y_3 - \beta_{22}\exp y_2 - \gamma_{24}\exp z_4 - \gamma_{20} \]

(iii) \[ u_3 = \exp y_4 - \beta_{30}\{\exp y_1 + \exp y_3 + \exp y_4 + y_5 - \exp y_6 + z_{10} \]
\[ - z_{11}\} - \gamma_{35}\exp z_5 - \gamma_{36}\exp z_6 - \gamma_{3(12)}z_{12} - \gamma_{30} \]
(iv) \[ u_4 = y_5 - \beta_{48} (\exp y_1 + \exp y_3 + \exp y_4 + y_5 - \exp y_6 + z_{10} - z_{11}) \]

\[ - \gamma_{47} z_7 - \gamma_{4(13)} z_{13} - \gamma_{40} \]

(v) \[ u_5 = \exp y_6 - \beta_{55} y_5 - \gamma_{58} \exp z_8 - \gamma_{5(11)} z_{11} - \gamma_{50} \]

(vi) \[ u_6 = \exp y_2 - \beta_{67} (\exp y_1 + \exp y_3 + \exp y_4 + y_5 - \exp y_6 + z_{10}) \]

\[ - \gamma_{6(9)} z_9 - \gamma_{60} \]

We have a total of 24 unknown parameters (18 coefficients plus 6 constants) to be estimated in this transformed model.

8.3 Treatment of Coefficients for Equation \( u_1 \) and Constants for all Equations

Let \( \bar{Y}_1 = \bar{C}d \)

and \( \bar{Y}_2 = \bar{Y}d \) where \( \bar{C}d = \frac{\Sigma C d}{T} \) and \( \bar{Y}d = \frac{\Sigma Y d}{T} \).

Then to obtain starting values for the parameters we linearise the first equation in terms of the logarithmic variables by taking a first order Taylor series expansion of the FIML estimates equation in the form:

\[ \exp y_1 = 0.0816 \exp y_2 + 52.091 z_3 + 0.6530 \exp z_1 \]

\[ - 0.1898 \exp z_2 + u_1. \]

This gives \( \beta_{12} = \hat{\beta}_{12} \left( \frac{\bar{Y}_2}{\bar{Y}_1} \right) \).
where $\hat{\beta}_{12}$ is the FIML estimate of $Y_d$ in equation (i), that is, $\hat{\beta}_{12} = 0.0816$.

Let $\bar{z}_1 = \bar{C}_d$, $\bar{z}_2 = \tilde{C}_d$, where $\bar{C}_d = \frac{\Sigma C_d}{T}$ and $\tilde{C}_d = \frac{\Sigma C_d}{T}$.

Then $\hat{\gamma}_{11} = \hat{\gamma}_{11} \left( \frac{\bar{z}_1}{\bar{y}_1} \right)$

$\hat{\gamma}_{12} = \hat{\gamma}_{12} \left( \frac{\bar{z}_2}{\bar{y}_1} \right)$

and $\hat{\gamma}_{13} = \hat{\gamma}_{13} \left( \frac{1}{\bar{y}_1} \right)$

again $\hat{\gamma}_{11}$, $\hat{\gamma}_{12}$, $\hat{\gamma}_{13}$ are the FIML estimates of $C_{d1}$, $C_{d2}$ and $N$ respectively.

The Hendry model includes seasonal dummies in each equation, which contributes 18 coefficients to the total of 36 parameters which he estimates. It was considered that 36 parameters were too many to estimate with the program in a full non-linear model, so that to reduce the number of parameters to 18 it would be better to deseasonalise the data, and omit the seasonal dummies from the equations. This required an adjustment to the starting values of the constant terms on the equations which were computed as follows:

Let $W_1 = y_1 - \beta_{12} y_2 - y_{11}z_1 - y_{12}z_2 - y_{13}z_3$

and now calculate the quarterly means of
\( \bar{W}_1(1) = \) first quarter mean of \( W_1 \)

\( \bar{W}_1(2) = \) second quarter mean of \( W_1 \)

\( \bar{W}_1(3) = \) third quarter mean of \( W_1 \)

\( \bar{W}_1(4) = \) forth quarter mean of \( W_1 \)

and then take as initial values,

\[ Y_{10} = \bar{W}_1(4) \]

\[ Y_{1(14)} = \bar{W}_1(1) - \bar{W}_1(4) \]

\[ Y_{1(15)} = \bar{W}_1(2) - \bar{W}_1(4) \]

\[ Y_{1(16)} = \bar{W}_1(3) - \bar{W}_1(4). \]

Now we have the following starting values of all the constant coefficients:

For all the equations except equation (i), we have

\[ Y_{10}^{*} = Y_{10} + \frac{1}{4} (Y_{1(14)} + Y_{1(15)} + Y_{1(16)}) \]

where \( Y_{10}, \ i = 1, \ldots, 6 \) are the values of constants from the FIML estimates.

For equation (i), we calculate \( W_1 \) as before but now take

\[ Y_{10}^{*} = \) Arithmetic mean of \( W_1 \) for the whole sample.\]
For the other unknown coefficients, the corresponding FIML estimates are taken as their starting values.

8.4 Deseasonised Data Series

Let \( X_t = \log x_t = a_0 + a_1 Q_{1t} + a_2 Q_{2t} + a_3 Q_{3t} + V_t \).

We choose the dummy variables to be \( Q_{1t} \), \( Q_{2t} \) and \( Q_{3t} \) where:

\[
\begin{array}{ccc}
Q_{1t} & Q_{2t} & Q_{3t} \\
3 & -1 & -1 \\
-1 & 3 & -1 \\
-1 & -1 & 3 \\
-1 & -1 & -1 \\
3 & -1 & -1 \\
-1 & 3 & -1 \\
-1 & -1 & 3 \\
-1 & -1 & -1 \\
& & \\
& & \\
& & \\
& & \\
\end{array}
\]

and \( t = 1, \ldots, T \).

Then the \( a_i \) are determined by multiple regression.

Now let

\[
x_t^* = x_t - a_1 Q_{1t} - a_2 Q_{2t} - a_3 Q_{3t}
\]

and
\[ \begin{bmatrix}
1 & 3 & -1 & -1 \\
1 & -1 & 3 & -1 \\
1 & -1 & -1 & 3 \\
1 & -1 & -1 & -1 \\
1 & 3 & -1 & -1 \\
\vdots & \vdots & \vdots & \vdots \\
\end{bmatrix} \]

where the last three columns of \( Z^* \) are \( \varphi_{1t} \), \( \varphi_{2t} \) and \( \varphi_{3t} \) respectively, then

\[ a = (Z^*Z^*)^{-1}(Z^*X), \]

where \( X \) is the data matrix.

Define

\[ a^+ = \begin{bmatrix}
a_1 & \ldots & \ldots \\
a_2 & \ldots & \ldots \\
a_3 & \ldots & \ldots \\
\end{bmatrix} \]

and

\[ Z^+ = \begin{bmatrix}
\varphi_{1t} & \varphi_{2t} & \varphi_{3t} \\
\vdots & \vdots & \vdots \\
\end{bmatrix} \]

then we have the adjusted data series of

\[ x^* = x - Z^+a^+. \]
8.5 NLMLE Specifications of the Model

(1) parameter list

| coefficients | $\beta_{12}$ | $\beta_{22}$ | $\beta_{38}$ | $\beta_{48}$ | $\beta_{55}$ | $\beta_{67}$ |
| NLMLE variables | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ |

| coefficients | $\gamma_{11}$ | $\gamma_{12}$ | $\gamma_{13}$ | $\gamma_{24}$ | $\gamma_{35}$ | $\gamma_{36}$ |
| NLMLE variables | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ | $v_{11}$ | $v_{12}$ |

| coefficients | $\gamma_{3(12)}$ | $\gamma_{47}$ | $\gamma_{4(13)}$ | $\gamma_{58}$ | $\gamma_{5(11)}$ | $\gamma_{69}$ |
| NLMLE variables | $v_{13}$ | $v_{14}$ | $v_{15}$ | $v_{16}$ | $v_{17}$ | $v_{18}$ |

| coefficients | $\gamma_{10}$ | $\gamma_{20}$ | $\gamma_{30}$ | $\gamma_{40}$ | $\gamma_{50}$ | $\gamma_{60}$ |
| NLMLE variables | $v_{19}$ | $v_{20}$ | $v_{21}$ | $v_{22}$ | $v_{23}$ | $v_{24}$ |

endogenous variables

| coefficients | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ |
| NLMLE variables | $v_{25}$ | $v_{26}$ | $v_{27}$ | $v_{28}$ | $v_{29}$ | $v_{30}$ |

exogenous variables

| coefficients | $z_1$ | $z_2$ | $z_3$ | $z_4$ | $z_5$ | $z_6$ | $z_7$ |
| NLMLE variables | $v_{31}$ | $v_{32}$ | $v_{33}$ | $v_{34}$ | $v_{35}$ | $v_{36}$ | $v_{37}$ |

| coefficients | $z_8$ | $z_9$ | $z_{10}$ | $z_{11}$ | $z_{12}$ | $z_{13}$ |
| NLMLE variables | $v_{38}$ | $v_{39}$ | $v_{40}$ | $v_{41}$ | $v_{42}$ | $v_{43}$ |

Stochastic equations

| coefficients | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ |
| NLMLE equations | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ |
The transformed stochastic equations become:

\[ F_1 = v_{25} - v_1 * v_{26} - v_7 * v_{31} - v_8 * v_{32} - v_9 * v_{33} - v_{19} \]

\[ F_2 = \exp v_{27} - v_2 * \exp v_{26} - v_{10} * \exp v_{34} - v_{20} \]

\[ F_3 = \exp v_{28} - v_3 * (\exp v_{25} + \exp v_{27} + \exp v_{28} + v_{29} - \exp v_{30} \]
\[ + v_{40} - v_{41}) - v_{11} * \exp v_{35} - v_{12} * \exp v_{36} - v_{13} * v_{42} - v_{21} \]

\[ F_4 = v_{29} - v_4 * (\exp v_{25} - \exp v_{27} + \exp v_{28} + v_{29} - \exp v_{30} + v_{40} \]
\[ - v_{41}) - v_{14} * v_{37} - v_{15} * v_{43} - v_{22} \]

\[ F_5 = \exp v_{30} - v_5 * v_{29} - v_{16} * \exp v_{38} - v_{17} * v_{41} - v_{23} \]

\[ F_6 = \exp v_{26} - v_6 * (\exp v_{25} + \exp v_{27} + \exp v_{28} + v_{29} - \exp v_{30} + v_{40} \]
\[ - v_{18} * \exp v_{39} - v_{24} \]

8.6 Results

For the BHHH procedure, the model converged after 116 iterations with 222 function evaluations (Tables la and lb).

log-likelihood function = -639.6143

CPU time = 743 seconds (CDC 7600)
## Table 1a

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Starting Values</th>
<th>Parameter Estimates</th>
<th>Standard Errors</th>
<th>T-Ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_{12}$</td>
<td>0.9432</td>
<td>0.6417</td>
<td>0.6947</td>
<td>0.9237</td>
</tr>
<tr>
<td>$\beta_{22}$</td>
<td>0.1458</td>
<td>0.1276</td>
<td>0.0750</td>
<td>1.7016</td>
</tr>
<tr>
<td>$\beta_{38}$</td>
<td>-0.0690</td>
<td>-0.0796</td>
<td>0.1416</td>
<td>-0.5621</td>
</tr>
<tr>
<td>$\beta_{48}$</td>
<td>0.1299</td>
<td>0.0092</td>
<td>0.0292</td>
<td>0.3148</td>
</tr>
<tr>
<td>$\beta_{55}$</td>
<td>0.5004</td>
<td>0.4491</td>
<td>0.1629</td>
<td>2.7574</td>
</tr>
<tr>
<td>$\beta_{67}$</td>
<td>0.2294</td>
<td>0.3503</td>
<td>0.1467</td>
<td>2.3881</td>
</tr>
<tr>
<td>$\gamma_{11}$</td>
<td>0.6484</td>
<td>0.7271</td>
<td>0.2101</td>
<td>3.4609</td>
</tr>
<tr>
<td>$\gamma_{12}$</td>
<td>-0.1872</td>
<td>-0.1279</td>
<td>0.3822</td>
<td>-0.3345</td>
</tr>
<tr>
<td>$\gamma_{13}$</td>
<td>0.1374</td>
<td>0.1115</td>
<td>0.0566</td>
<td>1.9703</td>
</tr>
<tr>
<td>$\gamma_{24}$</td>
<td>0.7733</td>
<td>0.8061</td>
<td>0.1160</td>
<td>6.9483</td>
</tr>
<tr>
<td>$\gamma_{35}$</td>
<td>0.7439</td>
<td>0.7855</td>
<td>0.3609</td>
<td>2.1762</td>
</tr>
<tr>
<td>$\gamma_{36}$</td>
<td>0.2584</td>
<td>0.1974</td>
<td>0.3520</td>
<td>0.5607</td>
</tr>
<tr>
<td>$\gamma_{3(12)}$</td>
<td>0.1042</td>
<td>0.1104</td>
<td>0.1320</td>
<td>0.8361</td>
</tr>
<tr>
<td>$\gamma_{47}$</td>
<td>0.4417</td>
<td>0.3360</td>
<td>0.2349</td>
<td>1.4302</td>
</tr>
<tr>
<td>$\gamma_{4(13)}$</td>
<td>0.2644</td>
<td>0.3580</td>
<td>0.2592</td>
<td>1.3814</td>
</tr>
<tr>
<td>$\gamma_{58}$</td>
<td>-4.8623</td>
<td>-4.6153</td>
<td>5.0589</td>
<td>-0.9123</td>
</tr>
<tr>
<td>$\gamma_{5(11)}$</td>
<td>0.2338</td>
<td>0.2420</td>
<td>0.0442</td>
<td>5.4709</td>
</tr>
<tr>
<td>$\gamma_{69}$</td>
<td>0.7127</td>
<td>0.5673</td>
<td>0.1587</td>
<td>3.5742</td>
</tr>
</tbody>
</table>

## Table 1b

<table>
<thead>
<tr>
<th>Constants</th>
<th>Starting Values</th>
<th>Estimated Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_{10}$</td>
<td>2.2292</td>
<td>-3.0199</td>
</tr>
<tr>
<td>$\gamma_{20}$</td>
<td>235.5893</td>
<td>191.2679</td>
</tr>
<tr>
<td>$\gamma_{30}$</td>
<td>16.0904</td>
<td>36.7463</td>
</tr>
<tr>
<td>$\gamma_{40}$</td>
<td>13.5268</td>
<td>96.1541</td>
</tr>
<tr>
<td>$\gamma_{50}$</td>
<td>483.5083</td>
<td>417.2354</td>
</tr>
<tr>
<td>$\gamma_{60}$</td>
<td>58.2551</td>
<td>48.3693</td>
</tr>
</tbody>
</table>
NLMLE Estimates of the model are:

\[ u_1 = \log C_d - 0.6417 \log Y_d - 0.7271 \log C_{d1} + 0.1279 \log C_{d2} - 0.115 N + 3.0199 \]

\[ u_2 = \exp(\log C_n) - 0.1276 \exp(\log Y_d) - 0.8061 \exp(\log C_{n1}) - 191.2679 \]

\[ u_3 = \exp(\log I) + 0.0796 \[ \exp(\log C_d) + \exp(\log C_n) + \exp(\log I) \]
+ Iv - \exp(\log M) + (G + X - T) - Y_1] - 0.7855 \exp(\log I_1) - 0.1974 \exp(\log I_2) - 0.1104 (Y_1 - Y_2) - 36.7463 \]

\[ u_4 = Iv - 0.0092 \[ \exp(\log C_d) + \exp(\log C_n) + \exp(\log I) + Iv \]
- \exp(\log (M) + (G + X - T) - Y_1] - 0.3360 Iv_1
- 0.3580 Iv_2 - 96.1541 \]

\[ u_5 = \exp(\log M) - 0.4491 Iv + 4.6153 \exp(\log P_m) - 0.2420 Y_1 - 417.2354 \]

\[ u_6 = \exp(\log Y_d) - 0.3503 \[ \exp(\log C_d) + \exp(\log C_n) + \exp(\log I) \]
+ Iv - \exp(\log M) + (G + X - T)] - 0.5673 \exp(\log Y_{d1}) - 48.3693 \]
For the method of GMP with analytical derivatives (NAG), the model converged after 242 iterations with 284 function evaluations.

log-likelihood function = -639.6141

CPU time = 2068 seconds (CDC 7600)

The estimated parameters (with the same starting values as BHHH) are shown in Table 2a and the estimated constants in Table 2b.

Table 2a

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{12}$</td>
<td>0.6401</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>0.1268</td>
</tr>
<tr>
<td>$b_{38}$</td>
<td>-0.0803</td>
</tr>
<tr>
<td>$b_{48}$</td>
<td>0.0092</td>
</tr>
<tr>
<td>$b_{55}$</td>
<td>0.4468</td>
</tr>
<tr>
<td>$b_{67}$</td>
<td>0.3523</td>
</tr>
<tr>
<td>$y_{11}$</td>
<td>0.7275</td>
</tr>
<tr>
<td>$y_{12}$</td>
<td>-0.1272</td>
</tr>
<tr>
<td>$y_{13}$</td>
<td>0.1114</td>
</tr>
<tr>
<td>$y_{24}$</td>
<td>0.8074</td>
</tr>
<tr>
<td>$y_{35}$</td>
<td>0.7864</td>
</tr>
<tr>
<td>$y_{36}$</td>
<td>0.1966</td>
</tr>
<tr>
<td>$y_{3(12)}$</td>
<td>0.1105</td>
</tr>
<tr>
<td>$y_{47}$</td>
<td>0.3365</td>
</tr>
<tr>
<td>$y_{4(13)}$</td>
<td>0.3568</td>
</tr>
<tr>
<td>$y_{58}$</td>
<td>-4.6243</td>
</tr>
<tr>
<td>$y_{5(11)}$</td>
<td>0.2420</td>
</tr>
<tr>
<td>$y_{69}$</td>
<td>0.5659</td>
</tr>
</tbody>
</table>
Table 2b

<table>
<thead>
<tr>
<th>Constants</th>
<th>Estimated Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_{10}$</td>
<td>-3.0168</td>
</tr>
<tr>
<td>$Y_{20}$</td>
<td>190.2829</td>
</tr>
<tr>
<td>$Y_{30}$</td>
<td>36.5368</td>
</tr>
<tr>
<td>$Y_{40}$</td>
<td>96.2071</td>
</tr>
<tr>
<td>$Y_{50}$</td>
<td>418.3268</td>
</tr>
<tr>
<td>$Y_{60}$</td>
<td>43.3531</td>
</tr>
</tbody>
</table>

From the results, if we compare the CPU times, the BHHH procedure seems to be a more efficient method than GMP. For this particular model, it is faster than GMP by a factor of $2^{\frac{1}{2}}$ in terms of computational time.

