Faster SVD for Matrices With Small m/n

David Bau*

TR 94-1414
March 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* Thanks to Nick Trefethen for first suggesting to me that the flop count for bidiagonalization shouldn't have a corner at m/n = 5/3. This work was supported by an NSF Graduate Fellowship and by NSF Grant DMS-9116110 and DOE Grant DE-FG02-94ER25199.

Faster SVD for Matrices with Small m/n

David Bau, Cornell University*

The singular values of a matrix are conventionally computed using either the bidiagonalization algorithm by Golub and Reinsch (1970) when $m/n < 5/3$, or the algorithm by Lawson and Hanson (1974) and Chan (1982) when $m/n > 5/3$. However, there is an algorithm that is faster and that does not involve a discontinuous choice, as follows: in all cases, perform a QR factorization as in Lawson-Hanson-Chan, but rather than do this right at the beginning, do it after zeros have already been introduced in the first $j = 2n - m$ rows and columns.

The same technique applies when computing singular vectors, with one small modification. If left singular vectors are needed, the new algorithm becomes advantageous only when $m > 1.2661n$, and the best j in this case is $3n - m$.

The benefits of the new algorithm appear in terms of classical scalar floating-point operation counts; the effects of locality and parallelization are not considered in the analysis.

1 Bidiagonalization of Skinny Matrices

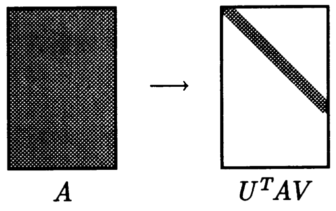
We seek a bidiagonalization of A, that is, a matrix

$$B = U^T A V \tag{1}$$

where U and V are orthogonal and $b_{ij} = 0$ for $j \neq i, i+1$.

Golub and Reinsch [GR 70] suggest that to bidiagonalize an $m \times n$ matrix, Householder reflections be applied alternately on the left and right so that, at the ith step, zeros are introduced into the $i+1, \ldots, m$ entries in the ith column and in the $i+2, \ldots, n$ entries in the ith row. The number of flops required (including additions) is $4mn^2 - \frac{4}{3}n^3$.

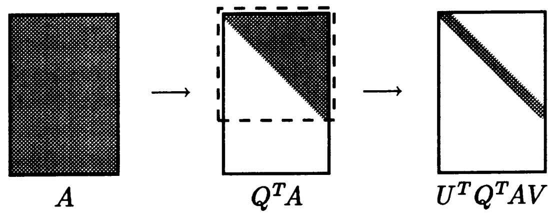
One-Phase Golub-Reinsch Bidiagonalization



*Thanks to Nick Trefethen for first suggesting to me that the flop count for bidiagonalization shouldn't have a corner at $m/n = 5/3$. This work was supported by an NSF Graduate Fellowship and by NSF Grant DMS-9116110 and DOE Grant DE-FG02-94ER25199.

Lawson and Hanson [LH 74, p.119] and also Chan [Chan 82] notice that if the ratio m/n is sufficiently large, it is cheaper to bidiagonalize in two phases. They first compute the QR factorization of the matrix, and then they apply the Golub-Reinsch procedure to bidiagonalize the resulting square R matrix. The QR factorization requires an additional $2mn^2 - \frac{2}{3}n^3$ flops, but the bidiagonalization of the remaining $n \times n$ matrix R uses only $\frac{8}{3}n^3$ flops, for a total of $2mn^2 + 2n^3$, cheaper than Golub-Reinsch if $m/n > \frac{5}{3}$.

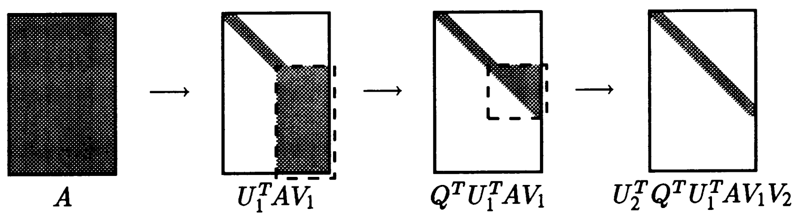
Two-Phase Lawson-Hanson-Chan Bidiagonalization



2 Bidiagonalization of Medium-Shaped Matrices

Extending the idea of Lawson and Hanson and Chan, we make the observation that, in the process of applying Golub-Reinsch bidiagonalization to any $m \times n$ matrix with $m > n$, the problem of bidiagonalizing skinny submatrices resurfaces. After the ith step of Golub-Reinsch, the remaining problem is to bidiagonalize the $(m-i) \times (n-i)$ lower-right submatrix, and this submatrix can become arbitrarily skinny. If the aspect ratio $(m-i)/(n-i)$ is sufficiently large, a QR factorization of the remaining submatrix is warranted to reduce it to a square matrix.

