C VAN LOAN

# Parallel algorithms for constrained and unconstrained least squares problems

## 1. INTRODUCTION

During the past few years we have been concerned with the parallel computation of the singular value decomposition (SVD). In the SVD we are given an m-by-n matrix $A$ and compute orthogonal $U$ (m-by-m) and $V$ (n-by-n) such that

$$U^T A V = \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \quad . \tag{1.1}$$

The $\sigma_i$ are the singular values and $U$ and $V$ can be chosen so that they satisfy $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ . The SVD is one of the most important matrix decompositions in numerical linear algebra and many of its numerous applications are detailed in Golub and Van Loan [ 3 ].

Perhaps the leading application of the SVD is the least square problem

$$\min \|Ax - b\|_2 \qquad A \in R^{mxn} , \quad b \in R^m \tag{1.2}$$

especially when $A$ is rank degenerate. Being able to solve this problem in real-time is of interest in several signal processing applications. Unfortunately, conventional computers are unable to handle this task because m and n are too big for the alotted time that one has to get the solution $x$. Thus, it is not surprising that the real-time signal processing community has become quite interested in special purpose architectures that can solve least square problems in parallel.

In §2 we describe one such architecture-a systolic array-that has been proposed for computing the SVD. The underlying algorithm is based on a Jacobi procedure. We show how a few simple extensions of the procedure can widen the class of problems that the architecture can handle. In particular, we present what we call a Jacobi submatrix SVD procedure that can be used to solve constrained least square problems of the variety

$$\min_{Bx = d} \|Ax - b\|_2 \quad , \quad A \in R^{mxn} , \; b \in R^m , \; B \in R^{pxn} , \; d \in R^p \tag{1.3}$$

In §3 we discuss a block analogue of the Jacobi SVD procedure that is suitable for the case of a few loosely coupled but powerful processors.

One problem with the SVD is that it is hard to update. That is, if we have the SVD of a matrix A and then want the SVD of a rank one perturbation of A, we pretty much have to start from scratch. This makes the QR factorization much more attractive in least squares settings where updating is required. For example, in a beamforming application that we are aware of, we must solve the problem (1.3) for a fixed A but with many different B. In §4 we show how to extend some parallel QR factorization procedures so that they can handle this case.

Many of the techniques that we discuss in this paper are detailed elsewhere. Our intention is to give an overview of work in this area and to impress upon the reader the need for parallel algorithms that are simple enough to be implemented on a special purpose device but general enough so that they can solve more than one narrowly defined problem.

## 2. SUBMATRIX SVD USING JACOBI ROTATIONS

Assume for the moment that A is a square n-by-n real matrix. In the Jacobi approach to computing the SVD, A is made increasingly diagonal through updates of the form

$$A := J(p,q,\theta_1)^T A\, J(p,q,\theta_2) \ . \tag{2.1}$$

Here, $J(p,q,\theta)$ is a Jacobi rotation in the (p,q) plane, i.e., $J(p,q,\theta) = (z_{ij})$ is the identity everywhere except

$$\begin{bmatrix} z_{pp} & z_{pq} \\ z_{qp} & z_{qq} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \ .$$

For a given index pair (p,q) in (2.1) it is possible to choose $\theta_1$ and $\theta_2$ such that

$$\begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} \cos(\theta_2) & \sin(\theta_2) \\ -\sin(\theta_2) & \cos(\theta_2) \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \tag{2.2}$$

This amounts to solving a 2-by-2 SVD problem. By choosing the rotations in in this fashion and updating A as in (2.1), the sum of the squares of A's

off-diagonal elements is reduced by the amount $a_{pq}^2 + a_{qp}^2$ . If the rotation indices (p,q) are chosen properly then the matrix A converges to the diagonal matrix of singular values. Formally, if

$$\text{off}(A) = \text{sqrt}\Big( \sum_{i \neq j} a_{ij}^2 \Big)$$

$\varepsilon$ = given tolerance, e.g., unit roundoff

$\tau = \varepsilon \| A \|_F / n$

$\{(p_i,q_i)\}_{i=1}^{N}$ be an ordering of $\{(p,q) \mid 1 \leq p < q \leq n\}$ , $N = \frac{n(n-1)}{2}$

then the following algorithm terminates:

```
Do While ( off(A) >  ε || A ||_F )
   For i = 1:N
      (p,q) := (p_i,q_i)                                        (2.3)
      If  a_pq^2 + a_qp^2 > τ^2
          then
              Solve the 2-by-2 SVD problem (2.2).
              A := J(p,q,θ_1)^T A J(p,q,θ_2)
```

