SIAM J. MATRIX ANAL. APPL.
Vol. 26, No. 4, pp. 918–946

A RANK-REVEALING METHOD WITH UPDATING, DOWNDATING, AND APPLICATIONS*

T. Y. LI† AND ZHONGGANG ZENG‡

Abstract. A new rank-revealing method is proposed. For a given matrix and a threshold for near-zero singular values, by employing a globally convergent iterative scheme as well as a deflation technique the method calculates approximate singular values below the threshold one by one and returns the approximate rank of the matrix along with an orthonormal basis for the approximate null space. When a row or column is inserted or deleted, algorithms for updating/downdating the approximate rank and null space are straightforward, stable, and efficient. Numerical results exhibiting the advantages of our code over existing packages based on two-sided orthogonal rank-revealing decompositions are presented. Also presented are applications of the new algorithm in numerical computation of the polynomial GCD as well as identification of nonisolated zeros of polynomial systems.

Key words. matrix, rank, rank-revealing, null space, singular value, updating, downdating, GCD, nonisolated solution, polynomial system

AMS subject classifications. 12D05, 15A03, 15A18, 65F30, 65H10

DOI. 10.1137/S0895479803435282

1. Introduction.

The numerical rank determination arises in many applications that involve matrix computations, such as those discussed in a series of proceedings, *SVD and Signal Processing,* I, II, III [5, 13, 18]. While the singular value decomposition (SVD) is undoubtedly the most reliable method of determining the rank numerically, there are certain drawbacks. Among them, it is quite expensive when matrices become large. Moreover, it may not be able to take the matrix structure into account, and it is not easy to update or downdate when rows/columns are inserted or deleted. Alternative methods have been proposed, such as rank-revealing QR decomposition (RRQR) [2, 3, 4] and rank-revealing two-sided orthogonal decompositions (UTV, or URV/ULV) [6, 16, 17].

In this paper, a new rank-revealing algorithm is presented. For a given $m \times n$ matrix A, instead of calculating a decomposition that reveals the approximate rank, our method calculates the approximate rank and null space of A directly. We briefly outline the method as follows. Without loss of generality, we assume $m \geq n$, and let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ be singular values of A. Since the smallest singular value $\sigma_{\min} \equiv \sigma_n$ satisfies

$$\sigma_{\min} = \min_{\|\mathbf{x}\|_2=1} \| A\mathbf{x} \|_2,$$

the problem of finding $\sigma_{\min}$ can be converted to solving the overdetermined system

$$\begin{pmatrix} \tau \mathbf{x}^\top \\ A \end{pmatrix} \mathbf{x} = \begin{pmatrix} \tau \\ 0 \end{pmatrix}, \quad \text{where } \tau > \sigma_n, \tag{1.1}$$

*Received by the editors September 24, 2003; accepted for publication (in revised form) by H. A. van der Vorst August 5, 2004; published electronically May 6, 2005.
http://www.siam.org/journals/simax/26-4/43528.html

†Department of Mathematics, Michigan State University, East Lansing, MI 48824 (li@math.msu.edu). This author's research was supported in part by NSF grant DMS-0104009.

‡Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625 (zzeng@neiu.edu). This author's research was supported in part by NSF grant DMS-0412003.

for its least squares solution $\mathbf{x}$. For this purpose, one may use the Gauss–Newton iteration [3]

$$\begin{cases} \mathbf{x}_{j+1} = \mathbf{x}_j - \begin{pmatrix} 2\tau\mathbf{x}_j^\top \\ A \end{pmatrix}^+ \begin{pmatrix} \tau\mathbf{x}_j^\top\mathbf{x}_j - \tau \\ A\mathbf{x}_j \end{pmatrix}, \\ \varsigma_{j+1} = \dfrac{\| A\mathbf{x}_{j+1} \|_2}{\| \mathbf{x}_{j+1} \|_2}, \quad j = 0, 1, \ldots. \end{cases} \tag{1.2}$$

