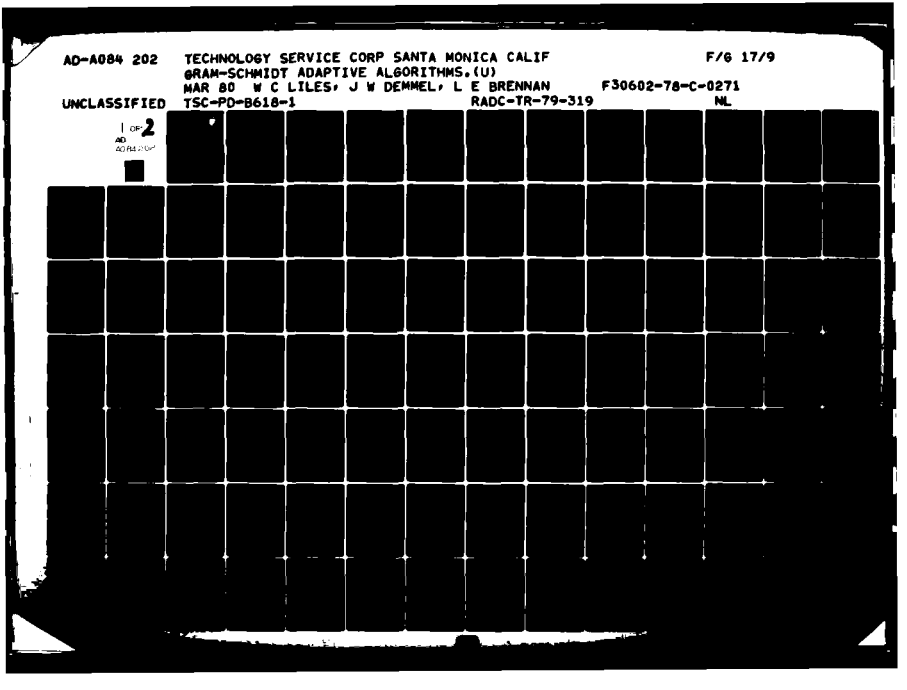
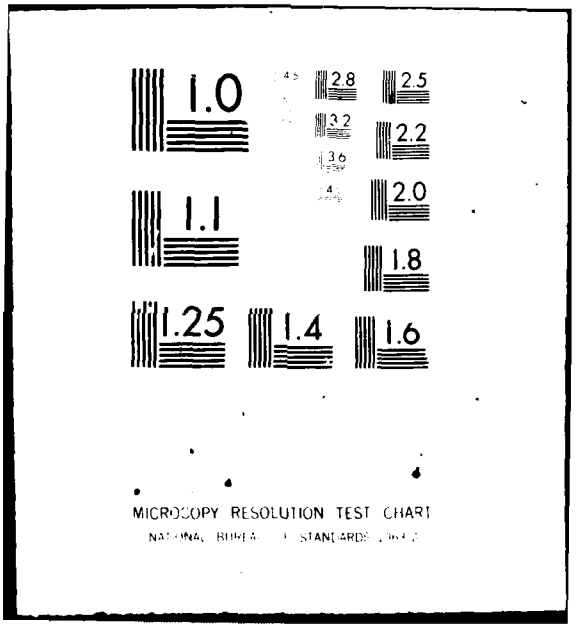
AD-A084 202 TECHNOLOGY SERVICE CORP SANTA MONICA CALIF F/G 17/9
GRAM-SCHMIDT ADAPTIVE ALGORITHMS.(U)
MAR 80 W C LILES, J W DEMMEL, L E BRENNAN F30602-78-C-0271
UNCLASSIFIED TSC-PD-B618-1 RADC-TR-79-319 NL




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

LEVEL II

RADC-TR-79-319
Final Technical Report
March 1980

GRAM-SCHMIDT ADAPTIVE ALGORITHMS

Technology Service Corporation

William C. Liles
James W. Demmel
Lawrence E. Brennan

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

GRAM-SCHMIDT ADAPTIVE ALGORITHMS

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-319 has been reviewed and is approved for publication.

APPROVED: Vincent C. Vannicola

VINCENT C. VANNICOLA
Project Engineer

APPROVED: Frank J Rehm

FRANK J. REHM
Technical Director
Surveillance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has been changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (OCTS), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-79-319	2. GOVT ACCESSION NO.: AD-A084202	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): GRAM-SCHMIDT ADAPTIVE ALGORITHMS	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, 8 Sep 78 — 8 Sep 79
	6. PERFORMING ORG. REPORT NUMBER: TSC-PD-B618-1
7. AUTHOR(s): William C. Liles, James W. Demmel, Lawrence E. Brennan	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0271
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Technology Service Corporation, 2811 Wilshire Boulevard, Santa Monica CA 90403	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 2305J902
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (OCTS), Griffiss AFB NY 13441	12. REPORT DATE: Mar 1980; 13. NUMBER OF PAGES: 188
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED; 15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Vincent C. Vannicola (OCTS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Universal adaptive algorithm (UAA)
Gram-Schmidt orthogonalization
Systems of linear equations
Parallel/pipeline architectures
Adaptive signal processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We consider solutions to the problem of computing adaptive weights in real time to maximize the signal-to-noise ratio of a multichannel input radar system with a large number of weights. The algorithms considered are based on the Gram-Schmidt orthogonalization process, and may be implemented on a modular multiprocessor computer using both parallel and pipeline techniques. The implementations are based on a simply and regularly connected processor array with simple and identical processors

(Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

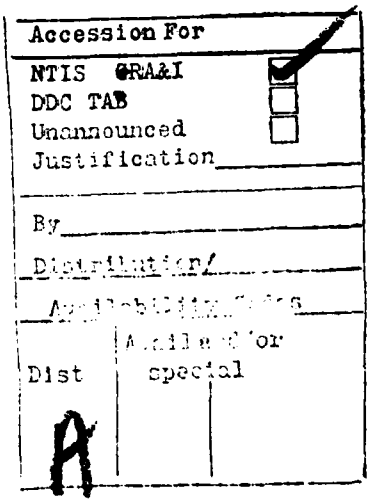
Item 20 (Cont'd)

(universal adaptive elements). The implementations can be made fast, fault tolerant, and easy to maintain.

EVALUATION

The significance of this contractual effort is the realization of generating a large number of adaptive weights simultaneously over more than one domain for radar systems. Such efforts will eventually render fully adaptive radar systems feasible. The effort supports RADC Technology Plan, TPO R4B, by providing a technical basis for adaptive radar operation in a hostile and interfering electromagnetic environment. The results derived herein will serve as a basis for using universally designed adaptive modules to solve the large array multidomain problems.

VINCENT C. VANNICOLA
Project Engineer



CONTENTS

CONTENTS (Cont'd)

Appendix

LIST OF FIGURES

Figure

LIST OF FIGURES (Cont'd)

Figure

LIST OF FIGURES (Cont'd)

Figure

LIST OF TABLES

Table

1. INTRODUCTION

1.1 STATEMENT OF THE PROBLEM

The objective of this study, conducted under Contract No. F30602-78-C-0271, is to determine a modular architecture for a multiprocessor system to compute adaptive weights to maximize the signal-to-noise ratio (SNR) of a multichannel input radar system with a large number of weights. The radar system to be considered has channel inputs in all three signal domains--spatial, temporal, and polarization--with multiple inputs in each domain. The adaptive system must sense the environment and adjust at least 200 complex weights imposed on the array elements, on the delay taps off each element, and on the polarization sensors, thus maximizing the signal-to-noise ratio. This sensing must be accomplished on a time-varying basis, as the environment dictates, with minimum convergence time, minimum noise, and maximum stability.

For a multiprocessor to be able to calculate at least 200 complex weights quickly enough to adapt to the changing environment, not only must a fast adaptive algorithm and a fast processor system be chosen, but the algorithm chosen must itself be well suited to the processor system. The algorithms to be explored are based on the Gram-Schmidt orthogonalization process. A modular architecture to support these algorithms is designed.

1.2 RESTATEMENT OF THE PROBLEM IN MATHEMATICAL TERMS

Each antenna element in an adaptive array produces an input signal, which may be written

$$X_j(t) = N_j(t) + S_j(t), \quad j=1,\ldots,n \tag{1}$$

where X is the complex waveform as a function of time t, N is the noise component, S is the signal component, the subscript j designates the particular antenna element, and n is the total number of antenna elements. X, N, and S may be thought of as complex column vectors of their components:

$$X(t) = \begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_n(t) \end{bmatrix} \quad . \tag{2}$$

Instead of these continuous waveforms, we will usually deal with the sample vectors $X^{(i)}$, $N^{(i)}$, and $S^{(i)}$, defined by:

$$X^{(i)} = X(t_i) \quad , \tag{3}$$

where the t_i are closely spaced points in time. Let W be a column vector of complex numbers. We form the filter function

$$F = W^*X \tag{4}$$

as the dot product of W and the input vector X (where W* = W conjugate transpose).

Our problem is to choose the weights W to optimize the SNR of the system, and thereby maximize the probability of detection of the signal in the presence of jammers and clutter. These weights are a function of the sequence of sample input vectors $X^{(i)}$. This report explores not only different algorithms for calculating the weights W through orthogonalizing the inputs $X^{(i)}$ and filter function F, but also the effects of different

modular parallel computer architectures on the implementations of these algorithms. These effects include execution speed, accuracy, convergence rates, and fault tolerance.

1.3 APPROACH

Let N be the column vector of noise inputs as, as in Section 1.2. Assuming the n components N_i have a multivariate Gaussian distribution, we can write their covariance matrix as

$$M = E(NN^*) \quad . \tag{5}$$

It has been shown [Brennan and Reed 1973]* that the optimal weights W are given by

$$W = kM^{-1}S \quad , \tag{6}$$

where S is the expected signal vector (called steering vector) and k is any nonzero constant (normally 1).

M is not known, but must be estimated from sample data

$$\hat{M} = \frac{1}{S} \sum_{j=1}^{S} X^{(j)} X^{(j)*} \quad , \tag{7}$$

*L. E. Brennan and I. S. Reed, "Theory of Adaptive Radar," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-9, No. 2, 1973, pp. 237-252.

where $X^{(j)}$ are sample voltage vectors and S is the number of samples. $\hat{M}$ is an n x n dense positive definite Hermitian matrix [Liles and Demmel 1978],* and so forming $\hat{M}$ as in Eq. (7) and then computing W from Eq. (6) with $\hat{M}$ in place of M takes a great deal of computation.

Now let T be a given linear transformation. Then the covariance matrix M_T of the random vector TN is given by

$$M_T + E[TN(TN)^*] = TE(NN^*)T^* = TMT^* \quad . \tag{8}$$

Thus, we have

$$W = M^{-1}S = T^*M_T^{-1}TS \quad . \tag{9}$$

If T can be chosen so that TS is easily computable, and M_T is diagonal, we will have simplified our problem. Such a T is given by the Gram-Schmidt orthogonalization process and produces a lower triangular matrix T such that the entries of the random vector TX are orthogonal.

The basic implementation of the processor for computing T and TX is shown in Figure 1. Each processor is very simple, and all are identical. The array operates with each row working in parallel on an intermediate step of TX, and with different rows working as stages in a pipeline to compute TX for different X's. The array overlaps its updates of its own internal coefficients w_{ij} with computing TX for different values of X.

*W. C. Liles and J. Demmel, "Solving Large Positive Definite Hermitian Linear Systems Utilizing Parallel/Pipeline Processors," Proceedings of the 1978 International Conference on Parallel Processing, IEEE, 1978, pp. 261-262.

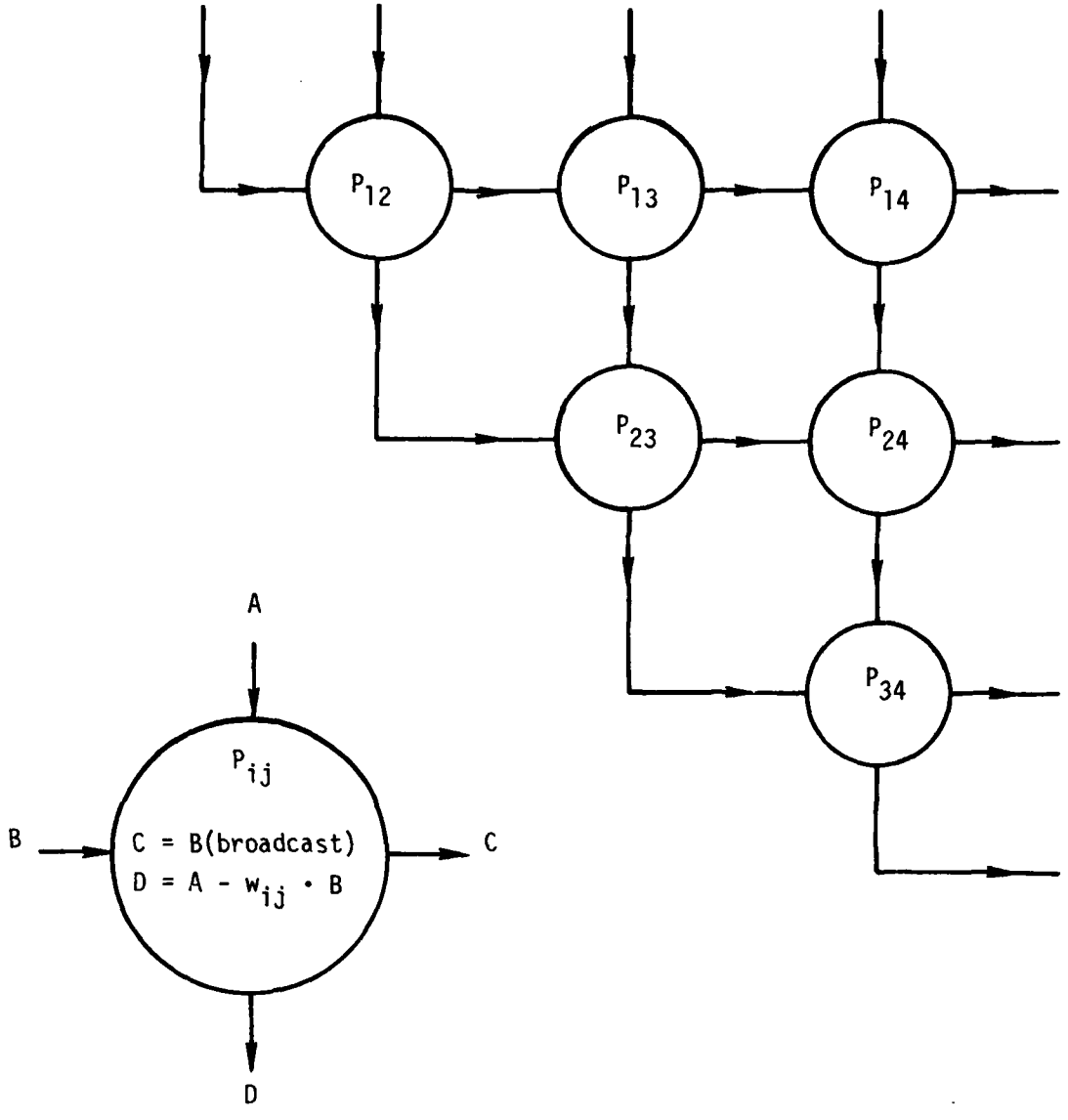


Figure 1. Basic Gram-Schmidt array for n = 4.
Arrows indicate direction of data flow.

Our approach is to analyze this basic scheme and its variations with respect to the following factors:

1. Variations on the basic mathematical algorithm, Section 2. We consider their different hardware implementations (e.g., cost and reliability properties) and stability (e.g., the number of bits required for intermediate steps).

2. Different ways of computing the internal coefficients, w_{ij}, Section 3.1. The different algorithms are analyzed with respect to their hardware implementations, number of samples required for convergence, and sensitivity to the number of jammers. Simulations were performed.

3. Error sensitivity and fault tolerance, Section 3.2. Both theoretical and simulation results are presented on the effects of catastrophic and gradual failures of processors and receiver/antenna systems.

4. Different ways to perform arithmetic, Section 3.3. We consider both fixed- and floating-point implementations, and try to derive probabilistic bounds on the growth of intermediate results during the computations.

5. Implementation with fewer processors, Section 3.4. We consider speed, cost, and reliability tradeoffs when using only n processors instead of $O(n^2)$ implementations of Figure 1.

6. Individual processor constructions, Section 3.5. The different ways of implementing each processor P_{ij} are considered with respect to speed and hardware complexity.

7. Radar engineering considerations, Section 3.6. We indicate how different system parameters might affect an engineer's decision of what version of the Gram-Schmidt algorithm to implement.

Finally, we select a typical set of radar system parameters and give a detailed hardware design for a Gram-Schmidt processor incorporating those parameters (see Section 3.7).

2. DESCRIPTION OF THE ALGORITHM

2.1 INTRODUCTION

An n-element radar system has inputs X_i, $1 \leq i \leq n$, where each component X_i is the sum of noise N_i and signal S_i, expressed in column vector notation as

$$X = N + S \quad . \tag{10}$$

The output of the system is the filter function, F, given by

$$F = W^*X \quad , \tag{11}$$

where W is a column vector of complex weights and * denotes conjugate transpose. Brennan and Reed [1973] have established that in order to maximize the signal-to-noise ratio (SNR) of the system, W should satisfy

$$MW = S \quad , \tag{12}$$

where M is the covariance matrix of the noise

$$M_{ij} = E(\overline{N}_i N_j) \quad . \tag{13}$$

In a real system, M is not known and must be estimated from incoming samples X. This section describes an algorithm based on Gram-Schmidt (G-S) orthogonalization to form M, solve the system of simultaneous linear equations determining W, (Eq. (12)), and compute F.

The literature contains both practical and highly theoretical results on solving systems of equations. Csanky's algorithm [1976]* requires the

*L. Csanky, "Fast Parallel Matrix Inversion Algorithms," SIAM J. on Computing, Vol. 5, 1976, pp. 618-623.

least number of operations of any algorithm [$O(\log^2 n)$] but is impractical because it requires too many processors connected in a complicated way, and is also numerically unstable. Gentleman [1978]* has given bounds on the time spent routing data between processors during Gaussian Elimination. Several investigators have benchmarked various algorithms on different machines [Calahan et al. 1976; Liles and Demmel 1978; Blakely 1977; Berra and Singhania 1976; Sameh and Kuck 1975].** Sameh and Kuck [1975, 1977a, 1977b, 1978]*** have published survey articles. Kung and Leiserson [1978]† have produced algorithms emphasizing simple processors and simple interconnect structures.

The matrix M in our problem has two very nice properties which make solving Eq.(12) easier than in the case of a general matrix: It is Hermitian (i.e.,

---

*W. M. Gentleman, "Some Complexity Results for Matrix Computations on Parallel Processors," Journal of the Association for Computing Machinery, Vol. 25, No. 1, January 1978, pp. 112-115.

**D. A. Calahan, W. N. Joy, and D. A. Orbits, Preliminary Report on Results of Matrix Benchmarks on Vector Processors, Systems Engineering Laboratory, SEL Report No. 94, University of Michigan, Ann Arbor, May 24, 1976; C. Blakely, "PEPE Application to BMD Systems," Proceedings of the 1977 International Conference on Parallel Processing, August 26-27, 1977, pp. 193-198; P. B. Berra and A. K. Singhania, Timing Figures for Inverting Large Matrices Containing Complex Numbers Using the Staran Associative Processor, Rome Air Development Center, RADC-TR-76-339, Griffiss Air Force Base, New York, November 1976; A. H. Sameh and D. J. Kuck, Linear System Solvers for Parallel Computers, Department of Computer Sciences, Report No. UIUCDCS-R-75-701, University of Illinois at Urbana-Champaign, February 1975.

***A. H. Sameh, "Numerical Parallel Algorithms--A Survey," High Speed Computer and Algorithm Organization, Academic Press, 1977a, pp. 207-228; A. H. Sameh and D. J. Kuck, "Parallel Direct Linear System Solvers - A Survey," Parallel Computers - Parallel Mathematics, North Holland, 1977b, pp. 25-30; A. H. Sameh and D. J. Kuck, "On Stable Parallel Linear System Solvers," Journal of the Association for Computing Machinery, Vol. 25, No. 1, January 1978, pp. 81-91.

†H. T. Kung and C. E. Leiserson, Algorithms for VLSI Processor Arrays, Department of Computer Sciences, Carnegie-Mellon University, 1978.

M = M*) and positive definite (i.e., all its eigenvalues are positive, or, equivalently, Z*MZ > 0 for all nonzero vectors Z) [Liles and Demmel 1978].

Positive definite Hermitian matrices arise frequently in numerical work, often implicitly as the sample covariance matrix of a set of sample vectors. This is the situation not only in the adaptive signal processing problem which motivated this study, but in many regression and least squares problems. For example, solving the system AX = B in a least squares sense is equivalent to solving the normal equations (A*A)X = A*B, where A*A is positive definite and Hermitian (if A has full rank and where A* = A conjugate transpose). Positive definite Hermitian matrices also arise explicitly in variational problems, such as solving self-adjoint differential equations with finite element methods.

The remainder of this section is organized as follows: Section 2.2 gives a statement of the problem and the approach to the problem; Section 2.3 presents an implementation of TS; Section 2.4 computes the w_{ij}'s; Section 2.5 gives an implementation of $M_T^{-1}(TS)$; Section 2.6 presents an implementation of $T^*(M_T^{-1}TS)$; and Section 2.7 describes possible simplification of the implementation in the case of a side-lobe canceller (SLC). Appendix C presents a numerical example of how to use the G-S array to solve a system of simultaneous linear equations.

2.2 STATEMENT OF THE PROBLEM AND APPROACH

Let the system to be solved be $MW = S$, where M is a positive definite Hermitian n x n matrix, S is an n x 1 vector, and W is an n x 1 vector of unknowns. If T is a nonsingular n x n matrix (to be specified later), define $M_T = TMT^*$, where $T^* = (\overline{T})^T$. Then M_T is also positive definite Hermitian (by Sylvester's law of inertia) and $W = M^{-1}S = T^*M_T^{-1}TS$. By choosing T so that T, M_T^{-1}, and T^* are easily computable, we will have simplified our problem. A good choice of T is given by the Gram-Schmidt orthogonalization process set forth in Eq. (14) [Rice 1966]*, where $\langle a, b\rangle_M$ denotes the complex inner product induced by the matrix M: $\langle a, b\rangle_M$ denotes the complex inner product induced by the matrix M: $\langle a, b\rangle_M \equiv b^*Ma$,

$$\left.\begin{aligned} Z_j^1 &\equiv S_j \qquad \text{(input)} \\ Z_j^{i+1} &= Z_j^i - \frac{\langle Z_j^i, Z_i^i\rangle_M}{\langle Z_i^i, Z_i^i\rangle_M} Z_i^i, \qquad 1 \le i \le n-1, \quad i < j \le n \\ Z_j &\equiv Z_j^j \qquad \text{(output: } Z = TS\text{)} \end{aligned}\right\} \quad . \tag{14}$$

The coefficients $w_{ij} = \langle Z_i^i, Z_i^i\rangle_M / \langle Z_i^i, Z_i^i\rangle_M$ are obtained by performing Gram-Schmidt orthogonalization on the standard basis e^j, $1 \le j \le n$, where $e_i^j = \{1$ if $i = j$ and 0 otherwise$\}$ with the inner product $\langle . , .\rangle_M$. Hence, the vectors e^j are orthogonal with respect to the inner product induced by M_T, which implies M_T is a diagonal matrix, since $\langle e^i, e^i\rangle_{M_T} = M_{T_{ij}} = 0$ unless $i = j$. Also, $M_{T_{ii}} = \langle e^i, e^i\rangle_{M_T} = \langle Z_i, Z_i\rangle_M$.

*J. R. Rice, "Experiments on Gram-Schmidt Orthogonalization," Math. Comput., 20 April 1966, pp. 325-328.

The T induced by Eq. (14) is unit lower triangular (lower triangular with ones on the diagonal), since $Z_j = S_j$ plus a linear combination of S_1 through S_{j-1}. Hence T^{-1} is also unit lower triangular, and both T* and $(T^*)^{-1}$ are unit upper triangular. Since $M = T^{-1}M_T(T^{-1})^*$, we have written M as the product of a unit lower triangular matrix $L = T^{-1}$, a diagonal matrix $D = M_T$, and $L^* = (T^{-1})^*$; $M = LDL^*$. This is the same decomposition provided by the Cholesky algorithm (without square roots [Liles and Demmel 1978]), and because the Cholesky decomposition is unique, (T^{-1}) and M_T are the matrices that would be produced by the Cholesky algorithm. In fact, we show in Subsection 2.6.2 that the coefficients w_{ij} are the entries of the matrix L.

Viewing M as the covariance matrix of a column vector V of random variables with a multivariate normal distribution with zero means (which is computed as the average of the covariances of s sample vectors V^k, $1 \leq k \leq s$: $M_{ij} = 1/s \sum_{k=1}^{s} \bar{V}_i^k V_j^k = E(\bar{V}_i V_j))$, M_T is then the covariance matrix of the random vector Z = TV:

$$
\begin{aligned}
M_T &= E(ZZ^*) \\
&= E[TV(TV)^*] \\
&= E(TVV^*T^*) \\
&= TE(VV^*)T^* \text{ since T is constant} \\
&= TMT^*
\end{aligned}
$$

Thus M_T diagonal means Z_i and Z_j are uncorrelated (orthogonal). This statistical interpretation helps us recognize that this method may be used to solve least squares and regression problems.

The variation on Eq. (14) given in Eq. (15) uses a nonunit lower triangular matrix T and corresponds to the Cholesky decomposition M = LL*. This variation

has a different implementation and different round off and stability properties, which will be discussed later.

$$
\begin{aligned}
& Z'^{1}_{j} \equiv S_j \qquad \text{(input)} \\
& \left\{ \begin{array}{l} Z^{i}_{i} = Z'^{i}_{i} / (\langle Z'^{i}_{i}, Z'^{i}_{i} \rangle_M)^{1/2} \\ Z'^{i+1}_{j} = Z'^{i}_{j} - \langle Z'^{i}_{j}, Z^{i}_{i} \rangle_M Z^{i}_{i} \end{array} \right\} \quad 1 \le i \le n,\ i < j \le n \\
& Z_j \equiv Z^{j}_{j} \qquad \text{(output: } Z = TS\text{)}
\end{aligned}
\qquad (15)
$$

Because of the normalization in the second line of Eq. (15), $M_{T_{ii}} = \langle Z_i, Z_i \rangle_M = 1$. Hence, $M_T = I$ = identity matrix. Writing $L = T^{-1}$ as before, we now have $M = LL^*$, the Cholesky decomposition (with square roots). As before, the coefficients $w'_{ij} = \{\langle Z'^{i}_{j}, Z^{i}_{i} \rangle$ if $i \neq j$; $1/(\langle Z'^{i}_{i}, Z'^{i}_{i} \rangle)^{1/2}$ if $i=j\}$ are the entries of the Cholesky factor L. The coefficients from Eq. (15) are related to the coefficients from Eq. (14) by $w_{ij} = w'_{ij} \cdot w'_{ii}$ for $i \neq j$. If V is a column vector of random variables as before, then the entries of $Z = TV$ are independent Gaussian random variables with unit variances.

In summary, the algorithm to solve $MW = S$ is:

1. Compute the w_{ij}'s, either directly from the entries of the M matrix, or from sample vectors.
2. Compute M_T^{-1}.
3. Compute TS .
4. Compute $M_T^{-1}(TS)$.
5. Compute $T^*(M_T^{-1}TS)$.

2.3 IMPLEMENTATION OF TS

Our implementation of TS is shown in Figure 2, for n = 4. Arrows indicate directions of data flow. As in Eq. (14) (Section 2.2),

$$w_{ij} = \langle Z_j^i, Z_i^i \rangle_M / \langle Z_i^i, Z_i^i \rangle_M \quad ,$$

and processor P_{ij} performs the computation

$$Z_j^{i+1} = Z_j^i - w_{ij} Z_i^i$$

The $n(n-1)/2$ processors operate as follows: The processors in row i operate simultaneously on values passed to them by the row above. Z_j^{i+1} is passed vertically from P_{ij} to $P_{i+1,j}$, and the rightmost processor $P_{i,i+1}$ ("diagonal" processor) broadcasts Z_{i+1}^{i+1} to all the processors in the next row simultaneously. All processors operate simultaneously and in lockstep, each row performing its operations on the intermediate results of a different input vector in one unit time step. Different S vectors may be piped into the top of the array, and a new vector entered each unit time step.

A timing diagram is shown in Figure 3 for the case n=4. Superscripts indicate different vectors; subscripts, different components. The horizontal axis is labelled in unit time steps. The top four rows indicate when data should be submitted to the input ports at the top of the array, and the bottom four rows indicate when the components of the output vectors $Z^j = TS^j$ are available at the output ports at the right of the array. One can see that the m products $Z^j = TS^j$, $1 \leq j \leq m$, can be formed in n+m time steps.

This implementation has the following advantages: It has a simple interconnect structure, with unidirectional and fixed-destination data flow; and the processors are very simple, identical, and independent of n.

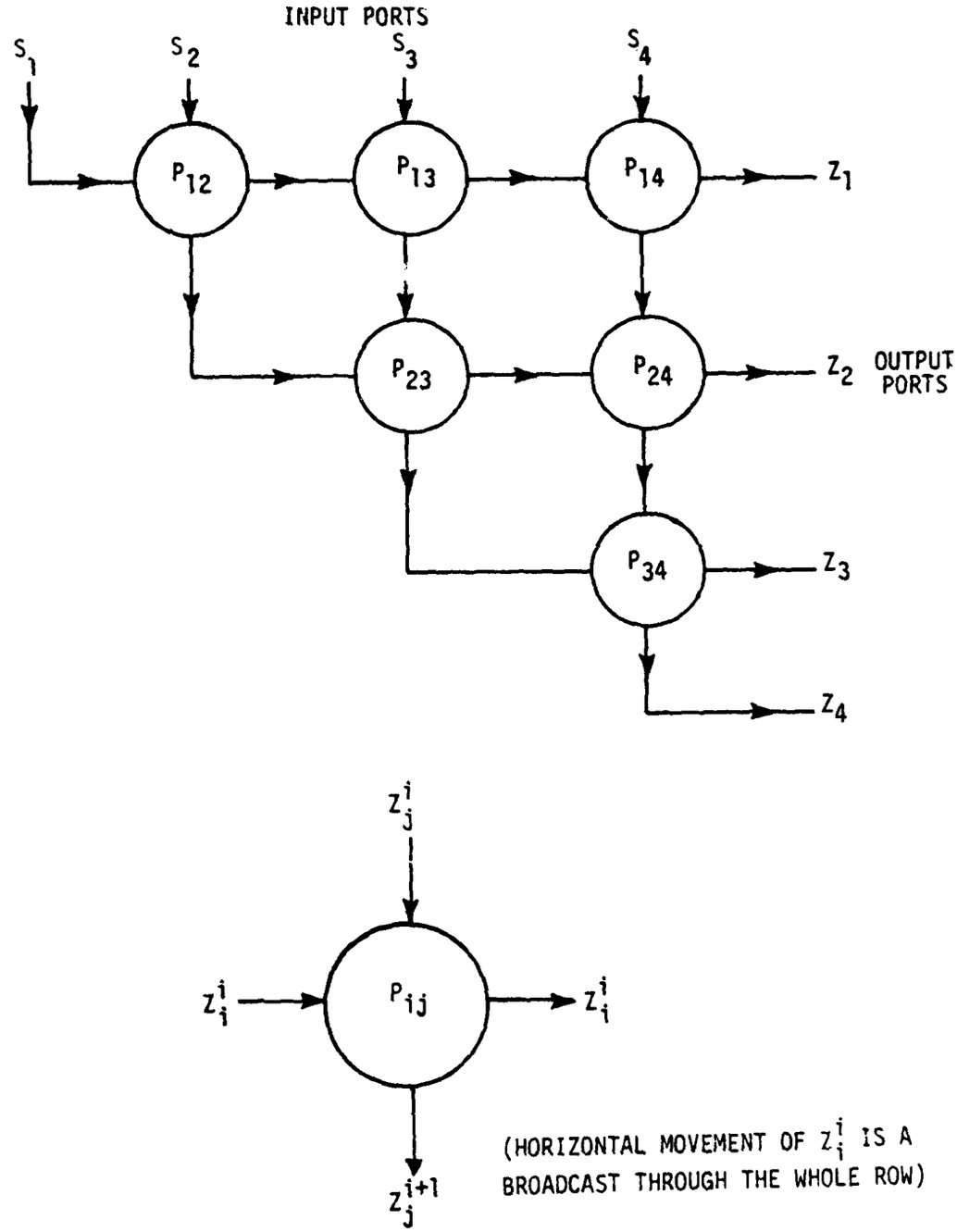


Figure 2. Processor array for computing TS.

	Time									
	1	2	3	4	5	6	7	8	9	10
Inputs										
Port No. 1	s_1^1	s_1^2	s_1^3	s_1^4	s_1^5	–				
Port No. 2	s_2^1	s_2^2	s_2^3	s_2^4	s_2^5	–				
Port No. 3	s_3^1	s_3^2	s_3^3	s_3^4	s_3^5	–				
Port No. 4	s_4^1	s_4^2	s_4^3	s_4^4	s_4^5	–				
Outputs										
Port No. 1		z_1^1	z_1^2	z_1^3	z_1^4	z_1^5	–			
Port No. 2			z_2^1	z_2^2	z_2^3	z_2^4	z_2^5	–		
Port No. 3				z_3^1	z_3^2	z_3^3	z_3^4	z_3^5	–	
Port No. 4					z_4^1	z_4^2	z_4^3	z_4^4	z_4^5	–

Figure 3. Timing diagram for computing TS.

Also, the processor array works without modification on vectors of size less than n.

In the case of Eq.(15) (with square roots) of Section 2.2, the only difference in implementation is that there must be a processor to divide Z'^{i}_{i} by its norm $(\langle Z'^{i}_{i}, Z'^{i}_{i} \rangle_{M})^{\frac{1}{2}}$ before broadcasting it.

2.4 COMPUTATION OF THE w_{ij}'s

The w_{ij}'s are computed in two ways, depending on whether the matrix M is known or must be estimated from vectors sampled from the underlying distribution. It turns out that the two methods require similar hardware. We first discuss the algorithm without square roots (Eq. (14)).

If the matrix M is available, the w_{ij}'s are computed by pipelining the columns of M into the top of the processor array, and performing the computations indicated in Figure 4. $Z_j(k)$ denotes the intermediate result from the k^{th} column of the matrix M. The IF-THEN-ELSE in Figure 4 may be implemented by having a control line for each row of the array: During normal operation (computing TS, or the ELSE clause in Figure 4) the control signal is off, and when the data from column i has finally reached row i in the array, the control signal should be turned on to tell the processors to compute w_{ij}. A timing diagram for the complete implementation appears later, in Figures 7 and 9 (Section 2.6).

The configuration just described requires a divide unit in each processor. Because each processor in row i computes the same reciprocal $1./Z_i(i)$, the diagonal processor, $P_{i-1,i}$, can also compute this value and broadcast it along with $Z_i^i(i)$ itself. This approach simplifies all but n - 1 of the processors, but requires more data to be broadcast.

Either configuration can compute all the w_{ij}'s in $n+1$ unit time steps.

If sample vectors are known instead of the actual matrix, the method is similar. If s sample vectors are given, all s must be passed through the array $n-1$ times, performing the computations shown in Figure 5a. Each time the s samples are passed through, another row of w_{ij}'s is calculated.

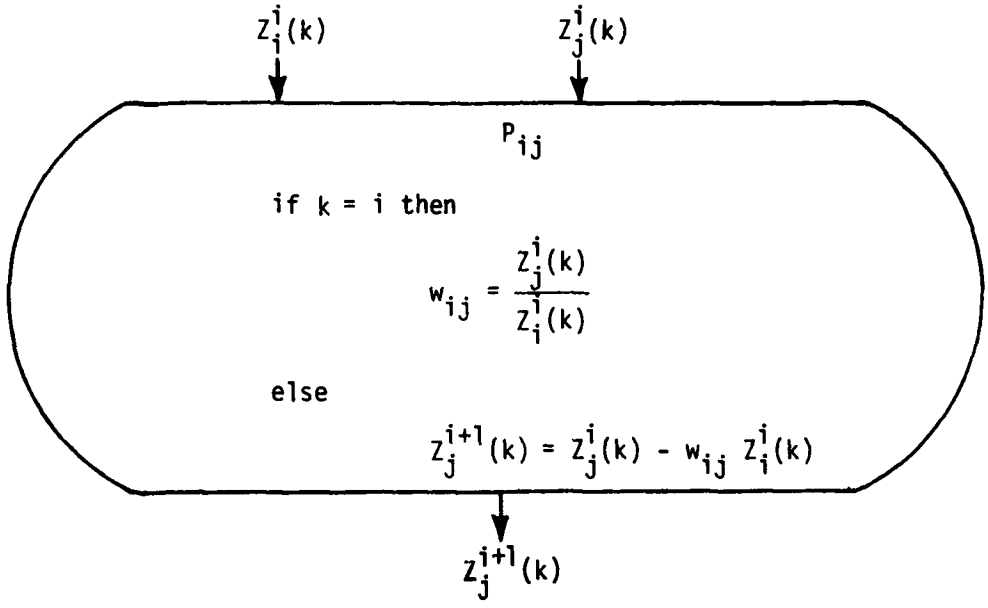


Figure 4a. Calculating w_{ij} with the matrix M as input (without square roots).

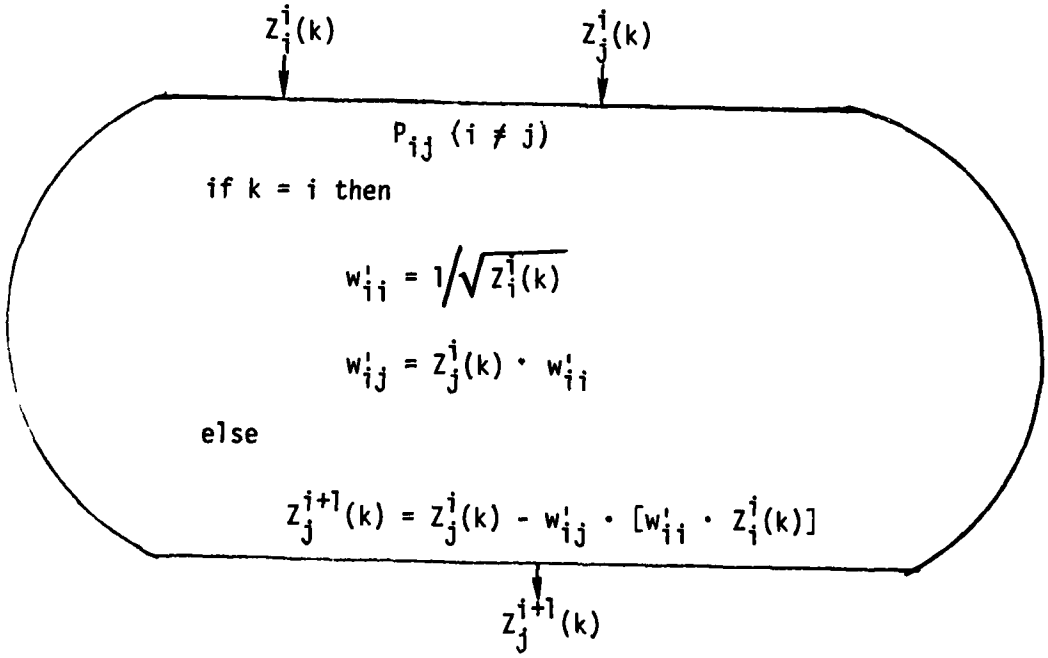


Figure 4b. Calculating w'_{ij} with the matrix M as input (with square roots).

