THE EXACT LIKELIHOOD FUNCTION OF A VECTOR ARMA MODEL*

José Alberto Mauricio
Instituto Complutense de Análisis Económico
Universidad Complutense
Campus de Somosaguas
28223 Madrid

ABSTRACT

This paper implements in Fortran 77 a new algorithm which has the same purpose as algorithm AS 242 of Shea (1989), namely to compute the exact likelihood function of a vector ARMA model. The new algorithm turns out to be faster in many relevant cases and not appreciably slower in any. In addition to advantages offered by the algorithm of Shea (1989), including the calculation of an appropiate set of residuals, it also permits the automatic detection of noninvertible models as a byproduct. The Fortran 77 code presented here combines improved versions of the algorithms due to Ljung and Box (1979) and Hall and Nicholls (1980) with an algorithm of Kohn and Ansley (1982). The resulting procedure puts together a set of useful features which can only be found separately in other existing methods.

RESUMEN

En este trabajo se presenta la codificación en Fortran 77 de un nuevo algoritmo para evaluar la función de verosimilitud exacta de un modelo ARMA multivariante. Este algoritmo resulta significativamente más rápido que el SHEA (1982) en muchos casos, mientras que no es claramente más lento en ninguno. Además de proporcionar un vector de residuos apropiado, permite detectar, como subproducto, modelos no invertibles. El código es una combinación de los algoritmos de Ljung y Box (1979) y Hall y Nicholls (1980) mejorados, con un algoritmo de Kohn y Ansley (1982). Como resultado se obtiene un algoritmo con ciertas propiedades que sólo pueden encontrarse por separado en los procedimientos disponibles actualmente.

* This paper is based on the author's doctoral dissertation Evaluación y Maximización de la Función de Verosimilitud de Procesos ARMA Multivariantes (1993, Madrid: Universidad Complutense de Madrid), and was financed under the project for a Forecast and Monitor Service for the Spanish Economy, sponsored by Caja de Madrid (Spain), and by the DGICYT project PB89-0129 (Spain). The author is grateful to Arthur B. Treadway, under whose supervision this research was carried out, and to Sonia Sotoca for some useful hints on the Fortran compiler.

N°E: X-53-229647-7

N°E: 5306620375

1. Introduction

This article presents a detailed implementation in Fortran 77 of a new method of evaluating the exact likelihood function of vector ARMA models. Although each of many existing algorithms (e.g. Ljung and Box 1979; Hall and Nicholls 1980; Shea 1989) has some useful features, it also lacks others that can be found in alternative procedures (see Mauricio 1993). Thus, the aim of this paper has been to put all of these features together into a single algorithm, and present them in a truly operational way. Towards this end, Section 2 summarizes the theoretical issues underlying the Fortran 77 code. Basically, the implemented algorithm is an improved version of the method of Hall and Nicholls (1980) that operates with reduced order matrices, combined with an improved extension of the algorithm of Ljung and Box (1979) that does not require any explicit matrix inversion. Further, the algorithm uses the method of Kohn and Ansley (1982) in order to evaluate the theoretical autocovariance and cross-covariance matrices of the model, which implies a significant advantage over the algorithm of Shea (1989), especially in the case of high-dimension models. The structure of the routines used to carry out all of these computations is described in Section 3. The rest of the paper deals with more technical issues (sections 4 through 7), with the comparison between the new algorithm and that of Shea (1989), which shows the potential superiority of the former (Section 8), and with the application of the algorithm to exact maximum likelihood estimation. The paper ends up with a commented listing of the Fortran 77 code.

2. Theory and Method

Let $\mathbf{w}_t$ be an m-dimensional vector-valued time series. It is assumed that $\mathbf{w}_t$ follows the vector ARMA(p,q) model

$$\mathbf{\Phi}(B)\tilde{\mathbf{w}}_t = \mathbf{\Theta}(B)\mathbf{a}_t\ , \tag{1}$$

where $\mathbf{\Phi}(B) = \mathbf{I} - \mathbf{\Phi}_1 B - \cdots - \mathbf{\Phi}_p B^p$, $\mathbf{\Theta}(B) = \mathbf{I} - \mathbf{\Theta}_1 B - \cdots - \mathbf{\Theta}_q B^q$, B is the back shift operator, $\tilde{\mathbf{w}}_t = \mathbf{w}_t - \mu$, $\mathbf{\Phi}_i$ $(i = 1, 2, \ldots, p)$, $\mathbf{\Theta}_i$ $(i = 1, 2, \ldots, q)$ and μ are $m\times m$, $m\times m$ and $m\times 1$ parameter matrices, respectively, and the $\mathbf{a}_t$'s are $m\times 1$ random vectors identically and independently distributed as $N(\mathbf{0}, \sigma^2\mathbf{Q})$, with $\sigma^2 > 0$ and $\mathbf{Q}$ $(m\times m)$ symmetric and positive definite. For stationarity, it is required that the zeros of $|\mathbf{\Phi}(B)|$ lie outside the unit circle. Likewise, the model will be invertible provided that the zeros of $|\mathbf{\Theta}(B)|$ lie outside the unit circle.

Consider a sample of size n and define $\tilde{\mathbf{w}} = (\tilde{\mathbf{w}}_1^T, \ldots, \tilde{\mathbf{w}}_n^T)^T$ (mean-corrected observations), $\mathbf{a} = (\mathbf{a}_1^T, \ldots, \mathbf{a}_n^T)^T$ (white noise perturbations), and $\mathbf{u}_* = (\tilde{\mathbf{w}}_{1-p}^T, \ldots, \tilde{\mathbf{w}}_0^T, \mathbf{a}_{1-q}^T, \ldots, \mathbf{a}_0^T)^T$ (unknown presample values). Then equation (1) may be written as

$$\mathbf{D}_{\Phi,n}\tilde{\mathbf{w}} = \mathbf{D}_{\Theta,n}\mathbf{a} + \mathbf{V}\mathbf{u}_* ,$$

where $\mathbf{D}_{\Phi,n}$ and $\mathbf{D}_{\Theta,n}$ are $nm \times nm$ block-matrices with identity matrices on the main diagonal, and $-\Phi_k$ and $-\Theta_k$, respectively, down the *k-th* subdiagonal. Further, $\mathbf{V}$ is the $nm \times (p+q)m$ block-matrix $\mathbf{V} = (\mathbf{G}_{\Phi,n}, \mathbf{G}_{\Theta,n})$, where $\mathbf{G}_{\Phi,n}$ and $\mathbf{G}_{\Theta,n}$ are the following $nm \times pm$ and $nm \times qm$ block-matrices:

$$\mathbf{G}_{\Phi,n} = \begin{bmatrix} \Phi_p & \Phi_{p-1} & \cdots & \Phi_1 \\ \mathbf{0} & \Phi_p & \cdots & \Phi_2 \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \Phi_p \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} , \qquad \mathbf{G}_{\Theta,n} = \begin{bmatrix} -\Theta_q & -\Theta_{q-1} & \cdots & -\Theta_1 \\ \mathbf{0} & -\Theta_q & \cdots & -\Theta_2 \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -\Theta_q \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} .$$

On the basis of the previous definitions, Nicholls and Hall (1979) have shown that the exact log-likelihood of the parameters $\Phi = (\Phi_1, \ldots, \Phi_p)$, $\Theta = (\Theta_1, \ldots, \Theta_q)$, μ, σ^2 and $\mathbf{Q}$ has the form

$$l(\Phi,\Theta,\mu,\sigma^2,\mathbf{Q}|\mathbf{w}) = -\frac{1}{2}\left[nm\log(2\pi\sigma^2) + n\log|\mathbf{Q}| + \log|\mathbf{A}^T\mathbf{A}| + \frac{1}{\sigma^2}S(\Phi,\Theta,\mu,\mathbf{Q}|\mathbf{w}) \right] , \quad (2)$$

where, by means of exploring in further detail the papers by Ljung and Box (1979) and Hall and Nicholls (1980), it can be shown (see Mauricio 1993) that

$$S(\Phi,\Theta,\mu,\mathbf{Q}|\mathbf{w}) = \eta^T\eta - (\mathbf{M}^T\bar{\mathbf{h}})^T(\mathbf{I} + \mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M})^{-1}(\mathbf{M}^T\bar{\mathbf{h}}) \quad (3)$$

and

$$|\mathbf{A}^T\mathbf{A}| = |\mathbf{I} + \mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M}| . \quad (4)$$

Subroutine ELF2 computes the expression on the right-hand side of equation (2) using (3) and (4). Also, it optionally returns the residual vector, computed from the following expression:

$$\hat{\mathbf{a}} = \hat{\mathbf{a}}_0 - \mathbf{D}_{\Theta,n}^{-1}\begin{bmatrix} \mathbf{M}(\mathbf{I} + \mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M})^{-1}\mathbf{M}^T\bar{\mathbf{h}} \\ \mathbf{0} \end{bmatrix} . \quad (5)$$

In order to evaluate equations (3), (4) and (optionally) (5), the following steps are performed within subroutine ELF2:

[1] Compute the Cholesky factor $\mathbf{Q}_1$ of matrix $\mathbf{Q}$ (i.e. $\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_1^T$), its determinant $|\mathbf{Q}| = |\mathbf{Q}_1|^2$ and a matrix $\mathbf{R}$ such that $\mathbf{R}\mathbf{Q}\mathbf{R}^T = \mathbf{I}$ (i.e. $\mathbf{R} = \mathbf{Q}_1^{-1}$).

[2] Evaluate the theoretical autocovariances $\Gamma(k) = \sigma^{-2}E[\tilde{\mathbf{w}}_t\tilde{\mathbf{w}}_{t+k}]$ ($k = 0, 1, \ldots, p-1$) and the theoretical cross-covariances $\Gamma_{\mathrm{wa}}(k) = \sigma^{-2}E[\tilde{\mathbf{w}}_t\mathbf{a}_{t+k}^T]$ ($k = 0, -1, \ldots, -q+1$).

