$$|e_i^h|^2 \leqslant e_0^h \cdot r_0 \qquad \forall i. \tag{18}$$

Moreover, it has been shown [7] that for exact correlation data,

$$e_0^h = r_0 \prod_{i=0}^{h-1} (1 - k_i^2) \qquad \text{and} \qquad -1 \leqslant k_i \leqslant +1. \tag{19}$$

Consequently, the following bound holds:

$$|e_i^h| \leqslant r_0. \tag{20}$$

Using the addition assumption $r_0 < 1$, the bound (20) shows that all the intermediate variables lie between -1 and $+1$. As a consequence, the implementation may be conducted using fixed point arithmetics only. The bound can be extended for higher orders $h' \geqslant h$. Due to the inequality,

$$e_0^{h'} \leqslant e_0^h \qquad \forall h' \geqslant h, \tag{21}$$

a range of variations for the further e_i^h is

$$|e_i^{h'}|^2 \leqslant e_0^h \qquad \forall h' \geqslant h. \tag{22}$$

Since (15c) is homogeneous in those variables, it is then possible to shift the stored values to the left in order to improve the precision in the following computations.

If between the stages $(h\text{-}1)$ and (h) the number of most significant bits to be zero is increased by $2b$, a shift of b bits to the left can then be made on the e_i^h.

V. Experimental Results

This method requires $(p^2 - p + 1)$ multiplications and divisions (i.e., 91 for a 10th-order model) and $(2p + 3)$ memory cells. It has been implemented using fixed point arithmetics on a standard 16-digit computer (PACER 100 EAI, multiply 5.6 μs). In that case, the calculation of the PARCOR coefficients for a 10th-order model requires 3.5 ms. Even if the correlation time is much more for speech signal, this allows the LPC to be applied several times for a better estimation (ARMA models, for instance [9]). The left shifts do not improve the results very much (i.e., no significant change on the pole location). The maximum shift reached only two digits. The inequality (20) was always satisfied, and the stability was guaranteed for more than 3000 computations of PARCOR coefficients on real speech waves.

The method has been extensively compared to the usual algorithms implemented using a floating point processor. In more than 100 experiments, the differences between the results is less than 0.005 on k_{10}. The frequency deviations do not exceed 1 Hz for the most significant poles at a sampling rate of 10 kHz. The differences are below the possible deviations occurring in the computations of the autocorrelation coefficients on a vocal signal.

VI. Conclusions

The PARCOR coefficients have proven to be of major interest in speech analysis, synthesis, transmission, and even as intermediate parameters. This approach has also been applied successfully on various types of other signals [10].

This paper has proposed a new computational algorithm for these coefficients by introducing new intermediate variables. The computations are then made on estimated impulse responses with no reference to the AR model coefficients. As a consequence, the proposed algorithm can be implemented using fixed point arithmetics only, and it may contribute to the development of specialized processors achieving the linear prediction in real time for fast signals.

References

[1] B. Atal and S. Hanauer, "Speech analysis and synthesis by linear prediction of the speech wave," *J. Acoust. Soc. Amer.*, vol. 41, pp. 293–309, 1967.

[2] F. Itakura and S. Saito, "Analysis synthesis telephony based on the maximum likelihood method," presented at the 6th Int. Congr. on Acoust., Tokyo, Japan, 1968.

[3] J. D. Markel, "Digital inverse filtering: A new tool for formant trajectories estimation," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 129–137, Apr. 1972.

[4] N. Levinson, "The Wiener RMS error criterion in filter design and prediction," *J. Math. Phys.*, vol. 25, no. 4, p. 261, 1947.

[5] E. A. Robinson, *Statistical Communication and Detection With Special Reference to Digital Data Processing of Radar and Seismic Signals.* New York: Hafner, 1967, p. 274.

[6] J. Durbin, "The fitting of time-series models," *Rev. Inst. Int. Stat.*, vol. 28, no. 3, pp. 344–348, 1973.