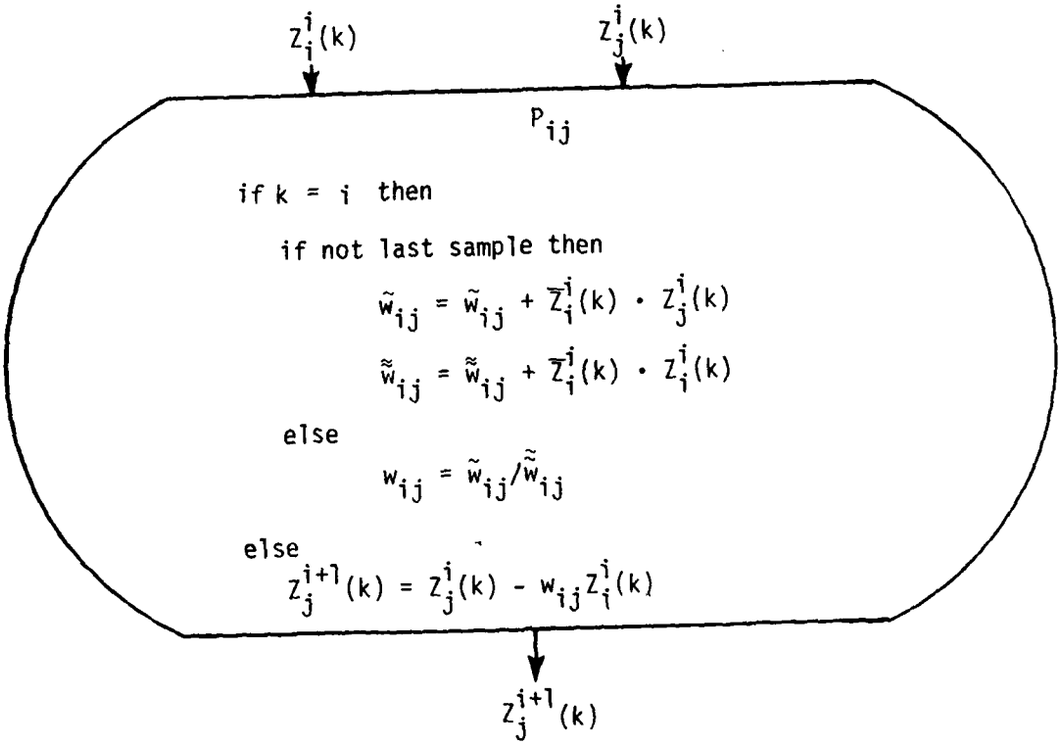


Figure 5a. Calculating w_{ij} with sample vectors as input (without square roots).

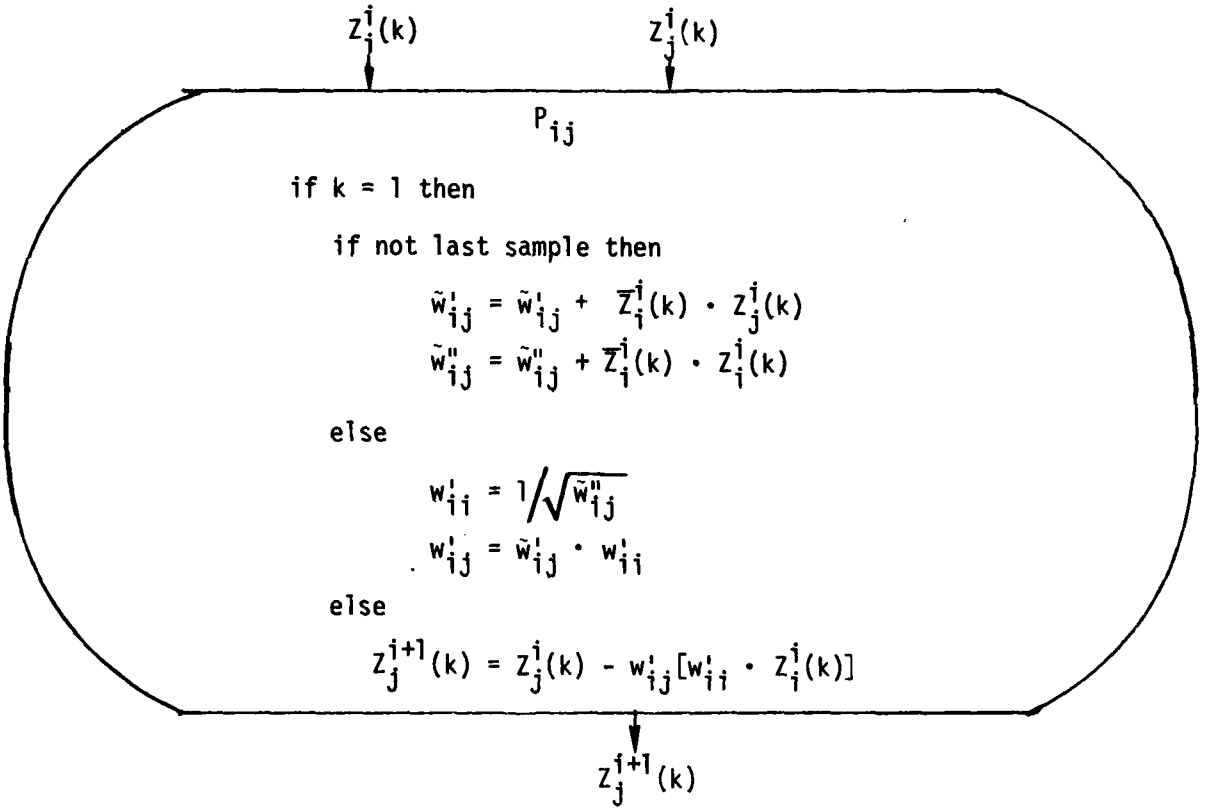


Figure 5b. Calculating w'_{ij} with sample vectors as input (with square roots).

In the expression $Z_j^i(k)$ the k denotes which row of w_{ij}'s is being calculated. A total of $[(s + 1)(n - 1) + 1]$ time steps are required to compute all the w_{ij}'s.

As in the case where the matrix M was input, an alternate configuration is to have only the diagonal processor compute $\tilde{\tilde{w}}_{ij} = \tilde{\tilde{w}}_{ij} + \bar{Z}_j^i(k) \cdot Z_i^i(k)$ and $1/\tilde{\tilde{w}}_{ij}$ and broadcast $1/\tilde{\tilde{w}}_{ij}$

The implementation of the algorithm with square roots (Eq. (15)) is similar and is shown in Figures 4b and 5b.

Since each processor P_{ij} in row i needs the same quantity w'_{ii} (or its reciprocal), it is possible to have a diagonal processor P_{ii} to calculate this value only and broadcast it (along with $Z_i^i(k)$, in the case of Figure 5b). This possibility increases communication needs slightly but makes each P_{ij} much simpler.

Other methods exist for computing the w_{ij}'s when sample vectors are used; they are variations on the general scheme shown above and are discussed in Section 3.1.

2.5 IMPLEMENTATION OF $M_T^{-1}(TS)$

Because M_T is diagonal, $M_T^{-1} = \text{diag}(\langle Z_i^i, Z_i^i \rangle_M^{-1})$, and these values are available in any processor ($1./\langle Z_i^i, Z_i^i \rangle_M$ is in P_j^i for all j), it is easy to compute $M_T^{-1}(TS)$: Just attach a multiply unit to each output port on the right of the array and multiply the output at port i by the value of $\langle Z_i^i, Z_i^i \rangle_M^{-1}$ borrowed from the processor just to the left, P_n^i. When the algorithm with square roots is used, M_T is the identity matrix and so nothing needs to be done.

2.6 IMPLEMENTATION OF $T^*(M_T^{-1}TS)$

This step of the algorithm, the formation of $T^*(M_T^{-1}TS)$, has three different implementations. The first, the unit vector method, uses the array shown in Figure 2 (Section 2.3) with n additional processors on the right. It can solve the m problems $MW^i = S^i$, $1 \leq i \leq m$, in time O(mn) with high efficiency

$$E = \text{processor on-time}/(\text{total time} \cdot \text{total processors})$$
$$= (m/m+1)\ [(n+1)^2 - 2)/(n+1)^2] \quad .$$

The second method, the reverse flow method, requires some changes to the array but preserves the simple interconnect scheme. It can solve the m problems in time O(m+n) but with somewhat lower efficiency

$$E = m/[m + 2(n+1)] \quad .$$

The third method, the transform space method, is essentially different from the other two in its use of the array to compute the filter function, F, directly. The first two methods compute the weights, W, which must then be used (by another piece of hardware) to compute F.

Overall timings of the different implementations are given. A more detailed comparison of the unit vector and reverse flow methods is given in Appendix F.

2.6.1 Unit Vector Method

The unit vector method forms $T^*(M_T^{-1}TS)$ with a vertically connected array of n simple processors connected to the output ports of the array as in Figure 6. First, the vector S is passed through the array, and the n numbers $Y_j = \left(\overline{M_T^{-1}TS}\right)_j$ are formed and stored in registers inside the processing elements (PE's). Then, the n unit vector E^j ($E_i^j = 1$ if $i = j$ and 0 otherwise) are piped through the array to form the n products TE^j. Given that

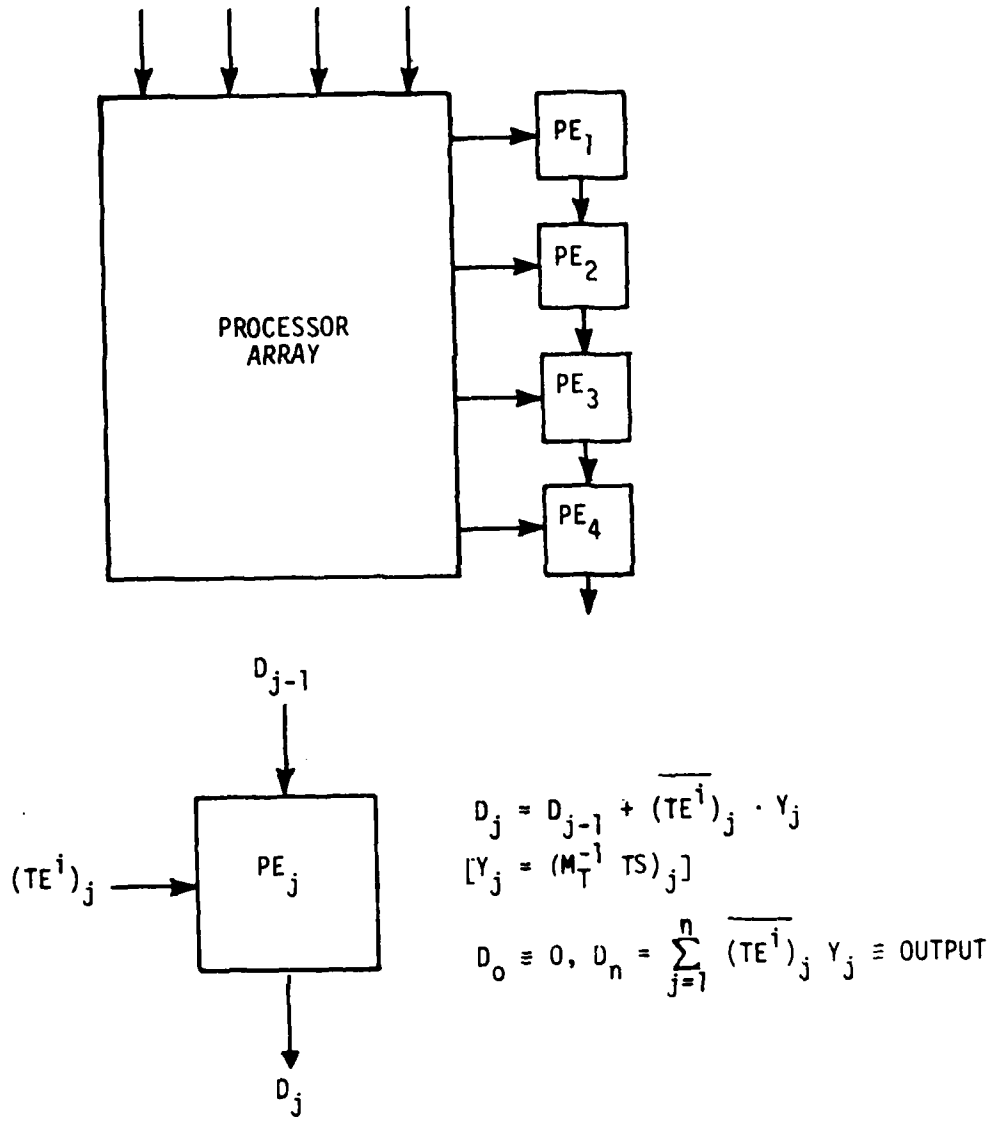


Figure 6. Unit vector method array.

$$(T*Y)_j = \sum_{k=1}^{n} \overline{T}_{kj} Y_k = \sum_{k=1}^{n} \overline{(TE^j)}_k Y_k$$

and that terms $(TE^j)_k$ and Y_k are both available at the k^{th} output port, these terms may be multiplied, added to a partial sum computed by PE_{k-1} during the preceding time step, and passed to PE_{k+1} for the next time step, as shown in Figure 6.

A timing diagram for the entire process of solving $MW^i = S^i$ for $n = 3$ and $1 \leq i \leq 2$ is shown in Figure 7. Y_j^i denotes $(M_T^{-1}TS^i)_j$, and D_j^i denotes the output of PE_j with input $(TE^i)_j$.

A timing analysis may be made easily from the timing diagram. A total of n time steps are required to compute the w_{ij}'s. Not counting these time steps, a total of $T_{UV} = (m+1)(n+1)$ units of time are required to pass m vectors S^i and mn unit vectors E^j through the array. We may compute the efficiency of the array where T_p is the time processor p is on, P is the number of processors, and T is total time required to finish processing, as follows:

$$E_{UV} = \sum_{\text{all processors}} T_p/(T*P) \quad .$$

Each of the $n(n-1)/2$ processors in the array has $T_p = m(n+1)$, and each of the n outboard PE's has $T_p = mn$; therefore,

$$E_{UV} = \left(\frac{m}{m+1}\right)\left[\frac{(n+1)^2 - 2}{(n+1)^2}\right] \quad .$$

Thus, E_{UV} approaches 1 as m and n grow. The unit vector method is thus an efficient implementation, and it is the same for both the with-square-roots and without-square-roots versions of the algorithm.

Time:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Array Input	#1	M_{11}	M_{12}	M_{13}	S_1^1	E_1^1	E_1^2	E_1^3	S_1^2	E_1^1	E_1^2	E_1^3				
	#2	M_{21}	M_{22}	M_{23}	S_2^1	E_2^1	E_2^2	E_2^3	S_2^2	E_2^1	E_2^2	E_2^3				
	#3	M_{31}	M_{32}	M_{33}	S_3^1	E_3^1	E_3^2	E_3^3	S_3^2	E_3^1	E_3^2	E_3^3				
Array Output	#1					Y_1^1	TE_1^1	TE_1^2	TE_1^3	Y_1^2	TE_1^1	TE_1^2	TE_1^3			
	#2						Y_2^1	TE_2^1	TE_2^2	TE_2^3	Y_2^2	TE_2^1	TE_2^2	TE_2^3		
	#3							Y_3^1	TE_3^1	TE_3^2	TE_3^3	Y_3^2	TE_3^1	TE_3^2	TE_3^3	
PE Output	#1							D_1^1	D_1^2	D_1^3		D_1^1	D_1^2	D_1^3		
	#2								D_2^1	D_2^2	D_2^3		D_2^1	D_2^2	D_2^3	
	#3									$\underbrace{D_3^1}_{(T*M_T^{-1}TS^1)}$	D_3^2	D_3^3		$\underbrace{D_3^1}_{(T*M_T^{-1}TS^2)}$	D_3^2	D_3^3

Figure 7. Timing diagram for the unit vector method.

2.6.2 Reverse Flow Method

The reverse flow method depends on the interesting fact that the array can compute T*Y by inputting the components of the vector Y at the right side of the array, performing all the computations in the reverse order (using $\bar{w}_{ij}$ instead of w_{ij}), and extracting the outputs at the top of the array. This process is illustrated in Figure 8. Now the broadcasts take place in the vertical direction, whereas when computing TS the broadcasts were in the horizontal direction.

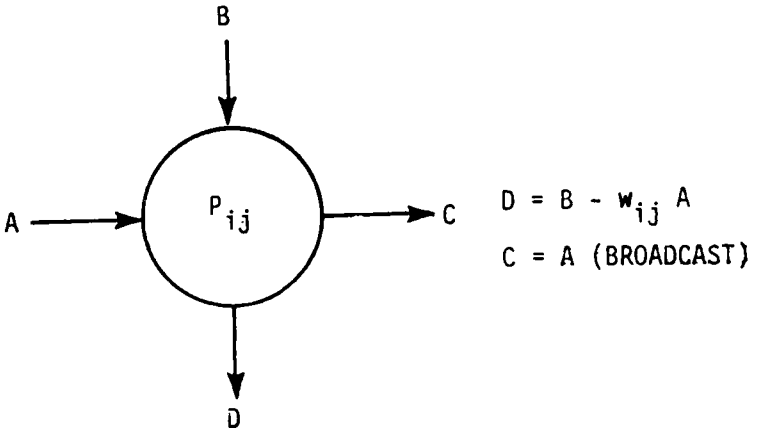
A proof of this fact for the algorithm without square roots (illustrated here for n = 4) depends on the fact that T can be written

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -w_{34} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -w_{23} & 1 & 0 \\ 0 & -w_{24} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -w_{12} & 1 & 0 & 0 \\ -w_{13} & 0 & 1 & 0 \\ -w_{14} & 0 & 0 & 1 \end{bmatrix}$$

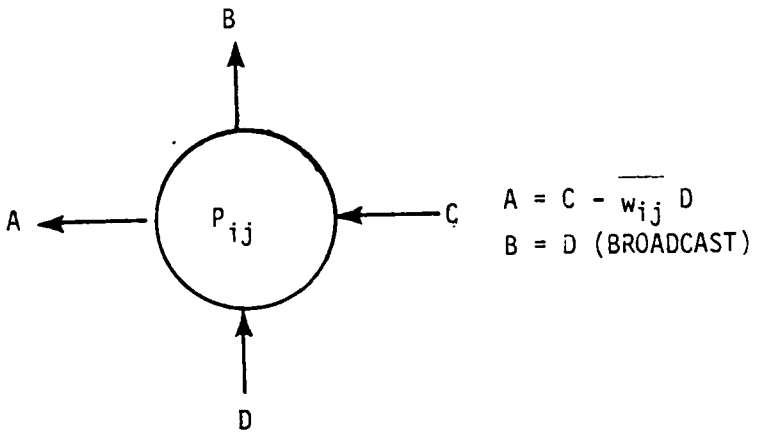
The i^{th} matrix in this product represents the computations performed by the $(n-i)^{th}$ row of the array. Hence,

$$T^* = [(T^*)^{-1}]^{-1} = [(T^{-1})^*]^{-1}$$

$$= \left[\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ w_{12} & 1 & 0 & 0 \\ w_{13} & 0 & 1 & 0 \\ w_{14} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & w_{23} & 1 & 0 \\ 0 & w_{24} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & w_{34} & 1 \end{bmatrix} \right)^* \right]^{-1}$$



DATA FLOW FOR COMPUTING T



DATA FLOW FOR COMPUTING T*

Figure 8. Reverse flow method.

$$= \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ w_{12} & 1 & 0 & 0 \\ w_{13} & w_{23} & 1 & 0 \\ w_{14} & w_{24} & w_{34} & 1 \end{bmatrix}^{*} \right)^{-1}$$

$$= \left(\begin{bmatrix} 1 & \bar{w}_{12} & \bar{w}_{13} & \bar{w}_{14} \\ 0 & 1 & \bar{w}_{23} & \bar{w}_{24} \\ 0 & 0 & 1 & \bar{w}_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1}$$

$$= \left(\begin{bmatrix} 1 & 0 & 0 & \bar{w}_{14} \\ 0 & 1 & 0 & \bar{w}_{24} \\ 0 & 0 & 1 & \bar{w}_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \bar{w}_{13} & 0 \\ 0 & 1 & \bar{w}_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \bar{w}_{12} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 1 & -\bar{w}_{12} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\bar{w}_{13} & 0 \\ 0 & 1 & -\bar{w}_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\bar{w}_{14} \\ 0 & 1 & 0 & -\bar{w}_{24} \\ 0 & 0 & 1 & -\bar{w}_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The i^{th} matrix in this product for T^* represents the computations performed by the $(n-i)^{th}$ column of the array during reverse flow.

The same type of argument shows that the w_{ij} are the entries of the matrix L in the Cholesky factorization $M = LDL^*$. Recall from Section 2.2 that $L = T^{-1}$

$$= \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -w_{34} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -w_{23} & 1 & 0 \\ 0 & -w_{24} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -w_{12} & 1 & 0 & 0 \\ -w_{13} & 0 & 1 & 0 \\ -w_{14} & 0 & 0 & 1 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ -w_{12} & 1 & 0 & 0 \\ -w_{13} & 0 & 1 & 0 \\ -w_{14} & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -w_{23} & 1 & 0 \\ 0 & -w_{24} & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -w_{34} & 1 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ w_{12} & 1 & 0 & 0 \\ w_{13} & 0 & 1 & 0 \\ w_{14} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & w_{23} & 1 & 0 \\ 0 & w_{24} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & w_{34} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ w_{12} & 1 & 0 & 0 \\ w_{13} & w_{23} & 1 & 0 \\ w_{14} & w_{24} & w_{34} & 1 \end{bmatrix}$$

The proof is similar for the algorithm with square roots:

$$T = \left(\left[\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & w'_{44} \end{matrix} \right] \left[\begin{matrix} 1 & & & \\ & 1 & & \\ & & w'_{33} & \\ & & -w'_{34} \cdot w'_{33} & 1 \end{matrix} \right] \left[\begin{matrix} 1 & & & \\ & w'_{22} & & \\ & -w'_{23} \cdot w'_{22} & 1 & \\ & -w'_{34} \cdot w'_{22} & & 1 \end{matrix} \right] \cdot \left[\begin{matrix} w'_{11} & & & \\ -w'_{12} \cdot w'_{11} & 1 & & \\ -w'_{13} \cdot w'_{11} & & 1 & \\ -w'_{14} \cdot w'_{11} & & & 1 \end{matrix} \right] \right)$$

$$T^* = [(T^{-1})^*]^{-1}$$

$$= \left[\left(\left[\begin{matrix} 1/w'_{11} & & & \\ w'_{12} & 1 & & \\ w'_{13} & & 1 & \\ w'_{14} & & & 1 \end{matrix} \right] \left[\begin{matrix} 1 & & & \\ & 1/w'_{22} & & \\ & w'_{23} & 1 & \\ & w'_{24} & & 1 \end{matrix} \right] \left[\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1/w'_{33} & \\ & & & w'_{34} \end{matrix} \right] \cdot \left[\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1/w'_{44} \end{matrix} \right] \right)^* \right]^{-1}$$

$$= \left(\left[\begin{matrix} 1/w'_{11} & & & \\ w'_{12} & 1/w'_{22} & & \\ w'_{13} & w'_{23} & 1/w'_{33} & \\ w'_{14} & w'_{24} & w'_{34} & 1/w'_{44} \end{matrix} \right]^* \right)^{-1}$$

$$= \begin{bmatrix} 1/w'_{11} & \bar{w}'_{12} & \bar{w}'_{13} & \bar{w}'_{14} \\ & 1/w'_{22} & \bar{w}'_{23} & \bar{w}'_{24} \\ & & 1/w'_{33} & \bar{w}'_{34} \\ & & & 1/w'_{44} \end{bmatrix}^{-1}$$

$$= \left(\begin{bmatrix} 1 & & & \bar{w}'_{14} \\ & 1 & & \bar{w}'_{24} \\ & & 1 & \bar{w}'_{34} \\ & & & 1/w'_{44} \end{bmatrix} \begin{bmatrix} 1 & & \bar{w}'_{13} & \\ & 1 & \bar{w}'_{23} & \\ & & 1/w'_{33} & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & \bar{w}'_{12} & & \\ & 1/w'_{22} & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1/w'_{11} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} w'_{11} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & -\bar{w}'_{12} \cdot w'_{22} & & \\ & w'_{22} & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & -\bar{w}'_{13} \cdot w'_{33} & \\ & 1 & -\bar{w}'_{23} \cdot w'_{33} & \\ & & w'_{33} & \\ & & & 1 \end{bmatrix}$$

$$\cdot \begin{bmatrix} 1 & & & -\bar{w}'_{14} \cdot w'_{44} \\ & 1 & & -\bar{w}'_{24} \cdot w'_{44} \\ & & 1 & -\bar{w}'_{34} \cdot w'_{44} \\ & & & w'_{44} \end{bmatrix} .$$

Note that the quantity being broadcast vertically in column j gets multiplied by w'_{jj} before broadcasting, since the processors in column j do not contain the number w'_{jj}. Also,

$$T^{-1} = \begin{bmatrix} 1/w'_{11} & & & \\ w'_{12} & 1 & & \\ w'_{13} & & 1 & \\ w'_{14} & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1/w'_{22} & & \\ & w'_{23} & 1 & \\ & w'_{24} & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1/w'_{33} & \\ & & w'_{34} & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1/w'_{44} \end{bmatrix}$$

$$= \begin{bmatrix} 1/w'_{11} & & & \\ w'_{12} & 1/w'_{22} & & \\ w'_{13} & w'_{23} & 1/w'_{33} & \\ w'_{14} & w'_{24} & w'_{34} & 1/w'_{44} \end{bmatrix} ,$$

so that the w_{ij}'s are the entries of the Cholesky factor L in $M = LL^*$, except that the w_{ii}'s are the reciprocals of the diagonal elements of L.

The same lockstep pipelined operation for data movement that was used for T is used for T*, except backwards. A timing diagram for the entire operation of solving $MW^i = S^i$ for $n = 3$ and $1 \leq i \leq 4$ is shown in Figure 9. A shift register of length $n-j$ is required at output port j in order to resubmit the components of the vector $M_T^{-1}TS$ to the array simultaneously. In Figure 9, Y^j denotes $M_T^{-1}TS^j$ and $W^j = M^{-1}S^j$, the solution.

As before, we analyze the timing excluding the time steps required to compute the w_{ij}'s. If m vectors S^i are to be input at the top, the total time required is m (to input the S^i's) plus n (to compute TS^i) plus 1 (to compute $M_T^{-1}TS^i$) plus 1 (to resubmit Y^i) plus n (to compute T*Y):

$$T_{RF} = m + 2(n + 1). = O(m + n) \quad .$$

This relation is to be contrasted with the $T_{UV} = (m + 1)(n + 1)$ for the unit vector method. Analyzing efficiency as before, but interpreting P_{ij} as two processors, one for T and one for T*, we have $n(n - 1)$ processors being used for m time units and n processors for m units, for an efficiency

$$E_{RF} = m/[m + 2(n + 1)] \quad .$$

This efficiency function has a behavior different from E_{UV} for the unit vector method: E_{RF} is low for small m and large n, whereas E_{UV} was an increasing

Time:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
input at top of array	#1	M_{11}	M_{12}	M_{13}	S_1^1	S_1^2	S_1^3	S_1^4	...							
	#2	M_{21}	M_{22}	M_{23}	S_2^1	S_2^2	S_2^3	S_2^4	...							
	#3	M_{31}	M_{32}	M_{33}	S_3^1	S_3^2	S_3^3	S_3^4	...							
output at right of array	#1					TS_1^1	TS_1^2	TS_1^3	TS_1^4	...						
	#2						TS_2^1	TS_2^2	TS_2^3	TS_2^4	...					
	#3							TS_3^1	TS_3^2	TS_3^3	TS_3^4	...				
$Y = M_T^{-1}TS$	#1						Y_1^1	Y_1^2	Y_1^3	Y_1^4	...					
	#2							Y_2^1	Y_2^2	Y_2^3	Y_2^4	...				
	#3								Y_3^1	Y_3^2	Y_3^3	Y_3^4	...			
input at right of array	#1									Y_1^1	Y_1^2	Y_1^3	Y_1^4	...		
	#2									Y_2^1	Y_2^2	Y_2^3	Y_2^4	...		
	#3									Y_3^1	Y_3^2	Y_3^3	Y_3^4	...		
output at top of array	#1												W_1^1	W_1^2	W_1^3	W_1^4 ...
	#2											W_2^1	W_2^2	W_2^3	W_2^4	...
	#3										W_3^1	W_3^2	W_3^3	W_3^4	...	

Figure 9. Timing diagram for the reverse flow method.

function of both m and n. This is a typical speed/efficiency design tradeoff often encountered when designing parallel systems [Chen 1971a; 1971b].*

2.6.3 Forming the Filter Function in Transform Space

The ultimate output of the system under consideration is the filter function

$$F = W^*X \quad ,$$

where X is a column vector of receiver inputs. The algorithm discussed in the preceding two subsections (2.6.1 and 2.6.2) computes W, assuming another piece of hardware is available to form the inner product W*X. It is possible to avoid computing W itself, and to form the filter function as follows:

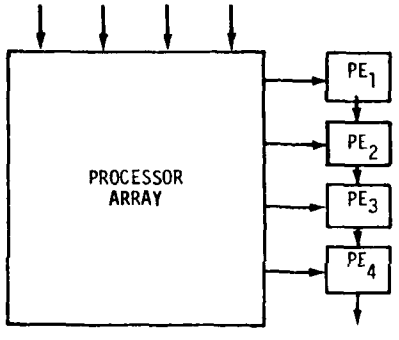
$$\begin{aligned} F &= W^*X \\ &= (M^{-1}S)^*X \\ &= S^*M^{-1}X \\ &= S^*T^*(T^*)^{-1}M^{-1}T^{-1}TX \\ &= (TS)^*(TMT^*)^{-1}TX \\ &= (TS)^*M_T^{-1}(TX) \end{aligned}$$

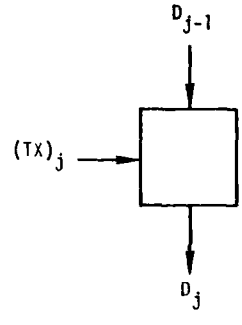
Recalling that M_T is a diagonal matrix, we see that the vector $(TS)^*M_T^{-1}$ may be computed once and stored, and then its inner product with TX computed.

The hardware required to implement this approach is virtually identical to that of the unit vector method. After the w_{ij}'s have been calculated

*T. C. Chen, "Unconventional Superspeed Computer Systems," Spring Joint Computer Conference, 1971a, pp. 365-366; T. C. Chen, "Parallelism, Pipelining, and Computer Efficiency," Computer Design, Vol. 10, 1971b, pp. 69-74.

(including the entries of M_T^{-1}), the steering vector S is passed through the array, and the values $[(TS)*M_T^{-1}]_i$ stored in a vertically connected array of processors to the right of the main array, as in Figure 10. Then, as the X vectors are pipelined into the top of the array, their results $(TX)_i$ will be available at PE_i where $[(TS)*M_T^{-1}]_i$ is stored, and inner products of the vectors will be formed just as in Figure 10. The timing diagram is similar to that in Figure 7, with the value of F = W*X being available from PE_n n+1 time steps after X is input.





$$D_j = D_{j-1} + (TX)_j \cdot Q_j$$

$$Q_j = [(TS)*M_T^{-1}]_j$$

$$D_o = 0$$

$$D_n = \sum_{j=1}^{n} Q_j (TX)_j$$

$$= (TS)*M_T^{-1}TX \equiv \text{OUTPUT}$$

Figure 10. Computations for forming the filter function in transform space.

This approach has two advantages over approaches described in the preceding two subsections. First, the array produces the desired output directly instead of just the weights W. The hardware is as simple or simpler than the unit vector method implementation and much simpler than the reverse flow method. Second, it is possible to continuously update the w_{ij}'s without temporarily halting computation of the weights; in this way, the w_{ij}'s are always computed from the most recent information. This point will be discussed further in Section 3.

The disadvantage of the transform space method is that in a real system, the X's may be sampled at a far faster rate than the array is able to process them. The X's are accepted one per time step of the array (and one F = W*X value is computed once per time step), whereas the sampling rate of the radar may be many times faster. (The unit vector method and reverse flow method do not have this disadvantage because, given W, F = W*X can be calculated extremely fast.) Whether this downsampling would adversely affect the overall system performance depends on the individual system.

2.7 THE SPECIAL CASE OF A SIDE-LOBE CANCELLER

It is possible to simplify our proposed implementation in the case of a side-lobe canceller (SLC), because of the special form of the steering signal (all zeroes, except for a one on the component corresponding to the main beam). It turns out that the best way to exploit this special form is to have the one in the steering signal occur in the last (n^{th}) component. With this S, TS = S (since T is unit lower triangular); thus, no work is required to pass S through the array.

The effect on the unit vector method is to reduce the sum defining the output,

$$(T^*Y)_j = \sum_{k=1}^{n} \overline{T}_{kj} Y_k = \sum_{k=1}^{n} (\overline{TE^j})_k Y_k \quad ;$$

to

$$W_j = (\overline{TE^j})_n M_{T_{nn}}^{-1} \quad ,$$

where $M_T = \text{diag}\,(MT_{jj})$. Thus, only one outboard processor is needed at output port n to compute W_j. The overall timing is the same.

The reverse flow method also simplifies because the vector to be passed backwards through the array is known a priori, $(0,\ldots,0,M_{T_{nn}}^{-1})^T$, eliminating the need for shift registers and reducing the overall length of the pipe by n steps.

The transform space method becomes exceedingly simple, with only one outboard processor required at output port n, and since

$$F = (TS)^* M_T^{-1}(TX) = (M_T^{-1} TX)_n \quad ,$$

its output is simply the filter function.

3. SYSTEM IMPLEMENTATION CONSIDERATIONS

This section describes the characteristics of different versions of the basic implementation discussed in Section 2, and various radar engineering considerations and how they affect the choice of implementation. We consider different ways of computing the internal coefficients (Section 3.1); error sensitivity and fault tolerance (Section 3.2); different ways to perform arithmetic and growth bounds for intermediate results (Section 3.3); implementation with $O(n)$ processors instead of $O(n^2)$ (Section 3.4); and construction of individual processors (Section 3.5). In Section 3.6 we summarize the variety of system implementation choices discussed in Sections 3.1 through 3.5, and show how different radar engineering system parameters affect implementation choice. Finally, in Section 3.7, we select a typical set of system parameters and give a detailed design for a Gram-Schmidt processor incorporating those parameters.

3.1 CALCULATION OF INNER PRODUCTS

The algorithm of Section 2.1 requires the calculation of many inner products $\langle Z_j^i, Z_i^i \rangle_M$. The method given, which computes the sample average $\sum_k Z_j^i(k)\overline{Z_i^i(k)}$ (see Figure 5), is one of several possible methods. All the methods involve averaging over samples, as in Figure 5, but can weight different samples differently:

$$E^m = \sum_{k=1}^{m} w_k E_k \quad . \tag{16}$$

Here E_k is one sample value (say, $Z_j^i(k)\overline{Z_i^i(k)}$), w_k is its weight, and E^m is the estimated average using m samples. For E^m to be an unbiased estimator of the true average, it is necessary that $w_k \geq 0$ and $\sum w_k = 1$. Subject to this

mild restriction, we are free to choose the w_k's so that our estimate E has desirable statistical properties, such as being able to closely follow the true average as it changes in time without being confused by noise spikes.

It is well known that if the underlying distribution from which the E_k's are sampled is stationary in time, then the best unbiased estimator of their mean is an equal weighting of all available samples:

$$E^m = \frac{1}{m} \sum_{k=1}^{m} E_k \quad . \tag{17}$$

When the underlying distribution is changing, as it will in any real system, then Eq. (17) introduces a bias, because it weights old samples as much as new ones. A good choice of w_k's must satisfy two conditions: It must average over enough samples to compute a statistically significant result (at least n samples (n=number of weights) are required just for the matrix M to be nonsingular), and it must weight the recent values heavily enough to follow the average quickly, but not so heavily that noise spikes confuse it.

The optimal choice of an averaging method probably would vary from system to system, but there are certain schemes which work well in many situations and are worth analyzing. The methods have different implementations, depending on the rest of the system (e.g., implementations of Sections 2.6.2 and 2.6.1 versus that of Section 2.6.3). After the discussion of each method, therefore, we analyze various implementations. Finally, we present test results using actual radar data. These results allow us to compare how well the methods maximize the signal-to-noise ratio (SNR) of the system.

Block Averaging

First, we have block averaging, the method described in Section 2. In this method, the w_{ij}'s are periodically reinitialized to zero, data are passed through the array, and the w_{ij}'s are formed, giving equal weight to each product in the sum (as in Eq. (17)). This form of averaging may be accomplished using the same set of K samples to compute the w_{ij}'s in each row, or different sets. If the same set of samples is used, buffering is required to save the data. Buffering occurs just before the inputs of the array, in order to pass the data through the array n-1 times. If buffering is not used, a total of K(n-1) samples is required, and the method is called cascaded block averaging. The first time through is to compute the first-row averages ($\tilde{w}_{1j}$ and $\tilde{\tilde{w}}_{1j}$) in the "if k=i" clause in Figure 5. The second pass performs the outer "else" clause of Figure 5. $Z_j^2(2) = Z_j^1(2) - w_{1j}Z_1^1(2)$ in row 1 and the "if k=i" clause in row 2. In general, the copy of the data input to the array between time steps (m-1)K+1 and mK finally arrives at row m and is used to compute w_{mj} between time steps (2n-2)K+1 and (2m-1)K. Steering vectors S (in MW=S) can be input starting at time mK+1, and all the values of TS are available at time (2n-3)K+1.

If no buffer is to be used, then a different set of K samples is input in each time period (m-1)K+1 to mK. This method, called cascaded block averaging, will compute values of w_{ij} close to their true values if the underlying distribution determining the input values changes slowly and enough samples are used. Recall the result of Reed, Mallett, and Brennan [1974]* that says approximately 2n samples are required for an expected SNR of 3 dB within optimum.

*I. S. Reed, J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-10, No. 6, November 1974, pp. 853-863.

Exponential Averaging

Second, we have exponential averaging. In this case, we take a weighted sum of the old average and the new data:

$$E^{m+1} = SE_{m+1} + (1-S)E^m \quad 0 < S < 1 \quad . \tag{18}$$

If S is close to 1, then the new average E^{m+1} is very responsive to new data E_{m+1}. If S is close to zero, E^{m+1} changes more slowly, being determined mainly by the old average. Exponential averaging can be implemented either with or without a buffer, as in block averaging, or, once the system has been started up and w_{ij} values are available, the w_{ij}'s can be updated by inputting just a few sample voltage vectors and averaging them in. This is an advantage over block averaging: With block averaging, each time the w_{ij}'s are to be updated, a lag time of (2n-3)K is needed, where K is at least n and preferably 2n; K may be significantly smaller with exponential averaging.

Note that it is possible either to exponentially smooth w_{ij} itself, or to smooth its numerator and denominator separately, before dividing them. (If, in addition to this separate smoothing, the denominator is approximated by its closest power of 2, the problem of doing division is reduced to shifting).

Window Averaging

Finally, we have window averaging. In this case, the last K samples are always used to compute the w_{ij}'s, thus using the most recently available information at all times. This method requires saving a buffer of the last K values used. When a new value (e.g., $\bar{Z}_i^i \cdot Z_j^i$) is computed, it is added to the window average, put on top of the buffer, and the oldest value removed from the bottom of the buffer and subtracted from the average. The buffer can conveniently be implemented as a shift register. In addition to always using the most recent data, window averaging has the same advantage as ex-

ponential over block averaging: It does not always require a lag of (2n-3)K samples to update the w_{ij}'s. The disadvantage, of course, is the need for a long shift register (at least n and probably 2n words) for each average (n(n-1)/2 or n(n-1), depending on the implementation).