Here and throughout, for an arbitrary matrix B of full column rank, B^+ stands for its pseudo-inverse. It can be shown that (Lemma 4.1 in section 4) the Gauss–Newton iteration in (1.2) is essentially the inverse iteration on the matrix $A^\top A$ without undesirable matrix multiplication. The global convergence of the iteration is therefore warranted, and $(\varsigma_j, \mathbf{x}_j)$ will converge to the singular pair $(\sigma_n, \mathbf{v}_n)$. In this article, unless otherwise mentioned, we always use "singular vector" to represent the *right* singular vector. After $\sigma_n = \sigma_{\min}$ is calculated along with its associated singular vector $\mathbf{v}_n$, the matrix

$$A_\varrho = \begin{pmatrix} \varrho\mathbf{v}_n^\top \\ A \end{pmatrix}, \quad \varrho \in \mathbb{R}, \tag{1.3}$$

has the same set of singular values along with the associated singular vectors as those of A except the smallest singular value σ_n of A is replaced by the singular value $\sqrt{\varrho^2 + \sigma_n^2}$ of A_ϱ with associated singular vector $\mathbf{v}_n$ (Corollary 5.2 in section 5). Therefore, if we choose $\varrho = \|A\|_F$, then the replacement $\sqrt{\varrho^2 + \sigma_n^2}$ becomes the largest singular value of A_ϱ. In the meantime, the second smallest singular value σ_{n-1} of A becomes the smallest one of A_ϱ, and iteration (1.2) for finding the smallest singular pair of A can be applied to A_ϱ to calculate the singular pair $(\sigma_{n-1}, \mathbf{v}_{n-1})$ of A. This process can be continued recursively to calculate as many singular values of A as desired in ascending order $\sigma_n \le \sigma_{n-1} \le \cdots$, along with their associated singular vectors $\mathbf{v}_n, \mathbf{v}_{n-1}, \ldots$. Once σ_k is larger than the prescribed threshold $\theta > 0$, we will admit k as the approximate rank of A and the computed $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n$ as an orthonormal basis for the approximate null space of A.

Our method has been implemented as a MATLAB package RankRev and applied to many applications. In section 7 we present comprehensive numerical results of our code compared with UTV Tools [6] and the MATLAB SVD function. To calculate the approximate rank and null space of a given matrix that has a low rank deficit, our code can be 20 times faster than the full SVD when the matrix size becomes very large. Compared with UTV Tools, our method seems to be more robust and accurate in general, especially when the singular value gap at the rank threshold is relatively small. Moreover, row/column updating and downdating in our method, elaborated in section 8, are quite simple and straightforward. Separate numerical results are presented in section 8.5 comparing our method with UTV Tools in this respect. While UTV Tools may return incorrect ranks in certain difficult cases, our code always produces accurate results on all the matrices tested.

While rank-revealing has a large variety of applications, the development of our algorithm follows the needs of two important applications which emerged recently: a stable numerical algorithm for the computation of the GCD of univariate polynomials and the identification of nonisolated numerical solutions of polynomial systems. The details of those applications will be illustrated in section 9.

2. Notation, terminology, and definitions. The terms rank, nullity, and null space are used in the *exact* sense as in common linear algebra textbooks. In *numerical* linear algebra, the *approximate rank*, also known as the *numerical rank*, has a specific meaning given in Definition 2.1 below. Since the approximate rank, approximate null space, and approximate nullity are important concepts in our discussion, to be more clear and concise we shall use the specific terms *approxi-rank*, *approxi-null space*, and *approxi-nullity* for those notions. The usual notation $\text{rank}(A)$ remains the exact rank of a matrix A.

Throughout this paper, matrices are denoted by upper-case letters such as A, B, Q, R, etc. Lower-case boldface letters like $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{x}$ represent column vectors. The notation $(\cdot)^\top$ denotes the transpose of a matrix or vector $(\cdot)$, and vector spaces are denoted by a boldface upper-case letters like $\mathbf{W}$ with $\mathbf{W}^\perp$ denoting its orthogonal complement.

The definition of approxi-rank was first given by Golub, Klema, and Stewart [7]. We shall use a somewhat simplified definition.

DEFINITION 2.1. *For a given threshold $\theta > 0$, a matrix $A \in \mathbb{R}^{m\times n}$ has* approxi-rank *k within θ, denoted by $\text{rank}_\theta(A) = k$, if k is the smallest rank of all matrices within a 2-norm distance θ of A. Namely,*

$$k = \min_{\|A-B\|_2 \le \theta} \{\text{rank}(B)\}. \tag{2.1}$$

In this case, we also say the approxi-nullity *of A within θ is $n - k$.*

Notice that the exact rank of a matrix may be considered the approxi-rank of the matrix within zero.

