Storage and Computationally Efficient Permutations of Factorized Covariance and Square-Root Information Arrays

R. J. Muellerschoen
Navigation Systems Section

A unified method to permute vector-stored Upper-triangular-Diagonal factorized covariance (UD) and vector-stored upper-triangular Square-Root Information (SRI) arrays is presented. The method involves cyclic permutation of the rows and columns of the arrays and retriangularization with fast (slow) Givens rotations (reflections). Minimal computation is performed, and a one-dimensional scratch array is required. To make the method efficient for large arrays on a virtual memory machine, computations are arranged so as to avoid expensive paging faults. This method is potentially important for processing large volumes of radio metric data in the DSN.

I. Introduction

In the reduction of observational data involving simultaneous least-squares estimation of many parameters, it is desirable to examine different modeling scenarios without reprocessing the data through the filter. Often the analyst will estimate a full contingent of system parameters in the initial filter pass. A reduced state estimate, reflecting a different modeling scenario, can then be obtained from this initial pass. Furthermore, the sensitivity of the reduced state to the excluded parameters is readily obtainable. Application of a priori information for the excluded parameters to the sensitivity and subsequent augmentation of the reduced state by these perturbations then yields a more conservative assessment of the reduced state errors. This type of analysis is particularly useful when the reduced state is sensitive to the excluded parameters that are insensitive to data [1].

Upper-triangular data structures, such as those encountered in an Upper-triangular-Diagonal factorized covariance filter (UD filter) or in an upper-triangular Square-Root Information filter (SRI filter), are conducive to computing these reduced state estimates. In a UD filter, it is necessary to permute the rows of the UD array in order to compute the reduced estimate. The rows corresponding to the parameters that are to be excluded must be moved below those corresponding to the parameters that are to be included in the reduced estimate. In an SRI filter, it is necessary to permute the columns of the SRI array in order to compute the reduced estimate. The

columns corresponding to the parameters that are to be excluded must be moved to the right of the columns corresponding to those that are to be included in the reduced estimate.[1]

A brute force method to accomplish these permutations reorders the rows (columns) of the UD (SRI) array into a two-dimensional work array. To retriangularize this two-dimensional array, one can then post(pre)-multiply it by an implicitly defined orthogonal transformation composed of Householder reflections.[2] This method is sound and numerically stable due to the orthogonality of the transformation. Furthermore, the commercially available Estimation Subroutine Library (ESL) has standard routines to conveniently perform these permutations and subsequent retriangularizations [2]. However, this method requires a two-dimensional work array to store the permuted UD (SRI) array. For large-scale systems, such as those encountered for Global Positioning System (GPS) studies, it is not at all uncommon to estimate on the order of 1000 parameters. This translates into four million extra bytes of storage. Additionally, over 3000 parameters are routinely solved when DSN VLBI data are processed with MASTERFIT software [3]. Moreover, for systems of this order, retriangularization is computationally expensive.

In 1986, Bierman (personal communication) suggested that the necessary reordering of UD arrays be performed as a series of pairwise permutations in such a way that the upper-triangular structure of the UD array is always maintained. Each pairwise permutation could be performed as a suboptimal noise-free measurement update:

$$P_{\text{update}} = (I - GH)\,P(I - GH)^t + RGG^t$$

where P_{update} is then the permuted covariance of the covariance P. The noise R is set equal to zero, and the gain G and design H are chosen so that $(I - GH)$ is a permutation operator. Such a permutation operator that permutes parameters i and j can be expressed as

$$I - GH = I - (e_i - e_j)(e_i - e_j)^t$$

where e_i is a column vector whose elements are zero except element i, which is 1. Trivially $H = (e_i - e_j)^t$ and $G = H^t$. In practice this computation is implemented with an optimal measurement update using the Bierman UD measurement update algorithm and a rank-1 update to include the effect of the suboptimality of the gain. This method eliminated the need for a two-dimensional scratch array. Software was eventually written that exploited the structure of the suboptimal gain G and design H. This method still required several scratch arrays and a multitude of pairwise permutations.[3]

