90-42C

Progress in Systems and Control Theory
Volume 5

Series Editor

Christopher I. Byrnes, Washington University

Associate Editors

S.-I. Amari, University of Tokyo, Japan
B.D.O. Anderson, Australian National University, Canberra, Australia
Karl J. Åström, Lund Institute of Technology, Lund, Sweden
Jean-Pierre Aubin, CEREMADE, Paris, France
H.T. Banks, University of Southern California, Los Angeles, California
John S. Baras, University of Maryland, College Park, Maryland
A. Bensoussan, INRIA, Paris, France
John Burns, Virginia Polytechnic Institute, Blacksburg, Virginia
Han-Fu Chen, Beijing University, People's Republic of China
M.H.A. Davis, Imperial College of Science and Technology, London, England
Wendell Fleming, Brown University, Providence, Rhode Island
Michel Fliess, CNRS-ESE, Gif-sur-Yvette, France
Keith Glover, University of Cambridge, Cambridge, England
D. Hinrichsen, University of Bremen, Federal Republic of Germany
Alberto Isidori, University of Rome, Italy
B. Jakubczyk, Academy of Sciences, Warsaw, Poland
Hidenori Kimura, Osaka University, Japan
Arthur Krener, University of California, Davis
H. Kunita, Kyushu University, Japan
Alexandre Kurzhansky, IIASA, Laxenburg, Austria
Harold M. Kushner, Brown University, Providence, Rhode Island
Anders Lindquist, Royal Institute of Technology, Stockholm, Sweden
Andrzej Manitius, George Mason University, Fairfax, Virginia
Clyde F. Martin, Texas Tech University, Lubbock, Texas
Sanjoy Mitter, Massachusetts Institute of Technology, Cambridge, Massachusetts
Giorgio Picci, LADSEB-CNR, Padova, Italy
Boris Pshenichnyj, Glushkov Institute of Cybernetics, Kiev
H.J. Sussmann, Rutgers University, New Brunswick, New Jersey
T.J. Tarn, Washington University, St. Louis, Missouri
V.M. Tikhomirov, Institute for Problems in Mechanics, Moscow
Pravin P. Varaiya, University of California, Berkeley
Jan C. Willems, University of Gröningen, The Netherlands
W.M. Wonham, University of Toronto

M.A. Kaashoek
J.H. van Schuppen
A.C.M. Ran
Editors

Signal Processing, Scattering and Operator Theory, and Numerical Methods

Proceedings of the International Symposium MTNS-89, Volume III

1990

Birkhäuser
Boston · Basel · Berlin

5. Acknowledgement.

We thank IBM Deutschland and in particular Dr. R. Janßen for giving us the opportunity to carry out this work.

6. References.

[1] M. Athans, P.L. Falb, *Optimal Control*, MacGraw Hill, New York (1966).

[2] K.E. Brenan, S.L. Campbell, L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elsevier, New York, N.Y. (1989).

[3] P. Deuflhard, E. Hairer, J. Zugck, *One-step and Extrapolation Methods for Differential-Algebraic Systems*, Numer. Math. 51 (1987), 501–516.

[4] F.R. Gantmacher, Matrix Theory, Vol. II, Chelsea, New York (1959).

[5] C.W. Gear, L. Petzold, *Differential-Algebraic Systems and Matrix Pencils*, in Matrix Pencils, B. Kågström, A. Ruhe, Eds., Springer, Berlin (1983), 75–90.

[6] G.H. Golub, C.R. Van Loan, *Matrix Computation*, North Oxford Academic, Oxford (1983).

[7] H.W. Knobloch, H. Kwakernaack, *Lineare Kontrolltheorie*, Springer, Berlin (1985).

[8] P. Kunkel, V. Mehrmann, *Numerical solution of differential algebraic Riccati equations*, Technical report, IBM Science Center Heidelberg, TR 89.03.006, to appear in Lin. Alg. Appl. (1989).

