Computing the Eigen Decomposition of a Symmetric Matrix in Fixed-point Arithmetic

K.M. Reddy
Ittiam Systems,
Bangalore 560 025, India.

T.J. Herron
Kromos Technology
Los Altos, CA 94022, USA.

ABSTRACT

Since a number of signal processing algorithms requiring the EVD of a symmetric matrix are implemented on fixed-point DSPs, there is considerable interest in matrix diagonalization algorithms that can be implemented in fixed-point arithmetic. We propose modifications to the Jacobi Cyclic Row algorithm that favor a fixed-point implementation. We then compare the eigenvalues and eigenvectors obtained from the Jacobi Algorithm implemented in fixed-point arithmetic to ones obtained from a floating point algorithm. For the matrices considered, we find that the EVD obtained from the fixed-point implementation matches closely a floating-point implementation of the EVD decomposition. We also give an estimate of computational complexity of the fixed point algorithm.

1.Introduction

A number of Signal Processing algorithms require the Eigen Value Decomposition (EVD) of a symmetric matrix to be computed – in particular those which utilize the Karhunen-Loeve Transformation (KLT). E.g., the MUSIC algorithm detects the frequencies in a signal by performing an EVD on the covariance matrix obtained from the samples of the received signal; in Principal Component Analysis (PCA), a popular technique used in image processing for feature extraction, data is projected along the directions of the principal eigenvectors of the input covariance matrix.

Well-known EVD algorithms like the Jacobi Cyclic Row and the QR algorithm have been extensively studied, and efficient and accurate floating-point implementations exist. There is considerable interest in making these floating-point implementations amenable to fixed-point implementation as fixed-point DSP processors are more economical than their floating-point counterparts, and a considerable number of DSP implementations that could take advantage of diagonalization are implemented on fixed-point processors. In this paper, we propose modifications to the Jacobi Cyclic row algorithm that are conducive to a fixed- point implementation.

The exact Jacobi Algorithm [1] involves the calculation of a square root, the calculation of a reciprocal of a square root, and multiple divisions; these are difficult to implement efficiently in fixed-point arithmetic with the desired precision. We implement instead approximate Jacobi rotations in which the square root computation is replaced by a simple linear piecewise approximation, the reciprocal of the square root is computed using the Newton Raphson method, and most divisions are eliminated.

We present results that compare the eigenvectors and eigenvalues obtained from our fixed-point implementation to those obtained from a floating-point realization. We show, for the test matrices considered, that the eigenvalues and eigenvectors obtained from the fixed-point algorithm are quite close to the eigenvalues obtained from the floating-point algorithm. We also show that the product of the eigenvector matrix (obtained from the fixed-point algorithm) and its transpose is very close to an identity matrix, indicating that the algorithm produces a nearly 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.

2.Cyclic Jacobi Method

In this section, we briefly describe the Jacobi Cyclic Row algorithm [1,8]. The Jacobi Cyclic Row algorithm attempts to reduce the $n \times n$ symmetric matrix A to diagonal form by applying orthonormal rotations to left and right of A. The orthonormal rotation used, $J(p,q,\theta)$, which refers to the rotation constructed to reduce the element A_{pq} to zero, is of the form in the following diagram. The form consists of an identity matrix with the zeros and ones in the pth & qth row and column replaced by the sine and cosine of an angle θ, which we denote by s and c respectively.

$$\begin{array}{cccccccc}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \\
| & & | & & | & & | & \\
0 & \cdots & c & \cdots & s & \cdots & 0 & p \\
| & & | & & | & & | & \\
0 & \cdots & -s & \cdots & c & \cdots & 0 & q \\
| & & | & & | & & | & \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & \\
 & & p & & q & & &
\end{array}$$

Successive steps in a Jacobi decomposition procedure involve choosing an index pair (p,q) that satisfies $1 \le p < q < n$ and finding a value of theta that reduces each such B_{pq} to zero via the following transformation:

$$B = J(p,q,\theta)^T \times A \times J(p,q,\theta) \qquad (1)$$

One can see that only the p^{th}, q^{th} row and columns are affected by the transformation. By multiplying out the transformation and using the fact that the trace is invariant under the transform, we obtain the following expressions for elements of B in the p^{th}, q^{th} rows and columns.

$$B_{rp} = cA_{rp} - sA_{rq}, \quad r \ne p, r \ne q \qquad (2)$$

$$B_{rq} = cA_{rp} + sA_{rq}, \quad r \ne p, r \ne q \qquad (3)$$

$$B_{pp} = A_{pp} + \Delta \qquad (4)$$

$$B_{qq} = A_{qq} - \Delta \qquad (5)$$

$$B_{pq} = A_{pq} - \Lambda \qquad (6)$$

where,

$\delta = A_{qq} - A_{pp}, \Delta = s(s\delta - 2cA_{pq})$ &

$\Lambda = s(c\delta + 2sA_{pq})$.