The log-likelihood function values and the parameter estimates of the two methods are reasonably close and give evidence that the model does converge to a strong local optimum. The time taken to converge for this model is sufficiently large as to discourage much experimentation with models of this size.
In this research, we have successfully developed a general differentiation program and applied it to non-linear econometric models. In particular, an estimation program (NLMLE) was written to estimate non-linear simultaneous equation systems. A serial version of this program was implemented on the CDC 7600 and ICL 2980 computers.

In order to improve the efficiency and decrease the computational time, a parallel version of the BHHH method was developed on the ICL DAP. In the course of this, three versions of the estimation program were designed which exploited the DAP architecture and at the same time improved the efficiency compared with the serial version. The three parallel versions were:

(i) A matrix mode implementation for models with up to 4096 observations.

(ii) A vector mode implementation for models up to 64 observations.

(iii) A matrix and vector modes implementation for models with between 65 and 128 observations.

For versions (ii) and (iii), the DAP was 'partitioned' into 64 and 32 'parallel processors', respectively, according to the number of observations. The advantage of doing this was to allow us to evaluate multiple step-sizes simultaneously in the line search during the optimisation procedure. Hence the number of function calls was only one per iteration; the algorithm also determines the best likelihood function value with the optimum step-size in one function call.
We have demonstrated that the DAP is relatively more efficient when the sample size of the model increases. The optimum performance was for models having close to 4096 observations. For this the DAP was approximately 30 times faster than the CDC 7600 and 127 times faster than the ICL 2980 computers. The most suitable application for such classes of models are panel data with large numbers of members of the sample in each cross-section.

Although inversions (ii) and (iii), the DAP time per function call was longer than the serial version, we were able to evaluate multiple step-sizes in the line search simultaneously. This meant we were able to reduce the number of iterations required for the model to converge since we could always locate the optimum step-size with the least function value (minimisation). If a model is not well behaved or the starting values of the parameter estimates are too far from the optimum, then it may take many iterations and function evaluations to converge, but on the DAP we showed that the time on each iteration could be much reduced by using multiple steps.

The serial version of the program worked well on the class of non-linear econometric models used to test the method. The function specification of the differentiation program enabled us to define any econometric functions whether they were non-linear in parameters, variables, or both. Arithmetic functions such as LOG, EXP, SINE, COSINE and ARC-TANGENT were provided. Thus it avoided the task of data transformation or parameter mapping.

The differentiation program provided analytic derivatives of the log-likelihood function which could be applied in a gradient-type or quasi-Newton type optimisation procedure. Also the modified one-dimensional line search procedure was very efficient compared with
other methods (see Chapter 2) and the various gradient stopping criteria helped to improve the efficiency of the estimation program.

We have also demonstrated that the BHHH method was relatively more efficient for the class of models we have tested. When the size of the model increased and the model became more complex, the BHHH method performed better than the GMP (analytical derivatives). So the BHHH method generally worked well when:

(i) the sample size was large;
(ii) the error variance $\sigma^2$ was relatively small;
(iii) the model was complex and the parameter set was reasonably large;
(iv) the model was correctly specified;
(v) the starting values $\theta^{(0)}$ must be close to $\theta^*$. 

Generally, for this class of models, the BHHH performed best, the GMP (with analytic derivatives) came second and GMP with numerical approximation to the derivatives performed worst. We recommend using BHHH or GMP if the analytic gradient of the log-likelihood function can be obtained easily. When the model is complex, BHHH would be a better choice.

Lastly, as we expected the BHHH method worked equally as well for other classes of model (see Chapter 8).

Thus we have written an efficient estimation program for general non-linear econometric models.

To improve the performance and efficiency of this program (perhaps for future research), we would suggest simplifying some derivative
functions generated from the differentiation program. This will avoid unnecessary repetitive calculations. Although we have done some simplification, these were relatively simple and trivial cases. A good, general and effective simplification routine would help to reduce the computational time substantially.

Another suggestion for the improvement is by using random directions in the DAP program. DAP is efficient at locating the optimum direction with the optimum step to give the optimum function value. All these could be done in parallel.

One final suggestion is to derive a completely new estimation procedure that would increase the degree of parallelism on the DAP. Research of this kind has been going on, especially on a variable-metric method.

It is our hope that ICL will build a 8 Mbyte DAP in the future. If this were done we could estimate a wider range of non-linear models, and perhaps by then, we could do more parallel computations on this kind of application.

It is also hoped that we can extend the present serial version of the estimation program, for example, by inserting some statistical tests. It might be worthwhile to program other parametric estimators such as non-linear three stage least squares.
Consider the Cholesky factors of a modified matrix \( H^{(k+1)} \) given by

\[
H^{(k+1)} = H^{(k)} + \Pi_1 z^{(k)} z^{(k)} + \Pi_2 \omega^{(k)} \omega^{(k)}
\]  

(A1)

We consider the case using the modification

\[
H^{(k+1)} = H^{(k)} + \sigma z^{(k)} z^{(k)}
\]
\[
= L^{(k)} D^{(k)} L^{(k)} + \sigma z^{(k)} z^{(k)}
\]  

(A2)

The diagonal elements of \( D \) will be denoted by \( d_1, d_2, \ldots, d_n \).

Given that the initial estimate of \( H \) is positive definite, then the succeeding estimates are also positive definite regardless of the rounding error incurred. Two points must be emphasised:

(a) Unless \( H^{(k+1)} \) is positive definite the Cholesky factorisation need not exist.

(b) Even if the Cholesky factorisation does not exist the numerical stability of any algorithm for modifying the triangular factors cannot be guaranteed when \( H^{(k+1)} \) is indefinite. Thus it is generally unsatisfactory to modify the Cholesky factors and then alter any negative elements a-posteriori since the accuracy of the factorisation could be in doubt.
Equation (A2) could be written as:

\[ H^{(k+1)} = L^{(k)} D^{(k)} (I + \sigma V V') D^{(k)} L^{(k)} \]  

(A3)

where

\[ L^{(k)} D^{(k)} V = z^{(k)} \]

To avoid the calculation of square roots, a new \( \tilde{V} \) is defined by the solution of the system of equations

\[ L^{(k)} \tilde{V} = z^{(k)} \]

so that

\[ \tilde{V} = D^{(k)} \tilde{V} \]

Equation (A3) can further be written in the form

\[ H^{(k+1)} = L^{(k)} D^{(k)} A^{(1)} A^{(1)} D^{(k)} L^{(k)} \]  

(A4)

where

\[ A^{(1)} = I - \sigma^{(1)} V^{(1)} V^{(1)} \]

Define \( V^{(j)} \) as the \((n-j+1) \times 1\) vector of the last \( n-j+1 \) elements of \( V \) \( (V^{(1)} = V) \). This implies that

\[ \sigma^{(1)} = \frac{1 \pm (1 + \sigma V' V)^{1/2}}{V' V} \]

provided the square root is real. The negative sign is always chosen since \( A^{(1)} \) is then positive definite. The equation \( \sigma^{(1)} \) can be
equivalently written as

$$\sigma^{(1)} = \frac{\sigma}{1 + \sigma \overline{y}[ D^{(k)}]^{-1} y}$$  \hspace{1cm} (A5)$$

In the next \(n-1\) steps \(A^{(1)}\) is reduced to lower-triangular form by a sequence of orthogonal matrices \(W^{(j)}\), \(j=1, 2, \ldots, n-1\), such that

$$\hat{L}^{(k)} = A^{(1)} W^{(1)} W^{(2)} \ldots W^{(n-1)}$$

where \(\hat{L}^{(k)}\) is a lower-triangular matrix. Such a reduction can be achieved by the use of elementary Hermitian matrices of the form

$$W^{(j)} = I - \tau^{(j)} \begin{bmatrix} \ell_{11} & \ell_{12} & \cdots & \ell_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ \ell_{n1} \end{bmatrix}$$

with \(u^{(j)}\) an \((n-j+1) \times 1\) vector and

$$\tau^{(j)} = \frac{2}{u^{(j)'} u^{(j)}}$$

For a general matrix \(A\), with elements \(a_{ij}\), the first stage of the reduction process is defined by the equations

$$u^{(1)'} = (a_{11} \pm \gamma^{(1)}, a_{12}, a_{13}, \ldots, a_{1n})$$

where

$$\gamma^{(1)^2} = \sum_{j=1}^{n} a_{ij}^2$$

and hence
\[ r^{(1)} = (\gamma^{(1)})^2 \pm \gamma^{(1)} a_{11} \]

The first row of \( A W^{(1)} \) is then of the form \((\pm \gamma^{(1)}, 0, \ldots, 0)\).

If these results are applied to the special case of the matrix \( A^{(1)} \) under consideration, the first stage of the desired reduction process is

\[ r^{(1)} = \pm \gamma^{(1)} \theta^{(1)} \]

where

\[ \theta^{(1)} = 1 - \sigma^{(1)} v_1^2 \]

\[ = 1 - \sigma^{(1)} d_1^{-1} v_1^2 \]

and

\[ \gamma^{(1)} = \sigma^{(1)} v_1^2 \sum_{j=2}^{n} v_j^2 + \theta^{(1)} \]

\[ = \sigma^{(1)} d_1^{-1} \sum_{j=2}^{n} d_j^{-1} v_j + \theta^{(1)} \]

The elements of \( u^{(1)} \) are by definition

\[ u_1^{(1)} = \theta^{(1)} \pm \gamma^{(1)} \]

\[ u_j^{(1)} = -\sigma^{(1)} v_1 v_j \quad j = 2, 3, \ldots, n \]

The sign before \( \gamma^{(1)} \) is always chosen to be the same as that of \( \theta^{(1)} \) so as to minimise rounding error.
$A^{(1)} W^{(1)}$ is of the form:

$$A^{(1)} W^{(1)} = \begin{bmatrix} \pm \gamma^{(1)} & 0 \\ \beta^{(1)} v^{(2)} & A^{(2)} \end{bmatrix}$$

where

$$A^{(2)} = I - \sigma^{(2)} v^{(2)} v^{(2)\prime}$$

with

$$\sigma^{(2)} = \sigma^{(1)} \left\{ 1 + v^{(1)} \tau^{(1)} \left[ (v^{(1)} + v^{(1)\prime} u^{(1)}) \right] \right\}$$

$$= \tau^{(1)} \sigma^{(1)} (1 \pm \gamma^{(1)})$$

Similarly, $\beta^{(1)}$ can be shown to be

$$\beta^{(1)} = -\sigma^{(1)} v^{(1)} + \tau^{(1)} \sigma^{(1)} v^{(1)} u^{(1)} + \mu^{(1)} \tau^{(1)} \sigma^{(1)} v^{(1)} u^{(1)} (A6)$$

$$= v^{(1)} u^{(1)\prime} \sigma^{(2)}$$

$$= \frac{d^{-1}}{\pm \gamma v \bar{v}} (1 - \gamma^2)$$

which can be written as

$$\beta^{(1)} = d^{-1} \bar{\beta}^{(1)}$$

Note that if $\bar{\nu} = 0$ then equation (A6) implies that $\bar{\beta}^{(1)} = 0$.

Also note that $\mu_j$ is an element of $u$. 
During the algorithm only $\tilde{\gamma}^{(1)}$ need be calculated. Post-multiplication by $W^{(j)}$, $j = 2, \ldots, n$, leaves the first row and column of $A^{(1)}W^{(1)}$ unaltered. Extending the definitions of $\tilde{\gamma}^{(1)}$ and $\gamma^{(1)}$ to $\tilde{\gamma}^{(j)}$ and $\gamma^{(j)}$ respectively, $H^{(k+1)}$ can be written as

$$H^{(k+1)} = L(k) D^{(k)} \tilde{L}(k) \tilde{L}^{(k)} D^{(k)} \tilde{L}^{(k)} L(k)',$$

where

$$\tilde{L}(k) = \begin{bmatrix}
1 & \pm \gamma^{(1)} & 0 & \cdots & 0 \\
0 & 1 & \pm \gamma^{(2)} & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{bmatrix} \begin{bmatrix}
\pm \gamma^{(1)} \\
\pm \gamma^{(2)} \\
\pm \gamma^{(3)} \\
\vdots \\
\pm \gamma^{(n)}
\end{bmatrix}.$$

which leads to

$$D^{(k)} \tilde{L}(k) = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix} \begin{bmatrix}
\pm \gamma^{(1)} d_1 \\
\pm \gamma^{(2)} d_2 \\
\vdots \\
\pm \gamma^{(n)} d_n
\end{bmatrix}.$$
This will be written as

\[ D^{(k)} L^{(k)} = L^{(k)} D^{(k+1)} L^{(k)} \]

Then

\[ H^{(k+1)} = L^{(k)} L^{(k)} D^{(k+1)} L^{(k)} D^{(k)} L^{(k)} \]

so that the required \( L^{(k+1)} \) is given by

\[ L^{(k+1)} = L^{(k)} L^{(k)} \]

The general \( L^{(k)} \) and \( L^{(k)} \) are dense lower-triangular matrices and straightforward multiplication would require \( n^3/6 + O(n^2) \) multiplications. The method described above requires \( 3n^2/2 + O(n) \) multiplications to obtain the modified factors.

It is clear that the matrix \( H^{(k+1)} \) will be positive definite unless \( \sigma^{(1)} \) given by equation (A5) is imaginary. In this case two possible strategies can be followed to give a positive definite of \( H^{(k+1)} \):

(i) The parameter \( \sigma \) can be made less negative.

(ii) The size of the diagonal elements of \( D \) can be increased.

A convenient modification of the formula for \( \sigma \) amount to alter \( H^{(k+1)} \) using (i) and (ii) can be determined from the expression

\[ \sigma^{(1)} = -\frac{\sigma}{1 + (1 + \sigma \bar{v}^T D^{(k)} \bar{v} - \bar{v})} \]
Appendix B

NLMLE Listing

Program FIMLX

Logical Iend

Logical Irec, Ltheta, Jacob

Logical Deriv, DerCuV, Success

Integer*8 Itime

Integer Iflag, Iresult

Common X(100, 50)

Common /A1/Nlist(4000), nprior(300), nprs(300), nDer(300, 60), Nadr(300, 1), Listen(20), Jumpad(20, 5), Jc(20), Cons(200)

Common /A2/ Vmu(100, 50), R(50, 50), D(50), Temp(50)

Common /A3/ LMax, Lmaxf, JMaxf, Lfunc(50), Jfunc(50), ICOUNT, DETF, DETJT

Common /A4/ NQ, NB, NI, MFUN, NT, K, Kvar, KPunc, KParam, IMeth, Istep, Nintf

Common /A5/ Ltheta, Jacob, Irec, MaxAD, MaxF, MaxCon, Ivec(70), Iadc, IIADCD, IFUNC, NFUNC, ITAN, NTAN, NFLAG, LDIFF, ICOn(4)

Common /A6/ Vlf(300, 100), NDers(20, 50), NDERJ(300, 60), NDERJS(20, 20)

Common /A7/ NDJ(10, 50, 10)

Common /A8/ RES(20, 20)

Common /Names/ NAME(20)

REAL NDers, NDERJS

Dimension V(24), VMus(24)

Dimension Delta(50), HesL(276), HESD(50), W(145), GR(50)

Dimension Itext(80)

Dimension Isym(200)

External BHHH, EO4EAZ

External FUNML, FUNSET, MONIT

Logical Loadld

Iflag = 0

IMETH = 0 BHHH

IMETH = 1 GMP ANALYTICAL DERIVATIVES

IMETH = 2 GMP NUMERICAL APPROXIMATION

ISTEP = 0 GSTEP

ISTEP = 1 BARD

Call Qmcmilltime(Iflag, Itime, Iresult)

Read(5, 10) IMETH, ISTEP

10 Format(2014)

MaxF = 0

NFunc = 0

KC = 0

IADC = 1

IRAN = 0

MaxCon = 0

Ltheta = .FALSE.

Jacob = .FALSE.

Read(5, 11) NB, NI, Nintf, NY, NZ, NParam, NT, NL, NR, Ntran

11 Format(2014)

Read(5, 151) (NAME(J), J = 1, NParam)

151 Format(20A4)

NQ = NB + NI

NA = NT - NL

NOAP1 = NL + 1

Call Input(NA, NOAP1, NVar, NR, NTRAN)

Write(6, 100) NQ, NA, NL, NParam

100 Format(1HO, I4, ' EQUATIONS', I8, ' OBSERVATIONS', I8, ' LAGS', I8, ' PARAMETERS')
IMETERS')

READ IN INITIAL VALUES OF PARAMETERS
PUT PARAMETERS, ENDOGENOUS VARIABLES IN VECTOR

READ(5,12) (V(I), I=1, NPARAM)
12 FORMAT(8F10.4)
IPARAM=NPARAM+NY
DO 105 I = 1, IPARAM
105 IVECT(I) = I
WRITE(6,107) (V(I), I=1, NPARAM)
107 FORMAT(1HO,' COEFFICIENTS */1HO, (8F16.5))
IPARAM=NPARAM+NY
NS=NQ
IF(NQ .LE. 20) NFUNC=20
NADMAX=1
NS=NQ
IF(NINTF .NE. 0) NS=NQ+NINTF
DO 106 N=1,NS
CALL RDCARD(ITEXT)
CALL FRML(ITEXT, NFUNC, N, IADC, IEND, MAXCON, MAXF, KC, ISYM)
LISTEN(N)=IADC
WRITE(6,200) (NLIST(J), J=NADMAX, IK)
200 FORMAT(1H ,3014)
NADMAX=IADC
IF(KC .EQ. 0) GO TO 104
JC(N)=KC
KC=0
GO TO 106
104 JC(N)=0
KC=0
106 CONTINUE
ICON(1)=MAXCON
NS=NQ
IF(NINTF .NE. 0) NS=NQ+NINTF
MAXAD=LISTEN(NS)
IF(IMITER .GT. 0) GO TO 405
CALL QMCMILLTIME(IIFLAG, ITIME, IRESULT)
WRITE(6,99) ITIME
CALL BHHH(NPARAM, V, VMUS)
CALL QMCMILLTIME(IIFLAG, ITIME, IRESULT)
WRITE(6,99) ITIME
STOP
405 N=NPARAM
SMALL=SQRT(XO2AAF(XTOL))
DO 410 I=1, NPARAM
410 DELTA(I)=SMALL
ICOUNT=1
IFAIL=0
LOADLD=.TRUE.
XTOL=0.001
ETA=0.9
STPMAX=1.0
IL=N*(N-1)/2
IW=7*N+1
IWW=6*N+1
MAXCAL=500
IPRINT=1
CALL QMCMILLTIME(IIFLAG,ITIME,IRESULT)
WRITE(6,99) ITIME
99 FORMAT(1H0,'TIME = ',120)
CALL FUNML(NPARAM,V,FUN)
ICOUNT=ICOUNT+1
FUN=FUN-0.8*(ABS(FUN))
IF(IMETH .GT. 1) CALL EO4CDF(NPARAM,V,FUN,GR,HESL,IL,HESD,LOADLD,IXTOL,DELTA,ETA,STPMAX,W,IW,FUNML,FUNSET,MONIT,IPRINT,MAXCAL,IIFAIL)
IF(IMETH .EQ. 1) CALL EO4DDF(NPARAM,V,FUN,VMUS,HESL,IL,HESD,LOADLD,IXTOL,EO4EAZ,ETA,STPMAX,W,IW,FUNML,BHHH,MONIT,IPRINT,LMAXCAL,IFAIL)
CALL QMCMILLTIME(IIFLAG,ITIME,IRESULT)
WRITE(6,99) ITIME
430 FORMAT(1HO,'DERIVATIVE FROM G-M'/1X,12F10.6)
IF(IFAIL .EQ. 0) GO TO 436
WRITE(6,431)
431 FORMAT(1HO,'ERROR EXITS FROM NAG ROUTINE BECAUSE')
GO TO(415,416,417,418),IFAIL
415 WRITE(6,432)
432 FORMAT(1HO,'A PARAMETER IS OUTSIDE 1ST EXPECTED RANGE')
GO TO 436
416 WRITE(6,433)
433 FORMAT(1HO,'MAXIMUM NUMBER OF ITERATIONS EXCEEDED')
GO TO 436
417 WRITE(6,434)
434 FORMAT(1HO,'THE ALGORITHM DOES NOT SEEM TO BE CONVERGING')
GO TO 436
418 WRITE(6,435)
435 FORMAT(1HO,'INITIAL SETTING OF FUNCTION SEEMS UNRELIABLE')
436 WRITE(6,437) (V(I),I=1,NPARAM)
437 FORMAT('PARAMETER ESTIMATES'/8X,10F12.6))
438 FORMAT(1HO,'LOG LIKELIHOOD FUNCTION = ',F12.6)
STOP
END

