Eigen Decompositions of Covariance Matrices on a Fixed Point DSP

T.J.Herron
Kromos Communications
Fremont, CA 94539, USA.
(tim@kromos.com)

K.M. Reddy, Rahul Garg, Kiran Devanahalli
Ittiam Systems (P) Ltd., Bangalore 560 025, India.
(krishna.reddy@ittiam.com, rahul.garg@ittiam.com,
kiran.devanahalli@ittiam.com)

ABSTRACT

A number of signal processing algorithms can profitably use the eigenvalue decomposition (EVD) of a covariance matrix. Since many of these algorithms are implemented on fixed-point Digital Signal Processors, there is interest in symmetric EVD algorithms that can be implemented in fixed-point arithmetic. Modifications to Jacobi Cyclic algorithms are proposed which make them amenable to fixed-point implementation. We present the algorithms in detail and propose criteria to be used in evaluating the quality of an algorithm's output. The fixed–point algorithm was implemented on a TMS320C6211 fixed point DSP and we present results that show that the EVDs of covariance matrices obtained are comparable to those obtained using Jacobi algorithms performed in high-precision floating point arithmetic.

1.Introduction

Many signal processing algorithms are based on the computation of the EVD of a covariance matrix. Efficient and accurate floating point implementations of algorithms for the EVD of symmetric matrices are well known. However, since fixed-point DSPs are cheaper than their floating-point counterparts, there is a need for reliable EVD algorithms that can be implemented in fixed-point arithmetic. In an earlier paper [8], we presented some modifications to the Jacobi Cyclic Algorithms that facilitated fixed-point implementation along with preliminary simulation results based on a few matrices. Jacobi methods are suitable for fixed-point implementation because they provide more control over scaling of values when compared to most other methods, e.g. QR iteration. In this paper we build on previous work [8] and present further results on fixed-point implementation of the Jacobi algorithms. We begin this paper by giving a brief introduction to the modified Jacobi Algorithms, followed by a description of a new, computationally less intensive, method to calculate the cosine and sine values of the rotation matrices. We have implemented our algorithms on a TMS62X DSP and we give simulation results obtained from 5 classes of randomly generated matrices. Finally we conclude our paper by listing areas for further study including sweep dependent thresholding, use of fast Jacobi rotations and means to deal with matrices with extremely large condition numbers.

2.Modified Jacobi Algorithms

Jacobi EVD methods attempt to zero out the off-diagonal elements of a real symmetric matrix, *A*, by using Givens rotation matrices, *J*, successively on each non-zero off-diagonal matrix element. To zero out an off-diagonal element of A, one has to compute $J^T A J$ (where J^T is the transpose of *J*). It can be shown that with every such rotation, the 'energy' in the off-diagonal elements of A is transferred to the diagonal elements. It can also be shown that eventually A is reduced to a diagonal matrix if off-diagonal elements are zeroed-out in one of several fixed patterns (Cyclic-Row, Brent-Luk, etc.) [1,3,6].

An orthonormal, 2-D, θ rotation of A is of the form,

$$\begin{bmatrix} 1 & . & . & . & 0 \\ . & c & . & -s & . \\ . & . & . & . & . \\ . & s & . & c & . \\ 0 & . & . & . & 1 \end{bmatrix} \times \begin{bmatrix} . & . & . & . & . \\ . & a & . & b & . \\ . & . & . & . & . \\ . & b & . & d & . \\ . & . & . & . & . \end{bmatrix} \times$$

$$\begin{bmatrix} 1 & . & . & . & 0 \\ . & c & . & s & . \\ . & . & . & . & . \\ . & -s & . & c & . \\ 0 & . & . & . & 1 \end{bmatrix} = \begin{bmatrix} . & . & . & . & . \\ . & a' & . & b' & . \\ . & . & . & . & . \\ . & b' & . & d' & . \\ . & . & . & . & . \end{bmatrix}$$

where $c=cos(\theta)$ and $s=sin(\theta)$ are computed to force $b'=0$.