Three-Phase Bidiagonalization



The only remaining problem is to determine the best point at which the QR factorization should be applied. A brief calculation follows.

If the Golub-Reinsch algorithm is interrupted after j steps in order to apply a QR factorization before proceeding with the rest of the bidiagonalization, the total number of flops used is asymptotically

$$4mn^2 - \frac{4}{3}n^3 - 2(m-j)(n-j)^2 + \frac{10}{3}(n-j)^3. \tag{2}$$

We wish to find $0 \leq j \leq n$ that minimizes this formula. Local minimization yields the solution when $n \leq m \leq 2n$; for these matrices, the j we seek is

$$j = 2n - m. \tag{3}$$

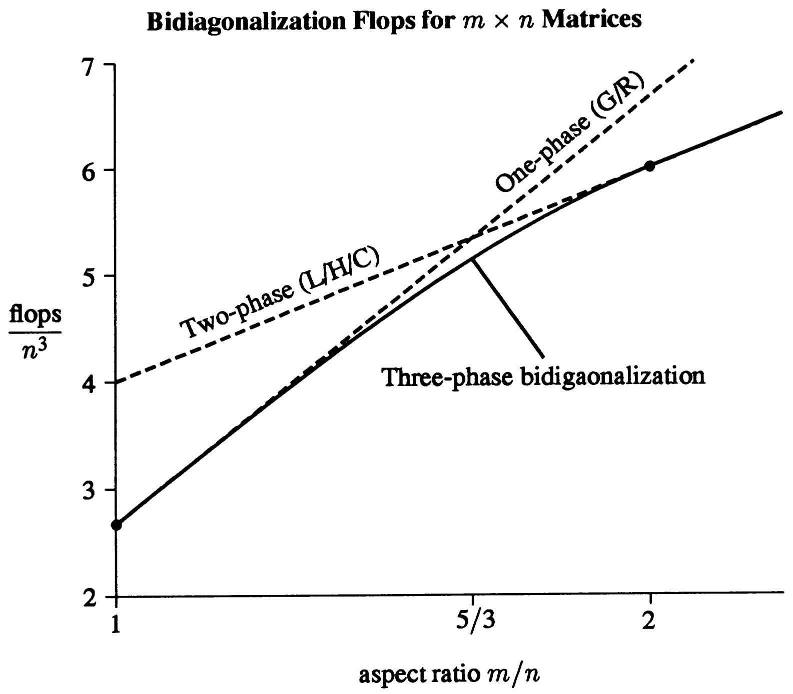
In other words, it is best to apply QR factorization at the point at which the remaining $(m-j) \times (n-j)$ submatrix has aspect ratio

$$\frac{m-j}{n-j} = \frac{2m-2n}{m-n} = 2. \tag{4}$$

Using this choice of j, we find that, for $n < m < 2n$, the number of flops required to compute a bidiagonalization is asymptotically

$$4mn^2 - \frac{4}{3}n^3 - \frac{2}{3}(m-n)^3. \tag{5}$$

For $m = n$ our algorithm reduces to Golub-Reinsch bidiagonalization. For $m \geq 2n$, we get $j = 0$, and it reduces to Lawson-Hanson-Chan bidiagonalization. In between, new algorithm is slightly cheaper than either of the other algorithms for matrices with $n < m < 2n$, at least in terms of classical, scalar floating-point operation counts. The new algorithm uses 3.7% fewer flops when $m/n = \frac{5}{3}$.