SUBROUTINE FUNSET(NPARAM,V,STEPC,FVEC)
DIMENSION V(NPARAM),FVEC(NPARAM),STEPC(NPARAM)
DO 20 I=1,NPARAM
STORE=V(I)
V(I)=V(I)+STEPC(I)
CALL FUNML(NPARAM,V,FUN)
FVEC(I)=FUN
V(I)=STORE
20 CONTINUE
RETURN
END
SUBROUTINE MONIT(NPARAM, V, FUN, GR, HES, IL, HED, NCALL)
DIMENSION V(NPARAM), GR(NPARAM), HED(NPARAM), HES(IL)
IF(NCALL .EQ. 0) GO TO 10
WRITE(6,2) NCALL
2 FORMAT(1HO, ' AFTER ', I5, ' FUNCTION CALLS')
GO TO 20
10 WRITE(6,1)
1  -FORMAT(1HO, ' FINAL SOLUTION IS ')
20 WRITE(6,3) (V(I), I=1, NPARAM)
3 FORMAT(1HO, ' PARAMETER ESTIMATES'/(8X, 10F10.6))
GNORM=0.0
BIG=HED(1)
SMALL=BIG
DO 30, I=1, NPARAM
  D1=HED(I)
  GNORM=GNORM+GR(I)*GR(I)
  IF(D1 .GT. BIG) BIG=D1
  IF(D1 .LT. SMALL) SMALL=D1
30 CONTINUE
GNORM=SQRT(GNORM)
WRITE(6,35) GNORM, BIG, SMALL
35 FORMAT(1HO, ' GRADIENT NORM ', F10.4, 2X, ' AND CONDITION NUMBER', +E10.4, 3X, E10.4)
WRITE(6,4) FUN
4 FORMAT(1HO, ' FUNCTION VALUE = ', F12.6/
RETURN
END
SUBROUTINE BHHH(NPARAM,V,VMUS)
LOGICAL IREC,LTHETA,JACOB
LOGICAL DERIV,DERCUV,SUCCESS
LOGICAL IFCK
COMMON X(100,50)
COMMON /A1/NLIST(4000),NPRIOR(300),NPRS(300),NDER(300,60),NADR(300,1),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTHETA,JACOB,IREC,MAMAD,MAXF,MAXCON,IVECT(70),IADC,IIADCD,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)
COMMON /A4/ NQ,NB,NI,MFUN,NT,K,KVAR,KFUNC,KPARAM,IMETH,ISTEP,NINTF
COMMON /A5/ LTHETA,JACOB,IREC,MAXAD,MAXF,MAXCON,IVECT(70),IADC,IIADCD,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)
COMMON /A6/ VLP(300,100),NDERS(20,50),NDERJ(300,60),NDERJS(20,20)
COMMON /A7/ NDJ(10,50,10)
COMMON /A8/ RES(20,20)
COMMON /NAMES/ NAME(20)
REAL NDERS,NDERJS
EXTERNAL FUNML
DIMENSION V(NPARAM),VMUS(NPARAM)
DIMENSION SER(50),TRA(50)
EE=1.E-10
IMAX=300
STEP=1.0
TOL=1.E-3
LTHETA=.FALSE.
JACOB=.FALSE.
MFUN=0
MAXSQZ=10
JSQZ=0
SUCCESS=.FALSE.
KPARAM=NPARAM+NQ
IF(IMETH .GT. 0 .AND. ICOUNT .GT. 1) GO TO 999

DIFFERENTIATE W.R.T. ENDOGENOUS VARIABLES TO GET JACOBIAN

CALL DIFIML(NPARAM,V)
IF(NQ .GT. 10 .OR. KPARAM .GT. 50) GO TO 999

DIFFERENTIATE W.R.T. PARAMETERS

DO 150 K1=1,NPARAM
K=K1
LTHETA=.TRUE.
CALL DIFIML(NPARAM,V)

DIFFERENTIATE JACOBIAN W.R.T. PARAMETER

JACOB=.TRUE.
CALL DIFIML(NPARAM,V)
JACOB=.FALSE.
150 CONTINUE

EVALUATE ALL DERIVATIVES AND EQUATIONS AT EACH TIME PERIOD

999 DO 1000 ITE=1,IMAX
IF(IMETH .GT. 0) GO TO 158
ICOUNT=ITE
WRITE(6,1001) ITE
1001 FORMAT(1HO,' ITERATION NUMBER ',13)
IF(ITE .GT. 1) GO TO 175
158 CALL FUNML(NPARAM,V,FV)
FUN=FV
GO TO 176
175 FUN=FUNNEW
176 DO 170 K1=1,NPARAM
SUM=0.
DO 180 IT=1,NT
180 SUM=SUM+VMU(IT,K1)
VMUS(K1)=SUM
170 CONTINUE
IF(IMETH .GT. 0) RETURN
WRITE(6,800)
800 FORMAT(1HO,'GRADIENT VMUS')
WRITE(6,801) (VMUS(J), J=1,NPARAM)
801 FORMAT(1H ,8(E12.6,2X))
VNORM=0.0
DO 288 I=1,NPARAM
288 VNORM=VNORM+VMUS(I)*VMUS(I)
VNORM=SQR(T(VNORM))
WRITE(6,289) VNORM
289 FORMAT(1HO,'GRADIENT NORM * ',F12.6)
IF(VNORM .LE. 1.E-4) GO TO 1200
IF(ITE .EQ. 1 .AND. NPARAM .LE. 10) CALL GCHECK(NPARAM,V,FUN,FUNML)
C
C COMPUTE RIJ=VMU(PRIME)VMU
C
DD=0.
DO 195 I=1,NPARAM
DO 190 J=1,NPARAM
SUM=0.
DO 185 M=1,NT
185 SUM=SUM+VMU(M,I)*VMU(M,J)
R(I,J)=SUM
190 CONTINUE
195 CONTINUE
IF(NPARAM .GT. 10) GO TO 501
WRITE(6,802)
802 FORMAT(1HO,' HESSIAN MATRIX RIJ')
DO 302 I=1,NPARAM
302 WRITE(6,109) (R(I,J),J=1,I)
109 FORMAT(1H ,8(E12.6,2X))
302 CONTINUE
501 CALL INVERT(R,NPARAM,DET)
IF(NPARAM .GT. 10) GO TO 502
WRITE(6,804)
502 FORMAT(1HO,'RIJ (INVERSE)')
DO 303 I=1,NPARAM
303 WRITE(6,109) (R(I,J),J=1,I)
303 CONTINUE
C
C COMPUTE DIRECTION D
DO 200 I=1,NPARAM
  SUM=0.
  DO 210 J=1,NPARAM
    SUM=SUM+R(I,J)*VMUS(J)
  D(I)=SUM
  DD=DD+D(I)*D(I)
  200 CONTINUE
  DD=SQRT(DD)

TEST FOR CONVERGENCE OF PARAMETER VECTOR THETA

DO 270 I=1,NPARAM
  IF(ABS(D(I)) .GT. TOL*(ABS(V(I))+TOL*10.)) GO TO 600
  270 CONTINUE
  SUCCESS=.TRUE.
  IF(ISTEP .EQ. 1) GO TO 700
  GO TO 685

CONVERGENCE NOT ACHIEVED, DO LINEAR STEPSIZE SEARCH USING
B+ALAMB*D VECTOR

ALAMB=AMIN1(1.0,STEP/DD)
DO 275 I=1,NPARAM
  TEMP(I)=V(I)
  275 CONTINUE
  GRAD=0.
  DO 280 I=1,NPARAM
    GRAD=GRAD+D(I)*VMUS(I)
  280 CONTINUE
  IF(ISTEP .EQ. 0) GO TO 680
  CALL BARD(FUN,FUNNEW,GRAD,ALAMB,MAXSQZ,IFOK,NSQZ,JSQZ,V,
            INPARAM,FUNML)
  700 WRITE(6,2000) FUN,FUNNEW,NSQZ,ALAMB,GRAD
  2000 FORMAT( 'FUN=',F12.5,4X,'FUNNEW=',F12.5,4X,'NSQZ=',I4,'STEPSIZE=',
             I,E12.5,4X,'GRAD=',E12.5)
  GO TO 690
  CALL GSTEP(FUN,FUNNEW,GRAD,ALAMB,NPARAM,V,FUNML,IFOK)
  685 WRITE(6,1999) FUN,FUNNEW,ALAMB,GRAD,MPUN
  1999 FORMAT( 'FUN=',F12.6,4X,'FUNNEW=',F12.6,4X,'STEPSIZE=',E12.6,
          +4X,'GRAD=',E12.6,4X,'MPUN=',I4)
  DD=ALAMB*DD
  SN=AMAX1(STEP/2.0,2.0*DD)
  STEP=AMIN1(SN,2.0*STEP)
  690 WRITE(6,2001) (D(I),I=1,NPARAM)
  2001 FORMAT( 'DIRECTION'/8X,10E12.5))
  WRITE(6,2002) (V(I),I=1,NPARAM)
  2002 FORMAT( 'PARAMETER ESTIMATES'/8X,10E12.5))
  IF(SUCCESS )GO TO 1200
  IF(.NOT. IFOK) GO TO 1500
  IF(ISTEP .EQ. 0) GO TO 1000
  DO 710 I=1,NPARAM
  V(I)=TEMP(I)-ALAMB*D(I)
  710 CONTINUE
  WRITE(6,2003) ITE
  2003 FORMAT('/CONVERGENCE NOT ACHIEVED AFTER',I4,'ITERATIONS.'/)
  GO TO 1026
CONVERGENCE

1200 WRITE(6,2004) ITE
2004 FORMAT(' CONVERGENCE ACHIEVED AFTER ',I4, ' ITERATIONS.')
1026 WRITE(6,2005) MFUN
2005 FORMAT(' NUMBER OF FUNCTION EVALUATIONS = ',I5)
    IF(ITE .GT. 1) FUN=FUN+NEW
    WRITE(6,2006) FUN
2006 FORMAT(' LOG LIKELIHOOD FUNCTION = ',F15.7)

COMPUTE STD-ERRORS AND T-RATIOS

OUTPUT STATISTICS

DO 3080 J= 1 , NPARAM
    SER(J)=SQRT(R(J,J))
    IF(.NOT. IFOK) V(J)=TEMP(J)
    TRA(J)=V(J)/SER(J)
3080 CONTINUE

WRITE(6,3100)
3100 FORMAT(1HO,5X,'PARAMETERS',10X,'STD-ERRORS',10X,'T-RATIOS')
WRITE(6,3110)
3110 FORMAT(1H ,5X,'-------------',10X,'----------',10X,'---------')
    DO 3115 J=1 , NPARAM
        WRITE(6,3120) NAME(J),V(J),SER(J),TRA(J)
3120 FORMAT(1H ,A4,2X,F12.6,8X,F12.6,7X,F12.6)
3115 CONTINUE

WRITE(6,3225)
3225 FORMAT(' RESIDUAL SUM OF SQUARES')
WRITE(6,3226)
3226 FORMAT(1H ,1X,10F10.4)
    DO 3227 I=1,NB
        WRITE(6,3228) (RES(I,J) , J= 1 , 1)
3228 FORMAT(1H ,5X,10F10.4)
3227 CONTINUE

WRITE(6,3311)
3311 FORMAT(' VARIANCE-COVARIANCE MATRIX')
WRITE(6,3312)
3312 FORMAT(1H ,10X,'---------------------')
    DO 3313 I=1,NPARAM
        WRITE(6,3314) (R(I,J) , J=1,I)
3314 FORMAT(1H ,10F12.6)
3313 CONTINUE
RETURN

FAILURE TO IMPROVE FUNCTION VALUE

1500 WRITE(6,2007) ITE
2007 FORMAT(/' FAILURE TO IMPROVE LIKELIHOOD FUNCTION AFTER ',I4, ' ITERATIONS.')
    GO TO 1026
END
SUBROUTINE BARD(FUN, FO, GRAD, RM, MAXSQZ, IFOK, NSQZ, JSQZ, V, NPARAM, IFUNCT)
COMMON X(100,50)
COMMON /A1/ NLIST(4000), NPRIOR(300), NPRS(300), NDER(300,60), INADR(300), LISTEN(20), JUMPAD(20,5), JC(20), CONS(200)
COMMON /A2/ VMU(100,50), R(50,50), D(50), TEMP(50)
COMMON /A3/ LTMAX, LMAXF, JTMAXF, LTFUNC(50), JMAXF, JFUNC(50), IICOUNT, DETF, DETJT
COMMON /A4/ NQ, NB, NI, MFUN, NT, K, KVAR, KFUNC, KPARAM, IMETH, ISTEP, NINTF
COMMON /A5/ LTHETA, JACOB, IREC, MAXAD, MAXF, MAXCON, IVECT(70), IADC, IAADC, IFUNC, NFUNC, ITAN, NTAN, NFLAG, LDIFF, ICON(4)
COMMON /A6/ VLF(300,100), NDERS(20,50), NDERJ(300,60), NDERJS(20,20)
COMMON /A7/ NDJ(10,50,10)
REAL NDERJS, NDERS
EXTERNAL FUNCT
DIMENSION V(NPARAM)
LOGICAL LTHETA, JACOB, IFOK
IFOK=.TRUE.
RMAX=1.E2
KFLAG=0
IFLAG=0
NSQZ=0
RO=1.0
STEP=RO
DO 140 J=1,NPARAM
V(J)=TEMP(J)-STEP*D(J)
CALL FUNCT(NPARAM, V, FV)
IF(KFLAG .EQ. 1) GO TO 13:
IF(IFLAG .EQ. 1) GO TO 15
FO=FV
R1=-GRAD*RO**2/(2.*FO-2.*FUN-2.*GRAD*RO)
WRITE(6,9000) GRAD, RO, FO, R1
9000 FORMAT(' GRADIENT=' ,F12.7,' OLD STEPSIZE=' ,E15.7,' OLD F=' ,E15.7,
I' NEW STEPSIZE=' ,E15.7)
IF(FO .LT. FUN .AND. NSQZ .EQ. 0) GO TO 10
IF(FO .LT. FUN) GO TO 13
R0=AMAX1(.25*RO,AMIN1(.75*R0,R1)) : NSQZ=NSQZ+1
JSQZ=JSQZ+1
IF(NSQZ .LE. MAXSQZ) GO TO 21
IFOK=.FALSE.
RM=RO
RETURN
10 RR=RO
R3=AMIN1(R1,.75*RMAX)
IF(ABS(R3-RO) .LE. .1*RO .OR. R1 .LE. .25*RO) GO TO 11
KFLAG=0
RO=R3
IFLAG=1
GO TO 2
15 F3=FO
NSQZ=NSQZ+1
JSQZ=JSQZ+1
IF(F3 .LT. FO) GO TO 12
11 RO=RR
   IFLAG=0
   KFLAG=1
   GO TO 2
13 RM=RO
   FO=FV
   RETURN
12 RM=R3
   FO=F3
   RETURN
END
SUBROUTINE DIFIML(NPARAM,V)  
LOGICAL IREC,LTHETA,JACOB  
LOGICAL DERIV,DERCUV,SUCCESS  
COMMON X(100,50)  
COMMON /A1/NLIST(4000),NPRIOR(300),NPRS(300),NDER(300,60),NADR(300,1),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)  
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)  
COMMON /A3/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNC(50),ICOUNT,DETF,DETJT  
COMMON /A4/ NQ,NB,NI,MFUN,NT,K,KVAR,KFUNC,KPARAM,IMETH,ISTEP,NINTF  
COMMON /A5/ LTHETA,JACOB,IREC,MAXAD,MAXF,MAXCON,IVECT(70),IADC,IIADC,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)  
COMMON /A6/ VLF(300,100),NDERS(20,50),NDERJ(300,60),NDERJS(20,20)  
COMMON /A7/ NDJ(10,50,10)  
REAL NDERS,NDFERJS  
DIMENSION V(NPARAM)  
L=1  
NDIFF=0  
NNF=0  
ITAN=0  
LDIFF=0  
IF(.NOT. LTHETA) NTAN=0  
LFLAG=0  
IF(.NOT. LTHETA) NFLAG=0  
JACOB IS TRUE FOR DIFFERENTIATING JACOBIAN W.R. T. PARAMETER  
LTHETA IS TRUE FOR DIFFERENTIATING W.R.T. PARAMETER  
LTHETA=FALSE FOR DIFFERENTIATING W.R.T. ENDOGENOUS VARIABLE  
IF(JACOB) GO TO 300  
NS=NQ  
IF(NINTF .NE. 0) NS=NQ+NINTF  
NJ=NS  
IF(NFLAG .EQ. 1 .AND. LTHETA) NS=NS+NTAN  
DO 185 I=1,NS  
IF(LTHETA .AND. NFLAG .EQ. 1 .AND. I .GT. NJ) GO TO 190  
IF(ICOUNT .GT. 1) GO TO 194  
NADMAX=LISTEN(I)  
WRITE(6,195) I,(NLIST(J),J=L,NADMAX)  
195 FORMAT(1HOEQUATION *,I4,/20I4)  
194 CALL PRIOR(NLIST,I,L,NPRIOR,NPRS,IREC,IFUNC,NDIFF,ICT)  
IF(LTHETA) GO TO 190  
198 IADC=NADR(IFUNC)+1  
DO 222 IDIFF=1,NDIFF  
DO 225 IVAL=1,NQ  
IF(LTHETA) GO TO 224  
IVAR=IVECT(NPARAM+IVAL)  
224 IADC=MAXAD+1  
NFUNC=NFUNC+1  
MAXF=MAXF+1  
NPRIOR(MAXF)=NFUNC  
NPRS(NFUNC)=MAXF  
NDER(IFUNC,IVAR)=NFUNC+1000  
NADR(NFUNC)=IADC  
CALL DIFF(IVAR,I)  
IF(LTHETA) GO TO 265  
IF(ITAN .EQ. 0) GO TO 250
LFLAG=1
IF(.NOT. LTHETA .AND. LFLAG .EQ. 1) NFLAG=LFLAG
NPOINT=NPRS(NS)+NTAN
NMOV=(MAXF-NPOINT)+NTAN
JP=MAXF
DO 246 JJ=1,NMOV
NPT=NPRIOR(JP)
NPRIOR(JP+1)=NPT
JP=JP-1
NPRS(NPT)=NPRS(NPT)+1
246 CONTINUE
MAXF=MAXF+1
NPRIOR(JP+1)=NFUNC
NPRS(NFUNC)=JP+1
ITAN=0
250 IADC=NADR(IFUNC)+1
225 CONTINUE
265 IF(.NOT. IREC) GO TO 270
ICT=ICT+1
IFUNC=NPRIOR(ICT)
IADC=NADR(IFUNC)+1
222 CONTINUE

