Algorithm AS 197: A Fast Algorithm for the Exact Likelihood of Autoregressive-Moving Average Models

G. Melard

Algorithm AS 197

A Fast Algorithm for the Exact Likelihood of Autoregressive-moving Average Models

By G. Mélard

Université Libre de Bruxelles, Belgium

[Received July 1982. Revised June 1983]

Keywords: Maximum likelihood; Autoregressive-moving average model; Fast algorithm

Fortran 66

Description and Purpose

This algorithm has the same purpose as Algorithm AS 154 of Gardner et al. (1980), namely to compute the exact likelihood function of a stationary autoregressive-moving average (ARMA) process of order \((p, q)\). That algorithm appears to be slower, and requires more storage than is necessary, particularly for large \(p\) and \(q\). The computer program described here is a combination of an improved version of an algorithm due to Pearlman (1980) with the quick recursion switching suggested by Gardner et al. (1980) and an algorithm of Wilson (1979). The program is extremely efficient both in terms of computing time and amount of storage.

Theory and Method

We want to compute the likelihood function of the ARMA \((p, q)\) process, defined by the equation

\[
 w_t = \phi_1 w_{t-1} + \ldots + \phi_p w_{t-p} + a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}
\]

associated to a time series \((w_t; t = 1, \ldots, n)\), under the assumptions that the \(a_t\) are normally and independently distributed with zero mean and constant variance \(\sigma^2\) and that the process is stationary. The basic principle consists (Ansley, 1979; Harvey and Phillips, 1979) of computing the values taken by the innovations \(\tilde{a}_t\) of the stochastic process \((w_t; t = 1, 2, \ldots)\). The likelihood is then given by the expression

\[
 (2\pi)^{-n/2} \left( \prod_{t=1}^{n} \sigma_t \right)^{-1} \exp \left\{ -\frac{1}{2} \sum_{t=1}^{n} (\tilde{a}_t/\sigma_t)^2 \right\},
\]

where \(\sigma_t = h_t \sigma\) is the standard deviation of \(\tilde{a}_t\). Maximizing (2) with respect to the parameters included in \(\phi\) and \(\theta\) is equivalent (Ansley, 1979) to minimizing the sum of squares

\[
 \left( \prod_{t=1}^{n} h_t^2 \right)^{1/n} \sum_{t=1}^{n} \left( \frac{\tilde{a}_t}{h_t} \right)^2.
\]

The maximum likelihood estimate of \(\sigma^2\) is then given by \(n^{-1} \Sigma (\tilde{a}_t/h_t)^2\), evaluated at the optimal parameter point. Subroutine FLIKAM can be used to compute (3) as the product FACT*SUMSQ.

One method of obtaining the \(\tilde{a}_t\) (Caines and Rissanen, 1974; Gardner et al., 1980) is to use the Kalman filter recursions based on a state space representation of (1), e.g.

Present address: Institut de Statistique—C.P. 210, Université Libre de Bruxelles, Bruxelles, Belgium.
where \( W_t \) is the \( r \times 1 \) state vector, \( r = \max (p, q + 1) \), \( H = (1 \ldots 0) \), \( G' = (1 - \theta_1 \ldots -\theta_{r-1}) \), \( F' = (F_{i,j}) \) is an \( r \times r \) matrix such that \( F_{i,1} = \phi_i \) and \( F_{i,j} = \delta_{i,j+1} \) \( (i = 2, \ldots, r) \) for \( i = 1, 2, \ldots, r \), with the notations \( \phi_i = 0, i > p \), \( \theta_0 = -1 \) and \( \theta_i = 0, i > q \). Pearlman (1980) has suggested replacing the (matrix) Ricatti-type difference equation used in the Kalman filter by a (vector) Chandrasekhar-type difference equation, see Rissanen (1973), Lindquist (1974) and Morf et al. (1974). His first algorithm consists of the recursions

\[
\hat{a}_t = w_t - HW_t, \\
\hat{W}_{t+1} = FW_t + K_t(\hat{a}_t/h_t^2), \\
K_{t+1} = K_t - \alpha_t FL_t, \\
L_{t+1} = FL_t - \alpha_t K_t, \\
h_{t+1}^2 = h_t^2(1 - \alpha_t^2),
\]

where \( \alpha_t \) is written for \( HL_t/h_t^2 \). The same idea was implicit in the paper of Caines and Rissanen (1974, p. 103, footnote).