[3] Compute the symmetric $gm \times gm$ matrix $\mathbf{V}_1\Omega\mathbf{V}_1^T$, with $g = \max(p,q)$, where $\mathbf{V}_1$ consists of the first gm rows of $\mathbf{V}$ and $\Omega = \sigma^{-2}E[\mathbf{u}_*\mathbf{u}_*^T]$. Then, calculate its Cholesky factor $\mathbf{M}$ (i.e. $\mathbf{V}_1\Omega\mathbf{V}_1^T = \mathbf{M}\mathbf{M}^T$). The *(i, j)-th* $m \times m$ block of $\mathbf{V}_1\Omega\mathbf{V}_1^T$ ($i = 1, 2, \ldots, g$; $j = 1, 2, \ldots, i$) is given by

$$(\mathbf{V}_1\Omega\mathbf{V}_1^T)_{ij} = \sum_{k=0}^{p-i}\Phi_{p-k}\mathbf{E}_{k+i,j} - \sum_{k=0}^{q-i}\Theta_{q-k}\mathbf{E}_{k+p+i,j} ,$$

where, for $j = 1, 2, \ldots, g$, the required $m \times m$ $\mathbf{E}_{ij}$ matrices are given by

$$\mathbf{E}_{ij} = \sum_{k=j-i}^{p-i}\Gamma(k)\Phi_{p-k-i+j}^T - \sum_{k=j-i}^{q-i}\Gamma_{\mathrm{wa}}(-q+p+k)\Theta_{q-k-i+j}^T \quad (i=1,2,\ldots,p) ,$$

$$\mathbf{E}_{ij} = \sum_{k=p+j-i}^{2p-i}\Gamma_{\mathrm{wa}}(-q+p-k)^T\Phi_{2p-k-i+j}^T - \mathbf{Q}\Theta_{q+p-i+j}^T \quad (i=p+1,p+2,\ldots,p+q) ,$$

with $\Gamma(k) = \Gamma(-k)^T$ for $k < 0$, $\Gamma_{\mathrm{wa}}(k) = \mathbf{0}$ for $k > 0$, and $\Theta_i = \mathbf{0}$ for $i > q$.

[4] Evaluate the $m \times m$ matrices Ξ_k ($k = 0, 1, \ldots, n-1$) as follows:

$$\Xi_k = \sum_{j=1}^{q}\Theta_j\Xi_{k-j} \quad (k = 1,2,\ldots,n-1) ,$$

with $\Xi_0 = \mathbf{I}$ and $\Xi_k = \mathbf{0}$ for $k < 0$. Then, premultiply every Ξ_k by (lower triangular) $\mathbf{R}$.

[5] Calculate the $m \times 1$ vectors $\eta_i = \mathbf{R}\hat{\mathbf{a}}_{0i}$ ($i = 1, 2, \ldots, n$), where

$$\hat{\mathbf{a}}_{0i} = \tilde{\mathbf{w}}_i - \sum_{j=1}^{p}\Phi_j\tilde{\mathbf{w}}_{i-j} + \sum_{j=1}^{q}\Theta_j\hat{\mathbf{a}}_{0,i-j} \quad (i=1,2,\ldots,n) ,$$

with $\tilde{\mathbf{w}}_i = \mathbf{0}$ for $i < 1$ and $\hat{\mathbf{a}}_{0i} = \mathbf{0}$ for $i < 1$.

[6] Compute the $m \times 1$ vectors $\bar{\mathbf{h}}_i$ ($i = 1, 2, \ldots, g$) as follows:

$$\bar{\mathbf{h}}_i = \sum_{j=0}^{n-i}\Xi_j^T\mathbf{R}^T\eta_{i+j} \quad (i=1,2,\ldots,g) .$$

Then, evaluate the $gm \times 1$ vector $\mathbf{M}^T\bar{\mathbf{h}}$, where $\bar{\mathbf{h}} = (\bar{\mathbf{h}}_1^T, \ldots, \bar{\mathbf{h}}_g^T)^T$.

[7] Evaluate the symmetric $gm \times gm$ matrix $\mathbf{H}^T\mathbf{H}$. The first block-column of $\mathbf{H}^T\mathbf{H}$ is given by

$$(\mathbf{H}^T\mathbf{H})_{i1} = \sum_{k=0}^{n-i} \mathbf{\Xi}_k^T \mathbf{R}^T \mathbf{R} \mathbf{\Xi}_{k+i-1} \quad (i=1,2,\ldots,g) \ ,$$

and the remaining diagonal and subdiagonal $m \times m$ blocks of matrix $\mathbf{H}^T\mathbf{H}$ are given by

$$(\mathbf{H}^T\mathbf{H})_{ij} = (\mathbf{H}^T\mathbf{H})_{i-1,j-1} - \mathbf{\Xi}_{n-i+1}^T \mathbf{R}^T \mathbf{R} \mathbf{\Xi}_{n-j+1} \ ,$$

with $i = 2, 3, \ldots, g$ and $j = 2, 3, \ldots, i$.

[8] Evaluate the symmetric $gm \times gm$ matrix $\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M}$, its Cholesky factor $\mathbf{L}$ (i.e. $\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M} = \mathbf{L}\mathbf{L}^T$) and its determinant $|\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M}| = |\mathbf{L}|^2$, which is, in turn, the determinant (4).

[9] Use forward substitution to solve for λ in the triangular system $\mathbf{L}\lambda = \mathbf{M}^T\tilde{\mathbf{h}}$.

[10] Compute the quadratic form (4) as $S(\mathbf{\Phi},\mathbf{\Theta},\mu,\mathbf{Q}\,|\,\mathbf{w}) = \eta^T\eta - \lambda^T\lambda$, where $\eta = (\eta_1^T, \ldots, \eta_n^T)^T$.

[11] Use backward substitution to solve for $\mathbf{c}$ in the triangular system $\mathbf{L}^T\mathbf{c} = \lambda$, calculate the $gm \times 1$ vector $\mathbf{d} = \mathbf{M}\mathbf{c}$, and compute $\hat{\mathbf{a}}_i = \hat{\mathbf{a}}_{0i} - \mathbf{r}_i$ $(i = 1, 2, \ldots, n)$, with the $m \times 1$ vectors $\mathbf{r}_i$ given by

$$\mathbf{r}_i = \sum_{j=1}^{i} \mathbf{\Xi}_{i-j}\mathbf{d}_j \quad (i=1,2,\ldots,n) \ ,$$

where $\mathbf{d}_j$ is the j-*th* $m \times 1$ block of $\mathbf{d}$ $(j = 1, 2, \ldots, g)$ and $\mathbf{d}_j = \mathbf{0}$ for $j > g$.

Subroutines CGAMMA, CXI and CRES are called from within subroutine ELF2 in order to carry out steps **[2]**, **[4]** and **[11]** (optional), respectively. Some matrix computations are performed by subroutines CHOLDC (which is called from within ELF2 to carry out steps **[1]**, **[3]** and **[8]**), CHOLFR (which is called from within ELF2 to carry out steps **[1]** and **[9]**) and CHOLBK (which is called from within CRES to carry out step **[11]**). Subroutine CGAMMA implements an algorithm due to Kohn and Ansley (1982), in order to evaluate the matrices $\mathbf{\Gamma}(k)$ $(k = 0, 1, \ldots, p-1)$ and $\mathbf{\Gamma}_{\mathrm{wa}}(k)$ $(k = 0, -1, \ldots, -q+1)$ in a computationally more efficient manner than the one coded in Shea (1989). The procedure in subroutine CGAMMA solves the following system of linear equations

$$\mathbf{\Gamma}(0) - \sum_{i=1}^{p} \mathbf{\Phi}_i \mathbf{\Gamma}(0) \mathbf{\Phi}_i^T - \sum_{i=1}^{p-1}\sum_{j=1}^{p-i} [\, \mathbf{\Phi}_{i+j}\mathbf{\Gamma}(i)\mathbf{\Phi}_j^T + \mathbf{\Phi}_j\mathbf{\Gamma}(i)^T\mathbf{\Phi}_{i+j}^T \,] = \mathbf{W}_0 \ , \qquad (6.1)$$

$$\mathbf{\Gamma}(k) - \sum_{i=1}^{k-1} \mathbf{\Gamma}(i)\mathbf{\Phi}_{k-i}^T - \sum_{i=0}^{p-k} \mathbf{\Gamma}(i)^T\mathbf{\Phi}_{k+i}^T = \mathbf{W}_k \quad (k = 1, 2, \ldots, p-1) \ , \qquad (6.2)$$

for $\tilde{\mathbf{\Gamma}}(0), \mathbf{\Gamma}(1), \ldots, \mathbf{\Gamma}(p-1)$, where $\tilde{\mathbf{\Gamma}}(0)$ is the diagonal and upper triangle of $\mathbf{\Gamma}(0)$,

$$\mathbf{W}_0 = \mathbf{Q} - (\mathbf{A} + \mathbf{A}^T) + \sum_{j=1}^{q} \mathbf{\Theta}_j \mathbf{Q} \mathbf{\Theta}_j^T \ ,$$

$$\mathbf{A} = \sum_{i=1}^{p}\sum_{j=i}^{q} \mathbf{\Phi}_i \mathbf{\Gamma}_{\mathrm{wa}}(i-j) \mathbf{\Theta}_j^T \ ,$$

$$\mathbf{W}_k = -\sum_{j=k}^{q} \mathbf{\Gamma}_{\mathrm{wa}}(k-j)\mathbf{\Theta}_j^T \quad (k = 1, 2, \ldots, p-1) \ ,$$

$$\mathbf{\Gamma}_{\mathrm{wa}}(-k) = -\mathbf{\Theta}_k\mathbf{Q} + \sum_{i=1}^{k} \mathbf{\Phi}_i \mathbf{\Gamma}_{\mathrm{wa}}(i-k) \quad (k = 1, 2, \ldots, q-1) \ ,$$

with $\mathbf{\Phi}_i = \mathbf{0}$ if $i > p$ and $\mathbf{\Gamma}_{\mathrm{wa}}(0) = \mathbf{Q}$ (note that $\mathbf{\Gamma}_{\mathrm{wa}}(k) = \sigma^{-2}E[\bar{\mathbf{w}}_t\mathbf{a}_{t+k}^T] = \mathbf{0}$ for $k > 0$). The resulting system contains $m(m+1)/2 + m^2(p-1)$ unknowns. Thus, subroutine CGAMMA solves for $m(m-1)/2 + m^2$ less unknowns than subroutine COVARS of Shea (1989), which unnecessarily solves for $\mathbf{\Gamma}(0)$ through $\mathbf{\Gamma}(p)$ instead of $\tilde{\mathbf{\Gamma}}(0)$ and $\mathbf{\Gamma}(1)$ through $\mathbf{\Gamma}(p-1)$.

3. Structure

SUBROUTINE ELF1(M, P, Q, N, W, PHI, THETA, QQ, ISMU, MU, ATF, A, SIGMA2, XITOL, LOGELF, F1, F2, WS, NWS, IWS, NIWS, IFAULT)

Formal parameters:

M	Integer	input:	the number of time series, m (≥ 1).
P	Integer	input:	the value of p (≥ 0).
Q	Integer	input:	the value of q (≥ 0), but $p = q = 0$ is not allowed.
N	Integer	input:	the length of each series, n (≥ 1).
W	Real array of dimension (*M, N*)	input:	on entry, *W(I, J)* must contain the *I-th* component of $\mathbf{w}_J$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, N$.
PHI	Real Array of dimension (*M, P * M + 1*)	input:	on entry, *PHI(I, (K − 1) * M + J)* must contain the *(I, J)-th* element of $\mathbf{\Phi}_K$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 1, 2, \ldots, P$.