The off-diagonal element, *b'*, and the diagonal elements, *a'* and *d'*, are computed from the matrix equation above by using the following *trace-invariant* formulae:

$$a' = a + \Delta,\ d' = d - \Delta,\quad b' = b - \Lambda \qquad (1)$$

where $\delta = d - a$, $\Delta = s(s\delta - 2cb)$, and $\Lambda = s(c\delta + 2sd)$. Note that we use expanded forms for equation (1) because we will want to apply rotations *J* that permit $b' \neq 0$ [1].

We start by looking for a value of θ, the rotation angle of *J*, that will make $b' = 0$. In fact, we compute $t = \tan\theta$ and then calculate rotation parameters *c* and *s* from *t*. Defining $\sigma := b/(d-a)$, we solve for *t* [1]:

$$t = \frac{2\sigma}{1 + \sqrt{1 + 4\sigma^2}}. \qquad (2)$$

Given t, we can then compute c and s using

$$c = \frac{1}{\sqrt{1 + t^2}}, \qquad s = ct \qquad (3)$$

The computation of both *t* and *c* involve square roots and divisions–two operations to avoid in fixed-point arithmetic. We must keep in mind, however, the following requirements before selecting alternatives to Equations (2) and (3). To able to zero out small off-diagonal matrix elements, *t* must be calculated accurately for small values of θ. Also, to maintain rotation matrices' orthogonality, *c* and *s* must be computed accurately so that $c^2+s^2=1$ holds to high precision. Equation (3) guarantees the latter provided we calculate *c* and *s* accurately from any given *t*.

We provide two methods to calculate *c* and *s*. The first calculates a linear approximation to *t* and then calculates *c* and *s* using the Newton-Raphson method. The other method uses a table of pre-computed values of *c* and *s*, avoiding *t* entirely.

2.1. Computing Trigonometric Approximations

Instead of reducing *b' to* zero, we can reduce it to approximately zero by calculating an approximation to (2), which we call t_a. Because *b'* will acquire energy as other off-diagonal elements have their energies reduced subsequently, allowing $b' \neq 0$ does no harm [3]. We computed a piecewise linear approximation to (2) by calculating good least squares fits - ones easy to calculate - of a line segment to each part of σ that is being approximated.

$$t_a = sign(\sigma) \times \begin{cases} 1, & |\sigma| \geq 2 \\ (12 + 7|\sigma|)/32, & 2 > |\sigma| \geq 1/2 \\ (2 + 23|\sigma|)/32, & 1/2 > |\sigma| \geq 1/4 \\ |\sigma|, & 1/4 > |\sigma| \end{cases} \qquad (4)$$

Note that t_a is accurate for small values of θ because the derivative of the approximation, $\left.\frac{dt_a}{d\sigma}\right|_{\sigma=0} = \left.\frac{dt}{d\sigma}\right|_{\sigma=0} = 1$. We also notice that to compute Equation (4), one can postpone the division in the definition of σ until the division's result is less than 1; an advantage for obtaining precise *t* values. Finally, we state that the maximum value of *q* such that $|b'| = q|b|$ due to rotations computed using (4) is *q=1/4*, and the "average" such *q* compares favorably to previously stated simple approximations of equation (2) [3].

The Newton-Raphson method [2,5] iteratively finds the zeros of a function, provided the derivative of the function is continuous. Since the derivative of *c* in Equation (3) is continuous, we can use this self-correcting method to compute *c* from *t*. Setting $x=1+t^2$ to simplify the notation, note that we can compute $y = 1/\sqrt{x}$ by finding the zero of the function, $f(y)=x-(1/y^2)$. Thus, the Newton-Raphson method yields,

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = \frac{1}{2}(3y_n - xy_n{}^3), \qquad (5)$$

where y_k is the value of *y* at the k^{th} iteration. Since *x* in Equation (3) is bounded as $1 \leq |x| \leq 2$ ($|t| \leq 1$), we can start with y_0 slightly less than one, and 5 iterations of (5) produce values accurate to 29-bits assuming that all operations are done with 32 bit precision. In addition, because $x \in [1,2]$ the largest value of y_k that occurs is 3, and we scale Equation (5) accordingly, to prevent overflow and to maximize accuracy [8].