The transform space method of Section 2 can be used as well as any of the above methods, but its real advantage lies in the fact that since samples (in contrast to steering vectors) are constantly being passed through the array, the w_{ij}'s can be updated simultaneously with the calculation of filter functions. Hence, when exponential or window averaging is used, no delay at all (after startup) is needed to update the weights. Each filter function is always a function of the most recent information.

Reed, Mallett, and Brennan [1974] have shown that the expected value of the achieved SNR is 10 $\log_{10}[(K+2-n)/(K+1)]$ dB below the SNR achieved with optimal weights. This expected loss is about 3 dB when the number of samples, K, equals 2n. Of course, if a great deal of clutter is present, more samples may be required to average it out. Also, Brennan [1974]* has recently shown that if the weights determined by K samples are in turn used to form filter functions from those same samples instead of a different set of samples (as in nontransform space operation), then the expected SNR is actually higher (see Appendix A).

We now present some graphs of system performance versus sample size for both real and simulated data, varying numbers of weights, and varying averaging schemes (Figures 11 through 21). System performance is measured by how many decibels down the achieved SNR is from the optimal SNR:

*L.E. Brennan, "Performance of the Sample Covariance Matrix Algorithm for Adaptive Arrays," unpublished manuscript, 19 July 1979.

$$\text{Performance} = 10 \log_{10}[(|S^*W|^2/W^*MW)/S^*M^{-1}S)]$$

The real data is taken from an operational 5-weight adaptive system, and the simulated data model is described in Appendix H. The true matrix M for the real data is estimated by averaging over all available samples (100). The different algorithms used are Cholesky, Gram-Schmidt using blocked averaging, Gram-Schmidt using exponential averaging (for various exponential weights, i.e., the factor S in Eq. (18)), and Gram-Schmidt using cascaded block averaging. In the case of simulated data, the various model parameters are the number of weights, number of jammers, location and power of the jammers, and the ratio of strongest jammer power to receiver noise power (per receiver). This last quantity is the spectral condition number of the matrix. The use of tapped delay lines (one-sample long) is also indicated.

For the real data (Figures 11 and 13) we see that the results for Cholesky and blocked G-S almost overlap. They are mathematically identical algorithms, and any small difference is attributable to roundoff error (all arithmetic is 32-bit floating point with 24-bit mantissas). Cascaded blocked G-S has essentially the same performance as blocked G-S; but exponential G-S lags behind, performance improving, but still poor as the weight decreases (which means the new samples are weighted less than the old).

For the simulated data with 5 weights (Figures 14 through 17), Cholesky, blocked G-S, and cascaded blocked G-S all have virtually identical performance, but exponential becomes poorer as the number of jammers increases (from 1 to 4).

With 10-channel simulated data using 5 jammers (Figure 18), cascaded G-S, blocked G-S, and Cholesky are again virtually identical. In fact,

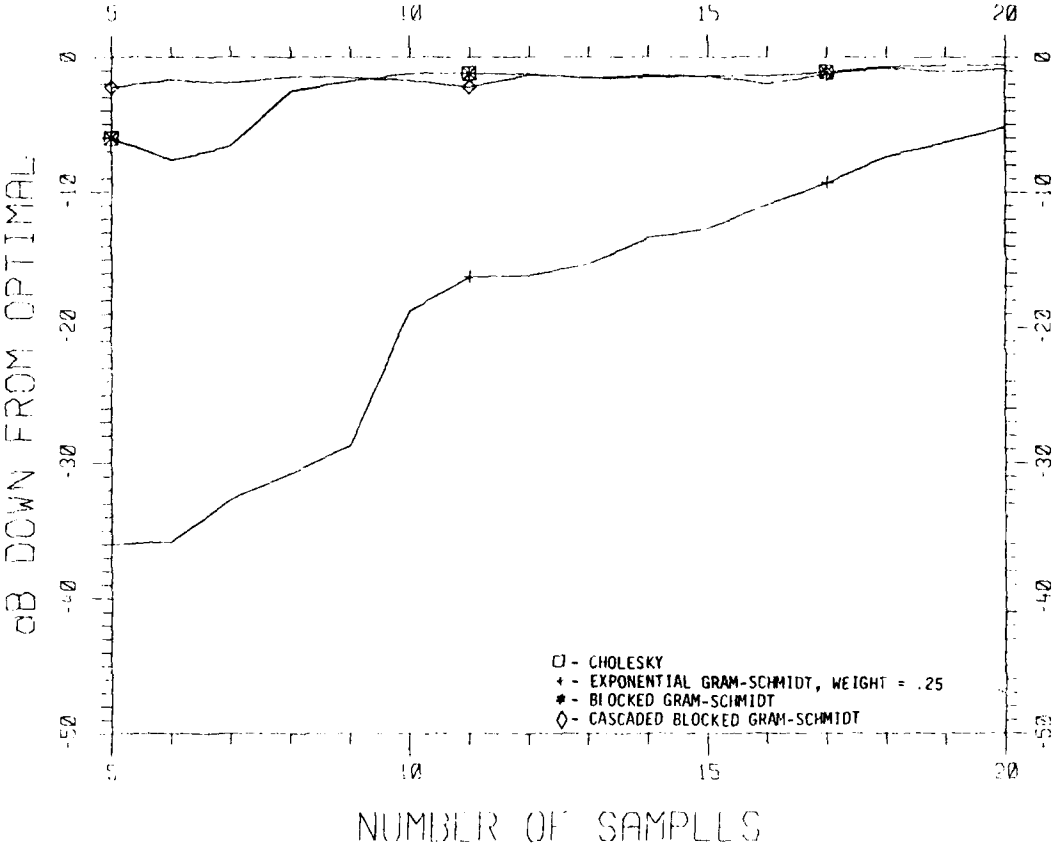


Figure 11. Real data; 5 weights.

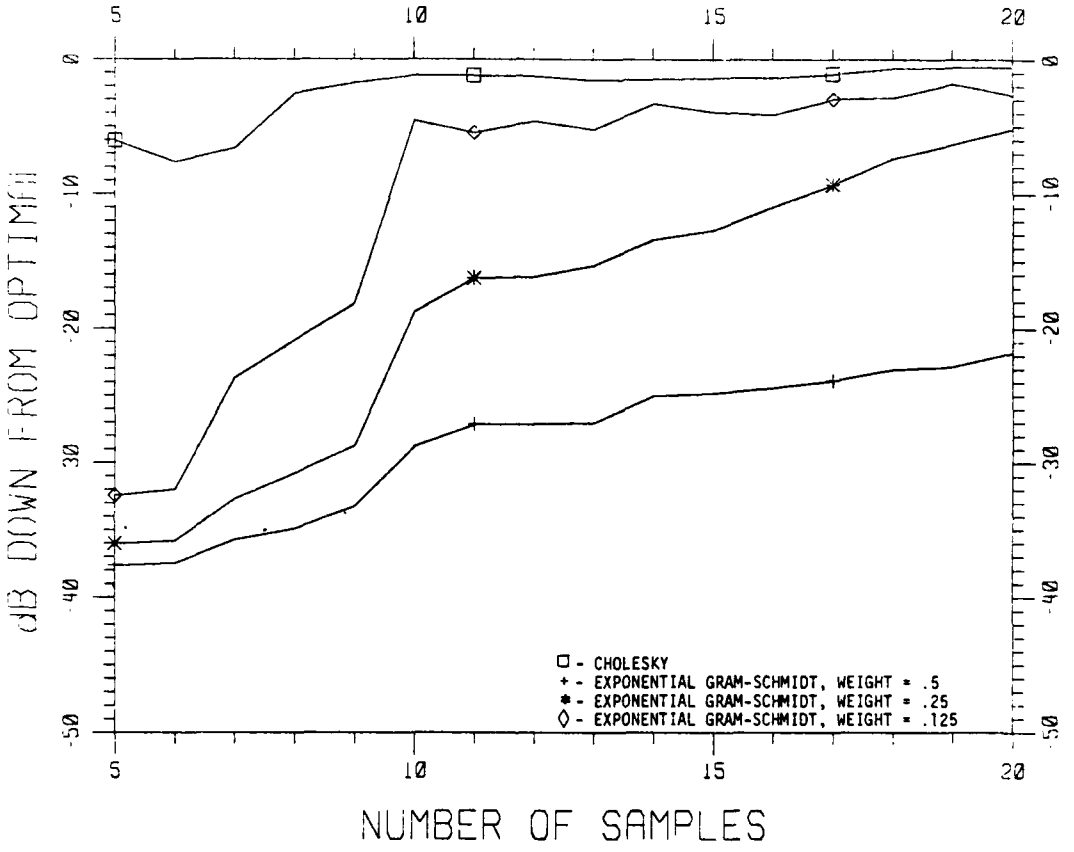


Figure 12. Real data; 5 weights.

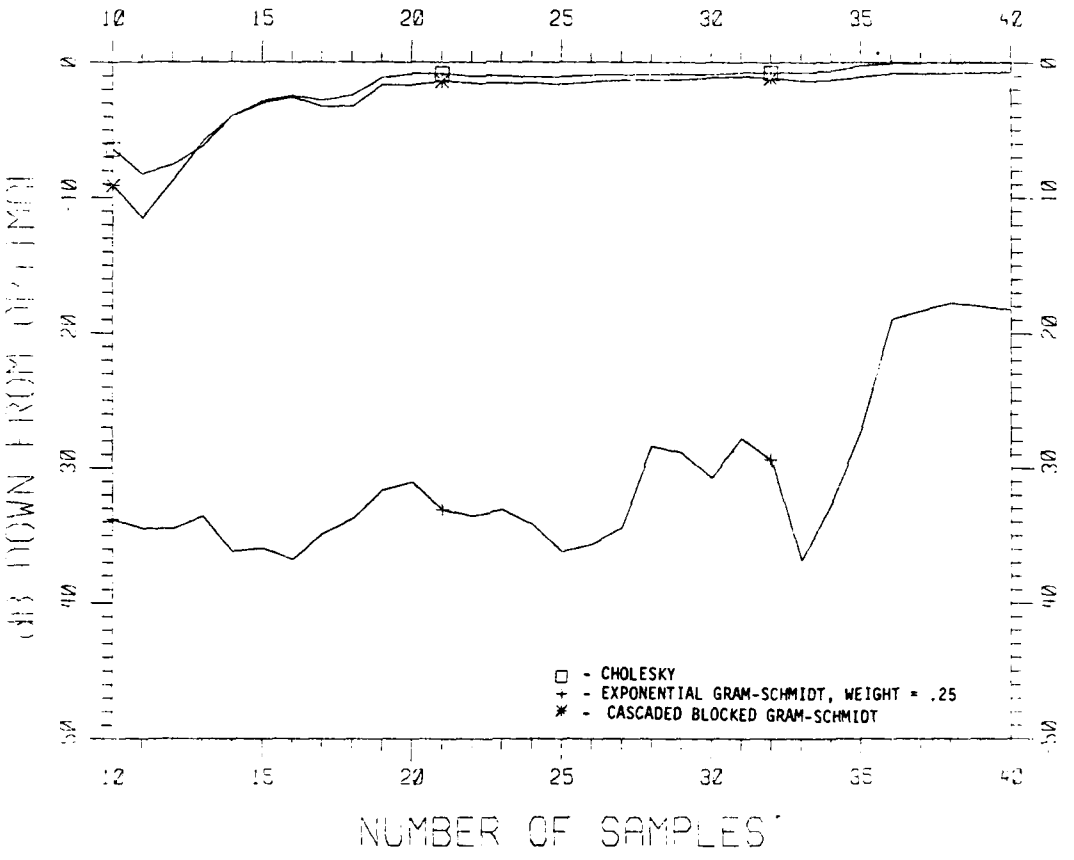


Figure 13. Real data; 5 weights, 1 tap per channel.

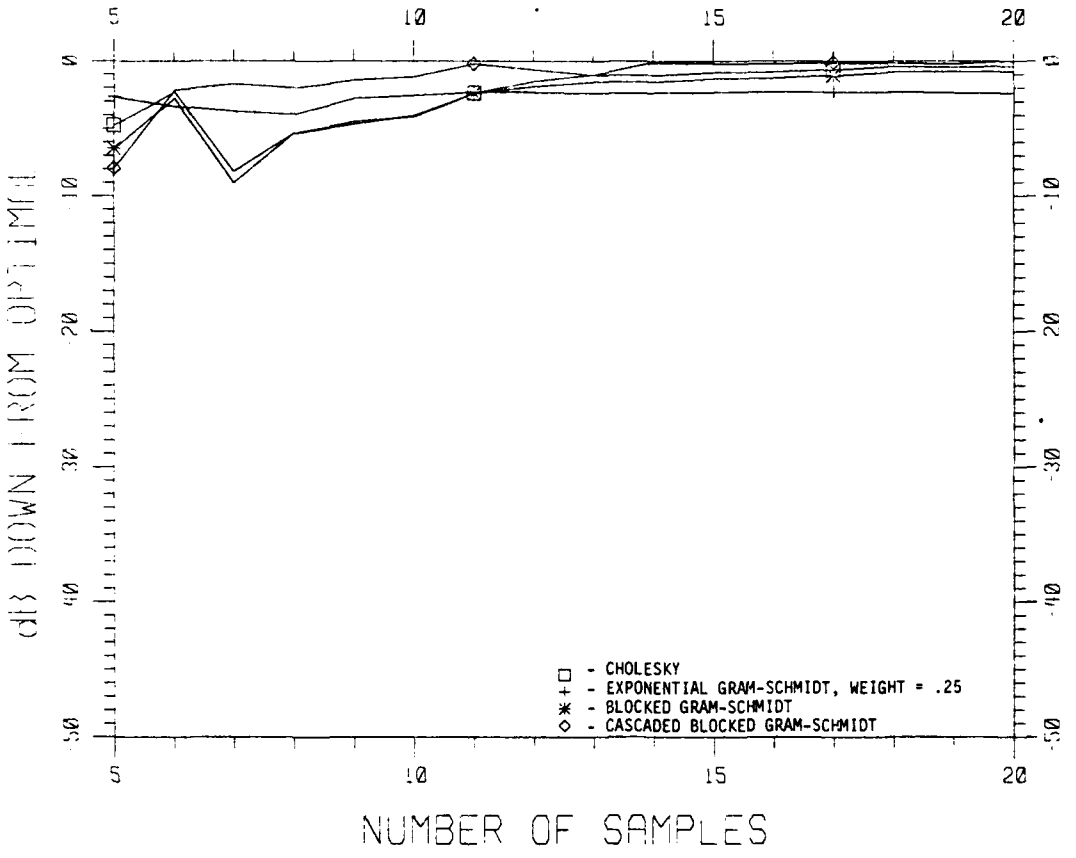


Figure 14. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 1
4. Location and power of jammer: (-36.9°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 50 dB.

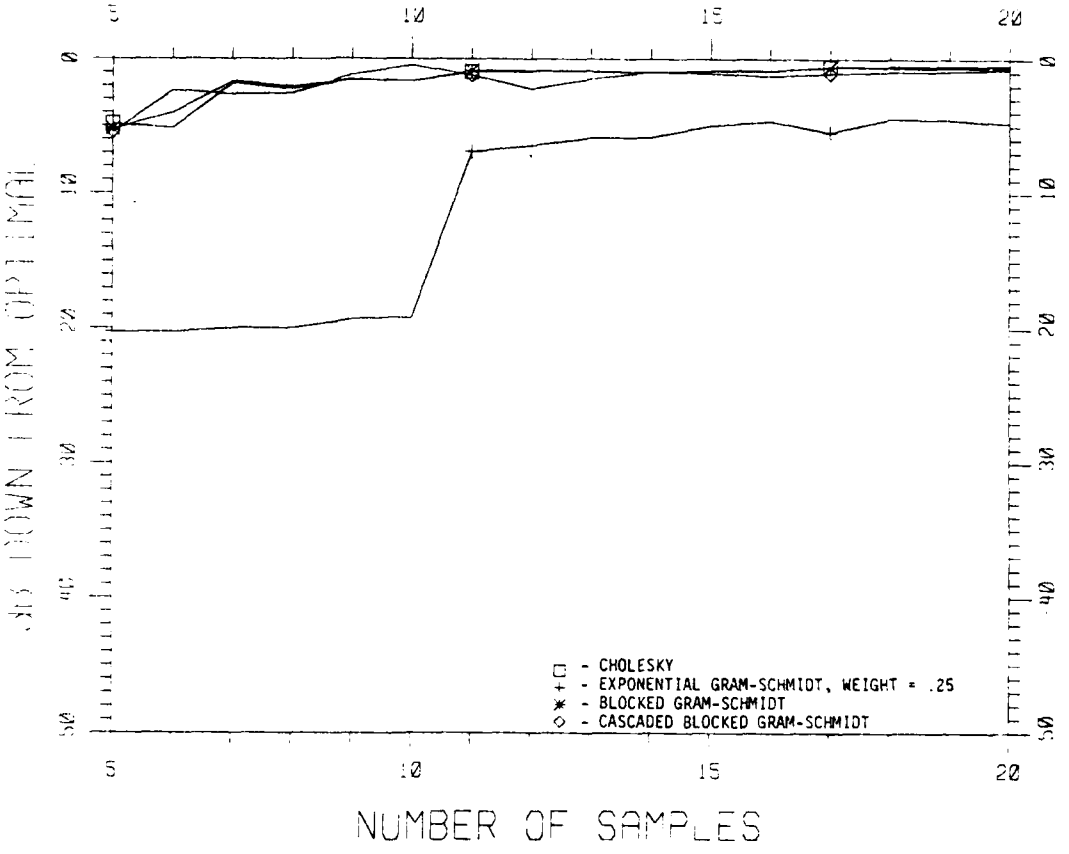


Figure 15. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 2
4. Location and power of jammers:
   #1 (-36.9°, 5.0 dB), #2 (-11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver power = 40 dB.

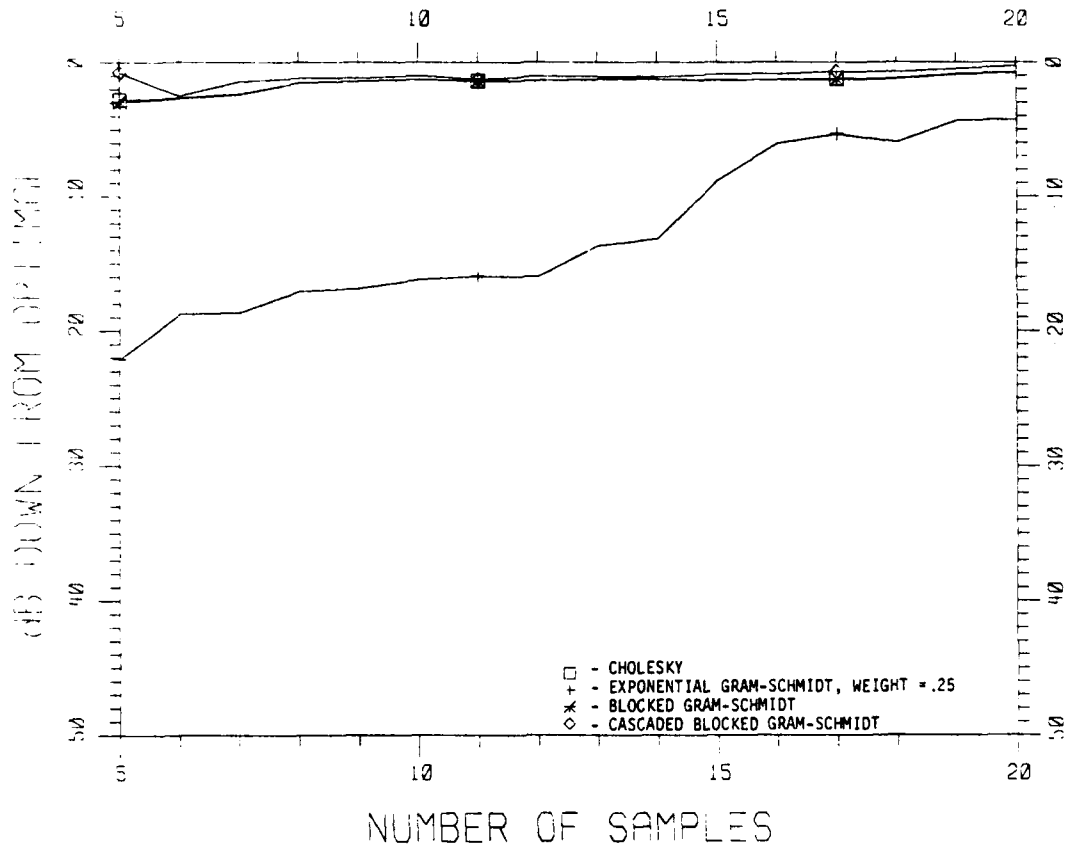


Figure 16. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB.

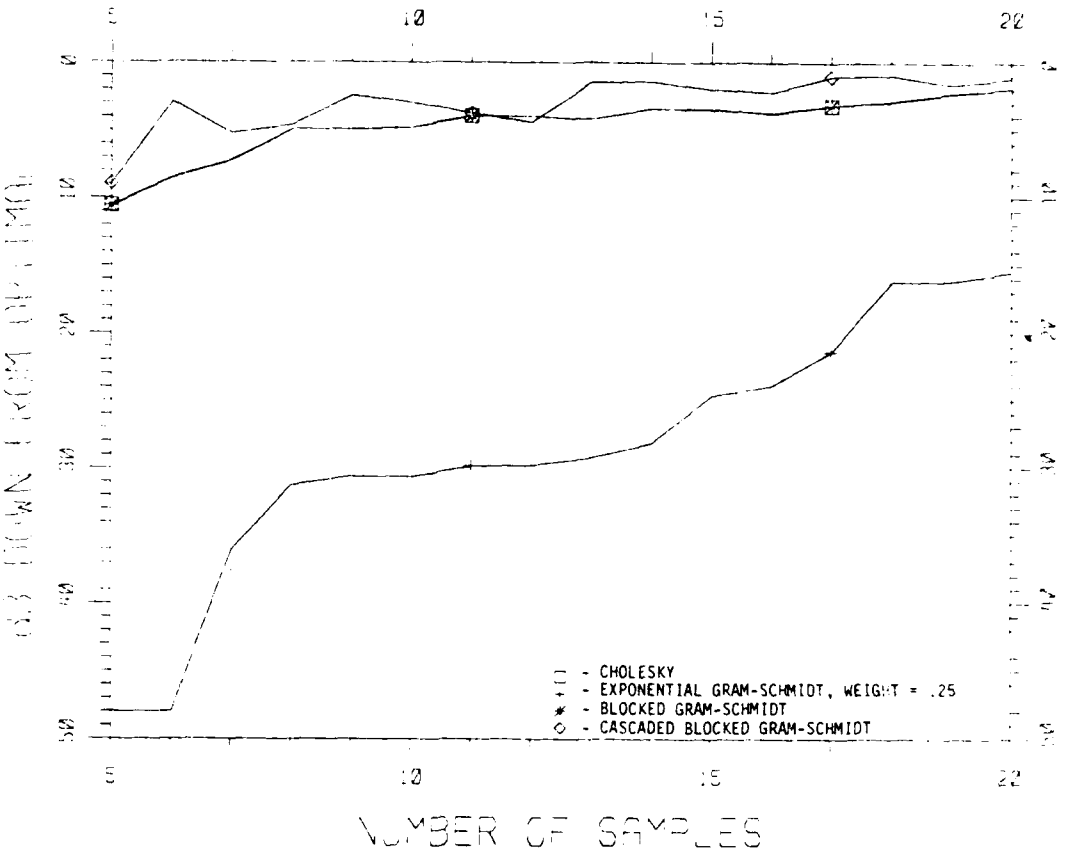


Figure 17. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 4
4. Location and power of jammers:
   #1 (-36.9°, 7.5 dB)
   #2 (-11.5°, 5.0 dB)
   #3 (+11.5°, 2.5 dB)
   #4 (+36.9°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 50 dB.

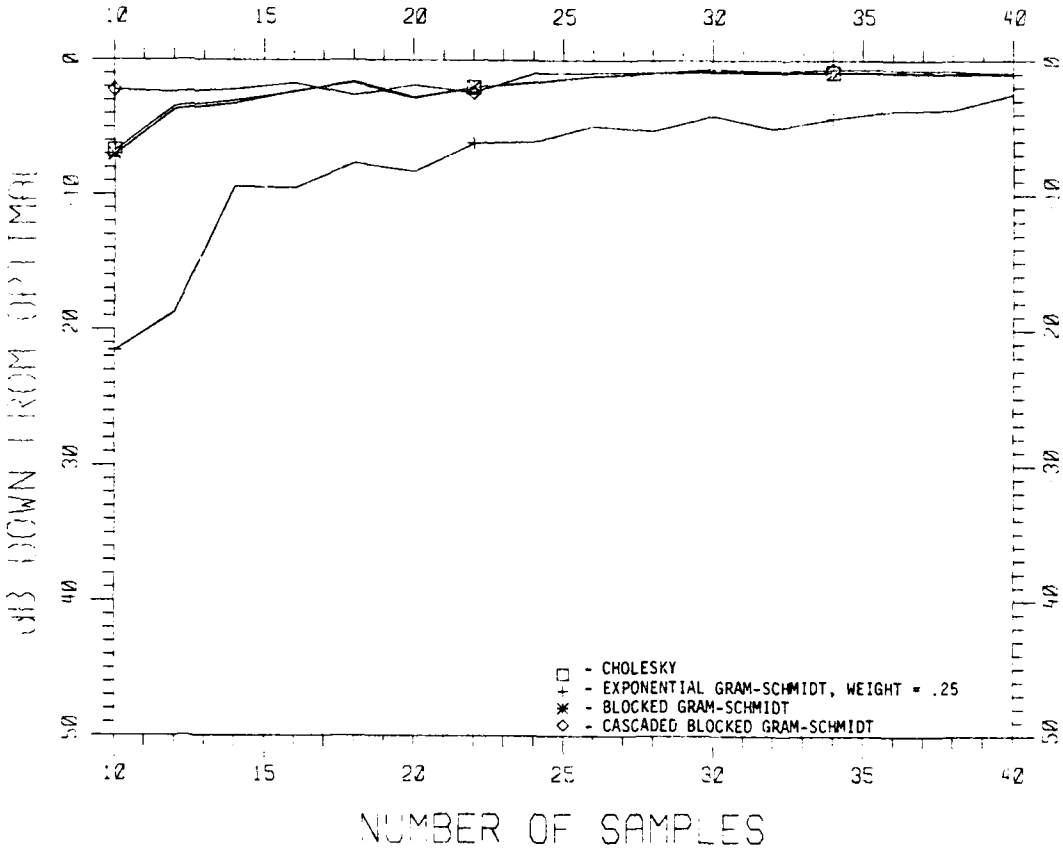


Figure 18. Simulated data:

1. Number of weights = 10
2. No tapped delays
3. Number of jammers = 5
4. Location and power of jammers:
   #1 (-53.1°, 8.0 dB)
   #2 (-36.9°, 6.0 dB)
   #3 (-23.6°, 4.0 dB)
   #4 (-11.5°, 2.0 dB)
   #5 (0.0°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB.

cascaded G-S performs somewhat better, because it uses n-1 times as much data as the other methods. Again, exponential performs the poorest, with a weight of 0.25 being best in this case (Figure 19).

With 35- and 50-channel simulated data, using 10 and 20 jammers, respectively, blocked and cascaded G-S and Cholesky are again virtually identical, with exponential G-S performing poorly (Figures 20 and 21, respectively).

Seven of the nine cases achieve an SNR within 3 dB of optimal after 2n samples; the other two cases are within 5 dB.

The location of the jammers (first number in parentheses in item 4 of the simulated-data captions) is given in degrees from boresight, and, jammer power (second number in parentheses) is expressed as decibels below the strongest jammer.

We spent a great deal of effort analyzing cases with low numbers of weights because the effects of changes in the number of jammers and exponential weights show up more quickly.

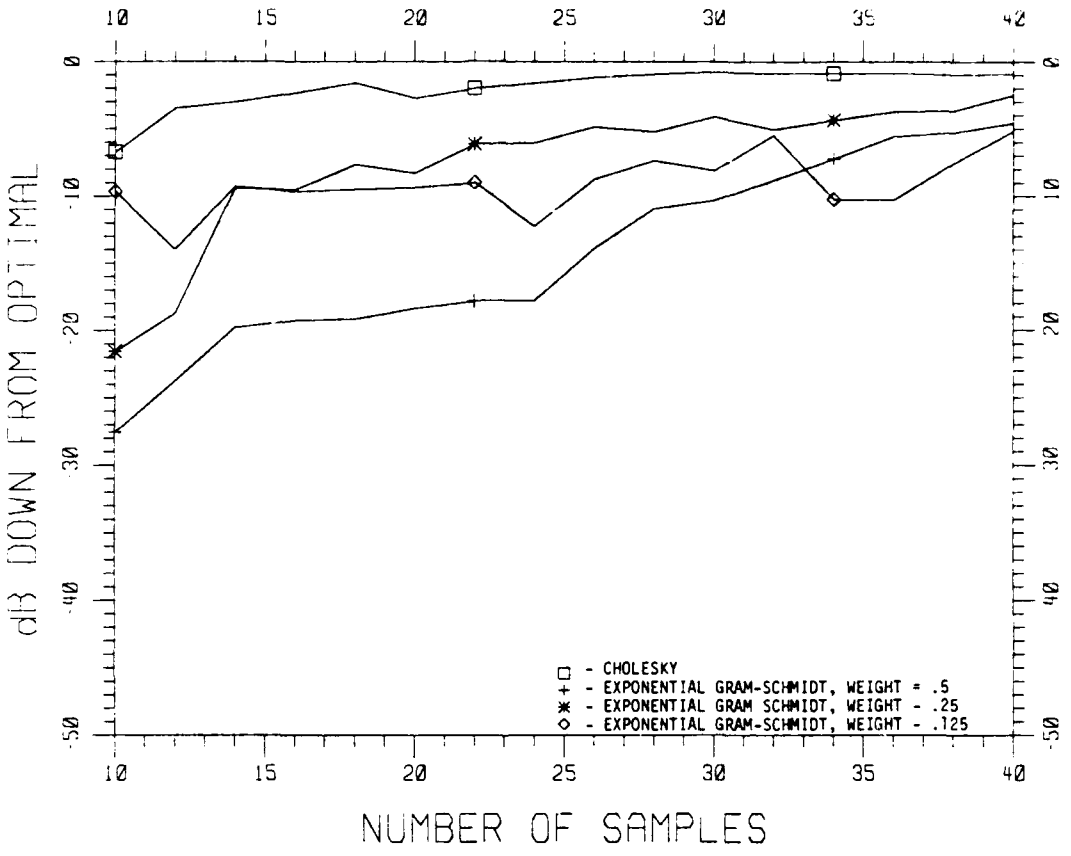


Figure 19. Simulated data:

1. Number of weights = 10
2. No tapped delays
3. Number of jammers = 5
4. Location and power of jammers:
   #1 (-53.1°, 8.0 dB)
   #2 (-36.9°, 6.0 dB)
   #3 (-23.6°, 4.0 dB)
   #4 (-11.5°, 2.0 dB)
   #5 (0.0°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB.

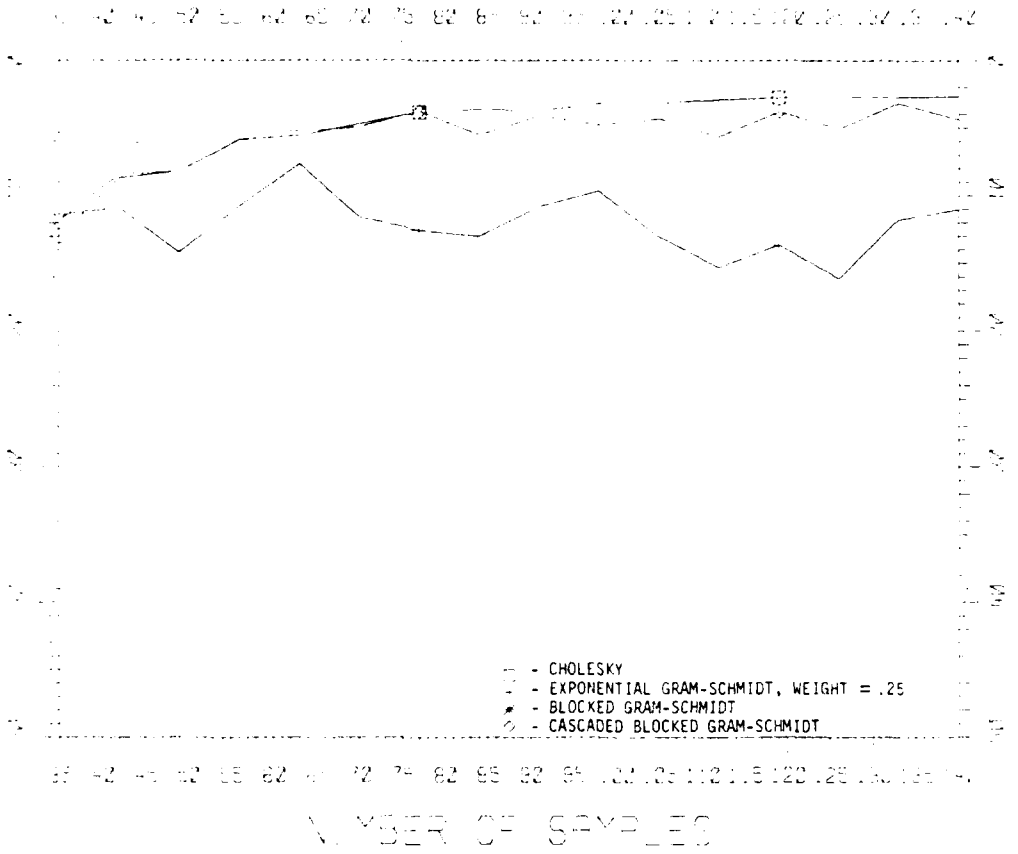


Figure 20. Simulated data:

1. Number of weights = 35
2. No tapped delays
3. Number of jammers = 10
4. Location and power of jammers:

#1 (-70.5°, 9.0 dB)	#6 (-41.1°, 4.0 dB)
#2 (-62.3°, 8.0 dB)	#7 (-36.9°, 3.0 dB)
#3 (-55.95°, 7.0 dB)	#8 (-32.9°, 2.0 dB)
#4 (-50.5°, 6.0 dB)	#9 (-29.1°, 1.0 dB)
#5 (-45.6°, 5.0 dB)	#10(-25.4°, 0 dB)

5. Ratio of power of strongest jammer to receiver noise power = 27 dB.

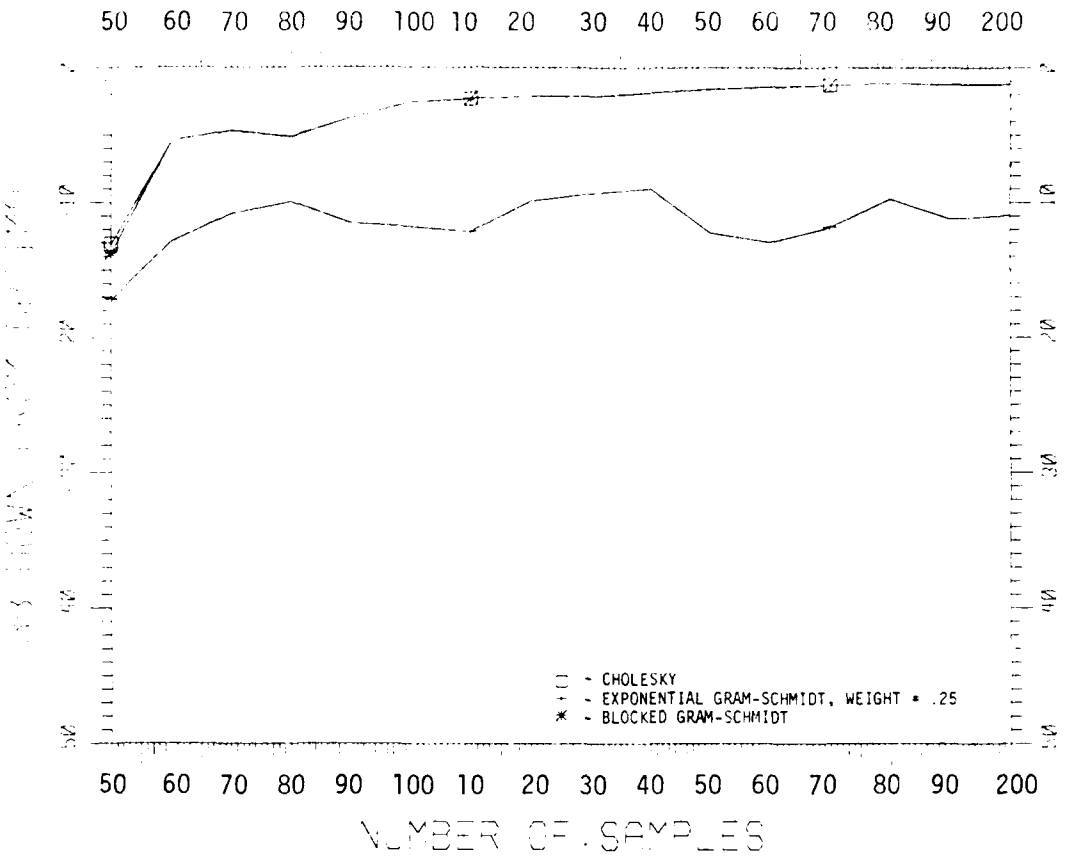


Figure 21. Simulated data:

1. Number of weights = 50
2. No tapped delays
3. Number of jammers = 20
4. Location and power of jammers:

#1 (-73.7°, 9.5 dB)	#11 (-34.05°, 4.5 dB)
#2 (-66.9°, 9.0 dB)	#12 (-31.3°, 4.0 dB)
#3 (-61.6°, 8.5 dB)	#13 (-28.7°, 3.5 dB)
#4 (-57.1°, 8.0 dB)	#14 (-26.1°, 3.0 dB)
#5 (-53.1°, 7.5 dB)	#15 (-23.6°, 2.5 dB)
#6 (-49.5°, 7.0 dB)	#16 (-21.1°, 2.0 dB)
#7 (-46.0°, 6.5 dB)	#17 (-18.7°, 1.5 dB)
#8 (-42.8°, 6.0 dB)	#18 (-16.3°, 1.0 dB)
#9 (-39.8°, 5.5 dB)	#19 (-13.9°, 0.5 dB)
#10(-36.9°, 5.0 dB)	#20 (-11.5°, 0 dB)

5. Ratio of power of strongest jammer to receiver noise power = 20 dB.

3.2 FAULT TOLERANCE AND NOISE SENSITIVITY

In this section we analyze the effect of the system's different failure modes on performance, as measured by the SNR.

External Failure Mode

The first category of failure mode is external, which means that although the processor array is functioning normally, something is wrong with the antenna/receiver subsystem supplying inputs. The possible failures we examine are 1) input j is zero (others normal), 2) input j is a nonzero constant (others normal), and 3) Gaussian noise of power ρ is added to input j (others normal).