The minimum in (2.1) is attainable [7, 12]: For $\theta > 0$, let $A = U\Sigma V^\top$ be the SVD of A with singular values satisfying

$$\sigma_1 \ge \cdots \sigma_k > \theta \ge \sigma_{k+1} \ge \cdots \ge \sigma_n. \tag{2.2}$$

Let $A_k = U\Sigma_k V^\top$ with $\Sigma_k = \text{diag}\{\sigma_1, \ldots, \sigma_k, 0, \ldots, 0\}$; then $\|A - A_k\|_2 = \sigma_{k+1} \le \theta$ and $\text{rank}(A_k) = \text{rank}_\theta(A) = k$ (see [7]). Moreover, A_k is nearest to A (with respect to the 2-norm) with rank k. In other words, for

$$\breve{\sigma} = \inf\big\{\mu \mid \text{rank}_\mu(A) = k\big\}, \tag{2.3}$$

we have $\|A - A_k\|_2 = \breve{\sigma}$. Let

$$\hat{\sigma} = \sup\big\{\eta \mid \text{rank}_\eta(A) = k\big\}.$$

We call the ratio $\gamma = \hat{\sigma}/\breve{\sigma}$ the *approxi-rank gap*. The size of this gap strongly influences the difficulties in achieving the accuracy of rank-revealing computation as shown in numerical examples in sections 7 and 8.5. If the singular values of A and the threshold θ satisfy (2.2), then clearly $\hat{\sigma} = \sigma_k$ and $\breve{\sigma} = \sigma_{k+1}$. When $\text{rank}_\theta(A) = k$, we called the null space of A_k the *approxi-null space* of A within θ since A_k is the nearest matrix to A with rank k. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the singular vectors of A (and A_k) associated with singular values σ_j, $j = 1, \ldots, n$; the approxi-null space of A is spanned by $\{\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n\}$. The approxi-nullity of A equals the dimension of the approxi-null space. Any vector $\mathbf{v}$ satisfying $\|A\mathbf{v}\|_2 \le \theta$ is called an *approxi-null vector* of A within θ.

3. The basic algorithm. As before, let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. We first establish the equivalence between finding the smallest singular value $\sigma_{\min} \equiv \sigma_n$ of A and solving the least squares problem of the quadratic system

$$\begin{pmatrix} \tau\mathbf{x}^\top \\ A \end{pmatrix} \mathbf{x} = \begin{pmatrix} \tau \\ 0 \end{pmatrix} \quad \text{with } \tau > \sigma_n. \tag{3.1}$$

PROPOSITION 3.1. *Let $\mathbf{u} \in \mathbb{R}^n$ be a vector satisfying*

$$\left\| \begin{pmatrix} \tau\mathbf{u}^\top \\ A \end{pmatrix} \mathbf{u} - \begin{pmatrix} \tau \\ 0 \end{pmatrix} \right\|_2^2 = \min_{x\in\mathbb{R}^n} \left\| \begin{pmatrix} \tau\mathbf{x}^\top \\ A \end{pmatrix} \mathbf{x} - \begin{pmatrix} \tau \\ 0 \end{pmatrix} \right\|_2^2$$

with a scaling factor $\tau > \sigma_n$. Then $\mathbf{u}$ is in the subspace $\mathbf{W}$ spanned by the singular vector(s) of A associated with the smallest singular value(s).