THETA	Real Array of dimension $(M, Q * M + 1)$	input:	on entry, *THETA*$(I, (K-1) * M + J)$ must contain the (I, J)-*th* element of $\mathbf{\Theta}_K$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 1, 2, \ldots, Q$.
QQ	Real Array of dimension (M, M)	input/output:	on entry, *QQ*(I, J) must contain the (I, J)-*th* element of $\mathbf{Q}$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, I$ (i.e. the lower triangle of $\mathbf{Q}$); on exit, if *IFAULT* = 0 or *IFAULT* ≥ 8 then the strict upper triangle of *QQ* is set equal to its lower triangle.
ISMU	Logical	input:	set equal to .TRUE. if $\mu \neq \mathbf{0}$ and .FALSE. if $\mu = \mathbf{0}$.
MU	Real array of dimension *M*	input:	if *ISMU* is set equal to .TRUE. then *MU*(I) must contain the *I-th* component of μ for $I = 1, 2, \ldots, M$; if *ISMU* is set equal to .FALSE. then *MU* is not used.
ATF	Logical	input:	set equal to .TRUE. if computation of the residual vector is required and .FALSE. otherwise.
A	Real array of dimension (M, N)	output:	if *ATF* is set equal to .TRUE. then, on successful exit, $A(I, J)$ contains the *I-th* component of $\hat{\mathbf{a}}_J$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, N$; if *ATF* is set equal to .FALSE. then, if *IFAULT* = 0 or *IFAULT* = 12, $A(I, J)$ contains the *I-th* component of η_J for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, N$.
SIGMA2	Real	input:	the value of σ^2.
XITOL	Real	input:	convergence tolerance for the $\mathbf{\Xi}_k$'s; on entry, it must be set to any negative number if an exact evaluation of the log-likelihood is desired; if an approximate evaluation is desired or if $q = 0$ then it should be set to a small positive number.
LOGELF	Real	output:	on successful exit, contains the value of the log-likelihood function (2).

F1	Real	output:	on successful exit, contains the value of the quadratic form (3).
F2	Real	output:	on successful exit, contains the value of $\lvert\mathbf{Q}\rvert^n$ times the determinant (4).
WS	Real array of dimension at least *NWS*	workspace:	
NWS	Integer	input:	the dimension of the array *WS* as declared in the user's calling (sub)program: $NWS \geq M * M * (3 + 3 * G * G + (P + Q) * G + B4) + B1 * B1 + B2 + B3$, where $G = \max(P, Q)$, $B1 = M * (M + 1) / 2 + M * M * (P - 1)$ if $P > 0$ and $B1 = 1$ otherwise, $B2 = \max(B1, G * M)$, $B3 = \max(B1, M)$ and $B4 = \max(N, Q)$.
IWS	Integer array of dimension at least *NIWS*	workspace:	
NIWS	Integer	input:	the dimension of the array *IWS* as declared in the user's calling (sub)program: $NIWS \geq B1$, where *B1* is as for *NWS*.
IFAULT	integer	output:	a fault indicator equal to 1 if $M < 1$; 2 if $N < 1$; 3 if $P < 0$; 4 if $Q < 0$; 5 if $P = 0$ and $Q = 0$; 6 if *NWS* is too small; 7 if *NIWS* is too small; 8 if *QQ* is not positive definite; 9 if equations (6) for calculating the $\mathbf{\Gamma}(k)$'s could not be solved (this indicates that the AR parameters are very close to the boundary of the stationarity region); 10 if the matrix $\mathbf{V}_1\mathbf{\Omega}\mathbf{V}_1^T$ is not positive definite (this indicates that the model

is not stationary);

11 if the Ξ_k's turn out to be explosive (this indicates that the model is not invertible);

12 if the matrix $\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M}$ is not positive definite;

0 otherwise (on a successful exit).

Subroutine ELF1 checks for errors in the input parameters, sets up workspace arrays, calls subroutine MACHEP to compute *machine epsilon* (used by subroutine CHOLDC) and then calls subroutine ELF2 to evaluate the exact log-likelihood function. A description of the formal parameters of ELF2 is given next.

SUBROUTINE ELF2(M, P, Q, N, G, W, PHI, THETA, QQ, ISMU, MU, ATF, A, SIGMA2, XITOL, LOGELF, F1, F2, EPS, BIG1, BIG2, BIG3, BIG4, Q1, Q1INV, MTMP4, VTMP1, VTMP2, MATPHI, MTMP0, MTPM2, MTMP3, MTMP1, GAMXI, INDX, IFAULT)

Formal parameters:

M, P, Q, N		input:	as for ELF1.
G	Integer	input:	max(*P*, *Q*), set by ELF1.
W, PHI, THETA		input:	as for ELF1.
QQ		input/output:	as for ELF1.
ISMU, MU, ATF		input:	as for ELF1.
A		output:	as for ELF1; *A* is given values in ELF2.
SIGMA2, XITOL		input:	as for ELF1.
LOGELF, F1, F2		output:	as for ELF1; *LOGELF*, *F1* and *F2* are given values in ELF2.
EPS	Real	input:	machine epsilon (set within ELF1).
BIG1	Integer	input:	$BIG1 = M * (M + 1) / 2 + M * M (P - 1)$ if $P > 0$ and $BIG1 = 1$ otherwise; this value is set by ELF1.
BIG2	Integer	input:	max(*BIG1*, *G* * *M*) (set by ELF1).
BIG3	Integer	input:	max(*BIG1*, *M*) (set by ELF1).

BIG4	Integers	input:	max(*N*, *Q*) (set by ELF1).
Q1, Q1INV, MTMP4, VTMP1, VTMP2, MATPHI, MTMP0, MTMP2, MTMP3, MTMP1, GAMXI	Real workspace arrays		
INDX	Integer workspace array		
IFAULT	Integer	output:	a fault indicator; on exit from ELF2, *IFAULT* will either have the value 8, 9, 10, 11, 12 or 0.

Subroutine ELF2 implements the steps described in the previous section in order to calculate the log-likelihood function. One of these steps is the computation of the theoretical covariance and cross-covariance matrices $\mathbf{\Gamma}(k)$ ($k = 0, 1, \ldots, p-1$) and $\mathbf{\Gamma}_{wa}(k)$ ($k = 0, -1, \ldots, -q+1$). This task is done by subroutine CGAMMA, which implements an algorithm due to Kohn and Ansley (1982).

SUBROUTINE CGAMMA(M, P, Q, PHI, THETA, QQ, BIG1, MAT, VV, WZERO, MZERO, GAMWA, RHS, INDX, IFAULT)

Formal parameters:

M, P, Q, PHI, THETA, QQ, BIG1		input:	as for ELF2.
MAT, VV, WZERO, MZERO, GAMWA, RHS, INDX	Workspace arrays		

GAMWA		output:	on successful exit, GAMWA(*I*, *K* $*$ *M* + *J*) contains the (*I*, *J*)-*th* element of $\boldsymbol{\Gamma}_{\text{wa}}(-K)$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 0, 1, \ldots, Q-1$.
RHS		output:	on successful exit, *RHS*(*J* $*$ (*J* − 1) / 2 + *I*) contains the (*I*, *J*)-*th* element of $\boldsymbol{\Gamma}(0)$ for $I = 1, 2, \ldots, M$, $J = I, I+1, \ldots, M$, and *RHS*(*M* $*$ (*M* + 1) / 2 + *M* $*$ *M* $*$ (*K* − 1) + *M* $*$ (*J* − 1) + *I*) contains the (*I*, *J*)-*th* element of $\boldsymbol{\Gamma}(K)$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 1, 2, \ldots, P-1$.
IFAULT	Integer	output:	a fault indicator equal to 1 if the equations (6) could not be solved (in which case *IFAULT* is returned from ELF2 as 9); 0 otherwise (on successful exit).

Subroutine ELF2 also calls subroutine CXI, which computes recursively the matrix sequence $\boldsymbol{\Xi}_k$ $(k = 0, 1, \ldots, n-1)$. This is a key step in order to check for invertibility of the model.

SUBROUTINE CXI(M, N, Q, THETA, XITOL, R, NLIM, XI, MTMP, IFAULT)

Formal parameters:

M, N, Q *THETA,* *XITOL*		input:	as for ELF2.
R		input:	on entry, *R*(*I*, *J*) must contain the (*I*, *J*)-*th* element of **R** for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, I$, as set by ELF2.
NLIM	Integer	output:	on successful exit, *NLIM* is such that $\boldsymbol{\Xi}_k = \mathbf{0}$ for $k > NLIM$.
XI, MTMP	Real workspace arrays		

XI		output:	on successful exit, *XI*(*I*, *M* $*$ *K* + *J*) contains the (*I*, *J*)-*th* element of $\mathbf{R}\boldsymbol{\Xi}_K$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 0, 1, \ldots, NLIM$.
IFAULT	Integer	output:	a fault indicator equal to 1 if the computation of the sequence $\boldsymbol{\Xi}_k$ turns out to be explosive (in which case *IFAULT* is returned from ELF2 as 11); 0 otherwise (on successful exit).

Optionally (if *ATF* was set equal to .TRUE. on entry to ELF1), subroutine ELF2 ends up with a call to subroutine CRES, in order to calculate the residual vector using some of the computations carried so far to evaluate the log-likelihood.

SUBROUTINE CRES(M, N, G, NLIM, XI, Q1, MATM, MATL, LAMBDA, RES)

Formal parameters:

M, N, G		input:	as for ELF2.
NLIM	Integer	input:	on entry, *NLIM* must be such that $\boldsymbol{\Xi}_k = \mathbf{0}$ for $k > NLIM$, as set by CXI from within ELF2.
XI	Real workspace array	input:	on entry, *XI*(*I*, *M* $*$ *K* + *J*) must contain the (*I*, *J*)-*th* element of $\mathbf{R}\boldsymbol{\Xi}_K$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, M$, $K = 0, 1, \ldots, NLIM$, as set by CXI from within ELF2.
Q1	Real workspace array	input:	on entry, *Q1*(*I*, *J*) must contain the (*I*, *J*)-*th* element of the Cholesky factor $\mathbf{Q}_1$ of **Q** for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, I$, as set by ELF2.
MATM	Real workspace array	input:	on entry, *MATM*(*I*, *J*) must contain the (*I*, *J*)-*th* element of the Cholesky factor **M** of $\mathbf{V}_1\boldsymbol{\Omega}\mathbf{V}_1^T$ for $I = 1, 2, \ldots, G * M$, $J = 1, 2, \ldots, I$, as set by ELF2.