(We have also assumed A to be symmetric.)

When implementing the Jacobi rotation (1), the value of B_{pq} should be zero and, thus, simplified equations for B_{pp} and B_{qq}, based on t, are ordinarily used [1]. However, since we will be computing an approximate Jacobi rotation in the sequel, the value of B_{pq} will not in general be exactly zero and we must use Equations (4,5,6). This does add about 6 extra flops to the 4n flops that are normally required to perform the Givens rotation.

To solve for the correct Jacobi rotation J to use in (1), we start by defining $\tau = \dfrac{A_{qq} - A_{pp}}{2A_{pq}}$ and $t = \tan\theta = \dfrac{s}{c}$. Then, using these to solve $B_{pq} = 0$ from (6), we obtain $t^2 + 2\tau t - 1 = 0$. Solving this quadratic equation, we get two values of t that make B_{pq} zero, $t = -\tau \pm \sqrt{1+\tau^2}$. It can be shown that picking the smaller value of t gives a more stable reduction. Finally, s and c can be calculated from t, the tangent, by using the following relations:

$$c = \frac{1}{\sqrt{1+t^2}} \quad \text{and} \quad s = t \times c \qquad (7)$$

In the Jacobi Cyclic Row algorithm, we zero-out the off diagonal elements of A row-wise; i.e. we select the index pairs *(p,q)* in the order: (1,2)(1,3)…(1,n), (2,3)…(2,n)…(*n-1*,*n*). The application of (1) for all $N = n \times (n-1)/2$ index pairs is called a *sweep*. The matrix *A* eventually reduces to a diagonal matrix – producing *A*'s eigenvalues - via successive sweeps because it can be shown that with every Jacobi rotation (1), the offdiagonal norm of the matrix *A*, which is defined as $\sqrt{\sum_{i=1}^{n} \sum_{j=1, j\ne i}^{n} |A(i,j)|}$, is reduced. There are alternative sweep orderings which have the same kind of convergence to a diagonal form [9], but the row ordering outlined above is reliable, and allows for some simple parallelism if a DSP has multiple MACs (used on multiple, distinct *(p,q)* indices).

Finally, note that because the matrix remains symmetric after each orthonormal rotation, it is sufficient to work with the upper triangular part of the matrix throughout the algorithm, saving some memory. However, the entire matrix of cumulative rotations performed on *A* must be kept track of because the columns of that matrix are the eigenvectors associated with *A*.

3.Computing Modified Jacobi Rotations

In order to compute *s* and *c* for each *J* using Equation (7), we first need to compute *t*. Defining $\sigma = 1/(2\tau)$, we find that the solution to the equation $B_{pq} = 0$ is:

$$t_{exact} = \frac{2\sigma}{1+\sqrt{1+4\sigma^2}}. \qquad (8)$$

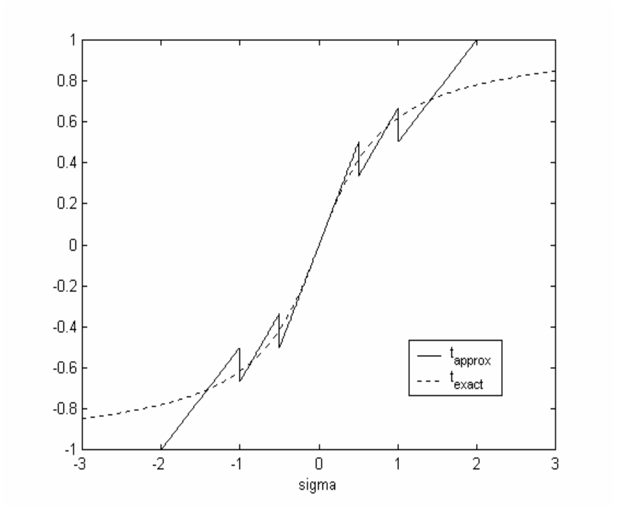
From the fixed point implementation point of view, computing the 2 divisions and the square root in (8) is discouraging. Thus, instead of performing an exact Jacobi rotation, given by (8), we want an easy to compute approximation, t_{approx} to the tangent function, t [3]. Instead of constructing the Jacobi rotation matrices *J* to obey $B_{pq} = 0$ exactly, it is sufficient to compute the sine-cosine pair such that the orthogonality of *J* is preserved and $|B_{pq}| = |d||A_{pq}|, \quad 0 \le |d| < 1$. The off-diagonal norm does reduce with each such rotation (1), the more so the closer to 0 |*d*| is [3]. Such a *J* is known as an *approximate Jacobi rotation*.