Proof. Let $A = U\Sigma V^\top$ be the SVD of A with orthogonal U and V. Let $\mathbf{z} = V^\top\mathbf{x}$ or $\mathbf{x} = V\mathbf{z}$, where $\mathbf{x} = (x_1, \ldots, x_n)^\top$ and $\mathbf{z} = (z_1, \ldots, z_n)^\top$. Let

$$f(x_1, \ldots, x_n) = \left\| \begin{pmatrix} \tau\mathbf{x}^\top \\ A \end{pmatrix} \mathbf{x} - \begin{pmatrix} \tau \\ 0 \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} \tau\mathbf{x}^\top\mathbf{x} - \tau \\ A\mathbf{x} \end{pmatrix} \right\|_2^2;$$

then

$$\begin{aligned} f(x_1, \ldots, x_n) &= \tau^2 \left(\mathbf{x}^\top\mathbf{x} - 1\right)^2 + \| A\mathbf{x} \|_2^2 = \tau^2 \left(\mathbf{z}^\top\mathbf{z} - 1\right)^2 + \| \Sigma\mathbf{z} \|_2^2 \\ &= \tau^2 \left(z_1^2 + \cdots + z_n^2 - 1\right)^2 + \sigma_1^2 z_1^2 + \cdots + \sigma_n^2 z_n^2 \equiv g(z_1, \ldots, z_n). \end{aligned}$$

Assume $g(\mathbf{z})$ reaches its minimum at $\mathbf{z} = \mathbf{y} \equiv (y_1, \ldots, y_n)^\top$. Then

$$\left.\frac{\partial g}{\partial z_j}\right|_{\mathbf{z}=\mathbf{y}} = 0, \quad j = 1, \ldots, n, \quad \text{i.e., } 4\tau^2 \left(y_1^2 + \cdots + y_n^2 - 1\right) y_j + 2\sigma_j^2 y_j = 0.$$

If $\mathbf{y} \neq 0$, let $J = \{1 \leq j \leq n \mid y_j \neq 0\}$. Then for $j \in J$, $\sigma_j^2 = 2\tau^2(1 - \sum_{l\in J} y_l^2)$. Hence, $\sigma_j^2 \leq 2\tau^2$, and $\sigma_j = \sigma$ for all $j \in J$ for certain $\sigma \in \{\sigma_1, \ldots, \sigma_n\}$ with $\sigma < \sqrt{2}\tau$. It follows that

$$\begin{aligned} g(y_1, \ldots, y_n) &= \tau^2 \left(\sum_{j\in J} y_j^2 - 1\right)^2 + \sum_{j\in J} \sigma_j^2 y_j^2 = \tau^2 \left(\sum_{j\in J} y_j^2 - 1\right)^2 + \sigma^2 \sum_{j\in J} y_j^2 \\ &= \tau^2 \left(\sum_{j\in J} y_j^2 - 1\right)^2 + \sigma^2 \left(\sum_{j\in J} y_j^2 - 1\right) + \sigma^2 \\ &= \tau^2 \left(-\frac{\sigma^2}{2\tau^2}\right)^2 + \sigma^2 \left(-\frac{\sigma^2}{2\tau^2}\right) + \sigma^2 = \sigma^2 - \frac{\sigma^4}{4\tau^2}. \end{aligned}$$

Therefore, the possible minimum values of $g(\mathbf{z})$ are $\sigma_j^2 - \frac{\sigma_j^4}{4\tau^2}$, $j = 1, \ldots, n$, and, perhaps, $g(0, \ldots, 0) = \tau^2$. Those values are all attainable since, for every singular pair $(\sigma_j, \mathbf{v}_j)$, letting $\mathbf{z} = sV^\top\mathbf{v}_j$ with $s^2 = 1 - \frac{\sigma_j^2}{2\tau^2}$ yields

$$g(\mathbf{z}) = \tau^2(s^2 - 1)^2 + \sigma_j^2 s^2 = \tau^2 \frac{\sigma_j^4}{4\tau^4} + \sigma_j^2 \left(1 - \frac{\sigma_j^2}{2\tau^2}\right) = \sigma_j^2 - \frac{\sigma_j^4}{4\tau^2}.$$

The function $h(\beta) = \beta^2 - \frac{\beta^4}{4\tau^2}$ is increasing in $[0, \tau]$, so

$$\min_{j=1,\ldots,n} \left\{ \sigma_j^2 - \frac{\sigma_j^4}{4\tau^2} \right\} = \sigma_n^2 - \frac{\sigma_n^4}{4\tau^2} \leq \sigma_n^2 < \tau^2$$

and $g(z_1, \ldots, z_n)$ reaches the minimum if $\sigma = \sigma_n$. Consequently, $\sigma_j = \sigma_n$ for all $j \in J$, and $\mathbf{u} = \sum_{j \in J} y_j \mathbf{v}_j$, where $\mathbf{v}_j$ is the singular vector associated with σ_j, $j = 1, \ldots, n$. □