MATL	Real workspace array	input:	on entry, *MATL(I, J)* must contain the *(I, J)-th* element of the Cholesky factor **L** of $\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{H}\mathbf{M}$ for $I = 1, 2, \ldots, G*M$, $J = 1, 2, \ldots, I$, as set by ELF2.
LAMBDA	Real workspace array	input:	on entry, *LAMBDA(I)* must contain the *I-th* element of λ (the solution of the triangular system $\mathbf{L}\lambda = \mathbf{M}^T\tilde{\mathbf{h}}$) for $I = 1, 2, \ldots, G*M$, as computed within ELF2.
RES	Real array of dimension *(M, N)*	input/output:	on entry, *A(I, J)* must contain the *I-th* component of η_J for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, N$, as set by ELF2; on exit, *A(I, J)* contains the *I-th* component of $\hat{\mathbf{a}}_J$ for $I = 1, 2, \ldots, M$, $J = 1, 2, \ldots, N$.

The following subroutines are also called from within ELF2. Subroutine CHOLDC returns the Cholesky factor of a symmetric real matrix **M**, and its determinant in the form $a2^b$. Subroutine CHOLFR solves for **x** in the system $\mathbf{Lx} = \mathbf{b}$, with **L** lower triangular, using forward substitution.

SUBROUTINE CHOLDC(M, N, D1, D2, EPS, IFAULT)

Formal parameters:

M	Real array of dimension *(N, N)*	input/output:	on entry, contains the matrix **M** (only the upper triangle is needed); on successful exit, contains the required Cholesky factor with the strict upper triangle set to zero.
N	Integer	input:	the order of **M**.
D1	Real	output:	on successful exit, *D1* is set equal to *a* in the expression $\lvert\mathbf{M}\rvert = a2^b$.
D2	Real	output:	on successful exit, *D2* is set equal to *b* in the expression $\lvert\mathbf{M}\rvert = a2^b$.
EPS	Real	input:	machine epsilon, as set by MACHEP with a call from within ELF1.
IFAULT	Integer	output:	a fault indicator equal to 1 if **M** is not positive definite; 0 otherwise (on a successful exit).

SUBROUTINE CHOLFR(MATL, N, RHSOL)

Formal parameters:

MATL	Real array of dimension *(N, N)*	input:	on entry, contains the lower triangular matrix **L**.
N	Integer	input:	the order of **L**.
RHSOL	Real array of dimension *N*	input/output:	on entry, contains the right-hand side vector **b**; on exit, contains the solution vector **x**.

Finally, subroutine CHOLBK is called from within subroutine CRES to solve for **c** in the system $\mathbf{L}^T\mathbf{c} = \lambda$, with **L** lower triangular, using backward substitution. Subroutine MACHEP is called from within subroutine ELF1 to calculate machine epsilon.

SUBROUTINE CHOLBK(MATL, N, RHSOL)

Formal parameters:

MATL	Real array of dimension *(N, N)*	input:	on entry, contains the lower triangular matrix **L**.
N	Integer	input:	the order of **L**.
RHSOL	Real array of dimension *N*	input/output:	on entry, contains the right-hand side vector λ; on exit, contains the solution vector **c**.

SUBROUTINE MACHEP (EPSIL)

Formal parameter:

EPSIL	Real	output:	machine epsilon.

4. Auxiliary Algorithms

Subroutines DECOMP and SOLVE in Moler (1972) are used to solve the system of linear equations (6) within subroutine CGAMMA.

5. Restrictions

Subroutine ELF2 will terminate abnormally if the model turns out to be either non-stationary or non-invertible. The model will be non-stationary if the matrix $\mathbf{V}_1\Omega\mathbf{V}_1^T$ is not positive definite. This is detected by subroutine CHOLDC, which in turn will make ELF2 to return *IFAULT* = 10. The model will be non-invertible if the computation of the matrix sequence Ξ_k is explosive, i.e. when

$$\left[\sum_{i=1}^{m}\sum_{j=1}^{m} |\Xi_h(i,j)|\right] > \sum_{k=1}^{\min(h,q)} \left[\sum_{i=1}^{m}\sum_{j=1}^{m} |\Xi_k(i,j)|\right]$$

for at least one $h < n - 1$. This check for non-invertibility takes place within subroutine CXI and if it holds, then ELF2 will return *IFAULT* = 11. Note, however, that the exact likelihood can still be evaluated when any root of the moving average operator lies on the unit circle, provided the other roots have moduli larger than unity.

To calculate the log-likelihood function for the model

$$\mathbf{w}_t = \mu + \mathbf{a}_t \,,$$

(i.e. to take $p = q = 0$), q should be set to 1 and the first m rows and columns of *THETA* to zero (note that $p = q = 0$ is not allowed by subroutine ELF1 since some of the workspace arrays in ELF2 would have zero length).

Although subroutine ELF1 will not flag an error if N is set to 1, it is left to the user to ensure that $N * M$ is greater than the number of parameters in the model.

6. Precision

When using the present algorithm on machines with small word length, all the real variables should be replaced by double precision variables. This amounts to replacing all declarations of type REAL by declarations of type DOUBLE PRECISION and all remaining occurrences of REAL by DBLE. In order to make an accurate and machine-independent decision on whether a given symmetric real matrix is not positive definite, subroutine CHOLDC makes use of the quantity machine epsilon as discussed in Dennis and Schnabel (1983, pp. 318-9). Overflow or underflow will not occur in the calculation of $|\mathbf{Q}|$ and $|\mathbf{I}+\mathbf{M}^T\mathbf{H}^T\mathbf{HM}|$ since these determinants are stored in the form $a2^b$ (Martin and Wilkinson, 1965).

7. Accuracy

When the model considered is invertible, the matrix sequence Ξ_k converges to $\mathbf{0}$, the more quickly the larger the moduli of the zeros of $|\Theta(B)|$ are (when $q = 0$, it is clear that $\Xi_k = \mathbf{0}$ for $k \geq 1$). This fact may be exploited in the subsequent computations involved in steps **[6]**, **[7]** and **[11]** within subroutine ELF2, since if $\Xi_k = \mathbf{0}$ for, say, $k > r^*$, then not all of these operations need to be carried out. The sequence Ξ_k may be considered to have converged when

$$\left[\sum_{i=1}^{m}\sum_{j=1}^{m} |\Xi_{r^*+1}(i,j)|\right] < \delta \,,$$

where the parameter $\delta > 0$ can be used to control the desired degree of approximation to the exact computation of the whole sequence. However, to avoid complications due to the presence of any $\Theta_i = \mathbf{0}$ for $i < q$ (i.e. in the case of seasonal or gapped models), once the above condition is satisfied, subroutine CXI calculates the next q Ξ_k's following Ξ_{r^*} in order to make sure that convergence has effectively occurred. The convergence criterion can be made sufficiently rigid (i.e. δ sufficiently small) that the error implied by considering $\Xi_k = \mathbf{0}$ for $k > r^*$, becomes negligible.

This property, which may save much computing time, is analogous to the 'quick recursions' property offered by the Chandrasekhar equations that form the basis of the method of Shea (1989, pp. 169-170). Thus, the comparisons between the exact (calculated with Ξ_k from $k = 1$ to $k = n-1$) and the 'approximate' (calculated with $\Xi_k = \mathbf{0}$ for $k > r^*$) log-likelihood functions for any given model, display results which are very similar to those reported by Shea (1989).

8. Timing and Related Algorithm

The algorithm implemented in subroutine ELF2 is faster than Algorithm AS 242 of Shea (1989) in many cases. To see this, the exact log-likelihood function has been evaluated for a variety of vector ARMA models. In Table 1, the ratio between the number of multiplications and divisions required by the algorithm of Shea (1989) and those required by subroutine ELF2, is presented for each of the models considered.

It can be seen that Algorithm AS 242 is slightly faster than the new algorithm only for low dimension ($m = 2$) high order models, whereas the new algorithm is faster (by a factor of more than two in many cases) for higher dimension ($m = 4$) models. In fact, the relative efficiency of the new algorithm increases with the dimension m of the model (and, almost in all cases, with the series length n too). Thus, the ratios for $m = 4$ and $n = 200$ are all advantageous to the new algorithm, irrespective of the orders p and q.

9. Additional Comments

The main purpose of subroutine ELF2 is to serve as an integral part of an exact maximum likelihood estimation program for vector ARMA models. It can be shown that maximizing the exact likelihood function is equivalent to minimizing

$$S(\Phi,\Theta,\mu,\mathbf{Q}\,|\,\mathbf{w})^m \; |\mathbf{Q}| \; |\Lambda^T\Lambda|^{\frac{1}{n}}$$

Table 1. Ratios between the number of multiplications and divisions required by the algorithm of Shea (1989) and those required by subroutine ELF2 to evaluate the exact likelihood for various vector ARMA models.

	$m = 2$		$m = 4$	
	$n = 100$	$n = 200$	$n = 100$	$n = 200$
AR(1)	1.19	1.10	3.88	2.65
AR(2)	1.27	1.14	3.41	2.79
MA(1)	2.46	2.49	2.81	2.83
MA(2)	1.88	1.91	2.09	2.11
ARMA(1,1)	2.52	2.54	3.39	3.21
$AR(1)_4$	1.18	1.13	2.16	2.07
$MA(1)_4$	1.47	1.53	1.60	1.67
$ARMA(1,1)_4$	1.56	1.66	2.04	2.03
$AR(1)_{12}$	0.84 *	0.85 *	1.27	1.27
$MA(1)_{12}$	0.81 *	1.00	0.84 *	1.06
$ARMA(1,1)_{12}$	0.91 *	1.05	1.25	1.29
$AR(1)\times MA(1)_4$	1.50	1.57	1.77	1.79
$AR(1)\times MA(1)_{12}$	0.81 *	1.00	0.86 *	1.08
$MA(1)\times AR(1)_4$	1.57	1.64	2.08	2.02
$MA(1)\times AR(1)_{12}$	0.90 *	0.98 *	1.27	1.27
$ARMA(1,1)\times AR(1)_4$	1.40	1.49	1.85	1.82
$ARMA(1,1)\times MA(1)_4$	1.37	1.46	1.59	1.65
$ARMA(1,1)\times AR(1)_{12}$	0.87 *	0.94 *	1.23	1.24
$ARMA(1,1)\times MA(1)_{12}$	0.75 *	0.96 *	0.80 *	1.02

NOTE: An asterisk (*) indicates that algorithm AS 242 requires less time-consuming operations than the new algorithm.

Since, on successful exit, subroutine ELF2 returns $S(\mathbf{\Phi}, \mathbf{\Theta}, \mu, \mathbf{Q}|\mathbf{w})$ (the quadratic form in the exact likelihood) and $|\mathbf{Q}|^n|\mathbf{\Lambda}^T\mathbf{\Lambda}|$ (the determinant in the exact likelihood) as *F1* and *F2*, respectively, it is straightforward to use that output in the computation of the objective function to be minimized. Further, since subroutine ELF2 automatically detects parameter values that imply non-stationarity, non-invertibility and/or non-positive definiteness of **Q**, an objective function that penalizes these situations can be constructed following the guidelines in Shea (1984) and Mauricio (1993). The residual vector should be evaluated (using subroutine CXI) only after the minimization routine has converged, since it is not used during the estimation process. Finally, note that the computations involved in this process can be speeded up using the approximation to the exact likelihood function (based on the convergence of the Ξ_k's to **0**) discussed previously.