A proof of termination may be found in [ 6]. The parameter $\tau$ is called the threshold parameter. Its presence in the algorithm ensures that we only perform "worthwhile" rotations. The algorithm may not terminate if $\tau$ is set to zero. How to solve the 2-by-2 subproblems is discussed in [ 2].

Perhaps the most interesting aspect of (2.3) is the ordering. The cyclic-by-row ordering is most often discussed in elementary texts:

$$(1,2),(1,3),\ldots,(1,n),(2,3),\ldots(2,n),\ldots.(n-1,n) \ .$$

For most reasonable orderings, (2.3) will converge after a small number of sweeps. (A single pass through the For-loop is called a sweep.)

The most exciting development in the last few years concerning Jacobi methods was the discovery of the "parallel" ordering in [ 1 ]. For n = 8, the parallel ordering is as follows:

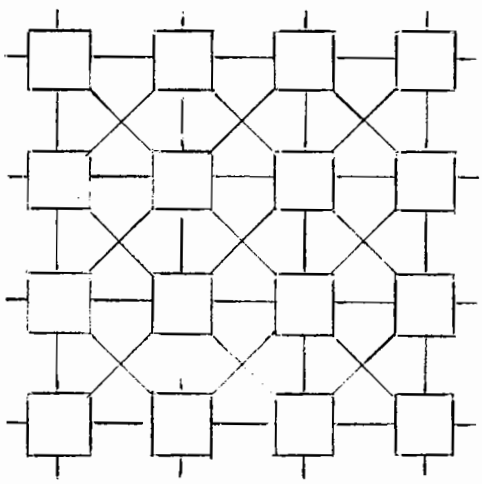
| | | | | |
|---|---|---|---|---|
| I. | (1,2) | (3,4) | (5,6) | (7,8) |
| II. | (1,4) | (2,6) | (3,8) | (5,7) |
| III. | (1,6) | (4,8) | (2,7) | (3,5) |
| IV. | (1,8) | (6,7) | (4,5) | (2,3) |
| V. | (1,7) | (5,8) | (3,6) | (2,4) |
| VI. | (1,5) | (3,7) | (2,8) | (4,6) |
| VII. | (1,3) | (2,5) | (4,7) | (6,8) |

The table should be read left-to-right, top-to-bottom. Notice that the rotations prescribed by a given row are "nonconflicting". That is, they involve independent subproblems. A parallel implementation of (2.3) might read:

Solve all the subproblems associated with rotation set I.
Perform the row updates associated with each subproblem.
Perform the column updates associated with each subproblem.
Solve all the subproblems associated with rotation set II.
⋮

The fact that the subproblems and updates are independent, means that they can be performed concurrently.

In [ 2 ] we show how an array of processors of the form



(2.4)

can be used to carry out a sweep in the parallel ordering in O(n) time steps. The diagonal processors solve 2-by-2 SVD problems while the off-diagonal

processors perform the 2-by-2 matrix multiplications necessary to carry out the updates $A := J(p,q,\theta_1)^T A\, J(p,q,\theta_2)$ .

At this point we'd like to dwell on an interesting practical question. Suppose we had an array of processors capable of solving 8-by-8 SVD problems. How could it be used to solve, say, a 6-by-5 problem? An obvious course of action might be to pad $A$ with zeroes to make it 8-by-8:

$$\bar{A} = \begin{matrix} x & x & x & x & x & o & o & o \\ x & x & x & x & x & o & o & o \\ x & x & x & x & x & o & o & o \\ x & x & x & x & x & o & o & o \\ x & x & x & x & x & o & o & o \\ x & x & x & x & x & o & o & o \\ o & o & o & o & o & o & o & o \\ o & o & o & o & o & o & o & o \end{matrix} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \tag{2.5}$$

One would then expect the array to compute the SVD

$$\begin{bmatrix} U & 0 \\ 0 & I_2 \end{bmatrix}^T \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & I_3 \end{bmatrix} = \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix}$$

whereupon $U^TAV = \Sigma$ . Trouble can arise, however, if $A$ has a nontrivial nullspace for then the orthogonal transformations in (2.5) need not be in direct sum form. In this case we say that the "genuine" nullspace of $A$ intermingles with the "artificial" nullspace of $\bar{A}$ that arises because of the bordering of zeroes. The nullspace of A can be retrieved from the computed nullspace of $\bar{A}$ , but this is inconvenient. More importantly, the accuracy of A's computed singular vectors that are associated with small singular values is degraded by the intermingling effect.

