Algorithm AS 154: An Algorithm for Exact Maximum Likelihood Estimation of Autoregressive-Moving Average Models by Means of Kalman Filtering

G. Gardner, A. C. Harvey, G. D. A. Phillips

Applied Statistics, Volume 29, Issue 3 (1980), 311-322.

Algorithm AS 154

An Algorithm for Exact Maximum Likelihood Estimation of Autoregressive–Moving Average Models by Means of Kalman Filtering

By G. GARDNER
Service in Information and Analysis Ltd, London, UK

A. C. HARVEY
London School of Economics

and G. D. A. PHILLIPS
University of Leeds

Keywords: MAXIMUM LIKELIHOOD; AUTOREGRESSIVE-MOVING AVERAGE MODEL; KALMAN FILTER

LANGUAGE

Fortran 66

DESCRIPTION AND PURPOSE

The algorithm presented here enables the *exact* likelihood function of a stationary autoregressive-moving average (*ARMA*) process to be calculated by means of the Kalman filter; see Harvey and Phillips (1976, 1979). Two subroutines are basic to the algorithm. The first, subroutine *STARMA*, casts the *ARMA* model into the "state space" form necessary for Kalman filtering, and computes the covariance matrix associated with the initial value of the state vector. The second subroutine, *KARMA*, carries out the recursions and produces a set of standardized prediction errors, together with the determinant of the covariance matrix of the observations. These two quantities together yield the exact likelihood, and this may be maximized by an iterative procedure based on a numerical optimization algorithm which does not require analytic derivatives.

Subroutine *KARMA* contains a device whereby the likelihood may be approximated to a level of accuracy which is under the control of the user. This enables a considerable amount of computing time to be saved, with very little attendant loss in precision.

Finally, another subroutine, *KALFOR*, may be used to compute predictions of future values of the series, together with the associated conditional mean square errors.

THEORY

An autoregressive–moving average process is defined by

$$w_t = \phi_1 w_{t-1} + \ldots \phi_p w_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots \theta_q \varepsilon_{t-q}, \quad t = 1, \ldots, n, \tag{1}$$

where the ε_t's are normally and independently distributed with mean zero and variance σ^2, and w_t is observable. Such a process will be referred to as an $ARMA(p, q)$ process and the set of parameters $(\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q)$ will be denoted by (ϕ, θ).

An $ARMA(p, q)$ process may be put in "state space" form by defining an $r \times 1$ vector, α_t, which obeys the "transition equation"

$$\alpha_t = T\alpha_{t-1} + R\varepsilon_t, \quad t = 1, \ldots, n, \tag{2}$$

where $r = \max(p, q+1)$ and

$$T = \begin{bmatrix} \phi_1 & \vdots & \\ \vdots & & I_{r-1} \\ \phi_{r-1} & \vdots & \\ \phi_r & \vdots & O'_{r-1} \end{bmatrix}, \quad \text{and} \quad R = \begin{bmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{r-1} \end{bmatrix}. \tag{3}$$

Note that, unless $p = q+1$, some of the ϕ_i's or θ_j's will be identically equal to zero. The associated "measurement equation" is

$$w_t = (1 \;\; O'_{r-1})\alpha_t = z'_t \alpha_t, \quad t = 1,...,n. \tag{4}$$

Equations (2) and (4) constitute a linear dynamic model. Given a_{t-1}, an estimate of the state vector at time $t-1$, together with a matrix P_{t-1} defined by

$$E[(a_{t-1} - \alpha_{t-1})(a_{t-1} - \alpha_{t-1})'] = \sigma^2 P_{t-1},$$

a prediction of α_t, $a_{t|t-1}$, may be made. This may then be updated once the tth observations, w_t, becomes available. The prediction and updating are carried out by means of a set of recursive equations known as the "Kalman filter". The parmeter σ^2 does not appear in these recursions.

In order to start the recursions, an initial estimator of the state α_0 is needed, together with the associated matrix P_0. The best estimator of α_0 for the *ARMA* model is $a_0 = 0$, and the matrix P_0 is therefore given by $\sigma^{-2} E[\alpha_0 \alpha'_0]$. The evaluation of P_0 constitutes a key feature of the present algorithm, and the method employed is discussed at some length in the next section.