When the input j is zero, the system continues to function, computing the correct weights for an n-1 weight system and a zero weight for input j. To verify this phenomenon, note that all the w_{ij} values, $1 \leq i \leq j-1$, are zero, because (in the notation of Figure 5) $\tilde{w}_{ij} = \tilde{w}_{ij} + \overline{Z}_i^i(k) \cdot Z_j^i(k)$ and Z_j^1 is zero, so w_{ij} is zero, Z_j^2 is zero, w_{2j} is zero, and so on. Thus, $Z_j^j = 0$, and (with the odd convention that 1/0=0) we have $w_{ji} = 0$, $j+1 \leq i \leq n$, since w_{ji} is $\tilde{w}_{ji}/\Sigma Z_j^j \overline{Z_j^j} = w_{ji} \cdot (1/0) = 0$. Thus the values computed in columns j+1 to n of the array have nothing subtracted from them in row j of the array: They are independent of input j. Since the values computed in columns 1 to j-1 are obviously independent of column j, all the other computed values are computed correctly as though input j were missing. Since $1/w_{jj}$ is taken to be zero, and the output at port j on the right of the array is multiplied by $1/w_{jj}$, we see the system operates as claimed for the transform space and unit vector methods, because they get a zero contribution from output j and only operate the array in the forward direction discussed above in this paragraph. To see that reverse flow also operates normally, note that the zero input at port j on the right remains zero as it moves left across the array (since the

w_{ji}'s in row j are zero) and when it is broadcast vertically by $P_{j,j+1}$, it has no effect on the values computed by rows 1 to j-1. Since rows j+1 to n are similarly unaffected, we see all weights except j are computed normally, and weight j is zero as desired.

The optimal SNR of the n-1 weight system can be expressed in terms of the old optimal SNR with n weights, the j^{th} optimal weight w_j (when all n are available) and m^{jj} = j^{th} diagonal element of the inverse of the true n x n covariance matrix:

$$SNR_{NEWOPT} = SNR_{OLDOPT} - \frac{|w_j|^2}{m^{jj}} \quad . \tag{19}$$

This result will be derived later.

When input j is a nonzero constant, a similar analysis shows incorrect weights are computed. Approximately 0 values for w_{ij} in column j are computed, since Z_i^i is Gaussian with zero mean. Thus Z_j^j is constant and nonzero, and $1/w_{jj}$ is nonzero. The values of w_{ji} in row j are again approximately zero, but output j, even after multiplication by $1/w_{jj}$, remains approximately constant and nonzero. Thus incorrect values are computed by all three methods.

To examine the case of Gaussian noise, we need some general results on the effects of perturbations in the weights on the SNR. These results are derived in Appendix B and presented below.

The SNR is given by:

$$SNR_W = |S^*W|^2/W^*MW \quad , \tag{20}$$

where SNR_W is the SNR achieved using W as weights. When using the optimal weights, $W_{OPT} = M^{-1}S$, we have

$$SNR_{OPT} = S^*W = W^*MW = S^*M^{-1}S \quad . \tag{21}$$

Suppose that $W = W_{OPT} + E$, where E is an error vector. Then we may write SNR_W in two equivalent forms:

$$SNR_W = \left(1 - \frac{E^*ME}{W^*MW}\right)\cdot SNR_{OPT} + \frac{E^*ME}{W^*MW} \cdot SNR_E$$

$$= SNR_{OPT} - E^*\left(M - \frac{SS^*}{SNR_{OPT}}\right)E \cdot \frac{SNR_{OPT}}{W^*MW} \quad . \tag{22}$$

The first expression for SNR_W shows it is a weighted linear combination of SNR_{OPT} and SNR_E, and the second expression shows how the degradation in SNR depends on the eigenvalues and eigenvectors of the positive semidefinite Hermitian matrix $\left(M - SS^*/SNR_{OPT}\right)$. In particular, a lower bound for SNR_W is

$$SNR_W \geq SNR_{OPT} - \lambda_{max}(M) \cdot ||E||^2 + O(||E||^3) \quad , \tag{23}$$

which shows that the SNR_W depends quadratically on E, and is worse if the largest eigenvalue of M, $\lambda_{max}(M)$, is large. The bound on Eq. (23) can be attained if S is orthogonal to the eigenvector of M belonging to $\lambda_{max}(M)$, and if E is proportional to it. Physically, since eigenvectors of the covariance matrix M are the expected signal of noise sources and the eigenvalue is the power of the noise source, the worst degradation in SNR shoud occur when the steering signal S is aiming the array in a direction orthogonal to the strongest noise source but E is aiming exactly in the direction of the strongest noise source.

We may also examine the effects of perturbations on arbitrary (nonoptimal) weights W. Suppose that instead of W weights, $\hat{W}=W+E$ are used. Then the new SNR using $\hat{W}$ is

$$SNR_{\hat{W}} = SNR_W - 2\,\frac{SNR_W}{W^*MW} \cdot Re\left[\left\{\left(M - \frac{SS^*}{SNR_W}\right)W\right\}^*E\right] + O(||E||^2) \quad , \tag{24}$$

where the quadratic and higher-order terms are indicated by $O(||E||^2)$. Now the dependence on E is linear, with the effect on $SNR_{\hat{W}}$ being largest when E is proportional to $(SS^*-SNR_W \cdot M)W$ (the SNR can increase or decrease), and to a linear approximation lies in the range

$$SNR_W - \frac{2SNR_W}{W^*MW} \left|\left|\left(M - \frac{SS^*}{SNR_W}\right) W\right|\right| \cdot ||E|| \leq SNR_{\hat{W}} \leq \left[SNR_W + \frac{2SNR_W}{W^*MW} \left|\left|\left(M - \frac{SS^*}{SNR_W}\right) W\right|\right| \cdot ||E||\right] \quad . \tag{25}$$

When E is orthogonal to $\left(M - SS^*/SNR_W\right)W$, the dependence of $SNR_{\hat{W}}$ on E is quadratic. Thus,

$$SNR_W = \text{linear terms in Eq. (20)} + \frac{SNR_W}{W^*MW} \text{ Re} \left[\left(4\text{Re}\frac{(W^*ME)}{W^*MW} W^* - E^*\right)\left(M - \frac{SS^*}{SNR_W}\right)E\right] + \text{higher-order terms in } ||E|| \quad . \tag{26}$$

When $W = W_{OPT}$, Eq. (26) simplifies to Eq. (23).

Now let us return to the specific problems of Gaussian noise of power ρ being added to input j. (The following results are also derived in Appendix B). Adding this noise is equivalent to adding ρ to the j^{th} diagonal elements of the true covariance matrix M to get the new matrix $\hat{M}$, $\hat{M} = M+A$, where $A_{k\ell} = \{\rho \text{ if } k=\ell=j, 0 \text{ otherwise}\}$. We may express $\hat{M}^{-1}$ in terms of M^{-1}

$$\hat{M}^{-1} = M^{-1} + M^{-1} \cdot \left(\frac{-\rho}{1+\rho m^{jj}}\right) \begin{bmatrix} & 0 & \\ m^{j1} & \ldots & m^{jn} \\ & 0 & \end{bmatrix} \leftarrow j^{th} \text{ row} \quad , \tag{27}$$

where $M = \{m_{ij}\}$ and $M^{-1} = \{m^{ij}\}$. Thus the new weights $\hat{W} = \hat{M}^{-1}S$ can be expressed in terms of the old weights

$$\hat{W} = W + M^{-1}\left(\frac{-\rho}{1+\rho m^{jj}}\right)\begin{bmatrix}0\\w_j\\0\end{bmatrix} \leftarrow j^{th} \text{ row} \quad . \tag{28}$$

Hence the new weights are the sum of the old weights and the j^{th} column of M^{-1} times $-\rho w_j/(1 + \rho m^{jj})$. In particular, the new j^{th} weight

$$\hat{w}_j = w_j/(1 + \rho m^{jj}) \quad .$$

So as ρ increases, $\hat{w}_j$ decreases, approximately in proportion to $1/\rho$ for ρ large. As ρ approaches infinity, $\hat{w}_j$ approaches zero and the system operates as though there were only n-1 weights, because all the information in channel j is being obscured by the large noise.

We must now distinguish between two subcases, depending on when the noise is added to the system. Case 1 occurs when the noise is added by the antenna/receiver subsystem, so that the sample's voltage vectors used to compute the filter functions also contain noise, and case 2 occurs when the noise is added later, thus affecting the weights only and not the sample voltages for the filter function. In the first case, where the noise is present from the beginning, it is as though the true covariance matrix of the noise process changes (by adding ρ to the diagonal); in the second case, the true covariance matrix remains the same, but the weights change.

In the first case, we can compute the optimal SNR of the new system with noise $SNR_{NEWOPT} = S^*\hat{M}^{-1}S$ using Eq. (23):

$$SNR_{NEWOPT} = SNR_{OLDOPT} - \left[\rho|w_j|^2/(1 + \rho m^{jj})\right] \quad . \tag{29}$$

Thus, the optimal SNR achievable with noise ρ added varies as shown in Figure 22. For ρ small it decreases approximately linearly with a slope of $-|w_j|^2$. If $|w_j|$ is large, then system performance depends heavily on the j^{th} channel; hence, adding noise to that channel is particularly bad. Also, if m^{jj} is small, then the lower bound for SNR_{NEWOPT}, $SNR_{OLDOPT} - \left(|w_j|^2/m^{jj}\right)$, is small.

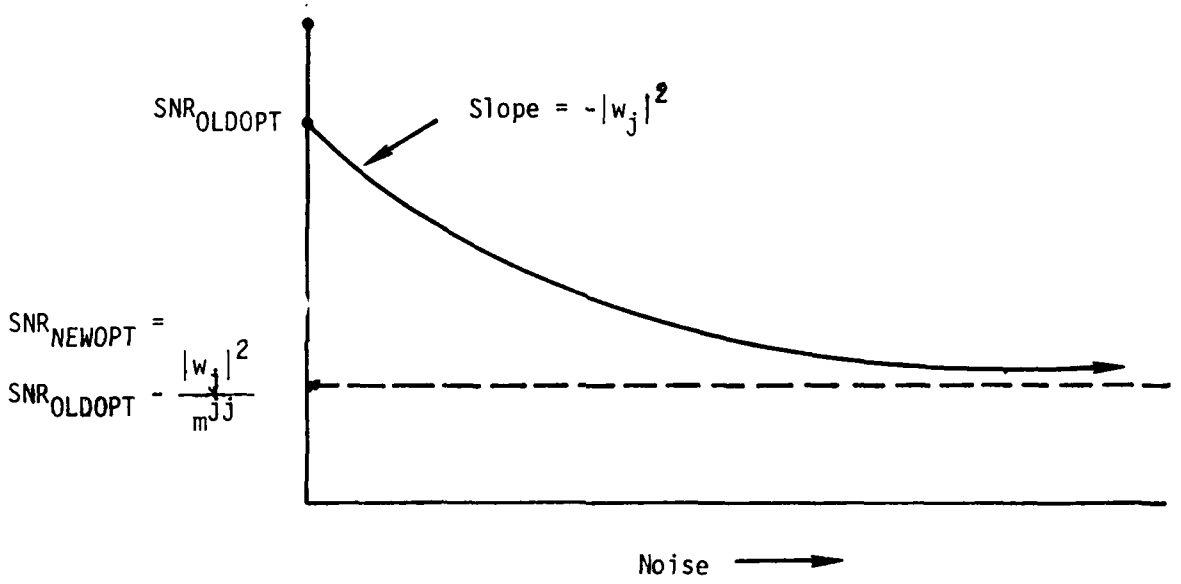


Figure 22. Effect of added noise on optimal SNR.

In fact, if $S = e_j = \{0,\ldots,0,1,0,\ldots,0\}$ (1 in j^{th} place), then $SNR_{OPT} = m^{jj}$ and $w_j = m^{jj}$. Then $SNR_{NEWOPT} = 1/\left(\rho + 1/SNR_{OLDOPT}\right)$, which approaches 0 as ρ approaches infinity.

It is possible to make a physical interpretation of m^{jj} being small. Mathematically, it means the covariance matrix $M_n \equiv M$ has an eigenvector approximately equal to $e_j = (0,\ldots,0,1,0,\ldots,0)$ (1 in j^{th} entry) whose

corresponding eigenvalue is large, which the matrix M_{n-1} = M with j^{th} row and j^{th} column removed does not have. Physically, this means the n weight system with matrix M_n is capable of adapting to a strong noise source, with steering signal e_j, whereas the n-1 weight system with matrix M_{n-1} cannot. Since

$$m^{jj} = \prod_{j=1}^{n-1} \lambda_j(M_{n-1}) \Big/ \prod_{j=1}^{n} \lambda_j(M_n)$$

($\lambda_j^{(x)}$ = j^{th} eigenvalue of matrix x), m^{jj} will be small if this relationship of eigenvalues and vectors of M_n and M_{n-1} holds. In other words, m^{jj} is small if the j^{th} channel is important for the system to be able to adapt to some large noise source. The system will then be especially sensitive to noise in that channel.

In the second case, where noise is added late so that it affects the weight computations but not the true covariance matrix M, we may use Eqs. (22), (24), and (25) to analyze system performance. Assuming that the weights used after perturbations are the optimal weights for the noise field seen by the array, i.e., $\hat{W} = \hat{M}^{-1}S$, we may use Eqs. (21) and (26) to compute the new $SNR_{\hat{W}}$:

$$SNR_{\hat{W}} = SNR_{OPT} - \rho^2|w_j|^2 m^{jj} \Big/ \left[SNR_{OPT}(1 + \rho m^{jj})^2 - \rho^2|w_j|^2(2 + \rho m^{jj})\right] \cdot \left[SNR_{OPT} - |w_j|^2/m^{jj}\right] \quad . \tag{30}$$

Note that $|w_j|^2/m_{jj} = SNR_E$, where $E = \hat{W} - W_{OPT}$. $SNR_{\hat{W}}$ is a decreasing function of ρ, with a lower bound of

$$SNR_{\hat{W}} \geq SNR_{OPT} - \left(|W_j|^2/m^{jj}\right) \quad , \tag{31}$$

which is the same lower bound as for SNR_{NEWOPT}, given in Eq. (25). Thus, if an infinite amount of noise is added to channel j, effectively turning the system into an n-1 weight system, the degradation in SNR can be bounded independent of where in the system the noise is added. In both cases, a large j^{th} weight $|w_j|$ or a small m^{jj} indicates the system is particularly sensitive to noise in that channel. This analysis also proves Eq. (16).

We now present some simulation results of adding noise to a 5-weight system with 3 jammers. Figure 23 is a control case with no noise added. The total noise power (summed over all inputs to which noise is added) is 4 dB above the strongest jammer. These results correspond to case 2 of the above discussion, where the noise affects the weight calculation only.

Apparently, inputs 2 and 3 (Figures 25 and 26) are very insensitive to noise; inputs 1 and 4 (Figures 24 and 27) are somewhat sensitive; and input 5 (Figure 28) is very sensitive. This sensitivity was determined by looking at the steady-state cancellation for either G-S or Cholesky (i.e., performance degradation for 20 samples). Since the steering signal for these latter simulations is (0,0,0,0,1), i.e., an SLC with channel 5 as the main, channel 5 should, by far, be the most sensitive. The other inputs are not as sensitive. Since there are only 3 jammers, as long as only one input is affected enough degrees of freedom remain to effectively cancel the jammers (unless, of course, channel 5 is affected). When 3, 4, or 5 channels are affected (as in the last three plots, Figures 29 through 31), so that there are not enough degrees of freedom to cancel the jammers and the extra noise, performance degrades markedly.

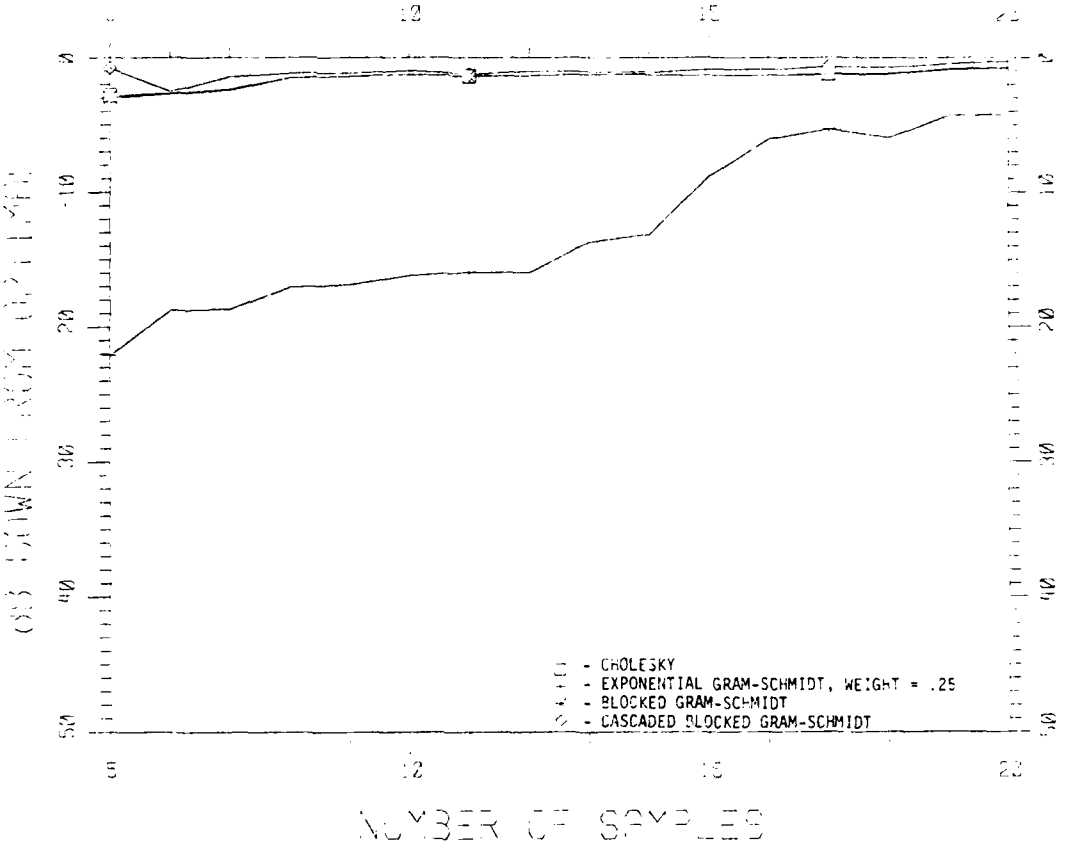


Figure 23. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. No noise added.

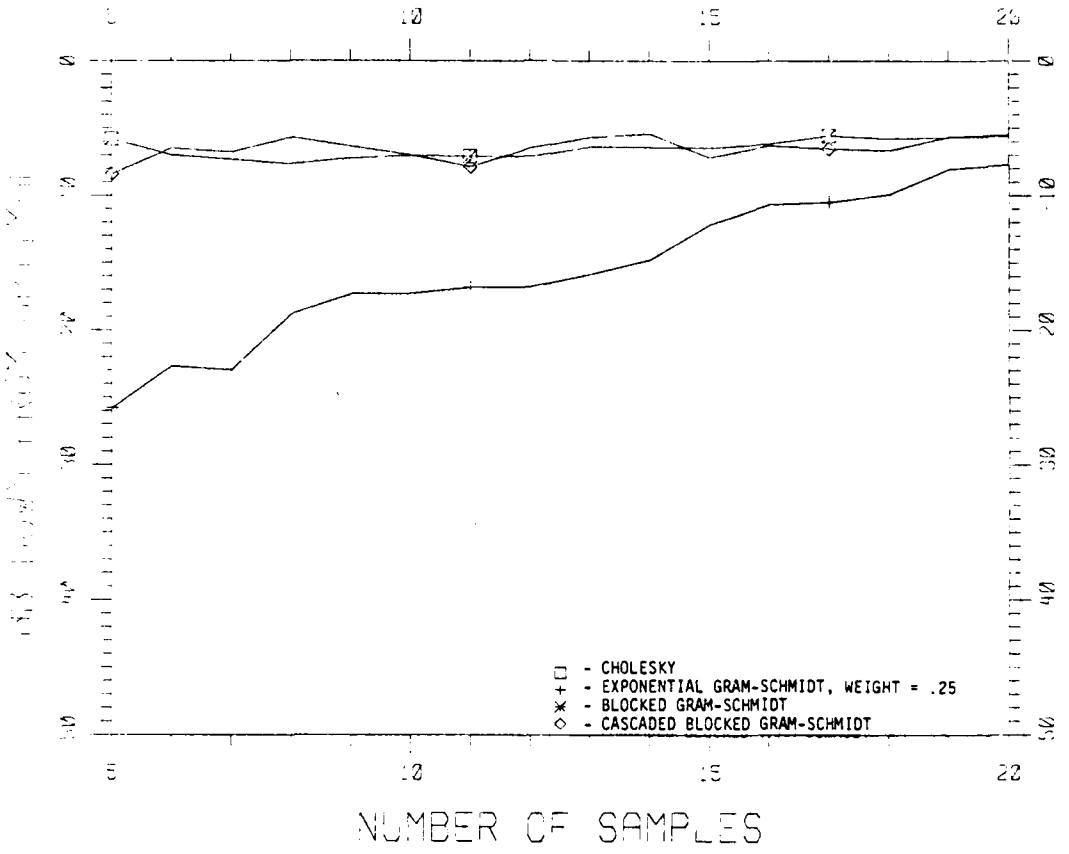


Figure 24. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channel 1 of total power 4 dB above strongest jammer.

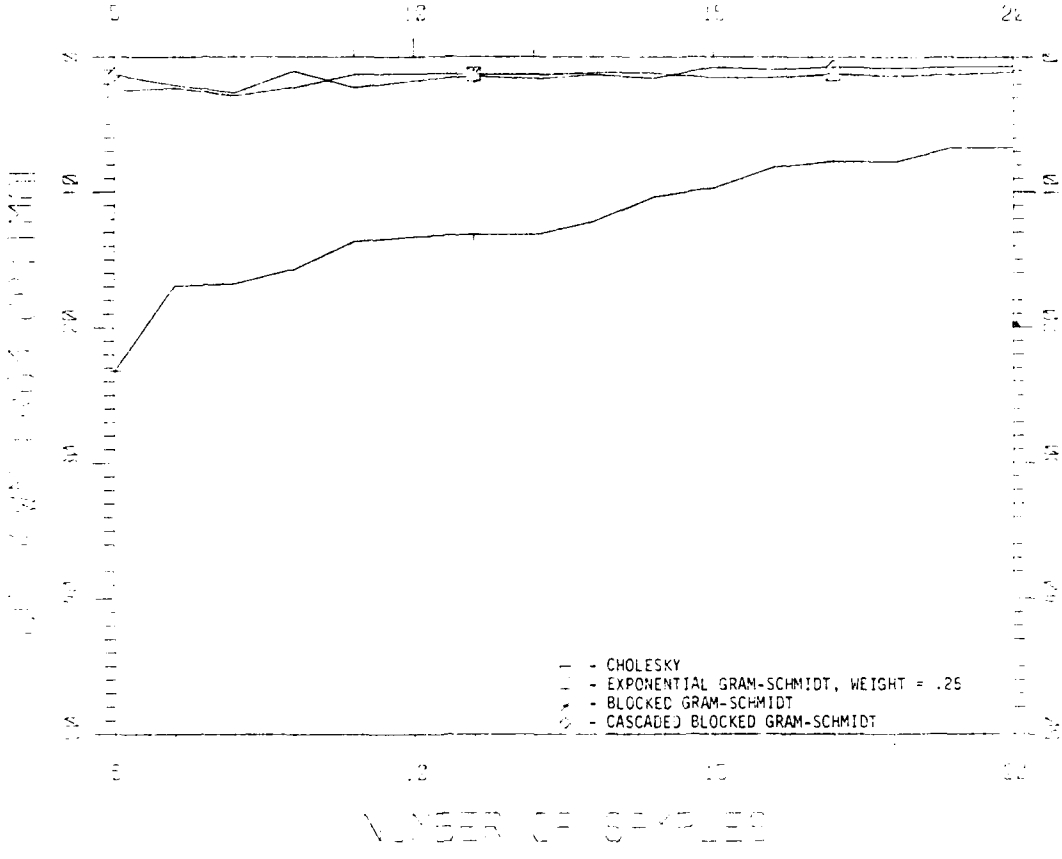


Figure 25. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB.
6. Noise added to channel 2 of total power 4 dB above the strongest jammer.

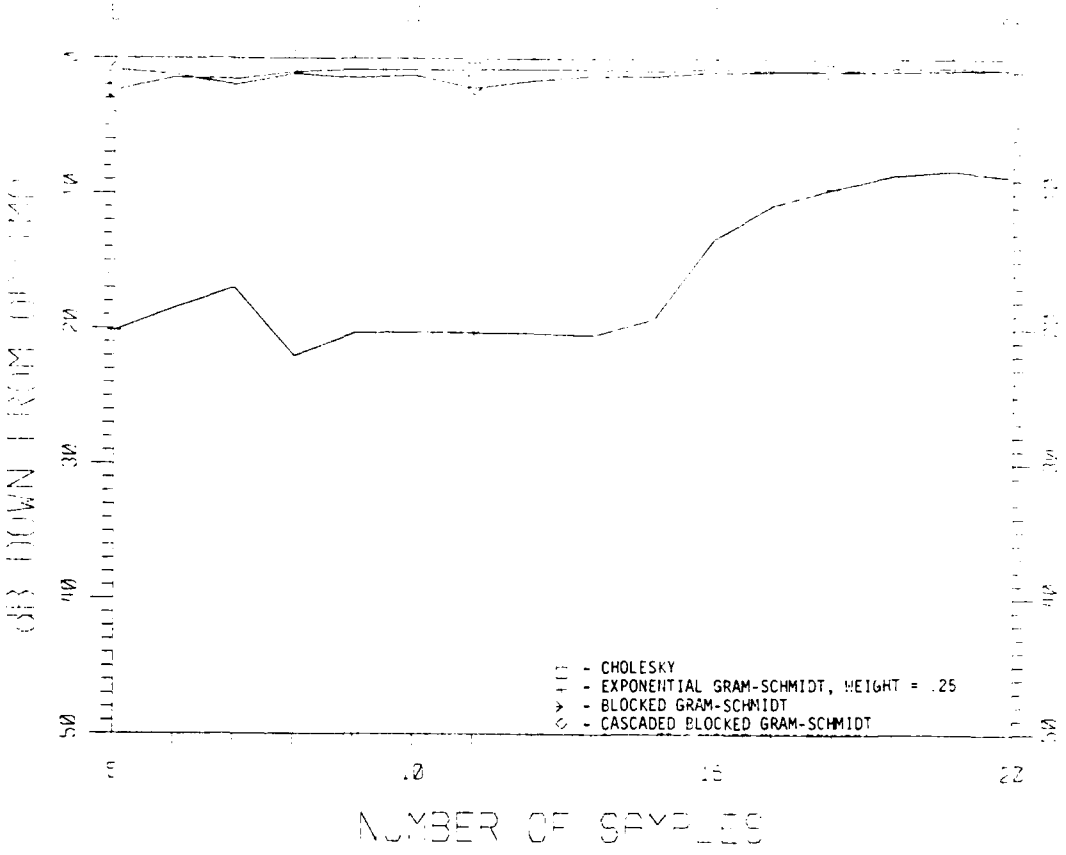


Figure 26. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channel 3 of total pwoer 4 dB above strongest jammer.

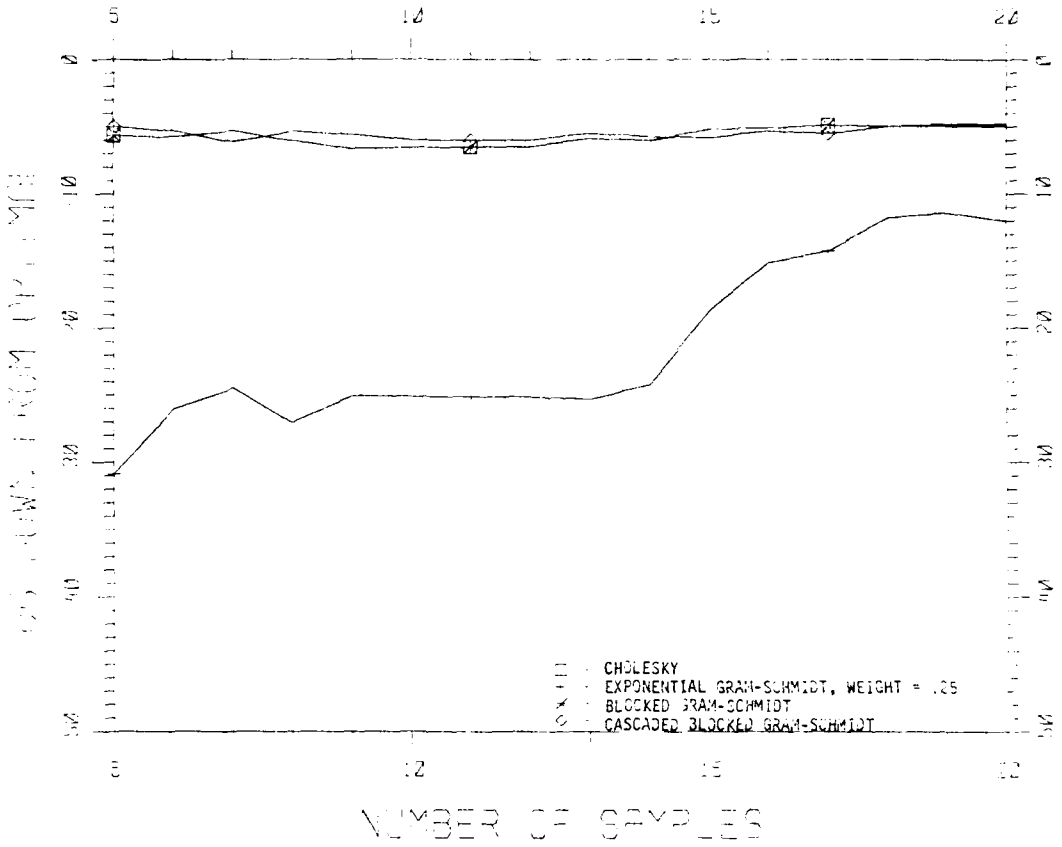


Figure 27. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channel 4 of total power 4 dB above strongest jammer.

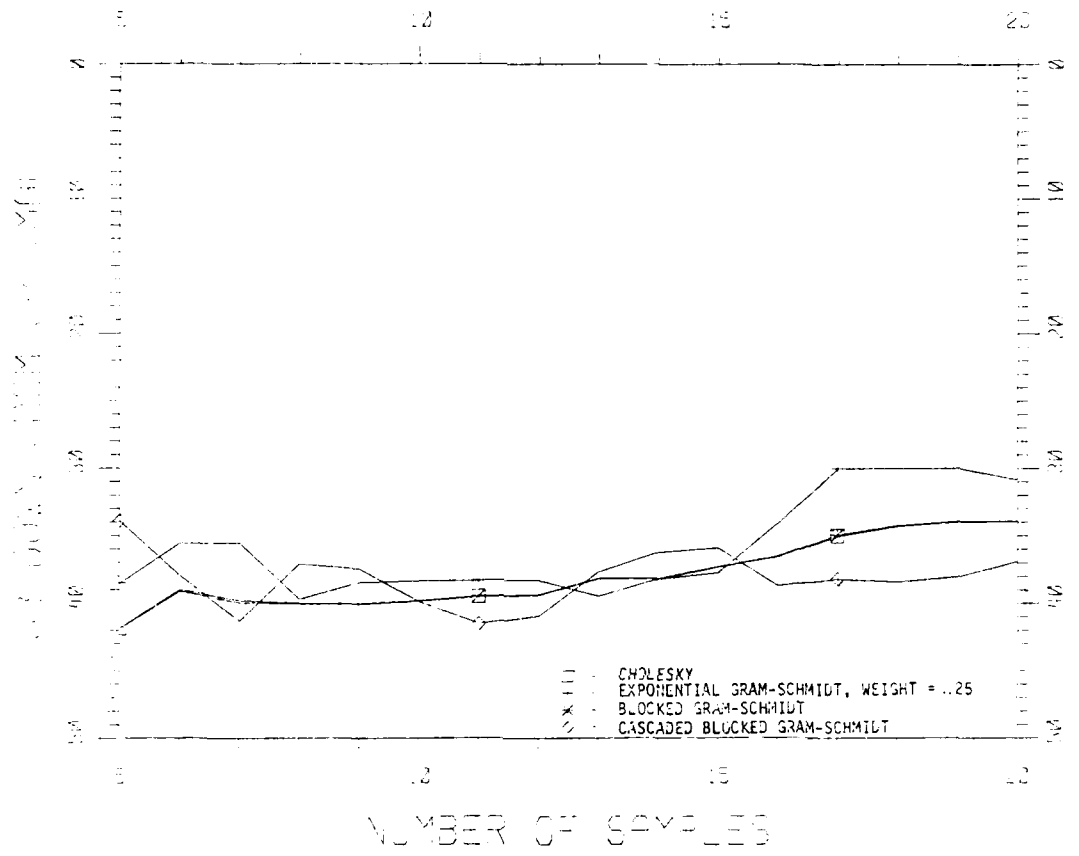


Figure 28. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channel 5 of total power 4 dB above strongest jammer.

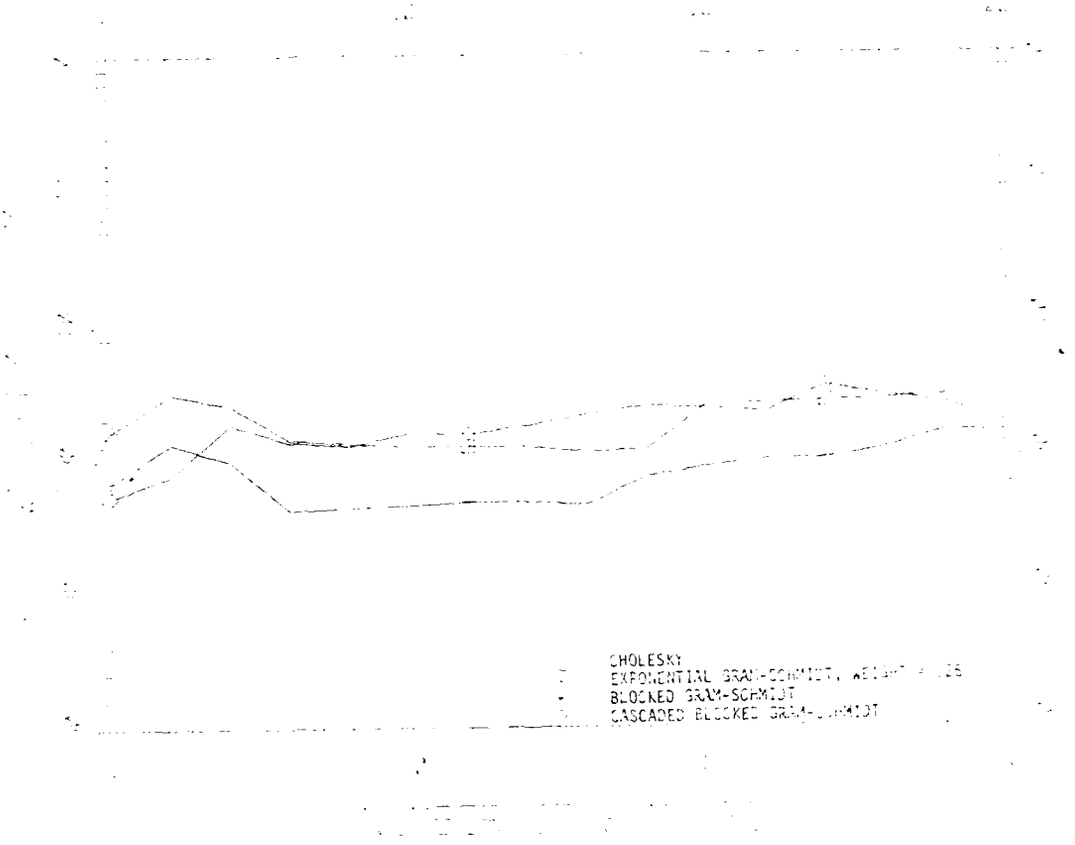


Figure 29. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channels 1, 2, 3 of total power 4 dB above strongest jammer.

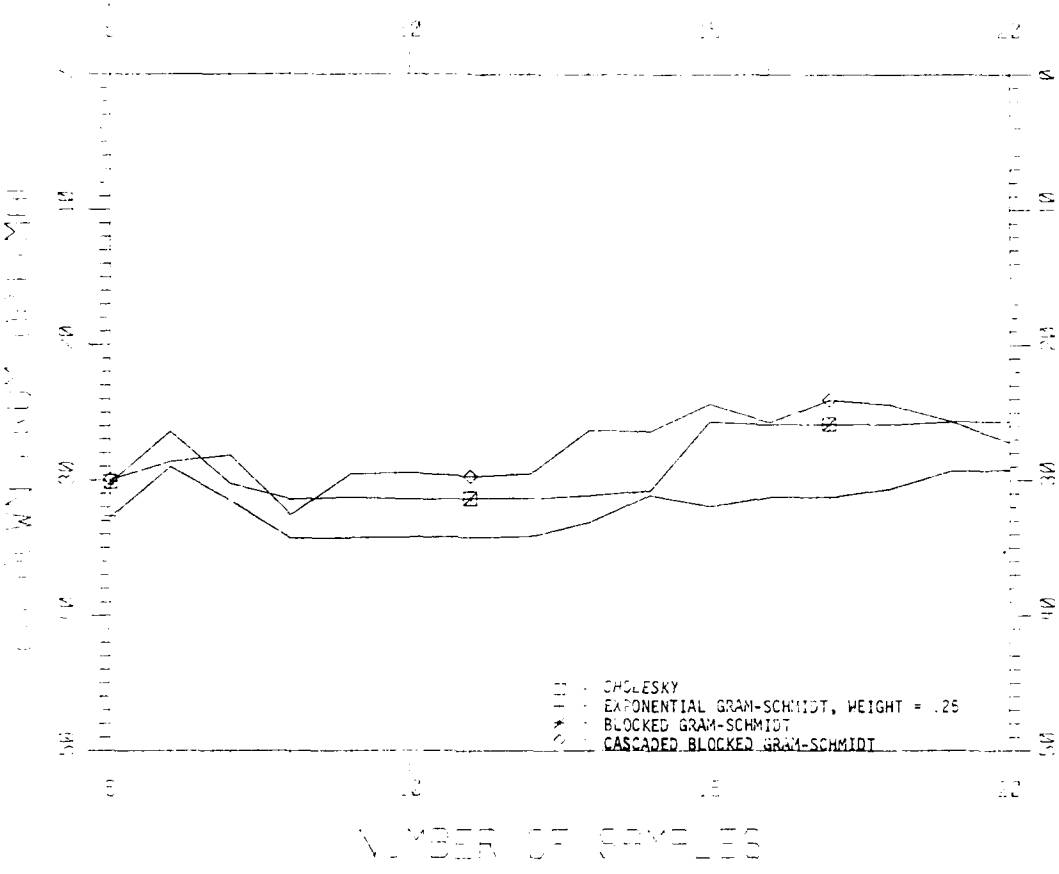


Figure 30. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channels 1, 2, 3, and 4 of total power 4 dB above strongest jammer.