Based on Proposition 3.1, the smallest singular value of A can be calculated via solving system (3.1) by the Gauss–Newton iteration [3]:

$$\begin{cases} \mathbf{x}_{j+1} = \mathbf{x}_j - \begin{pmatrix} 2\tau \mathbf{x}_j^\top \\ A \end{pmatrix}^+ \begin{pmatrix} \tau \mathbf{x}_j^\top \mathbf{x}_j - \tau \\ A\mathbf{x}_j \end{pmatrix}, \\ \varsigma_{j+1} = \dfrac{\| A\mathbf{x}_{j+1} \|_2}{\| \mathbf{x}_{j+1} \|_2}, \quad j = 0, 1, \ldots. \end{cases} \tag{3.2}$$

We shall prove in section 4 that the scalar sequence ς_j, $j = 1, 2, \ldots$, always converges to the smallest singular value $\sigma_{\min}$ of A. And if $\sigma_{\min}$ is a simple singular value, namely, $\sigma_{n-1} \neq \sigma_n$, then the vector sequences $\frac{1}{\varsigma_j} A\mathbf{x}_j$ and $\mathbf{x}_j$, $j = 1, 2, \ldots$, converge to the corresponding left and right singular vectors, respectively. When $\sigma_{\min}$ is not simple, ς_j still converges to $\sigma_{\min}$, while $\frac{1}{\varsigma_j} A\mathbf{x}_j$ and $\mathbf{x}_j$ converge into left and right singular subspaces associated with $\sigma_{\min}$.

When A has more than one zero singular values, the matrix $\left(\begin{smallmatrix} 2\tau \mathbf{x}_j^\top \\ A \end{smallmatrix}\right)$ becomes rank deficient and its pseudoinverse is undefined. While exact rank deficiency rarely happens in real computation, when it occurs, replacing A by $A+E$ with tiny $\|E\|_2$ will ensure the existence of the pseudoinverse. Such substitution has virtually no effect on the computing results. For details, see [8].

In the remainder of this paper, we shall frequently refer to the iteration (3.2) above as "applying the Gauss–Newton iteration on matrix A" for solving the least squares quadratic system in (3.1).

4. The convergence theory. The theory of the Gauss–Newton iteration warrants its local convergence under some restrictions, and the convergence rate is at least linear. The following lemma shows that the Gauss–Newton iteration (3.2) on the overdetermined quadratic system (3.1) is essentially the inverse iteration on the matrix $A^\top A$, and the convergence is therefore global.

LEMMA 4.1. *Let $A \in \mathbb{R}^{m \times n}$ be of full column rank, and let $\{\mathbf{x}_j\}$ be a vector sequence generated by iteration* (3.2). *Then there are constants c_j, $j = 0, 1, \ldots$, such that*

$$\mathbf{x}_{j+1} = c_j \left(A^\top A\right)^{-1} \mathbf{x}_j. \tag{4.1}$$

Proof. For simplicity, let $\mathbf{x}$ and $\mathbf{y}$ denote $\mathbf{x}_j$ and $\mathbf{x}_{j+1}$, respectively. Now,

$$\begin{aligned} \mathbf{y} &= \mathbf{x} - \begin{pmatrix} 2\tau \mathbf{x}^\top \\ A \end{pmatrix}^+ \begin{pmatrix} \tau \mathbf{x}^\top \mathbf{x} - \tau \\ A\mathbf{x} \end{pmatrix} \\ &= \mathbf{x} - \left[\left(2\tau \mathbf{x}, A^\top\right) \begin{pmatrix} 2\tau \mathbf{x}^\top \\ A \end{pmatrix} \right]^{-1} \left(2\tau \mathbf{x}, A^\top\right) \begin{pmatrix} \tau \mathbf{x}^\top \mathbf{x} - \tau \\ A\mathbf{x} \end{pmatrix} \\ &= \mathbf{x} - \left(4\tau^2 \mathbf{x}\mathbf{x}^\top + A^\top A\right)^{-1} \left[\left(2\tau^2 \mathbf{x}\mathbf{x}^\top + A^\top A\right) \mathbf{x} - 2\tau^2 \mathbf{x} \right] \end{aligned}$$

