COMPUTING THE SINGULAR VALUE DECOMPOSITION ON THE ILLIAC IV

Franklin T. Luk

TR 80-415

Department of Computer Science
Cornell University
Ithaca, New York 14853

COMPUTING THE SINGULAR VALUE DECOMPOSITION ON THE ILLIAC IV

Franklin T. Luk

Department of Computer Science
Cornell University
Ithaca, NY 14853

Accepted for publication in ACM Transactions on Mathematical Software.

Abstract:

In this paper, we study the computation of the singular value decomposition of a matrix on the ILLIAC IV computer. We describe the architecture of the machine and explain why the standard Golub-Reinsch algorithm is not applicable to this problem. We then present a one-sided orthogonalization method which makes very efficient use of the parallel computing abilities of the ILLIAC machine. Our method is shown to be Jacobi-like and numerically stable. Finally a comparison of our method on the ILLIAC IV computer with the Golub-Reinsch algorithm on a conventional machine demonstrates the great potential of parallel computers in the important area of matrix computations.

Key Words and Phrases:

ILLIAC-IV computer, Singular value decomposition, Golub-Reinsch algorithm, Jacobi-like method, Parallel matrix computations.

CR Categories: 5.14

COMPUTING THE SINGULAR VALUE DECOMPOSITION ON THE ILLIAC IV

1. Introduction

We study the computation of the singular value decomposition on the ILLIAC IV computer. Suppose that we have a real $m \times n$ matrix A. Its singular value decomposition (SVD) can be defined as

$$A = U\Sigma V^t \tag{1.1}$$

with

$$U^tU = V^tV = I_k \qquad \text{and} \qquad \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_k),$$

where

$$k = \min(m,n). \tag{1.2}$$

The matrices U and V consist of the orthonormalized eigenvectors associated with the k largest eigenvalues of AA^t and A^tA, respectively. The diagonal elements of Σ are the non-negative square roots of the k largest eigenvalues of AA^t, and are called the singular values. We assume that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \qquad \text{and} \qquad \sigma_{r+1} = \cdots = \sigma_k = 0, \tag{1.3}$$

i.e. the rank of A equals r. An alternative definition of the singular value decomposition is

$$A = U_r\Sigma_rV_r^t. \tag{1.4}$$

with

$$U_r^tU_r = V_r^tV_r = I_r \qquad \text{and} \qquad \Sigma_r = \text{diag}(\sigma_1, \ldots, \sigma_r).$$

The singular value decomposition is a very useful matrix decomposition (see [7]). Various methods have been proposed for its computation. The standard method was introduced by Golub and Kahan in 1965 [6]. They use first the Householder transformation to bidiagonalize the given matrix, and then the QR method to compute the singular values of the resultant bidiagonal form. Their method superceded a one-sided orthogonalization method given by Hestenes in 1958[10]. Hestenes' method is, however, easily adaptable to special purpose computations. It was suggested by Chartres (1962) [2] for a computer with a magnetic backing store, and was implemented on a mini-computer by Nash [12]. In this paper, we study the implementation of Hestenes' method on the ILLIAC IV computer and show that the method makes very efficient use of the parallel computing abilities of the ILLIAC machine.

Our new algorithm is likely to be highly beneficial for problems with large values of m and n that must be solved repeatedly. In fact, our project was launched because a seismologist at the United States Geological Survey wanted to solve his least squares problems on the ILLIAC computer. He was interested in the machine because of its large main memory and potential high execution speed. A nice exposition on least squares problems arising from earthquake studies is given in [17].

We are going to use the Frobenius norm for matrices, i.e.

$$\|A\| = \|A\|_F = \Big(\sum_{i,j} a_{ij}^2\Big)^{1/2} \quad \text{for } A \equiv (a_{ij}) ,$$

and the Euclidean norm for vectors, i.e.

$$\|\underline{x}\| = \|\underline{x}\|_2 = (\underline{x}^t\underline{x})^{1/2} .$$

2. The ILLIAC IV Computer

The ILLIAC IV computer was built by the Burroughs Corporation and is located at NASA/Ames Research Center, Moffett Field, California. The computer consists of 64 synchronous processing elements (PE's) under the direction of a single control unit (CU). Each PE has 2048 words of 64-bit memory with an access time of 188 nanoseconds, and is capable of performing a general floating-point operation in about 1.7 microseconds and a typical bookkeeping operation in about 1.2 microseconds. The PE instruction set is similar to that of conventional machines with two exceptions. First, each PE can communicate data to four other PE's through routing instructions. Second, the PE's can set their own mode registers to effectively disable or enable themselves. The CU takes about 0.7 microseconds to perform a bookkeeping operation.

The main memory of the ILLIAC is logically a 16-million word drum, which is divided into 52 bands and has a 40 millisecond rotation period. Data transfers to or from the PE memory are program initiated and are performed in blocks of 1024 words. The transfer time for 1024 words is about 66 microseconds; it takes about 4.2 milliseconds to refresh half the PE memory.