Our implementation of this algorithm includes an improvement for the case where \( p > q \). Indeed, (5–6) implies that

\[
w_t - \sum_{j=1}^{p(t)} \phi_j w_{t-j} = \hat{a}_t - \sum_{j=1}^{p(t)} \phi_j \hat{a}_{t-j} + \sum_{j=1}^{q(t)} K_{t-j,j}^{(t)} \hat{a}_{t-j},
\]

where \( p(t) = \min (t - 2, p) \), \( q(t) = \min (t - 1, r) \) and \( K_{t-j,j}^{(t)} \) is the \( j \)th element of \( K_{t-j} \). Since \( \hat{a}_t \) is the innovation at time \( t \) of the stochastic process \( (w_t; t = 1, 2, \ldots) \), the right-hand side of (10) is, for \( t \gg p + 2 \), the innovations representation (Cramér, 1961) of a non-stationary moving average process of order \( q \) which is known to be unique. Hence the terms for \( j > q \) vanish, so that \( K_{t-j,j}/h_{t-j}^2 = \phi_j \). Consequently, the \( j \)th elements of \( (K_t/h_t^2) \), \( j = q + 1, \ldots, r \), do not change when \( t \gg p - q + 1 \). Updating of these elements of \( K_t \) and \( L_t \) by (7)–(8) can be skipped over and the ratio \( (K_{t,j}/h_t^2) \) must accordingly be replaced by \( \phi_j \) in (6) for \( j \gg p + 1 \). Note that Pearlman (1980) states a similar property for his second algorithm (erroneously for \( t \gg q + 1 \) instead of \( t \gg p - q + 1 \)). The present variant of his first algorithm, in terms of the total number of multiplications and divisions, is as fast as these two algorithms, uniformly in \( p \) and \( q \).

The starting conditions for (5)–(9) are \( \hat{W}_1 = 0, K_1 = L_1 = FPH', h_1^2 = HPH' \), where \( P \) is the covariance matrix in the marginal distribution of \( W_t \). Since

\[
W_{t,i} = \sum_{j=i}^{r} (\phi_j w_{t-j+i-1} - \theta_{j-1} a_{t-j+i}),
\]

we have \( P_{1,1} = \gamma_0 \) and \( P_{t,1} = \mu_t \), where

\[
\mu_t = \sum_{j=i}^{r} (\phi_j \gamma_{j-i+1} - \theta_{j-1} \lambda_{j-i}), \quad i = 1, \ldots, r
\]

and

\[
\sigma^2 \gamma_k = \text{cov} (w_t, w_{t-k}), \quad \sigma^2 \lambda_k = \text{cov} (w_t, a_{t-k}), \quad k = 0, 1, \ldots
\]

Hence
\[ K_{1,i} = L_{1,i} = \phi_i \gamma_0 + \mu_{i+1}, \]  
(11)

where we let \( \mu_{r+1} = 0 \). Note that Gardner et al. (1980) mention the suggestion from a referee that the autocovariances be used in the calculation of \( P \).

The autocovariances \( \gamma_k (k = 0, 1, \ldots, R) \), where \( R = \text{max} (p, q) \), are determined by an algorithm due to Wilson (1979). The covariances \( \lambda_k (k = 0, 1, \ldots, q) \) are then given by the formula

\[
\lambda_k = -\theta_k + \sum_{j=1}^{\min(p, k)} \phi_j \lambda_{q-j}.
\]

The \( \gamma_k \) and \( \lambda_k \) are obtained by using subroutine TWACF.

We have retained the proposal of Gardner et al. (1980) to allow for a switching from the state space recursions (5)-(9) to the quick recursions

\[
\hat{a}_t = w_t - \sum_{j=1}^{p} \phi_j w_{t-j} + \sum_{j=1}^{q} \theta_j \hat{a}_{t-j},
\]

(12)

\[ h^2_{t+1} = 1 \]

as soon as \( h^2 < 1 + \delta \), where \( \delta \) is a small positive real number. The switching has been delayed until \( t \geq r + 1 \) in order to avoid unnecessary complications. For the same reason, unnecessary updating of some elements of \( K_t \) and \( L_t \) has been maintained until \( t \geq p - q + 1 \). The switching always occurs at time \( p + 1 \) for pure autoregressive processes.