In 1987, Wolff (personal communication) suggested that the same pairwise permutations could be performed on an SRI array with retriangularization accomplished after each column exchange with slow Givens reflections.[4] This method also eliminated the need for a two-dimensional work array. However, in the retriangularization process, elements not consistent with the upper-triangular data structure of the SRI array were created. Each inconsistent element required another slow Givens reflection. Subsequently, this method required on the order of n^2 slow Givens reflections, where n is the column distance between the two parameters of the exchange. Furthermore, this method required two explicit scratch arrays. One of the scratch arrays stored the right column of the exchange while the other was used to store elements that were created in the retriangularization process.[5]

Also in 1987, Pombra suggested a technique that eliminated the need to perform a multitude of these pairwise permutations in reordering a UD array.[6] His technique was to insert into the covariance an artificial parameter with zero variance and zero correlation with the other parameters. This is easily accomplished by inserting into the UD array a row and column of zeros. A parameter could then be moved directly into its proper place by performing a pairwise permutation of this artificial parameter and the parameter that is desired to reside in its place. After the permutation is performed, this artificial parameter can then be deleted from the UD array. This is also easily accomplished by removing from the UD array the appropriate row and column of zeros. This technique drastically

[1] S. C. Wu et al., *Oasis Mathematical Description*, JPL Publication D-3139 (internal document), Jet Propulsion Laboratory, Pasadena, California, April 1, 1986.

[2] Before this retriangularization can be performed on the UD array, it is of course necessary to scale the columns by the square root of the corresponding diagonal. These square roots can be avoided by performing a modified weighted Gram–Schmidt UD triangularization algorithm.

[3] P. J. Wolff, "UD Permutations via ESL Subroutine UCON," JPL IOM 314.5–1040 (internal document), Jet Propulsion Laboratory, Pasadena, California, August 29, 1986.

[4] Since the SRI array is not unique, either Givens reflections or rotations can be used.

[5] P. J. Wolff, "Permuting R Matrices in Place," IOM 314.5–1091 (internal document), Jet Propulsion Laboratory, Pasadena, California, March 10, 1987.

[6] S. A. Pombra, "Computationally Fast Version of UDEDIT," IOM 335.1-87–160 (internal document), Jet Propulsion Laboratory, Pasadena, California, July 7, 1987.

reduced the number of permutations necessary to reorder the array. However, suboptimal noise-free measurement updates were still used to perform the pairwise permutations.

It was recognized that the insertion of a row and column of zeros into the UD array, together with the subsequent row exchange between the artificial parameter and the parameter of interest, created an array similar to a Morf–Kailath prearray with zero noise variance [4]. Retriangularization of this array into an upper-triangular Morf–Kailath postarray is easily accomplished with fast Givens rotations.[7] This method requires a one-dimensional scratch vector which manifests itself as an augmentation of the UD array. An extra column of the UD array permits the insertion of a row and column of zeros. Furthermore, the method requires on the order of $\tilde{n}_{\text{row}}$ fast Givens rotations, where $\tilde{n}_{\text{row}}$ is the row distance between the parameter and the desired location of the parameter. Due to the nature of the fast Givens, square root computations are never required.

This technique can also be adopted to reordering SRI arrays. In a manner similar to the Wolff method, retriangularization is accomplished with slow Givens reflections. Unlike the Wolff method, however, no new elements are created. Hence, this method requires on the order of only $\tilde{n}_{\text{column}}$ slow Givens reflections, where $\tilde{n}_{\text{column}}$ is the column distance between the parameter and the desired location of the parameter.

Since the artificial parameter is only a convenient place holder, an alternative method could be constructed that does not incorporate the insertion/deletion step. Similar results can be achieved by cyclically permuting the rows and columns of the arrays. Retriangularization is then achieved as before. Furthermore, since the size of the array does not change, it is not necessary to augment the array with an extra column. However, a one-dimensional scratch vector is required.

Cyclic column permutations and retriangularization of SRI arrays have already appeared in the literature [5]. In fact, the discussed method to cyclically permute columns of an SRI array is similar to the left circular shift method used in the LINPACK routine SCHEX [6].[8] Until now, however, these results have not been extended to cyclic row permutations and retriangularization of UD arrays. The permutations are performed in such a way that the only difference between permuting UD and SRI arrays lies in the retriangularization process.

[7] Since the UD array is unique, fast Givens reflections cannot be used.

[8] SCHEX requires two one-dimensional scratch arrays to store the transformation pairs and does not store the upper-triangular array as a vector.