A floating-point number on the ILLIAC consists of a 1-bit sign, a 15-bit exponent to the radix 2, and a normalized 48-bit mantissa. The machine precision ϵ is thus about 3.55×10^{-15}. A fixed-point number has a 1-bit sign and a 48-bit mantissa.

3. Programming Languages for the ILLIAC

There are three languages available for programming the ILLIAC: its assembly language, ASK; a FORTRAN-like language, CFD [15]; and an ALGOL 60-like language, GLYPNIR [11]. Both CFD and

GLYPNIR do not hide the basic 64-wide architecture of the ILLIAC. We must restructure our data and algorithm so that the computation can be done in parallel in "strips" of width 64 or less.

Let us briefly describe the data declarations in GLYPNIR. The PE memory of the ILLIAC can be viewed as a two-dimensional structure where each word can be addressed by an ordered pair which specifies the PE memory module and the address within that module. A group of 64 words, each in a different module but each having the same address within its module, is called a superword or sword. We can divide the variable types in GLYPNIR into two major categories. The first represents words or vectors of words; they are called the CU variables. The second represents swords or vectors of swords; they are called the PE variables. There are also the Boolean variables and the so-called ADB variables.

A sword vector of length n represents an indexable vector of swords. It is thus in some sense an n x 64 array. A GLYPNIR program cannot directly handle two-dimensional arrays whose row and column dimensions exceed 64.

4. A Row Orthogonalization Method

There are three reasons why the standard SVD method of Golub and others (see [6] and [8]) may be undesirable on a parallel processor. First, although the Householder transformation is inherently parallel, the effective vector length decreases at each step, causing inefficiencies.

Second, the parallel QR method of Sameh and Kuck may be numerically unstable (see [9] and [14]). In contrast, the one-sided orthogonalization method of Hestenes [10] is easily adaptable to computation on a parallel machine. Third, data movement across the PE memories can be very expensive. The Hestenes method requires very little communication among the PE's.

The method of Hestenes consists of generating an orthogonal matrix V such that the non-null column vectors of the matrix

$$H = AV$$

are mutually orthogonal and non-increasing in norm. The nonzero columns of H are then normalized so that

$$H = (U_r|0) \left(\begin{array}{c|c} \Sigma_r & 0 \\ \hline 0 & 0 \end{array} \right)$$

with

$$U_r^t U_r = I_r \qquad \text{and} \qquad \Sigma_r = \text{diag}(\sigma_1, \ldots, \sigma_r).$$

Consequently,

$$A = U_r \Sigma_r V_r^t , \tag{1.4}$$

where

V_r is an $n \times r$ matrix consisting of the first r columns of V.

Nash [12] followed Hestenes' approach, but Chartres [2] chose to orthogonalize the rows of the given matrix A. We have decided on the row orthogonalization scheme, for it is easily adaptable to solving overdetermined linear equations.

We aim to generate an orthogonal matrix U^t so that the non-null row vectors of the matrix

$$K = U^t A$$

are mutually orthogonal and non-increasing in norm. We then normalize the nonzero rows of K to obtain

$$K = \left(\begin{array}{c|c} \Sigma_r & 0 \\ \hline 0 & 0 \end{array}\right) \begin{pmatrix} V_r^t \\ \hline 0 \end{pmatrix}$$

with

$$\Sigma_r = \mathrm{diag}(\sigma_1, \ldots, \sigma_r) \quad \text{and} \quad V_r^t V_r = I_r .$$

It follows that

$$A = U_r \Sigma_r V_r^t , \tag{1.4}$$

where

U_r is an $m \times r$ matrix consisting of the first r columns of U.

We are going to construct the matrix U as a product of plane rotations. Let us write the matrix A as

$$A = \begin{pmatrix} \underset{\sim}{a}_1^t \\ \underset{\sim}{a}_2^t \\ \vdots \\ \underset{\sim}{a}_m^t \end{pmatrix} , \tag{4.1}$$

where

$\underset{\sim}{a}_i^t$ is a $1 \times n$ row vector, for $i = 1, 2, \ldots, m$.

Given any two rows $\underset{\sim}{a}_i^t$ and $\underset{\sim}{a}_j^t$, with $i < j$, we would consider them orthogonal if

$$\|a_i\| < \epsilon \|A\| \qquad \text{or} \qquad \|\underset{\sim}{a}_j\| < \epsilon \|A\|, \tag{4.2}$$

where ϵ is the machine precision, or if

$$\frac{\underset{\sim}{a}_i^t \underset{\sim}{a}_j}{\|\underset{\sim}{a}_i\| \, \|\underset{\sim}{a}_j\|} < \tau, \tag{4.3}$$

where τ is a previously chosen tolerance. We do not transform orthogonal rows, but would permute them if

$$\|\underset{\sim}{a}_i\| < \|\underset{\sim}{a}_j\| .$$

Suppose now that the two given rows do not satisfy the orthogonality condition (4.2) or (4.3). Let us consider the action of a plane rotation:

$$\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} \underset{\sim}{a}_i^t \\ \underset{\sim}{a}_j^t \end{pmatrix} = \begin{pmatrix} \underset{\sim}{\hat{a}}_i^t \\ \underset{\sim}{\hat{a}}_j^t \end{pmatrix} . \tag{4.4}$$