10. References

Dennis, J. E. and Schnabel, R. B. (1983) *Numerical Methods for Unconstrainded Optimization and Nonlinear Equations*. New Jersey: Prentice-Hall.

Hall, A. D. and Nicholls, D. F. (1980) The Evaluation of Exact Maximum Likelihood Estimates for VARMA Models. *J. Statist. Comput. Simul.*, 10, 251-262.

Kohn, R. and Ansley, C. F. (1982) A Note on Obtaining the Theoretical Autocovariances of an ARMA Process, *J. Statist. Comput. Simul.*, 15, 273-283.

Ljung, G. M. and Box, G. E. P. (1979) The Likelihood Function of Stationary Autoregressive-Moving Average Models, *Biometrika*, 66, 265-270.

Martin, R. S. and Wilkinson, J. H. (1965) Symmetric Decomposition of Positive Definite Band Matrices, *Num. Math.*, 7, 355-361.

Mauricio, J. A. (1993) Exact Maximum Likelihood Estimation of Stationary Vector ARMA models, *I.C.A.E.*, D.T. Nº 9316.

Moler, C. B. (1972) Algorithm 423: Linear Equation Solver, *Commun. Ass. Comput. Mach.*, 15, 274.

Nicholls, D. F. and Hall, A. D. (1979) The Exact Likelihood Function of Multivariate Autoregressive-Moving Average Models, *Biometrika*, 66, 259-264.

Shea, B. L. (1984), Maximum Likelihood Estimation of Multivariate ARMA Processes via the Kalman Filter. In *Time Series Analysis: Theory and Practice* (ed. O. D. Anderson), vol. 5, pp. 91-101, Amsterdam: North-Holland.