II. Row Permutations of UD Arrays

To demonstrate the technique on UD arrays, let the factorization of a covariance P with parameters $U1, U2, U3, U4, U5$ be

$$P = UDU^t \tag{1}$$

where

$$U = \begin{bmatrix} u1_1 & u1_2 & u1_3 & u1_4 & u1_5 \\ 0 & u2_2 & u2_3 & u2_4 & u2_5 \\ 0 & 0 & u3_3 & u3_4 & u3_5 \\ 0 & 0 & 0 & u4_4 & u4_5 \\ 0 & 0 & 0 & 0 & u5_5 \end{bmatrix} \tag{2}$$

and

$$D = \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \tag{3}$$

If we designate a weighted inner product as

$$x \cdot y \equiv \sum_i x_i d_i y_i \tag{4}$$

then the covariance can be written in terms of its factorized elements:

$$P = \begin{bmatrix} u1 \cdot u1 & u1 \cdot u2 & u1 \cdot u3 & u1 \cdot u4 & u1 \cdot u5 \\ u2 \cdot u1 & u2 \cdot u2 & u2 \cdot u3 & u2 \cdot u4 & u2 \cdot u5 \\ u3 \cdot u1 & u3 \cdot u2 & u3 \cdot u3 & u3 \cdot u4 & u3 \cdot u5 \\ u4 \cdot u1 & u4 \cdot u2 & u4 \cdot u3 & u4 \cdot u4 & u4 \cdot u5 \\ u5 \cdot u1 & u5 \cdot u2 & u5 \cdot u3 & u5 \cdot u4 & u5 \cdot u5 \end{bmatrix} \tag{5}$$

As usual, let $ui_i = 1$ to obtain a unique factorization. With this convention, the diagonals of the D array can now be stored as the diagonals of the U array. This new array is referred to as the UD array:[9]

$$UD = \begin{bmatrix} d_1 & u1_2 & u1_3 & u1_4 & u1_5 \\ 0 & d_2 & u2_3 & u2_4 & u2_5 \\ 0 & 0 & d_3 & u3_4 & u3_5 \\ 0 & 0 & 0 & d_4 & u4_5 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \tag{6}$$

Say it is desired to reorder the parameters as $U1$, $U3$, $U4$, $U2$, $U5$. The first step is to insert an artificial parameter, with zero variance and zero correlation with the other parameters, before parameter $U5$. This is accomplished by inserting into the UD array a row of zeros above row 5 and a column of zeros left of column 5:

$$\begin{bmatrix} d_1 & u1_2 & u1_3 & u1_4 & 0 & u1_5 \\ 0 & d_2 & u2_3 & u2_4 & 0 & u2_5 \\ 0 & 0 & d_3 & u3_4 & 0 & u3_5 \\ 0 & 0 & 0 & d_4 & 0 & u4_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \tag{7}$$

The corresponding covariance is

$$\begin{bmatrix} u1 \cdot u1 & u1 \cdot u2 & u1 \cdot u3 & u1 \cdot u4 & 0 & u1 \cdot u5 \\ u2 \cdot u1 & u2 \cdot u2 & u2 \cdot u3 & u2 \cdot u4 & 0 & u2 \cdot u5 \\ u3 \cdot u1 & u3 \cdot u2 & u3 \cdot u3 & u3 \cdot u4 & 0 & u3 \cdot u5 \\ u4 \cdot u1 & u4 \cdot u2 & u4 \cdot u3 & u4 \cdot u4 & 0 & u4 \cdot u5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ u5 \cdot u1 & u5 \cdot u2 & u5 \cdot u3 & u5 \cdot u4 & 0 & u5 \cdot u5 \end{bmatrix} \tag{8}$$

[9] The upper-triangular elements of either a UD or an SRI array are stored contiguously in computer memory as a vector; the zero elements below the diagonal are obviously not needed to represent the array.

