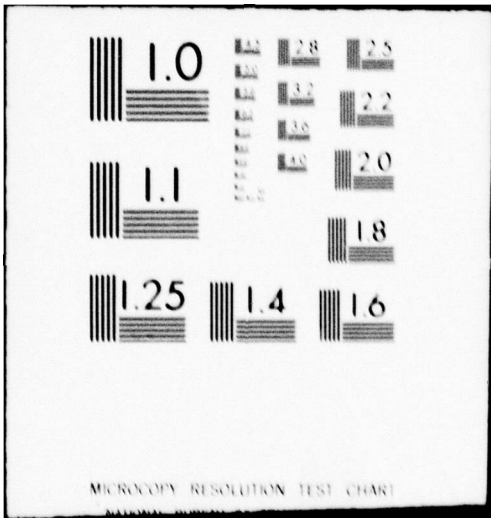
AD-A060 420 VICTORIA UNIV (BRITISH COLUMBIA) DEPT OF MATHEMATICS F/G 12/1
ALGORITHMS FOR LEAST-SQUARES LINEAR PREDICTION AND MAXIMUM ENTR--ETC(U)
AUG 78 I BARRODALE, R E ERICKSON

UNCLASSIFIED DM-142-IR NL

1 OF 1
AD A060 420

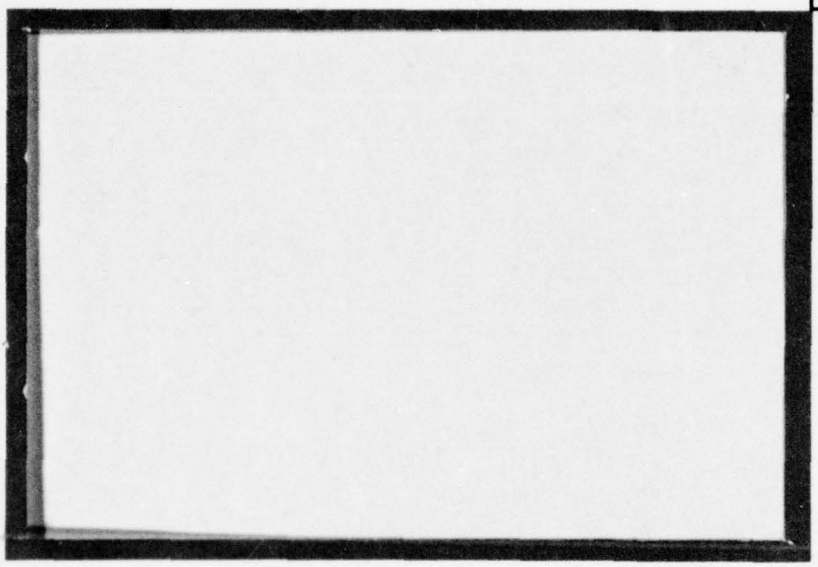
END
DATE FILMED
1-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

# UNIVERSITY OF VICTORIA

VICTORIA, B.C., CANADA









## DEPARTMENT OF MATHEMATICS

(3)



(6) ALGORITHMS FOR LEAST-SQUARES LINEAR PREDICTION AND MAXIMUM ENTROPY SPECTRAL ANALYSIS,

By

(10) I./Barrodale [redacted] R.E./Erickson

(Prepared for Chief, Defence Research Establishment Pacific, under D.N.D. contract #8SS77-08135)

(14) DM-142-IR

(11) AUG[redacted]78

(12) 53 p.

# ALGORITHMS FOR LEAST-SQUARES LINEAR PREDICTION AND MAXIMUM ENTROPY SPECTRAL ANALYSIS

I. Barrodale[1] and R.E. Erickson[2]

**Abstract**

Experience with the maximum entropy method of spectral analysis suggests that (a) it can produce inaccurate frequency estimates of short sample sinusoidal data, and (b) it sometimes produces calculated values for the filter coefficients that are unduly contaminated by rounding errors. Consequently, in this report we develop an algorithm for solving the underlying least-squares problem directly, without forcing a Toeplitz structure on the model. This approach leads to more accurate frequency determination for short sample harmonic processes, and our algorithm is computationally efficient and numerically stable. The algorithm can also be applied to two other versions of the linear prediction problem. A FORTRAN program is supplied.

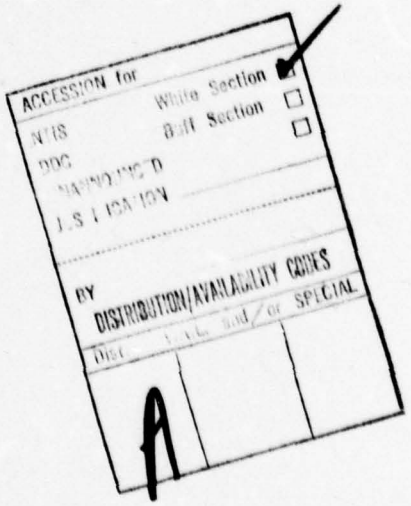
---

1. Mathematics Department, University of Victoria, Victoria, B.C.

2. Electromagnetics Section, Defence Research Establishment Pacific, Victoria, B.C.

CONTENTS





## 1. Introduction

The main purpose of this report is to describe an algorithm, and present a FORTRAN program, for solving three variants of the so-called linear prediction problem. Our algorithm calculates an "optimal" least-squares (LS) solution by solving a sequence of normal equations in a numerically stable manner, and it is efficient in both computer storage and time requirements. In particular, it can be used to determine the parameters of the autoregressive (AR) model associated with the maximum entropy method (MEM) of spectral analysis, and this application provided our primary motivation for developing the program given in Appendix E. However, the program should also prove to be useful in other applications, such as the design of adaptive signal whiteners, estimating the parameters of adaptive control systems, and time series modelling.

We also discuss some techniques for preserving accuracy in LS floating-point calculations which, although they are of fundamental importance in calculating satisfactory solutions, appear to have been neglected by many users. In addition, we briefly review some alternative methods to handle problems for which our algorithm is less efficient.

Currently, the most popular algorithm for determining the MEM parameters is due to Burg (1967, 1968, 1975), and a perusal of the recent literature in geophysics alone confirms that MEM spectral estimates obtained with this algorithm have gained wide acceptance. The computational efficiency of this algorithm stems from its use of the Levinson (1947) recursive properties of solutions to sequences of certain linear algebraic equations with coefficient matrices of symmetric Toeplitz

form. (A Toeplitz matrix has all its elements equal along each diagonal, i.e. $c_{i,j} = c_{i-j}$.)

However, when processing real data (e.g. geomagnetic micropulsation recordings) we have experienced two shortcomings inherent in Burg's algorithm. The first of these is that it can produce inaccurate frequency estimates of short sample sinusoidal data. This limitation appears to have been first reported by Chen and Stegen (1974). Erickson (1976), Ulrych and Clayton (1976), and Barber and Taylor (1977) provide empirical evidence which suggests that LS solutions to the AR model for MEM lead to more accurate frequency determination in these cases. Burg's algorithm forces a Toeplitz structure on the matrix of the system of equations which yields the AR parameters, and the resulting spectral estimates for short sample harmonic processes are usually inferior to the spectral estimates resulting from our LS algorithm.

The second shortcoming of Burg's algorithm is that it sometimes produces calculated values for the parameters that are unduly contaminated by rounding errors. This weakness is a consequence of its use of Levinson's recursive scheme, which has previously been reported to be numerically unstable in some circumstances by Box and Jenkins (1976). In contrast, our algorithm employs a numerically stable technique to generate successive systems of normal equations. The corresponding parameters are determined by Cholesky's method, which is known to possess remarkable numerical stability (e.g. see Martin, Peters, and Wilkinson (1971)).

Estimation of AR parameters by LS methods has previously been unpopular in MEM applications because (a) it involves more computational effort than Burg's algorithm, (b) the parameters may give rise to a filter which is

not minimum phase, (c) the calculated values for the parameters may "blow up", and (d) the calculated value for the residual sum of squares is often inaccurate (e.g. it may be negative!).

Although our algorithm certainly involves more arithmetic operations than Burg's algorithm, it is an efficient LS method for estimating reasonable numbers of parameters; later we refer to some alternative LS methods for problems involving larger numbers of parameters. Secondly, if the minimum phase condition is required in a given appli ıtion (for example, see Claerbout (1976)), a sensible procedure would be to reflect any poles resulting from the LS method back inside the unit circle. (Atal and Hanauer (1971) discuss such a technique, which preserves the amplitude response of the filter.) Finally, the problems reported in (c) and (d) above have been virtually eliminated in our algorithm, through careful attention being paid to the computations involved.

## 2. Statement of the problems

Given a time series $x_1, x_2, \ldots, x_n$, let us assume that $x_t$ can be estimated by $\hat{x}_t$, where

$$\hat{x}_t = \sum_{j=1}^{m} \hat{a}_j \, x_{t-j} , \quad \text{for } t = m+1, m+2, \ldots, n . \tag{1}$$

For given values of the parameters (or filter coefficients) $\hat{a}_j$, the nonrecursive digital filter (1) provides a <u>forward</u> prediction of $x_t$. Putting $\hat{e}_t = x_t - \hat{x}_t$ and adopting the LS criterion, the parameters $\hat{a}_j$ are determined by minimizing the residual sum of squares $\sum_{t=m+1}^{n} \hat{e}_t^2$.

(Equivalently, a LS solution is sought for the AR scheme of order $m$, defined by

$$x_t = \sum_{j=1}^{m} \hat{a}_j x_{t-j} + \hat{e}_t , \quad \text{for } t = m+1, m+2, \ldots, n .)$$

Conversely, suppose that $x_t$ can be estimated by $\bar{x}_t$, where

$$\bar{x}_t = \sum_{j=1}^{m} \bar{a}_j x_{t+j} , \quad \text{for } t = 1, 2, \ldots, n-m . \tag{2}$$

The filter (2) provides a backward prediction of $x_t$, and its parameters $\bar{a}_j$ can be determined by minimizing $\sum_{t=1}^{n-m} \bar{e}_t^2$, where $\bar{e}_t = x_t - \bar{x}_t$.

It is convenient to examine these problems in the context of solving systems of linear algebraic equations. Thus, the forward prediction problem (1) is described by the $(n-m) \times m$ matrix equation

$$\hat{\underset{\sim}{X}} \hat{\underset{\sim}{a}} = \hat{\underset{\sim}{y}} , \tag{3}$$

where $\hat{\underset{\sim}{X}} = \begin{bmatrix} x_m & x_{m-1} & \cdots & x_1 \\ x_{m+1} & x_m & \cdots & x_2 \\ \vdots & \vdots & & \vdots \\ x_{n-1} & x_{n-2} & \cdots & x_{n-m} \end{bmatrix}$, $\hat{\underset{\sim}{a}} = \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_m \end{bmatrix}$, and $\hat{\underset{\sim}{y}} = \begin{bmatrix} x_{m+1} \\ x_{m+2} \\ \vdots \\ x_n \end{bmatrix}$.

If $n > 2m$ this system is overdetermined and no $\hat{\underset{\sim}{a}}$ generally exists which satisfies (3), so it is useful to define a residual vector $\hat{\underset{\sim}{e}} = \hat{\underset{\sim}{y}} - \hat{\underset{\sim}{X}} \hat{\underset{\sim}{a}}$. A LS solution $\hat{\underset{\sim}{a}}_*$ to (3) is then defined as any $\hat{\underset{\sim}{a}}$ which minimizes the residual sum of squares $S = S(\hat{\underset{\sim}{a}}) = \hat{\underset{\sim}{e}}^T \hat{\underset{\sim}{e}}$, where the superscript $T$ denotes the transpose operation.