[9] V. Mehrmann, *The Linear Quadratic Control Problem: Theory and Numerical Algorithms*, Habilitationsschrift, Universität Bielefeld (1987).

[10] V. Mehrmann, *Existence, Uniqueness and Stability of Solutions to Singular Linear Quadratic Optimal Control Problems*, Lin. Alg. Appl., 121 (1989), 291–331.

[11] L. Petzold, *A description of DASSL: A Differential-Algebraic System Solver*, IMACS Trans. on Scient. Comp. Vol. 1, R.S. Stepleman, Ed. (1982).

[12] L. Petzold, *Differential/Algebraic Equations are not ODE's*, SIAM J. Sci. Stat. Comput. 3 (1982), 367–384.

[13] W.T. Reid, *Riccati Differential Equations*, Academic Press, New York (1972).

[14] W. Rheinboldt, *On the Computation of Multi-Dimensional Solution Manifolds of Parametrized Equtions*, Numer. Math. 53 (1988), 165–181.

SVD updating for tracking slowly time-varying systems. A parallel implementation.

Marc Moonen, Paul Van Dooren, Joos Vandewalle

Abstract

A procedure is presented for updating the singular value decomposition of a matrix as new rows are appended successively. The updating consists in an interlacing of QR-updatings and a Jacobi-type SVD-algorithm applied to the triangular factor. At any time step an approximate decomposition is computed from a previous approximation, with a limited number of operations ($O(n^2)$). When combined with exponential weighting, the algorithm is seen to be highly applicable to tracking problems, as will be demonstrated with an example. Furthermore, this algorithm can elegantly be mapped onto a systolic array, making use of slight modifications of well known systolic implementations for the matrix-vector product, the QR-updating and the SVD.

1 Introduction

Common techniques for updating the singular value decomposition (SVD) of a matrix after appending new rows, mostly reduce to solving the corresponding modified symmetric eigenvalue problem [2, 3, 5]. A drawback is the necessary knowledge of the *exact* original eigenstructure, in order to compute the updated eigenstructure. On the one hand, this considerably increases the computational load (e.g. in an on-line environment, where at each time step an *exact* updating is thus to be performed), while on the other, round-off errors are likely to accumulate unboundedly.

In this paper, we present an SVD updating procedure that interlaces a Jacobi-type SVD algorithm [1, 7] with a QR-updating each time a new row is to be worked in. After each QR-update, the SVD algorithm reduces the R-factor to an approximately diagonal matrix, by applying a sequence of plane rotations. It may be appropriate to try and perform only a fixed number of rotations (e.g. $n-1$, see below) on the triangular factor after each QR-update. This results in a seemingly *non-iterative* algorithm that at any time step outputs an *approximate decomposition* computed from a previous approximation, with only a limited number of operations (viz. $O(n^2)$). The crux is then to show how only a few SVD-steps after each QR-update can restore an acceptable approximation at every stage.

When combined with exponential weighting, this algorithm is seen to be highly applicable to *tracking problems*. Furthermore, the updating algorithm can be mapped onto a *systolic array* with $O(n^2)$ parallelism for $O(n^2)$ operations, by appropriately combining systolic implementations for the matrix-vector product, the QR-updating and the SVD.

The SVD updating procedure is presented in section 2 and illustrated by a practical example in section 3. Finally, in section 4, a systolic implementation is presented.

2 Updating the singular value decomposition

The SVD of a matrix $A_{m\times n}$ $(m \geq n)$ is a factorization of A into a product of three matrices.

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}V^t_{n\times n}$$

where $U^tU = I$, $V^tV = I$, and Σ is a pseudo-diagonal matrix, with the singular values on its main diagonal.