2.2 Using a Table of Trigonometric Values

We propose a computationally less expensive CORDIC-like method [4] to generate the values of *c* and *s* (given *a*, *d* and *b*) by using pre-computed values. The table provides accurate approximations for small values of *σ*, but is only a very rough approximation for larger values of *σ*. The idea behind this approximation stems from the fact that empirically, the distribution of values of σ which are encountered when diagonalizing a covariance matrix is roughly proportional to $2^{1/\sigma}$ at all but very small *σ*. Thus, a table of *c* and *s* values computed at logarithmically distributed *σ* points should provide an adequate approximation to Equation (3).

The approximation that the table implements when working in 1Q.31 fixed-point arithmetic uses Equation (2) computed for special values; values at the centers of the ranges implicit in Equation (6):

$$t_a = sign(\sigma) \times \begin{cases} t(3\times 2^{n-2}), \; 2^{n-1} \le |\sigma| < 2^n, n \in [-32,2] \\ 1, \quad otherwise \end{cases} \qquad (6)$$

The table itself contains pre-computed values of c and s in fixed-point format calculated from (6) and (3). In fact, the table stores only 36 pairs of *c* and *s* values by assuming that *(b-a)* and *d* are positive; once the value of *s* is determined from the table, we need to correct for the sign of *(b-a)* and *d* by multiplying *s* by *sign(b-a)* and *sign(d)*. By using this table, explicit computations of *σ, t, c* and *s* are avoided. And generating *c* and *s* in this way only requires the DSP to perform a comparison of the most significant non-zero bit positions of *(b-a)* and *d*, to reference to a small table according to (6), and to perform a sign change if one is needed.

3. Fixed-Point Algorithm Implementation

We state some results concerning the accuracy of our fixed-point implementations. The fixed-point implementations are tested using symmetric, positive definite matrices, which are encountered as covariance matrices in Signal Processing applications, rather than the more general symmetric matrices with random entries common in general EVD/SVD literature (e.g. [6]). Some covariance matrices that hold special positions in signal processing applications do have extra structure, such as the Toeplitz covariance matrices found in Wiener filtering. However, our algorithms only use the properties of symmetry and real-valued entries.

3.1 Constructing the Test Covariance Matrices

The square matrices that were used for testing the fixed-point implementations are constructed as follows. First, choose the diagonal elements of a diagonal matrix, *D*, from a uniform random distribution in the range [1, *σ*], where *σ* is the desired conditioning number of the matrix. Of course, 1 and *σ* are also chosen as diagonal elements to ensure that the matrix has the desired conditioning number. Second, the span of the vectors of a random, square matrix, whose entries are chosen from a uniform distribution lying in [-1,1], is orthonormalized; this matrix is used as an orthonormal rotation matrix, *R*. Finally, the test covariance matrix, *C*, is obtained by performing the two-sided Givens rotation: $C=R^TDR$.

3.2 Performance Criteria

We first define the measures we use for quantifying the performance of the fixed-point algorithms. For the eigenvalues, we use the absolute percentage error, *e*, defined as:

$$e = \left| \frac{e_{fxp} - e_r}{e_r} \right| \times 100 \qquad (7)$$

where e_{fxp} is the eigenvalue obtained from the fixed-point algorithm and e_r is the reference eigenvalue that is known to be accurate. For each matrix, we record the maximum relative error and denote it as e_{max}; we also consider the mean of all the relative errors of a matrix, which we denote e_{avg}.

The correspondence between the reference eigenvectors and fixed-point algorithm eigenvectors is checked by first forming a projection matrix, *A(u),*

$$A(u_k) = I - \frac{u_k^T u_k}{u_k u_k^T}, \qquad (8)$$

where u_k is the kth column vector of the reference eigenvector matrix *U* and *I* is the identity matrix. We then project the eigenvector matrix *Q*, obtained from the fixed-point algorithm, onto the hyperplane $A(u_k): B_k = A(u)Q$. Next, we form the vector r_k where $r_k = |B_k|$ and $|\bullet|$ is the vector norm operator applied along the columns of a matrix. If there were no numerical inaccuracies in computing *Q*, each vector r_k would be of the form: $r_k = [1,1,...,1,0,1,...,1,1]$. Finally, we collate the vectors r_k, $k = 1...N$, and place them in an *eigenvector similarity matrix*, denoted by S, where

$$S = \left[r_1^T \quad r_2^T \quad ... \quad r_k^T \right]^T \qquad (9)$$

Ideally, *S* should be of the form *(1-I),* where *1* is a matrix of all ones. This is true provided that we assume that the reference EVD and the EVD obtained from the fixed point implementation are both sorted so that the eigenvalues (and corresponding eigenvectors) are sorted in descending order. We summarize the information available in S by indicating the maximum deviation from zero, D_0, of the diagonal elements of S and the maximum deviation from unity, D_1, of the off-diagonal elements of S.