Rows 2 and 5 of Eq. (7) can then be exchanged:

$$\begin{bmatrix} d1 & u1_2 & u1_3 & u1_4 & 0 & u1_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & u3_4 & 0 & u3_5 \\ 0 & 0 & 0 & d_4 & 0 & u4_5 \\ 0 & d_2 & u2_3 & u2_4 & 0 & u2_5 \\ 0 & 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \tag{9}$$

In practice, however, elements d_2, $u2_3$, and $u2_4$ would not be moved, since a scratch array would be needed to store these elements. As will soon be evident, the second row can itself be used as a scratch array. The array in Eq. (9) implicitly corresponds to a nontriangular $\tilde{U}$ array

$$\tilde{U} = \begin{bmatrix} u1_1 & u1_2 & u1_3 & u1_4 & 0 & u1_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u3_3 & u3_4 & 0 & u3_5 \\ 0 & 0 & 0 & u4_4 & 0 & u4_5 \\ 0 & u2_2 & u2_3 & u2_4 & 0 & u2_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5 \end{bmatrix} \tag{10}$$

a diagonal $\tilde{D}$ array

$$\tilde{D} = \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \tag{11}$$

and a covariance $\tilde{P}$

$$\tilde{P} = \tilde{U}\tilde{D}\tilde{U}^t$$

$$= \begin{bmatrix} u1 \cdot u1 & 0 & u1 \cdot u3 & u1 \cdot u4 & u1 \cdot u2 & u1 \cdot u5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ u3 \cdot u1 & 0 & u3 \cdot u3 & u3 \cdot u4 & u3 \cdot u2 & u3 \cdot u5 \\ u4 \cdot u1 & 0 & u4 \cdot u3 & u4 \cdot u4 & u4 \cdot u2 & u4 \cdot u5 \\ u2 \cdot u1 & 0 & u2 \cdot u3 & u2 \cdot u4 & u2 \cdot u2 & u2 \cdot u5 \\ u5 \cdot u1 & 0 & u5 \cdot u3 & u5 \cdot u4 & u5 \cdot u2 & u5 \cdot u5 \end{bmatrix} \tag{12}$$

Except for the artificial parameter, this is the desired permuted covariance. The covariance factorization in Eq. (12) can be thought of as the product of an upper-triangular array and its transpose:

$$\tilde{P} = \tilde{U}\sqrt{\tilde{D}}(\tilde{U}\sqrt{\tilde{D}})^t \tag{13}$$

where

$$\sqrt{\tilde{D}} = \begin{bmatrix} \sqrt{d}_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{d}_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{d}_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{d}_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{d}_5 \end{bmatrix} \tag{14}$$

It is desirable to construct an orthogonal transformation T such that $\tilde{U}\sqrt{\tilde{D}}T$ is an upper-triangular array with zeros in the second row and column:

$$\tilde{U}\sqrt{\tilde{D}}T = \begin{bmatrix} u1_1\sqrt{d}_1 & u1_2\sqrt{d}_2 & u1_3\sqrt{d}_3 & u1_4\sqrt{d}_4 & 0 & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u3_3\sqrt{d}_3 & u3_4\sqrt{d}_4 & 0 & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & u2_2\sqrt{d}_2 & u2_3\sqrt{d}_3 & u2_4\sqrt{d}_4 & 0 & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix} T \tag{15}$$

$$\tilde{U}\sqrt{\tilde{D}}T = \begin{bmatrix} u1_1\sqrt{d}_1 & 0 & x & x & x & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & x & x & u4_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & x & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix} \tag{16}$$

A portion of the array in Eq. (15) is similar to a Morf-Kailath prearray with zero noise variance. Reduction of this prearray to the postarray in Eq. (16) is easily accomplished with three fast Givens rotations. As a result of the zero noise variance, the first fast Givens T_0 is simply a permutation of columns 2 and 5:

$$\tilde{U}\sqrt{\tilde{D}}T_0 = \begin{bmatrix} u1_1\sqrt{d}_1 & u1_2\sqrt{d}_2 & u1_3\sqrt{d}_3 & u1_4\sqrt{d}_4 & 0 & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u3_3\sqrt{d}_3 & u3_4\sqrt{d}_4 & 0 & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & u2_2\sqrt{d}_2 & u2_3\sqrt{d}_3 & u2_4\sqrt{d}_4 & 0 & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} u1_1\sqrt{d}_1 & 0 & u1_3\sqrt{d}_3 & u1_4\sqrt{d}_4 & u1_2\sqrt{d}_2 & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u3_3\sqrt{d}_3 & u3_4\sqrt{d}_4 & 0 & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & 0 & u2_3\sqrt{d}_3 & u2_4\sqrt{d}_4 & u2_2\sqrt{d}_2 & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix} \tag{17}$$