Updating the SVD after appending a new row, consists in computing the SVD of the modified matrix

$$\begin{bmatrix} \lambda.A \\ a^t \end{bmatrix} = U^*_{(m+1)\times n}\Sigma^*_{n\times n}V^{*^t}_{n\times n}$$

making use of the original SVD of A. The exponential weighting factor $\lambda \leq 1$ accounts for the reduction of "older" data. In on-line applications, a continuous updating after each sampling is to be performed, where in most cases only V and Σ are of interest.

We first briefly review a Jacobi-type SVD-algorithm, originally due to Kogbetliantz [7]. Next we show how an adaptive algorithm can be constructed by interlacing the SVD-diagonalization procedure with repeated QR-updates.

A Jacobi-type SVD algorithm

In order to compute the SVD of A, it is known to be computationally more efficient to first perform a QR-factorisation of A

$$A_{m\times n} = Q_{m\times n}.R_{n\times n}$$

where $Q^tQ = I$, and R is upper triangular. Obviously, V and Σ can be computed from the SVD of R as well.

$$R = U_R.\Sigma_R.V_R^t = U_R.\Sigma.V^t$$

The SVD of R can be computed by making use of a Jacobi strategy [8], that generates a sequence of triangular matrices R_k via the recurrence

$$R_k \;=\; \Theta_k^t.R_{k-1}.\Phi_k$$

with $R_0 = R$, and Θ_k and Φ_k to be defined next. As each iteration reduces the norm of the off-diagonal elements, R is gradually reduced to a diagonal matrix. Suppose that after r rotations a diagonal matrix (up to the relative precision ϵ) is obtained, then

$$\Theta_r^t.\Theta_{r-1}^t\ldots\Theta_1^t.(R).\Phi_1\ldots\Phi_{r-1}.\Phi_r = R_r \overset{\epsilon}{\simeq} \Sigma_R = \Sigma$$

and

$$V = V_R \overset{\epsilon}{\simeq} \Phi_1.\ldots.\Phi_{r-1}.\Phi_r$$

Θ_k and Φ_k are plane rotations through angles θ_k and ϕ_k in the $(i, i+1)$-plane, up to a row or column permutation. The rotation angles θ_k and ϕ_k are chosen such that the $(i, i+1)$ element in R_k transforms to zero,

$$\begin{bmatrix} r_{k+1}^{i,i} & 0 \\ 0 & r_{k+1}^{i+1,i+1} \end{bmatrix} := \begin{bmatrix} \sin\theta_k & \cos\theta_k \\ \cos\theta_k & -\sin\theta_k \end{bmatrix}\begin{bmatrix} r_k^{i,i} & r_k^{i,i+1} \\ 0 & r_k^{i+1,i+1} \end{bmatrix}\begin{bmatrix} \sin\phi_k & \cos\phi_k \\ \cos\phi_k & -\sin\phi_k \end{bmatrix}$$

Each iteration thus reduces to a 2×2 SVD on the main diagonal, setting the super-diagonal ("pivot"-) element $r^{i,i+1}$ equal to zero. The off-diagonal elements can then be annihilated in a cyclic manner . An element that is being annihilated in one step, reappears in the next step, but mostly with a reduced modulus. By making use of *outer rotations* on the one hand, and an elegant *pipelining* of forward and backward sweeps on the other, an efficient implementation can be obtained [12], where furthermore the triangular structure is preserved at every stage. It then suffices to let the pivot index successively take up all possible values

$$i = 1, 2, \ldots, n-1$$

in a cyclic manner, and perform 2×2 SVD's (with outer rotations) on the main diagonal accordingly [8]. The updating scheme presented in this paper interlaces a QR-update in between such sequences of $(n-1)$ 2×2 SVD's, as will be detailed next.

SVD-updating

If the rows of A become sequentially available, as is the case in an on-line environment, the diagonalization procedure can be interlaced with QR-updates [6]. After each QR-updating, the iterative diagonalization process of the R-factor is resumed. A sequential description is as follows.