The idea is to choose φ such that

$$\underset{\sim}{\hat{a}}_i^t \underset{\sim}{\hat{a}}_j = 0 \qquad \text{and} \qquad \|\underset{\sim}{\hat{a}}_i\| > \|\underset{\sim}{\hat{a}}_j\| .$$

The second condition ensures that the computation always proceeds towards an ordering of row norms. Following Nash [12], we let

$$\alpha = 2\underset{\sim}{a}_i^t\underset{\sim}{a}_j$$

$$\beta = \|\underset{\sim}{a}_i\|^2 - \|\underset{\sim}{a}_j\|^2 , \qquad (4.5)$$

and

$$\gamma = (\alpha^2 + \beta^2)^{1/2} .$$

Note that γ is positive since α is nonzero. Then, if β is positive we compute

$$\cos\varphi = \left(\frac{\gamma + \beta}{2\gamma}\right)^{1/2} \quad \text{and} \quad \sin\varphi = \frac{\alpha}{2\gamma\cos\varphi} , \qquad (4.6)$$

otherwise we let

$$\sin\varphi = \left(\frac{\gamma - \beta}{2\gamma}\right)^{1/2} \quad \text{and} \quad \cos\varphi = \frac{\alpha}{2\gamma\sin\varphi} . \qquad (4.7)$$

In (4.4), we could use the Fast Givens transformations (see [4]) which requires only 2n multiplications, an apparent 50% work reduction. But a heavy overhead in maintaining the scaling factors eats up the savings unless the row length n is moderately large [16].

As in the traditional Jacobi algorithm, the plane rotations are performed in a set sequence called a sweep, which consists of the [m(m-1)]/2 plane rotations on the row pairs

(1,2), (1,3), ... , (1,m), (2,3), ..., (2,m), (3,4), ..., (m-1,m).

The iterative procedure terminates if one complete sweep occurs in which all rows are orthogonal and no rows are interchanged.

Our orthogonalization method is in essence the Jacobi method implicitly applied to the matrix AA^t to compute its eigenvalues. We can refer to the literature [18] for the convergence properties of our method. We see that the convergence is quadratic and takes the order of 6 to 10 sweeps, i.e. from $3m^2$ to $5m^2$ plane rotations (see [13]).

We now present our method in its entirety. Two Boolean variables are introduced:

withu : true if matrix U is desired, false otherwise.

withv : true if matrix V is desired, false otherwise.

We make the arbitrary choice that

$$\tau = 10^{-12} .$$

Recall that

$$\varepsilon = 3.55 \times 10^{-15}$$

is the precision of the ILLIAC machine. The matrices U and V are written as

$$U = (\underset{\sim}{u}_1, \underset{\sim}{u}_2, \ldots, \underset{\sim}{u}_m) , \tag{4.8}$$

and

$$V = (\underset{\sim}{v}_1, \underset{\sim}{v}_2, \ldots, \underset{\sim}{v}_n) .$$

ALGORITHM 1 (SVD).

I. Initialize:

(1) Let

$$\varepsilon^2 := (3.55 \times 10^{-15})^2,$$
$$\tau^2 := 10^{-24},$$

and

$$c := 0 .$$

(2) For i:= 1,2,...,m do

compute $\|\underset{\sim}{a}_i\|^2$.

(3) Compute $\delta^2 := \varepsilon^2 \|A\|^2$.

(4) If (withu) then let U := I.

II. Repeat until $c = [m(m-1)]/2$:

(1) Let $c := 0$

(2) For $(i,j) := (1,2),(1,3),\ldots,(1,m),(2,3),\ldots,(2,m),$
$(3,4),\ldots,(3,m),\ldots,(m-1,m)$ do

If $\|\underset{\sim}{a}_j\|^2 < \delta^2$ then

(a) Let $c := c + 1$.

Else if $\|\underset{\sim}{a}_i\|^2 < \delta^2$ then

(a) Exchange $\underset{\sim}{a}_i$ and $\underset{\sim}{a}_j$.

(b) If (withu) then exchange $\underset{\sim}{u}_i$ and $\underset{\sim}{u}_j$.

Else if $\left(\dfrac{\underset{\sim}{a}_i^t \underset{\sim}{a}_j}{\|\underset{\sim}{a}_i\| \|\underset{\sim}{a}_j\|}\right)^2 < \tau^2$

then

(a) If $\|\underset{\sim}{a}_i\|^2 < \|\underset{\sim}{a}_j\|^2$ then

(i) Exchange $\underset{\sim}{a}_i$ and $\underset{\sim}{a}_j$.

(ii) If (withu) then exchange $\underset{\sim}{u}_i$ and $\underset{\sim}{u}_j$.