Application of the recursive formulae yields a set of n standardized residuals, denoted by $\tilde{v}_t$, $t = 1,...,n$, together with a set of n quantities, f_t, $t = 1,...,n$, proportional to the one-step prediction mean square errors. The log-likelihood function may then be maximized with respect to (ϕ, θ) by minimizing

$$L^*(\phi, \theta) = n \log S(\phi, \theta) + \sum_{t=1}^{n} \log f_t, \tag{5}$$

where $S(\phi, \theta) = \Sigma \tilde{v}_t^2$. The subroutine *KARMA* outputs the second term in expression (5) as *SUMLOG*, and $S(\phi, \theta)$ as *SSQ*.

An approximation to the likelihood may be obtained as follows. Once a certain number of observations, say t^*, have been processed, future values of $\tilde{v}_t$ are approximated by $\hat{v}_t$ which is obtained directly from the *ARMA* equation (1), i.e.

$$\hat{v}_t = w_t - \phi_1 w_{t-1} - ... \; \phi_p w_{t-p} - \theta_1 \hat{v}_{t-1} - ... \; \theta_q \hat{v}_{t-q}, \quad t = t^*+1, t^*+2, ..., \tag{6}$$

where $\hat{v}_t = \tilde{v}_t$, $t = t^*,..., t^*-q+1$. The value of t^*, the point at which the switch to the "quick recursions" takes place, is determined automatically as soon as $f_t < 1+\delta$. The choice of δ, which will generally be a small positive number, say 0·01 or 0·001, is open to the user. If δ is set equal to a negative number the full Kalman filter is carried out for all observations and the exact likelihood is obtained. Results concerning the trade-off between accuracy and computational efficiency for the approximation are given in Table 1. The figures show the time taken to compute the likelihood function for several values of θ, the parameter in a first-order moving average process. The results indicate that setting δ equal to a value of, say, 0·01 or 0·001 yields a negligible error of approximation while saving a considerable amount of computing time. This saving is particularly marked when $|\theta|$ is relatively small. With $\theta = 0{\cdot}5$, for example, the likelihood function may be computed very accurately in a time which is only marginally greater than that needed to compute the conditional sum of squares.

Once estimates of ϕ and θ have been obtained, one may wish to obtain predictions of future values of the series. The predicted value of w_{n+m}, and its mean square error, conditional on (ϕ, θ), are obtained from the recursions

$$a_{n+t|n} = Ta_{n+t-1|n},$$
$$P_{n+t|n} = TP_{n+t-1|n}T' + RR', \quad t = 1,\ldots,m, \tag{7}$$

where $a_{n|n} = a_n$ and $P_{n|n} = P_n$. The first element of a_{n+m} is the predicted value of w_{n+m}, while the top left-hand element of $P_{n+m|n}$ gives the associated conditional mean square error when multiplied by an estimate of σ^2.

METHOD

The initial matrix, P_0, obeys the equation

$$P_0 = TP_0T' + RR'. \tag{8}$$

If $V = RR'$, and if p_{ij}, t_{ij} and v_{ij} denote the element in the ith row and jth column of P_0, T and V respectively, then

$$p_{ij} = \sum_k \sum_l t_{ik} p_{kl} t_{jl} + v_{ij}, \tag{9}$$

i.e.

$$v_{ij} = p_{ij} - \sum_k \sum_l t_{ik} p_{kl} t_{jl}. \tag{10}$$

Thus each element of V is a linear combination of the elements of P_0. We may therefore write

$$\text{vec}(V) = S\,\text{vec}(P_0) \tag{11}$$

where S is an appropriate square matrix, whose form depends on the definition of vec$(\cdot)$. Expression (11) is a set of linear equations, from which we may obtain P_0.

We consider three definitions of vec(A), where A is a given symmetric square matrix:

(1) vec(A) is obtained by stacking the columns of A. In this case

TABLE 1

The evaluation of the likelihood for a MA(1) model by the modified Kalman filter method with different values of δ

		$n = 20$			$n = 60$		
		$\theta = 0{\cdot}5$	$\theta = 0{\cdot}8$	$\theta = 0{\cdot}99$	$\theta = 0{\cdot}5$	$\theta = 0{\cdot}8$	$\theta = 0{\cdot}99$
Exact likelihood	L^*	66·3606	66·9910	68·8726	255·643	255·903	259·166
	Time	0·36	0·36	0·35	0·95	0·95	0·94
$\delta = 0{\cdot}001$	L^*	66·3601	66·9906	—	255·641	255·903	—
	t^*	4	13	—	4	13	—
	Time	0·25	0·31	—	0·55	0·62	—
$\delta = 0{\cdot}01$	L^*	66·3541	66·9816	—‡	255·636	255·909	259·116
	t^*	3	8	—	3	8	54
	Time	0·24	0·27	—	0·55	0·58	0·91
$\delta = 0{\cdot}1$	L^*	66·2636	66·7719	68·8116	255·704	255·794	259·142
	t^*	1	3	9	1	3	9
	Time	0·23	0·24	0·28	0·53	0·55	0·58
Conditional sum of squares	L^*	66·0853	65·9915	66·4512	256·528	257·729	264·634
	Time	0·22	0·22	0·21	0·53	0·52	0·51

† Number of seconds on an ICL 4130 computer.