This is the required step to produce the zeros in the second row and column. The next fast Givens T_1 operates on columns 3 and 5 in such a way that the array element containing $u2_3\sqrt{d}_3$ becomes zero. The final fast Givens T_2 operates on columns 4 and 5 such that the array element containing $u2_4\sqrt{d}_4$ becomes zero. Each of these Givens transformations maps the elements in each row of the columns on which they operate into a linear combination of the same. Hence, elements not consistent with an upper-triangular structure are never created; a linear combination of zeros is still zero. For example, T_1 operates on Eq. (17) as follows:

$$\tilde{U}\sqrt{\tilde{D}}T_0T_1 =$$

$$\begin{bmatrix} u1_1\sqrt{d}_1 & 0 & u1_3\sqrt{d}_3 & u1_4\sqrt{d}_4 & u1_2\sqrt{d}_2 & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u3_3\sqrt{d}_3 & u3_4\sqrt{d}_4 & 0 & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & 0 & u2_3\sqrt{d}_3 & u2_4\sqrt{d}_4 & u2_2\sqrt{d}_2 & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & s & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} u1_1\sqrt{d}_1 & 0 & x & u1_4\sqrt{d}_4 & x & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & u3_4\sqrt{d}_4 & x & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & 0 & 0 & u2_4\sqrt{d}_4 & x & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix} \tag{18}$$

Likewise, T_2 operates on Eq. (18) as follows:

$$\tilde{U}\sqrt{\tilde{D}}T_0T_1T_2 = \begin{bmatrix} u1_1\sqrt{d}_1 & 0 & x & u1_4\sqrt{d}_4 & x & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & u3_4\sqrt{d}_4 & x & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & u4_4\sqrt{d}_4 & 0 & u4_5\sqrt{d}_5 \\ 0 & 0 & 0 & u2_4\sqrt{d}_4 & x & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & s & 0 \\ 0 & 0 & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} u1_1\sqrt{d}_1 & 0 & x & y & y & u1_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & y & y & u3_5\sqrt{d}_5 \\ 0 & 0 & 0 & y & y & u4_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & y & u2_5\sqrt{d}_5 \\ 0 & 0 & 0 & 0 & 0 & u5_5\sqrt{d}_5 \end{bmatrix} \tag{19}$$

Trivially, since the transformations T_i are orthogonal, the covariance $\tilde{P}$ in Eqs. (12) and (13) remains unchanged:

$$\tilde{P} = \tilde{U}\sqrt{\tilde{D}}T_0T_1T_2(\tilde{U}\sqrt{\tilde{D}}T_0T_1T_2)^t$$

$$= \tilde{U}\tilde{D}\tilde{U}^t \tag{20}$$

The importance of the zeros in the second row and column of Eq. (16) is now readily apparent. Since the insertion of a row and column of zeros into the UD array in Eq. (7) did not affect the individual elements of the covariance in Eq. (8), deletion of the second row and column of zeros from Eq. (16) will have a similar null effect on the individual elements of the covariance $\tilde{P}$.

III. Column Permutations of SRI Arrays

To demonstrate the technique on SRI arrays, represent an augmented SRI array R with parameter order $R1, R2, R3, R4, R5$ as

$$(R|Z) = \left[\begin{array}{ccccc|c} r1_1 & r2_1 & r3_1 & r4_1 & r5_1 & z_1 \\ 0 & r2_2 & r3_2 & r4_2 & r5_2 & z_2 \\ 0 & 0 & r3_3 & r4_3 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & r5_4 & z_4 \\ 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{21}$$

where Z is the residual vector. The following identity suggests the justification of incorporating an artificial parameter into an SRI array:

$$\begin{pmatrix} R_x & 0 & R_{xy} \\ 0 & \epsilon & 0 \\ 0 & 0 & R_y \end{pmatrix}^{-1} = \begin{pmatrix} R_x^{-1} & 0 & -R_x^{-1}R_{xy}R_y^{-1} \\ 0 & \epsilon^{-1} & 0 \\ 0 & 0 & R_y^{-1} \end{pmatrix} \tag{22}$$

If there are no intentions of computing the inverse of the SRI array, it is permissible to allow $\epsilon = 0$. Hence, the insertion and subsequent deletion of an artificial row and column of zeros into an SRI array does not affect the information content of the array.