Fortunately, these difficulties can be avoided if the 2-by-2 subproblems are carefully solved. The idea is to avoid the intermingling of $A$ and its bordering of zeroes. For example, if we are doing a (4,6) rotation, then the associated 2-by-2 SVD problem can be solved in two ways:

$$\begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix}^T \begin{bmatrix} x & 0 \\ x & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix}^T \begin{bmatrix} x & 0 \\ x & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & x \end{bmatrix}$$

Notice that if the second choice of sines and cosines is taken, then the zero-nonzero structure of the original Ä is destroyed insofar as columns 4 and 6 are swapped. Clearly, we should set $\theta_2 = 0$ when $(p,q) = (4,6)$. More generally, if we set $\theta_1 = 0$ whenever $q \geq 7$ and $\theta_2 = 0$ whenever $q \geq 6$ then one can prove that the "direct sum" SVD (2.6) ensues.

By being careful at the 2-by-2 subproblem level, as when avoiding the intermingling problem, one can develop Jacobi procedures that compute the SVD of a designated submatrix. This is an important capability as it widens the class of problems that the SVD array can solve. As a case in point, consider the constrained least squares problem (1.3). Suppose that we compute orthogonal $U_1$, $U_2$, and $V$ such that

$$\begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}^T \begin{bmatrix} B \\ A \end{bmatrix} V = \begin{bmatrix} D_B & 0 \\ A_1 & D_A \end{bmatrix} \tag{2.6}$$

where $D_B$ (p-by-p) and $D_A$ (m-by-(n-p)) are diagonal. This can be accomplished by computing the SVD

$$B = U_1 \begin{bmatrix} D_B & 0 \end{bmatrix} V_1^T$$

and then computing the SVD of the last n-p columns of $AV_1$. With these reductions (1.3) transforms to the easily solved problem

$$\min \| D_A y_2 - (U_2^T b - A_1 y_1) \|_2$$
$$D_B y_1 = U_1^T d$$

whereupon

$$x = V \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \qquad y_1 \in R^p \quad , \quad y_2 \in R^{n-p} \quad .$$

Let's see how a slight modification of algorithm (2.3) can be used to compute the SVD-like decomposition (2.6).

Suppose we have an 8-by-8 array and that $A \in R^{6x5}$ and $B \in R^{2x5}$. We initialize the processor array as follows:

```
b  b  b  b  b  o  o  o
b  b  b  b  b  o  o  o
a  a  a  a  a  o  o  o
a  a  a  a  a  o  o  o
a  a  a  a  a  o  o  o
a  a  a  a  a  o  o  o
a  a  a  a  a  o  o  o
o  o  o  o  o  o  o  o
```

Here, a (b) denotes an arbitrary nonzero element of A (B). The first calculation to be performed involves the SVD of the submatrix defined by rows 1 and 2 and columns 1 through 5. Noting that submatrices can be designated by bit array we can specify the above mentioned submatrix as follows:

```
R  R  R  R  R  N  N  N
R  R  R  R  R  N  N  N
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
```

Here, "R" indicates a submatrix entry while "N" indicates an entry "outside" the submatrix. When the Jacobi SVD procedure is applied to the 8-by-8 matrix there is a 2-by-2 bit array associated with each 2-by-2 SVD subproblem. There are four possibilities:

$$\begin{bmatrix} R & R \\ R & R \end{bmatrix} \quad \begin{bmatrix} R & R \\ N & N \end{bmatrix} \quad \begin{bmatrix} R & N \\ N & N \end{bmatrix} \quad \begin{bmatrix} N & N \\ N & N \end{bmatrix}$$