‡ A "—" indicates that no switch occurred; i.e. $f_t \geqslant 1 + \delta$ for all t.

$$S = I - T \otimes T'; \tag{12}$$

see Harvey and Phillips (1976).

(2) vec(A) is obtained by stacking the columns of the lower triangular part of A. This makes use of the symmetry of V and P_0, and reduces the problem to one of solving $r(r+1)/2$ linear equations.

When we take into account the form of T, we see that the matrix S contains many zeroes. We may therefore solve (10) by means of a series of Givens transformations of S, thus obtaining the QR decomposition of S. The matrix S is processed row by row, and the method takes into account leading zeros in the rows of S to reduce computing time. Solving the equations this way on an ICL 4130 computer for a $MA(4)$ process was faster than the standard NAG routine by a factor of about three.

A further saving in time may be made for pure moving average processes when the equation (10) forms a triangular system, and P may be found by backsubstitution.

(3) vec(A) is obtained by stacking the columns of the lower triangular part of A, beginning at column 2, with the first column attached at the end. This formulation attempts to bring more leading zeroes in the rows of S, leading to a reduction in the time taken to evaluate P_0. This formulation is used in subroutine *STARMA* for processes with auto-regressive components. For pure moving average processes, the method described in the preceding paragraph is used. [The referee has suggested an alternative approach in which P_0 is derived from the dispersion matrix of $w_0, \ldots, w_{1-r}, \varepsilon_0, \ldots, \varepsilon_{1-r}$, and computed using the algorithm of McLeod (1975).]

The recursions in subroutine *KARMA* are programmed efficiently by taking account of the zero values which arise in predetermined positions in the various matrices. An important saving is effected in computing $TP_t T'$ in the prediction recursion because of the special nature of P_t; see Harvey and Phillips (1976).

Structure

SUBROUTINE STARMA(IP, IQ, IR, NP, PHI, THETA, A, P, V, THETAB, XNEXT, XROW, RBAR, NRBAR, IFAULT)

Formal parameters

IP	Integer	input : the value of p
IQ	Integer	input : the value of q
IR	Integer	input : the value of $r = \max(p, q+1)$
NP	Integer	input : the value of $r(r+1)/2$
PHI	Real array (*IR*)	input : the value of ϕ in the first p locations output : contains the first column of T
THETA	Real array (*IR*)	input : the value of θ in the first q locations
A	Real array (*IR*)	output : on exit contains a_0
P	Real array (*NP*)	output : on exit contains P_0, stored as a lower triangular matrix, column by column
V	Real array (*NP*)	output : on exit contains RR', stored as a lower triangular matrix, column by column
THETAB	Real array (*NP*)	workspace : used to calculate P
XNEXT	Real array (*NP*)	workspace : used to calculate P
XROW	Real array (*NP*)	workspace : used to calculate P
RBAR	Real array (*NRBAR*)	workspace : used to calculate P
NRBAR	Integer	input : the value of $NP*(NP-1)/2$
IFAULT	Integer	output : a fault indicator, equal to 1 if $IP<0$ 2 if $IQ<0$ 3 if $IP<0$ and $IQ<0$

4 if $IP = IQ = 0$
5 if $IR \neq MAX(IP, IQ+1)$
6 if $NP \neq IR*(IR+1)/2$
7 if $NRBAR \neq NP*(NP-1)/2$
8 if $IP = 1$ and $IQ = 0$
(Subroutine *STARMA* is not appropriate for an $AR(1)$ process)
0 otherwise.