Say it is desired to change the parameter order to $R1, R3, R4, R2, R5$. The first step is to insert a row and column of zeros into the $(R|Z)$ array before the parameter $R5$:

$$\left[\begin{array}{cccccc|c} r1_1 & r2_1 & r3_1 & r4_1 & 0 & r5_1 & z_1 \\ 0 & r2_2 & r3_2 & r4_2 & 0 & r5_2 & z_2 \\ 0 & 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{23}$$

Columns 2 and 5 of Eq. (23) can then be exchanged. This is just a reordering of the design equation

$$\overline{(R|Z)} = \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{24}$$

It is desirable to construct an orthogonal transformation T_0 such that $T_0\overline{(R|Z)}$ has only zeros in the second row and column. Such a transformation is just a permutation of rows 2 and 5:

$$T_0\overline{(R|Z)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right]$$

$$= \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{25}$$

In practice, however, elements $r3_2$ and $r4_2$ would not be moved, since a scratch array would be needed to store these elements. As is evident, the second row can itself be used as a scratch array. It is now a simple matter of retriangularizing Eq. (25) with two slow Givens. The first Givens T_1 operates on rows 3 and 5 in such a way that the array element containing $r3_2$ becomes zero. The second Givens T_2 operates on rows 4 and 5 in such a way that the array element containing $r4_2$ becomes zero. Each of these Givens transformations maps the elements in each column of the rows on which they operate into a linear combination of the same. Elements not consistent with an upper-triangular structure are never created. For example, T_1 operates on Eq. (25) as follows:

$$T_1\overline{T_0(R|Z)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & s & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \pm s & 0 & \mp c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\times \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right]$$

$$= \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{26}$$

Likewise, T_2 operates on Eq. (26) as follows:

$$T_2T_1\overline{T_0(R|Z)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & s & 0 \\ 0 & 0 & 0 & \pm s & \mp c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\times \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right]$$

$$= \left[\begin{array}{cccccc|c} r1_1 & 0 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & y & y & y & y \\ 0 & 0 & 0 & 0 & y & y & y \\ 0 & 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \tag{27}$$

From Eq. (22) it is a trivial matter that the second row and column of zeros of Eq. (27) can then be deleted.

IV. Cyclic Permutations

Since the artificial parameter is only a convenient place holder, an alternative method can be constructed that does not incorporate the insertion/deletion step. By cyclically permuting the rows and columns of either the UD or SRI arrays, a data structure similar to Eq. (17) or Eq. (25), respectively, can be achieved. For example, cyclically left permuting columns 2 through 4 of Eq. (21) yields the following:

$$\left[\begin{array}{ccccc|c} r1_1 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \quad (28)$$

Cyclically upward permuting rows 2 through 4 of Eq. (28) then yields

$$\left[\begin{array}{ccccc|c} r1_1 & r3_1 & r4_1 & r2_1 & r5_1 & z_1 \\ 0 & r3_3 & r4_3 & 0 & r5_3 & z_3 \\ 0 & 0 & r4_4 & 0 & r5_4 & z_4 \\ 0 & r3_2 & r4_2 & r2_2 & r5_2 & z_2 \\ 0 & 0 & 0 & 0 & r5_5 & z_5 \end{array}\right] \quad (29)$$

This is the desired data structure of Eq. (25) without the artificial parameter. Slow Givens sweeps across rows 2,4 and rows 3,4 of Eq. (29) can then be applied to zero out the array elements containing $r3_2$ and $r4_2$, respectively. Since the size of the array never changes, this method would not require that the array be augmented with an extra column. However, one explicit scratch array is required to hold that part of Eq. (29) not consistent with an upper-triangular data structure.