COMPLETE DIFFERENTIATING ALL RECURSIVE FUNCTIONS,
NOW RETURN TO DIFFERENTIATE FUNCTION I

IREC=.FALSE.
IFUNC=I
GO TO 198
190 IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 193
IF(K .GT. 1) GO TO 196
IF(I .GT. 1) GO TO 197
LMAXF=MAXF
LMAX=MAXAD
LMAXF=NFUNC
LTCONC=MAXCON
GO TO 197
196 IF(I .GT. 1) GO TO 197
MAXAD=LMAX
MAXF=LMAXF
NFUNC=LMAXF
MAXCON=LTCONC
GO TO 197
193 IF(I .GT. 1) GO TO 197
LTFUNC(K)=NFUNC
197 IVAR=IVECT(K)
GO TO 198
270 IF(LTHETA .AND. NFLAG .EQ. 1 .AND. I .GE. NJ) GO TO 271
L=LISTEN(I)+1
GO TO 272
271 NNF=NNF+1
IF(NNF .GT. NTAN) RETURN
NPI=NPRS(NJ)
IFUNC=NPRIOR(NPI+NNF)
L=NADR(IFUNC)+1
IREC=.FALSE.
LDIFF=1
NDIFF=1
GO TO 273
272 NDIFF=0
273 NBB=NB
IF(NINTF .NE. 0) NBB=NB+NINTF
IF(NFLAG .EQ. 1) NBB=NB+NTAN
IF(LTHETA .AND. I .EQ. NBB) RETURN
185 CONTINUE
IF(.NOT. LTHETA) ICON(2)=MAXCON
IF(LTHETA) ICON(3)=MAXCON
RETURN

NOW DIFFERENTIATE JT W.R.T. THETA

300 IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 301
JTMAXF=NFUNC
JMAXF=MAXF
GO TO 302
301 JFUNC(K)=NFUNC

GET DERIVATIVE FUNCTION AND ITS HEAD ADDRESS

302 NDIFF=0
KDIFF=0
L=1
IF(NFLAG .EQ. 1) NBB=NB+NTAN
DO 310 I=1,NBB
KFLAG=0
CALL PRI0R(NLIST,I,L,NPRIOR,NPRS,IREC,IFUNC,NDIFF,ICT)
IF(IREC) KFLAG=1
NP0INT=ICT
KDIFF=NDIFF
DO 320 J=1,NQ
NDIFF=NDIFF+1
DO 315 IDIFF=1 ,NDIFF
IVAR=IVECT(NPARAM+J)
KFUNC=NDER(IFUNC,IVAR)
IF(KFUNC .LE. 1000) GO TO 325
KFUNC=KFUNC-1000
IADC=NADR(KFUNC)+1
IADCD=MAXAD+1
NFUNC=NFUNC+1
MAXF=MAXF+1
NPRIOR(MAXF)=NFUNC
NPRS(NFUNC)=MAXF
NADR(NFUNC)=IADCD
IF(IREC) NDER(KFUNC,K)=NFUNC+1000
IF(.NOT. IREC .AND. NFLAG .EQ. 1 .AND. I .LE. NINTF) NDER(KFUNC,K)=NFUNC+1000
IF(KFUNC .LE. 1000) GO TO 325
KVAR=J
CALL DIFF(IVAR,I)
IF(.NOT. IREC) GO TO 340
GO TO 330
325 IF(.NOT. IREC) GO TO 335
IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 321
330 NDERJ(IFUNC,J)=0
321 ICT=ICT+1
IFUNC=NPRIOR(ICT)
IF(IDIFF .EQ. KDIFF) IREC=.FALSE.
315 CONTINUE
GO TO 340
335 IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 322
NDERJ(IFUNC,J)=0
GO TO 340
322 NDJ(I,K,J)=0
340 ICT=NPOINT
NDERF=KDIFF
IFUNC=NPRIOR(ICT)
IF(KFLAG .EQ. 1) IREC=.TRUE.
320 CONTINUE
L=LISTEN(I)+1
NDERF=0
KDIFF=0
310 CONTINUE
ICON(4)=MAXCON
RETURN
END
SUBROUTINE DIEVAL(NPARAM,V)
LOGICAL IREC,LTHETA,JACOB
LOGICAL DERIV,DERCUV,SUCCESS
COMMON X(100,50)
COMMON /A1/NLIST(4000),NPRIOR(300),NPRS(300),NDER(300,60),NADR(300,1),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTMAX,LMAXF,JMAXF,LTFUNC(50),JMAXF,IFUNC(50),
ICOUNT,DETF,DETTJ
COMMON /A4/ NLMAXF,JTMAXF,JTMAXF,LTFUNC(50),JMAXF,IFUNC(50),
ICOUNT,DETF,DETTJ
COMMON /A5/ LTHETA,JACOB,IREC,MAXAD,MAXF,MAXCON,Ivec(70),IAJC
IIADC,IFUNC,NFNC,ITAN,NTAN,LDIFF,ICON(4)
COMMON /A6/ VLF(300,100),NDER(20,50),NDERJ(300,60),NDERJS(20,20)
COMMON /A7/ NDJ(10,50,10)
REAL NDERS,NDERJS
DIMENSION V(NPARAM)
DIMENSION VFLIST(200)

COMPUTE F AND F(PRIME)F
DO 380 IT=1,NT
L=1
NDIFF=0
NS=NB
IF(NINTF .NE. 0) NS=NB+NINTF
DO 400 I=1,NS
CALL PRIOR(NLIST,I,L,NPRIOR,NPRS,IREC,IFUNC,NDIFF,ICT)
VFUNC=0.
410 IADC=NADR(IFUNC)+1
IF(IREC) GO TO 420
CALL EVAL(IADC,I,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)
VFLIST(IFUNC)=VFUNC
VLF(I,IT)=VFUNC
GO TO 430
420 DO 425 J=1,NDIFF
CALL EVAL(IADC,I,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)
VFLIST(IFUNC)=VFUNC
VFUNC=0.
ICT=ICT+1
IFUNC=NPRIOR(ICT)
IADC=NADR(IFUNC)+1
425 CONTINUE
IREC=.FALSE.
IFUNC=I
GO TO 410
430 L=LISTEN(I)+1
NDIFF=0
400 CONTINUE

NOW COMPUTE JT,GET FUNCTION AND ITS HEAD ADDRESS
IF(NTAN .EQ. 0) GO TO 440
VFUNC=0.
ICT=NPRS(NS)+1
DO 442 I=1,NTAN
IFUNC=NPRIOR(ICT)
IADC=NADR(IFUNC)+1
CALL EVAL(IADC,I,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)
VFLIST(IFUNC)=VFUNC
VFUNC=0.
ICT=ICT+1
CONTINUE

NS=NQ
IF(NINTF .NE. 0) NS=NQ+NINTF
ICT=NPRIOR(NS)
IF(NTAN .GT. 0) ICT=ICT+NTAN
IFUNC=NPRIOR(ICT+1)
LTHETA=.FALSE.
JACOB=.FALSE.
IREC=.FALSE.
DO 450 I=1,NS
450 DO 455 J=1,NQ
  VFUNC=0.
  IVAR=IVECT(NPARAM+J)
  NFUNC=NDER(I,IVAR)
  IF(NFUNC .LE. 1000) GO TO 480
  NFUNC=NFUNC-1000
IF(IFUNC .EQ. NFUNC) GO TO 465
  ICT=ICT+1
  IREC=.TRUE.
  IADC=NADR(IFUNC)+1
  GO TO 470
465 IREC=.FALSE.
  IADC=NADR(IFUNC)+1
  ICT=ICT+1
470 CALL EVAL(IADC,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)
  VFLIST(IFUNC)=VFUNC
  IF(IREC) GO TO 490
  VFLIST(IFUNC,IT)=VFUNC
  IFUNC=NPRIOR(ICT+1)
  GO TO 455
490 IFUNC=NPRIOR(ICT+1)
  VFUNC=0.
  GO TO 460
480 IF(NFUNC .EQ. 0) VFUNC=0.
  IF(NFUNC .EQ. 1) VFUNC=1.
  IF(NFUNC .EQ. -1) VFUNC=-1.
  IF(NFUNC .GT. 1) GO TO 485
  IF(NFUNC .LT. -1) GO TO 486
  GO TO 487
485 VFUNC=CONS(NFUNC-1)
  GO TO 487
486 VFUNC=-CONS(IABS(NFUNC)-1)
487 NDERES(I,IVAR)=VFUNC
CONTINUE
  IF(.NOT. IREC) GO TO 450
  IFUNC=NPRIOR(ICT+1)
  GO TO 452
CONTINUE
  IF(IMETH .GT. 1 ) GO TO 380
C
C EVALUATE PARTIAL D(FI)/D(THETA(K))
GET FUNCTION NUMBER AND ITS HEAD ADDRESS

IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 505
ICT=NPRS(LMAXF+1)
GO TO 507

505 ICT=NPRS(LTFUNC(K)+1)
507 IFUNC=NPRIOR(ICT)
LTHETA=.TRUE.
JACOB=.FALSE.
NS=NB
IF(NINTF .NE. 0) NS=NB+NINTF
DO 500 I=1,NS
DERCUV=.FALSE.
VFUNC=0.
NFUNC=NDER(I,K)
IVAR=K
IF(NFUNC .LE. 1000) GO TO 526
NFUNC=NFUNC-1000

508 IF(IFUNC .EQ. NFUNC) GO TO 510
IREC=.TRUE.
IF(ICT .EQ. MAXF) GO TO 509
IC=NPRIOR(ICT)
ICN=NPRIOR(ICT+1)
IF(ICT .GT. ICN) GO TO 566
509 IADC=NADR(IFUNC)+1
GO TO 520

566 IADC=NADR(IFUNC)+1
DERCUV=.TRUE.
510 IREC=.FALSE.
IADC=NADR(IFUNC)+1
520 CALL EVAL(IADC,I,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)
IF(.NOT. DERCUV) GO TO 567
VFLIST(IFUNC)=VFUNC
VFUNC=0.
ICT=ICT+1
IFUNC=NPRIOR(ICT)
DERCUV=.FALSE.
IADC=NADR(IFUNC)+1
GO TO 520

567 IF(IREC) GO TO 525
VFLIST(IFUNC)=VFUNC
VLF(IFUNC,IT)=VFUNC
IF(ICT .GT. MAXF) GO TO 500
ICT=ICT+1
IFUNC=NPRIOR(ICT)
GO TO 500

525 VFLIST(IFUNC)=VFUNC
VFUNC=0.
ICT=ICT+1
IFUNC=NPRIOR(ICT)
IADC=NADR(IFUNC)+1
GO TO 508

526 IF(NFUNC .EQ. 0) VFUNC=0.
IF(NFUNC .EQ. 1) VFUNC=1.
IF(NFUNC .EQ. -1) VFUNC=-1.
IF(NFUNC .GT. 1) GO TO 527
IF(NFUNC .LT. -1) GO TO 528
GO TO 530

VFUNC=CONS(NFUNC-1)
GO TO 530

VFUNC=-CONS(IABS(NFUNC)-1)
NDERS(I,IVAR)=VFUNC
CONTINUE

IF(NFLAG .EQ. 0) GO TO 544
ICT=NPRS(NS)
NS=NS+NTAN
DO 540 I=1,NTAN
VFUNC=0.
ICT=ICT+1
IFUNC=NPRIOR(ICT)
VFUNC=NDER(IFUNC,K)
IF(NFUNC .GT. 1000) GO TO 543
GO TO 540

VFUNC=VFUNC-1000
IADC=NADR(NFUNC)+1
CALL EVAL(IADC,I,IFUNC,VFUNC,IT,MAXAD,NPARAM,V,VFLIST)

VFLIST(NFUNC)=VFUNC
CONTINUE

CONTINUE

NOW EVALUATE CROSS PARTIAL DERIVATIVE OF JT W.R.T. THETA
GET FUNCTION NUMBER AND ITS HEAD ADDRESS

IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 545
IF(JMAXF .EQ. MAXF) GO TO 551
ICT=NPRS(JTMAXF+1)
IFUNC=NPRIOR(ICT)
GO TO 551

IF(K .EQ. NPARAM) GO TO 553
JF=LTFUNC(K)+1
GO TO 554

IF(NPRS(JFUNC(K)) .EQ. MAXF) GO TO 551
JF=LTFUNC(K)+1
IFUNC=NPRIOR(ICT)

LTHETA=.FALSE.
JACOB=.TRUE.
IF(NFLAG .EQ. 1) NS=NS-NTAN
DO 550 I=1,NS
IREC=.FALSE.

DO 560 J=1,NQ
VFUNC=0.
IF(NQ .LE. 10 .OR. KPARAM .LE. 50) GO TO 556
NFUNC=NDERJ(I,J)
GO TO 557

NFUNC=NDJ(I,K,J)
IF(NFUNC .LE. 1000) GO TO 580
NFUNC=NFUNC-1000
IF(IPUNC .EQ. NFUNC) GO TO 565
IREC=.TRUE.
IADC=IADR(IFUNC)+1
GO TO 570
565 IREC=.FALSE.
IADC=IADR(IFUNC)+1
570 CALL EVAL(IADC, I, IFUNC, VFUNC, IT, MAXAD, NPARAM, V, VFLIST)
VFLIST(IFUNC)=VFUNC
IF(IREC)GO TO 575
VLF(IFUNC,IT)=VFUNC
IF(IT.EQ. MAXF) GO TO 560
ICT=ICT+1
IFUNC=NPRIOR(ICT)
GO TO 560
575 ICT=ICT+1
IFUNC=NPRIOR(ICT)
VFUNC=0.
GO TO 555
580 IF(NFUNC .EQ. 0) VFUNC=0.
IF(NFUNC .EQ. 1) VFUNC=1.
IF(NFUNC .EQ. -1) VFUNC=-1.
IF(NFUNC .GT. 1) GO TO 582
IF(NFUNC .LT. -1) GO TO 585
GO TO 586
582 VFUNC=CONS(NFUNC-1)
GO TO 586
585 VFUNC=-CONS(IABS(NFUNC)-1)
586 NDERJS(I, J)=VFUNC
560 CONTINUE
IF(.NOT. IREC) GO TO 550
IFUNC=NPRIOR(ICT+1)
GO TO 552
550 CONTINUE
380 CONTINUE
RETURN
END
SUBROUTINE PQEVAL(NPARAM,V)
LOGICAL IREC,ILTHETA,JACOB
LOGICAL DERIV,DERCUV,SUCCESS
COMMON X(100,50)
COMMON /A1/NLIST(4000),NPRIOR(300),NPRS(300),NDER(300,60),NADR(300,1),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNC(50),ICOUNT,DETF,DETJT
COMMON /A4/ NQ,NB,NI,MFUN,NF,LEF,IF,IIADCD,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)
COMMON /A5/ NQ,NB,NI,MFUN,NF,LEF,IF,IIADCD,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)
COMMON /A6/ VLF(300,100),NDERS(20,50),NDERJS(20,20)
COMMON /A7/ RES(20,20)
REAL NDERS,NDERJS
DIMENSION V(NPARAM)
DIMENSION F(100,20),S(50,50),H(50,50),DJ(50,50),PM(50,50)
DIMENSION QM(50,50),G(50),DT(50)
DETJT=0.