It is well known that a LS solution exists for any overdetermined

system of equations, and it is unique if the matrix for the system is of full rank ($= m$, for problem (3), assuming that $n > 2m$). The minimum value of the residual sum of squares $S$ is achieved when all its first partial derivatives are zero (i.e. $\partial S/\partial \hat{a}_j = 0$ in the case of (3)). Applying this condition to (3), it follows that $S$ is minimized when $\hat{\underset{\sim}{X}}^T \hat{\underset{\sim}{e}} = \underset{\sim}{0}$, and so $\hat{\underset{\sim}{a}}_\star$ must satisfy the equation

$$\hat{\underset{\sim}{X}}^T (\hat{\underset{\sim}{y}} - \hat{\underset{\sim}{X}}\, \hat{\underset{\sim}{a}}) = \underset{\sim}{0} . \tag{4}$$

The LS solution to (3) is therefore characterized by the $m \times m$ system of normal equations[†]

$$\hat{\underset{\sim}{X}}^T \hat{\underset{\sim}{X}}\, \hat{\underset{\sim}{a}}_\star = \hat{\underset{\sim}{X}}^T \hat{\underset{\sim}{y}} . \tag{5}$$

For the remainder of this paper we adopt a more concise notation and refer to $\hat{\underset{\sim}{a}}_\star$ as the solution to an $m \times m$ system of equations

$$\hat{\underset{\sim}{R}}\, \hat{\underset{\sim}{a}}_\star = \hat{\underset{\sim}{s}} , \tag{6}$$

where $\hat{\underset{\sim}{R}} = [\hat{r}_{i,j}] = \hat{\underset{\sim}{X}}^T \hat{\underset{\sim}{X}}$ and $\hat{\underset{\sim}{s}} = [\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_m]^T = \hat{\underset{\sim}{X}}^T \hat{\underset{\sim}{y}}$.

Similarly, the backward prediction problem (2) is described by the $(n-m) \times m$ matrix equation

$$\bar{\underset{\sim}{X}}\, \bar{\underset{\sim}{a}} = \bar{\underset{\sim}{y}} , \tag{7}$$

where $\bar{\underset{\sim}{X}} = \begin{bmatrix} x_2 & x_3 & \cdots & x_{m+1} \\ x_3 & x_4 & \cdots & x_{m+2} \\ \vdots & \vdots & & \vdots \\ x_{n-m+1} & x_{n-m+2} & \cdots & x_n \end{bmatrix}$, $\bar{\underset{\sim}{a}} = \begin{bmatrix} \bar{a}_1 \\ \bar{a}_2 \\ \vdots \\ \bar{a}_m \end{bmatrix}$, and $\bar{\underset{\sim}{y}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-m} \end{bmatrix}$.

---

[†]From (4) we see that the solution $\hat{\underset{\sim}{a}}_\star$ yields a residual vector $\hat{\underset{\sim}{e}}_\star$ that is <u>normal</u> to the column space of $\hat{\underset{\sim}{X}}$; hence the terminology.

The LS solution $\hat{\underset{\sim}{a}}_*$ to (7) is characterized by the $m \times m$ system of normal equations

$$\bar{\underset{\sim}{R}}\,\bar{\underset{\sim}{a}}_* = \bar{\underset{\sim}{s}}\,, \tag{8}$$

where $\bar{\underset{\sim}{R}} = [\bar{r}_{i,j}] = \bar{\underset{\sim}{X}}^T\bar{\underset{\sim}{X}}$ and $\bar{\underset{\sim}{s}} = [\bar{s}_1,\bar{s}_2,\ldots,\bar{s}_m]^T = \bar{\underset{\sim}{X}}^T\bar{\underset{\sim}{y}}$.

Now in Burg's algorithm for MEM it is assumed that $x_t$ can be estimated by a weighted sum of $m$ previous observations and a weighted sum of $m$ future observations, using the same weights $a_j$ in both directions. Hence the MEM problem is described by the $2(n-m) \times m$ matrix equation

$$\underset{\sim}{X}\,\underset{\sim}{a} = \underset{\sim}{y}\,, \tag{9}$$

where $\underset{\sim}{X} = \begin{bmatrix} \hat{\underset{\sim}{X}} \\ \bar{\underset{\sim}{X}} \end{bmatrix}$, $\underset{\sim}{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$, and $\underset{\sim}{y} = \begin{bmatrix} \hat{\underset{\sim}{y}} \\ \bar{\underset{\sim}{y}} \end{bmatrix}$.

The LS solution $\underset{\sim}{a}_*$ to (9) satisfies the $m \times m$ system of normal equations

$$\underset{\sim}{R}\,\underset{\sim}{a}_* = \underset{\sim}{s}\,, \tag{10}$$

where $\underset{\sim}{R} = [r_{i,j}] = \underset{\sim}{X}^T\underset{\sim}{X}$ and $\underset{\sim}{s} = [s_1,s_2,\ldots,s_m]^T = \underset{\sim}{X}^T\underset{\sim}{y}$. (If $2n > 3m$ the system (9) is overdetermined. Notice also that we distinguish between the forward, backward, and forward and backward prediction problems by means of a "$\hat{}$" superscript, "$-$" superscript, or no superscript, respectively.)

In view of the structure of $\underset{\sim}{X}$ and $\underset{\sim}{y}$, (10) can be rewritten as

$$(\hat{\underset{\sim}{R}} + \bar{\underset{\sim}{R}})\underset{\sim}{a}_* = \hat{\underset{\sim}{s}} + \bar{\underset{\sim}{s}} , \tag{11}$$

and comparing (6), (8) and (11) it follows that the MEM parameter vector $\underset{\sim}{a}_*$ is not related in any simple fashion to $\hat{\underset{\sim}{a}}_*$ and $\bar{\underset{\sim}{a}}_*$. Thus, the normal equations (6), (8) and (10) pertain to three distinct AR schemes of order $m$, and our FORTRAN program can be directed to compute any one of the corresponding LS solutions $\hat{\underset{\sim}{a}}_*$, $\bar{\underset{\sim}{a}}_*$ or $\underset{\sim}{a}_*$. These are the three variants of the linear prediction problem referred to in the first paragraph of §1.

## 3. The algorithm

Let us begin this section by summarizing very briefly the current state-of-the-art concerning algorithms for linear LS computations. (The book by Lawson and Hanson (1974) provides a very detailed coverage of this topic, complete with FORTRAN programs).

For an overdetermined system of $N$ equations in $M$ unknown parameters, forming the normal equations requires $NM^2/2$ operations[†] and solving them (by Cholesky's method) requires $M^3/6$ operations. However, numerical analysts generally prefer to avoid this approach, since forming the normal equations considerably worsens the numerical condition of any LS problem for which the corresponding overdetermined system of equations is itself ill-conditioned.

---

[†]An operation is a multiplication or division plus an addition or subtraction, and when comparing operation counts only the highest powers of $N$ and $M$ are given.

It is usually advocated instead that the LS solution be computed directly from the $N \times M$ overdetermined system by an orthogonalization method. Some popular algorithms of this type are the modified Gram-Schmidt process ($NM^2$ operations), Householder triangularization ($NM^2 - M^3/3$ operations), and singular value decomposition (at least $NM^2 + 5M^3$ operations). In a given precision floating-point arithmetic these orthogonalization methods successfully process a wider class of LS problems than the normal equations algorithm. Indeed, whenever single precision arithmetic is just adequate for computing the parameters in a particular LS problem by (say) Householder's method, the normal equations must typically be formed and solved in double precision arithmetic to calculate these parameters to comparable accuracy.

Nevertheless, when $N >> M$ (which is often the case in practice) the normal equations method involves only about half as many operations as an orthogonalization method. In addition, if the normal equations can be formed directly from the data rather than from an overdetermined system, then the normal equations algorithm requires only about $M(M+1)/2$ storage locations. Since orthogonalization methods require more than $NM$ storage locations (although this can be reduced by more sophisticated programming and some extra computation), the normal equations method can offer worthwhile savings in both computer storage and time requirements. (The savings are less significant when ill-conditioning dictates that the normal equations be formed and solved in a higher precision arithmetic than is required by an orthogonalization method.)

For the applications in which we are primarily interested it is frequently the case that $N > 10M$, and sometimes $N > 100M$. In the

notation of §2, $N = n-m$ and $M = m$ for (3) and (7), whereas $N = 2(n-m)$ and $M = m$ in the MEM model (9). Furthermore, by taking advantage of the special structure of (6), (8) or (10), the normal equations for any one of these AR schemes can be obtained directly from the given time series in $[n(m+1) + 0(m^3)]$ operations. We have therefore adopted the normal equations approach for solving the LS problems of this report.

In our discussion so far it has been tacitly assumed that the number of parameters $m$ is known a priori, although in practice this may not be true. Indeed, choosing an appropriate order for an AR scheme is one of the most difficult tasks in time series modelling. Our algorithm allows this choice to be made dynamically in a computationally efficient manner, as is explained later in this section.

Before stating the general algorithm, we first show for the normal equations (6) how $\hat{\underset{\sim}{R}}_3$ and $\hat{\underset{\sim}{s}}_3$ (corresponding to $m = 3$) can be obtained from $\hat{\underset{\sim}{R}}_2$ and $\hat{\underset{\sim}{s}}_2$ (corresponding to $m = 2$). Suppressing the elements above the diagonal (since any normal equations matrix is symmetric) we have

$$\hat{\underset{\sim}{R}}_2 = \begin{bmatrix} \hat{r}^{(2)}_{1,1} & \cdot \\ \hat{r}^{(2)}_{2,1} & \hat{r}^{(2)}_{2,2} \end{bmatrix} = \begin{bmatrix} \sum_{t=1}^{n-2} x_{t+1}\, x_{t+1} & \cdot \\ \sum_{t=1}^{n-2} x_t\, x_{t+1} & \sum_{t=1}^{n-2} x_t\, x_t \end{bmatrix},$$

$$\hat{\underset{\sim}{s}}_2 = \begin{bmatrix} \hat{s}^{(2)}_1 , \hat{s}^{(2)}_2 \end{bmatrix}^T = \begin{bmatrix} \sum_{t=1}^{n-2} x_{t+1}\, x_{t+2} , \sum_{t=1}^{n-2} x_t\, x_{t+2} \end{bmatrix}^T , \quad \text{and}$$

$$\hat{\underset{\sim}{R}}_3 = \begin{bmatrix} \hat{r}^{(3)}_{1,1} & \cdot & \cdot \\ \hat{r}^{(3)}_{2,1} & \hat{r}^{(3)}_{2,2} & \cdot \\ \hat{r}^{(3)}_{3,1} & \hat{r}^{(3)}_{3,2} & \hat{r}^{(3)}_{3,3} \end{bmatrix} = \begin{bmatrix} \sum_{t=1}^{n-3} x_{t+2}\, x_{t+2} & \cdot & \cdot \\ \sum_{t=1}^{n-3} x_{t+1}\, x_{t+2} & \sum_{t=1}^{n-3} x_{t+1}\, x_{t+1} & \cdot \\ \sum_{t=1}^{n-3} x_t\, x_{t+2} & \sum_{t=1}^{n-3} x_t\, x_{t+1} & \sum_{t=1}^{n-3} x_t\, x_t \end{bmatrix},$$

$$\hat{\underset{\sim}{s}}_3 = \left[ \hat{s}^{(3)}_1, \hat{s}^{(3)}_2, \hat{s}^{(3)}_3 \right]^T = \left[ \sum_{t=1}^{n-3} x_{t+2}\, x_{t+3}, \sum_{t=1}^{n-3} x_{t+1}\, x_{t+3}, \sum_{t=1}^{n-3} x_t\, x_{t+3} \right]^T .$$