[7] J. D. Markel and A. H. Gray, "On autocorrelation equations as applied to speech analysis," *IEEE Trans. Audio. Electroacoust.*, vol. AU-21, Apr. 1973.

[8] J. Makhoul, "Linear prediction, A tutorial review," *Proc. IEEE*, vol. 63, Apr. 1975.

[9] C. Gueguen and M. Mathieu, "Contribution des zéros à la modélisation du signal de parole," presented at the 7ème Journées du Galf, Nancy, France, 1976.

[10] M. Mathieu, "Analyse de l'électro-encéphalogramme par prédiction linéaire," thèse de Docteur-Ingénieur, Paris, France, 1976.

1977-3-3724

A Simplified Computational Algorithm for Implementing FIR Digital Filters

LAWRENCE R. RABINER

Abstract—An N-point finite impulse response (FIR) digital filter, when implemented in software, generally requires N multiplications, N additions, and $(N-1)$ shifts per output sample. An obvious simplification is to implement the shift register required to store $x(n)$ to $x(n-N+1)$ using a moving pointer, thereby eliminating the $(N-1)$ shifts per sample required in the most straightforward implementation. However, this simplification requires a check on the index of every sample to see if the end of the linear storage array has been passed, thereby voiding most of the gain in speed which was obtained. A simple technique is discussed for trading N storage locations for the $(N-1)$ shifts (or the $(N-1)$ index checks), thereby leading to an implementation in which only N multiplications, N additions, and one indexing operation are required per output sample. For implementing symmetric (i.e., linear phase) FIR filters, the resulting savings is even somewhat greater because two index pointers are required, and each must normally be checked to see that it remains within the bounds of the array.

FIR Implementation

If $x(n)$ is the input to an N-point finite impulse response (FIR) digital filter with impulse response $h(n)$, $0 \leqslant n \leqslant N-1$, and the output is called $y(n)$, then

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \tag{1}$$

which is realized as shown in Fig. 1(a). For each output

Manuscript received September 13, 1976; revised January 12, 1977.
The author is with Bell Laboratories, Murray Hill, NJ 07974.