GET F, FORM F·F AND INVERT F·F

DO 600 IT=1,NT
IF(K.GT.1) GO TO 618
DO 610 I=1,NB
IF(NINTF.EQ.0) GO TO 605
F(IT,I)=VLF(NINTF+I,IT)
GO TO 610
605 F(IT,I)=VLF(I,IT)
610 CONTINUE
600 CONTINUE
CALL FPRIME(S,F,NB,NT)
DO 801 I=1,NB
DO 801 J=1,NB
RES(I,J)=S(I,J)
801 CONTINUE
CALL INVERT(S,NB,DETF)

GET JT AND INVERT

618 DO 615 IT=1,NT
DO 620 I=1,NQ
DO 630 J=1,NQ
IVAR=IVECT(NPARAM+J)
NFUNC=NDER(I,IVAR)
IF(NINTF.NE.0) NFUNC=NDER(NINTF+I,IVAR)
IF(NFUNC.LE.1000) GO TO 625
NFUNC=NFUNC-1000
H(I,J)=VLF(NFUNC,IT)
GO TO 630
625 H(I,J)=NDERS(I,IVAR)
IF(NINTF.NE.0) H(I,J)=NDERS(NINTF+I,IVAR)
630 CONTINUE
620 CONTINUE
CALL INVERT(H,NB,DETJ)
DET = ALOG(ABS(DETJ))
DETJ = DETJ + DET
IF(IMETH .GT. 1) GO TO 615

GET PARTIAL D(J(IJ))/D(THETA(K)) MATRIX

DO 640 I = 1, NB
DO 650 J = 1, NQ
IF(NQ .LE. 10 OR KPARAM .LE. 50) GO TO 643
NFUNC = NDERJ(I, J)
IF(NINTF .NE. 0) NFUNC = NDERJ(NINTF+I, J) GO TO 644

643 NFUNC = NDJ(I, K, J)
IF(NINTF .NE. 0) NFUNC = NDJ(NINTF+I, K, J)
644 IF(NFUNC .LE. 1000) GO TO 645
NFUNC = NFUNC - 1000
D(I, J) = VLF(NFUNC, IT)
GO TO 650

645 D(I, J) = NDERJS(I, J)
IF(NINTF .NE. 0) D(I, J) = NDERJS(NINTF+I, J)
650 CONTINUE
640 CONTINUE

COMPUTE P
DO 660 I = 1, NB
DO 670 J = 1, NB
PSUM = 0.
DO 680 M = 1, NB
PSUM = PSUM + H(I, M) * DJ(I, J)
PM(I, J) = PSUM
660 CONTINUE
670 CONTINUE
P = 0.
DO 690 I = 1, NB
DO 690 J = 1, NB
IF(I .NE. J) GO TO 690
P = P + PM(I, J)
690 CONTINUE

GET PARTIAL D(FI)/D(THETA K) VECTOR

DO 700 I = 1, NB
PSUM = 0.
DO 710 J = 1, NB
PSUM = PSUM + S(I, J) * F(IT, J)
G(I) = PSUM
700 CONTINUE

COMPUTE G(IT) = F"F(INVERSE)*F"(T)

DO 700 I = 1, NB
PSUM = 0.
DO 710 J = 1, NB
PSUM = PSUM + S(I, J) * F(IT, J)
G(I) = PSUM
700 CONTINUE

GET PARTIAL D(FI)/D(THETA K) VECTOR

DO 720 I = 1, NB
NFUNC = NDER(I, K)
IF(NINTF .NE. 0) NFUNC = NDER(NINTF+I, K)
IF(NFUNC .LE. 1000) GO TO 715
NFUNC = NFUNC - 1000
DT(I) = VLF(NFUNC, IT)
GO TO 720
715 DT(I) = NDERS(I,K)
    IF(NINTF .NE. 0) DT(I) = NDERS(NINTF+I,K)
720 CONTINUE

C
C
C

    COMPUTE Q
    Q = 0.
    DO 730 I = 1, NB
       Q = Q + DT(I) * G(I)
    730 CONTINUE

C
C
C

    STORE (P-Q)
    VMU(IT,K) = Q - P
615 CONTINUE
RETURN
END

SUBROUTINE FPRIME(S,F,NB,NT)
DIMENSION S(50,50),F(100,20)
DO 220 I = 1, NB
    DO 225 J = 1, NB
    SUM = 0.
    DO 230 M = 1, NT
       SUM = SUM + F(M,I) * F(M,J)
    230 S(I,J) = SUM / FLOAT(NT)
    225 CONTINUE
220 CONTINUE
RETURN
END
SUBROUTINE PRIOR(NLIST, I, L, NPRIOR, NPRS, IREC, IFUNC, NDIFF, ICT)
LOGICAL IREC
DIMENSION NLIST(4000), NPRIOR(300), NPRS(300)

; THIS SUBROUTINE IS TO CHECK PRIORITY ORDERING OF FUNCTIONS 
; FUNCTION IN INNER BRACKET IS ALLOCATED WITH HIGHER PRIORITY 
; THAN FUNCTION IN OUTER BRACKET

IC = NLIST(L)
ICT = NPRS(IC)
IF (ICT .EQ. I) GO TO 210
IC = ICT
NPOINT = ICT
IREC = .TRUE.

200 ICT = ICT - 1
IF (ICT .EQ. 0) GO TO 220
IPR = NPRIOR(ICT)
IF (IPR .LT. I) GO TO 220
IR = NPRS(IPR)
IF (IR - IC) 205, 201, 201

201 WRITE(6, 202) IC
202 FORMAT(1HO, 'FUNCTION', I3, ' HAS THE WRONG PRIORITY')
RETURN

205 IC = IR
GO TO 200

210 IREC = .FALSE.
IFUNC = I
RETURN

215 IREC = .FALSE.
NDIFF = 1
IFUNC = NPRIOR(ICT)
RETURN

220 ICT = ICT + 1
IF (ICT .EQ. NPOINT) GO TO 215
NDIFF = NPOINT - ICT
IFUNC = NPRIOR(ICT)
RETURN
END
SUBROUTINE INVERT(A,N,D)
DIMENSION A(50,50),L(50),M(50)

THE INVERSE OF THE MATRIX IS CALCULATED USING GAUSS JORDAN
WITH COMPLETE PIVOTING. THE INVERSE REPLACES THE ORIGINAL
MATRIX. L AND M ARE WORK VECTORS OF LENGTH N. THE DETERMINANT
D IS CALCULATED

D=1.0
DO 190 K = 1 , N
L(K)=K
M(K)=K
BIG=A(K,K)
DO 20 I=K,N
DO 20 J = K ,  N
IF(ABS(BIG)-ABS(A(I, J))) 10,20,20
10 BIG=A(I,J)
L(K)=I
M(K)=J
20 CONTINUE

C CHECK FOR SINGULARITY
IF(BIG) 40,30,40
30 D=0.0
RETURN

C INTERCHANGE ROWS
40 I=L(K)
IF(I-K) 50,70,50
50 DO 60 J=1 ,N
TEMP=-A(K,J)
A(K,J)=A(I,J)
60 A(I,J)=TEMP

C INTERCHANGE COLUMNS
70 J=M(K)
IF(J-K) 80,100,80
80 DO 90 I=1 ,N
TEMP=-A(I,K)
A(I,K)=A(I,J)
90 A(I,J)=TEMP

C DIVIDE COLUMN BY MINUS PIVOT
100 DO 120 I=1 ,N
IF(I-K) 110,120,110
110 A(I,K)=A(I,K)/(-BIG)
120 CONTINUE

C REDUCE MATRIX
DO 160 I=1 ,N
IF(I-K) 130,160,130
130 TEMP=A(I,K)
DO 150 J=1 ,N
IF(J-K) 140,150,140
140 A(I,J)=TEMP*A(K,J)+A(I,J)
150 CONTINUE
160 CONTINUE

C DIVIDE ROW BY PIVOT
DO 180 J=1 ,N
IF(J-K) 170,180,170
170 A(K,J)=A(K,J)/BIG
180 CONTINUE
C     CALCULATE DETERMINANT
     D=D*BIG
C     TAKE RECIPROCAL
     190 A(K,K)=1.0/BIG
C     BACK SUBSTITUTION
     NM1=N-1
     IF(NM1) 200,270,200
     200 DO 260 KK=1,NM1
          K=N-KK
          J=L(K)
          IF(J-K) 210,230,210
     210 DO 220 I=1,N
          TEMP=A(I,K)
          A(I,K)=-A(I,J)
     220 A(I,J)=TEMP
     230 I=M(K)
     IF(I-K) 240,260,240
     240 DO 250 J=1,N
          TEMP=A(K,J)
          A(K,J)=-A(I,J)
     250 A(I,J)=TEMP
     260 CONTINUE
     270 RETURN
     END

SUBROUTINE RDCARD(ITEXT)
DIMENSION ITEXT(80)
READ(5,1000) (ITEXT(I),I=1,80)
1000 FORMAT(80A1)
WRITE(6,1001) (ITEXT(I),I=1,80)
1001 FORMAT(1H0,2X,40A1/1HO,2X,40A1)
RETURN
END
SUBROUTINE FRML(ITEXT,NFUNC,N,IADC,IEND,MAXCON,MAXF,KC,ISYM)

FORMULA PROCESSOR

COMMON /A1/NLIST(4000),NPRIOR(300),NPRS(300),NDR(300,60),NADR(300,1),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
DIMENSION IREPS(20),NANTS(20),NTERS(20),NANFS(20),NFACS(20)
DIMENSION LSYM(33),ITEXT(80)
DIMENSION ISYM(200)
LOGICAL IEND
DATA LSYM/1HV,1HF,1H*,1H/,1H+,1H-,1HO,1H1,1H2,1H3,1H4,1H5,1H6,
1H7,1H8,1H9,1H,1H),1H=,1HL,1HE,1HS,1HC,1HA,1HN,1HG,1HX,1HP,
1HI,1HO,1HR,1HT,1H(/)
DATA JBLANK/1H/
DATA JD/1HS/

INITIALIZATION

IEND=.FALSE.
JCOUNT=0
NDEPTH=0
NSYM=0
ISIGN=1
IREP=0
ISET=0
ISPEC=0
K=0
ICOUNT=0

INPUT FORMULA WITH SYMBOLS
COUNT NUMBER OF SYMBOLS AND PUT ADDRESS OF EACH
SYMBOL IN NLIST
NSYM IS NUMBER OF SYMBOLS IN EACH EXPRESSION
ICOUNT TESTS END OF EACH FUNCTION DEFINITION
JCOUNT IS TERMINATOR OF INPUT

2 DO 10 I=1,80
   JX=ITEXT(I)
10   DO 12 J=1,33
       IF(JX.NE.LSYM(J)) GO TO 12
       ICOUNT=0
       K=K+1
       GO TO 15
12      CONTINUE
       IF(JX.EQ.JBLANK) GO TO 13
       IF(JX.EQ.JD) GO TO 18
       WRITE(6,9)JX,I
       9 FORMAT(1HO,'ILLEGAL CHARACTER ',A1,'FOUNT AT SYMBOL',I3)
       RETURN
15 ISYM(K)=J
       NSYM=NSYM+1
       GO TO 14
13 ICOUNT=ICOUNT+1
14 IF(ICOUNT.GE.3) GO TO 16
       GO TO 10
18 JCOUNT=JCOUNT+1
IF(JCOUNT .NE. 4) GO TO 10
IEND=.TRUE.
RETURN
10 CONTINUE
IF(ICOUNT .GE. 3) GO TO 16
CALL RDCARD(ITEXT)
GO TO 2
16 ISYM(NSYM+1) = 18

INITIALIZE NUMBER ROUTINE TO GET INDEX OF FUNCTION VARIABLE, POWER OR CONSTANT
ISWIT = 1 AND 2 INTEGER, 3 AND 4 REAL NUMBER
IDEC = 0 AFTER "." AND INCREASE BY ONE TO COUNT NUMBER
OF DECIMAL PLACES
ID IS THE NEXT INTEGER IN THE SYMBOL LIST

c
ILIST=1
IC=ISYM(ILIST)
IF(IC .NE. 2) GO TO 100
ISWIT=1
IDEC=-1
IC=1
ILIST=2
NUM=0
CALL NUMBER(ISYM,IC,N,ILIST,IDEC,ISWIT,NUM,CONS,MAXCON,ID,IREP, ICONC,NFNUM)
GO TO (27,41,56,32),ISWIT
27 NLIST(IADC)=NUM
NFNUM=NUM
NADR(NFNUM)=IADC
MAXF=MAXF+1
NPROR(MAXF)=NFNUM
NPRS(NFNUM)=MAXF
NFCT=-1 SHOWS CONSTANT HAS NOT YET READ
NADNT=IADC+1
NADNF=NADNT+1
IADC=NADNF+2
NFCT=-1
NTERM=1
SYMBOL IS "," , SKIPE TO NEXT INTEGER IN ISYM
IF(ID .NE. 12) GO TO 100
28 ILIST(ILIST+1)
IC=ISYM(ILIST)
"\)" NOT ALLOWED
TEST FOR "-"
C
IF(IC .EQ. 18) GO TO 100
IF(IC .NE. 6) GO TO 31
29 ISIGN=-1
30 ILIST=ILIST+1
IC=ISYM(ILIST)
C
IF TEST IS TRUE EXPECTS "V" OR "P"
31 IF(IC .LE. 6) GO TO 40
ID=IC-7
C
IF TEST IS TRUE EXPECTS SPECIAL FUNCTIONS
IF(ID .GE. 11) GO TO 45
START PROCESSING CONSTANT

ISWIT = 4
IC = ILIST
NUM = 0
IDEC = -1
IF (ID .LT. 10) GO TO 21
IDEC = 0
22 CALL NUMBER (ISYM, IC, N, ILIST, IDEC, ISWIT, NUM, CONS, MAXCON, ID, IREP, ICONC, NFACT)
GO TO 32
21 NUM = 10 * NUM + ID
GO TO 22
32 IF (NFACT .EQ. -1) GO TO 34
IC = NLIST (NADNF + 1)
IF (IC .GT. 0) GO TO 33
ISIGN = -ISIGN
IC = -IC
33 IC = IC - 1
IF (IC .GT. 0) GO TO 35
34 IDT = MAXCON + 1
IF (ISIGN .LT. 0) IDT = -IDT
NLIST (NADNF + 1) = IDT
IF (NFACT .EQ. -1) NFACT = 0
ISPEC = 0
GO TO 37
35 CONS (IC) = CONS * CONS (IC)
MAXCON = MAXCON - 1
IC = IC + 1
IF (ISIGN .LT. 0) IC = -IC
NLIST (NADNF + 1) = IC
37 ISIGN = 1
IF (ID .EQ. 11) GO TO 75
IF (ID .EQ. 12) GO TO 100

IF TEST IS TRUE, EXPECTS "+" OR "-
ELSE IF ID = -3 THEN EXPECTS "/"
IF (ID .GT. -3) GO TO 39
IF (ID .NE. -3) GO TO 38
IREP = -15
GO TO 28
38 IF (ID .LT. -4) GO TO 100

TEST FOR "**"
IF (ISYM (ILIST + 1) .EQ. 3) GO TO 55
IREP = 0
GO TO 28

COMPLETES DESCRIPTION OF PREVIOUS TERM
SET UP PARAMETERS FOR NEXT TERM

39 NLIST (NADNF) = NFACT
IREP = 0
NADNF = IADC
NTERM = NTERM + 1
IADC = NADNF + 2
NFACT = -1
ID=-1 EXPECTS "-"
ID=-2 EXPECTS "+

IF(ID .EQ. -1) GO TO 29
GO TO 30

SECTION FOR "V" AND "F"

40 IF(IC .GT. 2) GO TO 100
IF(NFACT .GE. 0) GO TO 42
NLST(IADC-1)=ISIGN
ISIGN=1
NFACT=0

42 NFACT=NFACT+1
IF(ISIGN .LT. 0) GO TO 100
ISPEC=0

SET UP NUMBER ROUTINE FOR V OR F
STORE V OR F IN APPROPRIATE ADDRESS OF NLST

IDEC=-1
IFC=IC
ILST=ILIST
ILST+1=ILST+1
NUM=0
ISWIT=2
CALL NUMBER(ISYM,IC,W,ILST,IDEC,ISWIT,NUM,CONS,MAXCON,ID,IREP,
ICONC,NFNUM)

41 NUM=NUM+5
IF(IFC .EQ. 2) NUM=NUM+995
NLST(IADC)=NUM
IADC+1=IADC+1
IF(IREP .EQ. 0) GO TO 37
NLST(IADC)=-15
IADC+1=IADC+1
GO TO 37

SECTION FOR SPECIAL FUNCTIONS
1 = LOG
2 = EXP
3 = SIN
4 = COS
5 = ARCTAN

45 IF(NFACT .GE. 0) GO TO 44
NLST(IADC-1)=ISIGN
ISIGN=1
NFACT=0

IF TEST IS TRUE, EXPECTS "(

44 IF(ID .EQ. 26) GO TO 70
IF(ID .GE. 19) GO TO 100
IF(ISPEC .EQ. 1) GO TO 100
ISPEC=1
IDD=ID-12
GO TO (46,47,48,49,50,51),IDD

TEST FOR "LOG"

46 ILST=ILST+1
IF(ISYM(ILST) .NE. 30) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 26) GO TO 100
NLIST(IADC)=1
IADC=IADC+1
GO TO 30
C TEST FOR "EXP"
47 ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 27) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 28) GO TO 100
NLIST(IADC)=2
IADC=IADC+1
GO TO 30
C TEST FOR "SIN"
48 ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 29) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 25) GO TO 100
NLIST(IADC)=3
IADC=IADC+1
GO TO 30
C TEST FOR "COS"
49 ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 30) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 22) GO TO 100
NLIST(IADC)=4
IADC=IADC+1
GO TO 30
C TEST FOR "ARCT" FOR ARCTANGENT
50 ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 32) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 24) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 25) GO TO 100
NLIST(IADC)=5
IADC=IADC+1
GO TO 30
C
C SECTION FOR KEY WORD "NEXT"
C
51 ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 21) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 27) GO TO 100
ILIST=ILIST+1
IF(ISYM(ILIST) .NE. 32) GO TO 100
CALL RDCARD(ITEXT)
GO TO 1
C
C SECTION FOR "**"
C
55 NUM=0
ISWIT=3
IC=ILIST+1
ILIST = IC + 1
ISET = IREP
ID = ISYM(ILIST) - 7
IF (ID .NE. -1) GO TO 54
IREP = IREP - 15
IC = ILIST
ILIST = ILIST + 1
54 CALL NUMBER(ISYM, IC, N, ILIST, IDEC, ISWIT, NUM, CONS, MAXCON, ID, IREP, IONC, NFUNC)

56 IF (IDEC .GT. 0) GO TO 57
IF PREVIOUS OPERATOR WAS "/" THEN INSERT "**-1"

IF (IREP .EQ. 0) GO TO 58
IF (IREP .EQ. -15) NUM = -NUM
IREP = 0
IF (ISET .EQ. -15) IADC = IADC - 1
58 IF (NUM .GT. 10) GO TO 57
IF (NUM .LT. -15) GO TO 57
NUM = -NUM - 16
GO TO 59

57 IF (IREP .EQ. 0) GO TO 60
IF (ISET .EQ. -15) IADC = IADC - 1
IF (IREP .LT. -15) CONC = -CONC
IREP = 0
CONS(MAXCON) = -CCNC
60 NUM = -26 - MAXCON
59 NLIST(IADC) = NUM
IADC = IADC + 1
GO TO 37

SECTION FOR "(", EXPECTS NEW FUNCTION AND UNCONDITIONAL JUMP
INCREASE DEPTH OF NESTED FUNCTION
STORE PARAMETERS FOR OUTER FUNCTION

70 IADC = IADC + 1
NLIST(IADC) = -16
KC = KC + 1
JUMPAD(N, KC) = IADC
IADC = IADC + 2
NDEPT H = NDEPT H + 1
IREPS(NDEPT H) = IREP
NANTS(NDEPT H) = NADNT
NFACT = NFACT + 1
NFACS(NDEPT H) = NFACT
NTERS(NDEPT H) = NTERM
NANFS(NDEPT H) = NANDF
MAXF = MAXF + 1
NFUNC = NFUNC + 1
NLIST(IADC - 3) = NFUNC + 1000
NLIST(IADC) = NFUNC
NADR(NFUNC) = IADC

SET UP PRIORITY ORDERING AND REVERSE ORDERING OF FUNCTIONS
IC = NLIST(NADNT - 1)
ICT = NPRS(IC)
NMov = MAXF - ICT
ICT = MAXF
DO 71 IC = 1, NMov
NPT=NPRIOR(ICT-1)
NPRIOR(ICT)=NPT
ICT=ICT-1
NPRS(NPT)=NPRS(NPT)+1
71 CONTINUE
NPRIOR(ICT)=NFUNC
NPRS(NFUNC)=ICT