A similar approach can also be taken toward permuting UD arrays. Cyclic permutations of the rows and columns would yield the data structure in Eq. (17) without the artificial parameter:

$$\left[\begin{array}{ccccc} u1_1\sqrt{d_1} & u1_3\sqrt{d_3} & u1_4\sqrt{d_4} & u1_2\sqrt{d_2} & u1_5\sqrt{d_5} \\ 0 & u3_3\sqrt{d_3} & u3_4\sqrt{d_4} & 0 & u3_5\sqrt{d_5} \\ 0 & 0 & u4_4\sqrt{d_4} & 0 & u4_5\sqrt{d_5} \\ 0 & u2_3\sqrt{d_3} & u2_4\sqrt{d_4} & u2_2\sqrt{d_2} & u2_5\sqrt{d_5} \\ 0 & 0 & 0 & 0 & u5_5\sqrt{d_5} \end{array}\right] \quad (30)$$

Fast Givens sweeps down columns 2,4 and columns 3,4 of Eq. (30) can then be applied to zero out the array elements containing $u2_3\sqrt{d_3}$ and $u2_4\sqrt{d_4}$, respectively.

V. Implementation

When dealing with large arrays on a virtual memory machine, computations should be arranged to minimize expensive page faulting. The Givens sweeps should access contiguous storage locations. The usual column ordering of a vector-stored upper-triangular data structure with n rows and n columns is as follows:

$$S_{\text{column}} = \begin{bmatrix} s(1) & s(2) & s(4) & \cdots & s\left(\frac{n^2-n}{2}+1\right) \\ 0 & s(3) & s(5) & \cdots & s\left(\frac{n^2-n}{2}+2\right) \\ 0 & 0 & s(6) & \cdots & s\left(\frac{n^2-n}{2}+3\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s\left(\frac{n^2+n}{2}\right) \end{bmatrix} \quad (31)$$

Element $S_{\text{column}}(i, j)$ is easily referenced as $s[(j^2 - j/2) + i]$. For UD arrays, the Givens sweeps down the columns using the data storage in Eq. (31) would require minimal paging. For SRI arrays, the Givens sweeps across the rows with the data storage in Eq. (31) would be paging intensive. Hence, for SRI arrays, it proves to be more efficient to store the upper-triangular array with unnatural row ordering as follows:

$$S_{\text{row}} = \begin{bmatrix} s(1) & s(2) & s(3) & \cdots & s(n) \\ 0 & s(n+1) & s(n+2) & \cdots & s(2n-1) \\ 0 & 0 & s(2n) & \cdots & s(3n-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s\left(\frac{n^2+n}{2}\right) \end{bmatrix} \quad (32)$$

Element $S_{\text{row}}(i, j)$ can be referenced as $s[j + (i^2 - i/2) + (i - 1)(n - i)]$.[10] Since filtering software generally stores

[10] $S_{\text{row}}(i, j) = S_{\text{column}}(i, j)$; this is just a relabeling of the elements.

upper-triangular arrays according to Eq. (31), it would be necessary to relabel from S_{column} to S_{row} before the permutations and then back to S_{column} afterward.

It is equally easy to perform cyclic permutations with either Eq. (31) or Eq. (32). In the former, movement is along the columns; in the latter, movement is across the rows. The only paging-intensive step in cyclically permuting the rows and columns with data storage S_{column} (S_{row}) is in storing the upper row (left column) into a scratch array.

VI. Performance

To assess the permutation methods on UD arrays, an upper-triangular array was constructed with 499 parameters and an estimate column. The following permutations were then performed on a VAX 11/780. First, the parameters were permuted randomly. In practice, of course, this would never happen. It is more realistic to move blocks of parameters to the bottom of the array. Therefore, the following systematic permutations were performed. A block of 10 parameters starting in rows 1, 167, and 334 were moved to the bottom of the array. These permutations were then repeated with a block of 100 parameters. CPU time and page faults were accumulated for three different methods. First, the ESL routine HHPOST was used to apply a post-Householder to a row-reordered column-scaled UD array. Second, pairwise permutations were performed with suboptimal noise-free measurement updates. This is equivalent to the ESL routine U2U. Third, the permutations were performed with the discussed method of cyclic permutations with fast Givens sweeps. The results are presented in Table 1.

The ESL method with HHPOST performs the same regardless of the number and type of permutations performed. The performance of the other methods depends on the number and type of permutations. The cyclic permutation method with fast Givens sweeps is shown to be superior to the ESL method in U2U. Furthermore, in the cyclic permutation method, as the starting row moves down there are fewer elements to retriangularize.

The ESL has standard routines to perform permutations on SRI arrays. The routine R2A copies and reorders the columns of an upper-triangular array into a two-dimensional work array. A Householder transformation can then be applied with routine TDHHT to retriangularize back to an upper-triangular array.