Else

(a) Compute

$$\alpha := 2\underset{\sim}{a}_i^t \underset{\sim}{a}_j$$

$$\beta := \|\underset{\sim}{a}_i\|^2 - \|\underset{\sim}{a}_j\|^2 ,$$

and

$$\gamma := (\alpha^2 + \beta^2)^{1/2}.$$

(b) If $\beta > 0$ then

(i) Compute

$$\cos\varphi := \left(\frac{\gamma + \beta}{2\gamma}\right)^{1/2},$$

and

$$\sin\varphi := \frac{\alpha}{2\gamma\cos\varphi}.$$

Else

(i) Compute

$$\sin\varphi := \left(\frac{\gamma - \beta}{2\gamma}\right)^{1/2},$$

and

$$\cos\varphi := \frac{\alpha}{2\gamma\sin\varphi}.$$

(c) Compute

$$\underset{\sim}{w} := \cos\varphi \cdot \underset{\sim}{a}_i + \sin\varphi \cdot \underset{\sim}{a}_j,$$

$$\underset{\sim}{a}_j := -\sin\varphi \cdot \underset{\sim}{a}_i + \cos\varphi \cdot \underset{\sim}{a}_j,$$

and

$$\underset{\sim}{a}_i := \underset{\sim}{w}.$$

(d) Update

$$||\underset{\sim}{w}||^2 := (\cos\varphi)^2 ||\underset{\sim}{a}_i||^2 + (\sin\varphi)^2 ||\underset{\sim}{a}_j||^2 + \alpha\cos\varphi\sin\varphi$$

$$||\underset{\sim}{a}_j||^2 := (\sin\varphi)^2 ||\underset{\sim}{a}_i||^2 + (\cos\varphi)^2 ||\underset{\sim}{a}_j||^2 - \alpha\cos\varphi\sin\varphi$$

and

$$||\underset{\sim}{a}_i||^2 := ||\underset{\sim}{w}||^2.$$

(e) If (withu) then

$$\underset{\sim}{z} := \cos\varphi \cdot \underset{\sim}{u}_i + \sin\varphi \cdot \underset{\sim}{u}_j,$$

$$\underset{\sim}{u}_j := -\sin\varphi \cdot \underset{\sim}{u}_i + \cos\varphi \cdot \underset{\sim}{u}_j,$$

and

$$\underset{\sim}{u}_i := \underset{\sim}{z}.$$

III. Compute singular values:

(1) Let $i := 1$.

(2) Repeat until $i > n$ or $\|\underset{\sim}{a}_i\| < \delta$:

(a) Let $\sigma_i := \|\underset{\sim}{a}_i\|$.

(b) If (withv) then $\underset{\sim}{v}_i := \dfrac{\underset{\sim}{a}_i}{\|\underset{\sim}{a}_i\|}$.

(c) Let $i := i + 1$.

(3) Let $r := i - 1$.

We wish to compare the required work of Algorithm 1 and the Golub-Reinsch method [8]. One sweep of Algorithm 1 takes about $(5n + 4m)m^2/2$ multiplications if the U matrix is desired, and $5nm^2/2$ multiplications otherwise. We assume that

$$m \leq n ,$$

for we can compute the SVD of A^t if $m > n$. We further suppose that our Jacobi-like method takes 8 sweeps to converge and that only two QR steps are required per singular value for the Golub-Reinsch algorithm (cf. [1]). The following table gives the number of multiplications required by the two methods in four different cases.

Matrices Desired	Algorithm 1	Golub-Reinsch
U_r, Σ_r, V_r	$20m^2n + 16m^3$	$7m^2n + 11m^3/3$
U_r, Σ_r	$20m^2n + 16m^3$	$2m^2n + 4m^3$
Σ_r, V_r	$20n^2n$	$7m^2n - m^3$
Σ_r	$20m^2n$	$2m^2n - 2m^3/3$

We see that Algorithm 1 is about three times slower than the standard SVD algorithm in computing the full singular value decomposition. However, the special architecture of the ILLIAC IV computer can reduce the number of required multiplications by an asymptotic factor of 64. Our Jacobi-like algorithm is therefore very efficient on a parallel computer. We should mention that Chan [1] described a modified Golub-Reinsch algorithm that could save up to 50% of machine execution time if $m \ll n$.

5. Least Squares Solutions

We have defined the singular value decomposition of an m x n matrix A as

$$A = U\Sigma V^t \tag{1.1}$$

Its pseudo-inverse is an n x m matrix A^+ given by

$$A^+ = V\Omega U^t , \tag{5.1}$$

where

$$\Omega = \left(\begin{array}{c|c} \begin{matrix} \sigma_1^{-1} & & \\ & \ddots & \\ & & \sigma_r^{-1} \end{matrix} & \begin{matrix} 0 & \\ & \\ & 0 \end{matrix} \\ \hline \begin{matrix} 0 & \\ & \\ & 0 \end{matrix} & \begin{matrix} 0 & \\ & \\ & 0 \end{matrix} \end{array}\right) . \tag{5.3}$$

An important application of the pseudoinverse is in solving overdetermined systems of linear equations

$$AX = B \,, \tag{5.3}$$

where B is a given $m \times s$ matrix and

$$m \geq n \,. \tag{5.4}$$

We seek an $n \times s$ matrix X such that

$$\|B - AX\| = \min \,. \tag{5.5}$$

The solution matrix X is not unique unless the matrix A is of full rank. We therefore impose the condition that we want the matrix $\hat{X}$ of minimum norm in the solution space. It is well known (see, e.g., [6]) that $\hat{X}$ is unique and is given by

$$\hat{X} = A^{+}B \,. \tag{5.6}$$

Thus, we have that

$$\hat{X} = V\Omega C \,, \tag{5.7}$$

where

$$C = U^{t}B \,.$$

The matrix C can be generated by applying to the rows of B those plane rotations that we use to orthogonalize the rows of A. It is unnecessary to accumulate the plane rotations.