$$= \mathbf{x} - \left(4\tau^2\mathbf{x}\mathbf{x}^\top + A^\top A\right)^{-1}\left[\left(4\tau^2\mathbf{x}\mathbf{x}^\top + A^\top A\right)\mathbf{x} - 2\tau^2\mathbf{x}(\mathbf{x}^\top\mathbf{x}) - 2\tau^2\mathbf{x}\right]$$
$$= \left(4\tau^2\mathbf{x}\mathbf{x}^\top + A^\top A\right)^{-1} 2\tau^2\left(1 + \mathbf{x}^\top\mathbf{x}\right)\mathbf{x}.$$

This yields

$$\begin{aligned}\left(4\tau^2\mathbf{x}\mathbf{x}^\top + A^\top A\right)\mathbf{y} &= 2\tau^2\left(1 + \mathbf{x}^\top\mathbf{x}\right)\mathbf{x},\\ \left(A^\top A\right)\mathbf{y} &= \tau^2\left(2 + 2\mathbf{x}^\top\mathbf{x} - 4\mathbf{x}^\top\mathbf{y}\right)\mathbf{x},\\ \mathbf{y} &= 2\tau^2\left(1 + \mathbf{x}^\top\mathbf{x} - 2\mathbf{x}^\top\mathbf{y}\right)\left(A^\top A\right)^{-1}\mathbf{x}.\end{aligned} \tag{4.2}$$

So, $\mathbf{y} = c(A^\top A)^{-1}\mathbf{x}$ with $c = \frac{2\tau^2\,(1+\mathbf{x}^\top\mathbf{x})}{1+4\tau^2\,\mathbf{x}^\top(A^\top A)^{-1}\mathbf{x}}$. ∎

For a given matrix $A \in \mathbb{R}^{m\times n}$ and a threshold $\theta > 0$, we assume $\mathrm{rank}_\theta(A) = k$ and the singular values of A satisfy

$$\sigma_1 \geq \cdots \geq \sigma_k = \widehat{\sigma} > \theta \geq \breve{\sigma} = \sigma_{k+1} \geq \cdots \geq \sigma_n.$$

Then $\mathbf{W} = \mathrm{span}\{\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n\}$ is the approxi-null space of A, where $\mathbf{v}_j$ is the singular vector associated with σ_j for $j = k+1, \ldots, n$. The orthogonal complement $\mathbf{W}^\perp$ of $\mathbf{W}$ is $\mathrm{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$, and every vector $\mathbf{z} \in \mathbb{R}^n$ can be written as $\mathbf{z} = \widehat{\mathbf{z}} + \breve{\mathbf{z}}$ with $\widehat{\mathbf{z}} \in \mathbf{W}^\perp$ and $\breve{\mathbf{z}} \in \mathbf{W}$. We say a sequence of nonzero vectors $\{\mathbf{z}_j\}$ converges into $\mathbf{W}$ if

$$\lim_{j\to\infty} \frac{\| \widehat{\mathbf{z}}_j \|_2}{\| \breve{\mathbf{z}}_j \|_2} = 0, \quad \| \breve{\mathbf{z}}_j \|_2 \neq 0, \quad j = 0, 1, \ldots.$$

The approxi-rank depends critically on the threshold $\theta > 0$ one chooses, and the approxi-rank gap $\gamma = \widehat{\sigma}/\breve{\sigma}$ dictates its computing difficulties. The following proposition ensures that the vector sequence $\{\mathbf{x}_j\}$ generated by iteration (3.2) converges into the approxi-null space of A.