SUBROUTINE KARMA(IP, IQ, IR, NP, PHI, THETA,A, P, V, N, W, RESID, SUMLOG, SSQ, IUPD, DELTA,E, NIT)

Formal parameters

IP	Integer	input : the value of p
IQ	Integer	input : the value of q
IR	Integer	input : the value of $r = \max(p, q+1)$
NP	Integer	input : the value of $r(r+1)/2$
PHI	Real array (*IR*)	input : the first column of T
THETA	Real array (*IR*)	input : the value of θ in the first q locations
A	Real array (*IR*)	input : contains a_0
		output : contains a_t, where $t = t*$
P	Real array (*NP*)	input : contains P_0
		output : contains P_t, where $t = t*$
V	Real array (*NP*)	input : contains RR'
N	Integer	input : n, the number of observations
W	Real array (*N*)	input : the observations
RESID	Real array (*N*)	output : the corresponding standardized prediction errors
SUMLOG	Real	input : initial value of $\Sigma \log f_t$ (zero if no previous observations)
		output : final value of $\Sigma \log f_t$
SSQ	Real	input : initial value of $\Sigma \tilde{v}_t^2$ (zero if no previous observations)
		output : final value of $\Sigma \tilde{v}_t^2$
IUPD	Integer	input : if $IUPD = 1$ the prediction equations are by-passed for the first observation. This is necessary when the value of P_0 has been obtained from *STARMA*. In this case, $P_{1\|0} = P_0$ and $a_{1\|0} = a_0$ and using the prediction equations as coded in *KARMA* would lead to erroneous results. For values other than 1, the prediction equations are not by-passed
DELTA	Real	input : when $NIT = 0$ this parameter determines the level of approximation. Negative *DELTA* ensures that the Kalman filter is used for all observations. Otherwise the filter is performed while $f_t \geqslant 1+\delta$, "quick recursions" being used thereafter
E	Real array (*IR*)	workspace : used to store the last q standardized prediction errors

NIT	Integer	input : when set to zero see description of *DELTA* for the effect of *NIT*; for non-zero values, the "quick recursions" are performed throughout, so that a conditional likelihood is obtained output : number of observations dealt with by the Kalman filter, i.e. t^*

SUBROUTINE KALFOR(M, IP, IR, NP, PHI, A, P, V, WORK)

Formal parameters

M	Integer	input : the value of m, the number of steps ahead for which predictor is required
IP	Integer	input : the value of p
IR	Integer	input : the value of r
NP	Integer	input : $r(r+1)/2$
PHI	Real array (*IR*)	input : contains the first column of T, the transition matrix
A	Real array (*IR*)	input : current value of a_t output : predicted value of a_{t+m}
P	Real array (*NP*)	input : current value of P_t, stored in lower triangular form, column by column output : predicted value of P_{t+m}
V	Real array (*NP*)	input : contains RR' stored in lower triangular form, column by column
WORK	Real array (*IR*)	workspace :

Auxiliary algorithms

The subroutine *STARMA* calls the auxiliary algorithms *INCLU2* (Farebrother, 1976) and *REGRES* (Gentleman, 1974). These algorithms were originally presented as Algol 60 procedures. The following modified Fortran 66 versions of these procedures are listed after subroutine *KALFOR*:

SUBROUTINE INCLU2(NP, NRBAR, WEIGHT, XNEXT, XROW, YNEXT, D, RBAR, THETAB, SSQERR, RECRES, IRANK, IFAULT)
SUBROUTINE REGRES(NP, NRBAR, RBAR, THETAB, BETA)

The formal parameters of these subroutines correspond to those in the original Algol procedures except that NRBAR contains the value $p(p-1)/2$, *XNEXT* contains the independent variables for the current observation, and *XROW* is workspace. Both *XNEXT* and *XROW* are real arrays of length *NP*, and the values in *XNEXT* are unchanged by a call of *INCLU2*.

TIME

The figures in Table 2 show the number of seconds taken to compute the likelihood function of various $MA(q)$ models for set values of the *MA* parameters. The computations were carried out on an ICL 4130 machine at the University of Kent. The classical method of evaluating the exact likelihood function involves estimating the pre-sample residuals; see Box and Jenkins (1970, Chapter 7). Our algorithm was written to be as efficient as possible, employing what is essentially a specialization of the method described by Osborn (1977) for the *MA* multivariate case; see Harvey and Phillips (1976). The "quick recursions" were *not* used in the Kalman filter algorithm. However, the Kalman filter algorithm appears to be marginally faster than the classical method for small sample sizes. Both methods of computing the exact likelihood function take significantly longer than the conditional sum of squares approximation, although

TABLE 2

Comparison of times† required to evaluate the likelihood for MA(q) models for different sample sizes

q	n	Conditional sum of squares	Classical	Kalman filter
1	20	0·23	0·39	0·37
	40	0·39	0·61	0·65
	60	0·56	0·87	0·93
	80	0·73	1·12	1·19
	100	0·89	1·38	1·46
2	20	0·23	0·46	0·41
	60	0·58	1·14	1·16
	100	0·92	1·81	1·77
3	20	0·23	0·59	0·50
4	20	0·24	0·75	0·60

† Number of seconds required to execute each of the three algorithms on an ICL 4130 computer.

the results in Table 1 suggest a considerable saving of time in the Kalman filter method when the quick recursions are used.