We now present an algorithm based on Algorithm SVD for computing the minimum norm solution to the overdetermined system (5.3). There is an input parameter "cutoff." Our method sets to zero all those singular values of A that are smaller than cutoff $\cdot \|A\|$. The i row vector $\underset{\sim}{b}_i^t$ denotes the i-th row of B, for $i = 1,2,\ldots,m$.

ALGORITHM 2. (MINFIT)

I. Initialize:

(1) Let

$$\epsilon^2 := (0.55 \times 10^{-15})^2,$$

$$\tau^2 := 10^{-24}$$

and

$$c := 0 .$$

(2) For i:= 1, 2, ..., n do

compute $\| \underset{\sim}{a}_i \|^2$

(3) Compute $\delta^2 := \epsilon^2 \| A \|^2$.

II. Repeat until $c = [n(n-1)]/2$:

(1) Let $c := 0$.

(2) For $(i,j) := (1,2),(1,3),\ldots,(1,n),(2,3),\ldots,(2,n),$

$(3,4),\ldots,(3,n),\ldots,(n-1,n)$ do

If $\|\underset{\sim}{a}_j\|^2 < \delta^2$ then

(a) Let $c := c + 1$.

Else if $\|\underset{\sim}{a}_i\|^2 < \delta^2$ then

(a) Exchange $\underset{\sim}{a}_i$ and $\underset{\sim}{a}_j$.

(b) Exchange $\underset{\sim}{b}_i$ and $\underset{\sim}{b}_j$.

Else if

$$\left(\frac{\underset{\sim}{a}_i^t \underset{\sim}{a}_j}{\|\underset{\sim}{a}_i\| \|\underset{\sim}{a}_j\|} \right)^2 < \tau^2$$

then

(a) If $\|\underset{\sim}{a}_i\|^2 < \|\underset{\sim}{a}_j\|^2$ then

(i) Exchange $\underset{\sim}{a}_i$ and $\underset{\sim}{a}_j$.

(ii) Exchange $\underset{\sim}{b}_i$ and $\underset{\sim}{b}_j$.

Else

(a) Compute

$$\alpha := \underset{\sim}{a}_i^t \underset{\sim}{a}_j$$

$$\beta := \|\underset{\sim}{a}_i\|^2 - \|\underset{\sim}{a}_j\|^2 ,$$

and

$$\gamma := (\alpha^2 + \beta^2)^{1/2}$$

(b) If $\beta > 0$ then

(i) Compute

$$\cos\varphi := \left(\frac{\gamma + \beta}{2\gamma}\right)^{1/2}$$

and

$$\sin\varphi := \frac{\alpha}{2\gamma\cos\varphi}.$$

Else

(i) Compute

$$\sin\varphi := \left(\frac{\gamma - \beta}{2\gamma}\right)^{1/2}$$

and

$$\cos\varphi := \frac{\alpha}{2\gamma\sin\varphi}.$$

(c) Compute

$$\underset{\sim}{w} := \cos\varphi \cdot \underset{\sim}{a}_i + \sin\varphi \cdot \underset{\sim}{a}_j,$$

$$\underset{\sim}{a}_j := -\sin\varphi \cdot \underset{\sim}{a}_i + \cos\varphi \cdot \underset{\sim}{a}_j,$$

and $\underset{\sim}{a}_i := \underset{\sim}{w}$.

(d) Update $\|\underset{\sim}{w}\|^2 := (\cos\varphi)^2 \|\underset{\sim}{a}_i\|^2 + (\sin\varphi)^2 \|\underset{\sim}{a}_j\|^2 + \alpha\cos\varphi\sin\varphi$

$$\|\underset{\sim}{a}_j\|^2 := (\sin\varphi)^2 \|\underset{\sim}{a}_i\|^2 + (\cos\varphi)^2 \|\underset{\sim}{a}_j\|^2 - \alpha\cos\varphi\sin\varphi$$

and

$$\|\underset{\sim}{a}_i\|^2 := \|\underset{\sim}{w}\|^2.$$

(e) Compute

$$\underset{\sim}{z} := \cos\varphi \cdot \underset{\sim}{b}_i + \sin\varphi \cdot \underset{\sim}{b}_j,$$

$$\underset{\sim}{b}_j := -\sin\varphi \cdot \underset{\sim}{b}_i + \cos\varphi \cdot \underset{\sim}{b}_j,$$

and

$$\underset{\sim}{b}_i := \underset{\sim}{z}.$$

III. Compute least squares solution:

(1) Let

$$V := 0,$$

$$Y := 0,$$

and

$$i := 1.$$

(2) Repeat until $i > n$ or $\|\underset{\sim}{a}_i\| < \text{cutoff} \cdot \|A\|$:

(a) Compute

$$\underset{\sim}{y}_i := \frac{\underset{\sim}{a}_i}{\|\underset{\sim}{a}_i\|}$$

and

$$\underset{\sim}{b}_i := \frac{\underset{\sim}{b}_i}{\|\underset{\sim}{a}_i\|} .$$

(b) Let $i := i + 1$.