Note also that working storage is restricted to three vectors of length \( r + 1 \), which is really negligible by comparison with the \( n^2 \) memory cells required by the direct inversion of the covariance matrix. Our implementation of Wilson's algorithm uses the same three vectors as workspace without the need to reconstruct some coefficients (compare with Wilson, 1979, p. 303).

The algorithm can still be improved in the case of a seasonal moving average process, defined by the equation

\[ w_t = \theta(B) \Theta(B^q) a_t, \]

(13)

where \( B \) is the backshift operator, \( \theta(B) \) is a polynomial in \( B \) of degree \( q' \), \( \Theta(B^q) \) is a polynomial in \( B^q \) of degree \( q'' \) and \( s \) is the length of the seasonal cycle, such that \( q' < s \). Let \( \sigma^2 \Omega \) be the covariance matrix of \( w = (w_1, \ldots, w_n)' \) and \( \Omega = TT' \), the Cholesky factorization of \( \Omega \), where \( T \) is a lower triangular matrix. The elements \( T_{ij} \) of \( T \) are related to the elements \( K_{t,j} \) of vectors \( K_t \) by \( T_{t,j} = K_{t-j,j} / h_{t-j} \). Consequently by Theorem 4.1 of Ansley (1979), we have \( K_{t,j} = 0 \) for \( t = h's + k - j \) and \( j = h''s + l \) with \( h', h'' = 0, 1, \ldots, k = 1, \ldots, s - q' \), and \( l = q' + 1, \ldots, s - 1 \). There is no simple generalization of this property when \( p > 0 \), except when the autoregressive operator \( 1 - \phi_1 B - \ldots - \phi_p B^p \) is a polynomial in \( B^q \). A transformation like the one suggested by Ansley (1979, p. 64) can be used for mixed models but, since the algorithm is restricted to a time-invariant state space representation (4), the Kalman filter algorithm would become necessary. These refinements for seasonal models are not implemented in subroutine FLIKAM.

**Structure**

**SUBROUTINE FLIKAM(P, MP, Q, MQ, W, E, N, SUMSQ, FACT, VW, VL, MRP1, VK, MR, TOLER, IFault)**

**Formal parameters**

\( P \): Real array (\( MP \))

\( MP \): Integer

*input*: the value of \( \phi \) in the first \( p \) locations

*input*: the value of \( p \)
The constant $10^{-10}$ used in the eighth failure test is identified by variable `EPSIL1`, which is `DATA`-initialized.
Constant

The constant $10^{-10}$ used in the fifth failure test is known as variable $EPSIL2$, and is DATA-initialized.

Precision

On machines with small word length, all the real variables should be replaced by double precision variables. Overflow or underflow will not occur in the calculation of $(11 h^2)^{1/n}$ because the product is stored in the form $a2^b$ (Martin and Wilkinson, 1965; Ansley, 1979).

Time

The number of time-consuming operations—multiplications and divisions—is given by the formula

$$N_n(p, q) = N_0(p, q) + N(n, p, q),$$

where

$$N_0(p, q) = p^2 + \frac{q^2}{2} + 2pR + qS + \frac{R^2}{2},$$

$$N(n, p, q) = n(p + 3q + S),$$

where $R = \max(p, q)$, and $S = \min(p, q)$. Terms of lower power have been omitted. Note that the use of McLeod's (1975) algorithm would have given a term $O(p^3/2)$ in $N_0(p, q)$. The approximate conditional and unconditional methods (Box and Jenkins, 1976) correspond respectively to

$$N'(n, p, q) = n(p + q),$$

$$N''(n, p, q) = \{2n + 2\nu + (k - 1)(2n + 4\nu)\} (p + q),$$

where $\nu$ is the maximum leadtime for backforecasting or forecasting and $k$ is the number of iterations of the backforecasting procedure. With $\nu = n/2$ and $k = 1$, we obtain $N''(n, p, q) = 3n(p + q)$, i.e. more than $N(n, p, q)$ for the exact method. Table 1 shows the average computation times in milliseconds required to evaluate the conditional sum of squares (by the code used by Gardner et al., and that in the comment in FLIKAM), and the exact likelihood (by a method of Ansley improved, see Mélard, 1982, by the Algorithm AS 154 of Gardner et al. and the present algorithm).