To assess the permutation methods on SRI arrays, ESL routines R2A/TDHHT, the discussed method of cyclic permutations with slow Givens sweeps using data storage S_{column}, and the same with data storage S_{row} were used to permute an upper-triangular array with 499 parameters and a residual column. CPU time and page faults were accumulated for permuting, retriangularizing, and any necessary relabeling. The results are presented in Table 2.

The ESL method performs the same regardless of the number and type of permutations performed. The performance of the cyclic permutations with slow Givens sweeps depends on the number and type of permutations. As the starting column is moved to the right, not only are there fewer elements to retriangularize, but the lengths of Givens sweeps also become smaller, and fewer computations are required. Since the VAX 11/780 is a virtual memory machine, the method with data storage S_{row} is generally superior to the method with data storage S_{column}. For fixed memory machines or small SRI arrays, the method with data storage S_{column} should be superior, since no relabeling or unlabeling is required.

VII. Conclusion

The most efficient method to permute UD (SRI) arrays has been shown to be cyclic permutation with fast (slow) Givens sweeps for retriangularization. Since the only difference is in the retriangularization process, it has been possible to combine the methods into one subroutine. This method has been incorporated into the OASIS software [7] and should prove to be advantageous for GPS, TOPEX, and DSN high-earth-orbiter data processing and analysis.

References

[1] G. J. Bierman, *Factorization Methods for Discrete Sequential Estimation*, New York: Academic Press, pp. 139–140, 1977.

[2] G. J. Bierman and K. H. Bierman, *Estimation Subroutine Library User Guide*, Studio City, California: Factorized Applications, Inc., August 1984 (revised August 1985).

[3] J. L. Fanselow and O. J. Sovers, *Observation Model and Parameter Partials for the JPL VLBI Parameter Estimation Software MASTERFIT*, JPL Publication 83-89, Rev. 3, Jet Propulsion Laboratory, Pasadena, California, December 15, 1987.

[4] M. Morf and T. Kailath, "Square-Root Algorithms for Least-Squares Estimation," *IEEE Trans. Automatic Control*, vol. AC-20, no. 4, pp. 487–497, August 1975.

[5] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK User's Guide*, Philadelphia: SIAM, pp. 10.15–10.16, 1979.

[6] J. W. Daniel et al., "Reorthogonalization and Stable Algorithms for Updating the Gram–Schmidt QR Factorization," *Math. Comp.*, vol. 30, pp. 772–795, 1976.

[7] S. C. Wu and C. L. Thornton, "OASIS – A New GPS Covariance and Simulation Analysis Software System," in *Proceedings of the First International Symposium on Precise Positioning With the Global Positioning System*, pp. 337–346, April 1985.

Table 1. CPU and paging faults in permuting UD arrays

Permutations	CPU time; thousands of page faults		
	ESL HHPOST	ESL U2U	Cyclic permutations and fast Givens
Random	0:33:06 1025	1:06:31 2051	0:17:04 612
Rows 1–10	0:32:43 948	0:48:55 1823	0:01:24 40
Rows 167–176	0:31:49 970	0:40:00 1338	0:01:14 35
Rows 334–343	0:31:12 970	0:24:26 728	0:00:45 21
Rows 1–100	0:32:44 968	0:59:25 1985	0:11:23 363
Rows 167–266	0:31:32 946	0:48:47 1433	0:08:46 275
Rows 334–433	0:30:59 973	0:27:29 747	0:02:20 52

Table 2. CPU and paging faults in permuting SRI arrays

Permutations	CPU time; thousands of page faults		
	ESL TDHHT	Cyclic permutations and slow Givens using data storage S_{column}	Cyclic permutations and slow Givens using data storage S_{row}
Random	0:22:28 729	0:41:09 3253	0:20:07 575
Columns 1–10	0:20:29 717	0:01:36 48	0:01:38 55
Columns 167–176	0:21:13 723	0:00:55 31	0:01:03 41
Columns 334–343	0:21:53 719	0:00:20 11	0:00:39 27
Columns 1–100	0:21:25 727	0:13:11 424	0:11:10 411
Columns 167–266	0:21:15 714	0:06:14 229	0:05:14 205
Columns 334–433	0:21:03 723	0:00:51 52	0:01:43 104