Notice that the leading $2 \times 2$ submatrix of $\hat{\underset{\sim}{R}}_3$ can be obtained from $\hat{\underset{\sim}{R}}_2$ by subtracting the first term in each summation (or inner-product) which defines an element of $\hat{\underset{\sim}{R}}_2$, i.e.

$$\hat{r}^{(3)}_{i,j} = \hat{r}^{(2)}_{i,j} - x_{3-i}\, x_{3-j}, \quad \text{for } 1 \le j \le i \le 2 .$$

(Equivalently, the leading $2 \times 2$ submatrix of $\hat{\underset{\sim}{R}}_3$ can be expressed as $\hat{\underset{\sim}{R}}_2 - \hat{\underset{\sim}{\Delta}}_2$, where the matrix $\hat{\underset{\sim}{\Delta}}_2 = [x_2\ x_1]^T [x_2\ x_1]$ represents a <u>rank one modification</u> to $\hat{\underset{\sim}{R}}_2$.) Also, the last row of $\hat{\underset{\sim}{R}}_3$ (apart from its first element) can be obtained from the last row of $\hat{\underset{\sim}{R}}_2$ as follows,

$$\hat{r}^{(3)}_{3,j} = \hat{r}^{(2)}_{2,j-1} - x_{n-2}\, x_{n+1-j}, \quad \text{for } 2 \le j \le 3 ,$$

and

$$\hat{r}^{(3)}_{3,1} = \hat{s}^{(2)}_2 - x_{n-2}\, x_n .$$

Finally, $\hat{\underset{\sim}{s}}_3$ (apart from its last element) can be obtained from $\hat{\underset{\sim}{s}}_2$ as follows,

$$\hat{s}_i^{(3)} = \hat{s}_i^{(2)} - x_{3-i}\, x_3 , \qquad \text{for } 1 \le i \le 2 ,$$

and

$$\hat{s}_3^{(3)} = \sum_{t=1}^{n-3} x_t\, x_{t+3} .$$

Thus only <u>one</u> inner-product (the element $\hat{s}_3^{(3)}$) has to be calculated afresh when we derive the normal equations for $m = 3$ from the normal equations for $m = 2$ ; each of the remaining elements of $\hat{\underset{\sim}{R}}_3$ and $\hat{\underset{\sim}{s}}_3$ is obtained at the cost of one operation.

This scheme generalizes so that $\hat{\underset{\sim}{R}}_{m+1}$ and $\hat{\underset{\sim}{s}}_{m+1}$ can be obtained from $\hat{\underset{\sim}{R}}_m$ and $\hat{\underset{\sim}{s}}_m$ , with only the element $\hat{s}_{m+1}^{(m+1)}$ requiring more than one operation. When a backward prediction is needed, a similar scheme is available to generate $\bar{\underset{\sim}{R}}_{m+1}$ and $\bar{\underset{\sim}{s}}_{m+1}$ from $\bar{\underset{\sim}{R}}_m$ and $\bar{\underset{\sim}{s}}_m$ . Furthermore, the one element $\bar{s}_{m+1}^{(m+1)}$ which must be evaluated as an inner-product is equal to $\hat{s}_{m+1}^{(m+1)}$ , and so even when the relationships (1) and (2) hold simultaneously only one inner-product is calculated when $m$ increases to $m+1$ .

Let us now state the general algorithm which allows any one of the three $(m+1) \times (m+1)$ systems of normal equations to be generated efficiently from the corresponding $m \times m$ system of normal equations. The algorithm is arranged so as to avoid the use of intermediate storage, and since the matrices involved are symmetric the above-diagonal elements are never required.

Algorithm F: Forward prediction problem (6).

$$\hat{r}_{m+1,1} \leftarrow \hat{s}_m - x_{n-m}\, x_n$$

for $j = 2,\ldots,m+1$ do

$$\hat{r}_{m+1,j} \leftarrow \hat{r}_{m,j-1} - x_{n-m}\, x_{n+1-j}$$

for $i = 1,\ldots,m$ do

for $j = 1,\ldots,i$ do

$$\hat{r}_{i,j} \leftarrow \hat{r}_{i,j} - x_{m+1-i}\, x_{m+1-j}$$

for $i = 1,\ldots,m$ do

$$\hat{s}_i \leftarrow \hat{s}_i - x_{m+1-i}\, x_{m+1}$$

$$\hat{s}_{m+1} \leftarrow \sum_{t=1}^{n-m-1} x_t\, x_{m+1+t}$$

Algorithm B: Backward prediction problem (8).

$$\bar{r}_{m+1,1} \leftarrow \bar{s}_m - x_1\, x_{m+1}$$

for $j = 2,\ldots,m+1$ do

$$\bar{r}_{m+1,j} \leftarrow \bar{r}_{m,j-1} - x_j\, x_{m+1}$$

for $i = 1,\ldots,m$ do

for $j = 1,\ldots,i$ do

$$\bar{r}_{i,j} \leftarrow \bar{r}_{i,j} - x_{n-m+i}\, x_{n-m+j}$$

for $i = 1,\ldots,m$ do

$$\bar{s}_i \leftarrow \bar{s}_i - x_{n-m}\, x_{n-m+i}$$

$$\bar{s}_{m+1} \leftarrow \sum_{t=1}^{n-m-1} x_t\, x_{m+1+t}$$

Algorithm FAB: Forward and Backward prediction problem (10).

$$r_{m+1,1} \leftarrow s_m - x_{n-m}\, x_n - x_1\, x_{m+1}$$

for $j = 2,\ldots,m+1$ do

$$r_{m+1,j} \leftarrow r_{m,j-1} - x_{n-m}\, x_{n+1-j} - x_j\, x_{m+1}$$

for $i = 1,\ldots,m$ do

for $j = 1,\ldots,i$ do

$$r_{i,j} \leftarrow r_{i,j} - x_{m+1-i}\, x_{m+1-j} - x_{n-m+i}\, x_{n-m+j}$$

for $i = 1,\ldots,m$ do

$$s_i \leftarrow s_i - x_{m+1-i}\, x_{m+1} - x_{n-m}\, x_{n-m+i}$$

$$s_{m+1} \leftarrow 2 \times \sum_{t=1}^{n-m-1} x_t\, x_{m+1+t}$$

All three algorithms produce the normal equations of order $m+1$ by first forming the $(m+1)$th row, then overwriting the coefficient matrix and right-hand side vector of order $m$, and then calculating the required inner-product. For Algorithm F or Algorithm B this step involves $(m^2+5m+2)/2$ operations plus $n-m-1$ operations for the inner-product, while Algorithm FAB requires $m^2+5m+2$ operations plus $n-m-1$ operations for the inner-product.

Starting at $m=1$, and incrementing the value of $m$ by 1 at each step, we can form $\hat{\underset{\sim}{R}}$, $\bar{\underset{\sim}{R}}$, or $\underset{\sim}{R}$ together with $\hat{\underset{\sim}{s}}$, $\bar{\underset{\sim}{s}}$, or $\underset{\sim}{s}$ for any desired value of $m$. When $m=1$ there are two inner-product calculations required†, and so the total number of operations involved

† e.g. in problem (6) we calculate $\hat{r}^{(1)}_{1,1} = \sum_{t=1}^{n-1} x_t x_t$ and $\hat{s}^{(1)}_{1} = \sum_{t=1}^{n-1} x_t x_{1+t}$.

in forming the normal equations of all orders from 1 through m is $[n(m+1) + \frac{m^3}{6} + \frac{m^2}{2} - \frac{2m}{3} - 2]$ for Algorithm F or Algorithm B, and $[n(m+1) + \frac{m^3}{3} + \frac{3m^2}{2} - \frac{5m}{6} - 3]$ for Algorithm FAB. (As a comparison, if the normal equations were all derived independently with inner-products calculated from first principles, the operation count in forming problem (6) for orders 1 through m would be $n[\frac{m^3}{6} + m^2 + \frac{5m}{6}] - [\frac{m^4}{8} + \frac{3m^3}{4} + \frac{7m^2}{8} + \frac{m}{4}]$ ; when $n = 100$ and $m = 10$ this approach involves 25,410 operations, whereas Algorithm F requires only 1,308 operations.)

Assuming that an appropriate value for m is available, the relevant normal equations of order m can be formed by one of the Algorithms F, B, or FAB, and then solved by Cholesky's method ($m^3/6$ operations). However, let us suppose that the user is not sure what value for m is appropriate, but that he can specify values $m_{start}$ and $m_{last}$ such that $m_{start} \leq m \leq m_{last}$. In this event a sensible algorithm would not <u>solve</u> the normal equations until the value of m has been increased to $m_{start}$, even though the normal equations are <u>formed</u> for $m = 1,2,\ldots,m_{start},\ldots$. Our algorithm employs this strategy, and so if the user knows the "correct" value for m beforehand he simply sets $m_{start} = m_{last} = m$. However, typical applications of our algorithm call for a sequence of normal equations to be solved, and hence beginning at $m = m_{start}$ we generate a duplicate copy of the current normal equations since these are subsequently destroyed by the Cholesky factorization.

The problem of choosing an appropriate value for the number of parameters m depends upon the particular application involved. (The choice

might also depend on the amount of inherent error in the time series, and on the accuracy of the computing environment.) For MEM applications we often use a criterion due to Akaike to determine the order $m$ of the AR scheme; see Appendix A for a description of this criterion. The program given in Appendix E includes a subroutine which implements Akaike's scheme, and so users who wish to determine $m$ differently can simply replace this subroutine with one of their own choosing.

Most of our digital signal processing applications involve the use of minicomputers which operate in fixed precision floating-point arithmetic. Hence, the single precision program in Appendix E is designed to avoid any dependence on higher precision program segments. Consequently it is imperative to take extra care when performing any floating-point calculations where serious loss of significance can occur. Computational experience with our algorithm confirms that significance losses can be troublesome when calculating inner-products (particularly when $n$ is large), and also in computing the residual sum of squares. These two computations are discussed in Appendices C and D respectively, and we provide accurate subroutines for both calculations in Appendix E.

Finally, the computational efficiency of our algorithm has been optimized in its FORTRAN implementation by avoiding the use of two-dimensional arrays. Hence Appendix B contains an alternative description of Algorithm FAB in vector notation.

## 4. Further details and alternative algorithms

The algorithm described in §3 generates each successive system of normal equations from the preceding system, but each set of equations

is solved independently (i.e. without reference to the Cholesky solution of previous systems). Now Cholesky's method requires about $m^3/6 + 3m^2/2 + 0(m)$[†] operations to solve any $m \times m$ symmetric positive definite[††] system of linear equations. Hence, if the normal equations are solved independently for all orders from 1 through $m$, Cholesky's method uses a total of about $m^4/24 + 7m^3/12 + 0(m^2)$ operations. Nevertheless, in our typical applications the work required by Algorithm F to form all the normal equations is still more than the work of solving them; for example, this is the case when $n \geq m^2$ and $m \leq 15$.

In this section we refer briefly to some algorithms which do take advantage of previous solutions. We restrict our remarks primarily to the forward prediction problem (6), but analogous alternative methods exist for the backward prediction problem (8) and for the MEM problem (10). Although the method of §3 is usually more efficient than these alternative schemes when $m \leq 15$, they become more attractive for larger values of $m$. However, they also require more programming effort than is involved in Appendix E, particularly when appropriate techniques to preserve numerical accuracy are incorporated.