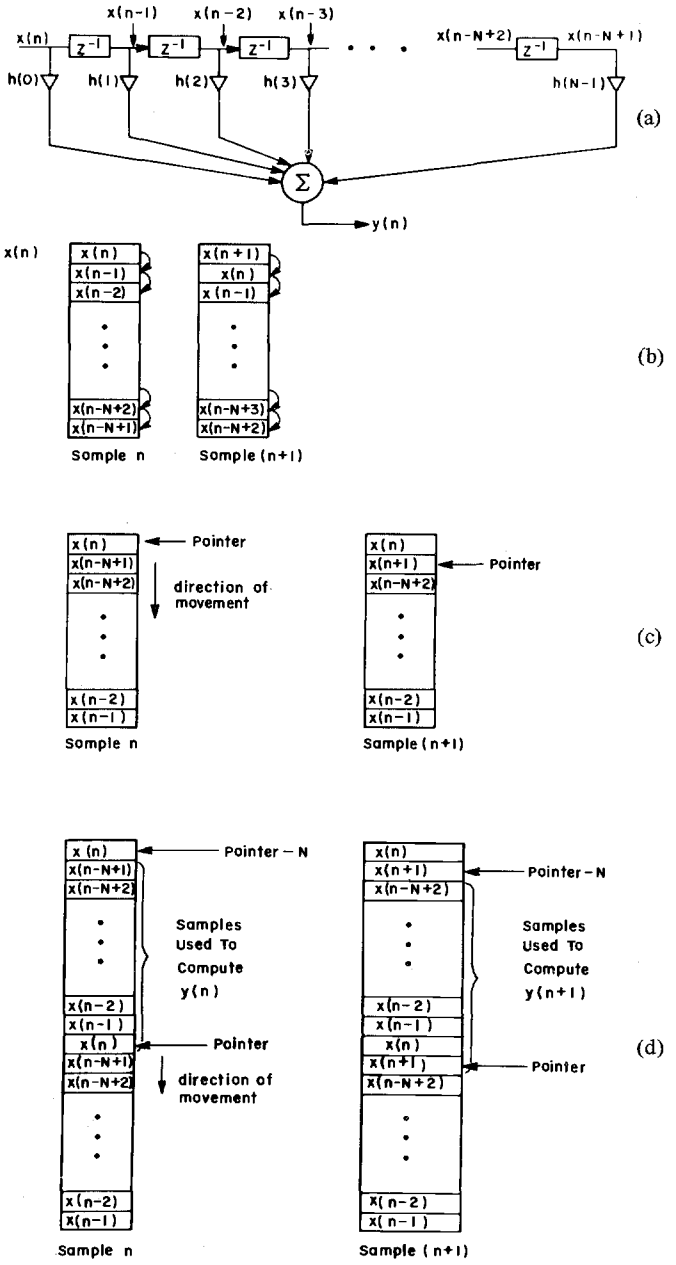

Fig. 1. Various methods of implementing (in software) the computation of (1).

sample, a total of N multiplications, N additions, and one shift of the entire shift register is required. When implemented in software, however, the shifting of a new input into the shift register and the moving down in memory of the other $N-1$ past input samples requires a total of $N-1$ memory shifts (each involving a load, index, and store). This procedure, as illustrated in Fig. 1(b), is clearly an inefficient one. It can be simplified by effectively converting the linear storage array into a dynamic storage array through the use of a moving pointer, as illustrated in Fig. 1(c). The purpose of the pointer is to indicate the place into which $x(n)$ is to be stored. The locations of all previous inputs ($x(n-1)$ to $x(n-N+1)$) can be readily obtained from the current pointer, and knowledge as to the direction of movement of the pointer.

The major difficulty with the technique of Fig. 1(c) is that as the index (location) of previous samples [$x(n-m)$ in (1)] is computed, it must be checked to ensure that it does not fall outside the range of the array. As such, the $N-1$ memory shifts have been traded for $N-1$ index checks, thereby canceling most of the computational gain.

To alleviate this problem, the implementation of Fig. 1(d) is proposed in which N extra storage locations are traded for the $N-1$ index checks, thereby speeding up the computation, but at the same time doubling the storage requirements. (For most computer implementations the additional N memory locations are generally a trivial cost.) Each new input sample $x(n)$ is stored in *two* locations—displaced by N samples in the array. An index pointer is maintained which shows where in the array to store $x(n)$. Since the array is $2N$ locations long, and since each half of the array is identical, to ensure that the indexed variable for $x(n-m)$ does not fall outside the range of the array merely requires that the index for $x(n)$ fall in the upper half of the array—a trivial requirement which is easily met. Thus, with the implementation of Fig. 1(d), no checking on indices is done inside of the output sample computation loop [i.e., in the computation of (1)]. Of course, for each new in-

```
C        EXAMPLE ILLUSTRATING USE OF FIR FILTER IMPLEMENTATION
C        N IS THE IMPULSE RESPONSE DURATION IN SAMPLES
      DIMENSION H(N)
C        LOAD H ARRAY FROM H(1) TO H(N)
      READ(1,1) (H(I),I=1,N)
 1    FORMAT(7F11.7)
C        INITIALIZE FILTER
C        X IS THE FILTER INPUT
C        Y IS THE FILTER OUTPUT
C        N2 IS THE LENGTH OF THE STORAGE ARRAY
      N2=2*N
      CALL FILT(1,X,Y,H,N,N2)
C        NSAMP IS THE NUMBER OF SAMPLES TO BE FILTERED
      DO 10 I=1,NSAMP
C        READ IN INPUT SAMPLE X
      READ(1,2) X
 2    FORMAT(F10.5)
      CALL FILT(0,X,Y,H,N,N2)
C        STORE THE OUTPUT SAMPLE Y
      WRITE(2,3) Y
 3    FORMAT(4H  Y=,F10.5)
 10   CONTINUE
      STOP
      END
      SUBROUTINE FILT(INIT,X,Y,H,N,N2)
      DIMENSION H(1),XSAV(N2)
      IF(INIT.EQ.0) GO TO 50
      DO 10 I=1,N2
 10   XSAV(I)=0.
      IPT=N+1
      RETURN
 50   XSAV(IPT)=X
      XSAV(IPT-N)=X
      Y=0.
      DO 60 I=1,N
 60   Y=Y+H(I)*X(IPT-I+1)
      IPT=IPT+1
      IF(IPT.GT.N2) IPT=N+1
      RETURN
      END
$0
```