(a) (b) (c) (d)

If the 2-by-2 subproblems in each of these cases is solved as follows,

$$\text{(a)}: \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & x \end{bmatrix}$$

$$\text{(b):}\quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix}$$

$$\text{(c):}\quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

$$\text{(d)}\quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

then after a sufficient number of sweeps the array is transformed to

```
x  o  o  o  o  o  o  o
o  x  o  o  o  o  o  o
x  x  x  x  x  o  o  o
x  x  x  x  x  o  o  o
x  x  x  x  x  o  o  o
x  x  x  x  x  o  o  o
x  x  x  x  x  o  o  o
o  o  o  o  o  o  o  o
```

Notice that the 2-by-2 subproblems are solved in a way that avoids the mingling of "N" entries and "R" entries.

The second step in the computation of (2.6) is to compute the SVD of the submatrix defined by columns 3, 4, and 5 and rows 3 through 7 . For this phase of the calculation the following bit array is pertinant:

```
N  N  N  N  N  N  N  N
N  N  N  N  N  N  N  N
N  N  R  R  R  N  N  N
N  N  R  R  R  N  N  N
N  N  R  R  R  N  N  N
N  N  R  R  R  N  N  N
N  N  R  R  R  N  N  N
N  N  N  N  N  N  N  N
```

There are five "types" of 2-by-2 subproblems in this case. If they are solved as follows:

$$\begin{bmatrix} R & R \\ R & R \end{bmatrix} : \quad \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & x \end{bmatrix}$$

$$\begin{bmatrix} R & N \\ R & N \end{bmatrix} : \quad \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ 0 & x \end{bmatrix}$$

$$\begin{bmatrix} R & N \\ N & N \end{bmatrix} : \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

$$\begin{bmatrix} N & N \\ N & R \end{bmatrix} : \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

$$\begin{bmatrix} N & N \\ N & N \end{bmatrix} : \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

then the matrix in the array converges to

$$\begin{bmatrix} x & o & o & o & o & o & o & o \\ o & x & o & o & o & o & o & o \\ x & x & x & o & o & o & o & o \\ x & x & o & x & o & o & o & o \\ x & x & o & o & x & o & o & o \\ x & x & o & o & o & o & o & o \\ x & x & o & o & o & o & o & o \\ o & o & o & o & o & o & o & o \end{bmatrix}$$

Thus, the decomposition (2.6) is achieved. The matrices $U_1$ and $U_2$ are the accumulation of all the left rotations while V is the accumulation of all the right rotations.

It is natural to wonder why algorithm (2.3) should converge with all these stipulations on how the 2-by-2 subproblems are solved. The reason is simple. If we associate a zero with each "N" entry then our "submatrix SVD" procedure

is equivalent to (2.3) with no intermingling which we know converges.

The bit array idea is attractive in that it is a simple way to "reprogram" the processor array. So long as the diagonal processors in (2.4) are set up to decipher 2-by-2 bit arrays, we're able to solve more than just simple SVD problems. This is what we mean by a simple extension of a basic algorithm that widens the class of problems that it can solve.

## 3. BLOCK JACOBI SVD

In the parallel matrix computation area, block algorithms can sometimes be used to achieve a more favorable ratio of computation to communication. Consider the processor array (2.4). Basically, information is passed in between neighboring processors after 2-by-2 matrix computations are performed. If the processors could solve larger problems then more time might be spent computing and less (relatively) on communication.

In this connection it is important to recognize that algorithm (2.3) has a block analogue. Suppose

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

where each $A_{ij}$ is a r-by-r matrix. (We have assumed that everything is square for expository purposes.) Here's the block version of (2.3) with accumulation of U and V:

```
V := I
U := I
Do While ( OFF(A) >  ε || A ||_F  )
    For i = 1:N
        (p,q) = (p_i,q_i)
        If  || A_pq ||_F^2  + || A_qp ||_F^2  > τ^2
              then
```

Solve the 2r-by-2r SVD problem

$$\bar{U}^T \begin{bmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{bmatrix} \bar{V} = \bar{\Sigma}$$

$$A := J(p,q,\bar{U})^T A\, J(p,q,\bar{V})$$

$$U := U\, J(p,q,\bar{U})$$

$$V := V\, J(p,q,\bar{V})$$

Here, OFF(·) is defined by

$$\mathrm{OFF}(A) = \mathrm{sqrt}\left( \sum_{i \neq j} \| A_{ij} \|_F^2 \right)$$

and $J(p,q,V)$ denotes the nr-by-nr orthogonal matrix $(Z_{ij})$ that is the identity everywhere except that $Z_{pp} = \bar{V}_{11}$, $Z_{pq} = \bar{V}_{12}$, $Z_{qp} = \bar{V}_{21}$, and $Z_{qq} = \bar{V}_{22}$.