Shea, B. L. (1989) Algorithm AS 242: The Exact Likelihood of a Vector Autoregressive-Moving Average Model, *Appl. Statist.*, 38, 161-204.

```
C
      SUBROUTINE ELF1( M, P, Q, N, W, PHI, THETA, QQ, ISMU, MU,
     *                 ATF, A, SIGMA2, XITOL, LOGELF, F1, F2,
     *                 WS, NWS, IWS, NIWS, IFAULT )
C
      INTEGER     M, P, Q, N, NWS, NIWS, IWS(NIWS), IFAULT,
     *            G, G2, M2, I1, I2, I3, I4, I5, I6, I7, I8,
     *            I9, I10, I11, BIG1, BIG2, BIG3, BIG4, TOTAL
      LOGICAL     ISMU, ATF
      REAL        W(M,N), PHI(M,P*M+1), THETA(M,Q*M+1), QQ(M,M), MU(M),
     *            A(M,N), SIGMA2, XITOL, LOGELF, F1, F2, WS(NWS), EPSIL

      INTRINSIC   MAX
      EXTERNAL    MACHEP, ELF2
C
C     [1]: Check the input data
C
      IFAULT = 0
      IF ( M .LT. 1 ) THEN
         IFAULT = 1
         RETURN
      ELSEIF ( N .LT. 1 ) THEN
         IFAULT = 2
         RETURN
      ELSEIF ( P .LT. 0 ) THEN
         IFAULT = 3
         RETURN
      ELSEIF ( Q .LT. 0 ) THEN
         IFAULT = 4
         RETURN
      ELSEIF ( P .EQ. 0 .AND. Q .EQ. 0 ) THEN
         IFAULT = 5
         RETURN
      ENDIF
C
C     [2]: Check that workspace is big enough
C
      G  = MAX( P, Q )
      G2 = G * G
      M2 = M * M
      IF ( P .GT. 0 ) THEN
         BIG1 = M * (M + 1) / 2 + M2 * (P - 1)
      ELSE
         BIG1 = 1
      ENDIF
      BIG2 = MAX( BIG1, G * M )
      BIG3 = MAX( BIG1, M )
      BIG4 = MAX( N, Q )
C
      I1  = 1
      I2  = I1  + M2
      I3  = I2  + M2
      I4  = I3  + M2
      I5  = I4  + BIG2
      I6  = I5  + BIG3
      I7  = I6  + BIG1 * BIG1
      I8  = I7  + M2 * G2
      I9  = I8  + M2 * G2
      I10 = I9  + M2 * G2
      I11 = I10 + M2 * (P + Q) * G
C
      TOTAL  = I11 + M2 * BIG4 - 1
C
      IF ( NWS .LT. TOTAL ) THEN
         IFAULT = 6
         RETURN
      ELSEIF ( NIWS .LT. BIG1 ) THEN
         IFAULT = 7
         RETURN
      ENDIF
C
C     [3]: Calculate machine epsilon (store as EPSIL)
C
      CALL MACHEP( EPSIL )
C
C     [4]: Call ELF2 to evaluate the exact log likelihood
C
      CALL ELF2( M, P, Q, N, G, W, PHI, THETA, QQ, ISMU, MU, ATF,
     *           A, SIGMA2, XITOL, LOGELF, F1, F2, EPSIL,
     *           BIG1, BIG2, BIG3, BIG4, WS(I1), WS(I2), WS(I3),
     *           WS(I4), WS(I5), WS(I6), WS(I7), WS(I8), WS(I9),
     *           WS(I10), WS(I11), IWS, IFAULT )
C
      END
```

```
C
C
C
      SUBROUTINE ELF2( M, P, Q, N, G, W, PHI, THETA, QQ, ISMU, MU, ATF,
     *                 A, SIGMA2, XITOL, LOGELF, F1, F2, EPS,
     *                 BIG1, BIG2, BIG3, BIG4, Q1, Q1INV, MTMP4, VTMP1,
     *                 VTMP2, MATPHI, MTMP0, MTMP2, MTMP3, MTMP1, GAMXI,
     *                 INDX, IFAULT )
C
      INTEGER     M, P, Q, G, N, BIG1, BIG2, BIG3, BIG4, INDX(BIG1),
     *            IFAULT, I, I1, I2, II, J, J1, J2, JJ,
     *            JL, K, KK, NLIM
      LOGICAL     ISMU, ATF
      REAL        W(M,N), PHI(M,P*M+1), THETA(M,Q*M+1), QQ(M,M), MU(M),
     *            A(M,N), SIGMA2, XITOL, LOGELF, F1, F2, EPS,
     *            Q1(M,M), Q1INV(M,M), MTMP4(M,M), VTMP1(BIG2),
     *            VTMP2(BIG3), MATPHI(BIG1,BIG1), MTMP0(G*M,G*M),
     *            MTMP2(G*M,G*M), MTMP3(G*M,G*M), MTMP1(M*(P+Q),M*G),
     *            GAMXI(M,M*BIG4), D1, D2, S1, S2, DETQ, DETOM,
     *            ZERO, ONE, TWO, LG2PI, REAL
      PARAMETER   ( ZERO = 0.0, ONE = 1.0, TWO = 2.0, LG2PI = 1.8378771 )
      INTRINSIC   EXP, LOG, MAX, REAL
      EXTERNAL    CHOLDC, CHOLFR, CGAMMA, CXI, CRES
C
C     Copy lower triangle of QQ into upper triangles of QQ and Q1
C
      DO 2 I = 1, M
      DO 1 J = I, M
      QQ(I,J) = QQ(J,I)
      Q1(I,J) = QQ(J,I)
    1 CONTINUE
    2 CONTINUE
C
C     [1]: Calculate the inverse of the Cholesky factor of QQ
C          (store as Q1INV) and the determinant of QQ (store as DETQ)
C
      CALL CHOLDC( Q1, M, D1, D2, EPS, IFAULT )
C
      IF ( IFAULT .GT. 0 ) THEN
         IFAULT = 8
         RETURN
      ENDIF
C
      DO 5 I = 1, M
      DO 3 J = 1, M
      VTMP1(J) = ZERO
      Q1INV(J,I) = ZERO
    3 CONTINUE
      VTMP1(I) = ONE
      CALL CHOLFR( Q1, M, VTMP1 )
      DO 4 J = I, M
      Q1INV(J,I) = VTMP1(J)
    4 CONTINUE
    5 CONTINUE
C
      DETQ = D1 * TWO ** D2
C
C     [2]: Calculate the theoretical autocovariance and
C          cross-covariance matrices (store as GAMXI and VTMP1)
C
      IF ( P .GT. 0 ) THEN
C
         CALL CGAMMA( M, P, Q, PHI, THETA, QQ, BIG1, MATPHI, VTMP2,
     *                MTMP2, MTMP3, GAMXI, VTMP1, INDX, IFAULT )
         IF ( IFAULT .GT. 0 ) THEN
            IFAULT = 9
            RETURN
         ENDIF
C
      ENDIF
C
C     [3]: Calculate M: Cholesky factor of V1*OMEGA*V1' (store as MTMP0)
C
      DO 20 I = 1, M * G
      DO 10 J = 1, M * G
      MTMP0(I,J) = ZERO
   10 CONTINUE
   20 CONTINUE
C
C     [3.1]: Calculate OMEGA*V1' (store as MTMP1)
C
      DO 40 I = 1, M * (P + Q)
      DO 30 J = 1, M * G
      MTMP1(I,J) = ZERO
   30 CONTINUE
   40 CONTINUE
C
      DO 140 I = 1, P
      DO 130 J = 1, G
C
      DO 80 K = J - I, P - I
      DO 70 II = 1, M
      DO 60 JJ = 1, M
      S1 = ZERO
      DO 50 KK = 1, M
C
      IF ( K .GT. 0 ) THEN
         JL = M * (M + 1) / 2 + M * M * (K - 1) + M * (KK - 1) + II
      ELSEIF ( K .LT. 0 ) THEN
         JL = M * (M + 1) / 2 + M * M * (- K - 1) + M * (II - 1) + KK
      ELSE
         IF ( KK .GE. II ) THEN
            JL = KK * (KK - 1) / 2 + II
         ELSE
            JL = II * (II - 1) / 2 + KK
         ENDIF
      ENDIF
C
      S1 = S1 + VTMP1(JL) * PHI(JJ, (P - K - I + J - 1) * M + KK)
   50 CONTINUE
      MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) =
     *      MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) + S1
   60 CONTINUE
   70 CONTINUE
   80 CONTINUE
C
      DO 120 K = J - I, Q - I
      IF ( P + K .LE. Q ) THEN
         DO 110 II = 1, M
         DO 100 JJ = 1, M
         S1 = ZERO
         DO 90 KK = 1, M
         S1 = S1 + GAMXI(II, (Q - P - K) * M + KK)
     *           * THETA(JJ, (Q - K - I + J - 1) * M + KK)
   90    CONTINUE
         MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) =
     *        MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) - S1
  100    CONTINUE
  110    CONTINUE
      ENDIF
  120 CONTINUE
C
  130 CONTINUE
  140 CONTINUE
C
      DO 230 I = P + 1, P + Q
      DO 220 J = 1, G
C
      DO 180 K = P + J - I, P + P - I
      IF ( P - K .LE. Q ) THEN
         DO 170 II = 1, M
         DO 160 JJ = 1, M
         S1 = ZERO
         DO 150 KK = 1, M
         S1 = S1 + GAMXI(KK, (Q - P + K) * M + II)
     *           * PHI(JJ, (P + P - K - I + J - 1) * M + KK)
  150    CONTINUE
         MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) =
     *        MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) + S1
  160    CONTINUE
  170    CONTINUE
      ENDIF
  180 CONTINUE
C
      IF ( P - I + J .LE. 0 ) THEN
         DO 210 II = 1, M
         DO 200 JJ = 1, M
         S1 = ZERO
         DO 190 KK = 1, M
         S1 = S1 + QQ(II,KK)
     *           * THETA(JJ, (Q + P - I + J - 1) * M + KK)
  190    CONTINUE
         MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) =
     *        MTMP1(II + (I - 1) * M, JJ + (J - 1) * M) - S1
  200    CONTINUE
  210    CONTINUE
      ENDIF
C
```

```
  220 CONTINUE
  230 CONTINUE
C
C     [3.2]: Calculate V1*OMEGA*V1' (store as MTMP0)
C
      DO 330 I = 1, G
      DO 320 J = I, G
C
      DO 270 K = 0, P - I
      DO 260 II = 1, M
      IF ( I .EQ. J ) THEN
         JL = II
      ELSE
         JL = 1
      ENDIF
      DO 250 JJ = JL, M
      S1 = ZERO
      DO 240 KK = 1, M
      S1 = S1 + PHI(II, (P - K - 1) * M + KK)
     *        * MTMP1(KK + (K + I - 1) * M, JJ + (J - 1) * M)
  240 CONTINUE
      MTMP0(II + (I - 1) * M, JJ + (J - 1) * M) =
     *     MTMP0(II + (I - 1) * M, JJ + (J - 1) * M) + S1
  250 CONTINUE
  260 CONTINUE
  270 CONTINUE
C
      DO 310 K = 0, Q - I
      DO 300 II = 1, M
      IF ( I .EQ. J ) THEN
         JL = II
      ELSE
         JL = 1
      ENDIF
      DO 290 JJ = JL, M
      S1 = ZERO
      DO 280 KK = 1, M
      S1 = S1 + THETA(II, (Q - K - 1) * M + KK)
     *        * MTMP1(KK + (K + P + I - 1) * M, JJ + (J - 1) * M)
  280 CONTINUE
      MTMP0(II + (I - 1) * M, JJ + (J - 1) * M) =
     *     MTMP0(II + (I - 1) * M, JJ + (J - 1) * M) - S1
  290 CONTINUE
  300 CONTINUE
  310 CONTINUE
C
  320 CONTINUE
  330 CONTINUE
C
C     [3.3]: Calculate M (overwrite MTMP0)
C
      CALL CHOLDC( MTMP0, M * G, D1, D2, EPS, IFAULT )
C
      IF ( IFAULT .GT. 0 ) THEN
         IFAULT = 10
         RETURN
      ENDIF
C
C     [4]: Calculate matrix polynomial R*XI(k) (overwrite GAMXI)
C
      CALL CXI( M, N, Q, THETA, XITOL, Q1INV, NLIM, GAMXI,
     *          MTMP4, IFAULT )
C
      IF ( IFAULT .GT. 0 ) THEN
         IFAULT = 11
         RETURN
      ENDIF
C
C     [5]: Calculate vector eta (store as A)
C
      DO 340 I = 1, M
      DO 335 J = 1, N
      A(I,J) = ZERO
  335 CONTINUE
  340 CONTINUE
C
C     [5.1]: Calculate conditional residuals recursively (store as A)
C
      DO 440 I = 1, N
C
      DO 350 J = 1, M
      VTMP1(J) = ZERO
  350 CONTINUE
      DO 380 J = 1, P
      IF ( I - J .GE. 1 ) THEN
         DO 370 II = 1, M
         S1 = ZERO
         DO 360 K = 1, M
         IF ( ISMU ) THEN
            S2 = W(K, I - J) - MU(K)