<table>
<thead>
<tr>
<th>Model</th>
<th>Conditional method</th>
<th>Exact method</th>
<th>Comment in FLIKAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AS 154</td>
<td>AS 154</td>
<td>AS 197</td>
</tr>
<tr>
<td>$(p, q)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 0)</td>
<td>1.1</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1.2</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>(2, 0)</td>
<td>1.2</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>(0, 2)</td>
<td>1.6</td>
<td>0.8</td>
<td>4.6</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>1.5</td>
<td>0.9</td>
<td>4.1</td>
</tr>
<tr>
<td>(13, 0)</td>
<td>2.7</td>
<td>1.9</td>
<td>12</td>
</tr>
<tr>
<td>(12, 1)</td>
<td>3.1</td>
<td>1.9</td>
<td>13</td>
</tr>
<tr>
<td>(1, 12)</td>
<td>4.4</td>
<td>1.9</td>
<td>19</td>
</tr>
<tr>
<td>(0, 13)</td>
<td>4.6</td>
<td>1.9</td>
<td>21</td>
</tr>
</tbody>
</table>

The ratio of the computing times of the
exact likelihood over the conditional approximation is not greater than 4 for most of the models which have been tried and is less than 3 for the high-order models. If the quick recursion switching is allowed for still better results can be obtained when \( q > 0 \). For a more complete discussion of this point, see Gardner et al. (1980). We have given the code used for the conditional method in the comment near the end of SUBROUTINE FLIKAM. The code given in the program of Gardner et al. (1980) was also considered. We recall that even ratios can be highly dependent on the computer and on the compiler, especially its level of optimization (see Mélard, 1982). Anyway, the code was written in order to possibly take advantage of compiler optimization, in accordance with the recommendations of, for example, Kernighan and Plauger (1978).

If computation of the ratio \( \hat{a}_r/h_t \) is not necessary (e.g. when a general purpose optimization algorithm is used instead of a non-linear least-squares algorithm) the algorithm can be slightly modified in order to avoid all square roots except \( q \) of them when switching to quick recursions occurs.

**Related Algorithms**

The number of multiplications and divisions required by the original algorithms of Pearlman (1980) is always as large as ours. Pearlman (1980) pointed out that algorithms based on the Morf et al. recursions are not necessarily faster than the algorithm of Ansley (1979) based on a Cholesky factorization of a band matrix. Between the original Ansley’s algorithm, the improved version (Mélard, 1982) and the Kalman filter algorithm of Gardner et al., (1980), the second one is faster when \( p > q \) whereas the third one requires about \( p^2 r^4/2 \) operations for the determination of the starting matrix \( P \). The storage requirements are respectively \( mn, r^2 \) and \( r^4/8 \). For ARMA models with time-dependent coefficients, an algorithm has been given by Mélard (1982).

Execution times of the programs on series of length 100 in the experiments reported in Table 1 seem to confirm that the program of Gardner et al. (1980) should not be used. The algorithm of Ansley improved (with the algorithm of McLeod, to tell the truth) is sometimes nearly as fast as the present algorithm except for high-order models, confirming the conclusions of Pearlman (1980).

The possible improvements for seasonal moving average processes (13) would reduce the approximate number of multiplications and divisions at each time from \( 3(q^"s + q"') \) to \( 3q"(1 + 2q") \).

**Acknowledgements**

This paper was written while at the “Université de Montréal” under the sponsorship of the international co-operation between Belgium and the Province of Quebec.

We thank the Algorithm Editor and the referee, especially for suggesting the examination of models with sparse coefficients.

**References**


SUBROUTINE FLIKAM(P, MP, Q, MQ, W, E, N, SUMSQ, FACT, VW, VL, 
* MR, MR1, VK, MR, TOLER, IFault)

ALGORITHM AS 197  APPL. STATIST. (1984) VOL.33, NO.1
COMPUTES THE LIKELIHOOD FUNCTION OF AN AUTOREGRESSIVE-
MOVING AVERAGE PROCESS, EXPRESSED AS FACT*SUMSQ.