(3) Let $r := i - 1$.

(4) Compute $X := \sum_{i=1}^{r} \underset{\sim}{y}_i \, \underset{\sim}{b}_i^t$.

One sweep of Algorithm 2 takes about $5m^2n/2$ multiplications. We neglect the terms involving s because $s \ll n$ in most applications. With the same assumptions on convergence rates as in the last section, the required work for our MINFIT algorithm is about $20m^2n$ multiplications while that for a similar method based on the Golub-Reinsch algorithm (see [8]) is about $2mn^2 + 4n^3$ multiplications. If $m \gg n$, it saves work to first reduce the regression matrix A to upper triangular form using Householder transformations, before applying the MINFIT algorithm (cf. [1]). Such a two-stage scheme requires about $3mn^2 + 11n^3/3$ multiplications. Fortunately, the parallel computing abilities of the ILLIAC machine reduce the work of our algorithm by an asymptotic factor of 64. Thus, our MINFIT algorithm is an effective solver for least squares problems.

6. Data Structures.

Let us first assume that $n \leq 64$. As our algorithms access A by rows, we lay out the rows of the matrix across the processing elements of the ILLIAC. We thus represent the matrix by a sword vector A[*] of order m.

We work with the rows of A to compute (a) the pairwise inner product, and (b) the new rows after a plane rotation. The GLYPNIR language provides a built-in function ROWSUM that sums the 64 numbers of a sword in 6 additions. The GLYPNIR expression

```
ROWSUM(A[I] * A[J])
```

computes the inner product of the i-th and j-th rows of A. If $n < 64$, we must disable the last $(64 - n)$ processing elements when we call the ROWSUM function. An alternative is to apply our algorithms to an $m \times 64$ matrix $\hat{A}$, given by

$$\hat{A} = (A|0) .$$

The following lines of GLYPNIR code compute the new i-th and j-th rows of A at the end of a plane rotation:

```
T    := A[I] * COSPHI + A[J] * SINPHI ;
A[J] := -A[I] * SINPHI + A[J] * COSPHI ;
A[I] := T ;
```

where T is a sword used for temporary storage.

We now consider the case when $n > 64$. Let

$$\ell = \left\lceil \frac{n}{64} \right\rceil ,$$

i.e. ℓ equals the smallest integer $\geq n/64$. We construct an $m \times 64\ell$ matrix $\hat{A}$, given by

$$\hat{A} = (A|0) \ .$$

The rows of $\hat{A}$ are then divided into ℓ equal segments:

$$\hat{A} = (A_1|A_2|\cdots|A_\ell) \ .$$

Thus, we may represent the matrix $\hat{A}$ by the ℓ sword vectors A1[*], ... , Aℓ[*], each of order m.

The GLYPNIR expression

```
ROWSUM(A1[I] * A1[J] + ··· + Aℓ[I] * Aℓ[J])
```

computes the inner product of the i-th and j-th rows of the matrix A. Plane rotations are applied to individual segments of the rows. For example, we may write the ℓ lines of code

```
A1[J] := -A1[I] * SINPHI + A1[J] * COSPHI ;
A2[J] := -A2[I] * SINPHI + A2[J] * COSPHI ;
   ⋮
Aℓ[J] := -Aℓ[I] * SINPHI + Aℓ[J] * COSPHI ;
```

to compute the new vector

$$\underset{\sim}{a}_j := -\underset{\sim}{a}_i \cdot \sin\varphi + \underset{\sim}{a}_j \cdot \cos\varphi \ .$$

A major shortcoming of this approach is that we must write a separate program for each value of ℓ. A better alternative is to store row i of A_j in sword A([i-1]·ℓ+j) [or sword A([j-1]·m+i)], and we have to write just one program.

Since the columns of the matrix U are transformed in the same manner as the rows of A, we lay out U so that its columns lie across the processing elements. Thus, we represent the matrix by a sword

vector U[*] of dimension m. For the two different cases of $m \leq 64$ and $m > 64$, we apply techniques similar to those discussed in the previous paragraphs.

The rows of the data matrix B are modified in an identical fashion as the rows of A. Therefore, we lay out the rows of B across the processing elements. The two cases of column dimension $s \leq 64$ and $s > 64$ for B are dealt with in the same manner as the corresponding cases for the matrix A.