Suppose that, at a certain time step k, the $A^{(k)}$-matrix is reduced to $R^{(k)}$ (upper triangular, almost diagonal). The corresponding V-matrix is $V^{(k)}$. (The $U^{(k)}$-matrix is not stored).

$$A^{(k)} = U^{(k)}.R^{(k)}.V^{(k)^t}$$

The modified matrix then equals

$$A^{(k+1)} = \begin{bmatrix} \lambda.A^{(k)} \\ a^{(k)^t} \end{bmatrix} = \begin{bmatrix} U^{(k)} & \\ & 1 \end{bmatrix}\begin{bmatrix} \lambda.R^{(k)} \\ a^{(k)^t}.V^{(k)} \end{bmatrix}V^{(k)^t}$$

Clearly, the R-factor is to be updated with the transformed input vector $a_*^{(k)^t} = a^{(k)^t}.V^{(k)}$, resulting in

$$A^{(k+1)} = \left(\begin{bmatrix} U^{(k)} & \\ & 1 \end{bmatrix}.Q^{(k)}_{(n+1)\times n}\right)\begin{bmatrix} R_*^{(k)} \\ 0 \end{bmatrix}V^{(k)^t}$$

The QR-updating clearly does not alter the $V^{(k)}$-matrix. Finally this updated $R_*^{(k)}$ is further diagonalized through a series of subsequent rotations.

$$\begin{aligned} R^{(k+1)} &= \Theta_{k_r}^t.\Theta_{k_{r-1}}^t\ldots\Theta_{k_1}^t.(R_*^{(k)}).\Phi_{k_1}\ldots\Phi_{k_{r-1}}.\Phi_{k_r} \\ V^{(k+1)} &= V^{(k)}.\Phi_{k_1}.\ldots.\Phi_{k_{r-1}}.\Phi_{k_r} \end{aligned}$$

(where k_i refers to the i^{th} rotation in time step k.)

Thus, the QR-updatings and the diagonalization procedure are interlaced, yielding an approximate V-matrix at every stage. The *number of rotations* k_r should be chosen as a function of the off-diagonal part ("off-norm") in $R_+^{(k)}$. Usually only very few rotations are necessary to restore a reasonable approximation at any time step. The choice of $(n-1)$ **rotations following each QR-update** (where the pivot index successively takes up all values $i = 1, \ldots, n-1$), seems particularly well suited for an efficient interlacing of QR-updates and SVD-steps. Moreover, for most practical applications, a reasonably low off-norm is observed when only $n-1$ rotations are performed after each QR-update, as will be demonstrated with an example in the next section.

3 Adaptive parameter estimation

The SVD-updating procedure can be used for adaptive (Total Least Squares) parameter estimation. An input-output sequence was generated, where the input $u^{(k)}$ $(k = 1, \ldots)$ is a white noise sequence, while the output $y^{(k)}$ at any time step k satisfies the second order difference equation

$$y^{(k)} = \alpha_1^{(k)}.y^{(k-1)} + \alpha_2^{(k)}.y^{(k-2)} + \alpha_3^{(k)}.u^{(k)} + \alpha_4^{(k)}.u^{(k-1)} + \alpha_5^{(k)}.u^{(k-2)}$$

The coefficients $\alpha_i^{(k)}$ were chosen to be sinusoidally time-varying.
Clearly, if these coefficients were constants, the exponentially weighted matrix

$$A^{(k)} = \begin{bmatrix} y^{(k)} & y^{(k-1)} & y^{(k-2)} & u^{(k)} & u^{(k-1)} & u^{(k-2)} \\ \lambda.y^{(k-1)} & \lambda.y^{(k-2)} & \lambda.y^{(k-3)} & \lambda.u^{(k-1)} & \lambda.u^{(k-2)} & \lambda.u^{(k-3)} \\ \lambda^2.y^{(k-2)} & \lambda^2.y^{(k-3)} & \lambda^2.y^{(k-4)} & \lambda^2.u^{(k-2)} & \lambda^2.u^{(k-3)} & \lambda^2.u^{(k-4)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \lambda^{k-3}.y^{(3)} & \lambda^{k-3}.y^{(2)} & \lambda^{k-3}.y^{(1)} & \lambda^{k-3}.u^{(3)} & \lambda^{k-3}.u^{(2)} & \lambda^{k-3}.u^{(1)} \end{bmatrix}$$