3 Computing Singular Values

Once a bidiagonalization is obtained, singular values are obtained by an iterative process related to the QR algorithm that applies Givens rotations to both sides of the matrix. The algorithm is described in [GVL 8.3.2]. After a $O(n^2)$ Givens rotations, the off-diagonal elements converge sufficiently to zero, and the singular values are left on the diagonal. For future reference, let C be the constant that gives sufficient convergence when Cn^2 rotations are applied on each side (typically $C \approx 2$).

The flop requirement of this process is only $O(n^2)$, so the work of the SVD algorithm is dominated by the $O(mn^2)$ flops of the bidiagonalization.

4 Computing Singular Vectors

If the singular vectors as well as the singular values need to be computed, the three-phase algorithm continues to be a cheaper alternative to the Golub-Reinsch and Lawson-Hanson-Chan algorithms in certain cases. The analysis is presented here.

4.1 Roster of Orthogonal Operations

The left and right singular vectors are computed by accumulating many small orthogonal operations. To analyze the cost of the accumulation, we begin by giving names to each elementary operation.

The first bidiagonalization phase operates by applying Householder reflections on the left and the right of A to obtain $U_1^T A V_1$. Explicitly,

$$U_1 = \prod_{i=1}^{j} U_i \tag{6}$$

$$V_1 = \prod_{i=1}^{j} V_i \tag{7}$$

where the applicaion of $U_i = U_i^T$ to the left of a dense $m \times n$ matrix requires $4n(m - i + 1)$ flops, and the applicaion of $V_i = V_i^T$ to the right requires $4m(n - i)$ flops.

The QR phase applies Householder reflections only on the left of the resulting matrix; let us write the reflections

$$Q = \prod_{i=j+1}^{n} U_i \tag{8}$$

as a natural extension of the sequence of U_i; each reflection still requires $4n(m - i + 1)$ flops to apply.

The secondary bidiagonalization phase again applies Householder reflections on the left and the right of the matrix; we write out the reflections as follows.

$$U_2 = \prod_{i=j+1}^{n} W_i \tag{9}$$

$$V_2 = \prod_{i=j+1}^{n} V_i \tag{10}$$

In comparison to the previous left reflections U_i, the reflections applied on the left now are relatively cheaper because they operate on fewer rows. Each reflection $W_i = W_i^T$ applied to the left of a dense $m \times n$ matrix requires only $4n(n - i + 1)$ flops. The reflections V_i applied on the right cost as much as before: $4n(m - i)$ flops.

Finally, reduction of the bidiagonal matrix B to a diagonal matrix Σ is accomplished by a sequence of Givens rotations on alternating sides of B. (Notice that, although the Givens operations must be realized as true Givens rotations when reducing B to Σ, they can be accumulated using fast Givens when constructing singular vectors, as long as attention is given to controlling the scaling of the fast Givens multipliers.)

If G_L^T is the accumulation of rotations on the left and G_R is the accumulation of rotations on the right, we have

$$\Sigma = G_L^T B G_R \tag{11}$$

$$G_L = \left(\prod_{i=1}^{Cn^2} Mi \right) D_L \tag{12}$$

$$G_R = \left(\prod_{i=1}^{Cn^2} Ni \right) D_R. \tag{13}$$

The cost of applying a single fast Givens transformation on the left of a dense $m \times n$ matrix is $4n$ flops; the cost of applying a fast Givens transformation on the right is $4m$ flops. Observe that for $m > n$, applying Givens on the left can be significantly cheaper than applying Givens on the right. This fact will impact the cost of accumulating left singular vectors.

Expanding B in full we have

$$\Sigma = U^T A V \tag{14}$$

$$= G_L^T U_2^T Q^T U_1^T A V_1 V_2 G_R \tag{15}$$

$$U = \prod_{i=1}^{j} U_i \prod_{i=j+1}^{n} U_i \prod_{i=j+1}^{n} W_i \prod_{i=1}^{Cn^2} M_i \; D_L \tag{16}$$

$$V = \prod_{i=j+1}^{n} V_i \prod_{i=1}^{j} V_i \prod_{i=1}^{n} N_i \; D_R. \tag{17}$$

4.2 First Method for Computing Left Singular Vectors

The left singular vectors are the first n columns of U, in other words, $\hat{U} = UI_mn$, where I_{mn} is the $m \times n$ principal submatrix of the identity. Perhaps the most natural way to compute this matrix is to multiply terms into I_{mn} starting from right to left. However, there are two other ways to compute this matrix that do better at exploting sparsity. The first method can be written by inserting parentheses into (16) as follows.

$$\hat{U} = \left(\left(\left(\prod_{i=1}^{j} U_i \left(\left(\prod_{i=j+1}^{n} U_i\right) \prod_{i=j+1}^{n} W_i\right)\right) \prod_{i=1}^{Cn^2} M_i\right) D_L\right) I_{mn}. \tag{18}$$

Let's begin with a breakdown of this approach. It consists of several steps.