         ELSE
            S2 = W(K, I - J)
         ENDIF
         S1 = S1 + PHI(II, (J - 1) * M + K) * S2
  360    CONTINUE
         VTMP1(II) = VTMP1(II) + S1
  370    CONTINUE
      ENDIF
  380 CONTINUE
C
      DO 390 J = 1, M
      VTMP2(J) = ZERO
  390 CONTINUE
      DO 420 J = 1, Q
      IF ( I - J .GE. 1 ) THEN
         DO 410 II = 1, M
         S1 = ZERO
         DO 400 K = 1, M
         S1 = S1 + THETA(II, (J - 1) * M + K) * A(K, I - J)
  400    CONTINUE
         VTMP2(II) = VTMP2(II) + S1
  410    CONTINUE
      ENDIF
  420 CONTINUE
C
      DO 430 II = 1, M
      IF ( ISMU ) THEN
         S2 = W(II,I) - MU(II)
      ELSE
         S2 = W(II,I)
      ENDIF
      A(II,I) = S2 - VTMP1(II) + VTMP2(II)
  430 CONTINUE
C
  440 CONTINUE
C
C     [5.2]: Premultiply each M-block of A by Q1INV (overwrite A)
C
      DO 480 I = 1, N
      DO 460 J = 1, M
      S1 = ZERO
      DO 450 K = 1, J
      S1 = S1 + Q1INV(J,K) * A(K,I)
  450 CONTINUE
      VTMP1(J) = S1
  460 CONTINUE
      DO 470 J = 1, M
      A(J,I) = VTMP1(J)
  470 CONTINUE
  480 CONTINUE
C
C     [6]: Calculate vector M'h (store as VTMP1)
C
      DO 490 I = 1, G * M
      VTMP1(I) = ZERO
  490 CONTINUE
C
C     [6.1]: Calculate vector h (overwrite VTMP1)
C
      DO 530 J = 1, G
      DO 520 I = 0, N - J
      IF ( I .LE. NLIM ) THEN
         DO 510 JJ = 1, M
         S1 = ZERO
         DO 500 K = 1, M
         S1 = S1 + GAMXI(K, I * M + JJ) * A(K,I+J)
  500    CONTINUE
         VTMP1(JJ + (J - 1) * M) = VTMP1( JJ + (J - 1) * M) + S1
  510    CONTINUE
      ENDIF
  520 CONTINUE
  530 CONTINUE
C
C     [6.2]: Premultiply VTMP1 by MTMP0' (overwrite VTMP1)
C
      DO 550 I = 1, M * G
      S1 = ZERO
      DO 540 K = I, M * G
```

```
      S1 = S1 + MTMP0(K,I) * VTMP1(K)
  540 CONTINUE
      VTMP1(I) = S1
  550 CONTINUE
C
C     Store M as MTMP3 if residuals have been requested
C
      IF ( ATF ) THEN
         DO 570 I = 1, M * G
         DO 560 J = 1, I
         MTMP3(I,J) = MTMP0(I,J)
  560    CONTINUE
  570    CONTINUE
      ENDIF
C
C     [7]: Calculate H'H (store as MTMP2)
C
      DO 590 I = 1, G * M
      DO 580 J = 1, G * M
      MTMP2(I,J) = ZERO
  580 CONTINUE
  590 CONTINUE
C
      DO 640 I = 1, G
      DO 630 K = 0, N - I
      IF ( K + I - 1 .LE. NLIM ) THEN
C
         DO 620 II = 1, M
         IF ( I .EQ. 1 ) THEN
            JL = II
         ELSE
            JL = M
         ENDIF
         DO 610 JJ = 1, JL
         S1 = ZERO
         DO 600 KK = 1, M
         S1 = S1 + GAMXI(KK, K * M + II)
     *             * GAMXI(KK, (K + I - 1) * M + JJ)
  600    CONTINUE
         MTMP2(II + (I - 1) * M, JJ) = MTMP2(II + (I - 1) * M, JJ) + S1
  610    CONTINUE
  620    CONTINUE
C
      ENDIF
  630 CONTINUE
  640 CONTINUE
C
      DO 690 I = 2, G
      DO 680 J = 2, I
      DO 670 II = 1, M
      IF ( I .EQ. J ) THEN
         JL = II
      ELSE
         JL = M
      ENDIF
      DO 660 JJ = 1, JL
      S1 = ZERO
      IF ( (N - I + 1 .LE. NLIM) .AND. (N - J + 1 .LE. NLIM) ) THEN
C
         DO 650 KK = 1, M
         S1 = S1 + GAMXI(KK, (N - I + 1) * M + II)
     *             * GAMXI(KK, (N - J + 1) * M + JJ)
  650    CONTINUE
C
      ENDIF
      MTMP2(II + (I - 1) * M, JJ + (J - 1) * M) =
     *      MTMP2(II + (I - 2) * M, JJ + (J - 2) * M) - S1
  660 CONTINUE
  670 CONTINUE
  680 CONTINUE
  690 CONTINUE
C
      DO 710 I = 1, G * M
      DO 700 J = I + 1, G * M
      MTMP2(I,J) = MTMP2(J,I)
  700 CONTINUE
  710 CONTINUE
C
C     [8]: Calculate I+M'H'HM and its Cholesky factor (overwrite MTMP0)
C
C     [8.1]: Calculate M'H'H (store as MTMP1)
C
      DO 740 I = 1, G * M
      DO 730 J = 1, G * M
      S1 = ZERO
      DO 720 K = I, G * M
      S1 = S1 + MTMP0(K,I) * MTMP2(K,J)
  720 CONTINUE
      MTMP1(I,J) = S1
  730 CONTINUE
  740 CONTINUE
C
C     [8.2]: Store M as MTMP2 (overwrite) and initialize MTMP0
C
      DO 760 I = 1, G * M
      DO 750 J = 1, G * M
      MTMP2(I,J) = MTMP0(I,J)
      MTMP0(I,J) = ZERO
  750 CONTINUE
  760 CONTINUE
C
C     [8.3]: Calculate I + M'H'HM (store as MTMP0)
C
      DO 790 I = 1, G * M
      DO 780 J = I, G * M
      S1 = ZERO
      DO 770 K = J, G * M
      S1 = S1 + MTMP1(I,K) * MTMP2(K,J)
  770 CONTINUE
      MTMP0(I,J) = S1
  780 CONTINUE
      MTMP0(I,I) = ONE + MTMP0(I,I)
  790 CONTINUE
C
C     [8.4]: Compute the Cholesky factor of I+M'H'HM
C            (overwrite MTMP0) and its determinant (store as DETOM)
C
      CALL CHOLDC( MTMP0, G * M, D1, D2, EPS, IFAULT )
C
      IF ( IFAULT .GT. 0 ) THEN
         IFAULT = 12
         RETURN
      ENDIF
C
      DETOM = D1 * TWO ** D2
C
C     [9]: Calculate LAMBDA using forward substitution (store as VTMP1)
C
      CALL CHOLFR( MTMP0, M * G, VTMP1 )
C
C     [10]: Calculate the sum of squares (return as F1)
C
      S1 = ZERO
      DO 810 I = 1, M
      DO 800 J = 1, N
      S1 = S1 + A(I,J) * A(I,J)
  800 CONTINUE
  810 CONTINUE
C
      S2 = ZERO
      DO 820 I = 1, M * G
      S2 = S2 + VTMP1(I) * VTMP1(I)
  820 CONTINUE
C
      F1 = S1 - S2
C
C     Calculate the determinant (return as F2)
C
      F2 = DETOM * (DETQ ** REAL( N ))
C
C     Calculate the exact log likelihood (return as LOGELF)
C
      LOGELF = -( REAL( N * M ) * (LG2PI + LOG( SIGMA2 )) +
     *            REAL( N ) * LOG( DETQ ) + LOG ( DETOM ) +
     *            F1 / SIGMA2 ) / TWO
C
C     [11]: Calculate residual vector if requested (return as A)
C
      IF ( ATF )
     *   CALL CRES( M, N, G, NLIM, GAMXI, Q1, MTMP3, MTMP0, VTMP1, A )
C
      END
C
C
C
      SUBROUTINE CGAMMA( M, P, Q, PHI, THETA, QQ, BIG1, MAT, VV,
     *                   WZERO, MZERO, GAMWA, RHS, INDX, IFAULT )
C
```

```
      INTEGER    M, P, Q, BIG1, INDX(BIG1), IFAULT, ROW, COL,
     *           H, I, II, J, JJ, K, L, R, S
      REAL       PHI(M,P*M+1), THETA(M,Q*M+1), QQ(M,M),
     *           MAT(BIG1,BIG1), VV(BIG1), WZERO(M,M), MZERO(M,M),
     *           GAMWA(M,Q*M+1), RHS(BIG1), SUM, ZERO, ONE
      PARAMETER  ( ZERO = 0.0, ONE = 1.0 )
      EXTERNAL   DECOMP, SOLVE
C
      IFAULT = 0
C
C     [1]: Compute the Q - 1 cross-covariance matrices (return as GAMWA)
C
      DO 20 I = 1, M
      DO 10 J = 1, M
      GAMWA(I,J) = QQ(I,J)
   10 CONTINUE
   20 CONTINUE
C
      DO 80 K = 1, Q - 1
      DO 70 I = 1, M
      DO 60 J = 1, M
      SUM = ZERO
      DO 30 H = 1, M
      SUM = SUM - THETA(I, (K - 1) * M + H) * QQ(H,J)
   30 CONTINUE
      DO 50 L = 1, K
      IF ( L .LE. P ) THEN
         DO 40 H = 1, M
         SUM = SUM + PHI(I, (L - 1) * M + H)
     *             * GAMWA(H, (K - L) * M + J)
   40    CONTINUE
      ENDIF
   50 CONTINUE
      GAMWA(I, K * M + J) = SUM
   60 CONTINUE
   70 CONTINUE
   80 CONTINUE
C
C     [2]: Compute diagonal and upper triangle of W(0) (store as WZERO)
C
      DO 100 I = 1, M
      DO  90 J = 1, M
      WZERO(I,J) = ZERO
   90 CONTINUE
  100 CONTINUE
C
      DO 180 I = 1, P
      DO 170 J = I, Q
      IF ( J - I .GE. 0 ) THEN
C
         DO 130 II = 1, M
         DO 120 JJ = 1, M
         SUM = ZERO
         DO 110 K = 1, M
         SUM = SUM + PHI(II, (I - 1) * M + K)
     *             * GAMWA(K, (J - I) * M + JJ)
  110    CONTINUE
         MZERO(II,JJ) = SUM
  120    CONTINUE
  130    CONTINUE
C
         DO 160 II = 1, M
         DO 150 JJ = 1, M
         SUM = ZERO
         DO 140 K = 1, M
         SUM = SUM + MZERO(II,K) * THETA(JJ, (J - 1) * M + K)
  140    CONTINUE
         WZERO(II,JJ) = WZERO(II,JJ) + SUM
  150    CONTINUE
  160    CONTINUE
C
      ENDIF
  170 CONTINUE
  180 CONTINUE
C
      DO 200 I = 1, M
      DO 190 J = I, M
      WZERO(I,J) = QQ(I,J) - WZERO(I,J) - WZERO(J,I)
  190 CONTINUE
  200 CONTINUE
C
      DO 270 J = 1, Q
C
      DO 230 II = 1, M
      DO 220 JJ = 1, M
      SUM = ZERO
      DO 210 K = 1, M
      SUM = SUM + THETA(II, (J - 1) * M + K) * QQ(K,JJ)
  210 CONTINUE
      MZERO(II,JJ) = SUM
  220 CONTINUE
  230 CONTINUE
C
      DO 260 II =  1, M
      DO 250 JJ = II, M
      SUM = ZERO
      DO 240 K = 1, M
      SUM = SUM + MZERO(II,K) * THETA(JJ, (J - 1) * M + K)
  240 CONTINUE
      WZERO(II,JJ) = WZERO(II,JJ) + SUM
  250 CONTINUE
  260 CONTINUE
C
  270 CONTINUE
C
C     [3]: Set up system of equations (store as MAT and RHS)
C
      DO 290 I = 1, BIG1
      DO 280 J = 1, BIG1
      MAT(I,J) = ZERO
  280 CONTINUE
      RHS(I) = ZERO
  290 CONTINUE
C
C     [3.1]: Compute the first M * (M + 1) / 2 rows
C
      DO 390 J = 1, M
      DO 380 I = 1, J
      ROW = J * (J - 1) / 2 + I
C
C     [3.1.1]: Compute the first M * (M + 1) / 2 columns
C
      DO 330 L = 1, M
      DO 320 K = 1, L
      COL = L * (L - 1) / 2 + K
      SUM = ZERO
      IF ( K .EQ. L ) THEN
         DO 300 R = 1, P
         SUM = SUM - PHI(I, (R - 1) * M + K) * PHI(J, (R - 1) * M + L)
  300    CONTINUE
      ELSE
         DO 310 R = 1, P
         SUM = SUM - PHI(I, (R - 1) * M + K) * PHI(J, (R - 1) * M + L)
     *             - PHI(I, (R - 1) * M + L) * PHI(J, (R - 1) * M + K)
  310    CONTINUE
      ENDIF
      MAT(ROW,COL) = SUM
  320 CONTINUE
  330 CONTINUE
C
C     [3.1.2]: Compute the remaining M * M * (P - 1) columns
C
      DO 370 S = 1, P - 1
      DO 360 L = 1, M
      DO 350 K = 1, M
      COL = M * (M + 1) / 2 + M * M * (S - 1) + M * (L - 1) + K
      SUM = ZERO
      DO 340 R = 1, P - S
      SUM = SUM - PHI(I, (R + S - 1) * M + K) * PHI(J, (R - 1) * M + L)
     *          - PHI(J, (R + S - 1) * M + K) * PHI(I, (R - 1) * M + L)