One good approximation, t_{approx}, to use is [3]:

$$t_{approx} = \begin{cases} sign(\sigma), & |\sigma| \ge 2 \\ \sigma / 2, & 2 > |\sigma| \ge 1 \\ \dfrac{2\sigma}{3}, & 1 > |\sigma| \ge \dfrac{1}{2} \\ \sigma, & \dfrac{1}{2} > |\sigma| \end{cases} \tag{9}$$

It turns out that an approximation to t need not be accurate when is σ is large, but must be accurate for smaller values of σ; in particular the derivative of the approximation ,

$\dfrac{dt_{approx}}{d\sigma} = 1$ at $\sigma = 0$, as is (9).



Note that the approximation (9) involves only a few compares and linear scales of σ, *and* we can postpone division until we are know that the value of σ is less than 1. Now that we have calculated an approximation to *t*, we must calculate the value of the sine and cosine. In order to preserve the orthogonality of the rotation matrix *J*, we must ensure that $c^2 + s^2 = 1$. From Equation (7) $c = \dfrac{1}{\sqrt{1+t^2}}$; however, this is not amenable to fixed-point arithmetic. But we are helped by the fact that $|t| \le 1$. We use the Newton-Raphson method [6] to recursively compute *c* by approximating the equation $y = 1/\sqrt{x}$ via looking for the closest zero of the function: $f(y) = x - \dfrac{1}{y^2}$. Thus, we can derive that

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = \frac{1}{2}(3y_n - xy_n^3), \tag{10}$$

where y_k is the value of y at the *kth* iteration. Since $|t| \le 1, \quad x \in [1,2]$. Therefore, we can start with y_0 slightly less than unity, and 4 or 5 iterations produces values for *c* in excess of 30-bits of accuracy, assuming that all operations are done with 32 bit precision.[1] In addition, given that $x \in [1,2]$, the maximum value that appears while computing y_k is 3. The rest of the computations involved in the Jacobi method involve matrix multiplications, which can be directly implemented in fixed-point arithmetic without any modification.

4.Results

The modified Jacobi Algorithm was implemented in Q1.31 arithmetic. We compare the eigenvalues and eigenvectors obtained from our fixed-point algorithm to the ones obtained from a floating-point implementation. The error in the eigenvalues is quantified using the maximum percentage relative error, $e_{\max}$, which is defined as the maximum percentage relative error, *e*, obtained for a given matrix where

$$e = \left| \frac{e_{fxp} - e_{fp}}{e_{fp}} \right| \times 100, \tag{11}$$

e_{fxp} is the eigenvalue obtained from the fixed-point algorithm and e_{fp} is the (reference) eigenvalue obtained from the exact Jacobi floating point implementation. We also define the average relative error, e_{avg}, as the mean percentage relative error (11) for a given matrix.

To find the correspondence between fixed-point and floating-point eigenvectors is more involved. We form a projection matrix, *P(u)*, by defining

$$P(u) = I - \frac{u^T u}{uu^T}, \tag{12}$$

where *u* is a column vector of the eigenvector matrix U_{fp} that is obtained from the floating point Jacobi algorithm. We then project the eigenvector matrix U_{fxp},

[1] In fact in [2], this Newton-Raphson method has been used to help compute matrix factorizations using just 16-bit fixed-point arithmetic.

obtained from the fixed-point algorithm, onto the hyperplane $P(u)$: $Q = P(u)U_{fxp}$. We form the (row) vector r where $r = |Q|$, where $|\;|$ is the vector norm operator applied along the columns. If there were no numerical inaccuracies due to fixed point arithmetic, the vector r should be a co-ordinate vector, i.e. only one element should be one and the rest of its elements being zero. If we carry out the above operations for each of the column vectors of U_{fp} and arrange the corresponding r's along the columns of a matrix, *R*, then we can get an estimate of how well the eigenvectors of *Q* approximate the true eigenvectors. Ideally these matrices should be of the form $(1 - I)$, where 1 is a matrix of all ones and I is the identity matrix.[2] We 'compress' the information available in *R* by indicating the maximum deviation from zero and the maximum deviation from unity.

We would also like to know how well the product of the eigenvector matrix and its transpose match with the identity matrix. (This multiplication is done in fixed-point arithmetic.) To quantify this error, the identity matrix is subtracted from the product of the eigenvector matrix and the transpose of the eigenvector matrix; the Frobenius norm of this difference (*F*) is a measure of the error in the orthogonality of the eigenvector matrix. Symbolically,

$$F := \left|Q^T Q - I\right|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(Q^T Q - I\right)_{i,j}^{2}}\,, \qquad (13)$$

where $|\;|_F$ is Frobenius-norm operator.