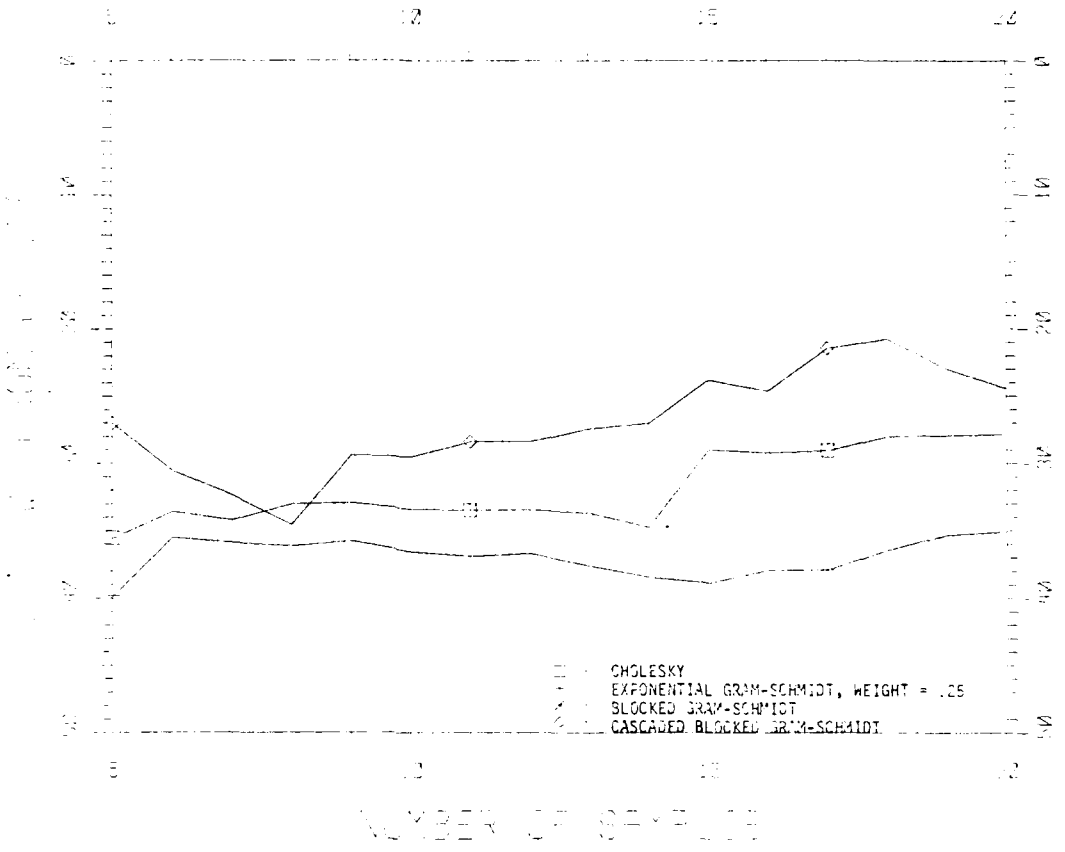


Figure 31. Simulated data:

1. Number of weights = 5
2. No tapped delays
3. Number of jammers = 3
4. Location and power of jammers:
   #1 (-36.9°, 6.67 dB)
   #2 (-11.5°, 3.33 dB)
   #3 (+11.5°, 0 dB)
5. Ratio of power of strongest jammer to receiver noise power = 40 dB
6. Noise added to channels 1, 2, 3, 4, and 5 of total power 4 dB above strongest jammer.

Internal Failure Mode

In the internal failure mode, processor P_{ij} outputs only constants. This case is similar to the external failure mode where input j is constant. Consider first the case where the output of P_{ij} is identically zero. Outputs 1 to j-1 remain unaffected. As in the external case, all the w_{kj}'s and outputs of P_{kj} for k>i are zero; so, w_{jj} is zero, and the rest of the array (rows j to n-1) operate as though the zero had been input at the very top of column j, and we have normal n-1 weight operation. The reverse flow situtation is similar, so that if P_{ij} fails by outputting only zero, it is equivalent to the external failure situation, where the array operates correctly as an n-1 weight system. When P_{ij} puts out a nonzero constant, incorrect weights are calculated, as with the external failure mode.

3.3 GROWTH OF INTERMEDIATE RESULTS

In this section, we derive probabilistic bounds on the growth of intermediate results during the computations.

We first consider the algorithm without square roots and using block averaging. We further subdivide the case into the situation where 1) the actual covariance matrix M is known (Figure 4a) and 2) where only samples are known (Figure 5a). When M is known, we have the following bounds on the $Z_j^i(k)$'s:

$$\lambda_{min} \leq Z_k^i(k) \leq m_{max} \quad k \geq i; \text{ these bounds are sharp} \qquad (32a)$$

$$|Z_j^i(k)| < 2 \cdot m_{max}; \text{ sharp within a factor of 4} \qquad (32b)$$

$$|Z_j^k(k)| \leq m_{max}; \text{ sharp within a factor of 2} \qquad (32c)$$

$$|w_{ij}| \leq \sqrt{\frac{m_{max}}{\lambda_{min}}}; \text{ sharp within a factor of 2} \quad , \qquad (32d)$$

where

$$m_{max} = \max_{ij} |m_{ij}| \quad (M = \{m_{ij}\}) \quad ,$$

$$\lambda_{min} = \text{smallest eigenvalue of M, and}$$

$$\lambda_{max} = \text{largest eigenvalue of M} \quad .$$

The value m_{max} can be bounded simply by using the number of bits and normalization used for each sample, and the number of samples used. If the largest sample value is X (in absolute value), and S samples are used, $m_{max} \leq SX^2$. Note that the sample matrix is S times as large as the true matrix (S = number of samples); because we do not divide the inner products, we calculate by S. The m_{max} used here is the maximum possible entry of the sample matrix and is S times as large as the maximum possible entry of the actual matrix.

Figure 32 plots the number of bits required to represent $\sum_{r=1}^{S} |X|^2$ where $|X| \leq 127$ (8-bit 2's complement representation). This plot puts an upper bound on the number of bits required to represent inner products. The simplest a priori bound on λ_{max} is $n \cdot m_{max}$; once the matrix is computed, the norms $||M||_1$, $||M||_\infty$, and tr (M) (trace M) (see Isaacson and Keller [1966]*) also provide upper bounds to λ_{max}. A lower bound for λ_{min} is provided by the level of receiver noise. Typically, ADC's (analog-to-digital converters, used to digitize the voltage inputs) are adjusted so that the receiver noise causes the least significant bit of the converted signal to be random; the receiver noise is therefore close to $S2^{-2t}X^2$, where t is the number of bits used in the ADC (this approximation may be off by a system-dependent factor close to 1). Thus $|w_{ij}| \leq \sqrt{SX^2/S2^{-2t}X^2} = 2^t$.

When using samples to compute the w_{ij}'s, we use the statistical interpretation of the receiver inputs being complex Gaussian random variables with zero mean and covariance matrix M. The intermediate quantities $Z'^i_j(k)$ are then also Gaussian random variables with zero means and computable variances. The quantity $\tilde{w}_{ij}$ has a complicated distribution, but for many samples we can approximate it by a Gaussian distribution, and $\tilde{\tilde{w}}_{ij}$ is similarly very close to a X^2 variable. The quotient of the two quantities, $w_{ij} = \tilde{w}_{ij}/\tilde{\tilde{w}}_{ij}$, also has a complicated distribution, and for this analysis we replace it by its expected value. The results are as follows:

$Z'^i_j(k)$ is a Gaussian random variable with zero means and variance $\text{var}[Z'^i_j(k)] = Z^i_j(j)$, where $Z^i_i(j)$ is computed as though the actual

*E. Isaacson and H. B. Keller, Analysis of Numerical Methods, New York, John Wiley & Sons, Inc., 1966.

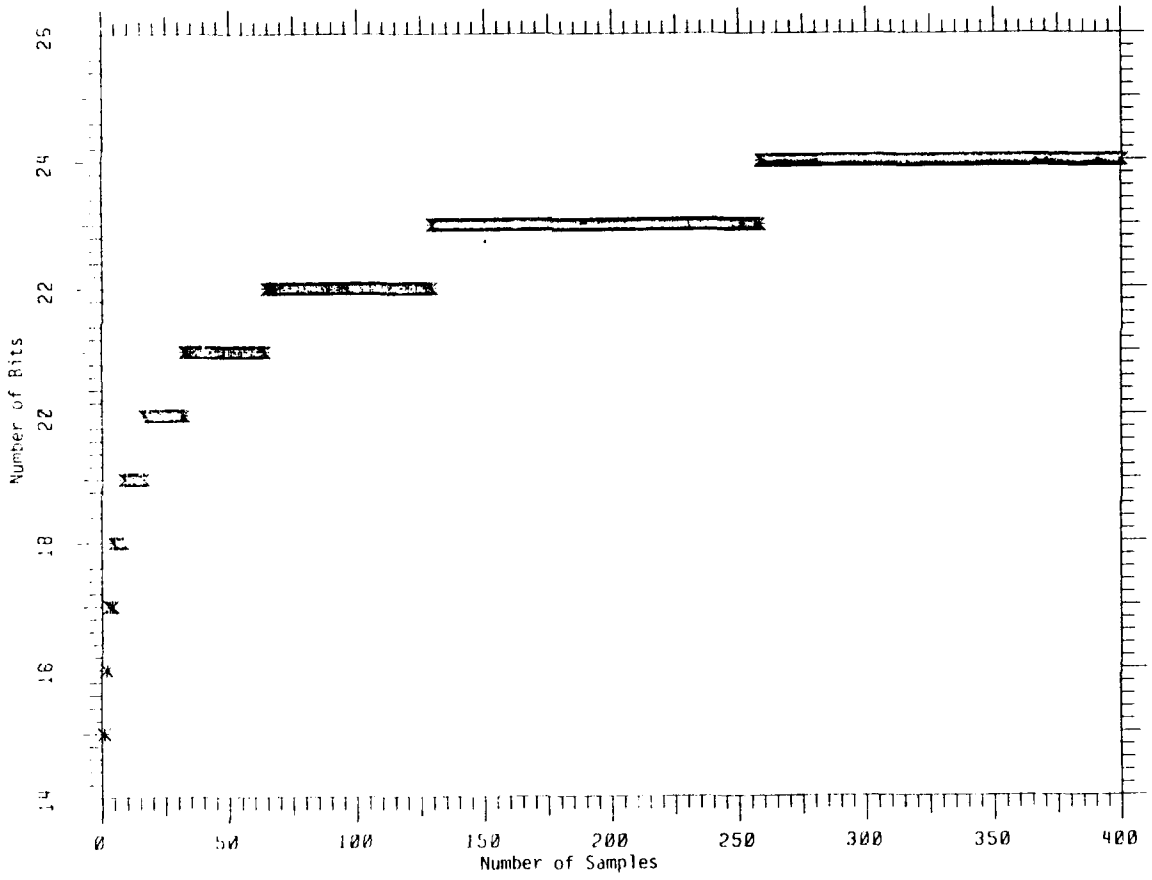


Figure 32. Number of bits required to represent inner products.

sample covariance matrix were being passed through. Thus $\lambda_{min} \leq \text{var}[Z'^{i}_{j}(k)] \leq m_{max}$, using the results of Eq. (32).

2. $\tilde{w}_{ij}$ is approximately Gaussian with mean $E(\tilde{w}_{ij}) = Z^{i}_{j}(i)$ (again as though the actual matrix were being passed); so $|E(\tilde{w}_{ij})| \leq m_{max}$, and variance $\text{var}(\tilde{w}_{ij}) \leq 2 \cdot m^2_{max}/S$ (S = number of samples).

3. $\tilde{\tilde{w}}_{ij}$ is approximately χ^2 with S degrees of freedom, a mean of $E(\tilde{\tilde{w}}_{ij}) = Z^{i}_{i}(i)$; so $|E(\tilde{\tilde{w}}_{ij})| \leq m_{max}$ and variance $\text{var}(\tilde{\tilde{w}}_{ij}) \leq 2 \cdot m^2_{max}/S$.

Thus, it is a simple matter to calculate the probability of overflow of the $Z'^{i}_{j}(k)$'s as a function of the number of bits, b, used to represent the $Z'^{i}_{j}(k)$'s in fixed point (where N(0,1) is a standard normal random variable):

$$\text{Probability of overflow} \leq P\left(|N(0,1)| \geq 2^b/\sqrt{m_{max}}\right) \quad .$$

If we let $C = b - \log_2\sqrt{m_{max}}$ = number of extra bits used beyond those needed to represent $\sqrt{m_{max}}$, we can make the following table:

C	0	1	2	3
Probability of overflow	$\leq$31.8%	$\leq$4.55%	$\leq 6.34 \times 10^{-3}$%	$\leq 1.24 \times 10^{-13}$%

We see that as C increases, the chances for overflow become negligible quickly.

Similarly, if C is the number of extra bits beyond those needed to represent m_{max}, then the probability of overflow when representing $\tilde{w}_{ij}$ is

$$\text{Probability of overflow} \leq P[|N(0,1)| \geq \sqrt{S/2}(2^C - 1)] \quad .$$

The following table evaluates the above function for various values of C and S.

S \ C	1	2	3
1	48.0%	3.6%	7.42×10^{-5}%
2	31.7%	0.27%	2.58×10^{-10}%
4	15.7%	2.14×10^{-3}%	$\sim2.\times10^{-21}$%
32	6.33×10^{-3}%	3.56×10^{-31}%	$\sim10^{-170}$%
400	1.59×10^{-42}%	$\sim10^{-383}$%	$\sim10^{-2171}$%

Again, the probability of overflow becomes negligible quickly.

If C is the number of extra bits needed beyond those to represent m_{max}, then for the probability of overflow when computing $\tilde{\tilde{w}}_{ij}$ to be $\leq 10^{-4}$, we need the following number of extra bits:

C	3	2	1
Range of S	1-3	4-48	$49-\infty$

This may be derived using $[P(\tilde{\tilde{w}}_{ij} \geq 2^C m_{max}) = P(\chi^2_S \geq 2^C S)]$.

Finally, when passing a steering vector through the array, it must be remembered that if S is an eigenvector of M corresponding to λ_{min}, then $W = M^{-1}S = (1/\lambda_{min})S$, so that quantities on the order of $1/\lambda_{min}$ must be introduced in the computation. These numbers of magnitudes potentially much larger than those already discussed can be introduced either during division by $\langle Z_i^i, Z_i^i \rangle$ at the right of the array or during back substitution. Thus the unit-vector and transform-space methods might have an advantage over the reverse flow method because they do not require passing numbers as large

as $1/\lambda_{min}$ back through the array. The unit vector method, of course, passes numbers as large as $\left(m_{max}/\lambda_{min}\right)^{\frac{1}{2}} = \max_{ij} |w_{ij}|$ through the array. Still, the transform space method and unit vector method might be able to use the simpler fixed-point arithmetic in the triangular array and then only have floating points in the outboard processors.

The analysis of the method with square roots is very similar to the above analysis. The $Z'_j(k)$'s, $\tilde{w}_{ij}$, and $\tilde{\tilde{w}}_{ii}$ have the same distributions as before, but the $1/\sqrt{\tilde{w}_{ii}}$ quantity broadcast is new. It is easy to derive its distribution from that of $\tilde{\tilde{w}}_{ij}$. In fact, since $\tilde{\tilde{w}}_{ii}$ is χ^2 with S degrees of freedom and mean between λ_{min} and m_{max}, we have the probability that the most significant digit of $1/\sqrt{w_{ii}}$ is more than C bits to the right of the most significant bit of $1/\sqrt{m_{max}}$ is

$$P\left(1/\sqrt{\tilde{\tilde{w}}_{ii}} \leq 2^{-C} \cdot 1/\sqrt{m_{max}}\right) \leq P\left(\chi^2_S \geq S2^{2C}\right) \quad ,$$

which is very small, as can be seen from the last table. Also, the probability that $1/\sqrt{\tilde{\tilde{w}}'_{ii}}$ will need more than C bits more than $1/\sqrt{\lambda_{min}}$ is

$$P\left(1/\sqrt{\tilde{\tilde{w}}_{ii}} \geq 2^{C} \cdot 2^{t}/\sqrt{m_{max}}\right) \leq P\left(\chi^2_S \leq 2^{-2C}S\right) \quad ,$$

which is also very small for C=1 and $S \geq 20$ (probability $\leq 2.8 \times 10^{-2}\%$).

The results presented in this section are proved in Appendix G.

3.4 GRAM-SCHMIDT PROCESSING WITH O(n) PROCESSORS

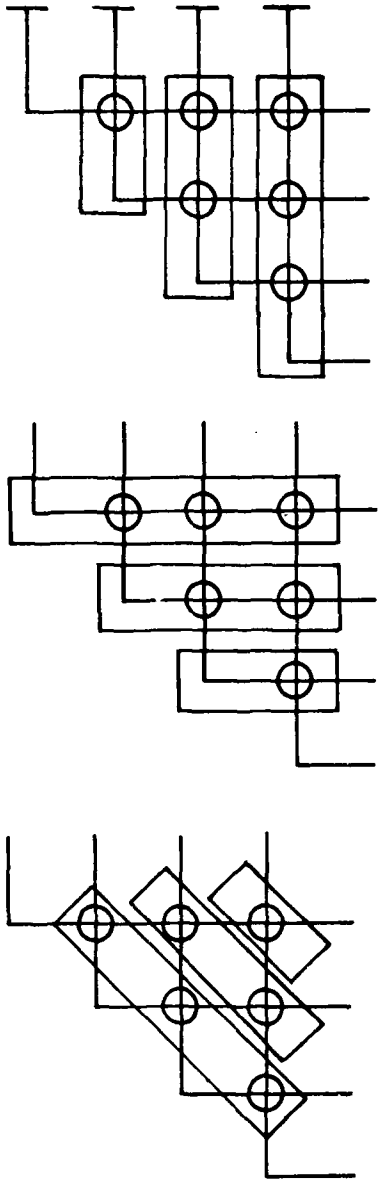
The Gram-Schmidt array with $O(n^2)$ processors can also be implemented with only O(n) processors. This implementation can be performed by collapsing the array in the vertical, horizontal, or diagonal dimension (Figure 33); i.e., letting the work of all processors in a column, in a row, or along a diagonal be performed by a single processor.

Each processor performs the same calculations as the processors in the $O(n^2)$ array but on multiple data sets. Three main differences between the three O(n) implementations are: 1) work is not equally shared by the processors; 2) they have different internal-memory requirements; and 3) the output may be routed internally to the same processor or externally to another processor. All of these will be discussed in more detail. The processor diagram for O(n) processors is shown in Figure 34.

As can be seen in Figure 33 for the case of four inputs, one processor must perform the work of from one to three processors of the original $O(n^2)$ implementation. This mismatch is increased as n increases. If W is the amount of work that one processor in an $O(n^2)$ array must perform, then the amount of work for a processor in an O(n) configuration can range from W to (n-1)W. Since the total time in a parallel system is determined by the slowest processor, then, on the average, half of the processors are idle.

The internal memory for each processor must be increased because it must store all of the information required for the calculations to be performed. This memory increase should, however, be small, mainly the inputs and internal weights, which amount to approximately 4(n-1) complex words.

In an $O(n^2)$ array, each processor is independent of its position. In an O(n) array, however, the processors must be "aware" of the interconnect

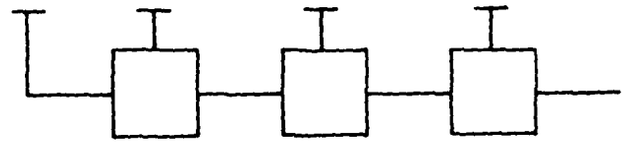


a. Vertical collapsing.

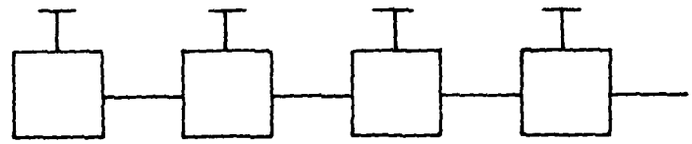
b. Horizontal collapsing.

c. Diagonal collapsing.

Figure 33. Collapsing an array into O(n) processors. The boxes represent work performed by a processor.



a. Standard configuration.



b. Alternative configuration for symmetry.

Figure 34. Configuration using O(n) processors.

structure so that they will know where to transfer the processor data. The controller can handle this requirement by specifying the routing and, as Figure 34 shows, all routing is regular. The routing problem is not serious and should not prevent consideration of this method.

3.4.1 Advantages and Disadvantages

The obvious disadvantage of O(n) processors is speed. The actual timing models are developed in Appendix E, but time is a function of many variables. In a block average system, most of the time is spent calculating the internal weights; in this case, an O(n) system is comparable to an $O(n^2)$ system (in actual practice, O(n) and $O(n^2)$ systems can be equal). The $O(n^2)$ system is advantageous when there are a large number of steering vectors. The determination of which system to use should be based on the relationship between the number of samples, S, and the number of steering vectors, K.

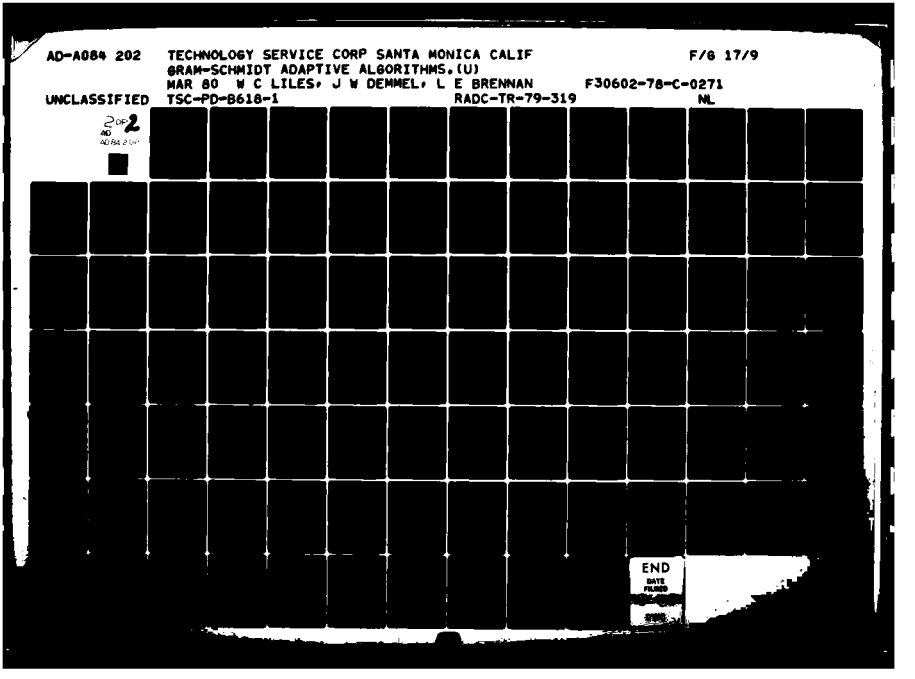
Fault tolerance and reliability offer potential advantages and disadvantages. The O(n) system can be much more reliable because of fewer parts; however, because of the increased demand on each processor, faster hardware operating in critical conditions may be required.

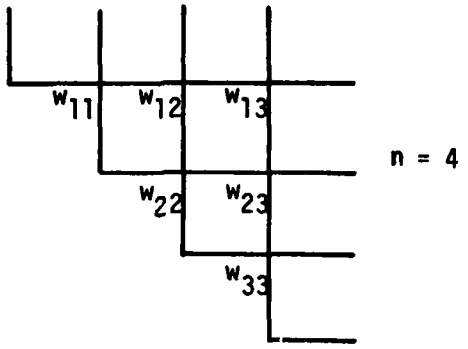
3.4.2 Tradeoffs Between the Three O(n) Designs

Which system, from Figure 33, should be used in a radar installation depends on many factors. We discuss briefly the major tradeoffs for each array.

Since Section 3.2 showed that if any column of the array became error-prone or stopped working, the system could still operate, the system in Figure 33a would be chosen for fault tolerance because one processor corresponds to one column. In the system in Figure 33b, if the first processor went bad, the entire system would become ineffective which is also true of that in Figure 33c.

Diagonal collapsing, Figure 33c, has the advantage of being able to implement reverse flow without requiring backward busses. This implementation is accomplished by transposing the internal weights and using forward flow. The transposition is simple because each processor contains all the internal weights for the transposition (Figure 35). This transposition is more difficult for the other configurations, but reverse-flow busses are cheaper to implement for O(n) systems because of fewer processors.


AD-A084 202 TECHNOLOGY SERVICE CORP SANTA MONICA CALIF F/G 17/9
GRAM-SCHMIDT ADAPTIVE ALGORITHMS,(U)
MAR 80 W C LILES, J W DEMMEL, L E BRENNAN F30602-78-C-0271
UNCLASSIFIED TSC-PD-8618-1 RADC-TR-79-319 NL
2 OF 2
AD
AD A084 202
END
DATE
FILMED



Transpose $i \rightarrow n + 2 - j$

$j \rightarrow n + 2 - i$

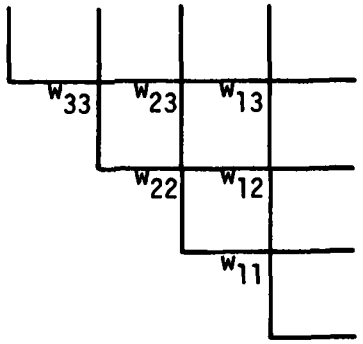


Figure 35. Transposing internal weights.

3.5 PROCESSOR INTERNAL PIPELINE/PARALLELISM

Each processor must compute the inner product of its two inputs in order to compute the internal weights. Once the internal weight has been computed, it is applied against the processor inputs to compute the outputs. In this section we discuss different methods of implementing these computations. Block averaging is assumed. Complex multiplication is performed with 4 multiplies and 2 adds. Complex addition is performed with 2 adds.

3.5.1 Compute Inner Product

The mathematical formula for calculating the numerator to determine the weights is

$$\sum_{i=1}^{S} X^{*}Y \quad ,$$

where X and Y are input samples and S is the number of samples. Typically $S \geq 2n$, where n is the number of adaptive weights. The primitive operations consist of multiply and add/sub. Let the time per operation be t. Input time is assumed to be 1/2t.

The techniques examined are:

Sequential

Sequential with overlap

Full pipeline

Parallel/2 multipliers

Parallel/4 multipliers

Pipeline/2 parallel multipliers

Pipeline/4 parallel multipliers

Optimal

Table 1 provides a comparison of the numerators computed by the above implementation techniques.

TABLE 1. COMPARISON OF INNER-PRODUCT NUMERATOR IMPLEMENTATION TECHNIQUES

Technique	No. of Multipliers	No. of Adders	Time for S Samples	Time Between Samples	Increase in Complexity
Sequential	1	1	10St	10t	↓
Sequential with Overlap	1	1	(6S + 2)t	6t	
Full Pipeline	4	4	10t + (S-1)t	t	
Parallel/ 2 Multipliers	2	1	8St	8t	
Parallel/ 4 Multipliers	4	4	7St	7t	
Pipeline/2 Parallel Multipliers	4	4	8t + (S-1)t	t	
Pipeline/4 Parallel Multipliers	4	4	5t + (S-1)t	t	
Optimal	4	4	4t + (S-1)t	t	

Sequential

In this case we have one multiplier and one adder with no overlap:

read X real
read X imaginary
read Y real
read Y imaginary
multiply X_R Y_R
multiply X_I Y_I
add
add add to running sum real
multiply X_R Y_I
multiply X_I Y_R
add
add add to running sum imaginary

Number of multipliers: 1
Number of adders: 1
Time for 5 samples: 10 St
Time between samples: 10 t

Sequential with Overlap

This is a slight modification of the sequential method, but the adder and multiplier are separate functional units, which enables both to be used simultaneously:

read X real add $X_IY_R + X_RY_I$ from previous sample
read X imaginary
read Y real add to running sum imaginary
read Y imaginary
multiply X_R Y_R
multiply X_I Y_I
multiply X_R Y_I add $X_RY_R + X_IY_I$
multiply X_I Y_R add to running sum real

Number of multipliers: 1
Number of adders: 1
Time for S samples: (6S +2)t
Time between samples: 6t

Full Pipeline

This method is similar to sequential, except that each stage is implemented so that multiple data streams can be executed simultaneously. We must also change the I/O either by transferring 2 words in parallel on each bus or making the bus operate at time 1/2t. We chose the latter for this analysis.

read X real
read X imaginary
read Y real
read Y imaginary
multiply X_R Y_R
multiply X_I Y_I
add
add
multiply X_R Y_I
multiply X_I Y_R
add
add

No. of multipliers: 4
No. of adders: 4
Time for S samples: 10t + (S-1)t
Time between samples: t

Parallel/2 Multipliers

Since complex multiply can be done in parallel, we connect two multipliers in parallel:

read X_R
read X_I
read Y_R
read Y_I
multiply
add
add
multiply
add
add
No. of multipliers: 2
No. of adders: 1
Time for S samples: 8St
Time between samples: 8t

Parallel/4 Multipliers

The next modification is to implement a 4-word-wide multiplier and a 2-word-wide adder.

read X_R
read X_I
read Y_R
read Y_I
multiply
add
multiply
add
add

No. of multipliers: 4
No. of adders: 4
Time for S samples: 7St
Time between samples: 7t

Pipeline/2 Parallel Multipliers

This is the pipeline method with 2 parallel multipliers at the multiplier stages:

read X_R
read X_I
read Y_R
read Y_I
multiply
add
add
multiply
add
add

No. of multipliers: 4
No. of adders: 4
Time for S samples: 8t + (S-1)t
Time between samples: t

Note that in this case the number of multipliers has not been increased, and time saved is only 2t, buth with added processor complexity.

Pipeline/4 Parallel Multipliers

This method has a full complex multiplier (4 multipliers) and complex adder (2 adders) implemented in parallel.

read X_R
read X_I
read Y_R
read Y_I
multiply
add
add

No. of multipliers: 4
No. of adders: 4
Time for S samples: 5t + (S-1)t
Time between samples: t

For the large additional complexity in the processor, this method is only 5t faster than full pipelined and 3t faster than the pipeline/2 parallel multiplier method.

Optimal

The parallel and pipeline methods explained above can take advantage of overlap simultaneously, as in the sequential method. The minimum system would use overlap in a full parallel sense. This system would look as follows for time slice t_i :

read $X_R(t_i)$
read $Y_R(t_i)$
multiply for t_{i-1}
add for t_{i-2}
add for t_{i-3}
read $X_I(t_i)$
read $Y_I(t_i)$

No. of multipliers: 4
No. of adders: 4

Time for S samples: $4t + (S-1)t$
Time between samples: t

The time for S samples is only a savings of t over the previous method.

A graphical comparison of the times required by all the above methods versus the number of samples is given in Figure 36.

3.5.2 Calculate Weights

In Section 3.5.1 we discuss the calculation of the numerator to determine the weights. We now discuss the calculation of the denominator and the division.

The denominator is calculated using the formula $\sum_{i=1}^{S} X^*X$. Since X times its conjugate is a real number, only the real part needs to be computed. This calculation can be performed either in parallel with the computation of the numerator or in sequential order. We have the following table for parallel computation (Table 2). The effective rate shown in Table 1 is not always attained. If the denominator calculation is carried out in parallel with the numerator calculation by separate processors, then the sampling rate will be governed by the slowest of the two processes because the data will be placed on the bus once for all processors. In the case of the sequential processing we have a time between samples of 10t and 5t for the denominator and numerator calculations, respectively. The actual system time between samples will therefore be 10t. Because of this dependency on the numerator calculation, we can use the same processor to compute the denominator without serious time penalties. If the denominator is calculated sequentially in the same processor as the numerator, time t can be saved on inputting the value X.

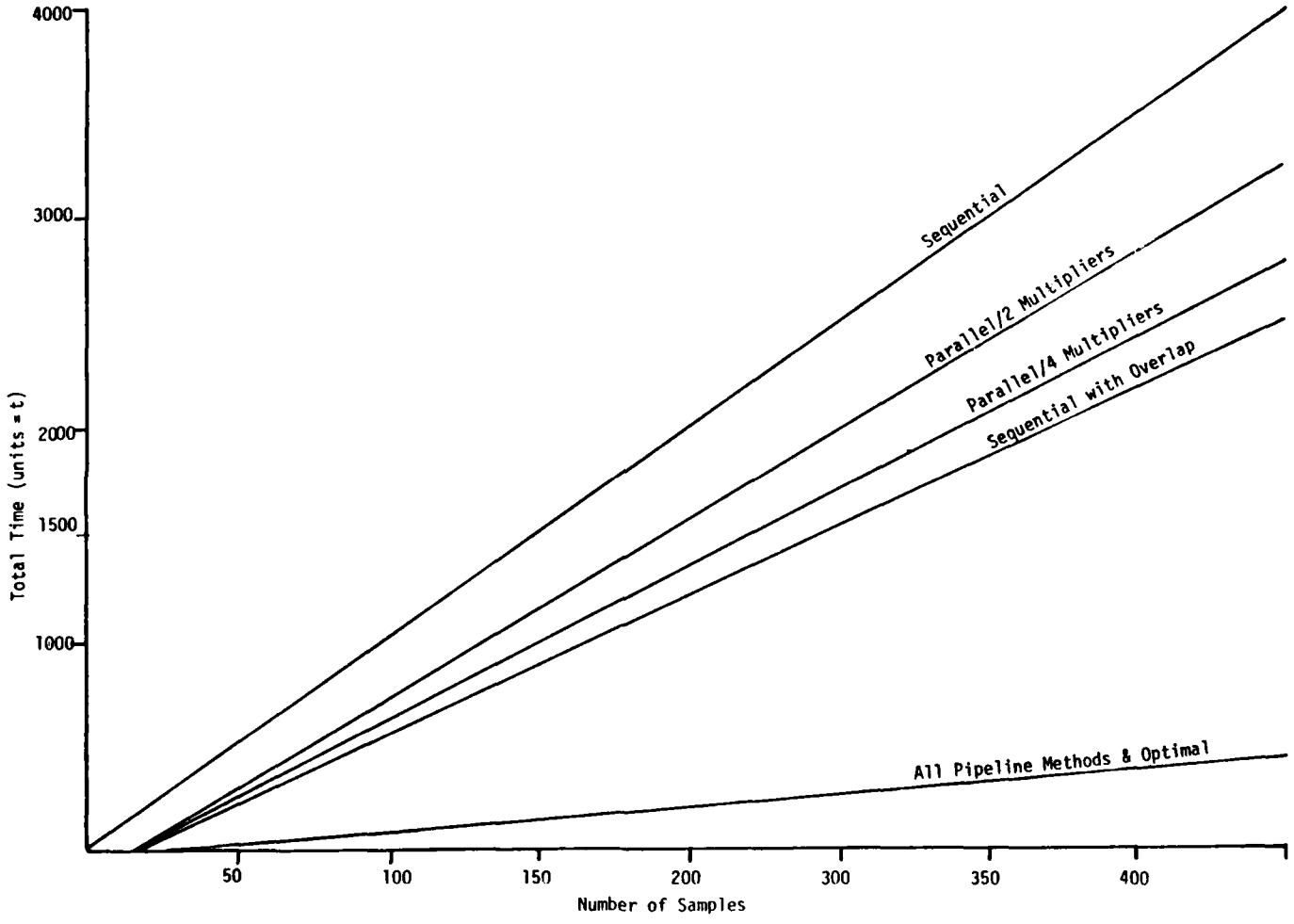


Figure 36. Inner-product computation time.

TABLE 2. COMPARISON OF INNER-PRODUCT DENOMINATOR IMPLEMENTATION TECHNIQUES

Technique	No. of Multipliers	No. of Adders	Time for S Samples	Time Between Samples
Sequential	1	1	5St	5t
Sequential with Overlap	1	1	(2S + 2)t	2t
Full Pipeline	2	2	5t + (S-1)t	t
Parallel/2 Multipliers	2	1	4St	4t
Parallel/4 Multipliers	Not Applicable			
Pipeline/2 Parallel Multipliers	2	2	4t + (S-1)t	t
Pipeline/4 Parallel Multipliers	Not Applicable			
Optimal	2	2	4t + (S-1)t	t

The division can be performed by reciprocal calculation and broadcasting the value in the horizontal direction. Assuming no overlap, the time to compute the reciprocal is the time required to perform one table lookup and two iterations of Newton-Raphson (see Appendix D). The program to find 1/C would be as follows:

Table lookup		X_1 estimate
Multiply	CX_1	
Sub	$2-CX_1$	
Multiply	$(2-CX_1)X_1$	X_2 new estimate
Multiply	CX_2	
Sub	$2-CX_2$	
Multiply	$(2-CX_2)X_2$	$X_3 = 1/C$

Time = 7t

To broadcast the value and multiply by the reciprocal in each processor requires the following code in each processor:

```
Read reciprocal
Multiply
```

Time = 2t

3.5.3 Apply Weight to Input Data

The formula to apply weights, w, is output = X - WY. This is the same form as the inner-product calculation, which can be expressed as

$$sum = sum + XY$$

Because of this similarity the programs are similar, as shown by the sequential method:

```
read  X_R
read  X_I
read  Y_R
read  Y_I
```

multiply $W_R Y_R$

multiply $W_I Y_I$

sub

sub X_R - $(W_R Y_R - W_I Y_I)$

multiply $W_R Y_I$

multiply $W_I Y_R$

add

sub X_I - $(W_R Y_I + W_I Y_R)$

out X_R

out X_I

Assuming output takes time 1/2t, all of the previous formulas are applicable if a step of t is added; i.e., 8t + (S-1)t becomes 8t + (S-1)t + t = 9t + (S-1)t.

3.6 DESIGN ALTERNATIVES AND TRADEOFFS

This section summarizes the variety of system configurations described in detail in the first part of this report (Sections 2.6 and 3.1 through 3.5), and indicates how various radar engineering considerations might influence the choice of configuration. Choosing the best configuration is a complicated task and depends heavily on the details of a given radar system; therefore, we only summarize some of the tradeoffs. In particular, we conclude that while a universal element may exist, it might be far from optimal for any individual system, since it would have to be designed for all possible worst-case situations, and hence be very expensive, large, and power-consuming.

The different design alternatives are summarized in Table 3. (For more details, see the indicated sections.)

The unit vector method requires simpler hardware than the reverse flow method, but is slower, solving m $n \times n$ systems in time $O(mn)$ instead of $O(m+n)$. The transform space method requires the simplest hardware, but every sample from which filter functions are to be computed must be passed through the array, which means there can be fewer filter function evaluations per unit time than with the other two methods.

The different means of calculating the weights not only require different amounts of hardware (most for window averaging, least for cascaded block) but have different statistical properties, requiring different numbers of samples to converge or update old estimates, and with different lag times.

The higher performance desired, or the smaller probability of overflow desired, the more bits are needed; and depending on the basic implementation chosen, it may be possible to have fewer bits in one part of the processor

TABLE 3. DESIGN ALTERNATIVES

1) Basic Implementation (See Section 2.6)
   a) Unit Vector Method
   b) Reverse Flow Method
   c) Transform Space Method

2) How to Compute Inner Products (See Section 3.1)
   a) Block Averaging
   b) Cascaded Block Averaging
   c) Exponential Averaging
   d) Window Averaging
   e) Number of Samples

3) Arithmetic Used (See Section 3.3)
   a) Fixed, Floating Point, Block Floating Point
   b) Number of Bits
   c) With or Without Square Roots

4) n Versus n^2 Processor Implementation (See Section 3.4)
   a) Horizontal, Vertical or Diagonal Collapsing
   b) Unit Vector, Reverse Flow, or Transform Space Method

5) How to Implement Each Individual Processor (See Section 3.5)

than another, and even have different kinds of arithmetic (fixed versus floating point) in different parts of the processor.

The choice between n and n^2 processors depends on the speed desired and cost considerations. The different O(n) implementations have different fault-tolerance properties, require processors of differing complexities, and differ in how compatible they are with the three basic implementations.

How to implement each individual processor also depends on the speed required and cost limits, and is certainly dependent on the number of bits and type of arithmetic chosen.

Some of the most important radar engineering factors affecting the choice of system configuration are given in Table 4. We discuss how they affect the choice, but remind the reader that the list is not meant to be exhaustive, nor is there necessarily a single-best configuration for all operating modes of a given radar system.

TABLE 4. RADAR ENGINEERING CONSIDERATIONS AFFECTING CHOICE OF SYSTEM CONFIGURATION

1)	Number of Weights
2)	Sampling Rate
3)	Electronic Versus Mechanical Scan
4)	Operating Modes
5)	Size of a PRI; Amount of Dead Time
6)	Fully Adaptive Versus SLC
7)	Number of Bits in the ADC
8)	Amount of Clutter
9)	Performance Required (in Terms of SNR Achieved)
10)	Cost Requirements
11)	Reliability Requirements
12)	Environment

How fast the array must process depends on the number of weights, sampling rate, type of scanning used, system mode, and the amount of dead time. A higher sampling rate requires either a faster array or downsampling. If the sampling rate is slow enough, the transform space method might be sufficiently fast. If there are many steering signals for a given covariance matrix (e.g., electronic scan), then reverse flow might be preferable to the unit vector method. If there is a small sampling window and large amount of dead time in each PRI, simple block averaging, with its long startup, might be sufficient instead of a more expensive implementation like window averaging. If overall speed requirements are low, n processors instead of n^2 might suffice.