  340 CONTINUE
      MAT(ROW,COL) = SUM
  350 CONTINUE
  360 CONTINUE
  370 CONTINUE
C
C     [3.1.3]: Set up RHS and diagonal of MAT
C
      RHS(ROW) = WZERO(I,J)
      MAT(ROW,ROW) = ONE + MAT(ROW,ROW)
C
  380 CONTINUE
  390 CONTINUE
C
C     [3.2]: Compute the remaining M * M * (P - 1) rows
C
      DO 470 S = 1, P - 1
C
```

```
      DO 460 I = 1, M
      DO 450 J = 1, M
      ROW = M * (M + 1) / 2 + M * M * (S - 1) + M * (I - 1) + J
C
C     [3.2.1]: Compute the first M * (M + 1) / 2 columns
C
      DO 400 L = 1, M
      IF ( L .LE. J ) THEN
         COL = J * (J - 1) / 2 + L
      ELSE
         COL = L * (L - 1) / 2 + J
      ENDIF
      MAT(ROW,COL) = - PHI(I, (S - 1) * M + L)
  400 CONTINUE
C
C     [3.2.2]: Compute the remaining M * M * (P - 1) columns
C
      DO 420 R = 1, P - 1
      DO 410 L = 1, M
      COL = M * (M + 1) / 2 + (R - 1) * M * M + (J - 1) * M + L
      IF ( R + S .LE. P ) MAT(ROW,COL) = - PHI(I, (R + S - 1) * M + L)
      IF ( S .GT. R ) THEN
         COL = M * (M + 1) / 2 + (R - 1) * M * M + (L - 1) * M + J
         MAT(ROW,COL) = MAT(ROW,COL) - PHI(I, (S - R - 1) * M + L)
      ENDIF
  410 CONTINUE
  420 CONTINUE
C
C     [3.2.3]: Set up RHS and diagonal of MAT
C
      RHS(ROW) = ZERO
      DO 440 II = S, Q
      DO 430  K = 1, M
      RHS(ROW) = RHS(ROW) - GAMWA(J, (II - S) * M + K)
     *                    * THETA(I, (II - 1) * M + K)
  430 CONTINUE
  440 CONTINUE
C
      MAT(ROW,ROW) = ONE + MAT(ROW,ROW)
C
  450 CONTINUE
  460 CONTINUE
  470 CONTINUE
C
C     [4]: Solve for autocovariance matrices (return as RHS)
C
      CALL DECOMP( BIG1, BIG1, MAT, INDX )
      IF ( INDX(BIG1) .EQ. 0 ) THEN
         IFAULT = 1
         RETURN
      ELSE
         CALL SOLVE( BIG1, BIG1, MAT, RHS, INDX )
      ENDIF
C
      END
C
C
C
      SUBROUTINE CXI( M, N, Q, THETA, XITOL, R, NLIM, XI, MTMP, IFAULT )
C
      INTEGER    M, N, Q, NLIM, IFAULT, H, I, II, J, JJ, K, NQ
      REAL       THETA(M,M*Q+1), XITOL, R(M,M), XI(M,M*N), MTMP(M,M),
     *           S1, S2, MX
      LOGICAL    DELTA
      PARAMETER  ( ZERO = 0.0, ONE = 1.0 )
      INTRINSIC  ABS
C
      IFAULT = 0
      DELTA = .FALSE.
      MX = ZERO
C
      DO 20 I = 1, M
      DO 10 J = 1, M * N
      XI(I,J) = ZERO
   10 CONTINUE
      XI(I,I) = ONE
   20 CONTINUE
C
C     [1]: Update index NLIM and calculate matrix sequence (store as XI)
C
      NLIM = 0
C
   30 IF ( (.NOT. DELTA) .AND. (NLIM .LT. N - 1) ) THEN
C
```

```
      NLIM = NLIM + 1
C
C     [1.1]: Calculate the XI matrix for this round
C
      DO 70 J = 1, Q
      IF ( NLIM .GE. J ) THEN
         DO 60 II = 1, M
         DO 50 JJ = 1, M
         S1 = ZERO
         DO 40 H = 1, M
         S1 = S1 + THETA(II, (J - 1) * M + H)
     *           * XI(H, (NLIM - J) * M + JJ)
   40    CONTINUE
         XI(II, NLIM * M + JJ) = XI(II, NLIM * M + JJ) + S1
   50    CONTINUE
   60    CONTINUE
      ENDIF
   70 CONTINUE
C
      S2 = ZERO
      DO 90 II = 1, M
      DO 80 JJ = 1, M
      S2 = S2 + ABS( XI(II, NLIM * M + JJ) )
   80 CONTINUE
   90 CONTINUE
C
      IF ( NLIM .LE. Q ) MX = MX + S2
      IF ( S2 .GT. MX ) THEN
         IFAULT = 1
         RETURN
      ENDIF
C
C     [1.2]: Check for effective convergence
C
      IF ( S2 .LT. XITOL ) THEN
         NQ = 1
         DELTA = .TRUE.
C
  100    IF ((NQ .LE. Q) .AND. (NLIM .LT. N - 1) .AND. (DELTA)) THEN
C
            NQ = NQ + 1
            NLIM = NLIM + 1
            DO 140 J = 1, Q
            IF ( NLIM .GE. J ) THEN
               DO 130 II = 1, M
               DO 120 JJ = 1, M
               S1 = ZERO
               DO 110 H = 1, M
               S1 = S1 + THETA(II, (J - 1) * M + H)
     *                 * XI(H, (NLIM - J) * M + JJ)
  110          CONTINUE
               XI(II, NLIM * M + JJ) = XI(II, NLIM * M + JJ) + S1
  120          CONTINUE
  130          CONTINUE
            ENDIF
  140       CONTINUE
C
            S2 = ZERO
            DO 160 II = 1, M
            DO 150 JJ = 1, M
            S2 = S2 + ABS( XI(II, NLIM * M + JJ) )
  150       CONTINUE
  160       CONTINUE
C
            IF ( NLIM .LE. Q ) MX = MX + S2
            IF ( S2 .GT. MX ) THEN
               IFAULT = 1
               RETURN
            ENDIF
            IF ( S2 .GT. XITOL ) DELTA = .FALSE.
            GOTO 100
         ENDIF
C
         IF ( DELTA ) NLIM = NLIM - NQ
      ENDIF
C
      GOTO 30
    ENDIF
C
C   [2]: Premultiply every XI by R (overwrite XI)
C
    DO 220 K = 0, NLIM
C
    DO 190 I = 1, M
```

```
      DO 180 J = 1, M
      S1 = ZERO
      DO 170 H = 1, I
      S1 = S1 + R(I,H) * XI(H, K * M + J)
  170 CONTINUE
      MTMP(I,J) = S1
  180 CONTINUE
  190 CONTINUE
C
      DO 210 I = 1, M
      DO 200 J = 1, M
      XI(I, K * M + J) = MTMP(I,J)
  200 CONTINUE
  210 CONTINUE
C
  220 CONTINUE
C
      END
C
C
C
      SUBROUTINE CRES( M, N, G, NLIM, XI, Q1, MATM, MATL, LAMBDA, RES )
C
      INTEGER    M, N, G, NLIM, H, I, J, JJ
      REAL       XI(M,M*N), Q1(M,M), MATM(G*M,G*M), MATL(G*M,G*M),
     *           LAMBDA(G*M), RES(M,N), SUM, ZERO
      PARAMETER  ( ZERO = 0.0 )
      EXTERNAL   CHOLBK
C
C     [1]: Solve for C in the system L'C = LAMBDA (overwrite LAMBDA)
C
      CALL CHOLBK( MATL, G * M, LAMBDA )
C
C     [2]: Calculate D = MC (overwrite LAMBDA)
C
      DO 20 I = M * G, 1, -1
      SUM = ZERO
      DO 10 J = 1, I
      SUM = SUM + MATM(I,J) * LAMBDA(J)
   10 CONTINUE
      LAMBDA(I) = SUM
   20 CONTINUE
C
C     [3]: Calculate residuals (return as RES)
C
      DO 60 I = 1, N
      DO 50 J = 1, I
C
      IF ( (I - J .LE. NLIM) .AND. (J .LE. G) ) THEN
         DO 40 JJ = 1, M
         SUM = ZERO
         DO 30 H = 1, M
         SUM = SUM + XI(JJ, (I - J) * M + H) * LAMBDA(H + (J - 1) * M)
   30    CONTINUE
         RES(JJ,I) = RES(JJ,I) - SUM
   40    CONTINUE
      ENDIF
C
   50 CONTINUE
   60 CONTINUE
C
      DO 90 J = 1, N
      DO 80 I = M, 1, -1
      SUM = ZERO
      DO 70 H = 1, I
      SUM = SUM + Q1(I,H) * RES(H,J)
   70 CONTINUE
      RES(I,J) = SUM
   80 CONTINUE
   90 CONTINUE
C
      END
C
C
C
      SUBROUTINE CHOLDC( M, N, D1, D2, EPS, IFAULT )
C
      INTEGER    N, IFAULT, I, J, K
      REAL       M(N,N), D1, D2, EPS, SUM, ML1, ML2, MLJ, MXO, MXA,
     *           ONE, ZERO, FOUR, P0625, SIXTEN
      PARAMETER  ( ZERO = 0.0, ONE = 1.0, FOUR = 4.0,
     *             P0625 = 0.0625, SIXTEN = 16.0 )
      INTRINSIC  ABS, SQRT
C
```

```
      IFAULT = 0
      D1 = ONE
      D2 = ZERO
C
C     [1]: Initialize finite arithmetic parameters
C
      ML1 = ZERO
      MXO = SQRT( ABS( M(1,1) ) )
      DO 10 J = 2, N
      SUM = SQRT( ABS( M(J,J) ) )
      IF ( SUM .GT. MXO ) MXO = SUM
   10 CONTINUE
C
      IF ( (MXO * MXO) .LE. SQRT( EPS ) ) THEN
         DO 30 I = 1, N
         DO 20 J = 1, I
         M(I,J) = ZERO
   20    CONTINUE
   30    CONTINUE
         RETURN
      ENDIF
C
      ML2 = SQRT( EPS ) * MXO
      MXA = ZERO
C
C     [2]: Calculate modified Cholesky decomposition
C
      DO 100 J = 1, N
      SUM = M(J,J)
      DO 40 I = 1, J - 1
      SUM = SUM - M(J,I) * M(J,I)
   40 CONTINUE
C
      IF ( ( SUM .NE. ABS( SUM ) ) .AND. ( ABS( SUM ) .GT. ML2 ) ) THEN
         IFAULT = 1
         RETURN
      ELSE
         M(J,J) = SUM
      ENDIF
C
      MLJ = ZERO
C
      DO 60 I = J + 1, N
      SUM = M(J,I)
      DO 50 K = 1, J - 1
      SUM = SUM - M(I,K) * M(J,K)
   50 CONTINUE
      M(I,J) = SUM
      IF ( ABS( M(I,J) ) .GT. MLJ ) MLJ = ABS( M(I,J) )
   60 CONTINUE
C
      IF ( (MLJ / MXO) .GT. ML1 ) THEN
         MLJ = MLJ / MXO
      ELSE
         MLJ = ML1
      ENDIF
C
      IF ( M(J,J) .GT. (MLJ * MLJ) ) THEN
         M(J,J) = SQRT( M(J,J) )
      ELSE
         IF ( MLJ .LT. ML2 ) MLJ = ML2
         IF ( MXA .LT. (MLJ * MLJ - M(J,J)) ) MXA = MLJ * MLJ - M(J,J)
         M(J,J) = MLJ
      ENDIF
C
      D1 = D1 * M(J,J) * M(J,J)
   70 IF ( D1 .GE. ONE ) THEN
         D1 = D1 * P0625
         D2 = D2 + FOUR
         GOTO 70
      ENDIF
   80 IF ( D1 .LT. P0625 ) THEN
         D1 = D1 * SIXTEN
         D2 = D2 - FOUR
         GOTO 80
      ENDIF
C
      DO 90 I = J + 1, N
      M(I,J) = M(I,J) / M(J,J)
   90 CONTINUE
  100 CONTINUE
C
      DO 120 J = 2, N
      DO 110 I = 1, J - 1
```

```
      M(I,J) = ZERO
  110 CONTINUE
  120 CONTINUE
C
      END
C
C
C
      SUBROUTINE CHOLFR( MATL, N, RHSOL )
C
      INTEGER     N, I, J
      REAL        MATL(N,N), RHSOL(N), SUM, ZERO
      PARAMETER   ( ZERO = 0.0 )
C
      RHSOL(1) = RHSOL(1) / MATL(1,1)
C
      DO 20 I = 2, N
      SUM = ZERO
      DO 10 J = 1, I - 1
      SUM = SUM + MATL(I,J) * RHSOL(J)
   10 CONTINUE
      RHSOL(I) = (RHSOL(I) - SUM) / MATL(I,I)
   20 CONTINUE
C
      END
C
C
C
      SUBROUTINE CHOLBK( MATL, N, RHSOL )
C
      INTEGER     N, I, J
      REAL        MATL(N,N), RHSOL(N), SUM, ZERO
      PARAMETER   ( ZERO = 0.0 )
C
      RHSOL(N) = RHSOL(N) / MATL(N,N)
C
      DO 20 I = N - 1, 1, -1
      SUM = ZERO
      DO 10 J = I + 1, N
      SUM = SUM + MATL(J,I) * RHSOL(J)
   10 CONTINUE
      RHSOL(I) = (RHSOL(I) - SUM) / MATL(I,I)
   20 CONTINUE
C
      END
C
C
C
      SUBROUTINE MACHEP( EPSIL )
C
      REAL        EPSIL, ONE, TWO
      PARAMETER   ( ONE = 1.0, TWO = 2. 0 )
C
      EPSIL = ONE
C
   10 IF ( EPSIL + ONE .GT. ONE ) THEN
         EPSIL = EPSIL / TWO
         GOTO 10
      ENDIF
      EPSIL = TWO * EPSIL
C
      END
```