would have its last singular value equal to zero, corresponding to a right singular vector (null space)

$$v_6 \parallel [-1 \;\; \alpha_1 \;\; \alpha_2 \;\; \alpha_3 \;\; \alpha_4 \;\; \alpha_5]'$$

With "slowly" time-varying coefficients $\alpha_i^{(k)}$, the smallest singular value would be *approximately* equal to zero, provided the weighting factor λ is chosen sufficiently small. At each time instance k, the corresponding right singular vector v_6 is then a reasonable approximation (up to a factor of proportionality) for

$$[-1 \;\; \alpha_1^{(k)} \;\; \alpha_2^{(k)} \;\; \alpha_3^{(k)} \;\; \alpha_4^{(k)} \;\; \alpha_5^{(k)}]'$$

The coefficients $\alpha_i^{(k)}$ can thus approximately be identified from the SVD of the above matrix, where in each time step a new row is appended. The SVD can then be computed adaptively with the SVD-updating procedure. The exponential weighting factor was set equal to $1 - 2^{-5}$, and only $n - 1 = 5$ rotations were performed after each QR-update. The true and the reconstructed coefficients are seen to be in good mutual aggreement (Figure 1).

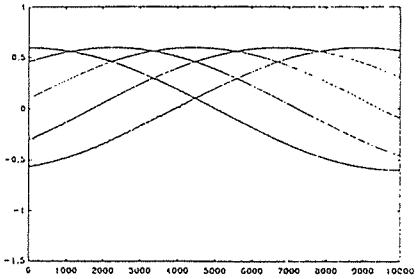


Figure 1: Reconstructed and true coefficients, as a function of time

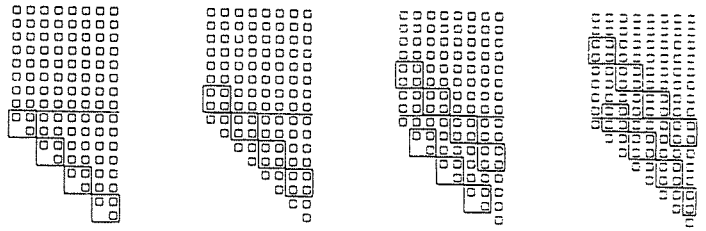


Figure 2: SVD array

4 A systolic implementation

The SVD-updating algorithm can elegantly be mapped onto a systolic array. In particular if after each QR-update, $n-1$ Jacobi rotations are performed, an efficient parallel implementation is conceivable with $O(n^2)$ parallelism for $O(n^2)$ complexity. (If the number of rotations after each QR-update is e.g. halved or doubled, the array can easily be modified accordingly.)

As we intend to combine SVD-diagonalization and QR-updating on the same array, let us first briefly review the triangular SVD-array, as described in [8] (Figure 2). Processor elements on the main diagonal perform 2×2 SVD's (combined with column and row permutations). Row transformation parameters are passed on to the right, while column transformation parameters are passed on upwards. Off-diagonal processors only apply and propagate these transformations. Column transformations are also propagated through the upper square part, containing the right singular V-matrix (V's first row in the top row, etc.).