Let us begin by restating the relationship that exists between $\hat{\underset{\sim}{R}}_{m+1} = \left[\hat{r}^{(m+1)}_{i,j}\right] = \left[\sum_{t=1}^{n-m-1} x_{m+1-i+t}\, x_{m+1-j+t}\right]$ and $\hat{\underset{\sim}{R}}_{m} = \left[\hat{r}^{(m)}_{i,j}\right] =$

---

[†]The $0(m)$ term in this operation count includes $m$ square root reciprocals, although an alternative version exists in which square roots are avoided.

[††]Any normal equations matrix is positive definite provided that the corresponding overdetermined system of equations is of full rank.

$\left[\sum_{t=1}^{n-m} x_{m-i+t}\, x_{m-j+t}\right]$. Letting $\hat{\Delta}_{m}$ be an $m \times m$ matrix with elements $\hat{\delta}^{(m)}_{i,j}$, and adopting the more concise notation $\rho_{m} = \left[\hat{r}^{(m+1)}_{m+1,1}, \hat{r}^{(m+1)}_{m+1,2}, \ldots, \hat{r}^{(m+1)}_{m+1,m}\right]^{T}$ and $\rho_{m+1} = \hat{r}^{(m+1)}_{m+1,m+1}$, we see that Algorithm F provides a constructive proof of the following result.

Theorem 1. For the forward prediction problem (6)

$$\hat{R}_{m+1} = \left[\begin{array}{c|c} \hat{R}_{m} - \hat{\Delta}_{m} & \rho_{m} \\ \hline \rho_{m}^{T} & \rho_{m+1} \end{array}\right], \quad \text{where } \hat{\delta}^{(m)}_{i,j} = x_{m+1-i}\, x_{m+1-j}.$$

Theorem 1 shows that $\hat{R}_{m+1}$ (apart from its extra row and column) can be obtained by subtracting a rank one matrix $\hat{\Delta}_{m}$ from $\hat{R}_{m}$, where

$$\hat{\Delta}_{m} = [x_{m}, x_{m-1}, \ldots, x_{1}]^{T}\, [x_{m}, x_{m-1}, \ldots, x_{1}].$$

When solving the $m \times m$ normal equations (6) by Cholesky's method, the first step $\left(0(m^{3}) \text{ operations}\right)$ is to obtain a factorization $\hat{L}_{m} \hat{L}_{m}^{T}$ of $\hat{R}_{m}$, where the matrix $\hat{L}_{m}$ is lower triangular. The second step $\left(0(m^{2}) \text{ operations}\right)$ obtains the solution to $\hat{L}_{m} \hat{L}_{m}^{T} \hat{a}_{m} = \hat{s}_{m}$, by solving $\hat{L}_{m} \hat{b}_{m} = \hat{s}_{m}$ and then $\hat{L}_{m}^{T} \hat{a}_{m} = \hat{b}_{m}$. Thus, <u>whenever problem (6) is to be solved for many successive values of $m$ it is worthwhile improving the efficiency of the first step</u>.

In view of Theorem 1, this may be accomplished by obtaining the Cholesky factorization $\hat{L}_{m+1} \hat{L}_{m+1}^{T}$ of $\hat{R}_{m+1}$ from the previous factorization $\hat{L}_{m} \hat{L}_{m}^{T}$ of $\hat{R}_{m}$. Methods for updating the factors of a symmetric positive definite matrix following a rank one modification are described by Gill, Golub, Murray and Saunders (1974), who also explain how these methods can

be used to process the extra row and column (exhibited in Theorem 1) of $\hat{R}_{m+1}$. (This special case relies on the fact that the rectangular matrix $\hat{X}_{m+1}$ can be obtained from $\hat{X}_m$ by deleting one of its rows and adding a new column.) When modifying Cholesky factorizations the problem of maintaining positive definiteness, in the presence of rounding error, must be handled carefully (see also Fletcher and Powell (1974)), but numerically stable methods are available. The most efficient of these require $cm^2 + 0(m)$ operations to obtain the factors of $\hat{R}_{m+1}$ from those of $\hat{R}_m$, where $3 < c < 5$. (In contrast, it only requires $m^2 + 0(m)$ operations to obtain the factors of $\hat{R}_m$ from those of $\hat{R}_{m+1}$. In our applications however, it is more convenient to increase $m$ than to decrease it.)

For the MEM problem (10) the matrix $R_{m+1}$ (apart from its extra row and column) can be obtained by subtracting a rank two matrix $\Delta_m$ from $R_m$. The matrix $\Delta_m$ is the sum of $\hat{\Delta}_m$ and its counterpart $\bar{\Delta}_m$ from Algorithm B, which has the form

$$\bar{\Delta}_m = [x_{n-m+1}, x_{n-m+2}, \ldots, x_n]^T [x_{n-m+1}, x_{n-m+2}, \ldots, x_n] .$$

The factors of $R_{m+1}$ can be obtained from the factors of $R_m$ either by applying one of the above rank one methods twice, or by using a rank two method (see Bennett (1965) for example).

Thus, solving the normal equations (6), (8), or (10) for all orders from 1 through $m$ by methods involving modifications to Cholesky factorizations requires $(c'+1)m^3/3 + 0(m^2)$ operations, where $c' = c$ for (6) or (8) and $c' = 2c$ for (10). For moderate values of $m$ these methods therefore offer no gain in efficiency for the algorithm of §3.

However, in a very recent paper by Morf, Dickinson, Kailath and Vieira (1977) this operation count has been reduced to $7m^2 + 15m - 7$ for problems (6) or (8): they do not consider the MEM problem (10). This order-of-magnitude improvement in the effort required to solve all $m$ normal equations is made possible because the matrix

$$\hat{\underset{\sim}{R}} = [\hat{\underset{\sim}{y}} \mid \hat{\underset{\sim}{x}}]^T [\hat{\underset{\sim}{y}} \mid \hat{\underset{\sim}{x}}]$$

is "close" to Toeplitz form, in a sense made precise by Friedlander, Morf, Kailath and Ljung (1977). Their algorithm also takes advantage of the product-form structure of $\hat{\underset{\sim}{R}}$, since it is the fact that $[\hat{\underset{\sim}{y}} \mid \hat{\underset{\sim}{x}}]$ is a rectangular Toeplitz matrix which permits the equations to be solved so efficiently by an extension of Levinson's recursive scheme.

In view of the fact that Levinson's original algorithm for (square) Toeplitz matrices is numerically unstable in some circumstances, it seems likely that this extended algorithm also suffers from occasional rounding error difficulties. Indeed, because of their high speed and low storage requirements, it would clearly be advantageous to develop numerically stable versions of both the original and extended Levinson algorithms. (Hopefully, this could be accomplished while retaining the $0(m^2)$ operation count.)

Finally, although the above summary of alternative algorithms gives some prominence to the number of operations involved in solving all $m$ normal equations, the additional $n(m+1)$ operations required by <u>all</u> the methods often dominates the total operation count. Thus, when $n >> m$, the gains in efficiency offered by these alternative methods may be more apparent than real.

## 5. The maximum entropy spectrum

Our primary motivation for developing the program in Appendix E was to determine the AR parameters for MEM spectral analysis. Hence, for the sake of completeness, in this section we provide a brief heuristic description of the spectrum calculation, followed by a numerical example which compares spectral estimates obtained by our algorithm and by Burg's method. (The theory of MEM and its relationship to AR processes has been described in numerous other reports; in particular, see Ulrych and Bishop (1975).)

Given a time series $x_1, x_2, \ldots, x_n$, for which $\Delta t$ is the uniform sampling interval, the first step is to apply Algorithm FAB to obtain the LS solution $\underset{\sim}{a}_* = [a_1^*, a_2^*, \ldots, a_m^*]^T$ and its associated minimum residual sum of squares $S_m = S_m(\underset{\sim}{a}_*)$. This yields a recursive predictive filter for the signal, whose mean square residual $P_m$ is given by[†]

$$P_m = \frac{1}{2(n-m)} S_m \,. \tag{12}$$

If the fit to the time series is good, the residuals $e_t$ should have small correlation, resulting essentially in a white noise process.

When the MEM parameters $a_j^*$ are used for forward prediction, the residual $e_t$ is defined as

---

[†] In the case of Algorithm F (or Algorithm B) the mean square residual is obtained by dividing the residual sum of squares by $(n-m)$.

$$e_t = x_t - \sum_{j=1}^{m} a_j^* x_{t-j} . \tag{13}$$

The $z$ transform of the difference equation (13) yields

$$E(z)/X(z) = 1 - \sum_{j=1}^{m} a_j^* z^{-j} \tag{14}$$

where $z^{-1}$ is the unit delay operator. When the time series of interest is input to the filter defined by (13), it produces (approximately) a white noise output. Thus, if a white noise sequence is applied to the inverse of this filter the resulting time series has a spectrum similar to that of the original. When the residual is white, $E(z)$ is a constant satisfying

$$|E(z)|^2 = P_m \Delta t \tag{15}$$

where $P_m$ is defined by (12). Solving (14) for $X(z)$, and taking the square of the modulus, we obtain the expression

$$P(f) = \frac{P_m \Delta t}{\left| 1 - \sum_{j=1}^{m} a_j^* e^{-i2\pi f j \Delta t} \right|^2} \tag{16}$$

for the spectrum $P(f)$ at frequency $f$. (The Fourier transform is obtained by evaluating the $z$ transform on the unit circle, by substituting $z = e^{i2\pi f \Delta t}$.)

All that remains is to evaluate (16) for as many values of $f$ as desired, between $0$ and the Nyquist (folding) frequency $1/(2\Delta t)$. When periodic signals are present the spectrum must be evaluated at

closely-spaced frequencies in order to obtain a satisfactory representation of its peaks.

We conclude this section by illustrating the superiority of LS solutions to the MEM problem for short sample sinusoidal data. Our test signal was formed by summing a 0.03 Hz and a 0.2 Hz sine wave, generated in single precision arithmetic (approximately 6 significant digits) and sampled 10 times per second. Subroutine FABNE (from Appendix E) and Burg's method (as described by Andersen (1974)) were used to obtain MEM spectral estimates by processing $n = 75$ data points. The resulting spectra are shown in Figure 1.

The number of cycles available for analysis are 1.5 and 0.23 for the 0.2 Hz and 0.03 Hz signals, respectively. The spectra are shown one above the other for increasing values of $m$ (the number of coefficients). The vertical scale is 80 dB per division (a factor of $10^4$ in power), and successive spectra are shifted up by 80 dB . We have drawn in the test signal frequencies for convenience.

Since the purpose of this example is to compare LS solutions with those obtained from Burg's method, we used double precision versions of both subroutine FABNE and Burg's algorithm in order to minimize rounding error effects[†]. Notice that the Burg algorithm causes frequency splitting, and that it does not converge as $m$ increases. In contrast, subroutine

---

[†]In single precision arithmetic subroutine FABNE produces correct results out to $m = 6$ , but at $m = 7$ it returns an error code indicating that positive definiteness has been lost.