If the system is a SLC instead of fully adaptive, some of the arithmetic can be simplified.

More bits on the ADC means more expensive hardware to retain accuracy. If there is so much clutter in the system that there are not enough weights to adapt to it, then more than the usual 2n samples may be required to get good performance. In general, the higher performance desired, the more bits of accuracy and the more samples required.

The many design alternatives discussed have widely varying costs (n versus n^2 processors, and number of bits carried for example), so there are many speed/accuracy/performance versus cost tradeoffs.

The different designs have different fault-tolerance properties, which may also vary with the type of fabrication techniques used.

The radar environment will influence the design greatly. A land-based system with large amounts of power, cooling, and spare parts available will

certainly have less stringent packaging, power, and reliability constraints than one that is used in the field.

The dimension of the design space can be reduced for problems of interest so that a universal adaptive algorithm (UAA) element does exist. One such breakdown is shown in Table 5. For this radar system, the UAA element would be near optimal for most configurations. For the 30-weight system, 21 bits are required; for the 50-weight system, 22 bits are required for inner-product calculation. But due to MSI and LSI integration sizes, either a 22-bit or even a 24-bit system would be implemented.

The system designer must decide which subset of all possible adaptive systems the processors must support.

TABLE 5. SPECIFIC RADAR DESIGN PARAMETERS

1)	Number of Weights - 30 to 50
2)	Sampling Rate - 20 MHz
3)	Electronic Scan
4)	Search and Track Modes
5)	Typical Times for Land-Based Surveillance Systems
6)	Fully Adaptive
7)	8-Bit 2 Complément Complex ADC
8)	Low Center
9)	3 to 8 dB Down from Optimal SNR
10)	Low Cost
11)	24-Hour Continuous Duty Every Day
12)	Land-Based Stationary Environment Constructed to Support Processor

3.7 UNIVERSAL ALGORITHM HARDWARE IMPLEMENTATION

The Gram-Schmidt processor has a very simple interconnect structure regardless of implementation. The bus structure for the pipelined ($O(n^2)$) configuration is shown in Figure 37. All buses have a single source, making this configuration's the simplest interconnect protocol. The recursive ($O(n)$) configuration is shown in Figure 38. It has a slightly more complicated interconnect protocol because one interconnect bus has several sources, which implies that each module must know when it should place its data on the bus. This bus can be implemented with either open-collector or tri-state logic. A comparison of the two configurations shows that each processing element (PE) has the same inputs and outputs; thus, with the addition of bus-sharing logic, a processor can be made to run in either configuration.

The bus interconnects may be either one-half of a complex word wide or a full complex word wide. The latter provides the potential for a higher processing rate.

A variation combining the $O(n^2)$ and $O(n)$ configurations may be implemented. This variation is shown in Figure 39 for the five-input case. It enables the system designer to meet the speed requirements without using more hardware than is really needed, a good alternative solution for large n when the $O(n^2)$ configuration would contain a prohibitive number of processors. The processor control is no more difficult than for the $O(n)$ configuration.

The configuration chosen by the system designer is a function of 1) the processor speed; 2) the number of degrees of freedom required; and 3) the required processing rate, which, in general, will be slower than the sampling rate.

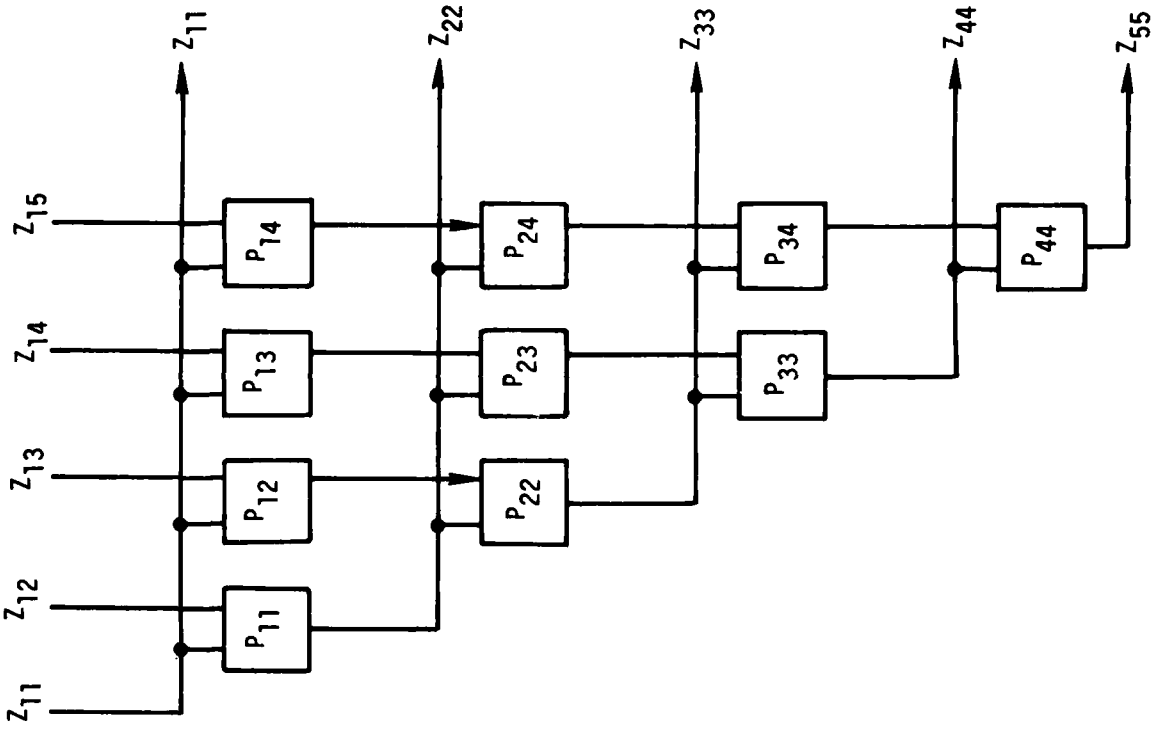


Figure 37. Bus structure for pipelined ($O(n^2)$) configuration.

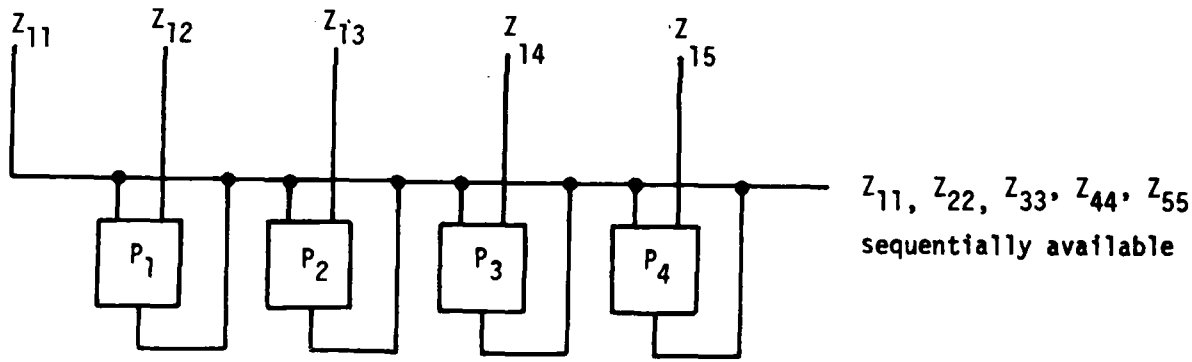


Figure 38. Bus structure for recursive (O(n)) configuration.

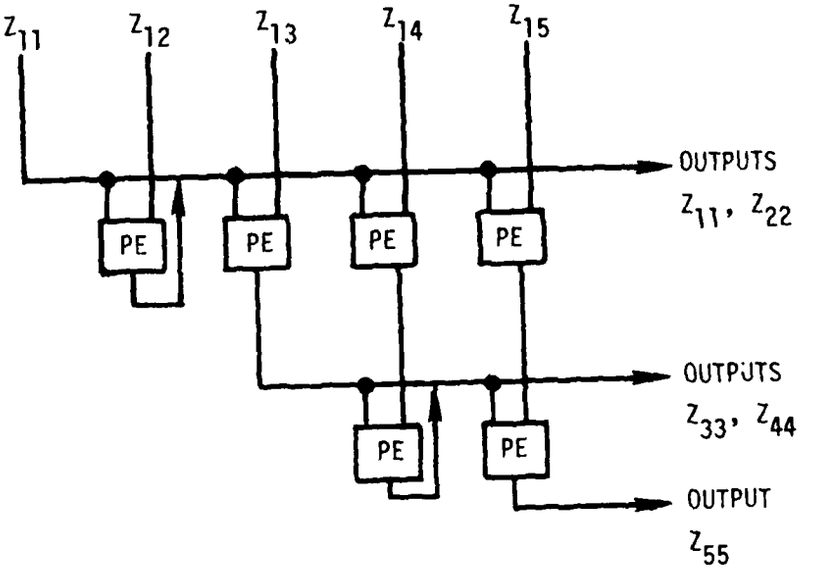


Figure 39. Variation combining $O(n^2)$ and $O(n)$ configurations. Five-input case is shown.

The processor configuration is governed by the Gram-Schmidt equations, shown in Figure 40. They have been broken down into their real and imaginary components. The majority of the operations performed are add, subtract, and multiply. Only one division is required after all the terms have been summed. The divisor in all processors in a row will be identical. If this process is moved to a dedicated module in each row, n-2 sets of redundant logic have been removed from each row. This removal will reduce the amount of logic per PE, reducing each PE's cost and mean-time-between-failures (MTBF). For a system with a large number of degrees of freedom, this could result in considerable savings in system cost and greatly improve the system's MTBF. The control becomes no more difficult and the bus structure does not change for either the $O(n^2)$ or $O(n)$ configurations (Figures 41 and 42).

The configuration discussed in the paragraph above is expanded here to show a possible hardware implementation. Two generic processing elements are shown. First, the node processing element (NPE) computes all equations except the divisor equation. This processor is present at all the nodes in the Gram-Schmidt array. The second processor, the diagonal processing element (DPE), computes the divisor and its inverse, and then broadcasts the inverse to the NPEs in its row. There is one DPE for each row of the Gram-Schmidt array.

A block diagram for a node processing element (NPE) is shown in Figure 43. This processor does not include the ability to calculate the divisor. This NPE can perform block and exponential averaging. In addition, it can operate in either the $O(n^2)$ or $O(n)$ configurations. Because of its generality, the NPE would not be practical to implement for use in a real

Block G-S Equations

$$\mathrm{Re}(\tilde{w}) = \mathrm{Re}(\tilde{w}) + \mathrm{Re}(Z_i^i)\mathrm{Re}(Z_j^i) + \mathrm{Im}(Z_i^i)\mathrm{Im}(Z_j^i)$$

$$\mathrm{Im}(\tilde{w}) = \mathrm{Im}(\tilde{w}) + \mathrm{Re}(Z_i^i)\mathrm{Im}(Z_j^i) - \mathrm{Im}(Z_i^i)\mathrm{Re}(Z_j^i)$$

$$\tilde{\tilde{w}} = \tilde{\tilde{w}} + \mathrm{Re}(Z_i^i)\mathrm{Re}(Z_i^i) + \mathrm{Im}(Z_i^i)\mathrm{Im}(Z_i^i)$$

$$\mathrm{Re}(w) = \mathrm{Re}(\tilde{w})(1/\tilde{\tilde{w}})$$

$$\mathrm{Im}(w) = \mathrm{Im}(\tilde{w})(1/\tilde{\tilde{w}})$$

$$\mathrm{Re}(Z_j^{i+1}) = \mathrm{Re}(Z_j^i) - [\mathrm{Re}(w)\mathrm{Re}(Z_i^i) - \mathrm{Im}(w)\mathrm{Im}(Z_i^i)]$$

$$\mathrm{Im}(Z_j^{i+1}) = \mathrm{Im}(Z_j^i) - [\mathrm{Re}(w)\mathrm{Im}(Z_i^i) + \mathrm{Im}(w)\mathrm{Re}(Z_i^i)]$$

Exponential G-S Equations

$$\mathrm{Re}(w_n) = \mathrm{Re}(w_{n-1})S + [\mathrm{Re}(Z_i^i)\mathrm{Re}(Z_j^i) + \mathrm{Im}(Z_i^i)\mathrm{Im}(Z_j^i)][(1/\tilde{\tilde{w}}_{n-1})(1 - S)]$$

$$\mathrm{Im}(w_n) = \mathrm{Im}(w_{n-1})S + [\mathrm{Re}(Z_i^i)\mathrm{Im}(Z_j^i) - \mathrm{Im}(Z_i^i)\mathrm{Re}(Z_j^i)][(1/\tilde{\tilde{w}}_{n-1})(1 - S)]$$

$$\tilde{\tilde{w}}_n = \tilde{\tilde{w}}_{n-1}S + [\mathrm{Re}(Z_i^i)\mathrm{Re}(Z_i^i) + \mathrm{Im}(Z_i^i)\mathrm{Im}(Z_i^i)](1 - S)$$

$$\mathrm{Re}(Z_j^{i+1}) = \mathrm{Re}(Z_j^i) - [\mathrm{Re}(w_{n-1})\mathrm{Re}(Z_i^i) - \mathrm{Im}(w_{n-1})\mathrm{Im}(Z_i^i)]$$

$$\mathrm{Im}(Z_j^{i+1}) = \mathrm{Im}(Z_j^i) - [\mathrm{Re}(w_{n-1})\mathrm{Im}(Z_i^i) + \mathrm{Im}(w_{n-1})\mathrm{Re}(Z_i^i)]$$

Figure 40. Gram-Schmidt (G-S) equations

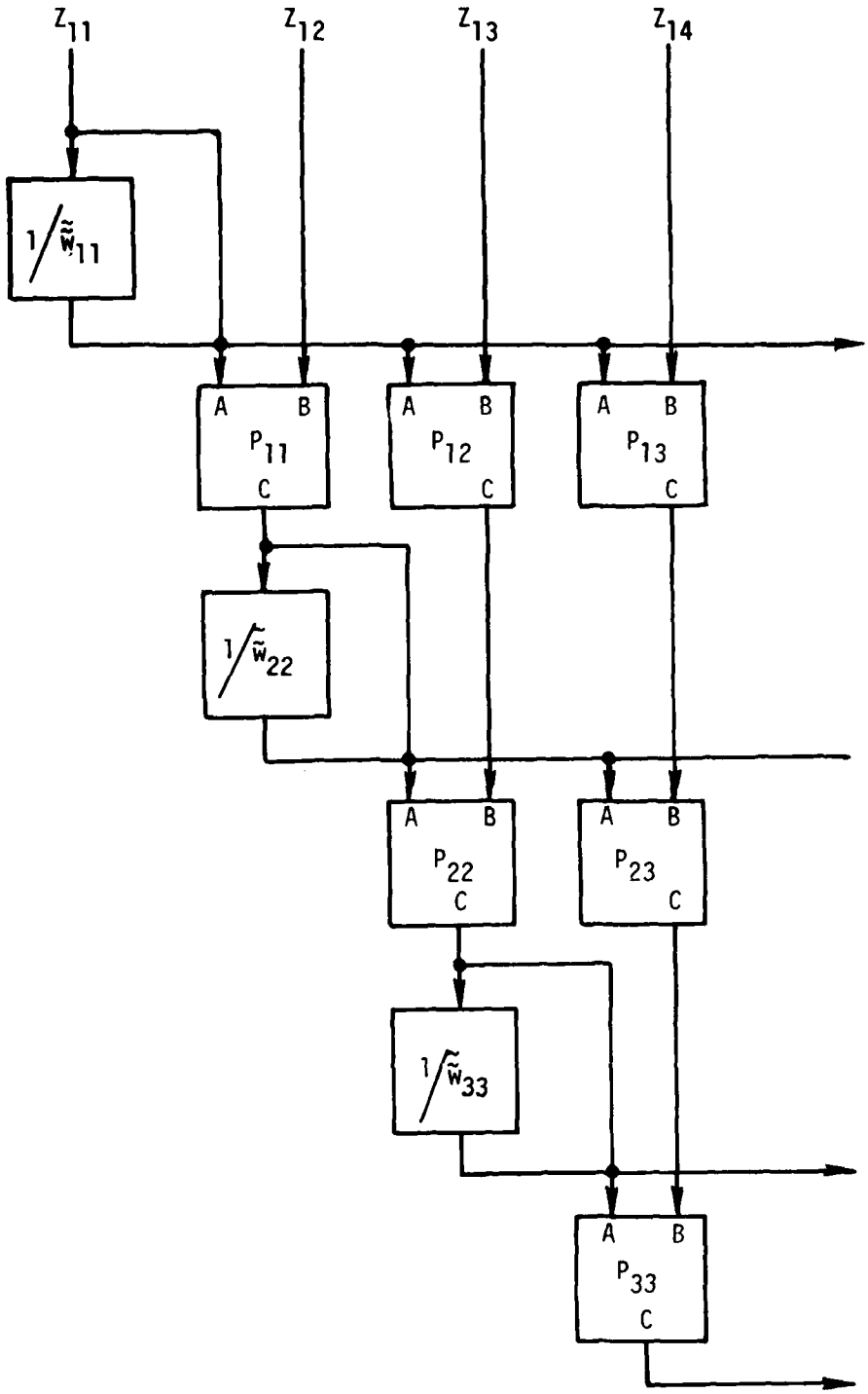


Figure 41. Processor for the $O(n^2)$ configuration.

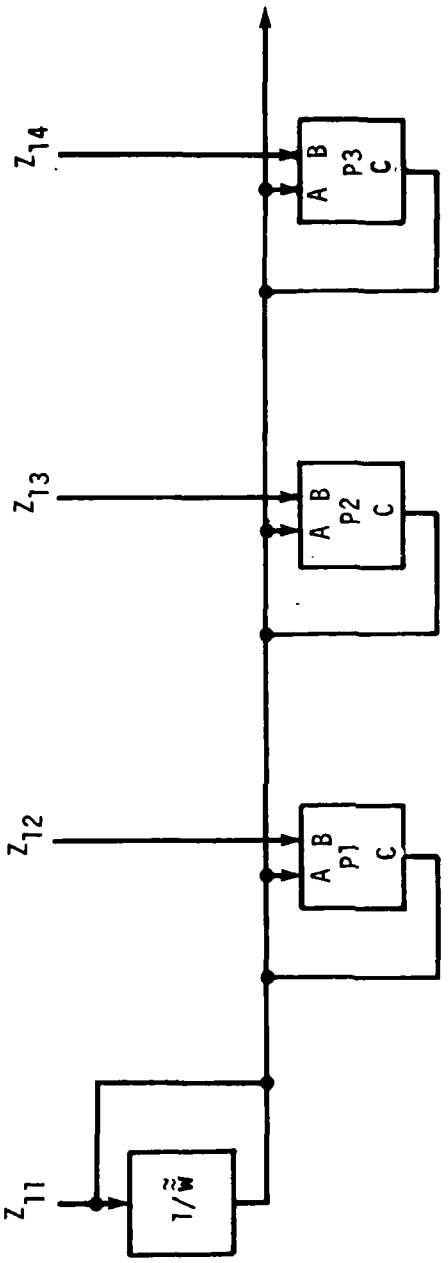


Figure 42. Processor for the O(n) configuration.

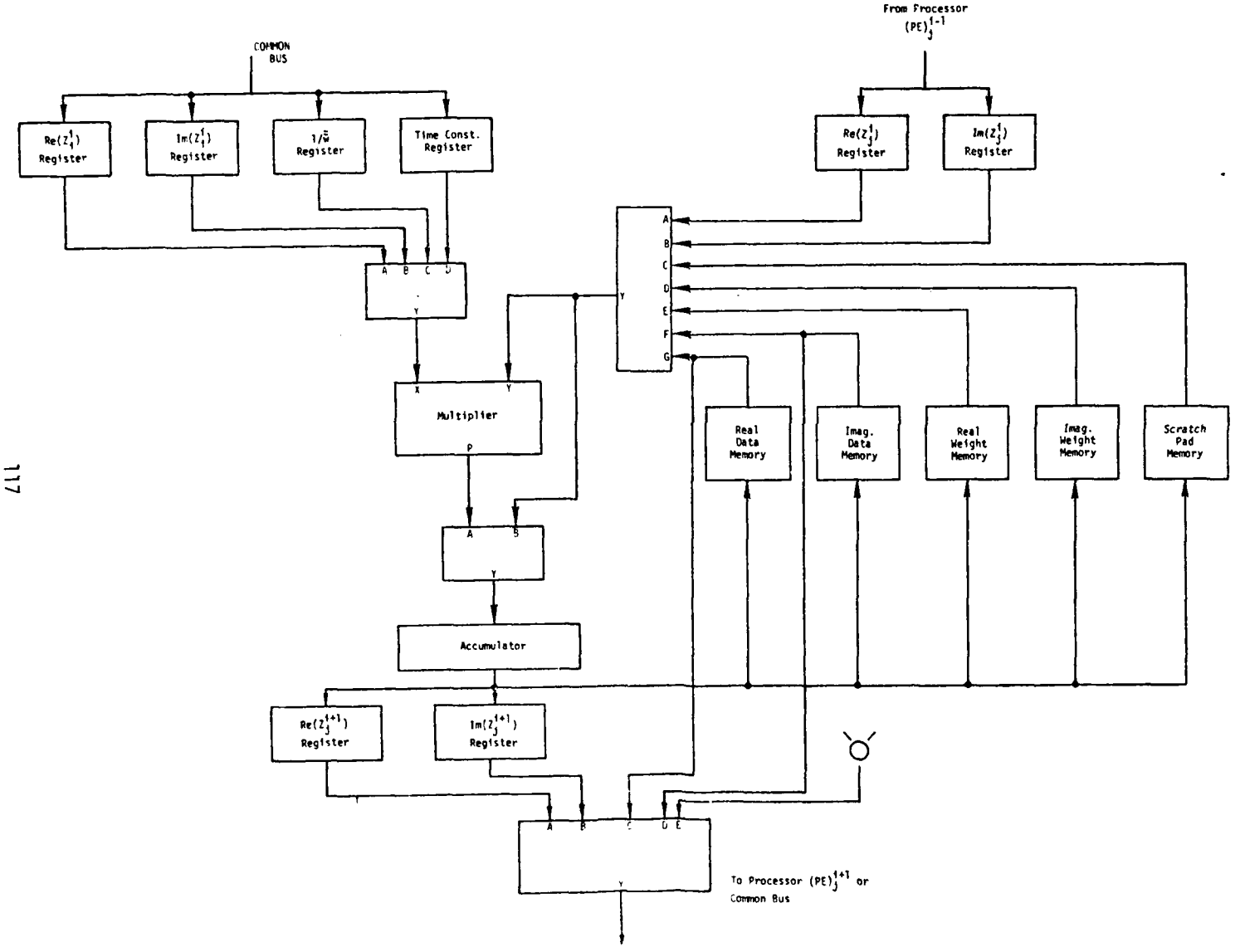


Figure 43. Block diagram for node processing element (NPE).

system. The inherent cost advantages of using Gram-Schmidt would be cancelled out.

The arithmetic operations performed within the NPE shown are serial in nature. The processing speed of the NPE could be increased by adding parallel computation abilities to the processing element. This parallelism could be expanded to the point where every equation is implemented in parallel hardware. Some middle ground between these two extremes, one that meets the processor speed requirements, will usually be chosen.

The block diagram for the diagonal processing element (DPE) that computes the common divisor is shown in Figure 44. This processor will operate in the same modes as the NPE and would not be practical to implement because of its generality. Examination of the block diagram reveals no functional block representing division. This block is unnecessary because of the following equation to calculate the inverse of a number:

$$C = 2 - AB_n$$

$$B_n = B_{n-1}C \quad ,$$

where

A is the value whose inverse is to be found,

B is the inverse, and

C is the correction factor to the inverse.

This equation is computed recursively, approximately doubling the accuracy of the inverse on each iteration. The first value of the inverse is

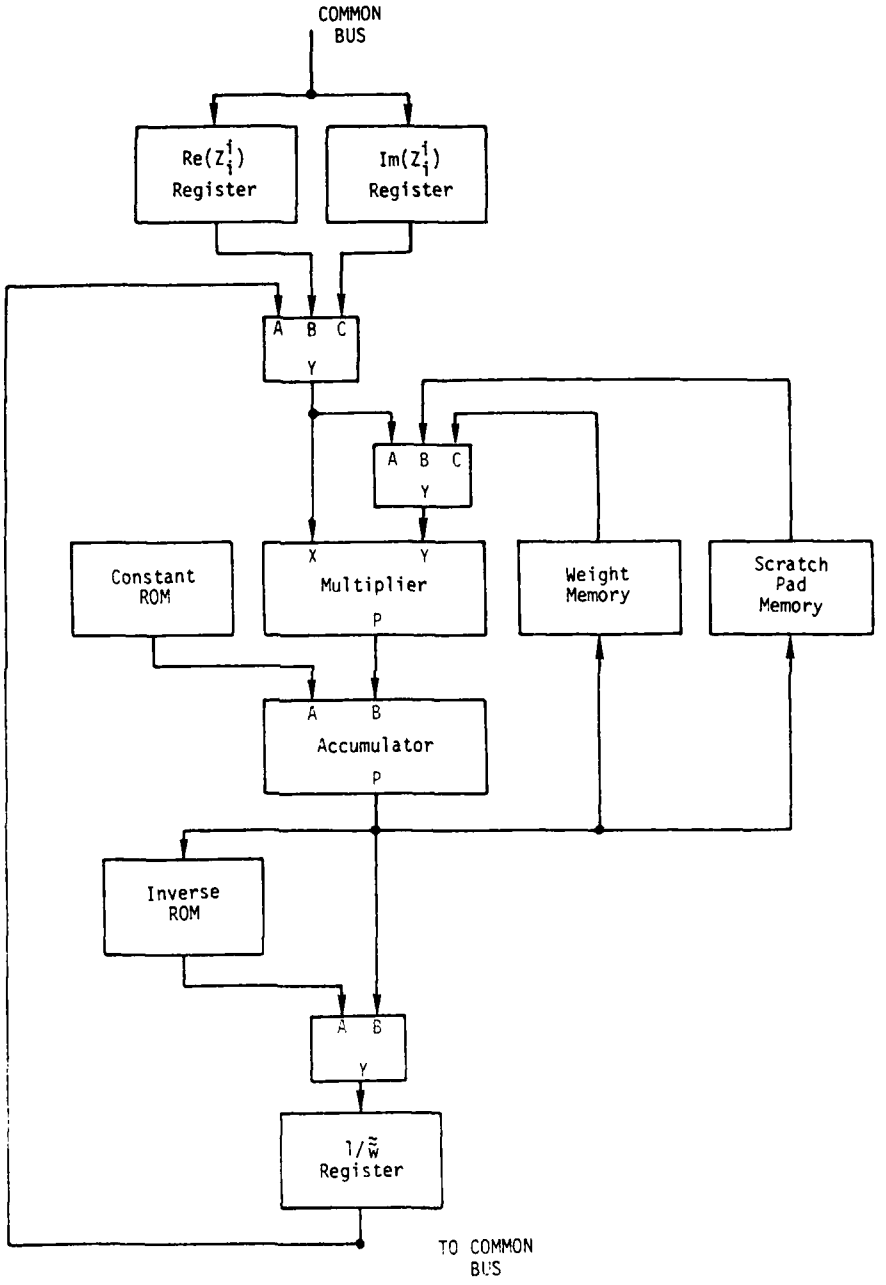


Figure 44. Block diagram for diagonal processing element (DPE).

a coarse value found in a lookup table. These two processing elements form the basis for the Gram-Schmidt processor. Their physical implementations depend on the system requirements.

4. CONCLUSIONS

We have studied in detail the Gram-Schmidt orthogonalization technique and modular architecture for its implementation. The analysis and simulations show that for blocked average Gram-Schmidt, the Gram-Schmidt system is identical in performance to sample covariance matrix techniques, in particular, Cholesky decomposition. This result shows that a universal adaptive algorithm does exist with a modular architecture since the Cholesky method is universal.

Also, because the architecture is tailored to the Gram-Schmidt algorithm, each processor performs simple operations of the form $z = x + Ay$. The complexity of each processor is low, so that high throughput paths can be achieved.

Figure 45 shows the timing curves for different Gram-Schmidt implementations. Figure 46 shows a banded region for Gram-Schmidt compared against the timing curves determined under a previous RADC contract [Liles et al. 1978].* As can be seen from these figures, systems capable of real-time operation are feasible.

Table 6 shows a summary of the conclusions of this study. We have already discussed the first conclusion, the existence of a good universal adaptive algorithm (UAA) which uses a minimum number of samples and has a modular architecture.

The importance of the second conclusion is that while a UAA may exist, it would have to be large, expensive, and power-hungry to satisfy the performance requirements of all possible radar systems. For example, a land-based system with 200 weights and a 20-mHz sampling rate, a stable, air-conditioned

*W. C. Liles, J. W. Demmel, J. D. Mallett, I. S. Reed, and L. E. Brennan, Multidomain Algorithm Evaluation, 2 volumes, TSC-PD-B525-1, Final Report on RADC Contract F30603-76-C-0319, Technology Service Corporation, Santa Monica, California, January 1978.

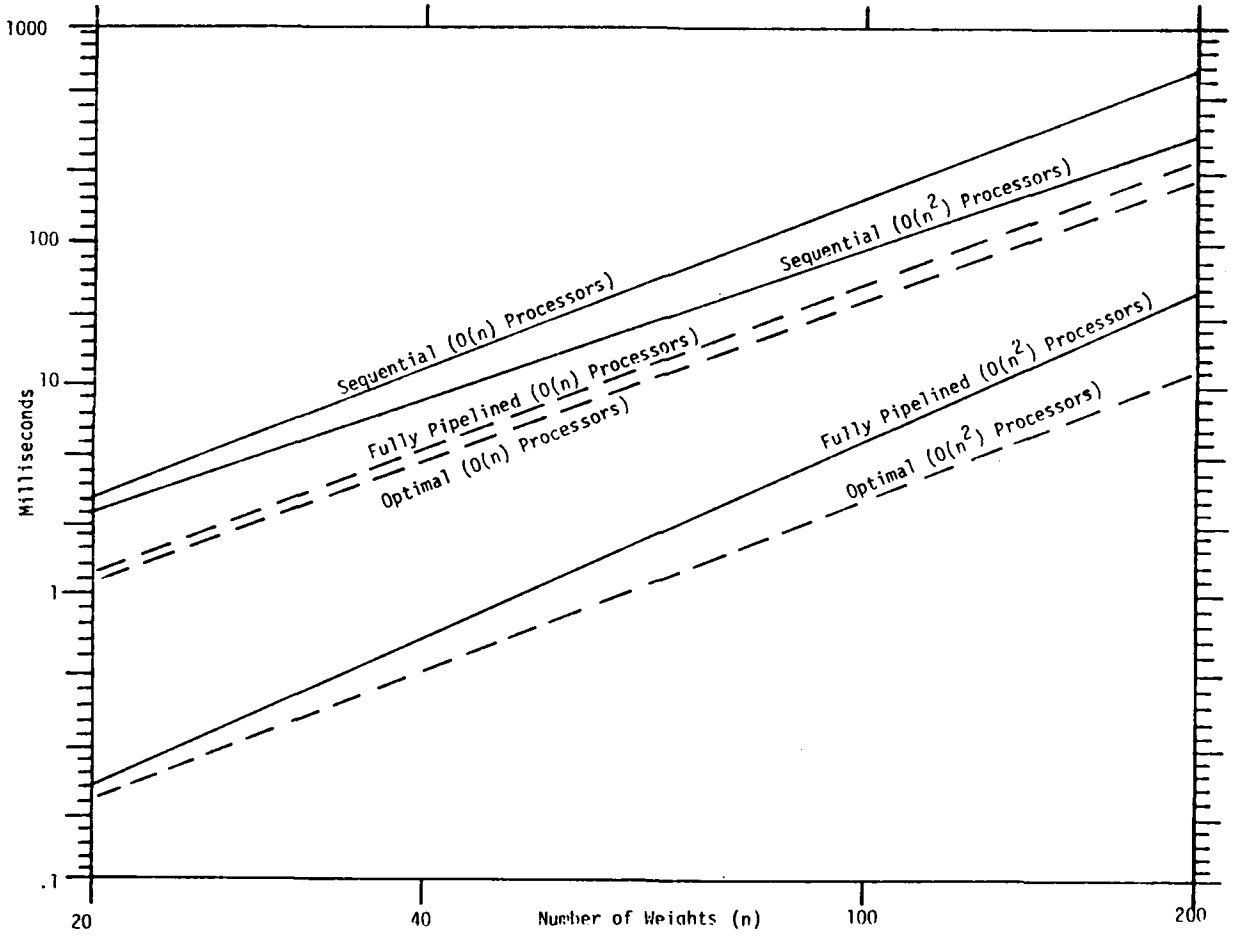


Figure 45. Timing curves for different Gram-Schmidt implementations. One steering vector; two times the number of weights equals number of samples.

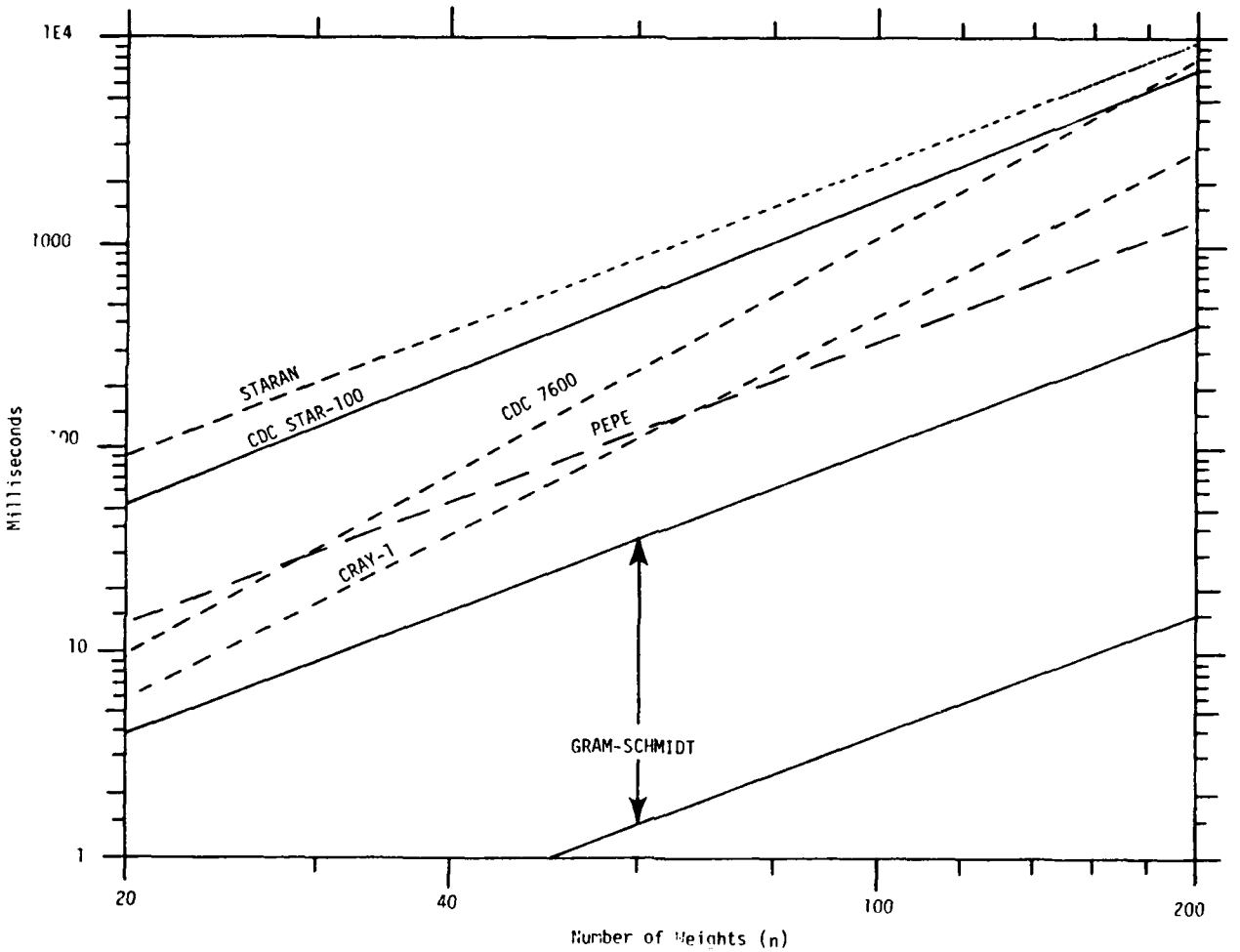


Figure 46. Timing curves for various processors, showing banded region for Gram-Schmidt.

TABLE 6. CONCLUSIONS

1. Universal adaptive algorithm with modular architecture exists.
2. Universal adaptive algorithm implementation for subset of all adaptive space exists.
3. Fast implementations are possible (Figure 45).
4. Processors are simple.
5. Processors can be improved as hardware advances are made.
6. The processor array can be designed to have good fault-tolerance properties.
7. Possible to get a priori probability bounds on growth of intermediate results.

environment with ample space puts far different requirements on a UAA than a 5-weight system small, cool, and light enough to carry on someone's back. But given a reasonably uniform subset of system requirements, a UAA does exist for that subset. A subset of interest is ground-based, permanent installations, 30 to 100 weights, high reliability, and adequate power supply. As advances in digital electronics occur, the size of subset areas increases until a universal implementation does exist.

One must remember that the algorithm, the processor interconnect structure, and the general processor design are all universal. The only thing which can change is particular implementations (number of bits, etc.) of the processor.

The third conclusion dealing with speed has been discussed above and is demonstrated by Figures 45 and 46.

The fourth conclusion deals with the complexity of a processor. Each processor is very simple as shown in Sections 2 and 3.5. Because of this simplicity and the advances of VLSI, shortly a processor on a chip will be possible. Since the processor functions are well specified, as new hardware becomes available, processors using the new components can be intermixed with other processors without system degradation. This intermixing is due to functional replacement on a processor level (conclusion 5).

Conclusion 6 deals with fault tolerance of the system. The Gram-Schmidt array is no more susceptible to noise caused by misaligned or broken converters, receivers, antennas, than any other technique. This was an unexpected, but welcome, conclusion, since Gram-Schmidt implementations have been shown to be prone to noise on digital computers. But due to our implementation of Gram-Schmidt and the nature of radar signals, the noise problem

is not a concern. Also, if any column of the array becomes bad, the system loses one degree of freedom and remains operative--a condition known as graceful degradation.

Conclusion 7 was the most difficult to arrive at, and more work should be performed to sharpen the results. Conclusion 7 states that a priori probability bounds on the internal Gram-Schmidt weights and interprocessor communication can be determined. These bounds are used to determine the number of bits used at different parts of the processing system, and what kind of arithmetic (fixed point or floating point) needs to be performed in order to avoid overflow/underflow and bound effects of roundoff error. In Section 3.3 we discuss in more detail the formulation of these bounds and actual number of bits.

5. RECOMMENDATIONS

Given the results of this study, we feel confident that the modular approach has benefits for the U.S. Air Force in the following areas: 1) logistics, 2) maintainability, 3) ease on new system design, and 4) low life cycle cost.

We believe that a Gram-Schmidt system of O(n) processors should be designed for high-speed updating of weights. The processor should be designed to be integrated into a 30- to 100-weight system. This system would demonstrate proof of concept and would serve as a test bed for further research. The processor designed for an O(n) configuration can also be used to study $O(n^2)$ array architectures; the inverse is not true. Thus, one processor can be used to study both possible configurations.