Off-diagonal elements with odd and even row numbers are being annihilated in an alternative fashion *(odd-even ordering)*. It can easily be verified that an odd-even ordering corresponds to a cyclic by rows or columns ordering, except for a different start-up phase. The 2×2 SVD's that are performed in parallel on the main diagonal can then be thought of as corresponding to different pipelined sequences of $n-1$ rotations, where in each sequence the pivot index successively takes up the values

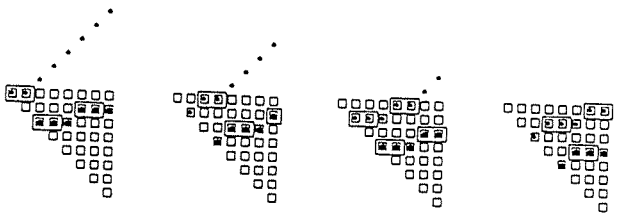


Figure 3: Modified QR-updating array

$i = 1, \ldots, n-1$. As pointed out in section 2, the QR-updatings should be inserted in between these sequences.

A QR-updating is performed by applying a sequence of orthogonal transformations *(Givens rotations)* [6]. The well known Gentleman-Kung QR-updating array [4] can be matched to the SVD-array, as shown in Figure 3. Processor elements on the main diagonal compute and propagate the rotation parameters, while off-diagonal processors apply and propagate rotation parameters and intermediate results. While all operations remain unaltered as compared to the original array, the pipelining is somewhat different, so that the data vectors run through the array in a slightly different manner. The distribution of active processors however now turns out to be the same as in the SVD-array. Framed elements in Figure 3 then constitute the upper parts of the 2×2 blocks in Figure 2.

Without going into details, we give in Figure 4 a snapshot of the overall SVD updating array (see [10] for a detailed description). New data vectors are continuously fed in, to the left-hand side of the array. The matrix-vector products $a^t.V$ are performed in the square part, in between two SVD-rotation fronts. Intermediate result are propagated upwards, while the a-components are propagated to the right. The resulting vectors $a^* = a^t.V$ become available at the top end, and are bounced back into the array, as indicated (■ becomes •). While propagating downwards, the a_*-vector crosses upgoing rotation fronts. Clearly, the corresponding transformations should be applied to the a_*-vector as well (briefly : $a^t.(V.\theta) = (a^t.V).\theta = a_*^t\theta$). Each frame in the square array then corresponds to a (column) transformation, that is applied to a 2×2 block of the V-matrix, *and* to the two available components of the a_*-vector. These components are propagated one step downwards next. The downward propagation of an a_*-vector is thus carried out in exactly the same manner as the propagation in the modified Gentleman-Kung array (Figure 3).

Finally, the triangular array performs the SVD-diagonalization, with interlaced QR-updatings. In each frame, column and row transformations corresponding to the SVD diagonalization are performed first (see also Figure 2), where now column transformations should be applied to the a_*-components as well. In a second step, only row transformations are performed corresponding to the modified QR-updating, affecting the a_*-components and the upper part of the 2×2 -blocks (see also Figure 3).

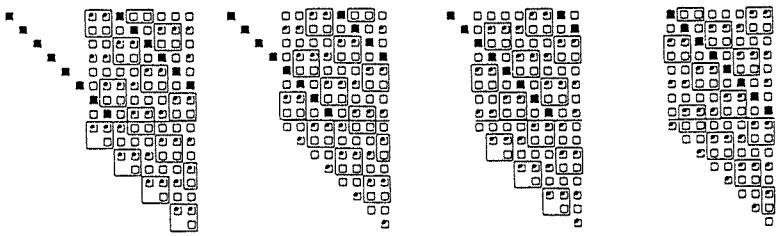


Figure 4: SVD-updating array

5 Conclusion

An SVD updating procedure was constructed as a combination of QR-updating and Kogbetliantz's SVD algorithm applied to the triangular factor. This algorithm can be mapped into a systolic array, making use of slightly modified systolic implementations for each part. The SVD-updating is then performed at a speed comparable to the speed of merely QR-updating. We refer to [9, 10] and [11] for further details, as well as for a performance analysis, square root free algorithms and extensions towards generalized decompositions.

Acknowledgement: M. Moonen is supported by the Belgian National Fund for Scientific Research (N.F.W.O.).