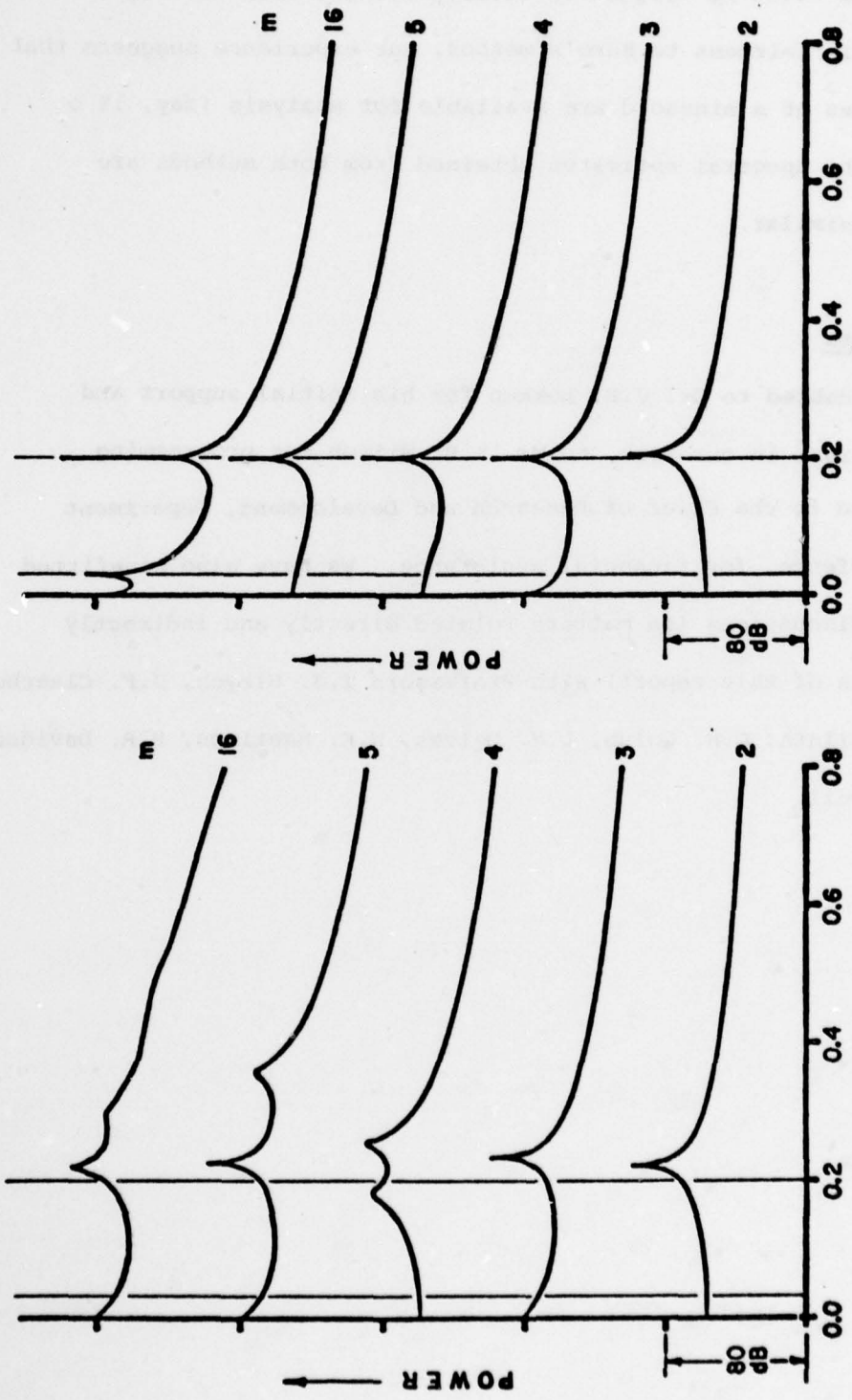



Figure 1. Comparison of BURG and FABNE spectra for a 0.2 and 0.03 Hz sine wave test signal. Spectra are plotted for increasing numbers of coefficients.

FABNE converges to the correct answer for the 0.2 Hz sine wave when $m = 3$, and the 0.03 Hz signal is clearly evident when $m = 16$.

Finally, in fairness to Burg's method, our experience suggests that when many cycles of a sinusoid are available for analysis (say, 15 or more cycles) the spectral estimates obtained from both methods are usually quite similar.

Acknowledgements

We are indebted to Dr. J.E. Lokken for his initial support and continued interest in our work, to Mr. K.B. Wilson for programming assistance, and to the Chief of Research and Development, Department of National Defence, for financial assistance. We have also benefitted greatly from discussions (on matters related directly and indirectly to the contents of this report) with Professors T.J. Ulrych, J.F. Claerbout, M. Morf, T. Kailath, G.H. Golub, L.M. Delves, W.K. Hastings, R.R. Davidson, and M.J.D. Powell.

References

Akaike, H., 1969. "Fitting autoregressive models for prediction". Ann. Inst. Stat. Math., vol. 21, pp. 243-247.

Andersen, N., 1974. "On the calculation of filter coefficients for maximum entropy spectral analysis". Geophysics, vol. 39, pp. 69-72.

Atal, B.S. and Hanauer, S.L., 1971. "Speech analysis and synthesis by linear prediction of the speech wave". J. Acoust. Soc. Amer., vol. 50, pp. 637-655.

Barber, F.G. and Taylor, J., 1977. "A note on free oscillations of Chedabucto Bay". Dept. Environment. Mar. Sci. Directorate, Man. Rep. Series No. 47, Ottawa, Ont.

Bennett, J.M., 1965. "Triangular factors of modified matrices". Numer. Math., vol. 7, pp. 217-221.

Box, G.E.P. and Jenkins, G.M., 1976. Time Series Analysis. Holden-Day, San Francisco, Calif.

Burg, J.P., 1967. "Maximum entropy spectral analysis". 37th Ann. Int. Meet., Soc. Explor. Geophys., Oklahoma City, Okla.

Burg, J.P., 1968. "A new analysis technique for time series data". NATO Adv. Study Inst. on Signal Processing, Enschede, Netherlands.

Burg, J.P., 1975. "Maximum entropy spectral analysis". Ph.D. thesis, Stanford University, Palo Alto, Calif.

Chen, W.Y. and Stegen, G.R., 1974. "Experiments with maximum entropy power spectra of sinusoids". J. Geophys. Res., vol. 79, pp. 3019-3022.

Claerbout, J.F., 1976. Fundamentals of Geophysical Data Processing. McGraw-Hill, New York.

Dahlquist, G. and Björk, A., 1974. Numerical Methods. Prentice-Hall, Englewood Cliffs, N.J.

Erickson, R.E., 1976. "Some experiments with maximum entropy spectral analysis". 23rd Pacif. N.W. Reg. Meet., Amer. Geophys. Union, University of Victoria, Victoria, B.C.

Fletcher, R. and Powell, M.J.D., 1974. "On the modification of $LDL^T$ factorizations". Math. Comp., vol. 28, pp. 1067-1087.

Friedlander, B., Morf, M., Kailath, T. and Ljung, L., 1977. "New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices". Submitted for publication.

Gill, P.E., Golub, G.H., Murray, W. and Saunders, M.A., 1974. "Methods for modifying matrix factorizations". Math. Comp., vol. 28, pp. 505-535.

Jones, R.H., 1976. "Autoregression order selection". Geophysics, vol. 41, pp. 771-773.

Kahan, W., 1972. "A survey of error analysis", pp. 1214-1239, from Information Processing 71. North-Holland, Amsterdam.

Lawson, C.L. and Hanson, R.J., 1974. Solving Least Squares Problems. Prentice-Hall, Englewood Cliffs, N.J.

Levinson, N., 1947. "The Wiener RMS error criterion in filter design and prediction". J. Math. Phys., vol. 25, pp. 261-278.

Martin, R.S., Peters, G. and Wilkinson, J.H., 1971. "Symmetric decomposition of a positive definite matrix", pp. 9-30, from Handbook for Automatic Computation, vol. II, by Wilkinson, J.H. and Reinsch, C. Springer-Verlag, New York.

Morf, M., Dickinson, B., Kailath, T. and Vieira, T., 1977. "Efficient solution of covariance equations for linear prediction". IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-25, pp. 429-433.

Ulrych, T.J. and Bishop, T.N., 1975. "Maximum entropy spectral analysis and autoregressive decomposition". Rev. Geophys., vol. 13, pp. 183-200.

Ulrych, T.J. and Clayton, R.W., 1976. "Time series modelling and maximum entropy". Phys. Earth Planet. Inter., vol. 12, pp. 188-200.

Appendix A: Akaike's final prediction error (FPE) criterion

The determination of the order of the AR process is of central importance in MEM spectral analysis. Akaike's (1969) final prediction error (FPE) for an n-point time series is given by

$$\text{(A1)} \qquad FPE_m = \frac{n + m + 1}{n - m - 1} P_m$$

where $P_m$ denotes the mean square residual (12) for the LS solution $\underset{\sim}{a}_*$ to the $m$ parameter problem (10). The "optimal" AR order $m$ is then determined by the minimum value of $FPE_m$ for $m = 1,2,\ldots$ . If the mean is not subtracted from the data prior to the calculation of the AR parameters, expression (A1) must be replaced by

$$\text{(A2)} \qquad FPE_m = \frac{n + m}{n - m} P_m \ .$$

Various opinions on the effectiveness of the FPE criterion have been expressed (e.g. see Ulrych and Bishop (1975), Ulrych and Clayton (1976), Jones (1976), Barber and Taylor (1977), and references therein), but most of this experience relates to its application to Burg's algorithm. Our experience with the LS algorithm of this paper suggests that while the FPE criterion provides a useful automatic method of selecting an appropriate value for $m$, it should be used with caution. (If $m$ is too small the resulting spectrum will be too smooth, and if $m$ is too large extraneous peaks will occur.)

For <u>any</u> of the three linear prediction problems (6), (8), or (10) the minimum residual sum of squares $S_m$ decreases as $m$ increases. Therefore, even when the statistical assumptions underlying the FPE criterion do not hold, it seems reasonable to allow $m$ to be determined as the

smallest order for which $S_m \approx S_{m+1}$ . When $n >> m$ this can be accomplished by choosing $m$ to be the first local minimum of $FPE_m$ . Thus, subroutine AKSTOP of Appendix E determines the "optimal" $m$ for problems (6), (8), or (10) as the smallest integer in the interval $[m_{start}, m_{last}]$ such that $FPE_{m+1} > FPE_m$ .

## Appendix B: Vector notation employed in the FORTRAN program

Since some FORTRAN compilers do not process two-dimensional arrays very efficiently, our program uses only vectors (i.e. linear arrays) when forming and solving the normal equations. Thus, the lower triangle of the matrix $\underset{\sim}{R}$ (or $\hat{\underset{\sim}{R}}$ or $\bar{\underset{\sim}{R}}$) is stored by rows as a vector $\underset{\sim}{v}$ with $m(m+1)/2$ elements. For example, when $m=4$ we have:

$$\begin{bmatrix} r_{1,1} & & & \\ r_{2,1} & r_{2,2} & & \\ r_{3,1} & r_{3,2} & r_{3,3} & \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix} \rightarrow \begin{bmatrix} v_1 & & & \\ v_2 & v_3 & & \\ v_4 & v_5 & v_6 & \\ v_7 & v_8 & v_9 & v_{10} \end{bmatrix} .$$

For the convenience of readers who require a statement of our algorithm that is closely related to the code in Appendix E, we provide the following description of Algorithm FAB in this vector notation:

$k \leftarrow m(m+1)/2 + 1$

$v_k \leftarrow s_m - x_{n-m}\, x_n - x_1\, x_{m+1}$

$j \leftarrow 1$

for $i = k+1,\ldots,k+m$ do

$j \leftarrow j + 1$

$v_i \leftarrow v_{i-m-1} - x_{n-m}\, x_{n+1-j} - x_j\, x_{m+1}$

$k \leftarrow 0$

for $i = 1,\ldots,m$ do

for $j = 1,\ldots,i$ do

$k \leftarrow k+1$

$v_k \leftarrow v_k - x_{m+1-i} - x_{m+1-j} - x_{n-m+i}\, x_{n-m+j}$

for $i = 1,\ldots,m$ do

$s_i \leftarrow s_i - x_{m+1-i}\, x_{m+1} - x_{n-m}\, x_{n-m+i}$

$$s_{m+1} \leftarrow 2 \times \sum_{t=1}^{n-m-1} x_t\, x_{m+1+t}$$

## Appendix C: Accurate calculation of inner-products

Inner-product calculations occur frequently in scientific programming. Indeed, in any system of normal equations every element of the coefficient matrix and the right-hand side vector is defined as an inner-product. In a typical program an algorithmic statement such as

(C1) $$\text{sum} \leftarrow \sum_{t=1}^{N} x_t w_t$$

is usually implemented in floating-point arithmetic by computing $\sum_{t=1}^{N} x_t w_t$ from single precision data $x_t$ and $w_t$ by forming double precision products and double precision sums, and then assigning the final result to the single precision variable sum. However, for many

of our applications it is necessary to avoid any dependence on higher precision arithmetic, and so the above scheme cannot be employed.