Finally, we would like to know how well the product of the eigenvector matrix and its transpose match with the identity matrix (where this multiplication is done in fixed-point arithmetic). To quantify this error, we define the quantity F:

$$F := \left|Q^T Q - I\right|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(Q^T Q - I\right)_{i,j}^2}\ , \qquad (10)$$

where $|\bullet|_F$ is Frobenius-norm operator. The closer F is to zero, the more orthogonal are the matrix eigenvectors to one another; a property we call *channel independence*, one that is often more important than is the accuracy of the eigenvectors to the ideal eigenvectors.

3.3 Simulation Results

We give below the results obtained with symmetric 24x24 matrices generated as mentioned earlier. Five classes of matrices were generated, denoted C2, C3, C4, C5 and C6 with respective conditioning numbers 1e2, 1e3, 1e4, 1e5 and 1e6; each class contained 100 matrices. We decomposed the matrices using 12 row sweeps of the fixed-point diagonalization algorithm and compared them with reference EVDs obtained from 20 sweeps of the Jacobi Cyclic Row algorithm implemented in floating point arithmetic. A sweep of the Jacobi Row algorithm is defined by attempting to zero-out each off-diagonal element by moving down the matrix rows in order [6]. The first table below gives the mean results obtained for each class of matrix obtained using the piecewise linear approximation method, while the second table lists the results for the pre-computed table is used.

The algorithm was implemented on a TMS320C6211 fixed point DSP using Q1.31 arithmetic. The code was a combination of compiled C code and hand written assembly routines. In particular, all the arithmetic operations were written in assembly while the 'control' code was written in C. The program occupied approximately 9.6 KB of memory space. It was found that approximately 30,000 processor cycles was needed for one complete sweep of the Jacobi algorithm on a 24X24 matrix; this translated to about 0.2 ms/sweep on a 150 MHz TMS6211 DSP.

We see that the results obtained from the two methods are comparable. It is also apparent that the accuracy decreases when the condition number of the matrix increases. In addition, the small computed values of F show that the product of the eigenvector matrix and its transpose is very close to an identity matrix, indicating that the algorithm produces a rather pure KLT. This fact is very important because an orthogonal eigenvector matrix indicates that the data channels produced when the KLT is *applied* to data will be distinct and orthogonal, even if the eigenvalues do not exactly match those of the ideal EVD. For example, when a KLT is used for data compression, small eigenvalues's eigenvectors are directions which contain little non-redundant data. And since the largest eigenvalue errors are in fact in the smaller (usually the smallest) eigenvalue(s), these algorithms can often be used with matrices having condition numbers of 1e6 or larger.

Finally, we tested both algorithms on several matrices that have near-multiplicities in several eigenvalues. By studying the resulting S matrices, we found that our EVD algorithms only fail to separate the nearly-repeated eigenvalues, while performing well in completing the rest of the EVD.

4. Extensions and Scope for Further Work

The accuracy of the algorithms can be improved by using *sweep-dependent thresholding* [6], in which off-diagonal elements are skipped if their magnitude is less than some threshold value. Thresholding reduces the number of arithmetic operations performed, leading to a decrease in the errors caused by the use of finite precision arithmetic. One thresholding rule that was tried [6] and that did result in modest improvement was of the form 2^{-zn}, $z \in [4,5,6,7]$ and n is the sweep number. More investigation is needed to determine an optimal thresholding rule, though studies indicate that a rule of the form $2^{-(2n+z)}$ performs better if the matrix to be diagonalized is initially scaled to have trace 1.

Although our Jacobi implementations are reasonably fast, they can be improved by using *fast Jacobi rotations* [3,7]. Fast Jacobi rotations use simple factorizations to reduce the number of matrix multiplications when applying Jacobi rotations. This leads to a large saving in computation for even moderately large matrices, with some occasional bit-shifting as the only additional work required. Plus, the computational reductions due to Fast Jacobi rotations give better numerical precision.