C SET UP PARAMETERS FOR INNER FUNCTION
NADNT*IADC+1
NADNF=NADNT+1
IADC=NADNF+2
IREP=0
NFACT=-1
NTERM=1
GO TO 28

C SECTION FOR ")"
75 IF(ILIST .GT. NSYM) GO TO 91
IF(NDEPTH .GT. 0) GO TO 76
74 WRITE(6,150) NFNUM,ILIST
150 FORMAT(1HO,' DEFINITION OF FUNCTION',I3,' HAS A SURPLUS RIGHT BRACKET AT SYMBOL',I3)
ILIST=ILIST+1
ID=ISYM(ILIST)-7
GO TO 37

C PUT ADDRESS FOR JUMP FROM OPENING BRACKETER
C CLEAR UP END OF INNER FUNCTION
C RESET VALUES FOR OUTER FUNCTION
76 IC=NADNT-2
NLIST(IC)=IADC
NLIST(NADNT)=NTERM
NLIST(NADNF)=NFACT
NTERM=NTERS(NDEPTH)
NADNT=NANTS(NDEPTH)
NFACT=NFACS(NDEPTH)
NADNF=NANFS(NDEPTH)
IREP=IREPS(NDEPTH)
NDEPTH=NDEPTH-1

C TEST WHETHER BRACKETS ARE UNNECESSARY
ID=IADC-IC
IF(ID .GT. 6) GO TO 78
IF(ID .LT. 6) RETURN
ID=NLIST(IC-2)
IF(ID .NE. 1) GO TO 78
NLIST(IC-2)=NLIST(IADC-1)
IADC=IC-1
ILIST=ILIST+1
ID=ISYM(ILIST)-7
GO TO 37

C JUMP IS UNNECESSARY BECAUSE OUTER AND INNER FUNCTIONS END TOGETHER
78 ILIST=ILIST+1
ID=ISYM(ILIST)-7
IF(ID .EQ. 11) GO TO 72
IF(I REP .EQ. 0) GO TO 37
N LIST(IADC) = -15
IADC = IADC + 1
IREP = 0
GO TO 37
72 IF(ILIST .GT. NSYM) GO TO 73
IF(N DEPTH .EQ. 0) GO TO 74
73 N LIST(IC-1) = I REP
N LIST(IC) = 0
IREP = 0
DO 82 JZ = 1, KC
IF(J umpAD(N, JZ) .EQ. IC-1) J umpAD(N, JZ) = 0
82 CONTINUE
GO TO 75
C END OF FUNCTION SPECIFICATION
91 N LIST(NADNT) = NTERM
N LIST(NADNF) = N FACT
IF(N DEPTH .EQ. 0) RETURN
WRITE(6,94) NFNUM
94 FORMAT(1HO,'DEFINITION OF FUNCTION',I3,'HAS TOO FEW LEFT HAND BRACKETS')
IC = NADNT - 3
N LIST(IC) = IREPS(N DEPTH)
N LIST(IC+1) = 0
NTERM = NTERS(N DEPTH)
NFACT = NFACS(N DEPTH)
NADNT = NANTS(N DEPTH)
NADNF = NANFS(N DEPTH)
N DEPTH = N DEPTH - 1
GO TO 91
100 WRITE(6,101) NFNUM, ILIST
101 FORMAT(1HO,'ERROR IN THE DEFINITION OF FUNCTION',I3,'AT SYMBOL',
I3,'SHOULD NOT OCCUR')
RETURN
END
SUBROUTINE NUMBER(ISYM, IC, N, ILIST, IDEC, ISWIT, NUM, CONS, MAXCON, ID, IIREP, CONC, NFNUM)
DIMENSION CONS(200), ISYM(200)
19 IC = IC + 1
   ID = ISYM(IC) - 7
   ID NEGATIVE INDICATES END OF NUMBERS
   ID GREATER THAN 10 INDICATES """" OR """"-"""" FOLLOWS NUMBER
   IF (ID .LT. 0) GO TO 23
   IF (ID .GT. 12) GO TO 100
   IF (ID .GT. 10) GO TO 23
   IF (IDEC .GE. 0) GO TO 20
   IF (ID .LT. 10) GO TO 21
   IDEC = 0
   GO TO 19
20 IDEC = IDEC + 1
21 NUM = 10 * NUM + ID
   GO TO 19
23 IF (ILIST .EQ. IC) GO TO 100
   ILIST = IC
   IF (ISWIT .GT. 2) GO TO 24
   IF (IDEC .GT. 0) GO TO 102
   RETURN
24 CONC = FLOAT(NUM)
   IF (ISWIT .EQ. 3 .AND. IDEC .LE. 0) RETURN
   IF (IDEC .LE. 0) GO TO 26
   SCALE = 1.
   DO 25 I = 1, IDEC
   SCALE = 10. * SCALE
   25 CONTINUE
   CONC = CONC / SCALE
26 IC = ISWIT + IREP
   IF (IC .NE. -11) GO TO 17
   IREP = 0
   CONC = 1.0 / CONC
   MAXCON = MAXCON + 1
   CONS(MAXCON) = CONC
   RETURN
100 WRITE(6, 101) NFNUM, ILIST
   101 FORMAT(1HO,'ERROR IN THE DEFINITION OF FUNCTION', I3, 'AT SYMBOL', I3, 'SHOULD NOT OCCUR')
   RETURN
102 WRITE(6, 103) NFNUM, ILIST
   103 FORMAT(1HO,'DECIMAL POINT IN THE DEFINITION FUNCTION' I3, 'AT SYMBOL', I3, 'SHOULD NOT OCCUR')
   RETURN
END
SUBROUTINE DIFF(IVAR, I)

DIFFERENTIATION SUBROUTINE

COMMON /A1/NLIST(4000), N_PRIOR(300), NPRS(300), NDER(300, 60), NADR(300, I), LISTEN(20), JUMPAD(20, 5), JC(20), CONS(200)
COMMON /A3/LTMAX, LMAXF, JTMAXF, LTFUNC(50), JMAXF, JFUNC(50),
ICOUNT, DETF, DETJT
COMMON /A4/NQ, NB, NI, MFUN, NT, K, KVAR, KFUNC, KPARAM, IMETH, ISTEP, NINTF
COMMON /A5/LTHETA, JACOB, IREC, MAXAD, MAXF, MAXCON, IVEC(T(70), IADC,
IIADCD, IFUNC, NFUNC, ITAN, NTAN, NFLAG, LDIFF, ICON(4)
COMMON /A6/VLF(300, 100), NDER(20, 50), NDERJ(300, 60), NDERJS(20, 20)
COMMON /A7/NDJ(10, 50, 10)
REAL NDERS, NDERJ
LOGICAL JACOB
LOGICAL IREC
LOGICAL SKIP
LOGICAL LJUMP
LOGICAL LTHETA
NTERMD=0
NLIST(IADCD)=NFUNC
L=IADCD
IADCD=IADCD+2
NTERM=NLIST(IADC)
IADC=IADC+1

DIFFERENTIATE TERM BY TERM
ICLIST IS CONSTANT IN CURRENT TERM
ITRAN GIVES START OF DESCRIPTION OF FACTER IN CURRENT TERM

DO 100 I1=1, NTERM
SKIP=.TRUE.
NFACT=NLIST(IADC)
IADC=IADC+1
IF(NFACT .EQ. 0) GO TO 96
ICLIST=NLIST(IADC)
ITRAN=IADC+1
GO TO 95
96 IADC=IADC+1
GO TO 100

DIFFERENTIATE USING PRODUCT FORMULAE
WITH ONE TERM FOR EACH FACTOR

95 DO 97 J=1, NFACT
LJUMP=.FALSE.
IADC=IADC+1
ICOP=IADC
ID=NLIST(IADC)
IF(ID .GT. 5) GO TO 1
IADC=IADC+1
ID1=NLIST(IADC)
IDNT=1
GO TO 2
1 ID1=ID
IDNT=0
ID2 IS USED TO TEST FOR EXPONENT

2 ID2=NLIST(IADC+1)
IF(ID2 .GE. 0) GO TO 3
IADC=IADC+1.
IF(ID2 .EQ. -16) GO TO 3
IF(ID1 .LT. 1000) GO TO 4
IDMT=IDMT-1
IC=ID1-IVAR
IF(IC .NE. 5) GO TO 97
GO TO 40
4 IF(NDER(ID1-1000,IVAR) .EQ. 0) GO TO 97
GO TO 40
40 IFACC=IADCD
IADCD=IFACC+1
NLIST(IADCD)=ICLIST
IADCD=IADCD+1
IFACT=0
IF(J .EQ. 1) GO TO 9
IC=ITRAN
ISWIT=0
IFLAG=0
IF(ISWIT .EQ. 0) GO TO 73
IF(ISWIT .NE. 0 .AND. NLIST(IC) .LT. 0) ICOP=ICOP+1
IF(ISWIT .NE. 0 .AND. IC .GT. 0 .AND. IC .LE. 5) ICOP=ICOP+1
IF(ISWIT .NE. 0 .AND. NLIST(IC) .EQ. -16) ICOP=NLIST(IC)+1
73 IF(IC .NE. -16) GO TO 5
IC=NLIST(IC)
GO TO 6
8 LJUMP=.TRUE.
IADC=NLIST(IADC+1)-1
GO TO 2
5 NLIST(IADCD)=IC
IADCD=IADCD+1

TRANSFER INITIAL AND FINAL NON-DIFFERENTIATED FACTORS TO THE DERIVATIVE
OMITS JUMPS FROM THE DERIVATIVE
6 IF(IFLAG .EQ. 1) GO TO 74
IF(ICC .NE. ICOP) GO TO 7
74 IF(ISWIT .EQ. 1) GO TO 43
GO TO 44
43 IFLAG=1
44 IF(IFACT .LT. NFACT) GO TO 7
9 IF(ID .GT. 5) GO TO 16
GO TO (10,11,12,13,14),ID
C
DERIVATIVE HAS THE FORM X**-1 OR F**-1
10 NLIST(IADCD)=I1D
IADCD=IADCD+1
NLIST(IADCD)=-15
IADCD=IADCD+1
GO TO 16
C
DERIVATIVE HAS THE FORM EXP(X) OR EXP(F)
11 NLIST(IADCD)=2
GO TO 15
C
DERIVATIVE HAS THE FORM COS(X) OR COS(F)
12 NLIST(IADCD)=4
GO TO 15
C
DERIVATIVE HAS THE FORM -SIN(X) OR -SIN(F)
13 NLIST(IADCD)=3
ICLIST=-ICLIST
IF(ICLIST .EQ. 0) ICLIST=-1
NLIST(IFACC+1)=ICLIST
GO TO 15
C
FOR ARCTAN DEFINE A NEW FUNCTION AS 1+X**2 OR 1+F**2
C
INVERT THE FUNCTION
14: NFUNC=NFUNC+1
IF(LTHETA .OR. JAC0B) MAXF=MAXF+1
NLIST(IADCD)=NFUNC+1000
NADR(NFUNC)=IADCD
IADCD=IADCD+1
NLIST(IADCD)=-16
IADCD=IADCD+1
NLIST(IADCD)=IADCD+9
IADCD=IADCD+1
NLIST(IADCD)=NFUNC
NADR(NFUNC)=IADCD
IADCD=IADCD+1
NLIST(IADCD)=2
IADCD=IADCD+1
NLIST(IADCD)=0
IADCD=IADCD+1
NLIST(IADCD)=1
IADCD=IADCD+1
NLIST(IADCD)=1
IADCD=IADCD+1
NLIST(IADCD) = 1
IADCD = IADCD + 1
NLIST(IADCD) = ID1
IADCD = IADCD + 1
NLIST(IADCD) = -18
IADCD = IADCD + 1
NLIST(IADCD) = -15
IADCD = IADCD + 1
IF (.NOT. LTHETA) GO TO 48
NPRIOR(MAXF - 1) = NFUNC
NPRS(NFUNC - 1) = MAXF
NPRIOR(MAXF) = NFUNC - 1
NPRS(NFUNC) = MAXF - 1
IF (LTHETA .OR. JACOB) GO TO 16
48
ITAN = 1
NTAN = NTAN + 1
GO TO 16
15
IADCD = IADCD + 1
NLIST(IADCD) = ID1
IADCD = IADCD + 1
FUNCTION CORRESPONDS TO PARTIAL D(FJ)/D(X1)
16
IF (ID1 .LE. 1000) GO TO 17
IC = ID1 - 10000
ICC = NDER(IC, IVAR)
IF (ICC .GT. 1000) GO TO 41
IDNT = IDNT - 1
GO TO 35
41
NLIST(IADCD) = ICC
IADCD = IADCD + 1
C
INSERT EXTRA TERM FOR THE EXPONENT
17
IF (ID2 .GE. 0) GO TO 25
42
IF (ID .GT. 5) GO TO 18
NLIST(IADCD) = ID
IADCD = IADCD + 1
C
REPEATS SPECIFICATION OF FK
18
NLIST(IADCD) = ID1
IDNT = IDNT + 1
IADCD = IADCD + 1
IF (ID2 .GE. -26) GO TO 19
MAXCON = MAXCON + 1
IC = -ID2 - 26
C
(PJ - 1) IS HELD AS A NEW CONSTANT
CONS(MAXCON) = CONS(IC) - 1.0
CONS = CONS(IC)
NLIST(IADCD) = -26 - MAXCON
IADCD = IADCD + 1
GO TO 20
C
IF X**2 NO EXPONENT IN DERIVATIVE
19
NLIST(IADCD) = ID2 + 1
IADCD = IADCD + 1
IF (ID2 .EQ. -18) IADCD = IADCD - 1
IC = -ID2 - 16
CONST = FLOAT(IC)
20
MAXCON = MAXCON + 1
IF (ICLIST .GT. 1) GO TO 24
IF (ICLIST .LT. -1) GO TO 22
IF( ICLIST .EQ. -1) CONST=-CONST
GO TO 23

22 ICLIST=-ICLIST
CONST=CONST*(-CONS(ICLIST-1))
GO TO 23

24 CONST=CONST*CONS(ICLIST-1)

C NEW CONSTANT CI*PJ

23 CONS(MAXCON)=CONST
NLIST(IFACC+1)=MAXCON+1

C SIMPLIFY IF CONSTANT ONLY FOR DERIVATIVE FACTOR

C

25 IF( J .EQ. NFACT) GO TO 31
IF(ID1 .LE. 1000) GO TO 39
IF(ICC .GT. 1000) GO TO 39

35 IF(ICC .GT. 1) GO TO 33
IF(ICC .LT. -1) GO TO 32
IF(ICC .EQ. 1) CST=1.0
IF(ICC .EQ. -1) CONST=-1.0
GO TO 34

33 CONST=CONS(ICC-1)
GO TO 34

32 ICC=ICC
CONST=-CONS(ICC-1)

34 MAXCON=MAXCON+1
IF(ICLIST .GT. 1) GO TO 37
IF(ICLIST .LT. -1) GO TO 36
IF(ICLIST .EQ. -1) CONST=-CONST
GO TO 38

36 ICLIST=-ICLIST
CONST=CONST*(-CONS(ICLIST-1))
GO TO 38

37 CONST=CONST*CONS(ICLIST-1)

38 CONS(MAXCON)=CONST
NLIST(IFACC+1)=MAXCON+1
ICLIST=MAXCON+1
IF(ID2 .LT. 0) GO TO 42

39 IF( J .EQ. NFACT) GO TO 31
ISWIT=1
ICC=IADC+1
IMAX=ICC+1
ICOP=IMAX
IC=1
LJUMP=.FALSE.
IF(J .GT. 1) GO TO 47
ID=NLIST(ICC)
GO TO 27

47 IFACT=J
IFLAG=0
GO TO 7

27 IF(ID .GT. 5) GO TO 26
ICC=ICC+1
NLIST(IADC)=ID
IADC=IADC+1
ID=NLIST(ICC)

26 IC=IC+1
NLST(IADCD)=ID
ICC=ICC+1
IADCD=IADCD+1

29. ID=NLST(ICC)
   IF(ID .GE. 0) GO TO 30
   ICC=ICC+1
   IF(ID .NE. -16) GO TO 28
   LJUMP=.TRUE.
   ICC=NLST(ICC)
   GO TO 29

28 IPWR=ICC-1
   IF(JC(I) .EQ. 0) GO TO 87
   KK=JC(I)
   IF((KK .EQ. 1) .AND. (JUMPAD(I,KK) .EQ. 0)) GO TO 87
   DO 94 JK = 1,KK
       KC=JUMPAD(I,JK)
       IF(KC .EQ. 0) GO TO 94
       KC=NLST(KC+1)
   KC=NLIST(KC)
   GO TO 89
   CONTINUE
   GO TO 87
87 IADC=IADC+1
100 CONTINUE