The execution time of Algorithm MINFIT is independent of s, for $s \leq 64$. If s is much greater than m, then the execution time will be proportional to $\left\lceil \frac{s}{64} \right\rceil$.

7. Numerical Properties

Let us examine the question of numerical stability. An error analysis of the action of plane rotations on a matrix was given by Wilkinson in his classic text [19]. His error bounds were later improved by Gentleman [5]. We use their results to study the effects of the plane rotations in one sweep of our algorithm.

Let

$$M = \frac{1}{2} m(m-1) , \tag{7.1}$$

and let R_j represent the j-th plane rotation, for $j = 1,2,\ldots,M$. We can show that the computed matrix $\bar{A}_M$ after one sweep of rotations satisfies the inequality

$$\|\bar{A}_m - R_M R_{M-1} \cdots R_1 A\| \leq 2^{-48}(m+n-2)(1+2^{-48})^{m+n-2}\|A\| . \tag{7.2}$$

The right-hand side of the inequality (7.2) is an extreme upper bound. We expect the statistical distribution of the rounding errors to reduce the error to well below the level of the bound; for this reason alone, a factor of the order of $(m+n-2)^{1/2}$ in place of $(m+n-2)$ might be more realistic. We see that our algorithm is extremely stable.

As the matrix U is formed as a product of plane rotations, we examine here the deviation from orthogonality of such a product. Let $\bar{Q}_M$ represent the computed product of the plane rotations in one sweep. We have the inequality that

$$\|\bar{Q}_M - R_M R_{M-1} \cdots R_1\| \leq 2^{-48}\, m^{1/2}(m+n-2)(1+2^{-48})^{m+n-2} . \quad (7.3)$$

Again statistical consideration indicates that a factor of the order of $m^{1/4}(m+n-2)^{1/2}$ instead of $m^{1/2}(m+n-2)$ is probably more realistic. The matrix U is thus very close to an orthogonal matrix.

The tolerance τ controls the accuracy of the solution. At convergence of our algorithm, we have that

$$\|V_r^t V_r - I_r\| \leq r.\tau \quad (7.4)$$

Indeed, our numerical experiments show that the accuracy of the computed singular values and vectors of A is of the order of τ.

8. Test Results

We have written GLYPNIR programs implementing Algorithms SVD and MINFIT. Tests were carried out on the ILLIAC IV computer.

EXAMPLE 1 (see [8]).

We have chosen the following matrices:

$$A = \begin{bmatrix} 22 & 10 & 2 & 3 & 7 \\ 14 & 7 & 10 & 0 & 8 \\ -1 & 13 & -1 & -11 & 3 \\ -3 & -2 & 13 & -2 & 4 \\ 9 & 8 & 1 & -2 & 4 \\ 9 & 1 & -7 & 5 & -1 \\ 2 & -6 & 6 & 5 & 1 \\ 4 & 5 & 0 & -2 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 1 & 0 \\ 2 & -1 & 1 \\ 1 & 10 & 11 \\ 4 & 0 & 4 \\ 0 & -6 & -6 \\ -3 & 6 & 3 \\ 1 & 11 & 12 \\ 0 & -5 & -5 \end{bmatrix}.$$

The singular values of A are $\sqrt{1248}$, 20, $\sqrt{384}$, 0 and 0. Our SVD program computed those values to machine precision. The minimum norm solution to the overdetermined system

$$AX = B$$

is given by

$$X = \begin{pmatrix} -\frac{1}{12} & 0 & -\frac{1}{12} \\ 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} \\ -\frac{1}{12} & 0 & -\frac{1}{12} \\ \frac{1}{12} & 0 & \frac{1}{12} \end{pmatrix}.$$

Our MINFIT program returned a solution accurate to 14 decimal digits.

EXAMPLE 2 (see [6]).

Let us compute the full singular value decomposition of the n x n matrix

$$A = \begin{pmatrix} 1 & -1 & -1 & -1 & -1 & \cdots \\ & 1 & -1 & -1 & -1 & \cdots \\ & & 1 & -1 & -1 & \cdots \\ & & & 1 & -1 & \cdots \\ & & \bigcirc & & 1 & \cdots \\ & & & & & \cdots \end{pmatrix},$$

which is ill-conditioned as it has a very small singular value.

The matrix becomes singular if we add -2^{-n+2} to its (n,1) position.

We applied our SVD program to this choice of A for different values of n. For comparison, we have chosen the SVD subroutine in the EISPACK eigenvalue package from the Argonne National Laboratory [3]. The EISPACK routine implements the method of Golub and Reinsch [8] and has been coded for high execution efficiency. We applied the routine to the same matrix on an IBM 370/168 computer at the Stanford Linear Accelerator Center. The code was compiled by the FORTRAN H EXTENDED compiler with optimization level 2.