A single precision FORTRAN program segment of the form

(C2)
```
      SUM = 0.0
      DO 10 I = 1, N
   10 SUM = SUM + X(I)*W(I)
```

can produce a calculated value for SUM which differs from its true value by almost $1.06u \times \sum_{t=1}^{N} |(N+2-t)x_t w_t|$, where $u$ represents the roundoff unit of the floating-point system used (e.g. see Dahlquist and Björk (1974)). Even though the actual error incurred is often considerably less than this upper bound, for large values of $N$ the program (C2) does not usually compute inner-products accurately enough when forming the normal equations. (For example, if $n = 500$, $u \approx 10^{-6}$, $|x_t| \leq 1$, and $|w_t| \leq 1$, typical results from (C2) contain only about four significant decimal figures.)

We have therefore adopted the following single precision program segment when computing $\sum_{t=1}^{N} x_t w_t$ for use in forming the normal equations (6), (8), or (10). (It is based on a program due to Kahan (1972) for computing $\sum_{t=1}^{N} x_t$, which has a very small error bound that is essentially independent of $N$.)

(C3)
```
      SUM = 0.0
      C = 0.0
      DO 20 I = 1, N
      Y = C + X(I)*W(I)
      T = SUM + Y
      C = (SUM-T) + Y
   20 SUM = T
      SUM = SUM + C
```

The role of the variable C in (C3) is to recover the information that is lost when adding the products $x_t w_t$, but notice that no attempt is made to increase the precision of these products. Thus (C3) simulates double precision accumulation of single precision products, which is usually satisfactory when SUM is of large magnitude compared to these products.

This is almost always the situation that occurs in our applications, where most of the N products $x_t w_t$ are positive. The reason for this is because the term $w_t$ in (C1) is replaced by $x_{m+1+t}$ when forming the normal equations, and our typical time series $x_1, x_2, \ldots, x_n$ have few sign changes relative to n (i.e. most pairs $x_t, x_{m+1+t}$ are of the same sign when $n >> m$).

*Subroutine* ACCSUM of Appendix E is based on the program segment (C3). It is used to compute each inner-product that is calculated from first principles; thus it is called twice when $m=1$ and once for each value of $m = 2,3,\ldots$. Provided that SUM is large enough to ignore the rounding errors made in forming the products $x_t w_t$, subroutine ACCSUM produces a calculated value of SUM for which the relative error is bounded by $5u$. This bound applies to any practical value of N, and in our experience the actual magnitude of the relative error rarely exceeds u when $N \leq 1{,}000$.

## Appendix D: Calculating the residual sum of squares

Given an overdetermined $N \times M$ system of equations $\underset{\sim}{X}\,\underset{\sim}{a} = \underset{\sim}{y}$, where $\underset{\sim}{X} = [x_{i,j}]$ and $\underset{\sim}{y} = [y_1, y_2, \ldots, y_N]^T$, the residual sum of

squares $S = S_M(\underset{\sim}{a})$ is defined by

$$S_M(\underset{\sim}{a}) = \underset{\sim}{e}^T\underset{\sim}{e} = (\underset{\sim}{y} - \underset{\approx}{X}\,\underset{\sim}{a})^T(\underset{\sim}{y} - \underset{\approx}{X}\,\underset{\sim}{a})$$

(D1)

$$= \sum_{i=1}^{N} (y_i - \sum_{j=1}^{M} a_j\, x_{i,j})^2 .$$

For any given $\underset{\sim}{a}$, evaluating $S$ directly from the second line of (D1) involves $NM$ operations. Although $\underset{\approx}{X}$ and $\underset{\sim}{y}$ take special forms in the linear prediction problems (3), (7), and (9), their structure yields no reduction in this operation count. (Recall that $N = n-m$ for (3) and (7), $N = 2(n-m)$ for (9), and $M = m$ in all three problems.) Thus, when $n >> m$, calculating the residual sum of squares via (D1) for any value of $m$ requires almost as much computation as forming the normal equations of all orders from 1 through $m$ by Algorithms F, B, or FAB. Furthermore, when solving a sequence of normal equations (6), (8), or (10) for successive values of $m$, $S$ must be evaluated by (D1) ab initio for each value of $m$.

In view of this it is quite understandable that the following fast method of calculating $S$ is popular, particularly among engineers and statisticians. Expanding the first line of (D1) we obtain

(D2) $$S_M(\underset{\sim}{a}) = \underset{\sim}{y}^T\underset{\sim}{y} - \underset{\sim}{a}^T(\underset{\approx}{X}^T\underset{\sim}{y}) + \underset{\sim}{a}^T(\underset{\approx}{X}^T\underset{\approx}{X}\,\underset{\sim}{a} - \underset{\approx}{X}^T\underset{\sim}{y}) ,$$

and since the LS solution $\underset{\sim}{a}_*$ satisfies the normal equations

(D3) $$\underset{\approx}{X}^T\underset{\approx}{X}\,\underset{\sim}{a}_* = \underset{\approx}{X}^T\underset{\sim}{y} ,$$

substituting (D3) into (D2) gives

(D4) $$S_M(\underset{\sim}{a}_*) = \underset{\sim}{y}^T\underset{\sim}{y} - a_*^T(\underset{\sim}{X}^T\underset{\sim}{y}) \ .$$

Calculating the minimum residual sum of squares by formula (D4) involves only $N + M$ operations, since the term $\underset{\sim}{X}^T\underset{\sim}{y}$ was formed previously as the right-hand side vector of the normal equations (D3).

A further reduction can be made in this operation count when solving a sequence of normal equations (6), (8), or (10) for successive values of $m$. Specifically, when $m = 1$ the term $\underset{\sim}{y}^T\underset{\sim}{y}$ in (D4) can be obtained in two operations from $\underset{\sim}{x}^T\underset{\sim}{x}$. For example, in the case of (6), when $m = 1$ we calculate $\hat{\underset{\sim}{y}}^T\hat{\underset{\sim}{y}}$ as $(\hat{\underset{\sim}{x}}^T\hat{\underset{\sim}{x}} - x_1^2 + x_n^2)$. Thereafter, for each $m = 2,3,\ldots$, we obtain the current value of $\hat{\underset{\sim}{y}}^T\hat{\underset{\sim}{y}}$ in one operation via the assignment statement $\hat{\underset{\sim}{y}}^T\hat{\underset{\sim}{y}} \leftarrow \hat{\underset{\sim}{y}}^T\hat{\underset{\sim}{y}} - x_m^2$. Problem (8) is handled in an analogous fashion, while problem (10) requires two operations to update the value of $\underset{\sim}{y}^T\underset{\sim}{y}$ for each $m$. Thus, for any one of the three linear prediction problems, calculating the minimum residual sum of squares by formula (D4) involves no more than $m+2$ operations for each successive value of $m$.

Our program in Appendix E can be directed to obtain the minimum residual sum of squares in this efficient manner, merely by setting the input parameter RSCODE of subroutine FABNE equal to 0. Alternatively, setting RSCODE = 1 causes this minimum to be calculated via subroutine ACCSSQ, which employs formula (D1) rather than (D4).

As is clear from the above operation counts, the use of subroutine ACCSSQ is expensive in terms of computer time, but our computational experience leaves no doubt as to the necessity for this optional subroutine in practice. Formula (D4) involves a subtraction of two calculated terms

which are almost always of the same magnitude, and which frequently agree to more than two or three significant decimal figures. Indeed, this has to be the case when the minimum residual sum of squares is small relative to $\underset{\sim}{y}^T\underset{\sim}{y}$, and a single precision implementation of formula (D4) is then bound to suffer from significance losses. (In our opinion, many of the reported difficulties in implementing "optimal" selection rules (such as Akaike's criterion) can be attributed, at least in part, to careless usage of formula (D4).)

Since our algorithm forms and solves the normal equations in a numerically stable manner, it is not possible to significantly increase the accuracy obtainable from (D4) without resorting to higher precision arithmetic. (We note in passing that whereas Cholesky's method produces a calculated LS solution $\underset{\sim}{a}_\star$ for which the third term in (D2) is of negligible size, this is not always the case with some other algorithms. Hence, by ignoring the third term in (D2) these less stable methods sometimes introduce a sizeable truncation error into (D4).) In order to avoid the use of higher precision arithmetic, we abandon (D4) whenever significance losses become troublesome and resort instead to the slower formula (D1). When solving a sequence of problems for successive values of $m$, we usually delay this switch to subroutine ACCSSQ until the calculated values from (D4) are about two or three orders of magnitude less than the quantity $\underset{\sim}{y}^T\underset{\sim}{y}$. (Less experienced users may wish to switch periodically to subroutine ACCSSQ in order to gauge the accuracy of formula (D4) for a particular set of data.)

Appendix E: The FORTRAN subprogram

This appendix contains listings of six single precision FORTRAN subroutines which, together with an appropriate driver program, solve any one of the three linear prediction problems (3), (7), or (9). The user-supplied driver program must define the input parameters to subroutine FABNE, and arrange for display of the output parameters from FABNE. The other five subroutines (CHFAC, CHSOL, AKSTOP, ACCSUM, and ACCSSQ) are invoked automatically by FABNE, and hence require no action on the part of the user.

Subroutine FABNE forms the forward and/or backward prediction normal equations (6), (8), or (10) for a given time series $x_1, x_2, \ldots, x_n$, solves the normal equations by calls to subroutines CHFAC and CHSOL, and determines the "optimal" number of filter coefficients within the range $m_{start} \le m \le m_{last}$ by calls to subroutine AKSTOP. Calls are also made by FABNE to subroutine ACCSUM when forming inner-products, and sometimes (when directed by the FABNE input parameter RSCODE) to subroutine ACCSSQ when calculating the minimum residual sum of squares. FABNE is invoked by the following calling sequence:

```
CALL FABNE(X,N,MSTART,MLAST, ICODE, AKCODE,RSCODE, MDIM,V,
                    VC,S,SC,A,M,SSQ,P,OCODE)
```

Description of parameters

Input parameters

X - is a vector containing the time series.

N - is the length of the time series.

MSTART - is the minimum value of m for which the normal equations are solved.

MLAST - is the maximum value of m for which the normal equations could be solved.

ICODE - is an input code with values as follows:

1 - the forward prediction equations (6) are used,

-1 - the backward prediction equations (8) are used,

0 - the forward and backward equations (10) are used.

AKCODE - is an input code with values as follows:

0 - the sample mean has not been subtracted out of the time series,

1 - the sample mean has been subtracted out of the time series.

(This information is passed by FABNE to AKSTOP.)