In cases where the conditioning number of the covariance matrix exceeds the ability of a fixed-point DSP to compute an adequate EVD as above, we have to resort to more sophisticated algorithms. One straightforward extension to the algorithms is to first compute the Cholesky decomposition of the covariance matrix (which has a conditioning number that is the square root of the covariance matrix's conditioning number). Then, use either Hestenes's method [6] or triangular Kogbetliantz's method [3,6] on the triangular Cholesky matrix to compute an SVD: square the resulting singular values to obtain the original covariance matrix's eigenvalues. However, both methods require more computation; Hestenes's method can take quite a long time to converge; and Kogbetliantz's method requires more time to compute the Givens rotations, some of whose formulas cannot be approximated (rotations can be *fast* and *squared*, however) [3,6]. Because of this and the overhead in computing the Cholesky decomposition (plus its scaling challenges), it remains to be seen which additional classes of covariance matrices can be accurately decomposed in fixed-point arithmetic.

Matrix Class	σ	$e_{\max}$ (%)	e_{avg} (%)	D_0	D_1	F
C2	1e2	3.4 e-3	2.3e-4	5.4e-7	6.5e-5	5.3e-7
C3	1e3	3.2e-2	1.4e-3	5.3e-7	8.0e-5	5.2e-7
C4	1e4	3.2e-1	1.3e-2	5.3e-7	1.0e-4	5.2e-7
C5	1e5	3.3	1.3e-1	5.3e-7	1.3e-4	5.3e-7
C6	1e6	3.4e1	1.4	5.3e-7	7.3e-5	5.2e-7

Table 1 Results obtained using the linear piecewise least-squares approximation

Matrix Class	σ	$e_{\max}$ (%)	e_{avg} (%)	D_0	D_1	F
C2	1e2	3.5 e-3	2.5e-4	5.8e-7	5.0e-4	6.4e-7
C3	1e3	3.5e-2	1.6e-3	5.8e-7	5.3e-4	6.4e-7
C4	1e4	3.7e-1	1.5e-2	5.8e-7	6.7e-4	6.3e-7
C5	1e5	3.6	1.5e-1	5.7e-7	5.8e-4	6.3e-7
C6	1e6	3.7e1	1.7	5.8e-7	4.9e-4	6.3e-7

Table 2 Results obtained using the pre-computed table

5. Conclusion

We have proposed modifications to Jacobi cyclic EVD algorithms that are amenable to a fixed-point implementation. We have shown, for many symmetric matrices, that the EVDs obtained from such fixed-point implementations are usable.

6. References

[1] Golub G.H, van Loan C.F, *Matrix Computations*, 3 ed., John Hopkins University Press, Baltimore 1996.

[2] Golub G.H. and Mitchell I., "Matrix Factorizations in Fixed Point on the C6x VLIW Architecture", Stanford Univ., 1998. (http://wwwccm.stanford.edu/Students/mitchell/)

[3] Götze J., "On the Parallel Implementation of Jacobi and Kogbetliantz Algorithms", SIAM J. Scientific Computing, Vol. 15, No. 6, pp. 1331--1348, November 1994.

[4] Götze, J., Hekstra, G.J, "An Algorithm and Architecture Based on Orthonormal μ-Rotations for computing the Symmetric EVD", Integration – The VLSI Journal, Special Issue on Algorithms and Parallel VLSI Architecture, pp. 21-29, 1995.

[5] Press W.H., et al., *Numerical Recipes in C*, 2 ed., Cambridge University Press, 1992.

[6] Rands Jensen P.M., "On Jacobi-Like Algorithms for Computing the Ordinary Singular Value Decomposition", Tech. Report, R.91-41, Dept. of Communication Technology, Institute of Electronic Systems, Aalborg University, Denmark, 1991.

[7] Rath, W., "Fast Givens Rotations for Orthogonal Similarity Transformations", Numerische Math., v. 40, pp. 47-56, 1982.

[8] Reddy, K.M. and Herron, T.J., "Computing the Eigen Decomposition of a Symmetric Matrix in Fixed-Point Arithmetic", 10th Annual Symposium on Multimedia Communications and Signal Processing, 2001, Bangalore, India.