	ILLIAC IV		IBM 370/168	ILLIAC TIME
n	iter	time	time	IBM TIME
16	7	0.21	0.101	2.08
32	8	1.02	0.57	1.79
48	8	2.36	1.76	1.34
64	9	4.56	4.03	1.13
96	10	13.04	12.81	1.02
128	9	21.94	29.68	0.74

Unfortunately, our ILLIAC times are only estimates. The ILLIAC timing routine did not work for the whole summer of 1979. However, we have actual timings of an algorithm similar to Algorithm 1, and it requires $28\ m^2n + 16\ m^3$ multiplications per sweep. The ILLIAC times in the table are those actual timings multiplied by a factor of nine-elevenths.

It should be pointed out that the GLYPNIR compiler produces very inefficient code. We had the experience that a CFD program implementing the same algorithm gave a saving of 41% in execution time over the GLYPNIR co

We must admit that in most applications one is interested in mini ing the overall cost of solving a problem, rather than the time required solve it. However, it is not easy to determine the costs of solving the problem on the two machines. They both use complicated algorithms, takin into account factors such as storage, priority, time-of-the-day, etc. We can only say that for our examples one second of ILLIAC time is about ten times as expensive as one second of IBM time at the Stanford Linear Accel tor Center.

We were unable to run examples with larger values of n on the ILL due to storage limitations placed on our account by the ILLIAC-TENEX syst Nonetheless, we observe that our ILLIAC routine becomes more efficient relative to the EISPACK routine with increasing values of n. The executi time of the former is crudely proportional to

$$(\text{iter} \times \left\lceil \frac{n}{64} \right\rceil \times n^2),$$

while that of the latter to n^3. There is thus a good potential in matri computations of a parallel computer with many processors.

The ILLIAC IV computer is still the only machine of its kind. PHOENIX project has been proposed and it calls for building a parallel c consisting of 1024 processors. The cost was estimated at 35 million dol in 1977. It was suggested that such a machine could eliminate the need new wind tunnel at the NASA/AMES Research Center which might cost ten t as much.

REFERENCES

[1] Chan, T. F. C., "On computing the singular value decomposition," Report STAN-CS-77-588, Computer Science Dept., Stanford University (1977).

[2] Chartres, B.A., "Adaptation of the Jacobi method for a computer with magnetic-tape backing store," Computer J. 5 (1962), 51-60.

[3] Garbow, B.S., Boyle, J.M., Dongarra, J.J., and Moler, C.B., Matrix Eigensystem Routines--EISPACK Guide Extension Springer-Verlag, Berlin (1977).

[4] Gentleman, W.M., "Least squares computations by Givens transformations without square roots," J. Inst. Maths. Applics. 12 (1973), 329-336.

[5] Gentleman, W.M., "Error analysis of QR decompositions by Givens transformations," Lin. Alg. Applics. 10 (1975), 189-197.

[6] Golub, G.H., and Kahan, W., "Calculating the singular values and pseudo-inverse of a matrix,"J. SIAM Ser. B: Numer. Anal. 2 (1965), 205-224.

[7] Golub, G.H., and Luk, F.T., "Singular value decomposition: applications and computations," ARO Report 77-1, Transactions of the 22-nd Conference of Army Mathematicians (1977), 577-605.

[8] Golub, G.H., and Reinsch, C., "Singular value decomposition and least squares solutions," Numer. Math. 14 (1970), 403-420.

[9] Heller, D., "A survey of parallel algorithms in numerical linear algebra," SIAM Review 20 (1978), 740-777.

[10] Hestenes, M.R., "Inversion of matrices by biorthogonalization and related results," J. Soc. Indust. Appl. Math. 6 (1958), 51-90.

[11] Lawrie, D.H., Layman, T., Baer, D., and Randal, J.M., "GLYPNIR--a programming language for ILLIAC IV," Comm. ACM 18 (1975), 157-164.

[12] Nash, J.C., "A one-sided transformation method for the singular value decomposition and algebraic eigenproblem," Computer J. 18 (1975), 74-76.

[13] Rutishauser, H., "The Jacobi method for real symmetric matrices," Numer. Math. 9 (1966), 1-10.

[14] Sameh, A.H., and Kuck, D.J., "A parallel QR algorithm for symmetric tridiagonal matrices," IEEE Trans. Computers C-26 (1977), 147-153.

[15] Stevens, K.G., Jr., "CFD--a FORTRAN-like language for the ILLIAC IV," ACM Sigplan Notices 10 No. 3 (1975), 72-76.

[16] Stewart, G.W., private communication (1978).

[17] Wiggins, R.A., "The general linear inverse problem: implication of surface waves and free oscillations for earth structure," Reviews of Geophysics and Space Physics 10 (1972), 251-285.

[18] Wilkinson, J.H., "Note on the quadratic convergence of the cyclic Jacobi process," Numer. Math. 4 (1962), 296-300.

[19] Wilkinson, J.H., The Algebraic Eigenvalue Problem, Clarendon, Oxford (1965).

Acknowledgements

The author acknowledges the generous support of the Institute for Advanced Computation. He is grateful to Drs. David Stevenson and Harold Brown for their help in programming the ILLIAC IV, and to Professor Gene Golub, Professor James Wilkinson and especially the anonymous referee for their valuable suggestions, which have greatly improved the manuscript.