RSCODE - is an input code with values as follows:

0 - the minimum residual sum of squares is calculated by the fast formula (D4),

1 - the minimum residual sum of squares is calculated by subroutine ACCSSQ, which uses the slower (but more accurate) formula (D1).

MDIM - is used to dimension the work space arrays and must be set to at least MLAST*(MLAST+1)/2.

Workspace

V,VC,S,SC - are workspace arrays used to form (in V and S) and solve (in VC and SC) the normal equations.

Output parameters

A - is a vector containing the "optimal" filter coefficients.

M - is the number of "optimal" filter coefficients.

SSQ - is the minimum residual sum of squares.

P - is the corresponding mean square residual.

OCODE - is an output code with values as follows:

0 - an "optimal" solution has been calculated (i.e. the usual exit).

1 - the maximum number of filter coefficients (MLAST) was used, but "optimality" was not achieved.

2 - premature termination has occurred because the current normal equations matrix is not positive definite (possibly due to rounding errors). In this case the previous solution is returned instead.

Subroutine CHFAC produces the Cholesky factorization $\underset{\sim}{L}\underset{\sim}{L}^T$ of any symmetric positive definite matrix $\underset{\sim}{A}$. The lower triangle of $\underset{\sim}{A}$ is stored by rows in the vector A, which is eventually overwritten by $\underset{\sim}{L}$. Subroutine CHSOL solves $\underset{\sim}{L}\underset{\sim}{L}^T\underset{\sim}{x} = \underset{\sim}{b}$, and hence it must be preceded by CHFAC so that the Cholesky factorization is already available. The right-hand side $\underset{\sim}{b}$ is stored in the vector B, which is eventually overwritten by the solution $\underset{\sim}{x}$. These two FORTRAN subroutines, which are based on the ALGOL procedures choldet 2 and cholsol 2 of Martin, Peters, and Wilkinson (1971), were kindly supplied to us by Dr. M.G. Cox of the National Physical Laboratory, Teddington, England.

Subroutine AKSTOP implements the Akaike FPE criterion described in Appendix A. Users who wish to substitute some other optimality criterion

in place of Akaike's rule should have no difficulty in replacing AKSTOP by a subroutine of their own design.

Subroutine ACCSUM calculates inner-products accurately, assuming that rounding errors made in evaluating the products are negligible (see Appendix C). If higher precision arithmetic is allowed, ACCSUM could be replaced by a higher precision subroutine based on the simpler program segment (C2) of Appendix C. In most environments, savings in computer time would result from such a substitution.

Subroutine ACCSSQ (which is only used when RSCODE = 1) calculates the minimum residual sum of squares by the slow formula (D1) of Appendix D.

Finally, a double precision version of all six subroutines given below can be obtained by making the following changes:

(a) replace all REAL declarations by DOUBLE PRECISION declarations,

(b) replace SQRT by DSQRT in lines 17 and 40 of CHFAC,

(c) replace the constant 1.E20 by 1.D20 in the DATA statement in line 93 of FABNE.

```
      SUBROUTINE FABNE(X, N, MSTART, MLAST, ICODE, AKCODE,
     * RSCODE, MDIM, V, VC, S, SC, A, M, SSQ, P, OCODE)
      REAL A, S, V, X, SC, VC, SSQ, P
      INTEGER M, N, MDIM, MLAST, MSTART, ICODE, AKCODE,
     * OCODE, RSCODE
      DIMENSION X(N), V(MDIM), VC(MDIM), S(MLAST),
     * SC(MLAST), A(MLAST)
C
C THIS SUBROUTINE FORMS THE FORWARD AND/OR BACKWARD NORMAL
C EQUATIONS FOR AN N-POINT TIME SERIES X, SOLVES THE
C NORMAL EQUATIONS BY CALLS TO SUBROUTINES CHFAC AND
C CHSOL, AND DETERMINES THE 'OPTIMAL' NUMBER OF FILTER
C COEFFICIENTS WITHIN THE RANGE MSTART.LE.M.LE.MLAST BY
C CALLS TO SUBROUTINE AKSTOP. (IF THE REQUIRED NUMBER M OF
C FILTER COEFFICIENTS IS KNOWN BEFOREHAND, MSTART AND
C MLAST SHOULD BOTH BE SET EQUAL TO M.) THE NORMAL
C EQUATIONS ARE FORMED EFFICIENTLY FOR M=1,2,3,..., AND
C THEY ARE SOLVED FOR M=MSTART,MSTART+1,MSTART+2,... UNTIL
C EITHER M=MLAST OR 'OPTIMALITY' IS ACHIEVED. CALLS ARE
C MADE TO SUBROUTINE ACCSUM WHEN FORMING INNER-PRODUCTS,
C AND SOMETIMES TO SUBROUTINE ACCSSQ WHEN CALCULATING THE
C RESIDUAL SUM OF SQUARES (SEE DESCRIPTION OF PARAMETER
C RSCODE BELOW).
C
C
C FORMAL PARAMETER LIST:
C
C INPUT PARAMETERS
C
C X       A VECTOR REPRESENTING THE TIME SERIES.
C N       THE DIMENSION OF X.
C MSTART  THE FIRST VALUE OF M FOR WHICH THE NORMAL
C         EQUATIONS ARE SOLVED (MSTART.GE.1).
C MLAST   THE LARGEST VALUE OF M FOR WHICH THE
C         NORMAL EQUATIONS COULD BE SOLVED (MLAST.GE.
C         MSTART); ALSO USED TO DIMENSION S,SC, AND A.
C ICODE   AN INTEGER INPUT CODE WITH VALUES:
C              -1 - ONLY BACKWARD PREDICTION IS USED,
C               1 - ONLY FORWARD PREDICTION IS USED,
C               0 - BOTH FORWARD AND BACKWARD PREDICTION
C                   IS USED.
C AKCODE AN INTEGER INPUT CODE USED IN SUBROUTINE AKSTOP
C         (SEE SUBROUTINE AKSTOP).
C RSCODE AN INTEGER INPUT CODE WITH VALUES:
C               0 - A 'FAST' CALCULATION OF THE RESIDUAL
C                   SUM OF SQUARES IS USED,
C               1 - A MORE ACCURATE (BUT SLOWER) CALCULATION
C                   OF THE RESIDUAL SUM OF SQUARES IS USED BY
C                   CALLING SUBROUTINE ACCSSQ.
C MDIM    THE DIMENSION OF V AND VC (MDIM.GE.MLAST*
C         (MLAST+1)/2).
C
C WORKSPACE
C
C V       A VECTOR REPRESENTING THE LOWER TRIANGLE OF
```

```
C         THE NORMAL EQUATIONS MATRIX, WHICH IS STORED
C         IN V BY ROWS.
C VC      A COPY OF V WHICH IS USED IN THE CHOLESKY
C         FACTORIZATION.
C S       A VECTOR REPRESENTING THE RIGHT HAND SIDE OF
C         THE NORMAL EQUATIONS.
C SC      A COPY OF S WHICH IS USED IN THE CHOLESKY
C         SOLUTION.
C
C OUTPUT PARAMETERS
C
C A       A VECTOR CONTAINING THE OPTIMAL FILTER
C         COEFFICIENTS.
C M       THE OPTIMAL NUMBER OF FILTER COEFFICIENTS.
C SSQ     THE RESIDUAL SUM OF SQUARES FOR THE OPTIMAL
C         M COEFFICIENT FILTER.
C P       THE MEAN SQUARE RESIDUAL FOR THE OPTIMAL
C         M COEFFICIENT FILTER,GIVEN BY
C
C              P = SSQ / DENOM,
C
C                        2(N-M) IF ICODE.EQ.0
C         WHERE DENOM =
C                        (N-M) IF ICODE.EQ.(1.OR.-1).
C
C OCODE   AN OUTPUT CODE WITH VALUES:
C              0-'OPTIMAL' SOLUTION FOUND (I.E. NORMAL EXIT),
C              1-M.EQ.MLAST, BUT 'OPTIMALITY' WAS NOT
C                ACHIEVED (THE USER MAY WISH TO INCREASE
C                THE VALUE OF MLAST).
C              2-PREMATURE TERMINATION, BECAUSE THE CURRENT
C                NORMAL EQUATIONS MATRIX IS NOT POSITIVE
C                DEFINITE (POSSIBLY DUE TO ROUNDING ERRORS).
C                THE PREVIOUS SOLUTION IS RETURNED IN THIS
C                EVENT.
C
C LOCAL VARIABLES
C
      REAL CHSS, SSQL, PL, SUM, YTY, BIG
      INTEGER I, J, K, KD, KPM, KP1, MP1, NMM, NP1, IFAIL,
     * NMMM1, ITEND
C
C THE CONSTANT BIG CAN BE SET TO ANY LARGE REAL NUMBER.
C IT IS USED TO ASSIGN AN INITIAL VALUE TO SSQ AND P
C WHEN MSTART.GT.1.
C
      DATA BIG /1.E20/
C
C FORM AND SOLVE THE NORMAL EQUATIONS FOR M=1.
C
      OCODE = 0
      NP1 = N + 1
      M = 1
      NMM = N - M
      CALL ACCSUM(X, N, NMM, 0, SUM)
```

```
      IF (ICODE) 10, 20, 30
C BACKWARD
   10 YTY = SUM
      SUM = SUM - X(1)**2 + X(N)**2
      GO TO 40
C BOTH
   20 SUM = 2.0*SUM - X(1)**2 + X(N)**2
      YTY = SUM
      GO TO 40
C FORWARD
   30 SUM = SUM
      YTY = SUM - X(1)**2 + X(N)**2
   40 V(1) = SUM
      CALL ACCSUM(X, N, NMM, 1, SUM)
      IF (ICODE) 50, 60, 70
C BACKWARD
   50 SUM = SUM
      GO TO 80
C BOTH
   60 SUM = SUM*2.0
      GO TO 80
C FORWARD
   70 SUM = SUM
   80 S(1) = SUM
      A(1) = S(1)/V(1)
      SSQL = YTY - A(1)*S(1)
      IF (RSCODE.EQ.1) CALL ACCSSQ(M, N, X, A, ICODE, SSQL)
      PL = SSQL/(N-M)
      IF (ICODE.EQ.0) PL = SSQL/(2.0*(N-M))
      IF (MLAST.EQ.1) GO TO 310
      IF (MSTART.EQ.1) GO TO 90
      SSQL = BIG
      PL = BIG
   90 GO TO 140
C
C SOLVE THE NEW NORMAL EQUATIONS AND TEST FOR OPTIMALITY.
C
  100 CONTINUE
      KD = M*(M+1)/2
      IF (M.LT.MSTART) GO TO 140
      KD = M*(M+1)/2
      DO 110 K=1,KD
         VC(K) = V(K)
  110 CONTINUE
      DO 120 I=1,M
         SC(I) = S(I)
  120 CONTINUE
      CALL CHFAC(M, MDIM, VC, IFAIL)
      IF (IFAIL.NE.0) GO TO 300
      CALL CHSOL(M, MDIM, VC, SC, CHSS)
      SSQ = YTY - CHSS
      IF (RSCODE.EQ.1) CALL ACCSSQ(M, N, X, SC, ICODE, SSQ)
      P = SSQ/(N-M)
      IF (ICODE.EQ.0) P = SSQ/(2.0*(N-M))
      CALL AKSTOP(P, PL, M, N, AKCODE, ITEND)
```

```
      IF (ITEND.EQ.1) GO TO 310
      DO 130 I=1,M
         A(I) = SC(I)
  130 CONTINUE
      IF (M.GE.MLAST) GO TO 320
      SSQL = SSQ
      PL = P
C
C FORM THE NORMAL EQUATIONS FOR SUCCESSIVE VALUES OF M.
C
  140 IF (ICODE) 150, 200, 250
C BACKWARD
  150 MP1 = M + 1
      K = M*(M+1)/2 + 1
      V(K) = S(M) - X(1)*X(MP1)
      KP1 = K + 1
      KPM = K + M
      J = 1
      DO 160 I=KP1,KPM
         J = J + 1
         V(I) = V(I-MP1) - X(J)*X(MP1)
  160 CONTINUE
      K = 0
      DO 180 I=1,M
         DO 170 J=1,I
            K = K + 1
            V(K) = V(K) - X(NMM+I)*X(NMM+J)
  170    CONTINUE
  180 CONTINUE
      DO 190 I=1,M
         S(I) = S(I) - X(NMM)*X(NMM+I)
  190 CONTINUE
      NMMM1 = NMM - 1
      CALL ACCSUM(X, N, NMMM1, MP1, SUM)
      S(MP1) = SUM
      YTY = YTY - X(NMM)**2
      M = MP1
      NMM = N - M
      GO TO 100
C BOTH
  200 MP1 = M + 1
      K = M*(M+1)/2 + 1
      V(K) = S(M) - X(NMM)*X(N) - X(1)*X(MP1)
      KP1 = K + 1
      KPM = K + M
      J = 1
      DO 210 I=KP1,KPM
         J = J + 1
         V(I) = V(I-MP1) - X(NMM)*X(NP1-J) - X(J)*X(MP1)
  210 CONTINUE
      K = 0
      DO 230 I=1,M
         DO 220 J=1,I
            K = K + 1
            V(K) = V(K) - X(MP1-I)*X(MP1-J) -
     *       X(NMM+I)*X(NMM+J)
```

```
  220    CONTINUE
  230 CONTINUE
      DO 240 I=1,M
         S(I) = S(I) - X(MP1-I)*X(MP1) - X(NMM)*X(NMM+I)
  240 CONTINUE
      NMMM1 = NMM - 1
      CALL ACCSUM(X, N, NMMM1, MP1, SUM)
      S(MP1) = 2.E0*SUM
      YTY = YTY - X(MP1)**2 - X(NMM)**2
      M = MP1
      NMM = N - M
      GO TO 100
C FORWARD
  250 MP1 = M + 1
      K = M*(M+1)/2 + 1
      V(K) = S(M) - X(NMM)*X(N)
      KP1 = K + 1
      KPM = K + M
      J = 1
      DO 260 I=KP1,KPM
         J = J + 1
         V(I) = V(I-MP1) - X(NMM)*X(NP1-J)
  260 CONTINUE
      K = 0
      DO 280 I=1,M
         DO 270 J=1,I
            K = K + 1
            V(K) = V(K) - X(MP1-I)*X(MP1-J)
  270    CONTINUE
  280 CONTINUE
      DO 290 I=1,M
         S(I) = S(I) - X(MP1-I)*X(MP1)
  290 CONTINUE
      NMMM1 = NMM - 1
      CALL ACCSUM(X, N, NMMM1, MP1, SUM)
      S(MP1) = SUM
      YTY = YTY - X(MP1)**2
      M = MP1
      NMM = N - M
      GO TO 100
C
C PREPARE THE OUTPUT.
C
  300 OCODE = 2
      IF (M.GT.MSTART) GO TO 310
      M = 1
      SSQ = SSQL
      P = PL
      GO TO 320
  310 SSQ = SSQL
      P = PL
      IF (M.EQ.1) GO TO 320
      M = M - 1
  320 IF (ITEND.EQ.0 .AND. OCODE.NE.2) OCODE = 1
      RETURN
      END
```

```
      SUBROUTINE CHFAC(N, NT, A, IFAIL)
      REAL A
      INTEGER N, NT, IFAIL
      DIMENSION A(NT)
C
C     CHOLESKY DECOMPOSITION LU (U = TRANSPOSE OF L) OF
C     N*N SYMMETRIC POSITIVE DEFINITE MATRIX A. LOWER
C     TRIANGLE OF A STORED BY ROWS IN A, AND OVERWRITTEN
C     BY LOWER TRIANGLE OF L.  NT MUST BE NO SMALLER
C     THAN N*(N + 1)/2. BASED ON THE MARTIN-PETERS-
C     WILKINSON ALGORITHM CHOLDET2.
C
C     NO PROGRAM UNITS CALLED.
C
C     LOCAL VARIABLES -
C
      REAL ONE, X, ZERO, SQRT
      INTEGER I, IERROR, J, K, P, Q, R
C
C     CONSTANTS -
C
      ZERO = 0.E0
      ONE = 1.0E0
C
      IERROR = 1
      P = 0
      DO 40 I=1,N
         Q = P + 1
         R = 0
         DO 30 J=1,I
            X = A(P+1)
            IF (Q.GT.P) GO TO 20
            DO 10 K=Q,P
               R = R + 1
               X = X - A(K)*A(R)
   10       CONTINUE
   20       R = R + 1
            P = P + 1
            IF (I.EQ.J .AND. X.LE.ZERO) GO TO 50
            IF (I.EQ.J) A(P) = ONE/SQRT(X)
            IF (I.NE.J) A(P) = X*A(R)
   30    CONTINUE
   40 CONTINUE
      IERROR = IERROR - 1
   50 IFAIL = IERROR
      RETURN
      END
```

```
      SUBROUTINE CHSOL(N, NT, L, B, SS)
      REAL B, L, SS
      INTEGER N, NT
      DIMENSION B(N), L(NT)
C
C     SOLVES LUX = B (U = TRANSPOSE OF THE N*N LOWER
C     TRIANGULAR MATRIX L), X OVERWRITING B.  LOWER
C     TRIANGLE OF L STORED BY ROWS IN L.  FORMS IN SS
C     THE SQUARE OF THE 2-NORM OF Y = UX.  NT MUST BE
C     NO SMALLER THAN N*(N + 1)/2.  BASED ON THE MARTIN-
C     PETERS-WILKINSON ALGORITHM CHOLSOL2.
C
C     NO PROGRAM UNITS CALLED
C
C     LOCAL VARIABLES -
C
      REAL T, X, ZERO
      INTEGER I, IREV, K, KREV, P, Q, S
C
C     CONSTANTS -
C
      ZERO = 0.E0
C
C     SOLVE LY = B AND FORM SS -
C
      T = ZERO
      P = 1
      DO 30 I=1,N
         Q = I - 1
         X = B(I)
         IF (Q.LT.1) GO TO 20
         DO 10 K=1,Q
            X = X - L(P)*B(K)
            P = P + 1
   10    CONTINUE
   20    B(I) = X*L(P)
         P = P + 1
         T = T + B(I)**2
   30 CONTINUE
      SS = T
C
C     SOLVE UX = Y -
C
      DO 60 IREV=1,N
         I = N + 1 - IREV
         S = P - 1
         P = S
         Q = I + 1
         X = B(I)
         IF (Q.GT.N) GO TO 50
         DO 40 KREV=Q,N
            K = N + Q - KREV
            X = X - L(S)*B(K)
            S = S - K + 1
   40    CONTINUE
   50    B(I) = X*L(S)
   60 CONTINUE
      RETURN
      END
```