REAL P(MP), Q(MQ), W(N), E(N), VW(MR1), VL(MR1), VK(MR)

REAL FACT, SUMSQ, TOLER, EPSIL1, ZERO, P0625, ONE, TWO, FOUR,
* SIXTE, A, ALF, AOR, DETCAR, DETMAN, FLJ, FN, R, VL1, VW1

REAL ABS, SQRT

DATA EPSIL1 /1.0E-10/
DATA ZERO, P0625, ONE, TWO, FOUR, SIXTE, A, ALF, AOR, DETCAR, DETMAN, FLJ, FN, R, VL1, VW1

FACT = ZERO
DETMAN = ONE
DETCAR = ZERO
SUMSQ = ZERO
MXPQ = MAXO(MP, MQ)
MXPQP1 = MXPQ + 1
MQP1 = MQ + 1
MPP1 = MP + 1

CALCULATION OF THE AUTOCOVARIANCE FUNCTION OF A PROCESS WITH
UNIT INNOVATION VARIANCE (WV) AND THE COVARIANCES BETWEEN THE
VARIABLE AND THE LAGGED INNOVATIONS (VL).

CALL TWACF(P, MP, Q, MQ, WV, MXPQP1, VL, MXPQP1, VK, MXPQ, IFault)
IF (MR .NE. MAXO(MP, MQP1)) IFault = 6
IF (MRP1 .NE. MR + 1) IFault = 7
IF (IFault .GT. 0) RETURN

COMPUTATION OF THE FIRST COLUMN OF MATRIX P (VK)

VK(1) = WV(1)
IF (MR .EQ. 1) GOTO 150
DO 140 K = 2, MR
VK(K) = ZERO
IF (K .GT. MP) GOTO 120
DO 110 J = K, MP
JP2MK = J + 2 - K
VK(K) = VK(K) + P(J) * WV(JP2MK)
110 CONTINUE
120 IF (K .GT. MQP1) GOTO 140
DO 130 J = K, MQP1
JP1MK = J + 1 - K
VK(K) = VK(K) - Q(J - 1) * VL(JP1MK)
130 CONTINUE
140 CONTINUE
COMPUTATION OF THE INITIAL VECTORS L AND K (VL, VK).

150 R = VK(1)
   VL(MR) = ZERO
   DO 160 J = 1, MR
   VM(J) = ZERO
   IF (J .NE. MR) VL(J) = VK(J + 1)
   IF (J .LE. MP) VL(J) = VL(J) + P(J) * R
   VK(J) = VL(J)

160 CONTINUE

INITIALIZATION

LAST = MPP1 - MQ
LOOP = MP
JFROM = MPP1
VM(MPP1) = ZERO
VL(MXQPQ1) = ZERO

EXIT IF NO OBSERVATION, OTHERWISE LOOP ON TIME.

IF (N .LE. 0) GOTO 500
   DO 290 I = 1, N
   TEST FOR SKIPPED UPDATING
   IF (I .NE. LAST) GOTO 170
   LOOP = MIND(MP, MQ)
   JFROM = LOOP + 1

   TEST FOR SWITCHING
   IF (MQ .LE. 0) GOTO 300
   170 IF (R .LE. EPSIL) GOTO 400
      IF (ABS(R - ONE) .LT. TOLER .AND. I .GT. MXPQ) GOTO 300
   UPDATING Scalars

   DETMAN = DETMAN * R
   IF (ABS(DETMAN) .LT. ONE) GOTO 200
   DETMAN = DETMAN * PD625
   DETCAR = DETCAR + FOUR
   GOTO 190
   200 IF (ABS(DETMAN) .GE. PD625) GOTO 210
   DETMAN = DETMAN * SIXTEEN
   DETCAR = DETCAR - FOUR
   GOTO 200
   210 VW1 = VW(1)
      A = W(I) - VW1
      E(I) = A / SQRT(R)
      AOR = A / R
      SUMSQ = SUMSQ + A * AOR
      VL1 = VL(1)
      ALF = VL1 / R
      R = R - ALF * VL1
      IF (LOOP .EQ. 0) GOTO 230
   UPDATING VECTORS

   DO 220 J = 1, LOOP
      FLJ = VL(J + 1) + P(J) * VL1
      VM(J) = VM(J + 1) + P(J) * VW1 + AOR * VK(J)
      VL(J) = FLJ - ALF * VK(J)
      VK(J) = VK(J) - ALF * FLJ
   CONTINUE
   220 IF (JFROM .GT. MQ) GOTO 250
   DO 240 J = JFROM, MQ
      VWM(J) = VM(J + 1) + AOR * VK(J)
      VL(J) = VL(J + 1) - ALF * VK(J)
      VK(J) = VK(J) - ALF * VL(J + 1)
   CONTINUE
   240 IF (JFROM .GT. MP) GOTO 270
   250
DO 260 J = JFROM, MP
   260 VW(J) = VW(J + 1) + P(J) * W(I)
270 CONTINUE
290 CONTINUE
GOTO 390