Although these computational comparisons are restricted to *MA* models, we believe that for general *ARMA* models the conclusions concerning the performance of our Kalman filter algorithm, *vis-à-vis* algorithms based on other methods are likely to be similar. Indeed a yardstick for gauging the relative efficiency of the Kalman filter method in this context is provided by noting that the time taken to evaluate the likelihood of an $ARMA(5,4)$ model was 1·00 seconds for $n = 20$. This figure may be compared with the corresponding figure for the $MA(4)$ process presented in Table 2. The dimensions of the matrices in the filter are exactly the same for these two processes, but the likelihood for the pure *MA* model is computed more rapidly because of the special features exploited in evaluating P_0.

ACKNOWLEDGEMENTS

We are grateful to the SSRC for financial support in connection with our project "Testing for Specification Error in Econometric Models" which was carried out at the University of Kent in 1976–77. We would also like to thank the editor and referee for their comments and suggestions.

REFERENCES

BOX, G. E. P. and JENKINS, G. M. (1970). *Time Series Analysis: Forecasting and Control.* San Francisco: Holden-Day.

FAREBROTHER, R. W. (1976). Remark AS R17. Recursive residuals—a remark on Algorithm AS 75. Basic procedures for large, sparse or weighted linear least squares problems. *Appl. Statist.*, **25**, 323–324.

GENTLEMAN, W. M. (1974). Algorithm AS 75. Basic procedures for large, sparse or weighted linear least squares problems. *Appl. Statist.*, **23**, 448–454.

HARVEY, A. C. and PHILLIPS, G. D. A. (1976). Maximum likelihood estimation of autoregressive-moving average models by Kalman filtering. University of Kent: QSS Discussion Paper No. 38.

—— (1979). Maximum likelihood estimation of regression models with autoregressive-moving average disturbances. *Biometrika*, **66**, 49–58.

MCLEOD, I. (1975). The derivation of the theoretical autocovariance function of autoregressive-moving average time series. *Appl. Statist.*, **24**, 255–256.