The processor should accumulate internally at least 24 bits (preferably 32) and transmit to other processors 16 bits. The internal memory for I & Q inputs should be at least 512 complex words.

REFERENCES

Berra, P. B. and A. K. Singhania, Timing Figures for Inverting Large Matrices Containing Complex Numbers Using the Staran Associative Processor, Rome Air Development Center, RADC-TR-76-339, Griffiss Air Force Base, New York, November 1976, A034266.

Blakely, C., "PEPE Application to BMD Systems," Proceedings of the 1977 International Conference on Parallel Processing, August 26-27, 1977, pp. 193-198.

Brennan, L. E., "Performance of the Sample Covariance Matrix Algorithm for Adaptive Arrays," unpublished manuscript, 19 July 1979.

Brennan, L. E., and I. S. Reed, "Theory of Adaptive Radar," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-9, No. 2, 1973, pp. 237-252.

Calahan, D. A., W. N. Joy, and D. A. Orbits, Preliminary Report on Results of Matrix Benchmarks on Vector Processors, Systems Engineering Laboratory, SEL Report No. 94, University of Michigan, Ann Arbor, May 24, 1976.

Chen, T. C., "Unconventional Superspeed Computer Systems," Spring Joint Computer Conference, 1971, pp. 365-366.

_____, "Parallelism, Pipelining, and Computer Efficiency," Computer Design, Vol. 10, 1971, pp.69-74.

Csanky, L., "Fast Parallel Matrix Inversion Algorithms," SIAM J. on Computing, Vol. 5, 1976, pp. 618-623.

Dahlquist, G., and A. Björk, Numerical Methods (trans. N. Anderson), Englewood Cliffs, New Jersey, Prentice-Hall, 1974.

Franklin, J. N., Matrix Theory, Englewood Cliffs, New Jersey, Prentice-Hall, 1968.

Gentleman, W. Morven, "Some Complexity Results for Matrix Computations on Parallel Processors," Journal of the Association for Computing Machinery, Vol. 25, No. 1, January 1978, pp. 112-115.

Isaacson, E., and H. B. Keller, Analysis of Numerical Methods, New York, John Wiley and Sons, Inc., 1966.

Kung, H. T. and C. E. Leiserson, Algorithms for VLSI Processor Arrays, Department of Computer Sciences, Carnegie-Mellon University, 1978.

Liles, W. C., and J. Demmel, "Solving Large Positive Definite Hermitian Linear Systems Utilizing Parallel/Pipeline Processors," Proceedings of the 1978 International Conference on Parallel Processing, IEEE, 1978, pp. 261-262.

Liles, W. C., J. W. Demmel, J. D. Mallett, I. S. Reed, and L. E. Brennan, Multidomain Algorithm Evaluation, 2 volumes, TSC-PD-B525-1, Final Report on RADC Contract F30603-76-C-0319, Technology Service Corporation, Santa Monica, California, January 1978, RADC-TR-78-59, Vol I A054357, Vol II A054358.

Rao, C. R., Linear Statistical Inference and its Applications, New York, John Wiley and Sons, Inc., 1965.

Reed, I. S., J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-10, No. 6, November 1974, pp. 853-863.

Rice, J. R., "Experiments on Gram-Schmidt Orthogonalization," Math. Comput., 20 April 1966, pp. 325-328.

Sameh, A. H., "Numerical Parallel Algorithms--A Survey," High Speed Computer and Algorithm Organization, Academic Press, 1977a, pp. 207-228.

Sameh, A. H., and D. J. Kuck, Linear Systems Solvers for Parallel Computers, Department of Computer Sciences, Report NO. UIUCDCS-R-75-701, University of Illinois at Urbana-Champaign, February 1975.

————, "Parallel Direct Linear Systems Solvers - A Survey," Parallel Computers - Parallel Mathematics, North Holland, 1977b, pp. 25-30.

————, "On Stable Parallel Linear System Solvers," Journal of the Association for Computing Machinery, Vol. 25, No. 1, January 1978, pp. 81-91.

Appendix A

PERFORMANCE OF THE SAMPLE COVARIANCE MATRIX ALGORITHM FOR ADAPTIVE ARRAYS

A.1 INTRODUCTION

When an adaptive array is implemented digitally, the sample covariance matrix algorithm provides a direct method of computing the adaptive weights and rapid convergence independent of the eigenvalues of the covariance matrix. Previous analyses of this algorithm[1] have assumed that the weights are computed using one set of array element outputs and these weights are applied to later array outputs. This report considers the case where the adaptive weights are tested against the same set of data used in the weight computation.

For many applications, the multiple channel sidelobe canceller is the preferred adaptive configuration. It can be shown that the sidelobe canceller is a special case of the more general adaptive array.[2] In the next section it is shown that the general adaptive array problem can be transformed to an equivalent sidelobe canceller problem, a form which is more convenient for some analyses. It is also shown that the array performance is independent of this transformation, and that the effective weights, output S/N, etc., can be computed in any convenient coordinate system provided the transformation of coordinates is non-singular.

[1] I. S. Reed, J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-10, No. 6, November 1974, pp. 853-863.

[2] S. P. Applebaum, "Adaptive Arrays," IEEE Trans. on Antennas and Propagation, Vol. AP-24, No. 5, September 1976, pp. 585-598.

A.2 COORDINATE TRANSFORMATIONS

Let X denote the column vector of array element outputs, $X_T = (x_1, x_2, \ldots, x_N)$, and S_x denote the corresponding signal vector. The noise covariance matrix of the array outputs is

$$M_x = E\, X\, X^{\dagger}, \tag{A-1}$$

where M_x is a NxN Hermitian matrix for an N element array, E denotes the expectation, $\dagger$ the complex transpose, and all noise components (but no signal) are included in M_x.

The weights which maximize the S/N ratio are

$$W_x = M_x^{-1}\, S_x \tag{A-2}$$

and the corresponding array output is

$$Z = W_x^{\dagger}\, X = S_x^{\dagger}\, M_x^{-1}\, X \tag{A-3}$$

With optimum weights, the output S/N ratio is

$$r = \frac{(W_x^{\dagger}\, S_x)^2}{W_x^{\dagger}\, M_x\, W_x} = S_x^{\dagger}\, M_x^{-1}\, S_x \tag{A-4}$$

Let T denote any non-singular transformation, and

$$Y = T\, X \tag{A-5}$$

In the new coordinate system,

$$M_y = E\ Y\ Y^\dagger = T\ M_x\ T^\dagger \tag{A-6}$$

$$S_y = T\ S_x \tag{A-7}$$

$$W_y = M_y^{-1}\ S_y \tag{A-8}$$

Combining Eqs. (A-5) through (A-8),

$$W_y = (T^\dagger)^{-1}\ W_x \tag{A-9}$$

and

$$Z = W_y^\dagger\ Y = W_x^\dagger\ X \tag{A-10}$$

i.e., the output of the array is unchanged by the transformation.

The sample covariance matrix in X coordinates is

$$\hat{M}_x = \frac{1}{K} \sum_{k=1}^{k} X_k\ X_k^\dagger \tag{A-11}$$

where X_k denotes the k^{th} independent sample of array element outputs and K is the number of samples in the estimator of M_x. The weights based on a sample covariance matrix are

$$\hat{W}_x = \hat{M}_x^{-1}\ S_x \tag{A-12}$$

Replacing M_x and M_y with the corresponding esimators, $\hat{M}_x$ and $\hat{M}_y$, and following the analysis of the preceding paragraph, it can easily be shown

the array output with weights based on a sample covariance matrix is also independent of the transformation T. Hence, the analysis of adaptive array performance, including the sample covariance matrix algorithm, can be performed in any convenient coordinate system provided the required transformation is non-singular.

A.3 PROBABILITY DISTRIBUTION OF S/N

For any arbitrary adaptive array, the input vector X can be transformed to a new set of coordinates in which the signal is present in only one component and the noise covariance matrix is diagonal. First, let $V = M^{-1/2} X$ to diagonalize the noise covariance matrix. Then,

$$M_V = E\, M_X^{-1/2} X\, X^{\dagger} M_X^{-1/2} = I$$

$$S_V = M_X^{-1/2} S_X \quad . \tag{A-13}$$

Next, rotate the coordinates by a unitary transformation U so that the S vector is non-zero only in the first component, and normalize its amplitude to unity

$$Y = (S_X^{\dagger} M_X^{-1} S_X)^{-1/2} U V$$

$$= r_o^{-1/2} U M_X^{-1/2} X \tag{A-14}$$

where r_o is the S/N ratio with optimum weights, given by (A-4). Then,

$$M_y = r_o^{-1} \, U \, M_x^{-1/2} \, M_x \, M_x^{-1/2} \, U^{\dagger} = \frac{1}{r_o} I \qquad \text{(A-15)}$$

and

$$S_y = r_o^{-1/2} \, U \, M_x^{-1/2} \, S_x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad . \qquad \text{(A-16)}$$

Note that the output S/N ratio in the new coordinate system is

$$S_y^{\dagger} \, M_y^{-1} \, S_y = (M_y)_{11}^{-1} \qquad \text{(A-17)}$$

i.e., the (1,1) element of the matrix M_y^{-1}.

The sample covariance matrix algorithm can be analyzed conveniently in the new coordinate system. The subscript y will be dropped in the following equations. Again, the sample covariance matrix is

$$\hat{M} = \frac{1}{K} \sum_{k=1}^{K} Y_k \, Y_k^{\dagger} \qquad \text{(A-18)}$$

The weights based on this estimator are

$$\hat{W} = \hat{M}^{-1} \, S \qquad \text{(A-19)}$$

When these weights are tested on a different set of samples than those used in estimating $\hat{W}$, the S/N ratio r_1 is

$$r_1 = \frac{|\hat{W}^\dagger S|^2}{\hat{W}^\dagger M \hat{W}} \tag{A-20}$$

The ratio of r_1 to the S/N with optimum weights is

$$\rho_1 = \frac{r_1}{r_o} = \frac{(S^\dagger \hat{M}^{-1} S)^2}{(S^\dagger M^{-1} S)(S^\dagger \hat{M}^{-1} M \hat{M}^{-1} S)} \tag{A-21}$$

The probability density of this variable ρ_1 was derived in [1].

In some cases of interest, the weights may be applied to the same set of samples used in computing $\hat{W}$. In this case, the output S/N ratio is

$$r = S^\dagger \hat{M}^{-1} S = (\hat{M}^{-1})_{11} \tag{A-22}$$

The analysis of [1] can be extended to obtain an expression for the probability density of r.

The sample covariance matrix has a complex Wishart distribution[1,3], i.e.,

$$P(A) = \frac{|A|^{K-N}}{I(M)} \exp\left\{-\mathrm{tr}(M^{-1} A)\right\} \tag{A-23}$$

[3] N. R. Goodman, "Statistical Analysis Based on a Certain Multivariate Complex, Gaussian Distribution," Ann. Math. Stat., Vol. 34, March 1963, pp. 152-177.

where $|A|$ denotes the determinant of A, N is the number of elements in the array, tr denotes the trace of the matrix, and $A = K\hat{M}$. The constant I(M) is a function of K, N, and the covariance matrix M. In Eq. (A-23), P(A) is the joint probability density of the elements of A, and is restricted to those matrices A which are positive definite. It assumes that the underlying noise process is complex Gaussian.

Consider the following representation of the matrix A.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{A-24}$$

where A_{11} is a scalar, A_{21} is a (N-1)x1 column vector equal to A_{12}^{+}, and A_{22} is a (N-1)x(N-1) matrix. As in [1], A can be factored as follows

$$A = \begin{pmatrix} 1 & A_{21}^{+} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} A_{11}-A_{12}\ A_{22}^{-1}\ A_{21}, & 0 \\ A_{22}^{-1}\ A_{21} & , I \end{pmatrix}, \tag{A-25}$$

and

$$|A| = |A_{11}-A_{12}\ A_{22}^{-1}\ A_{21}|\ |A_{22}| \tag{A-26}$$

Let

$$D_{11} = A_{11} - A_{12}\, A_{22}^{-1}\, A_{12}^{\dagger}$$

$$D_{12} = A_{12} = A_{21}^{\dagger} \qquad \text{(A-27)}$$

$$D_{22} = A_{22}$$

The Jacobian of the transformation from (A_{11}, A_{12}, A_{22}) to (D_{11}, D_{12}, D_{22}) is one, so

$$P(D_{11},D_{12},D_{22}) = D_{11}^{K-N}\, |D_{22}|^{K-N}\, \frac{1}{I(M)} \exp\left\{ -(D_{11}+D_{12}\, D_{22}^{-1}\, D_{12}^{\dagger} + \text{tr}\, D_{22})r_o \right\}$$

$$= P(D_{11})\, P(D_{12},D_{22}) \quad , \qquad \text{(A-28)}$$

where

$$P(D_{11}) = C_1\, D_{11}^{K-N} \exp\left\{-r_o D_{11}\right\} \qquad \text{(A-29)}$$

and C_1 is a constant.

Representing A^{-1} in the same form as Eq. (A-24),

$$A^{-1} = \begin{pmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{pmatrix} \quad , \qquad \text{(A-30)}$$

it can easily be shown that $A^{11} = \frac{1}{D_{11}}$. Since $A = K\hat{M}$ and, from Eq. (A-22)

the output S/N ratio is

$$r = (\hat{M}^{-1})_{11} = K A^{11} = \frac{K}{D_{11}} \tag{A-31}$$

Let $\rho = r/r_o = \frac{k}{r_o D_{11}}$. From Eq. (A-29)

$$P(\rho) = \frac{C_2 K^{K-N+1}}{\rho^{K-N+2}} \exp\left\{-\frac{K}{\rho}\right\}, \tag{A-32}$$

Normalizing this distribution, $C_2 = \frac{1}{(K-N)!}$, and

$$P(\rho) = \frac{K^{K-N+1}}{(K-N)!\ \rho^{K-N+2}} \exp\left[-\frac{K}{\rho}\right] \tag{A-33}$$

This is the probability density function for the normalized output S/N ratio, $\rho = r/r_o$, when the same samples are used in $\hat{M}$ for computing the weights and for testing the weights.

From Eq. (A-33),

$$\bar{\rho} = \frac{K}{K-N} \tag{A-34}$$

The mean output S/N ratio for the same samples is greater than with optimum weights, $W = M^{-1} S$. That is, $\bar{\rho} > 1$.

Appendix B

DERIVATION OF NOISE SENSITIVITY RESULTS

This appendix derives the results used in Section 3.2 We know [Brennan and Reed 1973] that the SNR of an adaptive system with true covariance matrix M, steering signal S, and weights W is

$$SNR_W = |S^*W|^2/W \cdot MW \quad . \tag{B-1}$$

When $W = W_{OPT} = M^{-1}S$, we get

$$S^*W = S^*(M^{-1}S) = S^*(M^{*-1}M)M^{-1}S = (M^{-1}S)^*M(M^{-1}S) = W^*MW \quad ;$$

so

$$SNR_{OPT} = |S^*W|^2/W^*MW = S^*W = W^*MW = S^*M^{-1}S$$

When $W = W_{OPT} + E$, where E is an error vector, we get

$$SNR_W = |S^*(W_{OPT} + E)|^2/(W_{OPT} + E)^*M(W_{OPT} + E)$$

$$= [|S^*W_{OPT}|^2 + 2Re(S^*W_{OPT})(\overline{S^*E}) + |S^*E|^2]$$

$$\div (W_{OPT}^*MW_{OPT} + 2ReW_{OPT}^*ME + E^*ME)$$

$$= (SNR_{OPT}^2 + 2SNR_{OPT}Re(S^*E) + |S^*E|^2)/(SNR_{OPT} + 2ReS^*E + E^*ME)$$

$$= SNR_{OPT} + (|S^*E|^2 - SNR_{OPT}E^*ME)/W^*MW$$

$$= [1 - (E^*ME/W^*MW)]SNR_{OPT} + (E^*ME/W^*MW)(|S^*E|^2/E^*ME)$$

$$= [1 - (E^*ME/W^*MW)]SNR_{OPT} + (E^*ME/W^*MW)SNR_E$$

$$= SNR_{OPT} + (E^*S^*SE - SNR_{OPT}E^*ME)/W^*MW$$

$$= SNR_{OPT} - E^*[M - (S^*S/SNR_{OPT})]E \cdot (SNR_{OPT}/W^*MW) \quad . \qquad (B\text{-}2)$$

Since M is positive definite Hermitian, S*S is positive semidefinite Hermitian of rank 1, and

$$x^*[M - (S^*S/SNR_{OPT})]x = x^*Mx/SNR_{OPT} \cdot [SNR_{OPT} - (x^*S^*Sx/x^*Mx)]$$

$$= (x^*Mx/SNR_{OPT})(SNR_{OPT} - SNR_x) \geq 0 \quad ,$$

we have $M - (S^*S/SNR_{OPT})$ is positive semidefinite Hermitian of rank n-1 ($W = M^{-1}S$ is in the null space) and

$$0 \leq \lambda[M - (S^*S/SNR_{OPT})] \leq \lambda_{max}(M) - \lambda_{min}S^*S = \lambda_{max}(M) \quad .$$

Also

$$W^*MW = (W_{OPT} + E)^*M(W_{OPT} + E) = W_{OPT}{}^*MW_{OPT} + 2ReW_{OPT}{}^*ME + E \cdot ME$$

$$\geq SNR_{OPT} - 2||S|| \cdot ||E|| + \lambda_{min} \cdot ||E||^2 \quad ;$$

so

$$SNR_{OPT}/W^*MW \leq 1 + (2||S|| \cdot ||E||/SNR_{OPT}) + O(||E||^2) = 1 + O(||E||).$$

When $\hat{W} = W + E$, W not necessarily optimal, we get

$$SNR_{\hat{W}} = [|S^*W|^2 + 2Re(S^*W)(\overline{S^*E}) + |S^*E|^2]/(W^*MW + 2ReW^*ME + E^*ME)$$

$$= \text{(assuming } |2ReW^*ME - E^*ME| < 1 \text{ for small E)}$$

$$= [|S^*W|^2 + 2Re(W^*SS^*E) + |S^*E|^2] \cdot (1/W^*MW)$$

$$\cdot [1 - (2ReW^*ME/W^*MW) - (E^*ME/W^*MW) + 4Re^2(W^*ME)/(W^*MW)^2 + O(||E||^3)]$$

$$= |S^*W|^2/W^*MW + (|S^*W|^2/W^*MW) \cdot (-2ReW^*ME/W^*MW) + 2ReW^*SS^*E/W^*MW$$

$$+ (|S^*W|^2/W^*MW)(-E^*ME/W^*MW) + (S^*W)^2/W^*MW \cdot [4Re^2W^*ME/(W^*MW)^2]$$

$$- [4Re(W^*SS^*E)Re(W^*ME)/(W^*MW)^2] + |S^*E|^2/W^*MW + O(||E||^3)$$

$$= SNR_W + SNR_W(-2/W^*MW) \cdot Re[W^*(M - SS^*/SNR_W)E]$$

$$+ (SNR_W/W^*MW) \cdot \{(E^*SS^*E)(1/SNR_W) - E^*ME$$

$$+ 4ReW^*ME/(W^*MW) \cdot [ReW^*ME - Re(W^*SS^*E)/SNR_W]\} + O(||E||^3)$$

$$= SNR_W - 2(SNR_W/W^*MW) \cdot Re\{[(M - SS^*/SNR_W)W]^*E\}$$

$$+ SNR_W/W^*MW \quad Re\{[4(ReW^*ME/W^*MW)W^* - E^*](M - SS^*/SNR_W)E\}$$

$$+ O(||E||^3) \quad . \qquad \text{(B-3)}$$

Now we turn to the problem of inverting $\hat{M} = M + A$, where M is a covariance matrix and $A_{k\ell} = \{\rho$ if $k=\ell=j$, 0 otherwise$\}$. We may write

$$\hat{M}^{-1} = (M + A)^{-1} = M^{-1}M(M + A)^{-1} = M^{-1}[(M + A)M^{-1}]^{-1}$$

$$= M^{-1}(I + AM^{-1})^{-1} \quad . \tag{B-4}$$

Since AM^{-1} is all zeroes except for row j, which is the j^{th} row of M^{-1} multiplied by ρ, we can write

$$\hat{M}^{-1} = M^{-1}\left(I + \rho\begin{bmatrix} 0 \\ m^{j1} \ . \ . \ . \ m^{jn} \\ 0 \end{bmatrix} \leftarrow j^{th} \text{ row}\right)^{-1} \tag{B-5}$$

where $M = \{m_{ij}\}$ and $M^{-1} = \{m^{ij}\}$. The inverse of the expression in parentheses is given simply by

$$\left(I + \rho\begin{bmatrix} 0 \\ m^{j1} \ . \ . \ . \ m^{jn} \\ 0 \end{bmatrix}\right)^{-1} = I - \frac{\rho}{1 + \rho m^{jj}} \cdot \begin{bmatrix} 0 \\ m^{j1} \ . \ . \ . \ m^{jn} \\ 0 \end{bmatrix} \tag{B-6}$$

so that $\hat{M}^{-1}$ is

$$\hat{M}^{-1} = M^{-1} + M^{-1}\left(\frac{-\rho}{1 + \rho m^{jj}}\right)\begin{bmatrix} 0 \\ m^{j1} \ . \ . \ . \ m^{jn} \\ 0 \end{bmatrix} \quad . \tag{B-7}$$

This formula is a special case of the Sherman-Morrison formula [Dahlquist and Björck 1974, p. 161].*

The new weights $\hat{W} = \hat{M}^{-1}S$ are given by

$$\hat{W} = W + M^{-1}\left(\frac{-\rho}{1 + \rho m^{jj}}\right)\begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} \leftarrow \text{row } j \tag{B-8}$$

*G. Dahlquist and A. Björk, Numerical Methods (trans. N. Anderson), Englewood Cliffs, New Jersey, Prentice Hall, 1974.

where $W = \{w_j\} = M^{-1}S$ are the original weights. Thus the new weights equal the old weights plus the j^{th} column of M^{-1} times $[-\rho/(1 + \rho m^{jj})]$. We can express $S^*\hat{M}^{-1}S$, the optimal SNR of the system with noise, in terms of the old optimal SNR = $S^*M^{-1}S$, using Eq. (B-8):

$$SNR_{NEWOPT} = S^*\hat{M}^{-1}S = S^*\hat{W} = S^*W + S^*M^{-1} \cdot [-\rho/(1 + \rho m^{jj})] \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix}$$

$$= SNR_{OLDOPT} + W^* \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} [-\rho/(1 + \rho m^{jj})]$$

$$= SNR_{OLDOPT} - [\rho |w_j|^2/(1 + \rho m^{jj})] \quad . \tag{B-9}$$

If the covariance matrix is still M, but weights $\hat{W} = \hat{M}^{-1}S$ are used, we may compute the new SNR using $E = \hat{W} - W$ and Eq. (B-2). We need to know

$$E^*ME = \left[M^{-1} \left(\frac{-\rho}{1 + \rho m^{jj}} \right) \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} \right]^* M \left[M^{-1} \left(\frac{-\rho}{1 + \rho m^{jj}} \right) \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} \right]$$

$$= \frac{-\rho^2}{(1 + \rho m^{jj})^2} \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix}^* M^{-1} \begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} = \frac{\rho^2 |w_j|^2 m^{jj}}{(1 + \rho m^{jj})^2} \tag{B-10}$$

and

$$S^*E = W^*ME = W^*\left(\frac{-\rho}{1 + \rho m^{jj}}\right)\begin{bmatrix} 0 \\ w_j \\ 0 \end{bmatrix} = \frac{-\rho|w_j|^2}{1 + \rho m^{jj}} \qquad \text{(B-11)}$$

and

$$W^*MW = SNR_{OPT} + 2Re W^*_{OPT}ME + E^*ME$$

$$= SNR_{OPT} + 2\rho|w_j|^2/(1 + \rho m^{jj}) + \frac{\rho^2|w_j|^2 m^{jj}}{(1 + \rho m^{jj})^2} \quad . \qquad \text{(B-12)}$$

Then the SNR_W is

$$SNR_W = SNR_{OPT} - \frac{SNR_{OPT}}{SNR_{OPT} - 2\rho|w_j|^2/(1 + \rho m^{jj}) + \rho^2|w_j|^2 m^{jj}/(1 + \rho m^{jj})^2}$$

$$\cdot [\rho^2 |w_j|^2 m^{jj}/(1 + \rho m^{jj})^2 - \rho^2|w_j|^4/(1 + \rho m^{jj})^2/SNR_{OPT}]$$

$$= SNR_{OPT} - \frac{\rho^2|w_j|^2 m^{jj}/(1 + \rho m^{jj})^2}{SNR_{OPT} - \rho|w_j|^2/(1 + \rho m^{jj})[2 - \rho m_{jj}/(1 + \rho m_{jj})]}$$

$$\cdot \left[SNR_{OPT} - \frac{|w_j|^2}{m^{jj}}\right]$$

$$= SNR_{OPT} - \rho^2|w_j|^2 m^{jj}/[SNR_{OPT}(1 + \rho m^{jj})^2 - \rho^2|w_j|^2(2 + \rho m^{jj})]$$

$$\cdot \left[SNR_{OPT} - \frac{|w_j|^2}{m^{jj}}\right] \quad . \qquad \text{(B-13)}$$

By differentiating Eq. (B-13) with respect to ρ, we can show SNR_W is a monotonic decreasing function of ρ, and that $\lim_{\rho\to\infty} SNR_W = SNR_{OPT} - |w_j|^2/m^{jj}$, the same lower bound as for Eq. (B-9).

Appendix C

EXAMPLE OF USING GRAM-SCHMIDT TO SOLVE A SYSTEM OF SIMULTANEOUS EQUATIONS

Assume the matrix equation

$$Ax = b \quad ,$$

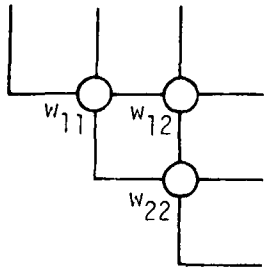
where A is real positive definite Hermitian and

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 4 & 12 & 20 \\ 6 & 20 & 36 \end{bmatrix}$$

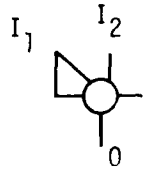
$$x = \text{unknown} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$b = \begin{bmatrix} 4 \\ 16 \\ 30 \end{bmatrix}$$

The network used is shown below.



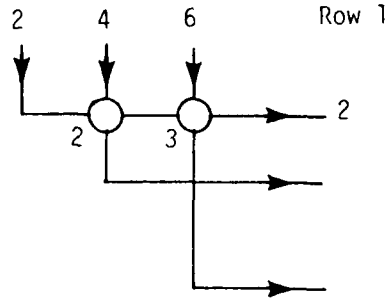
Here,



$$0 = I_2 - WI_1 \quad .$$

We will solve for x by decomposing A into the form LDL^T. This decomposition is performed by passing each row of the matrix through the network, using the following steps:

Step 1. Pass row 1 of the matrix through the array to calculate the first row of weights.

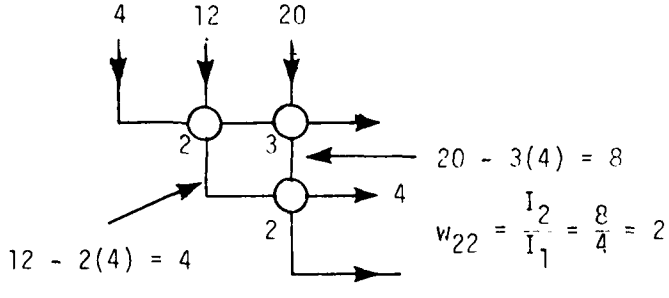


$$w_{11} = a_{12}/a_{11} = 4/2 = 2$$

$$w_{12} = a_{13}/a_{11} = 6/2 = 2$$

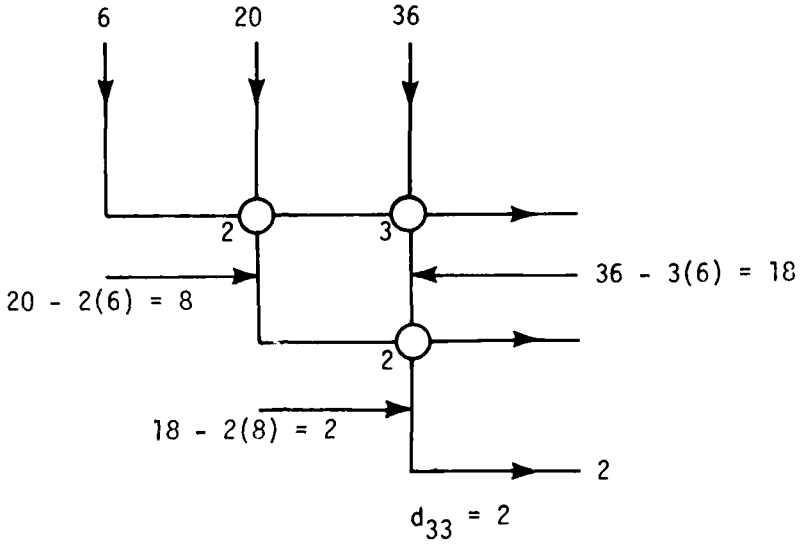
$$d_{11} = a_{11} = 2$$

Step 2. Pass row 2 of matrix through the array to calculate second row of weights.



$$d_{22} = 4$$

Step 3. Pass row 3 of the matrix through the array to calculate d_{33}.



We now know that the L and D matrices are as follows:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

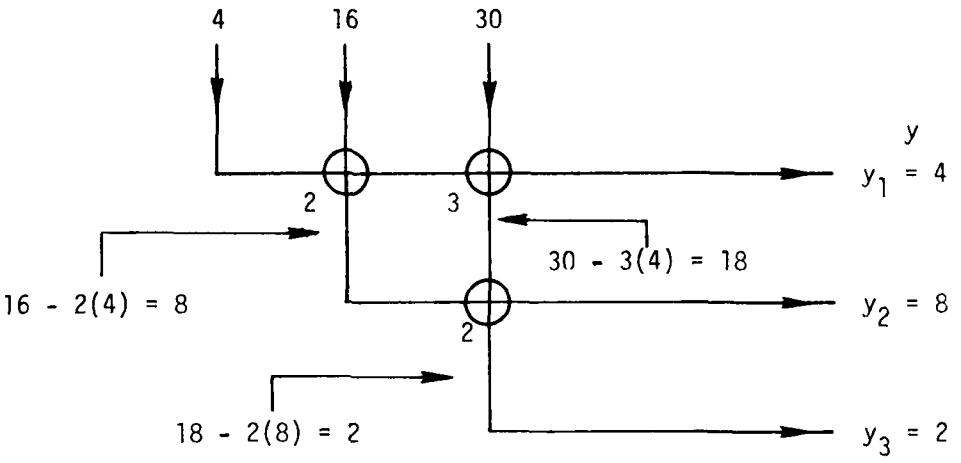
To solve the system of equations, we have to perform the following steps as well, because $LDL^Tx = b \rightarrow x = L^{T-1}[D^{-1}(L^{-1}b)]$:

Step 4. $y = L^{-1}b$.

Step 5. $z = D^{-1}y$.

Step 6. $x = L^{T-1}z$.

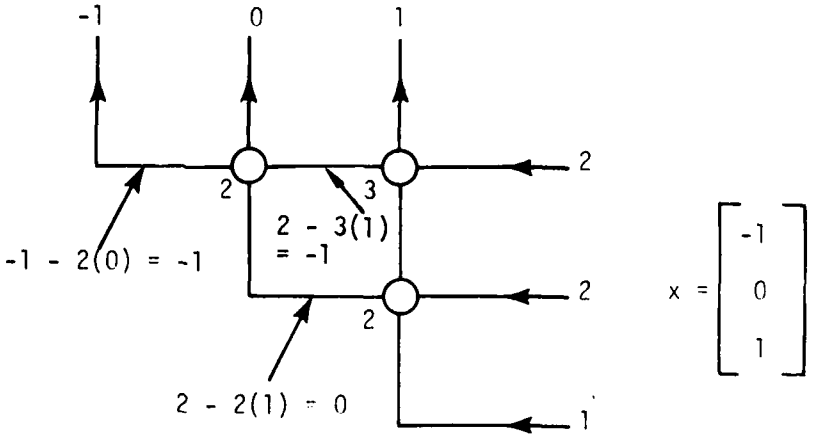
Step 4. Pass b through the array to perform $L^{-1}b$



Step 5. $z = D^{-1}y$; and since D is diagonal, it is easy to invert.

$$D^{-1} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \qquad y = \begin{bmatrix} 4 \\ 8 \\ 2 \end{bmatrix} \qquad z = D^{-1}y = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

Step 6. $x = L^{T-1}z$, which will be done by the reverse flow method.
We pass z through the network in opposite direction from normal.



The vector x has now been determined.

We now present a solution to the problem being solved which differs from the previous solution because it eliminates steps 4 and 5 by using an augmented from of the matrix (to be explained below).

Solve $Ax = b$

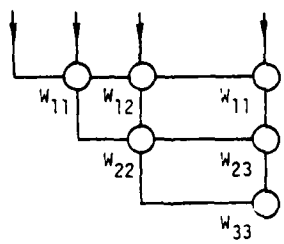
$$A = \begin{bmatrix} 2 & 4 & 6 \\ 4 & 12 & 20 \\ 6 & 20 & 36 \end{bmatrix}$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

$$b = \begin{bmatrix} 4 \\ 16 \\ 30 \end{bmatrix}$$

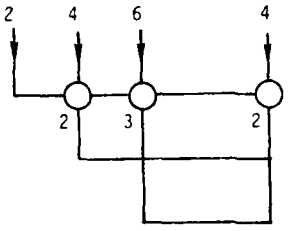
$$\text{A augmented} = \begin{bmatrix} 2 & 4 & 6 & 4 \\ 4 & 12 & 20 & 16 \\ 6 & 20 & 36 & 30 \end{bmatrix}$$

A augmented is simply the A matrix with b added as the last column. The network also has an added column as shown below.



The steps of the augmented form of solution are as follows:

Step 1. Pass row 1 of matrix through array to calculate first row of weights.

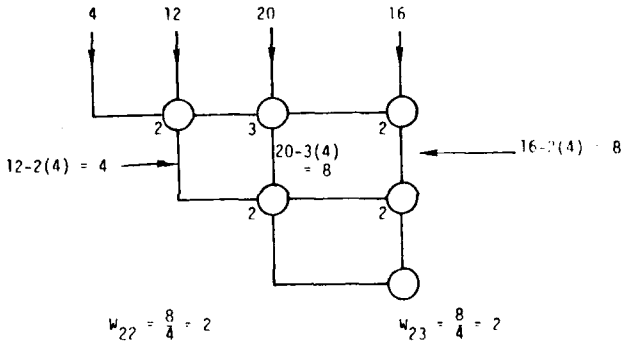


$$W_{11} = \frac{a_{12}}{a_{11}} = \frac{4}{2} = 2$$

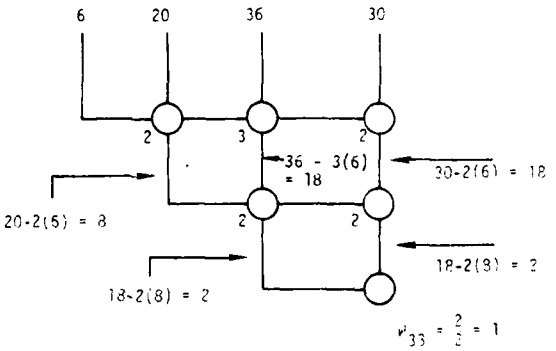
$$W_{12} = \frac{a_{13}}{a_{11}} = \frac{6}{2} = 3$$

$$W_{13} = \frac{b_1}{a_{11}} = \frac{4}{2} = 2$$

Step 2. Pass row 2 of matrix through array to calculate second row of weights.



Step 3. Pass row 3 of matrix through array to calculate third row of weights.



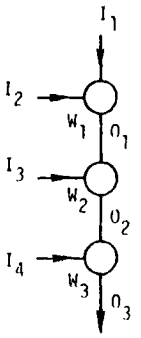
Note that

$$W_{13} = 2$$
$$W_{23} = 2$$
$$W_{33} = 1$$

These values are the same as z in step 5 of the previous solution. We can proceed with the reverse flow or unit vector method to determine x.

Step 4. We now show one step of the unit vector method as discussed in Section 2.6.1. The unit vector method passes a unit vector through the array and performs the dot product of the array output with the vector z to determine an element of x.

A column of the array performs a calculation very similar to dot product and can be used in that manner, as shown below:



$$O_1 = I_1 - W_1 I_2$$

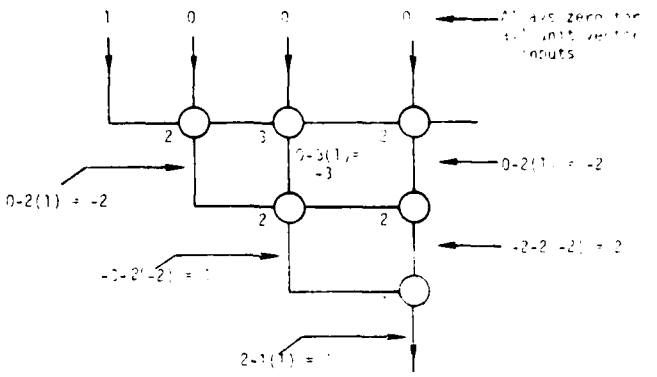
$$O_2 = O_1 - W_2 I_3$$

$$O_3 = O_2 - W_3 I_4$$

$$O_3 = I_1 - W_1 I_2 - W_2 I_3 - W_3 I_4$$

$$= I_1 - \sum_{i=1}^{3} W_i I_{i+1}$$

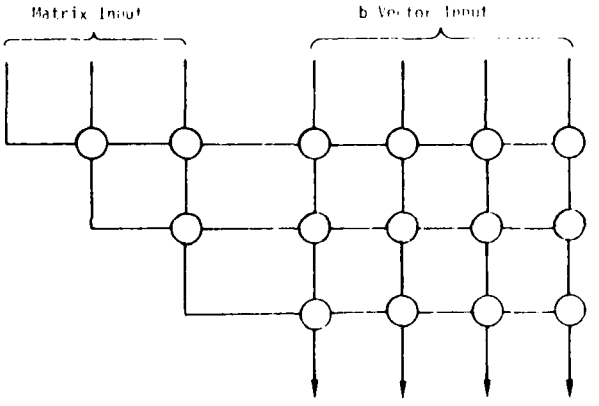
By setting I_1=0, the column output is the negative of the dot product of the column weights and the input vector. We now perform the unit vector method to calculate x_1.



The array output is 1; therefore, x_1=-1 because the last column output is the negative of the dot product. x_1=-1 is the same answer obtained in the previous solution.

Multiple b's (Multiple Steering Vectors)

This modification consists of adding enough extra columns to handle the additional b vectors. The illustration shows the configuration for 4b vectors, and a matrix of order 3.



Let N be the matrix dimension. Then N additional processors are needed for each b vector. For Kb vectors, K*N additional processors are needed.

To determine x for Ax=b, either reverse flow or unit vector methods can be used. If the unit vector method is used, the total time is independent of the number of b vectors. The total time would be derived as follows:

N = dimension of matrix

t = time per array stage

There are N unit vectors and N stages; thus, total time is

$$T_{UV} = Nt + (N-1)t = (2N-1)t$$

For the reverse flow method, the time is a function of the number of b vectors (K). Using the same definitions as above, the total time

$$T_{RF} = (N-1)t + (K-1)t$$
$$= (N+K-2)t \quad .$$

The reverse flow method is faster for K<N+1 but also requires more complicated processors.