References

[1] BRENT R.P., LUK F.T., 1985. The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays. SIAM J.Sc.Stat.Comp., Vol.6, No 1., pp 69-84.

[2] BUNCH J.R.,NIELSEN C.P. 1978. Updating the singular value decomposition Num.Math.31, pp 111-129.

[3] BUNCH J.R.,NIELSEN C.P.,SORENSEN D.C. 1978. Rank one modification of the symmetric eigenproblem. Num.Math.31, pp 31-48.

[4] GENTLEMAN W.M, KUNG H.T., 1981. *Matrix triangularization by systolic arrays.* Real-Time Signal Processing IV, Proc. SPIE, Vol. 298.

[5] GOLUB G.H. 1973. Some modified eigenvalue problems. SIAM Review, vol. 15, no. 2, pp 318-334.

[6] GILL P.E., GOLUB G.H., MURRAY W., SAUNDERS M.A., 1974. *Methods for modifying matrix factorizations.* Mathematics of Computation, Vol. 28, 126, pp 505-535.

[7] KOGBETLIANTZ E., 1955. Solution of linear equations by diagonalization of coefficient matrices. Quart. Appl. Math. 13, 123-132.

[8] LUK F.T., 1986. A triangular processor array for computing singular values. Lin. Alg. Appl., 77, pp 259-273.

[9] MOONEN M., VANDEWALLE J. 1988. (G)SVD updating for tracking slowly time-varying systems. ESAT K.U.Leuven, Internal report.

[10] MOONEN M., VAN DOOREN P. , VANDEWALLE J. *Parallel updating algorithms, part one : the ordinary singular value decomposition.* Submitted for publication.

[11] MOONEN M., VAN DOOREN P. , VANDEWALLE J. *Parallel updating algorithms, part two : generalized singular value decompositions.* Submitted for publication.

[12] STEWART G.W., 1985. A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. SIAM J. Sc. Stat. Comp., Vol. 6, No. 4, pp 853-863.

RECURSIVE LEAST SQUARES COMPUTATIONS

Robert J. Plemmons

Abstract. We consider parallel implementations of algorithms for recursive least squares computations based upon the information matrix and the covariance matrix updating methods. The target architecture is a shared-memory multiprocessor, and test results on an Alliant system with 8 vector processors demonstrate the parallel efficiencies of the algorithms. The results also show that the covariance method in a form suggested by Pan and Plemmons is easily the most efficient on the Alliant multiprocessor. However, the singular values of the factor matrix decrease in covariance updating, and we give the results of a scheme for adaptively tracking the smallest singular value. Applications include robust regression in statistics and modification of the Hessian matrix in optimization, but the primary motivation for this work is the need for fast algorithms for recursive least squares computations in signal processing.

1. INTRODUCTION

1.1. Background and Applications. The problem considered in this paper is that of designing efficient parallel algorithms for recursive least squares computations based upon updating the information matrix or the covariance matrix factorization. Applications of recursive least squares updating include modification of the Hessian matrix in certain optimization schemes, (e.g., the quasi-Newton method) [5],[13], and estimation methods in adaptive signal processing and control [1-4],[8],[11],[14]. Although many situations in which signal processing is important require special purpose systolic array type devices, our interest in this paper is adaptive signal processing for computations in non real-time applications and on general high performance parallel computers. Interesting non real-time applications of recursive least squares computations in signal processing include adaptive spectral estimation. Applications here include, for example: speech recognition [1], seismology [3], and fetal cardio-analysis [8].

1.2. Least Squares. The *linear least squares problem* can be posed as follows without reference to any specific application:

Given a real $m \times n$ matrix X with full column rank n and a real m-vector s, find the n-vector w that solves

$$\min \|s - Xw\|_2, \tag{1}$$

where $\|\cdot\|_2$ denotes the usual Euclidean norm. The solution to (1) is given by

$$w = (X^T X)^{-1} X^T s. \tag{2}$$