OSBORN, D. R. (1977). Exact and approximate maximum likelihood estimators for vector moving average processes. *J. R. Statist. Soc. B*, **39**, 114–118.

```
      SUBROUTINE STARMA(IP, IQ, IR, NP, PHI, THETA, A, P, V, THETAB,
     *  XNEXT, XROW, RBAR, NRBAR, IFAULT)
C
C        ALGORITHM AS 154  APPL. STATIST. (1980) VOL.29, NO.3
C
C        INVOKING THIS SUBROUTINE SETS THE VALUES OF V AND PHI, AND
C        OBTAINS THE INITIAL VALUES OF A AND P.
C        THIS ROUTINE IS NOT SUITABLE FOR USE WITH AN AR(1) PROCESS.
C        IN THIS CASE THE FOLLOWING INSTRUCTIONS SHOULD BE USED FOR
C        INITIALISATION.
C        V(1) = 1.0
C        A(1) = 0.0
C        P(1) = 1.0 / (1.0 - PHI(1) * PHI(1))
C
      DIMENSION PHI(IR), THETA(IR), A(IR), P(NP), V(NP), THETAB(NP),
     *  XNEXT(NP), XROW(NP), RBAR(NRBAR)
C
C        CHECK FOR FAILURE INDICATION.
C
      IFAULT = 0
      IF (IP .LT. 0) IFAULT = 1
      IF (IQ .LT. 0) IFAULT = IFAULT + 2
      IF (IP * IP + IQ * IQ .EQ. 0) IFAULT = 4
      K = IQ + 1
      IF (K .LT. IP) K = IP
      IF (IR .NE. K) IFAULT = 5
      IF (NP .NE. IR * (IR + 1) / 2) IFAULT = 6
      IF (NRBAR .NE. NP * (NP - 1) / 2) IFAULT = 7
      IF (IR .EQ. 1) IFAULT = 8
      IF (IFAULT .NE. 0) RETURN
C
C        NOW SET A(0), V AND PHI.
C
      DO 10 I = 2, IR
      A(I) = 0.0
      IF (I .GT. IP) PHI(I) = 0.0
      V(I) = 0.0
      IF (I .LE. IQ + 1) V(I) = THETA(I - 1)
   10 CONTINUE
      A(1) = 0.0
      IF (IP .EQ. 0) PHI(1) = 0.0
      V(1) = 1.0
      IND = IR
      DO 20 J = 2, IR
      VJ = V(J)
      DO 20 I = J, IR
      IND = IND + 1
      V(IND) = V(I) * VJ
   20 CONTINUE
C
C        NOW FIND P(0).
C
      IF (IP .EQ. 0) GOTO 300
C
C        THE SET OF EQUATIONS S * VEC(P(0)) = VEC(V)
C        IS SOLVED FOR VEC(P(0)).
C        S IS GENERATED ROW BY ROW IN THE ARRAY XNEXT.
C        THE ORDER OF ELEMENTS IN P IS CHANGED, SO AS TO
C        BRING MORE LEADING ZEROS INTO THE ROWS OF S,
C        HENCE ACHIEVING A REDUCTION OF COMPUTING TIME.
C
      IR1 = IR - 1
      IRANK = 0
      IFAIL = 0
      SSQERR = 0.0
      DO 40 I = 1, NRBAR
   40 RBAR(I) = 0.0
      DO 50 I = 1, NP
      P(I) = 0.0
      THETAB(I) = 0.0
      XNEXT(I) = 0.0
```

```
   50 CONTINUE
      IND = 0
      IND1 = 0
      NPR = NP - IR
      NPR1 = NPR + 1
      INDJ = NPR1
      IND2 = NPR
      DO 110 J = 1, IR
      PHIJ = PHI(J)
      XNEXT(INDJ) = 0.0
      INDJ = INDJ + 1
      INDI = NPR1 + J
      DO 110 I = J, IR
      IND = IND + 1
      YNEXT = V(IND)
      PHII = PHI(I)
      IF (J .EQ. IR) GOTO 100
      XNEXT(INDJ) = -PHII
      IF (I .EQ. IR) GOTO 100
      XNEXT(INDI) = XNEXT(INDI) - PHIJ
      IND1 = IND1 + 1
      XNEXT(IND1) = -1.0
  100 XNEXT(NPR1) = -PHII * PHIJ
      IND2 = IND2 + 1
      IF (IND2 .GT. NP) IND2 = 1
      XNEXT(IND2) = XNEXT(IND2) + 1.0
      WEIGHT = 1.0
      CALL INCLU2(NP, NRBAR, WEIGHT, XNEXT, XROW, YNEXT,
     *  P, RBAR, THETAB, SSQERR, RECRES, IRANK, IFAIL)
      XNEXT(IND2) = 0.0
      IF (I .EQ. IR) GOTO 110
      XNEXT(INDI) = 0.0
      INDI = INDI + 1
      XNEXT(IND1) = 0.0
  110 CONTINUE
      CALL REGRES(NP, NRBAR, RBAR, THETAB, P)
C
C        NOW RE-ORDER P.
C
      IND = NPR
      DO 200 I = 1, IR
      IND = IND + 1