C C QUICK RECURSIONS
C
300 NEXTI = I
   IFault = -NEXTI
   DO 310 I = NEXTI, N
500 CONTINUE
   E(I) = W(I)
   IF (MP .EQ. 0) GOTO 340
   DO 330 J = 1, MP
500 CONTINUE
   IMJ = I - J
   E(I) = E(I) - P(J) * W(IMJ)
320 CONTINUE
330 CONTINUE
340 IF (MQ .EQ. 0) GOTO 370
   DO 360 J = NEXTI, N
500 CONTINUE
   DO 350 J = 1, MQ
500 CONTINUE
   IMJ = I - J
   E(I) = E(I) + Q(J) * E(IMJ)
350 CONTINUE
360 CONTINUE

C C RETURN SUM OF SQUARES AND DETERMINANT
C
370 DO 380 I = NEXTI, N
380 SUMSQ = SUMSQ + E(I) * E(I)
   CODE FOR CONDITIONAL SUM OF SQUARES
   REPLACES ALL EXECUTABLE STATEMENTS UPTO AND
   INCLUDING THAT LABELLED 380
   FACT = ZERO
   DETMAN = ONE
   DETCAR = ZERO
   SUMSQ = ZERO
   MXPQ = MAXD(MP, MQ)
   DO 380 I=MXPQ,N
   E(I)=W(I)
   IF (MP.LE.0) GOTO 340
   DO 320 J=1,MP
500 CONTINUE
   IMJ=I-J
   E(I)=E(I)-P(J)*W(IMJ)
320 CONTINUE
340 IF (MQ.LE.0) GOTO 380
   DO 350 J=1,MQ
500 CONTINUE
   IMJ=I-J
   E(I)=E(I)+Q(J)*E(IMJ)
350 CONTINUE
380 SUMSQ=SUMSQ+E(I)*E(I)

C 390 FN = N
   FACT = DETMAN ** (ONE / FN) * TWO ** (DETCAR / FN)
   RETURN
C C EXECUTION ERRORS
C
400 IFault = 8
   RETURN
500 IFault = 9
   RETURN
   END
C C SUBROUTINE TWACF(P, MP, Q, MQ, ACF, MA, CVLI, MXPQP1, ALPH, MXPQ,
   * IFault)
C C ALGORITHM AS 197.1 APPL. STATIST. (1984) VOL.33, NO.1
IMPLEMENTATION OF THE ALGORITHM OF G. TUNNICLIFFE WILSON
(J. STATIST. COMPUT. SIMUL. 8, 1979, 301-309) FOR THE
COMPUTATION OF THE AUTOCOVARIANCE FUNCTION (ACF) OF AN ARMA
PROCESS OF ORDER (MP,MQ) AND UNIT INNOVATION VARIANCE.
THE AUTOREGRESSIVE AND MOVING AVERAGE COEFFICIENTS ARE STORED
IN VECTORS P AND Q, USING BOX AND JENKINS NOTATION. ON OUTPUT
VECTOR CVLI CONTAINS THE COVARIANCES BETWEEN THE VARIABLE AND
THE (K-1)-LAGGED INNOVATION, FOR K=1,...,MQ+1.

REAL P(MP), Q(MQ), ACF(MA), CVLI(MXPQP1), ALPHAP(MXPQ)

REAL EPSIL2, ZERO, HALF, ONE, TWO, DIV

DATA EPSIL2 /1.0E-10/
DATA ZERO, HALF, ONE, TWO /0.0, 0.5, 1.0, 2.0/

IFault = 0
IF (MP .LT. 0 .OR. MQ .LT. 0) IFault = 1
IF (MXPQ .NE. MAXD(MP, MQ)) IFault = 2
IF (MXPQP1 .NE. MXPQ + 1) IFault = 3
IF (MA .LT. MXPQP1) IFault = 4
IF (IFault .GT. 0) RETURN

INITIALIZATION AND RETURN IF MP=MQ=0

ACF(1) = ONE
CVLI(1) = ONE
IF (MA .EQ. 1) RETURN
DO 10 I = 2, MA
10 ACF(I) = ZERO
IF (MXPQP1 .EQ. 1) RETURN
DO 20 I = 2, MXPQP1
20 CVLI(I) = ZERO
DO 90 K = 1, MXPQ
90 ALPHAP(K) = ZERO

COMPUTATION OF THE A.C.F. OF THE MOVING AVERAGE PART,
STORED IN ACF.