PROPOSITION 4.2. *Suppose $A \in \mathbb{R}^{m\times n}$ and $\mathrm{rank}_\theta(A) = k$ with a nontrivial approxi-null space $\mathbf{W}$ and approxi-rank gap γ. Then for $\mathbf{x}_0$ not orthogonal to $\mathbf{W}$, the iteration* (3.2) *generates a vector sequence $\{\mathbf{x}_j\}$ and a scalar sequence $\{\varsigma_j\}$, where $\mathbf{x}_j$ converges into $\mathbf{W}$ linearly in the sense*

$$\frac{\left\| \widehat{\mathbf{x}}_j \right\|_2}{\left\| \breve{\mathbf{x}}_j \right\|_2} \leq \left(\frac{1}{\gamma}\right)^{2j} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \breve{\mathbf{x}}_0 \right\|_2}, \quad j = 0, 1, \ldots, \tag{4.3}$$

and ς_j satisfies

$$\sigma_n \leq \varsigma_j \leq \breve{\sigma} + \left(\frac{1}{\gamma}\right)^{2j} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \breve{\mathbf{x}}_0 \right\|_2}\sigma_1. \tag{4.4}$$

Proof. Let $\mathbf{x}_0 = u_1\mathbf{v}_1 + \cdots + u_n\mathbf{v}_n$. From Lemma 4.1,

$$\mathbf{x}_1 = \eta\left(\frac{u_1}{\sigma_1^2}\mathbf{v}_1 + \cdots + \frac{u_n}{\sigma_n^2}\mathbf{v}_n\right)$$

for certain $\eta \in \mathbb{R}$, and with $\alpha = \frac{\eta}{\breve{\sigma}^2}$,

$$\mathbf{x}_1 = \alpha\left(\frac{\breve{\sigma}^2}{\sigma_1^2}u_1\mathbf{v}_1 + \cdots + \frac{\breve{\sigma}^2}{\sigma_n^2}u_n\mathbf{v}_n\right) = \widehat{\mathbf{x}}_1 + \breve{\mathbf{x}}_1,$$

where

$$\left\| \widehat{\mathbf{x}}_1 \right\|_2 = \left\| \alpha \left(\frac{\breve{\sigma}^2}{\sigma_1^2} u_1 \mathbf{v}_1 + \cdots + \frac{\breve{\sigma}^2}{\sigma_k^2} u_k \mathbf{v}_k \right) \right\|_2 \le |\alpha| \left(\frac{1}{\gamma} \right)^2 \left\| \widehat{\mathbf{x}}_0 \right\|_2,$$
$$\left\| \breve{\mathbf{x}}_1 \right\|_2 = \left\| \alpha \left(\frac{\breve{\sigma}^2}{\sigma_{k+1}^2} u_{k+1} \mathbf{v}_{k+1} + \cdots + \frac{\breve{\sigma}^2}{\sigma_n^2} u_n \mathbf{v}_n \right) \right\|_2 \ge |\alpha| \left\| \breve{\mathbf{x}}_0 \right\|_2.$$

Since $\| \breve{\mathbf{x}}_0 \|_2 \neq 0$, we have $\frac{\| \widehat{\mathbf{x}}_1 \|_2}{\| \breve{\mathbf{x}}_1 \|_2} \le (\frac{1}{\gamma})^2 \frac{\| \widehat{\mathbf{x}}_0 \|_2}{\| \breve{\mathbf{x}}_0 \|_2}$. By a simple induction, inequality (4.3) follows. For inequality (4.4),

$$\begin{aligned} \sigma_n &\le \frac{\| A\mathbf{x}_j \|_2}{\| \mathbf{x}_j \|_2} \le \frac{\left\| A \widehat{\mathbf{x}}_j \right\|_2}{\| \mathbf{x}_j \|_2} + \frac{\left\| A \breve{\mathbf{x}}_j \right\|_2}{\| \mathbf{x}_j \|_2} \\ &\le \left\| A \left(\frac{\widehat{\mathbf{x}}_j}{\left\| \breve{\mathbf{x}}_j \right\|_2} \right) \right\|_2 + \left\| A \left(\frac{\breve{\mathbf{x}}_j}{\left\| \breve{\mathbf{x}}_j \right\|_2} \right) \right\|_2 \\ &\le \sigma_1 \left[\left(\frac{1}{\gamma} \right)^{2j} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \breve{\mathbf{x}}_0 \right\|_2} \right] + \breve{\sigma}. \qquad \square \end{aligned}$$

As an important special case, if there is a significant gap in magnitude between σ_{n-1} and σ_n, then the iteration (3.2) converges to σ_n and its associated singular vector $\mathbf{v}_n$.