We show the results for 6 typical covariance matrices (denoted CV1-CV6) of varying conditioning numbers η, which is the ratio of the maximum eigenvalue of *A* to the minimum eigenvalue of *A*. CV1-CV5 are matrices of size 12X12, while CV6 is a 20X20 matrix. 6 sweeps were used for the fixed-point routine for matrices CV1-CV5, while 8 sweeps were used for CV6. 20 sweeps were used for the double precision floating point routine to obtain the reference eigenvalues and eigenvectors.

Since we are dealing with fixed point arithmetic, we must ensure that the matrix to be diagonalized is so scaled such that saturation or underflow does not affect the accuracy of the algorithm. It was observed that for maximum accuracy of the eigenvalues and eigenvectors, the matrix should be scaled such that the ratio of its Frobenius-norm to the number of rows (or columns, as the matrix is square) must lie in the range 0.04 to 0.1. Scaling a matrix such that this ratio is approximately 0.05 is probably a safe bet. All the results given below are with matrices scaled in this fashion.

Matrix	η	$e_{\max}$ (%)	e_{avg} (%)	Max. Deviation from Zero	Max. Deviation from Zero
CV1	2.2	2.1e-5	1.3e-5	6.2e-5	4.9e-8
CV2	63	6.1e-4	2.5e-4	1.2e-4	5.7e-8
CV3	93	4.6e-4	2.2e-4	9.1e-5	4.5e-8
CV4	128	5.3e-4	2.4e-4	7.7e-5	3.7e-8
CV5	1066	3.9e-4	3.4e-4	1.2e-4	3.9e-8
CV6	1.6e5	2.7	0.59	8.1e-2	2.7e-3

(It was seen that $F \le 10^{-6}$ in all the cases considered above.)

It is seen that for matrices with smaller condition numbers, the EVD obtained from the fixed point algorithm matches closely with the one obtained form the floating point implementation. However the algorithm has difficulty with matrices with large condition numbers.

5.Computational Complexity

We now estimate the computational complexity of our Jacobi fixed point implementation when it is applied to a symmetric *n X n* matrix *A*. Since a MAC (Multiply and Accumulate) is an operation intrinsic to most DSPs, we give the computational complexity estimates in terms of MACs. We assume that a MAC is given by: a = a + b * c, where a, b, c are 32 bit registers or 32 bit memory locations. We also assume that the addition of two 32 bit operations results in a 32 bit number (saturated if necessary), and that during the multiplication of the two 32 bit numbers, only the first 32 bits are retained. The estimate of MACS required for each symmetric rotation is given below:

Computation of Approximate Givens Matrix:	70 MACs
Multiplication of *A* by Givens matrices:	$4n + 6$ MACs
Update of Eigenvector matrix:	$4n$ MACs

Since a Jacobi sweep involves $\frac{n \times (n-1)}{2}$ Givens rotations, we find that we need to perform about $n \times (n-1) \times (4n + 35)$ MACs every sweep.

[2] For convenience, we assume that the eigenvalues are sorted in descending order with the eigenvector matrices also being sorted correspondingly, so that the first column of the eigenvector matrix contains the eigenvector corresponding to the largest eigenvalue, etc.

6.Conclusion

Modifications to the Jacobi Cyclic Row algorithm were described which made it possible for the algorithm to be implemented in fixed point arithmetic. The EVDs of a few covariance matrices obtained with the fixed point algorithm were compared with an accurate floating point reference and were found to match closely for matrices with reasonable conditioning numbers.

7.References

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

[2] Golub G.H., Mitchell I., Matrix Factorizations in Fixed Point on the C6x VLIW Architecture, Stanford, 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., Monitoring the Stage of Diagonalization in Jacobi Type Methods. In Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, pages III.41-III.444, Adelaide (Australia), 1994.

[5] Haykin S., *Adaptive Filter Theory*, 3 ed., Prentice Hall, New, York, 1996.

[6] Kreyszig E., *Advanced Engineering Mathematics*, 8 ed., John Wiley & Sons, New York, 1999.

[7] Moonen, M., and de Moor, B., *SVD and Signal Processing III: Algorithms, Applications and Architectures*, Elsevier Science Publishers B. V. (North Holland), Amsterdam, 1995.

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

[9] Rands Jensen P.M., On Jacobi-Like Algorithms for Computing the Ordinary Singular Value Decomposition, Tech. Report, R.91-41, Dept. of Communication Tech., Institute of Electronic Systems, Aalborg University, Denmark, 1991 (http://citeseer.nj.nec.com/186835.html).