      XNEXT(I) = P(IND)
  200 CONTINUE
      IND = NP
      IND1 = NPR
      DO 210 I = 1, NPR
      P(IND) = P(IND1)
      IND = IND - 1
      IND1 = IND1 - 1
  210 CONTINUE
      DO 220 I = 1, IR
  220 P(I) = XNEXT(I)
      RETURN
C
C        P(0) IS OBTAINED BY BACKSUBSTITUTION FOR
C        A MOVING AVERAGE PROCESS.
C
  300 INDN = NP + 1
      IND = NP + 1
      DO 310 I = 1, IR
      DO 310 J = 1, I
      IND = IND - 1
      P(IND) = V(IND)
      IF (J .EQ. 1) GOTO 310
      INDN = INDN - 1
      P(IND) = P(IND) + P(INDN)
  310 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE KARMA(IP, IQ, IR, NP, PHI, THETA, A, P,
     *  V, N, W, RESID, SUMLOG, SSQ, IUPD, DELTA, E, NIT)
C
C        ALGORITHM AS 154.1  APPL. STATIST. (1980) VOL.29, NO.3
C
C        INVOKING THIS SUBROUTINE UPDATES A, P, SUMLOG AND SSQ BY
C        INCLUSION OF DATA VALUES W(1) TO W(N). THE CORRESPONDING
C        VALUES OF RESID ARE ALSO OBTAINED.
C        WHEN FT IS LESS THAN (1 + DELTA), QUICK RECURSIONS ARE USED.
C
      DIMENSION PHI(IR), THETA(IR), A(IR), P(NP), V(NP),
     *  W(N), RESID(N), E(IR)
      IR1 = IR - 1
      DO 10 I = 1, IR
   10 E(I) = 0.0
      INDE = 1
C
C        FOR NON-ZERO VALUES OF NIT, PERFORM QUICK RECURSIONS.
C
      IF (NIT .NE. 0) GOTO 600
      DO 500 I = 1, N
      WNEXT = W(I)
C
C        PREDICTION.
C
      IF (IUPD .EQ. 1 .AND. I .EQ. 1) GOTO 300
C
C        HERE DT = FT - 1.0
C
      DT = 0.0
      IF (IR .NE. 1) DT = P(IR + 1)
      IF (DT .LT. DELTA) GOTO 610
      A1 = A(1)
      IF (IR .EQ. 1) GOTO 110
      DO 100 J = 1, IR1
  100 A(J) = A(J + 1)
  110 A(IR) = 0.0
      IF (IP .EQ. 0) GOTO 200
      DO 120 J = 1, IP
  120 A(J) = A(J) + PHI(J) * A1
  200 IND = 0
      INDN = IR
      DO 210 L = 1, IR
      DO 210 J = L, IR
      IND = IND + 1
      P(IND) = V(IND)
      IF (J .EQ. IR) GOTO 210
      INDN = INDN + 1
      P(IND) = P(IND) + P(INDN)
  210 CONTINUE
C
C        UPDATING.
C
  300 FT = P(1)
      UT = WNEXT - A(1)
      IF (IR .EQ. 1) GOTO 410
      IND = IR
      DO 400 J = 2, IR
      G = P(J) / FT
      A(J) = A(J) + G * UT
      DO 400 L = J, IR
      IND = IND + 1
      P(IND) = P(IND) - G * P(L)
  400 CONTINUE
  410 A(1) = WNEXT
      DO 420 L = 1, IR
  420 P(L) = 0.0
      RESID(I) = UT / SQRT(FT)
      E(INDE) = RESID(I)
      INDE = INDE + 1
      IF (INDE .GT. IQ) INDE = 1
      SSQ = SSQ + UT * UT / FT
      SUMLOG = SUMLOG + ALOG(FT)
```

```
  500 CONTINUE
      NIT = N
      RETURN
C
C        QUICK RECURSIONS
C
  600 I = 1
  610 NIT = I - 1
      DO 650 II = I, N
      ET = W(II)
      INDW = II
      IF (IP .EQ. 0) GOTO 630
      DO 620 J = 1, IP
      INDW = INDW - 1
      IF (INDW .LT. 1) GOTO 630
      ET = ET - PHI(J) * W(INDW)
  620 CONTINUE
  630 IF (IQ .EQ. 0) GOTO 645
      DO 640 J = 1, IQ
      INDE = INDE - 1
      IF (INDE .EQ. 0) INDE = IQ
      ET = ET - THETA(J) * E(INDE)
  640 CONTINUE
  645 E(INDE) = ET
      RESID(II) = ET
      SSQ = SSQ + ET * ET
      INDE = INDE + 1
      IF (INDE .GT. IQ) INDE = 1
  650 CONTINUE
      RETURN
      END
C
      SUBROUTINE KALFOR(M, IP, IR, NP, PHI, A, P, V, WORK)
C
C        ALGORITHM AS 154.2  APPL. STATIST. (1980) VOL.29, NO.3
C
C        INVOKING THIS SUBROUTINE OBTAINS PREDICTIONS