C CHECK IF DERIVATIVE IS CONSTANT OR ZERO .
C
IF(NTERMD .EQ. 0) GO TO 102
IF((NTERMD .EQ. 1) .AND. (NFACTD .EQ. 0)) GO TO 103
GO TO 101
102 NFNC=NFNC-1
   MAXF=MAXF-1
106 NDER(IFUNC,IVAR)=0
108 IADCD=IADCD-2
   IF(ICOUNT .GT. 1) RETURN
   WRITE(6,401) IFUNC,IVAR
401 FORMAT(1HO,'DF',I2,'/DX',I1,' = O')
GO TO 105
107 IF(.NOT. IREC) IFUNC=I
   IF(NQ .GT. 10 .OR. KPARAM .GT. 50) GO TO 125
   NDJ(I,IVAR,KVAR)=0
   IF(.NOT. IREC .AND. NFLAG .EQ. 1 .AND. I .LE. NINTF)
   +NDER(KFUNC,IVAR)=0
   IF(I .LE. NINTF) NDER(KFUNC,IVAR)=0
IVAR=KVAR
GO TO 126
125 IVAR=KVAR
NDERJ(IFUNC,IVAR)=0
126 IF(IREC) NDER(KFUNC,K)=0
GO TO 108
103 NFUNC=NFUNC-1
MAXF=MAXF-1
IF(JACOB) GO TO 120
121 ICLIST=NLIST(IADCD-1)
IF(JACOB) GO TO 122
NDER(IFUNC,IVAR)=ICLIST
GO TO 123
122 IF(NQ .GT. 10 .OR. KPARAM .GT. 50) NDERJ(IFUNC,IVAR)=ICLIST
IF(I .LE. NINTF) NDER(KFUNC,K)=ICLIST
IF(IREC) NDER(KFUNC,K)=ICLIST
123 IADCD=IADCD-4
IF(ICLIST .EQ. 1) GO TO 110
IF(ICLIST .EQ. -1) GO TO 109
IF(ICLIST .GT. 1) GO TO 111
IF(ICLIST .LT. -1) GO TO 112
109 CONST=-1.0
GO TO 115
110 CONST=1.0
GO TO 115
111 CONST=CONS(ICLIST-1)
GO TO 115
112 ICLIST=-ICLIST
CONST=CONS(ICLIST-1)
115 IF(ICOUNT .GT. 1) RETURN
WRITE(6,402) IFUNC,IVAR,CONST
402 FORMAT(1H0,'DF',12,'/DX',11,' = ',F8.4)
GO TO 105
120 IF(.NOT. IREC) IFUNC=I
IF(NQ .GT. 10 .OR. KPARAM .GT. 50) GO TO 127
NDERJ(I,IVAR,KVAR)=NLIST(IADCD-1)
IF(.NOT. IREC .AND. NFLAG .EQ. 1 .AND. I .LE. NINTF) +NDER(KFUNC,IVAR)=NLIST(IADCD-1)
127 IVAR=KVAR
GO TO 121
101 NLIST(MAXAD+2)=NTERM
MAXAD=IADCD-1
IF(ICOUNT .GT. 1) RETURN
WRITE(6,400) IFUNC,IVAR,(NLIST(J),J=L,MAXAD)
400 FORMAT(1HO,'DF',I2,'/DX',I1,2X,20I4)
WRITE(6,550) MAXF,NFUNC,NFUNC,MAXF
550 FORMAT(1HO,'NPRIOR(',I2,') = ',I2,5X,'NPRS(',I2,') = ',I2)
105 RETURN
END
SUBROUTINE EVAL(IADC, I, IFUNC, VFUNC, IT, MAXAD, NPARAM, V, VLIST)
COMMON X(100, 50)
COMMON /A1/NLIST(4000), NPRIOR(300), NPRS(300), NDER(300, 60), NADR(300, 50, I), LISTEN(20), JUMPAD(20, 5), JC(20), CONS(200)
COMMON /A6/ VLF(300, 100), NDERV(20, 50), NDERJ(300, 60), NDERJS(20, 20)
REAL NDBERS, NDERJS
DIMENSION V(NPARAM)
LOGICAL LJUMP
LJUMP=.FALSE.
NTERM=NLIST(IADC)
IF(NTERM .EQ.0) RETURN
IADC=IADC+1
DO 100 I1=1, NTERM
VTERM=1.
> NFACT=NLIST(IADC)
IADC=IADC+1
ICLIST=NLIST(IADC)
IF(NFACT .EQ. 0) GO TO 96 ;
GO TO 95
96 IADC=IADC+1
IF(ICLIST .GE. 0) GO TO 80
IF(ICLIST .EQ. -1) GO TO 82
ICLIST=-ICLIST
VFUNC*VFUNC-CONS(ICLIST-1)
GO TO 100
82 VFUNC=-1.+VFUNC
GO TO 100
80 IF(ICLIST .GT. 1) GO TO 81
VFUNC=1.+VFUNC
GO TO 100
81 VFUNC=VFUNC+CONS(ICLIST-1)
GO TO 100
95 DO 97 J=1, NFACT
IADC=IADC+1
ID=NLIST(IADC)
IF(ID .GT. 5) GO TO 1
IADC=IADC+1
ID1=NLIST(IADC+1)
GO TO 2
1 ID1=ID
2 IF(IADC .EQ. MAXAD) GO TO 84
IADC=NLIST(IADC+1)
IF(IADC .GE. 0) GO TO 3
IADC=IADC+1
IF(IADC .EQ. -16) GO TO 8
IFWR=IADC
IF(JC(I) .EQ. 0) GO TO 3
KK=JC(I)
IF((KK .EQ. 1) .AND. (JUMPAD(I, KK) .EQ. 0)) GO TO 3
DO 93 JK=1, KK
KC=JUMPAD(I, JK)
IF(KC .EQ. 0) GO TO 93
93 CONTINUE
GO TO 3
91 IF(LJUMP) GO TO 3
84 ID2=0
3 IF(ID1 .GT. 1000) GO TO 4
   IC=ID1-5
   IF(IC .LE. NPARAM) GO TO 86
   VFACT=X(IT,IC-NPARAM)
   GO TO 94
86 VFACT=V(IC)
   GO TO 94
4 NFUNC=ID1-1000
   VFACT=VFLIST(NFUNC)
   GO TO 94
8 LJUMP=.TRUE.
   IADC=NLIST(IADC+1)-1
   GO TO 2
9 IF(ID .GT. 5) GO TO 99
   GO TO(10,11,12,13,14),ID
10 VFACT=ALOG(VFACT)
   GO TO 99
11 VFACT=EXP(VFACT)
   GO TO 99
12 VFACT=SIN(VFACT)
   GO TO 99
13 VFACT=COS(VFACT)
   GO TO 99
14 VFACT=ATAN(VFACT)
   GO TO 99
94 IF(ID2 .GE. 0) GO TO 9
   IF(ID2 .EQ. -15) GO TO 98
   IF(ID2 .GE. -26) GO TO 19
   ICONST=-ID2-26
   CONST=CONS(ICONT)
   VFACT=VFACT**ICONT
   GO TO 9
19 IC=-ID2-16
   CONST=FLOAT(IC)
   IF(IC .GT. 0) GO TO 200
   IC=-IC
   VFACT=1./VFACT**IC
   GO TO 9
200 VFACT=VFACT**IC
   GO TO 9
C TEST DIVISION SYMBOL BELONGS TO CURRENT FUNCTION
98 VFACT=1./VFACT
   GO TO 9
99 VTERM=VTERM*VFACT
97 CONTINUE
   IF(ICLIST .GE. 0) GO TO 21
   IF(ICLIST .EQ. -1) GO TO 21
   ICLIST=-ICLIST
   CONST=CONS(ICLIST-1)
   GO TO 22
21 CONST=FLOAT(ICLIST)
   IF((ICLIST .EQ. 1) .OR. (ICLIST .EQ. -1)) GO TO 22
   CONST=CONS(ICLIST-1)
22 VFUNC=VFUNC*VTERM*CONST
   IADC=IADC+1
100 CONTINUE
RETURN
END
SUBROUTINE INPUT(N,NL,NVAR,NR,NTRAN)
    DIMENSION XL(6,60)
    DIMENSION NAME(100)
    COMMON X(100,50)
    NVAR=NL*NR
    NA=N+NL-1
    DO 45 1=1,NA
        READ(5,50) (X(I,J),J=1,NR)
    50 FORMAT(6F10.4)
    45 CONTINUE
    IF(NTRAN .EQ. 0) GO TO 10
    CALL DATALT(N,NR,NVAR,NAME)
    10 IF(NL-1 .LE. 0) GO TO 14
        DO 6 I=1,NL
            DO 6 J=1,NVAR
                XL(I,J)=X(I,J)
    6 XL(I,J)=X(I,J)
    MK=0
    DO 7 I=1,N
        DO 7 J=1,NR
            IJ=I+NL-1
            X(I,J)-X(IJ,J)
    7 X(I,J)=X(IJ,J)
    DO 13 K=2,NL
        MK=MK+NR
        NC=N-K+1
        DO 11 I=1,NC
            DO 11 J=1,NR
                IK=I+K-1
                IJ=J+MK
                K1=K-1
                DO 12 IN=1,K1
                    IT=NL-K+IN
                    X(IN,IJ)=XL(IT,J)
    12 X(IN,IJ)=XL(IT,J)
        X(IK,IJ)=X(I,J)
    13 CONTINUE
    14 WRITE(6,15)
    15 FORMAT(1H1,10X,'DATA'///)
    DO 52 I=1,N
        WRITE(6,27) (X(I,J),J=1,NVAR)
    27 FORMAT(1H , 10F10.4)
    52 CONTINUE
RETURN
END
SUBROUTINE DATALT(N,NR,NVAR,NAME)
DIMENSION NAME(100), COMMON X(100,50)
READ(5,888) KD,IDA
READ(5,10) (NAME(I),I=1,KD)
DO 15 J=1,IDA
READ(5,11) NOP,NA,NB,NC,VAL
IF(NOP .EQ. 10) GO TO 40
IVAL=IFIX(VAL)
ND=NC-1
CALL ALMLAG(ND,N,NB,IVAL,NA)
GO TO 15
40 IF(NB .NE. 0) WRITE(6,7) NOP,NA,NAME(NA),NB,NAME(NB),NC,NAME(NC)
IF(NP .EQ. 6 .OR. NOP .EQ. 7) X(I,NC)=VAL
IF(NP .EQ. 3 .AND. VAL .EQ. 0.) VAL=1.
IF(NP .EQ. 4 .AND. VAL .EQ. 0.) VAL=1.
DO 30 I=1,N
SP=0.
SQ=1.
GO TO (1,2,3,4,5,6,8,9),NOP
1 IF(NB .NE. 0) SP=X(I,NB)
X(I,NC)=X(I,NA)+SP+VAL
GO TO 30
2 X(I,NC)=X(I,NA)-X(I,NB)+VAL
GO TO 30
3 IF(NB .NE. 0) SQ=X(I,NB)
X(I,NC)=X(I,NA)*SQ*VAL
GO TO 30
4 IF(X(I,NB) .EQ. 0) WRITE(6,25) NOP,NB
IF(X(I,NB) .EQ. 0.) GO TO 15
X(I,NC)=X(I,NA)*VAL/X(I,NB)
GO TO 30
5 IF(X(I,NA) .GT. 0) WRITE(6,25) NOP,NA
IF(X(I,NA) .GT. 0.) X(I,NC)=ALOG(X(I,NA))
GO TO 30
6 X(I+1,NC)=X(I,NA)
WRITE(6,25) NOP,NA
IF(NOP .NE. 7) GO TO 30
X(I,NB)=X(I,NA)-X(I,NC)
GO TO 30
8 X(I,NC)=EXP(X(I,NA))
GO TO 30
9 IF(X(I,NA) .LE. 0) WRITE(6,25) NOP,NA
IF(X(I,NA) .GT. 0) X(I,NC)=SQRT(X(I,NA))
30 CONTINUE
15 CONTINUE
25 FORMAT(1HO,' ILLEGAL OPERATION FOR ',I4,2X,' ON VARIABLE ',I4)
7 FORMAT(1HO,' OPERATION',I4,' PERFORMED ON VARIABLES',I4,' ('',I4,' ) AND ',I4,' ('',A4,' ) TO CREATE VARIABLE',I4,' ('',A4, I' ) 
18 FORMAT(1HO,' OPERATION',I4,' PERFORMED ON VARIABLE',I4,' ('',A4, I' ) TO CREATE VARIABLE',I4,' ('',A4,' ) ')
NVAR=KD
RETURN
SUBROUTINE ALMLAG(ND,N,NB,IVAL,NA)
COMMON X(100,50)
ITNEW=N-NB
LOPP=IVAL+1
ML1=NB+1
DO 1 I=1,ITNEW
IMAXL=I+NB
DO 2 J=1,LOPP
NJ=ND+J
JO=J-1
S=0.
DO 3 K=1,ML1
KO=K-1
3 S=S+X(IMAXL-KO,NA)*(ML1-KO)**JO
2 X(IMAXL,NJ)=S
1 CONTINUE
N=N+LOPP
RETURN
END
SUBROUTINE FUNML(NPARAM,V,FV)
COMMON X(100,50)
COMMON /A1/ NLIST(4000),NPRIOR(300),NPRI(300),NDER(300,60),INADR(300),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNC(50),IICOUNT,DETF,DETFJT
COMMON /A4/ NQ,NB,NI,MFUN,NT,K,KVAR,KFUNC,KPARAM,IMETH,ISTEP,NINFT
COMMON /A5/ LTHETA,JACOB,IREC,MAXAD,MAXF,MAXCON,IVECT(70),IADC,IIADC,IFUNC,NFUNC,ITAN,NTAN,NFLAG,LDIFF,ICON(4)
COMMON /A6/ VLF(300,100),NDERS(20,50),NDERJ(300,60),NDERJS(20,20)
COMMON /A7/ NDJ(10,50,10)
REAL NDERS,NDERJS
DIMENSION V(NPARAM)
LOGICAL LTHETA,JACOB
IF(ICOUNT .GT. 1 .AND. (NQ .LE. 10 .OR. KPARAM .LE. 50)) GO TO 151
IF(IMETH .EQ. 0 .AND. (NQ .LE. 10 .OR. KPARAM .LE. 50)) GO TO 151
IF(IMETH .EQ. 0) GO TO 152
LTHETA=.FALSE. !
JACOB=.FALSE. !
CALL DIFIML(NPARAM,V)
152 DO 155 K=1,NPARAM
LTHETA=.TRUE. !
CALL DIFIML(NPARAM,V)
JACOB=.TRUE. !
CALL DIFIML(NPARAM,V)
JACOB=.FALSE. !
155 CONTINUE
151 DO 150 K=1,NPARAM
K=K1
CALL DIEVAL(NPARAM,V)
CALL PQEVAL(NPARAM,V)
150 CONTINUE
FV=-((DETJT-0.5*FLOAT(NT)*ALOG(DETF))
MFUN=MFUN+1
RETURN
END
SUBROUTINE GCHECK(NPARAM,V,FUN,FUNCT)
COMMON X(100,50)
COMMON /A1/ NLIST(4000),NPRIOR(300),NPRS(300),NDER(300,60),INADR(300),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNC(50),IICOUNT,DETF,DETF
COMMON /A4/ NQ,NB,NI,MFUN,NT,K,KVAR,KFUNC,KPARAM,IMETH,ISTEP,NINTF
COMMON /A5/ LTHETA,JACOB,IREC,MAXAD,MAXF,MAXCON,IVECT(70),IAADC,IIADC
COMMON /A7/ NDJ(10,50,10)
REAL NDER,NNDERJS
DIMENSION V(NPARAM)
DIMENSION STORE(50)
EXTERNAL FUNCT
DATA EPS/1.E-3/
DO 10 I=1,NPARAM
  V(I)=V(I)+EPS
  CALL FUNCT(NPARAM,V,F)
  FHI=F-FUN
  FPLUS=F
  V(I)=V(I)-EPS*2.
  CALL FUNCT(NPARAM,V,F)
  FLO=F-FUN
  FMUS=F
  STORE(I)=(FPLUS-FMUS)/(2.*EPS)
10 CONTINUE
WRITE(6,9002) (STORE(I),I=1,NPARAM)
9002 FORMAT(10H0,'APPROXIMATE GRADIENT ',10F10.4)
RETURN
END
SUBROUTINE GSTEP(FUN,FV,GRAD,SC,NPARAM,V,FUNCT,IFOK)
COMMON X(100,50)
COMMON /A1/ NLIST(4000),NPRIOR(300),NRPS(300),NDER(300,60),
INADR(300),LISTEN(20),JUMPAD(20,5),JC(20),CONS(200)
COMMON /A2/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A3/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNCT(50),
IICOUNT,DETF,DETFJ
COMMON /A4/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A5/ LTMAX,LMAXF,JTMAXF,LTFUNC(50),JMAXF,JFUNCT(50),
IICOUNT,DETF,DETFJ
COMMON /A6/ VMU(100,50),R(50,50),D(50),TEMP(50)
COMMON /A7/ VMU(100,50),R(50,50),D(50),TEMP(50)
REAL NDERS,NLERJS
EXTERNAL FUNCT
DIMENSION V(NPARAM)
LOGICAL LTHETA,JACOB,IFOK
IFOK=.TRUE.
IDC=0
IQ=1
PREC=.5E-11
E1=.01
E2=.7
SL=0.
DL=1.
FL=FUN
DF=PREC*(ABS(FL)+PREC)
IKT=0
FV=FUN
DO 1 1=1,NPARAM
1 V(I)=TEMP(I)-SC*D(I)
CALL FUNCT(NPARAM,V,FV)
IF(FV .GT. FUN) GO TO 94
EXTRAPOLATE TO BRACKET MINIMUM
93 IKT=IKT+1
FO=-SC*GRAD
DD=(FV-(FUN+FO))/FO+1.0
IF(DD .LE. E2) GO TO 3
IF(FV .GE. FL) GO TO 3
IF(IKT .GT. 5 .AND. ABS(1.0-DD) .GE. E1) GO TO 3
SL=SC
DL=DD
FL=FV
IF(IKT .GT. 5) GO TO 13
IF(DD .GE. .95) SC=10.0*SC
IF(DD .LT. .95) SC=.5*SC/(1.0-DD)
DO 2 I=1,NPARAM
2 V(I)=TEMP(I)-SC*D(I)
CALL FUNCT(NPARAM,V,FV)
GO TO 93
3 IF(DD .GE. E1) GO TO 14
IF(ABS(1.0-DL) .GE. E1) GO TO 13
GO TO 4

MINIMUM BRACKETED
c

94  FO= -SC*GRAD
    DD=(FV-(FUN+FO))/FO+1.0
    IF(DD .LT. E1) GO TO 4
    IF(ABS(1.0-DD) .GE. E1) GO TO 14
    SL=SC
    DL=DD
    FL=FV
    IF(IQ .EQ. 0) GO TO 7
    GO TO 5
4  SR=SC
    DR=DD
    FR=FV
    IF(ABS(SC*GRAD) .LE. DF) GO TO 7

    CHECK SIZE OF BRACKET
7  SC=.5*(SR+SL)
    DO 8 I=1,NPARAM
8  V(I)=TEMP(I)-SC*D(I)
    CALL FUNCT(NPARAM,V,FV)
    GO TO 94

    USE MIDPOINT OF INTERVAL IF QUADRATIC INTERPOLATION FAILS
6  V(I)=TEMP(I)-SC*D(I)
    CALL FUNCT(NPARAM,V,FV)
    IF(FV .LT. FUN) GO TO 9
    IF(ABS(SR*GRAD) .GT. DF) GO TO 9

    RECALCULATE PROJECTED GRADIENT
9  IQ=1
    GO TO 94

    EXPLORE REVERSE SEARCH DIRECTION
10 IF(Grad .EQ. 0) GO TO 13
    DO 11 I=1,NPARAM
11 V(I)=TEMP(I)+SC*D(I)
    CALL FUNCT(NPARAM,V,FT)
    IF(FT .GE. FUN) GO TO 13
    DO 12 I=1,NPARAM
12 D(I)=-D(I)
    GRAD=(FV-FT)/SC
DL=1.0
SL=0.0
FV=FT
FL=FUN
IKT=0
GO TO 93

EXIT WHEN TERMINATION CONDITION NOT MET AT SC

13 SC=SL
   FV=FL
14 DO 15 I=1,NPARAM
15 V(I)=TEMP(I)-SC*D(I)
   IF(FV .GE. FUN) IFOK=.FALSE.
   RETURN
END
To illustrate how to use NLMLE with the specifications defined in Chapter 5, consider the following model:

\[
\begin{align*}
    y_{1t} &= (\theta_1 \theta_3 - \frac{1}{2} \theta_2)y_{2t} + \frac{\theta_1}{\theta_2} y_{1,t-1} + 2\theta_3 z_{1t} \\
    &\quad + \theta_3 \theta_4 z_{2t} + u_{1t} \\
    y_{2t} &= \theta_4 y_{1t} + \theta_5 z_{3t} + u_{2t}
\end{align*}
\]

Assume we have 60 observations in the data and the starting values of the coefficients are also given.