substituting (D3) into (D2) gives



```
      SUBROUTINE AKSTOP(P, PL, M, N, AKCODE, ITEND)
      REAL P, PL
      INTEGER M, N, AKCODE, ITEND
C
C FOLLOWING THE AKAIKE GOODNESS-OF-FIT CRITERION, WE
C SAY THAT M-1 IS OPTIMAL IF M IS THE SMALLEST POSITIVE
C INTEGER FOR WHICH,
C
C IF THE MEAN IS SUBTRACTED OUT,
C
C       P*(N+(M+1))/(N-(M+1)).GT.PL*(N+M)/(N-M),
C
C BUT OTHERWISE,
C
C       P*(N+M)/(N-M).GT.PL*(N+(M-1))/(N-(M-1)).
C
C
C HERE N IS THE LENGTH OF THE TIME SERIES,
C M IS THE NUMBER OF COEFFICIENTS, P IS THE MEAN SQUARE
C RESIDUAL, AND PL IS THE CORRESPONDING QUANTITY FOR
C M-1 COEFFICIENTS.
C
C AKCODE IS AN INTEGER INPUT PARAMETER WHICH
C MUST BE SET TO 1 IF THE MEAN IS SUBTRACTED
C OUT OF THE TIME SERIES, AND WHICH MUST BE
C SET TO 0 OTHERWISE.
C ITEND IS AN INTEGER OUTPUT PARAMETER WHICH IS SET
C TO 1 IF THE AKAIKE CRITERION IS SATISIFIED, AND WHICH
C IS SET TO 0 OTHERWISE.
C
C LOCAL VARIABLES -
      REAL XN, XM
      ITEND = 0
      XN = N
      XM = M
      IF (AKCODE.NE.1) GO TO 10
      IF (P*((XN+XM+1.0)/(XN-XM-1.0)).GT.PL*((XN+XM)/(XN-XM)
     * )) ITEND = 1
      RETURN
   10 CONTINUE
      IF (P*((XN+XM)/(XN-XM)).GT.PL*((XN+XM-1.0)/(XN-XM+1.0)
     * )) ITEND = 1
      RETURN
      END
```

```
      SUBROUTINE ACCSUM(X, N, NMM, MOVE, SUM)
      REAL X, SUM
      INTEGER N, NMM, MOVE
      DIMENSION X(N)
C
C CALCULATES INNER-PRODUCTS ACCURATELY, ASSUMING THAT
C ROUNDING ERRORS MADE IN EVALUATING THE PRODUCTS ARE
C NEGLIGABLE.
C
C LOCAL VARIABLES -
C
      REAL S, C, Y, T
      INTEGER I
      S = 0.0
      C = 0.0
      DO 10 I=1,NMM
         Y = C + X(I)*X(I+MOVE)
         T = S + Y
         C = (S-T) + Y
         S = T
   10 CONTINUE
      SUM = S + C
      RETURN
      END
```

```
      SUBROUTINE ACCSSQ(M, N, X, A, ICODE, SSQ)
      REAL X, A, SSQ
      INTEGER M, N, ICODE
      DIMENSION X(N), A(M)
C
C CALCULATES SUM OF SQUARES ACCURATELY, FROM FIRST
C PRINCIPLES.
C
C LOCAL VARIABLES -
      REAL APX, TSSQ
      INTEGER I, J, NMM
      TSSQ = 0.0
      NMM = N - M
      IF (ICODE) 50, 10, 10
C FORWARD ONLY
   10 CONTINUE
      SSQ = 0.0
      DO 30 I=1,NMM
         APX = 0.0
         DO 20 J=1,M
            APX = APX + A(J)*X(M+I-J)
   20    CONTINUE
         SSQ = SSQ + (X(M+I)-APX)**2
   30 CONTINUE
      IF (ICODE) 50, 40, 80
C SAVE FORWARD SSQ FOR 'BOTH' CALCULATION.
   40 TSSQ = SSQ
C BACKWARD ONLY
   50 CONTINUE
      SSQ = 0.0
      DO 70 I=1,NMM
         APX = 0.0
         DO 60 J=1,M
            APX = APX + A(J)*X(J+I)
   60    CONTINUE
         SSQ = SSQ + (X(I)-APX)**2
   70 CONTINUE
C ADD FORWARD AND BACKWARD TO FORM 'BOTH' SSQ.
      SSQ = SSQ + TSSQ
   80 CONTINUE
      RETURN
      END
```