Appendix D

RECIPROCAL AND SQUARE-ROOT CALCULATION

Depending on which Gram-Schmidt method is used (LDL* or LL*), either the reciprocal or the reciprocal of the square root or a real number must be calculated at each level. These calculations are performed using Newton-Raphson iteration. Newton-Raphson has the advantage of quadratic conversion (the number of accurate bits doubles with each iteration step) and the processing can be performed without divisions. The general recursive relationship is $x_{i+1} = x_i - F(x_i)/F'(x_i)$. For reciprocal of A the function $F(x) = (1/x - A)$. Applying the reciprocal function to the general function we have the following:

$$x_{i+1} = x_i - \frac{F(x)}{F'(x)}$$

$$x_{i+1} = x_i - \frac{\frac{1}{x_i} - A}{-x_i^{-2}}$$

$$= x_i + x_i - Ax_i^2$$

$$= (2-Ax_i)x_i$$

The seed value for x_o is determined by table lookup and is an approximation of 1/A. Each iteration step doubles the accurate bits.

The square root operation is performed almost identically. The function $F = (1/x^2 - A)$, so the recursive formula becomes

$$x_{i+1} = (3 - x_i^2 A)x_i/2$$

The divide by 2 is easy to implement, using a shift operation. The reciprocal of the square-root operation is slower than the reciprocal operation due to an extra multiplication and shift.

Appendix E

TIMING EQUATIONS FOR BLOCK AVERAGE AND O(n) PROCESSORS

The processor computation abilities are those listed in Section 3.4. The O(n) processors are connected and perform the processing as shown in Figure E-1. The reason for the processor on the first input is to compute the first denominator and its reciprocal.

The timing equations developed in Section 2.6 for an $O(n^2)$ array apply to the O(n) array when one equation is to be solved. When more equations are to be solved (in a pipeline by the $O(n^2)$ array) the differences in time between the O(n) and $O(n^2)$ processor implementations become apparent. The three stages of calculation using the O(n) array are:

1. Calculate all internal weights.
2. Pass S through the array.
3. Solve for the weight W.

We build up the timing equation in the above order for three implementations:

Sequential

Full pipeline

Optimal

E.1 CALCULATE ALL INTERNAL WEIGHTS

For each implementation method, we have the following steps:

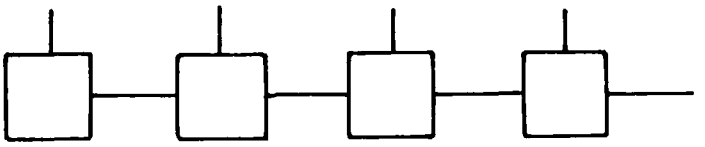
A. Compute numerator and denominator.

B. Compute reciprocal.

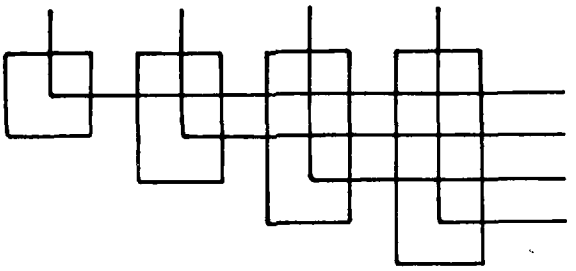
C. Multiply by reciprocal.

D. Pass data through array, applying internal weights.

The timings for these basic steps are:



a. Processor configuration.



b. Processing configuration.

Figure E-1. Model used for O(n) timing measurements.

	A	B	C	D
Sequential	10 St	7t	2t	11 St
Full pipeline	10t+(S-1)t	7t	2t	11t+(S-1)t
Optimal	4t+(S-1)t	7t	2t	5t+(S-1)t

We must calculate internal weights at N-1 levels; so the general timing equation is

$$T = (N-1)A + (N-1)B + (N-1)C + (N-2)D$$

The specific timing equations are:

	Arbitrary N,S	S = 2N
Sequential	(21NS+9N-32S-9)t	$(42N^2-55N-9)t$
Full Pipeline	(25N+28N-3S-38)t	$(4N^2+22N-38)t$
Optimal	(2SN+16N-3S-20)t	$(4N^2+10N-20)t$

The only difference between the times for O(n) processors and $O(n^2)$ processors is that with $O(n^2)$ processors, there is overlap between applying the internal weights calculating the numerator. This overlap can be obtained with O(n) processors if the processing power of each processor is doubled. Another approach is to use twice the number of processors as shown in Figure E-2. This is still an O(n) processor configuration and the processors can be identical.

E.2 PASS S THROUGH THE ARRAY

With only O(n) processors, no pipelining can be performed. By designing the processors with recirculating pipelines, we can handle L steering vectors at a time, where L is the length of the pipeline. For K steering vectors, $\lceil K/L \rceil$ passes are required. Another method is to pass all K vectors

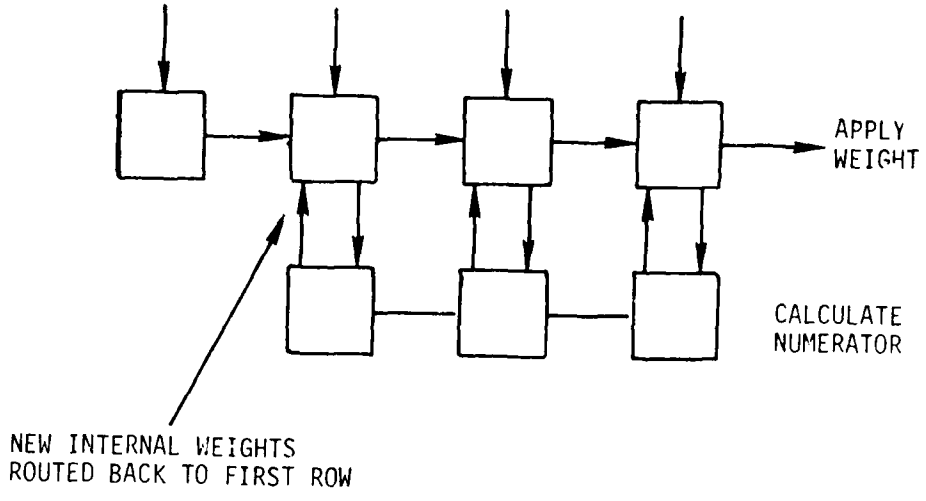


Figure E-2. O(n) processor configuration with same speed to calculate internal weights as $O(n^2)$ processors using block averaging.

through sequentially. In this case, a memory (first-in, first-out (FIFO) buffer) of length K-L is needed for recirculating (Figure E-3). If K-L $\geq$ L, then additional processing may be disired on the feedback path instead of simply a FIFO-buffer; this is the model we will use. This model applies only to full pipeline and optimal implementations. The sequential implementation does not offer any pipeline possibilities. The timing formulas are:

	Arbitrary N,K	K = 1
Sequential	(N-1)11Kt	11(N-1)t
Full Pipeline	(N-1)max(K,11)t + (K-1)t	11(N-1)t
Optimal	(N-1)max(K,5)t + (K-1)t	5(N-1)t

The maximum function is due to the recirculating buffer. If K is less than the length of the pipeline, then we use the pipeline length, or else we must include the recirculating path.

E.3 SOLVE FOR WEIGHTS

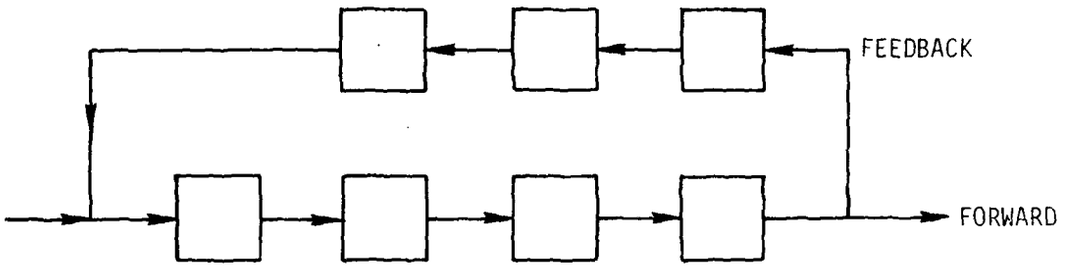
We solve for both unit vector and reverse flow methods.

Unit Vector Timings

For the unit vector method, we must pass through K+NK vectors. We can use the timing formulas just developed for passing K steering vectors by substituting K+NK for K. These formulas are:

	Arbitrary N, K+NK	N $\geq$ 10
Sequential	$(N^2-1)11Kt$	$(N^2-1)11Kt$
Full Pipeline	(N-1)max(K+NK,11)t + (K+NK-1)t	$(N^2K+NK-1)t$
Optimal	(N-1)max(K+NK,5)t + (K+NK-1)t	$(N^2K+NK-1)t$

Since we are interested in large cases, the overall timing formulas now given are with N $\geq$ 10; so a recirculating pipeline is used.



For K Input Vectors
L Forward Pipeline Length
Require
K-L Feedback Stages
Effective Pipeline Length =
Max(K,L)

Figure E-3. Recirculating pipelines.

Timing Formulas for Complete Process

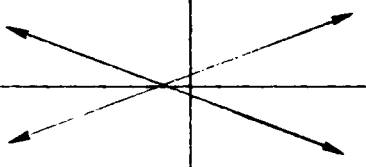
	Arbitrary S,K; N > 10	N > 10, S=2N,K=1
Sequential	$(21SN+11N^2K+9N-11K-32S-9)t$	$(53N^2-55N-20)t$
Full Pipeline	$(2SN+N^2K+NK+28N-3S-39)t$	$(5N^2+23N-39)t$
Optimal	$(2SN+N^2K+NK+16N-3S-21)t$	$(8N^2+11N-21)t$

For the special case (N > 10, S=2N,K=1) the above are approximately two times slower than the $O(n^2)$ implementations. As K increases, this difference becomes larger due to this N^2K term in the above formulas.

Reverse Flow Timings

The reverse flow timings, in general, are not simply twice the forward flow for K steering vectors as is the case with $O(n^2)$ processors. The exception is when diagonal collapsing is used because then the array is symmetric from both the input and output ports. The reverse flow timing for horizontal comparison is the same as forward flow from K steering vectors for vertical comparison. For vertical comparison just interchange horizontal and vertical in the above sentence. This is summarized in Table E-1. We will only derive the timing for vertical collapsing.

TABLE E-1. TIMINGS FOR THREE O(n) IMPLEMENTATIONS

Implementation	Pass K Steering Vectors Through Array	Reverse Flow K Steering Vectors
Vertical Collapsing		
Horizontal Collapsing		
Diagonal Collapsing		

NOTE: Times at end of double header arrows are the same.

The last processor must operate on (N-1) streams before the other processors can compute their tasks. After the last finished there are (N-2) processors waiting for the output, which must be processed in order. A question which arises is whether to process one steering vector at a time or the i^{th} element from all steering vectors before the $i-1^{st}$ element. Both of these methods have the same throughput rate.

The timing equations for reverse flow only are:

	Arbitrary N,K
Sequential	(KN-K+N-2)11t
Full Pipeline	(KN-K+11N-12)t
Optimal	(KN-K+5N-6)t

The computer timing equations for all three steps are:

	Arbitrary N,K,S	S=2N,K=1
Sequential	(21SN+22NK+20N-22K-32S-31)t	$(42N^2-22N-53)t$
Full Pipeline	[25N+NK+39N-3S-51+(N-1)max(K,11)]t	$(4N^2+45N-62)t$
Optimal	[25N+NK+21N-3S-27+(N-1)max(K,5)]t	$(4N^2+21N-32)t$

Because of the symmetries between horizontal and vertical collapsing, the above formulas apply to both.

Comparison of Unit Vector and Reverse Flow Times

The reverse flow method is faster if the following conditions hold:

	N > 10
Sequential	All cases
Full Pipeline	$N^2K+12>11N + (N-1)max(K,11)$
Optimal	All cases

If we add the restriction that N must be greater than 22, reverse flow is always superior. This restriction is not unreasonable, since our effort is directed at large adaptive arrays.

Special Case

As already mentioned an O(n) system can determine the internal weights as quickly as an $O(n^2)$ system if the processors are twice as powerful or doubled up. The process of applying the transformation to the steering vector and then using the unit vector or reverse flow method can also be performed on an O(n) system at the same rate as on an $O(n^2)$ system with the same processor speeds. The method depends upon the diagonal collapsing (Figure E-4). In this processor organization of each row has independent processors as does each column. Therefore, if only one input vector exists (one steering vector, K=1) then at each stage no waiting is encountered for busy processors. The following table specifies the conditions in which this method is equal in time to an $O(n^2)$ array.

	Unit Vector	Reverse Flow
Sequential	Never	K=1
Full Pipeline	$K+NK \leq 11$	$K \leq 11$
Optimal	$K+NK \leq 5$	$K \leq 5$

The above table shows that the unit vector method is never equal to $O(N^2)$ array times for N of interest (N > 10). The reverse flow method rating is independent of N, which is desirable. We also know that with diagonal collapsing, the reverse flow method can be used with forward flow.

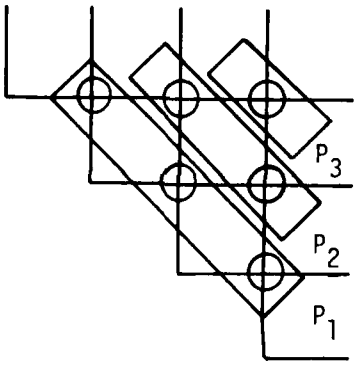


Figure E-4. Diagonal collapsing.

Appendix F

PROCESSING TIMING FOR BLOCK AVERAGE $O(n^2)$ SYSTEM

This appendix determines system throughputs as a function of the degrees of freedom, N, the number of samples, S, and the number of steering vectors, K. This process can be divided into three subfunctions:

1. Calculate all internal weights.
2. Pass S through array.
3. Solve for weights, using unit vector or reverse flow method.

We build up the timing equation in the above order for three implementations:

Sequential (worst case)

Full pipeline

Optimal (best case)

We also assume separate processors for denominator calculations.

F.1 CALCULATE ALL INTERNAL WEIGHTS

For each implementation method we have the following steps:

A. Compute numerator and denominator.

B. Compute reciprocal.

C. Multiply by reciprocal.

D. Pass data through, applying internal weight.

The last step can be combined with the first step, so that as the internal weights are applied the outputs are used to calculate the numerator and denominator of the next row. The timings for these basic steps are:

	A	B	C	D
Sequential	10St	7t	2t	11St
Full pipeline	10t + (S-1)t	7t	2t	11t + (S-1)t
Optimal	4t + (S-1)t	7t	2t	5t + (S-1)t

For a system of degree N there are N-1 internal weights; so the general timing formula in terms of stages is:

$$T = A + (N-1)B + (N-1)C + (N-2)[DA] \quad ,$$

where [DA] represents the time to apply the weights and calculate the numerator in the pipeline. The resultant specific formulas are:

	Arbitrary S & N	S = 2N
Sequential	11SN + 20N - 12S - 9	$22N^2 - 4N - 9$
Full pipeline	SN + 30N - S - 20	$2N^2 + 28N - 20$
Optimal	SN + 17N - S - 13	$2N^2 + 15N - 13$

F.2 PASS S THROUGH ARRAY

This is the same calculation as applying the internal weights to K input vectors (remember K is the number of steering vectors). The time formulas are as follows:

	Arbitrary N,K	K=1
Sequential	(11N + 11K - 22)t	(11N - 11)t
Full pipeline	(11N + K - 12)t	(11N - 11)t
Optimal	(5N + K - 6)t	(5N - 5)t

After passing the steering vectors through the array, the resultant vector must be multiplied by all the reciprocals of the denominators. Since these values have already been calculated, only a real-times-complex multiply must be performed, as follows:

	Sequential Multiply Multiply	Full pipeline Multiply Multiply	Optimal Multiply
Time	2t	2t	t
Total Time	(11N + 11K - 20)t	(11N + K - 10)t	(5N + K-5)t

F.3 SOLVE FOR WEIGHTS

We first explore the unit vector (UV) method. This method, as explained in Section 2.6.1, consists of passing N (or N-1) unit vectors through the arrays and performing a dot product on the output. This process is similar to passing S data vectors through the arrays, so the same formula holds. Since the steering vector (or vectors) will pass through the arrays first, we are passing a total of K+NK vectors (N unit vectors for each steering vector). The processing time for dot product is entirely overlapped by the pipeline nature of the array except for one complex multiply and add at the end. The delay for this last multiply/add is equivalent to applying the weight for one input (11t, 11t, and 5t for the three methods discussed).

Unit Vector Times

Sequential	11NK + 22N + 11K - 22
Full pipeline	NK + 11N + K - 12
Optimal	KN + 5N + K - 6

Time for the entire processing time can now be determined by adding these times to the times for passing through the raw data vectors. The resultant timing formulas are:

	Arbitrary N, S, K	K=1, S=2N
Sequential	11SN + 11NK + 33N - 12S + 11K - 23	$22N^2$ + 20N - 12
Full pipeline	SN + NK + 41N - S + K - 31	$2N^2$ + 40N - 30
Optimal	SN + KN + 22N - S + K - 19	$2N^2$ + 21N - 18

Reverse Flow Times

The reverse flow method is exactly the opposite of passing the steering vectors through the array, so the basic timing formula is:

PASS RAW DATA + 2 * PASS STEERING VECTOR + COMPLEX MULTIPLY

The extra complex multiply is to perform the final calculation of passing the steering vectors. The timing formulas are:

	Arbitrary N, S, K	K=1, S=2N
Sequential	11SN + 44N - 12S + 22K - 51	$22N^2$ + 20N - 29
Full pipeline	SN + 52N - S + 2K - 42	$2N^2$ + 50N - 40
Optimal	SN + 27N - S + 2K - 24	$2N^2$ + 25N - 22

Comparison of Reverse Flow and Unit Vector Times

We can compare the timing equations for reverse flow and unit vector methods to determine which is faster. The results of this comparison are shown below:

	Reverse Flow Faster If
Sequential	All cases
Full pipeline	$NK + 11 > 11N + K$
Optimal	$NK + 6 > 5N + K$

As K approaches N the reverse flow method is better. Which method is actually implemented is determined not only by K and N but also by the timing requirements of the system to determine the required level of complexity in the processors.

Appendix G

PROOF OF RESULTS ON GROWTH OF INTERMEDIATE RESULTS

We first consider the case of the algorithm without square roots, where the actual elements of the covariance matrix are passed through the array. Since the array does nothing more than the Cholesky LDL* decomposition of M, we will first change notation to show how the $Z_j^i(k)$ correspond to intermediate values in the Cholesky decomposition, and then with this simpler notation prove the stated results.

Our claim is that

$$Z_k^i(k) = m_{kk} - \sum_{r=1}^{i-1} \ell_{kr}\bar{\ell}_{kr}dr \qquad i \leq k \qquad \text{(G-1a)}$$

$$Z_k^k(k) = m_{kk} - \sum_{r=1}^{k-1} \ell_{kr}\bar{\ell}_{kr}dr = d_k \qquad \text{(G-1b)}$$

$$Z_j^i(k) = m_{jk} - \sum_{r=1}^{i-1} \bar{\ell}_{kr}\ell_{jr}dr \qquad i \leq k \qquad \text{(G-1c)}$$

$$w_{kj} = Z_j^k(k)/Z_k^k(k) = \ell_{jk} \qquad j > k \qquad \text{(G-1d)}$$

where $L = \{\ell_{jk}\}$(unit lower triangular), $D = \text{diag}\{d_j\}$, $M = \{m_{jk}\}$, and $M=LDL^*$. The proof is by noting that the array and the Cholesky decomposition perform the same column operations on M, and that the array recomputes some intermediate values (because no time is lost due to parallelism, and interprocessor communication is simplified).

We may now show

$$\lambda_{min} \leq Z_k^k(k) = m_{kk} - \sum_{r=1}^{k-1} \ell_{kr}\bar{\ell}_{kr}dr \leq m_{max} \qquad \text{(G-2a)}$$

$$\lambda_{min} \leq Z_k^i(k) = m_{kk} - \sum_{r=1}^{i-1} \ell_{kr}\bar{\ell}_{kr}dr \leq m_{max} \quad (i<k) \qquad \text{(G-2b)}$$

where λ_{min} = minimum eigenvalue of M and $m_{max} = \max_{ij} |m_{ij}|$.

We need several well-known facts. $m_{max} = \max_i |m_{ii}|$ since our matrix is positive definite Hermitian [Isaacson and Keller 1966]. The separation theorem [Franklin 1968]* states that if M is an nxn positive definite Hermitian matrix, and if M_m is the mxm matrix consisting of the first m rows and n columns of M with eigenvalues $\lambda_1^m \le \lambda_2^m \le \ldots \le \lambda_m^m$, then the λ_j^i satisfy $\lambda_1^m \le \lambda_1^{m-1} \le \lambda_2^m \le \ldots \le \lambda_{m-1}^{m-1} \le \lambda_m^m$. In other words, the m-1 eigenvalues lie in between, or separate the m eigenvalues of the larger matrix. Also, if M=LDL*, D=diag(d_j), then $\det M_m = \prod_{j=1}^{m} \lambda_j^m = \prod_{j=1}^{m} d_j$, the equality of which follows from the fact that $\det M = \det D = \Pi d_j$ and the Cholesky decomposition of M_m produces the same ℓ_{ij} and d_i values for $i \le m$. Finally, to prove Eq. (G-2a), we write

$$d_m = \prod_{j=1}^{m} d_j \Big/ \prod_{j=1}^{m-1} d_j = \det M_m / \det M_{m-1} = \prod_{j=1}^{m} \lambda_j^m \Big/ \prod_{j=1}^{m-1} \lambda_j^{m-1} \quad ;$$

so since

$$\lambda_j^{m-1} \le \lambda_{j+1}^m \quad ,$$

$$d_m \ge \prod_{j=1}^{m} \lambda_j^m \Big/ \prod_{j-1}^{m-1} \lambda_j^m{}_{+1} = \lambda_1^m \ge \lambda_1^n = \lambda_{min} \quad .$$

The other half of the inequality follows by our noticing that the defining equation for d_k (Eq. (G-1b)) starts with $m_{kk} \le m_{max}$ and subtracts positive numbers $|\ell_{kr}|^2 d_r$ ($d_r \ge \lambda_{min} > 0$ since M is positive definite). Eq. (G-2b) follows from (G-2a) since $\lambda_{min} \le Z_k^k(k) \le Z_k^i(k) \le m_{kk} \le m_{max}$, and the sum for $Z_k^k(k)$ subtracts more positive terms than the sum for $Z_k^i(k)$. That these bounds are sharp is obvious by considering any diagonal matrix.

---

*J. N. Franklin, Matrix Theory, Englewood Cliffs, New Jersey, Prentice-Hall, 1968.

Next, we show

$$|Z_j^k(k)| \leq m_{max} \tag{G-2c}$$

$$|Z_j^i(k)| \leq 2m_{max} \quad (i<k) \quad . \tag{G-2d}$$

Let $\tilde{\ell}_{kr} = \ell_{kr}\sqrt{d_r}$ so that $d_k\,\ell_{jk} = Z_j^k(k) = m_{jk} - \sum_{r=1}^{k-1} \tilde{\ell}_{kr}\,\bar{\tilde{\ell}}_{jr}$ $(k<j)$ so $Z_j^k(k) = \sqrt{d_k}\ (\sqrt{d_k}\,\ell_{jk}) = \tilde{\ell}_{kk}\,\tilde{\ell}_{jk}$. Since $m_{kk} = \sum_{r=1}^{k} |\tilde{\ell}_{kr}|^2$ and $m_{jj} = \sum_{r=1}^{j} |\tilde{\ell}_{jr}|^2$ we have $\tilde{\ell}_{kk} < \sqrt{m_{max}}$ and $\tilde{\ell}_{jk} < \sqrt{m_{max}}$ so $|Z_j^k(k)| \leq m_{max}$, proving Eq. (G-2c). Also $|Z_j^i(k)| = |m_{jk} - \sum_{r=1}^{k-1} \tilde{\ell}_{kr}\,\bar{\tilde{\ell}}_{jr}| = |m_{jk} - (k^{th}$ row of L up to i-1, j^{th} row of L up to i-1)$| \leq |m_{jk}| + |(k^{th}$ row of L up to i-1, j^{th} row of L up to i-1)| (where $(\cdot,\cdot)$ is a complex dot product)
$\leq m_{max} + ||k^{th}$ row of L up to i-1$|| \cdot ||j^{th}$ row of L up to i-1$||$
(by Cauchy Schwartz, where $||\cdot||$ = vector norm)
$\leq m_{max} + ||k^{th}$ row of L$|| \cdot ||j^{th}$ row of L$||$

$$= m_{max} + \sqrt{\sum_{r=1}^{k} |\tilde{\ell}_{kr}|^2}\ \sqrt{\sum_{r=1}^{j} |\tilde{\ell}_{jr}|^2}$$

$$= m_{max} + \sqrt{m_{ik}}\sqrt{m_{jj}} \leq m_{max} + m_{max} = 2m_{max},$$

proving Eq. (G-2d). To see that Eq. (G-2c) is sharp and (G-2d) is sharp to within a factor of 2, consider the matrix

$$\begin{bmatrix} a+b & a-b \\ a-b & a+b \end{bmatrix} \text{ for } 0 < b << a.$$

Finally, we prove

$$|w_{jk}| = |Z_j^k(k) \Big/ Z_k^k(k)| \leq \sqrt{\frac{m_{max}}{\lambda_{min}}} \quad . \tag{G-2e}$$

Since $m_{kk} = \sum_{r=1}^{k} |\ell_{kr}|^2$ we have $|\tilde{\ell}_{kr}| \leq \sqrt{m_{max}}$.

Also, $|w_{kj}| = |\tilde{\ell}_{jk} \Big/ \sqrt{d_k}| \leq \frac{\sqrt{m_{max}}}{\sqrt{d_k}} \leq \sqrt{\frac{m_{max}}{\lambda_{min}}}$ since $d_k \geq \lambda_{min}$. To see that this bound is sharp to within a factor of 2, consider

$$\begin{bmatrix} \frac{1}{a^2} & \frac{1}{a} \\ \frac{1}{a} & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \text{ for large } a;$$

so $\lambda_{min} \approx \frac{1}{2a^2}$, $m_{max} = 2$, $\sqrt{\frac{m_{max}}{\lambda_{min}}} \approx 2a$.

Next we consider the version of the algorithm without square roots when sample vectors are used to compute the w_{ij}'s. We denote the intermediate results by $Z'^i_j(k)$ to distinguish them from the $Z^i_j(k)$ above, when the actual matrix was used. We interpret the inputs $Z^1_j(k)$ as Gaussian random variables with zero means and covariance matrix M, and compute the distributions of the $Z^i_j(k)$ which are functions of the $Z^i_j(k)$. The actual distributions of the $Z^i_j(k)$ are too complicated to compute exactly, but we

make the approximation of replacing the w_{ij}'s by their means as soon as they are computed. When the number of samples is large, this is an excellent approximation. (For other properties of these distributions, see [Rao 1965]*.) It is in fact possible to write down the exact distribution of the w_{ij}'s, see Rao, p. 508. With this approximation we note that $Z_j^i(k)$ is a linear combination of the input random variables, and hence is Gaussian with zero mean. If we compute its variance, then we will know the distribution of its possible values and hence how many bits are required to represent it.

We now prove

$$\text{var}[Z'^i_j(k)] = Z_j^i(j) \qquad \text{(G-3a)}$$

where $Z_j^i(j)$ is computed as though the actual matrix were being passed through. Thus $\lambda_{min} \leq \text{var}[Z'^i_j(k)] \leq m_{max}$, using the results of the last analysis.

In other words, after the random variables have passed through i rows of the array, their variances equal the i^{th} partial sums of the expressions for d_j. Let V be the column vector of random variables, so that $E(VV^*) = M$ and $E[(L^{-1}V)(L^{-1}V)^*] = L^{-1}E(VV^*)L^{-1*} = D$. Let L_m represent the transformation performed by the first m rows. Then

*C.R. Rao, Linear Statistical Inference and Its Applications, New York, John Wiley and Sons, 1965.

$$L_m = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -w_{mm+1} & 1 & \\ & & \vdots & & \ddots \\ & & -w_{mn} & & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & & \\ -w_{12} & 1 & \\ -w_{13} & & \ddots \\ \vdots & & \\ -w_{1n} & & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & +w_{m+1m+2} & 1 & \\ & & \vdots & & \ddots \\ & & +w_{m+1n} & & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ & & w_{n-1n} \end{bmatrix} \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ & & -w_{n-1n} \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & -w_{m+1m+2} & \\ & & \vdots & \ddots \\ & & -w_{m+1n} & & 1 \end{bmatrix} \cdots \cdot L_{-m}$$

$$= \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & w_{m+1m+2} & & \\ & & \vdots & 1 & \\ & & & & \ddots \\ & & w_{m+1n} & & w_{n-1n} \quad 1 \end{bmatrix} \cdot L^{-1} \quad .$$

Partitioning L as

$$L = \left[\begin{array}{ccc|ccc} 1 & & & & & \\ w_{12} & 1 & & & 0 & \\ \vdots & \ddots & \ddots & & & \\ \vdots & & \ddots & & & \\ \vdots & & & & & \\ w_{1m} & \cdots & 1 & & & \\ \hline w_{1m+1} & \cdots & w_{mm+1} & 1 & & \\ \vdots & & \vdots & w_{m+1m+2} & 1 & \\ \vdots & & \vdots & \vdots & \ddots & \\ w_{1n} & & w_{mn} & w_{m+1n} & & 1 \end{array}\right] = \left[\begin{array}{c|c} L_1 & 0 \\ \hline A & L_2 \end{array}\right]$$

and D as

$$D = \left[\begin{array}{ccc|ccc} d_1 & & & & 0 & \\ & \ddots & & & & \\ & & d_m & & & \\ \hline & & & d_{m+1} & & \\ 0 & & & & \ddots & \\ & & & & & d_n \end{array}\right] = \left[\begin{array}{c|c} D_1 & 0 \\ \hline 0 & D_2 \end{array}\right]$$

we see we can write

$$L_{-m} = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & L_2 \end{array}\right] L^{-1} \quad ;$$

so $M_m = E[(L_{-m}V)(L_{-m}V)^*] = L_{-m} M L_{-m}^*$

$$= \left[\begin{array}{c|c} I & 0 \\ \hline 0 & L_2 \end{array}\right] L^{-1} LDL^* L^{-1*} \left[\begin{array}{c|c} I & 0 \\ \hline 0 & L_2^* \end{array}\right]$$

$$= \left[\begin{array}{c|c} D_1 & 0 \\ \hline 0 & L_2 D_2 L_2^* \end{array}\right]$$

It is the diagonal elements of M_m that are our answers. The first m diagonal elements are obviously equal to their final values. Now $M = LDL^*$

$$= \left[\begin{array}{c|c} L_1 & 0 \\ \hline A & L_2 \end{array}\right]\left[\begin{array}{c|c} D_1 & 0 \\ \hline 0 & D_2 \end{array}\right]\left[\begin{array}{c|c} L_1^* & A^* \\ \hline 0 & L_2^* \end{array}\right] \left[\begin{array}{c|c} L_1 D_1 L_1^* & L_1 D_1 A^* \\ \hline AD_1 L_1^* & AD_1 A^* + L_2 D_2 L_2^* \end{array}\right]$$

Hence

$$(*)\ (L_2 D_2 L_2^*)_{ij} = (AD_1 A^* + L_2 D_2 L_2^*)_{ij} - (AD_1 A^*)_{ij}$$

$$= m_{ij} - \sum_{r=1}^{m} \ell_{ir} \overline{\ell}_{jr}\, dr \quad .$$

Set i=j and we get the desired result.

Next we prove that $\tilde{w}_{ij}$ is approximately Gaussian with mean $E(\tilde{w}_{ij}) = Z_j^i(i)$; so $|E(\tilde{w}_{ij})| \leq m_{max}$ and variance

$$\text{var}(\tilde{w}_{ij}) = \frac{1}{S}\left(Z_i^i(i)\, Z_j^i(j) + |Z_i^i(j)|^2\right)$$

$$= \frac{1}{S}\left[d_i \cdot \left(m_{jj} - \sum_{r=1}^{i-1} |\ell_{jr}|^2 dr\right) + \left|m_{ij} - \sum_{r=1}^{i-1} \ell_{ir}\, \bar{\ell}_{jr}\, dr\right|^2\right]$$

$$\leq 2\, m_{max}^2/S \quad . \tag{G-3b}$$

If Q_1 and Q_2 are normal with covariance matrix $\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$, it is easy to see $E(Q_1\bar{Q}_2) = m_{12}$ and $\text{var}(Q_1\bar{Q}_2) = m_{11}\, m_{22} + m_{12}\, m_{21}$. Since linear combinations of multivariate Gaussian random variables are multivariate Gaussian, we need only find out what the variances and covariances are of the variables used to form $\tilde{w}_{ij}$. We approximate $\tilde{w}_{ij}$ as Gaussian by the central limit theorem. They are given by (*) with m=i, and this is our result. The 1/S factor is used to cancel one of the S factors "concealed" in m_{max}^2. Since $\tilde{w}_{ij} \approx N(\mu,\sigma^2)$ with $|\mu| \leq m_{max}$ and $\sigma^2 \leq 2m_{max}^2/S$ we have

$$P(|w_{ij}| \geq 2^t\, m_{max}) = P(w_{ij} \leq 2^t\, m_{max}) + P(w_{ij} \leq -2^t\, m_{max})$$

$$= P\left(\frac{w_{ij} - \mu}{\sqrt{\frac{2}{S}}\, m_{max}} \geq \frac{2^t m_{max} - \mu}{\sqrt{\frac{2}{S}}\, m_{max}}\right) + P\left(\frac{w_{ij} - \mu}{\sqrt{\frac{2}{S}}\, m_{max}} \leq \frac{-2^t m_{max} + m_{max}}{\sqrt{\frac{2}{S}}\, m_{max}}\right)$$

$$\leq P\left[N(0,1) \geq \frac{2^t m_{max} - m_{max}}{\sqrt{\frac{2}{S}}\, m_{max}}\right] + P\left[N(0,1) \leq \frac{-2^t m_{max} + m_{max}}{\sqrt{\frac{2}{S}}\, m_{max}}\right]$$

$$= P\left[N(0,1) \geq \sqrt{\frac{S}{2}}\,(2^t - 1)\right] + P\left[N(0,1) \leq -\sqrt{\frac{S}{2}}\,(2^t - 1)\right]$$

$$= P\left[N(0,1) \geq \sqrt{\frac{S}{2}}\,(2^t - 1)\right] \quad .$$

$\tilde{\tilde{w}}_{ij}$ is approximately χ^2 with S degrees of freedom, mean

$E(\tilde{\tilde{w}}_{ij}) = Z^i_i(i)$, $|E(\tilde{\tilde{w}}_{ij})| \leq m_{max}$ and variance

$$var(\tilde{\tilde{w}}_{ij}) \leq 2\, m^2_{max}/S \quad . \qquad \text{(G-3c)}$$

Since $E(\tilde{\tilde{w}}_{ij}) = var[Z'^i_i(i)]$, the proof of the first part is easy. Recalling that the expectation of the fourth power of a standard normal variable is 3, we obtain $var(\tilde{\tilde{w}}_{ij}) = 2\, var^2[Z'^i_i(i)]/S \leq 2\, m^2_{max}/S$. Thus

$$P\left(\tilde{\tilde{w}}_{ij} \geq 2^t m_{max}\right) = P\left(\tilde{\tilde{w}}_{ij}\,\frac{S}{m_{max}} \geq 2^t S\right) = P\left(\chi^2_S \leq 2^t S\right) \quad .$$

Appendix H

SAMPLE VOLTAGE VECTOR MODEL

To determine if the sample covariance matrix approach is feasible when N, the number of weights, is at least 200, we had to use a model of the sample voltage vectors to write a computer simulation to test the algorithm and its implementation.

The radar test problem is arranged so that the interferers can be specified by their eigenvalues. To do this simply, a linear antenna array with uniform spacing and weighting was chosen. With this configuration it is easy to form multiple beams and place an interferer in each beam so that each beam output contains only power from that interferer. Since each beam output is then independent, the covariance matrix of the beam output is diagonal and the interferer powers are the eigenvalues. It is easy to transform the problem to element space, using a unitary transformation.

To implement this approach, the interferers are placed so that all except one are at nulls of the beam pattern for each beam. The voltage output of a beam formed by summing all element outputs is given by

$$X_b = [1 + e^{i\psi} \ldots + e^{i(N-1)\psi}] = \frac{\sin N(\psi/2)}{\sin \psi/2} \quad ,$$

where

$\psi = 2\pi D \sin \theta$,

θ = an angle measured from the boresight of the antenna to a point source, and

D = the antenna element separation in wavelengths.

If interferer positions are chosen so that $\sin [N(\psi_n - \psi_m)/2] = 0$ for $n \neq m$, they satisfy the condition for independence. This will be true when

$N/2(\psi_n - \psi_m) = k\pi$, where k is an integer. The desired interferer positions are, therefore,

$$\psi_k = \frac{2\pi k}{N} \quad , k = 1,\ldots,N-1 \quad . \tag{H-1}$$

In element space the voltage at each of N elements is given by

$$X_n = \sum_{k=1}^{K} R_k \sqrt{\lambda_k}\, e^{i2\pi k(N-1)} + \sqrt{Q_n} \cdot R_n \tag{H-2}$$

for the K = N-1 interferers. The λ_k are the powers in each interferer and the R's are independent zero-mean random variables with $|\overline{R_k}|^2 = 1$. The R_k are included, when needed, to simulate the interferer variations between samples, and receiver noise whose power is Q_n.

Since the interferer positions are known, the true covariance matrix can be computed as

$$M_{m,n} = \overline{X_m^* X_n} = 1/K \sum_{k=1}^{K} \lambda_k\, e^{-i\left[2\pi k/N(m-n)\right]} \quad . \tag{H-3}$$

A stochastic sample covariance matrix using S samples can be obtained by forming

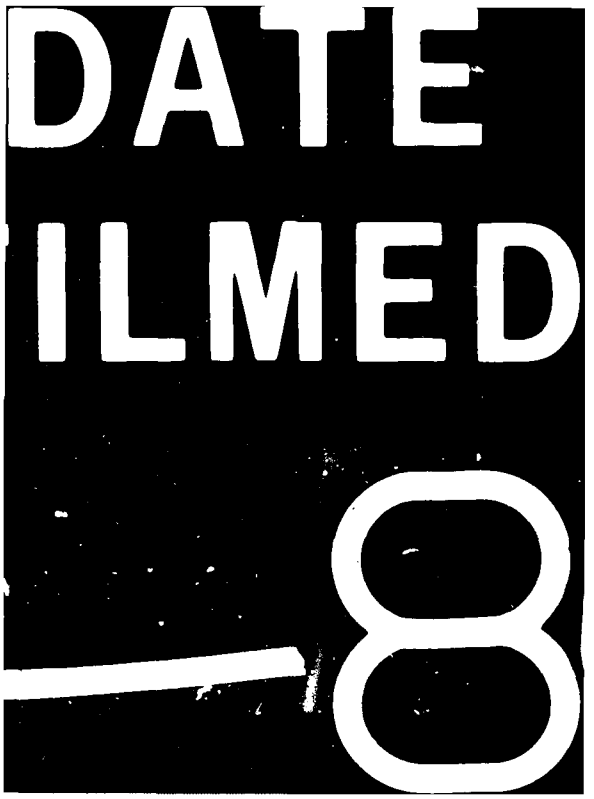
$$\hat{M}_{m,n} = 1/S \sum_{s=1}^{S} {}_sX_m^*\, {}_sX_n \quad , \tag{H-4}$$

where the voltages are obtained from Eq. (H-2).

With this model the eigenvalues λ_k can be chosen to span any desired range of values, and the number of interferers can be varied up to N-1 while one specifies the eigenvalues.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


DATE
ILMED
8