Rewrite model as:

\[
\begin{align*}
    u_{1t} &= y_{1t} - (\theta_1 \theta_3 - \frac{1}{2} \theta_2)y_{2t} - \frac{\theta_1}{\theta_2} y_{1,t-1} - 2\theta_3 z_{1t} - \theta_3 \theta_4 z_{2t} \\
    u_{2t} &= y_{2t} - \theta_4 y_{1t} - \theta_5 z_{3t}
\end{align*}
\]

Now transform all the equations and variables into NLMLE specifications:

1. Equations: \(\{u_1,u_2\} + \{F_1,F_2\}\)
2. Parameters set: \(\{\theta_1,\theta_2,\theta_3,\theta_4,\theta_5\} + \{v_1,v_2,v_3,v_4,v_5\}\)
3. Endogenous variables: \(\{y_{1t},y_{2t}\} + \{v_6,v_7\}\)
4. Predetermined variables: \(\{z_{1t},z_{2t},z_{3t},y_{1,t-1}\} + \{v_8,v_9,v_{10},v_{11}\}\)

The transformed model now becomes:

\[
\begin{align*}
F_1 &= v_6 - (v_1 * v_3 - \frac{1}{2} * v_2) * v_7 - (v_1/v_2) * v_{11} \\
    &\quad - 2 * v_3 * v_3 * v_8 - v_3 * v_4 * v_9 \\
F_2 &= v_7 - v_4 * v_6 - v_5 * v_{10}
\end{align*}
\]
Notice that powers can be expressed as "**", e.g. $v_3^2$ as $v_3 ** 2$.

Input Instructions

1. **BHHH** GSTEP (2I4)
   
   0 0

2. **IMAX** TOLB (I4,FLO.4)
   
   50 0.0001

3. **NB** **NI** NINTF NY NZ N NT NL NVAR (20I4)
   
   2 0 0 2 4 5 60 1 6

4. Names of Variables (2OA4)
   
   $y_{lt}$ $y_{2t}$ $z_{lt}$ $z_{2t}$ $z_{3t}$ $Y_{lt}$.

5. Data series, input ordering as (4) by variables (8F10.4) i.e. the X matrix.

6. Starting values of parameters (8F10.4) i.e. $\{0_1, ..., 0_5\}$ i.e. the V vector.

7. Equations: $F1 =$
   
   $F2 =$ as above.

8. $\$\$\$\$.
APPENDIX D

Program Output - A Typical Run From Model (iii)

NONLINEAR ARCTANGENT MODEL:

METHOD = BHHH
STEP = GSTEP (MODIFIED LINE SEARCH)
MAXIMUM NUMBER OF ITERATIONS = 50
TOLERANCE LEVEL FOR CONVERGENCE = 0.001
5 EQUATIONS 50 OBSERVATIONS 6 PARAMETERS
COEFFICIENTS

0.3 1.0 1.0 5.0 5.0 5.0

INPUT FUNCTIONS:

F1 = V1 * ATAN (V2 * V7) + V3 * V7 + \(0.1 + V3 \times 2 \times V8\)
F2 = V1 * ATAN (V2 * V9) + \(0.1 \times V3 \times 2 \times V7 + V3 \times V8\)
F3 = V1 * ATAN (V2 * V9) - \(0.1 \times V3 \times 2 \times V8 + (V3 + 0.1 \times V3 \times 2)\)
F4 = V1 * ATAN (V2 * V10) + \(0.1 \times V3 \times 2 \times V9 + (V3 - 0.1 \times V3 \times 2) \times V10\)
F5 = V1 * ATAN (V2 * V11) + \(0.1 \times V3 \times 2 \times V8 + 0.1 \times V3 \times 2 \times V10 \times V3 \times V11\)
F6 = \((V4 \times V12 + V17 \times 2) + F1\)
F7 = \((V5 \times V13 + V17 \times 2) + F2\)
F8 = \((V4 \times V14 + V18 \times 2) + F3\)
F9 = \((V6 \times V15 + V18 \times 2) + F4\)
F10 = \((V4 \times V16 + V18 \times 2) + V5\)
PARAMETERS: V1 V2 V3 V4 V5 V6
ENDOGENOUS VARIABLES: V7 V8 V9 V10 V11
EXOGENOUS VARIABLES: V12 V13 V14 V15 V16 V17 V18
START TIME = 8.1430

ITERATION NUMBER 1

GRADIENT
-.379390E+03 .212810E+02 -.200745E+04 -.293029E+04 .657361E+03 -.749687E+03

GRADIENT NORM = 3708.817889

FUN = -940.897488 FUNNEW = -943.141208

STEPSIZE = .100000E+01 GRAD = .493929E+01

NCALL = 2

DIRECTION
-.47345E-02 .44575E-01 .52228E-03 -.92267E-03 .11082E-02 .25493E-03

PARAMETER ESTIMATES
.30473E+00 .95543E+00 .99948E+00 .50009E+01 .49989E+01 .49997E+01

ITERATION NUMBER 2

GRADIENT
-.760862E+02 .150458E+00 -.746201E+03 .881762E+03 .241066E+03 -.484311E+03

GRADIENT NORM = 1277.802584

FUN = -943.141288 FUNNEW = -943.445479

STEPSIZE = .441979E+00 GRAD = .136577E+01

NCALL = 4
DIRECTION
.98958E-02 -.35333E-01 -.21791E-02 .22932E-03 -.82251E-03 -.10201E-02

PARAMETER ESTIMATES
.30036E+00 .97104E+00 .10004E+01 .50008E+01 .49993E+01 .50002E+01

ITERATION NUMBER 3

GRADIENT
-.260989E+02 .776491E+00 -.157761E+03 -.225522E+02 -.102499E+03 -.354972E+02

GRADIENT NORM = 194.537814

FUN = -943.445479 FUNNEW = -943.459954

STEPsize = .100000E+01 GRAD = .275175E-01

NCALL = 5

DIRECTION
-.16085E-03 .96259E-03 .77717E-04 -.25802E-04 .33775E-03 .10365E-04

PARAMETER ESTIMATES
.30052E+00 .97008E+00 .10004E+01 .50008E+01 .49989E+01 .50002E+01

ITERATION NUMBER 4

GRADIENT
-.588064E+01 -.899695E+00 -.363169E+02 .268113E+01 .128140E+02 -.563097E+01

GRADIENT NORM = 39.463905

FUN = -943.459954 FUNNEW = -943.460528

STEPsize = .453732E+00 GRAD = .253337E+02
NCALL = 7

DIRECTION

.44748E-03 -.18952E-02 -.93841E-04 .52135E-05 -.14690E-04 -.40123E-04

PARAMETER ESTIMATES

.30032E+00 .97094E+00 .10004E+01 .50008E+01 .49989E+01 .50002E+01

ITERATION NUMBER 5

GRADIENT

-.146527E+01 -.348852E+00 -.654604E+01 -.175159E+02 .587007E+01 .110445E+02

GRADIENT NORM = 22.546955

FUN = -943.460428 FUNNEW = -943.460633

STEPSIZE = .100000E+01 GRAD = .346181E-03

NCALL = 8

DIRECTION

-.48335E-04 .80070E-04 .15275E-04 -.46903E-05 .32733E-04 .11678E-04

PARAMETER ESTIMATES

.30037E+00 .97086E+00 .10004E+01 .50008E+01 .49989E+01 .50002E+01

ITERATION NUMBER 6

GRADIENT

-.146241E+01 -.891401E-01 -.984296E+01 .129665E+02 .300475E+01 -.757161E+01

GRADIENT NORM = 18.262456
FUN = -943.460633   FUNNEW = -943.460688

STEPSIZE = .375083E+00   GRAD = .297599E-03

NCALL = 10

DIRECTION

.14706E-03 -.58491E-03 -.31749E-04 .43311E-05 -.12071E-04 -.16921E-04

PARAMETER ESTIMATES

.30031E+00 .97108E+00 .10004E+01 .50008E+01 .49989E+01 .50002E+01

ITERATION 7

GRADIENT

-.269233E+00 -.181003E+01 -.127856E+01 .465367E+00 .107269E+01 .395240E-01

GRADIENT NORM = 1.753947

FUN = -943.460688   FUNNEW = -943.460688

STEPSIZE = .375083E+00   GRAD = .297599E-03

NCALL = 10

DIRECTION

-.29042E-05 -.53806E-06 .14347E-05 -.62081E-08 .43648E-05 .42346E-06

PARAMETER ESTIMATES

.30031E+00 .97108E+00 .10004E+01 .50008E+01 .49989E+01 .50002E+01

CONVERGENCE ACHIEVED AFTER 7 ITERATIONS.

NUMBER OF FUNCTION EVALUATIONS = 10

LOG LIKELIHOOD FUNCTION = -943.460688
INVERSE HESSIAN MATRIX

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00099</td>
<td>0.000363</td>
<td>0.00021</td>
<td>0.00001</td>
<td>0.00009</td>
<td>0.00009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.001705</td>
<td>0.00072</td>
<td>0.00007</td>
<td>0.00025</td>
<td>0.00032</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.00000</td>
<td>-0.00000</td>
<td>-0.00000</td>
<td>-0.00000</td>
<td>-0.00000</td>
<td>-0.00000</td>
<td>-0.00000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PARAMETERS

<table>
<thead>
<tr>
<th>PARAMETERS</th>
<th>STD -ERRORS</th>
<th>T-RATIOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.300312</td>
<td>0.009934</td>
<td>30.230912</td>
</tr>
<tr>
<td>0.971079</td>
<td>0.041293</td>
<td>23.516612</td>
</tr>
<tr>
<td>1.000402</td>
<td>0.002251</td>
<td>444.500513</td>
</tr>
<tr>
<td>5.000848</td>
<td>0.000504</td>
<td>9921.666183</td>
</tr>
<tr>
<td>4.998896</td>
<td>0.002478</td>
<td>2017.202929</td>
</tr>
<tr>
<td>5.000198</td>
<td>0.001148</td>
<td>4346.444014</td>
</tr>
</tbody>
</table>

END TIME = 21.0350
A Parallel Inversion Routine

REAL JT,,N,N), UNIT_MATRIX (,,N,N), DET_JT,, PIVOT_ELEMENT ,, COLUMNS_PIVOT (,,N), SIGMA (,), TEMP (,), AA(,), ATEMP (,), BETEMP (,) LOGICAL SWAP (,), SIGN_CHANGE (,) EQUVALENCE (AA, TEMP), (PIVOT_ELEMENT, ATEMP, BETEMP)

C

C INITIALISE UNIT MATRIX

C DET_JT = 1.0
DO 10 I = 1; N
DO 10 J = 1, N
UNIT_MATRIX (,,I,J) = 0.0 IF(I.EQ.J) UNIT_MATRIX (,,I,I) = 1.0
10 CONTINUE

C

C SELECT PIVOT COLUMN AND PIVOT ELEMENT

C DO 90 K = 1, N-1
COLUMN_PIVOT (,,K) = K
PIVOT_ELEMENT = JT (,,K,K)
K1 = K + 1
DO 30 I = K1, N
SWAP = ABS(PIVOT_ELEMENT) - ABS(JT (,,I,K)).LT.O.O
PIVOT_ELEMENT (SWAP) = JT (,,I,K)
30 COLUMN_PIVOT (SWAP,K) = I
CHECK FOR SINGULARITY

IF (ALL(PIVOT_ELEMENT.NE.0.0)) GO TO 40
DETXT_JT(PIVOT_ELEMENT.EQ.0.0) = 0.0
RETURN

INTERCHANGE COLUMNS

DO 50 I = K1, N
SWAP = COLUMN_PIVOT(,,K).EQ.I
IF (.NOT.ANY(SWAP)) GO TO 50
DO 45 J = 1, N
TEMP = JT(,,K,J)
JT(SWAP,K,J) = JT(,,I,J)
JT(SWAP, I,J) = TEMP
TEMP = UNIT_MATRIX(,,K,J)
UNIT_MATRIX(SWAP,K,J) = UNIT_MATRIX(,,I,J)
UNIT_MATRIX(SWAP,I,J) = TEMP
50 CONTINUE

DIVIDE COLUMN BY PIVOT

DO 60 J = 1, N
JT(,,K,J) = JT(,,K,J)/PIVOT_ELEMENT
60 UNIT_MATRIX(,,K,J) = UNIT_MATRIX(,,K,J)/PIVOT_ELEMENT

REDUCE MATRIX
DO 80 I = K1, N
AA = JT(,,I,K)
DO 65 J = K, N
65 JT(,,I,J) = JT(,,I,J) - AA * JT(,,K,J)
DO 70 J = 1, N
70 UNIT_MATRIX (,,I,J) = UNIT_MATRIX (,,I,J) - AA * UNIT_MATRIX (,,K,J)
80 CONTINUE

C

CALCULATE DETERMINANT

C

SIGN_CHANGE = .FALSE.
DO 85 I = K1, N
85 SIGN_CHANGE = SIGN_CHANGE .OR. (COLUMN_PIVOT (,,I).EQ.I)
PIVOT_ELEMENT (SIGN_CHANGE) = - PIVOT_ELEMENT
90 DET_JT = DET_JT * PIVOT_ELEMENT
DET_JT = DET_JT + JT(,,N,N)

C

BACK SUBSTITUTION

C

ATEMP = JT(,,N,N)
DO 95 J = 1, N
95 JT(,,N,J) = UNIT_MATRIX (,,N,J)/ATEMP
DO 120 I = 2, N
II = N+1 - I
IIP1 = II + 1
AA = JT(,,II,II)
DO 100 J = IIP1, N
100 UNIT_MATRIX (,,N,J) = JT(,,II,J)
DO 110 J = 1, N
SIGMA = 0.0

DO 105 K = IIPL, N

105 SIGMA = SIGMA + UNIT_MATRIX (,,N,K) * JT(,,K,J)

110 JT(,,II,J) = (UNIT_MATRIX (,,II,J) - SIGMA)/AA

120 CONTINUE
APPENDIX F

Sets of Non-linear Models

(a) Model (ii), \( n = 5, \quad p = 12 \)

The model is:

\[
\begin{align*}
 f_1^*(y_t) &= \gamma_1 \tan^{-1}(\alpha_{1t}) + \theta_1 y_{1t} + 0.1 \cdot \theta_1^2 y_{2t} \\
 f_2^*(y_t) &= \gamma_2 \tan^{-1}(\alpha_{2t}) + 0.1 \cdot \theta_2^2 y_{1t} + \theta_2 y_{2t} \\
 f_3^*(y_t) &= \gamma_3 \tan^{-1}(\alpha_{3t}) - 0.1 \cdot \theta_3^2 y_{1t} + (\theta_1 + 0.1 \cdot \theta_1^2) y_{3t} \\
 f_4^*(y_t) &= \gamma_4 \tan^{-1}(\alpha_{4t}) + 0.1 \cdot \theta_4^2 y_{1t} + (\theta_1 - 0.1 \cdot \theta_1^2) y_{4t} \\
 f_5^*(y_t) &= \gamma_5 \tan^{-1}(\alpha_{5t}) + 0.1 \cdot \theta_5^2 y_{1t} + 0.1 \cdot \theta_1^2 y_{4t} + \theta_1 y_{5t}
\end{align*}
\]

and

\[
\begin{align*}
 u_{1t} &= (\eta_{11} z_{1t} + z_{6t}^2) + f_1^*(y_t) \\
 u_{2t} &= (\eta_{22} z_{2t} + z_{6t}^2) + f_2^*(y_t) \\
 u_{3t} &= (\eta_{33} z_{3t} + z_{7t}^2) + f_3^*(y_t) \\
 u_{4t} &= (\eta_{44} z_{4t} + z_{7t}^2) + f_4^*(y_t) \\
 u_{5t} &= (\eta_{55} z_{5t} + z_{7t}^2) + f_5^*(y_t)
\end{align*}
\]
Model (iii), $n = 5, p = 6$

The model is:

\[
\begin{align*}
    f_1^*(y_t) &= \gamma \tan^{-1}(a_{y_1t}) + \theta_{y_1t} + 0.1 \cdot \theta^2 y_{2t} \\
    f_2^*(y_t) &= \gamma \tan^{-1}(a_{y_2t}) + 0.1 \cdot \theta^2 y_{1t} + \theta y_{2t} \\
    f_3^*(y_t) &= \gamma \tan^{-1}(a_{y_3t}) - 0.1 \cdot \theta^2 y_{2t} + (\theta + 0.1 \cdot \theta^2)y_{3t} \\
    f_4^*(y_t) &= \gamma \tan^{-1}(a_{y_4t}) + 0.1 \cdot \theta y_{2t} + (\theta - 0.1 \cdot \theta^2)y_{4t} \\
    f_5^*(y_t) &= \gamma \tan^{-1}(a_{y_5t}) + 0.1 \cdot \theta^2 y_{2t} + 0.1 \cdot \theta^2 y_{4t} + \theta y_{5t}
\end{align*}
\]

and

\[
\begin{align*}
    u_{1t} &= (n_{z_{1t}} + z_{6t}^2) + f_1^*(y_t) \\
    u_{2t} &= (n_{z_{2t}} + z_{6t}^2) + f_2^*(y_t) \\
    u_{3t} &= (n_{z_{3t}} + z_{7t}^2) + f_3^*(y_t) \\
    u_{4t} &= (n_{z_{4t}} + z_{7t}^2) + f_4^*(y_t) \\
    u_{5t} &= (n_{z_{5t}} + z_{7t}^2) + f_5^*(y_t)
\end{align*}
\]
(c) Model (v), \( n = 2, \ p = 6 \)

The model is:

\[
\begin{align*}
\hat{f}_1(y_t) &= \gamma \tan^{-1}(\alpha y_{1t}) + (\theta + \theta^2)y_{1t} + \theta^2 y_{2t} \\
\hat{f}_2(y_t) &= \gamma \tan^{-1}(\alpha y_{2t}) - \theta^2 y_{1t} + (\theta + \theta^2)y_{2t}
\end{align*}
\]

and

\[
\begin{align*}
u_{1t} &= (n_{11}z_{1t} + (z_{3t})^\delta) + \hat{f}_1(y_t) \\
u_{2t} &= (n_{22}z_{2t} + (z_{3t})^\delta) + \hat{f}_2(y_t)
\end{align*}
\]

(d) Model (vi), \( n = 2, \ p = 4 \)

The model is:

\[
\begin{align*}
\hat{f}_1(y_t) &= \gamma \tan^{-1}(\alpha y_{1t}) + (\theta + \theta^2)y_{1t} + \theta^2 y_{2t} \\
\hat{f}_2(y_t) &= \gamma \tan^{-1}(\alpha y_{2t}) - \theta^2 y_{1t} + (\theta \cdot \theta^2)y_{2t}
\end{align*}
\]

and

\[
\begin{align*}
u_{1t} &= (n_{11}z_{1t} + z_{3t}^\delta) + \hat{f}_1(y_t) \\
u_{2t} &= (n_{22}z_{2t} + z_{3t}^\delta) + \hat{f}_2(y_t)
\end{align*}
\]
REFERENCES


ICL (1979), DAP: APAL Language.


________ (Nov. 1977), An Introduction to Array Processors, Systems International.

________ (1980), "Practical Parallel Processors and Their Uses", Technical Report, Queen Mary College DAP Support Unit.