Fig. 2. A simple program which implements the structure of Fig. 1(d).

put sample the pointer location is indexed by one location and must be checked to ensure that it remains within the bounds of the array. However, the computation required here is trivial compared to the computation involved in (1).

For linear phase filters, somewhat larger savings accrue since two index pointers are generally required to access $x(n-m)$ and $x(n-N+1+m)$ as m varies from 0 to $(N-1)/2$. Using the structure of Fig. 1(d) (with appropriate modifications) neither index pointer need be checked within the output computation to ensure that it stays within the range of the array.

Fig. 2 shows a Fortran realization of the structure of Fig. 1(d) with a main calling program which filters 1000 samples of a signal.

A Note on Beam Pattern Analysis Using the Replica Pulse Approximation

D. E. ROBINSON

Abstract—**It has previously been reported that for field points near the axis, accurate location of field extrema using the replica pulse representation requires that the replicas be shifted by a constant amount from the positions derived geometrically. This note demonstrates that the same shift is required for field points distant from the axis.**

The use of a small number of replica pulses to represent the complete response at a field point due to a piston radiator has been reported by Deyne [1] for the case of a circular piston, and by Freedman [2] for plane and gently curved radiators of arbitrary shape. In each case, the epoch or arrival time of the various replicas was related to the travel times from particular points on the radiator surface. For the plane circular piston, illustrated in Fig. 1, the travel times of interest are from the point on the surface closest to the field point (t_0) and the nearest and furthest edges (t_1 and t_2). This approach is equivalent to approximating the impulse response at the field

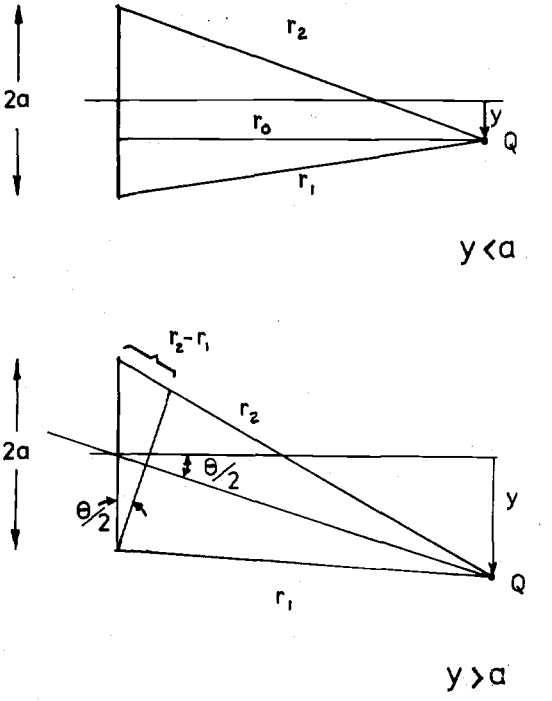

Fig. 1. Geometry to derive the travel times t_0, t_1, and t_2 from the radiator to the field point Q corresponding to distance r_0, r_1, and r_2, respectively, shown for the two field regions.

Manuscript received August 3, 1976; revised January 31, 1977.

The author is with the Ultrasonics Institute, Australian Department of Health, Sydney, N.S.W., Australia.