1. The first n columns of $\prod_{i=j+1}^{n} U_i$ are accumulated by multiplying them into I_{mn}, beginning with the rightmost terms. This step requires $2(m-j)(n-j)^2 - (2/3)(n-j)^3$ flops.

2. The Householder reflections from the QR factorization $\prod_{i=j+1}^{n} W_i$ are accumulated on to the right side of result from the first step by multiplying the leftmost terms first. This step requires $2(m-j)(n-j)^2$ flops.

3. The Householder reflections of of $\prod_{i=1}^{j} U_i$ are accumulated on to the left side of the result from the second step by multiplying the rightmost terms first. This step requires $2mn^2 - 2(m-j)(n-j)^2 - (2/3)n^3 + (2/3)(n-j)^3$ flops.

4. The fast Givens operations are accumulated on to the result from the last step beginning with the leftmost terms. At the end, the diagonal scaling factor is multiplied in. The accumulation requires $4Cmn^2$ flops.

The total number of flops is

$$(4C+2)mn^2 + 2(m-j)(n-j)^2 - (2/3)n^3. \tag{19}$$

When $j = n$, this reduces to the $(4C+2)mn^2 - (2/3)n^3$, which is the cost of accumulating left singular vectors using the algorithm of Golub and Reinsch. In fact, this is the right way to use this method, because, as will be shown now, $j = n$ is the optimal choice of j.

The total cost of computing left singular vectors, including the bidiagonalization cost discussed earlier, is

$$(4C+6)mn^2 - 2n^3 + (10/3)(n-j)^3. \tag{20}$$

Differentiating and optimizing (the derivative is $-10(n-j)^2$), we find that $j = n$ is the optimal choice of j. In other words, the best strategy when using this method for accumulating left singular vectors is to stick with the Golub-Reinsch algorithm and never execute a QR factorization.

The best cost in flops with this method is therefore, as mentioned before,

$$(4C + 6)mn^2 - 2n^3 \tag{21}$$

This is the performance of the Golub-Reinsch algorithm. In a moment, however, we shall see that for matrices with m/n slightly larger than 1, there is a more efficient algorithm for computing left singular vectors, based on a different multiplication order.

4.3 Second Method for Computing Left Singular Vectors

The second method for computing the left singular vectors is merely based on accumulating the product (16) in another order. The method can be written as follows:

$$\hat{U} = \left(\prod_{i=1}^{j} U_i \left(\prod_{i=j+1}^{n} U_i\right)\right) \left(\left(\left(\prod_{i=j+1}^{n} W_i\right) \prod_{i=1}^{Cn^2} M_i\right) D_L\right) I_{mn}. \tag{22}$$

Following is an algorithmic breakdown of the method.

1. The first n columns of $\prod_{i=1}^{n} U_i$ are accumulated by multiplying the Householder reflections into I_{mn}, beginning with the rightmost terms. This requires a total of $2mn^2 - (2/3)n^3$ flops.

2. The $(n-j)$ nontrivial columns and rows of $\prod_{i=j+1}^{n} W_i$ are accumulated by multiplying the rightmost terms first into I_{nn}. This requires $(4/3)(n-j)^3$ flops.

3. The fast Givens operations are accumulated on to the result from step two, beginning with the leftmost terms (and, at the end, the diagonal scaling factor is multiplied in). On a dense starting matrix, the accumulation would require $4Cn^3$ flops. However, the matrix from step two is not dense; the first j rows have many zeros that are largely preserved by the particular Givens operations we are multiplying. Each sequence of Givens operations combines a column i with $i+1$, and they are applied a sequences of increasing i. After n Givens operations, at most one column and one subdiagonal element of the other sparse columns are filled in. The upshot is that the flop count is slightly lower than the dense case: the total accumulation requires $4Cn^3 - (2/3)j^3 - 2j^2(n-j)$ flops.