C        OF A AND P, M STEPS AHEAD.
C
      DIMENSION PHI(IR), A(IR), P(NP), V(NP), WORK(IR)
      IR1 = IR - 1
      DO 300 L = 1, M
C
C        PREDICT A.
C
      A1 = A(1)
      IF (IR .EQ. 1) GOTO 110
      DO 100 I = 1, IR1
  100 A(I) = A(I + 1)
  110 A(IR) = 0.0
      IF (IP .EQ. 0) GOTO 200
      DO 120 J = 1, IP
  120 A(J) = A(J) + PHI(J) * A1
C
C        PREDICT P.
C
  200 DO 210 I = 1, IR
  210 WORK(I) = P(I)
      IND = 0
      IND1 = IR
      DT = P(1)
      DO 220 J = 1, IR
      PHIJ = PHI(J)
      PHIJDT = PHIJ * DT
      DO 220 I = J, IR
      IND = IND + 1
      PHII = PHI(I)
      P(IND) = V(IND) + PHII * PHIJDT
      IF (J .LT. IR) P(IND) = P(IND) + WORK(J + 1) * PHII
      IF (I .EQ. IR) GOTO 220
      IND1 = IND1 + 1
      P(IND) = P(IND) + WORK(I + 1) * PHIJ + P(IND1)
  220 CONTINUE
  300 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE INCLU2(NP, NRBAR, WEIGHT, XNEXT, XROW, YNEXT,
     *  D, RBAR, THETAB, SSQERR, RECRES, IRANK, IFAULT)
C
C       ALGORITHM AS 154.3  APPL. STATIST. (1980) VOL.29, NO.3
C
C       FORTRAN VERSION OF REVISED VERSION OF ALGORITHM AS 75.1
C       APPL. STATIST. (1974) VOL.23, NO. 3.
C       SEE REMARK AS R17 APPL. STATIST. (1976) VOL. 25, NO. 3.
C
      DIMENSION XNEXT(NP), XROW(NP), D(NP), RBAR(NRBAR), THETAB(NP)
C
C       INVOKING THIS SUBROUTINE UPDATES D, RBAR, THETAB, SSQERR
C       AND IRANK BY THE INCLUSION OF XNEXT AND YNEXT WITH A
C       SPECIFIED WEIGHT. THE VALUES OF XNEXT, YNEXT AND WEIGHT WILL
C       BE CONSERVED. THE CORRESPONDING VALUE OF RECRES IS CALCULATED.
C
      Y = YNEXT
      WT = WEIGHT
      DO 10 I = 1, NP
   10 XROW(I) = XNEXT(I)
      RECRES = 0.0
      IFAULT = 1
      IF (WT .LE. 0.0) RETURN
      IFAULT = 0
C
      ITHISR = 0
      DO 50 I = 1, NP
      IF (XROW(I) .NE. 0.0) GOTO 20
      ITHISR = ITHISR + NP - I
      GOTO 50
   20 XI = XROW(I)
      DI = D(I)
      DPI = DI + WT * XI * XI
      D(I) = DPI
      CBAR = DI / DPI
      SBAR = WT * XI / DPI
      WT = CBAR * WT
      IF (I .EQ. NP) GOTO 40
      I1 = I + 1
      DO 30 K = I1, NP
      ITHISR = ITHISR + 1
      XK = XROW(K)
      RBTHIS = RBAR(ITHISR)
      XROW(K) = XK - XI * RBTHIS
      RBAR(ITHISR) = CBAR * RBTHIS + SBAR * XK
   30 CONTINUE
   40 XK = Y
      Y = XK - XI * THETAB(I)
      THETAB(I) = CBAR * THETAB(I) + SBAR * XK
      IF (DI .EQ. 0.0) GOTO 100
   50 CONTINUE
      SSQERR = SSQERR + WT * Y * Y
      RECRES = Y * SQRT(WT)
      RETURN
  100 IRANK = IRANK + 1
      RETURN
      END
C
      SUBROUTINE REGRES(NP, NRBAR, RBAR, THETAB, BETA)
C
C       ALGORITHM AS 154.4  APPL. STATIST. (1980) VOL.29, NO.3
C
C       REVISED VERSION OF ALGORITHM AS 75.4
C       APPL. STATIST. (1974) VOL.23, NO.3
C       INVOKING THIS SUBROUTINE OBTAINS BETA BY BACKSUBSTITUTION
C       IN THE TRIANGULAR SYSTEM RBAR AND THETAB.
C
      DIMENSION RBAR(NRBAR), THETAB(NP), BETA(NP)
      ITHISR = NRBAR
      IM = NP
      DO 50 I = 1, NP
      BI = THETAB(IM)
      IF (IM .EQ. NP) GOTO 30
      I1 = I - 1
      JM = NP
      DO 10 J = 1, I1
      BI = BI - RBAR(ITHISR) * BETA(JM)
      ITHISR = ITHISR - 1
      JM = JM - 1
   10 CONTINUE
   30 BETA(IM) = BI
      IM = IM - 1
   50 CONTINUE
      RETURN
      END
```