All of the properties of the scalar Jacobi SVD carry over to the block algorithm. However, there is some additional flexibility. For example, it turns out to be unnecessary to compute the full SVD in each subproblem. See [ 6 ] for details. We are implementing the block Jacobi SVD procedure in an environment that has several FPS 164's connected to an IBM host. We will report on that experience elsewhere.

## 4. A PARALLEL IMPLEMENTATION OF THE METHOD OF WEIGHTING

The last topic that we would like to cover concerns the solution of the constrained problem (1.3). It is well-known how to use the QR factorization to solve this problem:

1. $B^T = [Q_{1B}, Q_{2B}] \begin{bmatrix} R_B \\ 0 \end{bmatrix}$ . (QR factorization)

2. Solve the lower triangular system $R_B^T y_1 = d$ for $y_1 \in R^p$ .

3. Set $A_1 = AQ_{2B}$ , compute its QR factorization and solve the unconstrained least square problem

$$\min \| A_1 y_2 - (b - AQ_{1B} y_1) \|_2$$

for $y_2 \in R^{n-p}$ .

4. $x = Q_{1B} y_1 + Q_{2B} y_2$ .

Note that two QR factorizations are required. These could be solved very fast using any of several QR processor arrays that have been recently proposed. See [ 4 ] for example.

The trouble with this approach is twofold. First, there is substantial communication overhead because two "calls" to the QR processor array are required. Second, in certain signal processing applications one is interested in the solution to (1.3) for many different B. In this context, one must repeat the entire computation for each B.

A way round these difficulties is to use the method of weights. In this approach the unconstrained problem

$$\min \left\| \begin{bmatrix} \lambda B \\ A \end{bmatrix} x - \begin{bmatrix} \lambda d \\ b \end{bmatrix} \right\|_2$$

is solved for a suitably large weight $\lambda$ . Mathematically, the larger the weight the better $x$ will approximate the true solution. Note that if the QR factorization of $A$ is known then it can be updated to obtain a QR factorization of

$$C = \begin{bmatrix} \lambda B \\ A \end{bmatrix} .$$

This can be implemented very elegantly on a very slightly modified Heller-Ipsen QR array [ 4 ].

Followers of the method of weighting know that it can be unstable for large $\lambda$ . However, we recently showed how to implement the technique with small, "safe" weights. A single QR factorization of the matrix $C$ is required. Thereafter, a few iterative improvement steps are performed to obtain an accurate solution $x$. See [ 5 ] for details. Our work on this problem is yet another example of how research in the parallel matrix computation area can lead to better sequential algorithms.

REFERENCES


1. BRENT, R. and F. LUK. The Solution of singular value and symmetric eigenvalue problems on multiprocessor arrays, SIAM J. Sci. Stat. Comp 6 (1985) 69-84.

2. BRENT, R., F. LUK, and C. VAN LOAN Computation of the singular value decomposition using mesh-connected processors, J. VLSI and Computer Systems 1 (1985) 242-270.

3. GOLUB, G.H. and C. VAN LOAN Matrix Computations (1983) , Johns Hopkins University Press, Baltimore, Maryland.

4. HELLER, D. and I. IPSEN Systolic networks for orthogonal decompositions, SIAM J. Sci. Stat. Comp. 4 (1983) 261-269 .

5. VAN LOAN, C. On the method of weighting for equality constrained least squares problems, SIAM J. Numer. Anal. (1985) to appear.

6. VAN LOAN, C. The block Jacobi method for computing the singular value decomposition, Cornell Computer Science Technical Report TR 85-680, 1985.



Charles Van Loan
Department of Computer Science
Cornell University
Ithaca, New York 14853
USA