4. The $m \times n$ matrix from the first step is multiplied by the $m \times n$ matrix from the third step, using an elementwise matrix multiplication. The number of flops required is $2mn^2$.

The number of flops totals $4mn^2 + (4C - 2/3)n^3 - (2/3)j^3 - 2j^2(n-j) + (4/3)(n-j)^3$. When $j = 0$ this reduces to $(4C + 2/3)n^3 + 4mn^2$, which is the cost of accumulating left singular vectors using the Lawson-Hanson-Chan scheme.

Notice that, when m is large relative to n, the second method promises to be more efficient than the first because it avoids multiplying the Givens operations by too many rows. On the other hand, when m is very close to n, the first method will be more efficient because it avoids the extra step of an explicit matrix multiplication. The correct crossover point between the two methods will be calculated in a moment; first we finish the analysis of the second method.

The number of flops required to compute left singular vectors if the second method is use (including the cost of bidiagonalization) is

$$8mn^2+(4C-2)n^3-2(m-j)(n-j)^2+(14/3)(n-j)^3-2j^2(n-j)-(2/3)j^3. \quad (23)$$

Setting the derivative (which is $4(n-j)(m-3n+j)$) to zero, we obtain the following solution to j:

$$j = \begin{cases} 0 & \text{if } m \geq 3n \\ 3n-m & \text{if } m < 3n \end{cases} \quad (24)$$

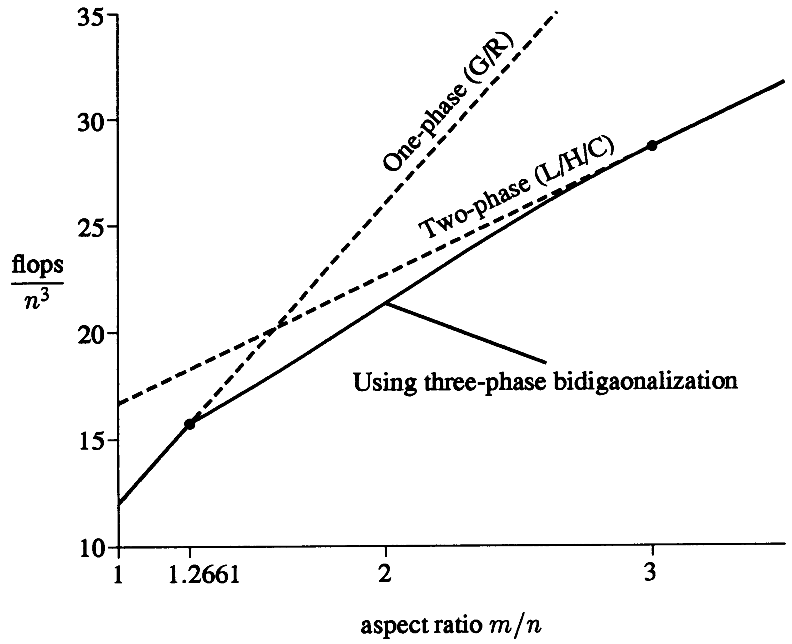
For $m \geq 3n$ the method reduces to the algorithm of Lawson, Hanson, and Chan. The cost in flops is $6mn^2 + (4C + 8/3)n^3$.

For $m < 3n$ we have $j = 3n - m$, so the cost in flops is

$$-\frac{2}{3}m^3 + 4m^2n + (4C + \frac{8}{3})n^3 \quad (25)$$

The formula for this flop count involves m^3 and m^2n terms which are not terribly enlightening; a plot of the flop count for matrices of various aspect ratios is more so.

Flops to Compute the Left Singular Vectors of $m \times n$ Matrices



The graph is drawn assuming that $C = 2$. The original Golub-Reinsch SVD is faster than the newly proposed method for m close to n; the crossover point where the flop counts (25) and (21) are equal is at $m/n = 1.2661$. At $m/n = \frac{19}{12}$, the new method requires 10.5% fewer flops than either the Lawson-Hanson-Chan or the Golub-Reinsch methods.