IF (MQ .EQ. 0) GOTO 180
DO 130 K = 1, MQ
CVLI(K + 1) = -Q(K)
ACF(K + 1) = -Q(K)
KC = MQ - K
IF (KC .EQ. 0) GOTO 120
DO 110 J = 1, KC
JPK = J + K
ACF(K + 1) = ACF(K + 1) + Q(J) * Q(JPK)
110 CONTINUE
120 ACF(1) = ACF(1) + Q(K) * Q(K)
130 CONTINUE

INITIALIZATION OF CVLI = T.W.-S PHI -- RETURN IF MP=0.

180 IF (MP .EQ. 0) RETURN
DO 190 K = 1, MP
ALPHAP(K) = P(K)
CVLI(K) = P(K)
190 CONTINUE

COMPUTATION OF T.W.-S ALPHA AND DELTA
(DELTA STORED IN ACF WHICH IS GRADUALLY OVERWRITTEN)

DO 290 K = 1, MXPQ
KC = MXPQ - K
IF (KC .GE. MP) GOTO 240
DIV = ONE - ALPHAP(KC + 1) * ALPHAP(KC + 1)
IF (DIV .LE. EPSIL2) GOTO 700
IF (KC .EQ. 0) GOTO 290
DO 230 J = 1, KC
KCP1MJ = KC + 1 - J
ALPHAP(J) = (CVLI(J) + ALPHAP(KC + 1) * CVLI(KCP1MJ)) / DIV
230 CONTINUE
240 IF (KC .GE. MQ) GOTO 260
    J1 = MAX0(KC + 1 - MP, 1)
    DO 250 J = J1, KC
    KCP1MJ = KC + 1 - J
    ACF(J + 1) = ACF(J + 1) + ACF(KC + 2) * ALPHA(KCP1MJ)
250 CONTINUE
260 IF (KC .GE. MP) GOTO 290
    DO 270 J = 1, KC
    CVLI(J) = ALPHA(J)
290 CONTINUE
C
C COMPUTATION OF T.W.-S NU
C (NU IS STORED IN CVLI, COPIED INTO ACF)
C
ACF(1) = HALF * ACF(1)
DO 330 K = 1, MXPQ
  IF (K .GT. MP) GOTO 330
  KP1 = K + 1
  DIV = ONE - ALPHA(K) * ALPHA(K)
  DO 310 J = 1, KP1
  KP2MJ = K + 2 - J
  CVLI(J) = (ACF(J) + ALPHA(K) * ACF(KP2MJ)) / DIV
310 CONTINUE
DO 320 J = 1, KP1
320 ACF(J) = CVLI(J)
330 CONTINUE
C
C COMPUTATION OF ACF (ACF IS GRADUALLY OVERWRITTEN)
C
DO 430 I = 1, MA
  MIIM1P = MIN0(I - 1, MP)
  IF (MIIM1P .EQ. 0) GOTO 430
  DO 420 J = 1, MIIM1P
  IMJ = I - J
  ACF(I) = ACF(I) + P(J) * ACF(IMJ)
420 CONTINUE
430 CONTINUE
ACF(1) = ACF(1) * TWO
C
C COMPUTATION OF CVLI - RETURN WHEN MQ=0
C
CVLI(1) = ONE
IF (MQ .LE. 0) GOTO 600
DO 530 K = 1, MQ
  CVLI(K + 1) = -Q(K)
  DO 520 J = 1, Q(K)
  MIKP = MIN0(K, MP)
  DO 510 J = 1, MIKP
  KP1MJ = K + 1 - J
  CVLI(K + 1) = CVLI(K + 1) + P(J) * CVLI(KP1MJ)
520 CONTINUE
530 CONTINUE
C
600 RETURN
C
EXECUTION ERROR DUE TO (NEAR) NON-STATIONARITY
C
700 IFault = 5
RETURN
END