4.4 Right Singular Vectors

The right singular vectors are the columns of V; we compute them by evaluating the product (17) in the following order:

$$V = \left(\left(\prod_{i=j+1}^{n} V_i \left(\prod_{i=1}^{j} V_i\right)\right) \prod_{i=1}^{n} N_i\right) D_R \tag{26}$$

The multiplication goes in two phases:

1. The Householder reflections $\prod_{i=1}^{n} V_i$ are accumulated onto the left of I_{nn} beginning with the rightmost terms. This requires $(4/3)n^3$ flops.

2. The Givens rotations are accumulated onto the right of the resulting matrix, beginning with the leftmost terms. This requires $4Cn^3$ flops.

The total number of flops is

$$(4C + 4/3)n^3. \tag{27}$$

Notice that the acummulation of right singular vectors has no dependence on the choice of j. Therefore, it doesn't make any difference whether Golub-Reinsch, Lawson-Hanson-Chan, or the three-phase method is used: the cost of computing right singular vectors is the same in all three cases.

The choice between the various algorithms and choices of j should be made based on the cost of computing the bidiagonalization and the left singular vectors if they are needed. The best algorithm for the case where right singular vectors are not computed will remain the best algorithm for the case where right singular vectors are computed.

5 Summary of Results

The new algorithm for computing singular values provides a pleasing, unified framework in which both the Golub-Reinsch algorithm and the Lawson-Hanson-Chan algorithm can be described as special cases.

The new algorithm does not reduce smoothly to the Golub-Reinsch algorithm when left singular vectors are needed because Golub-Reinsch uses a method for accumulating left singular vectors that does not generalize well for different choices of j. On the other hand, when using the new algorithm to compute left singular vectors, a 10% reduction in floating point operations can be achieved for certain m/n, making the algorithm interesting from the point of view of performance.

The reduction in floating point operations of the new algorithm is not dramatic; with modern computer architectures, this degree of savings in reduced floating-point opereration counts may be dwarfed by the effects of data locality and parallelization. However, an adjustment of the parameter j that is chosen to minimize clock cycles rather than floating point operations will likely allow the new algorithm to achieve a performance advantage in practice.

SVD Flops for $m \times n$ Matrices

Result	G/R SVD	3-Phase SVD	L/H/C SVD
Σ	$4mn^2 - \frac{4}{3}n^3$	$4mn^2 - \frac{4}{3}n^3 - \frac{2}{3}(m-n)^3$	$2mn^2 + 2n^3$
Σ, V	$4mn^2 + 8n^3$	$4mn^2 + 8n^3 - \frac{2}{3}(m-n)^3$	$2mn^2 + \frac{34}{3}n^3$
$\Sigma, \hat{U}$	$14mn^2 - 2n^3$	$-\frac{2}{3}m^3 + 4m^2n + \frac{32}{3}n^3$	$6mn^2 + \frac{32}{3}n^3$
$\Sigma, \hat{U}, V$	$14mn^2 + \frac{22}{3}n^3$	$-\frac{2}{3}m^3 + 4m^2n + 20n^3$	$6mn^2 + 20n^3$

Aspect Ratios for the Three Algorithms

Result	G/R SVD	3-Phase SVD	L/H/C SVD
Σ or Σ, V	$m = n$	$n \le m \le 2n$	$2n \le m$
$\Sigma, \hat{U}$ or $\Sigma, \hat{U}, V$	$m \le 1.2661n$	$1.2661n \le m \le 3n$	$3n \le m$

References

[Chan 82] Chan, T.F. 1982. "An improved algorithm for computing the singular value decomposition," *ACM Trans. Math. Soft. 8*, 72–83.

[GR 70] Golub, G.H. and Reinsch, C. 1970. "Singular value decomposition and least squares solutions," *Numer. Math. 14*, 403–420.

[GVL 89] Golub, G.H. and Van Loan, C. 1989. *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD. (See pp. 236–239, 427–435.)

[LH 74] Lawson, C.L and Hanson, R.J. 1974. *Solving Least Squares Problems.* Prentice-Hall, Englewood Cliffs, NJ. (See pp. 110–119.)