COROLLARY 4.3. *If $\sigma_{n-1} > \sigma_n$ and $\mathbf{x}_0$ satisfies $\mathbf{x}_0^\top \mathbf{v}_n \neq 0$, then for each j the matrix $B_j = \begin{pmatrix} 2\tau \mathbf{x}_j^\top \\ A \end{pmatrix}$ in the Gauss–Newton iteration in* (3.2) *is of full rank with a well-defined pseudoinverse. Moreover, the sequences $\{\varsigma_j\}$ and $\{\frac{\mathbf{x}_j}{\| \mathbf{x}_j \|_2}\}$ converge to σ_n and $\mathbf{v}_n$, respectively, with*

$$\left\| \frac{\mathbf{x}_j}{\| \mathbf{x}_j \|_2} - \mathbf{v}_n \right\|_2 \le \left(\frac{\sigma_n}{\sigma_{n-1}} \right)^{2j} \left[1 + \left(\frac{\sigma_n}{\sigma_{n-1}} \right)^{2j} \right] \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \breve{\mathbf{x}}_0 \right\|_2},$$
$$\left| \varsigma_j - \sigma_n \right| \le \left(\frac{\sigma_n}{\sigma_{n-1}} \right)^{2j} \sigma_1 \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \breve{\mathbf{x}}_0 \right\|_2}, \quad j = 1, 2, \ldots.$$

Proof. Since $\sigma_{n-1} > \sigma_n \ge 0$, $A\mathbf{v}_j \neq 0$ for $j = 1, \ldots, n-1$. So, B_0 is of full rank because of the assumption $\mathbf{x}_0^\top \mathbf{v}_n \neq 0$. Similarly B_j is of full rank for all $j > 0$ since $\mathbf{x}_j^\top \mathbf{v}_n \neq 0$ from (4.3). The proof of the remaining assertions is a straightforward verification. $\square$

5. Computing the approxi-null space. The iteration (3.2) produces a vector $\mathbf{w}_1$ in the approxi-null space $\mathbf{W}$ of A. When the approxi-nullity of A is bigger than one, we may stack a scalar multiple of $\mathbf{w}_1^\top$ on top of A to form a new matrix B. We will show in this section that when iteration (3.2) is applied to B it may produce another approxi-null vector $\mathbf{w}_2$ of A that is orthogonal to $\mathbf{w}_1$. This deflation-iteration process can be continued recursively to produce an orthonormal basis for the approxi-null space $\mathbf{W}$.

PROPOSITION 5.1. *Under the same assumptions of Proposition* 4.2*, for any unit vector* $\mathbf{w} \in \mathbf{W}$*, the matrix*

$$B = \begin{pmatrix} \varrho \mathbf{w}^\top \\ A \end{pmatrix} \quad \text{with } \varrho \geq \widehat{\sigma} \tag{5.1}$$

has singular values $\{\sigma'_j\}_{j=1}^n$ *satisfying*

$$\sigma'_1 \geq \cdots \geq \sigma'_{k+1} \geq \widehat{\sigma} > \breve{\sigma} \geq \sigma'_{k+2} \geq \cdots \geq \sigma'_n, \tag{5.2}$$

and its approxi-null space $\mathbf{W}'$ *spanned by the singular vectors of* B *associated with* $\sigma'_{k+2}, \ldots, \sigma'_n$ *is a subspace of* $\mathbf{W}$*.*

Proof. Since $\mathbf{w} \in \mathbf{W}$, we can write $\mathbf{w} = \rho_{k+1}\mathbf{v}_{k+1} + \cdots + \rho_n \mathbf{v}_n$ with $\rho_{k+1}^2 + \cdots + \rho_n^2 = 1$. The SVD $A = U\Sigma V^\top$ yields

$$\begin{aligned}
&\begin{pmatrix} 1 & \\ & U^\top \end{pmatrix} BV \\
&\quad = \begin{pmatrix} 0 & \cdots & 0 & \varrho\rho_{k+1} & \cdots & \varrho\rho_n \\ \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_n \\ & & & & & \end{pmatrix} = P \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ & & & \varrho\rho_{k+1} & \cdots & \varrho\rho_n \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_n \end{pmatrix} \\
&\quad = P \begin{pmatrix} I_{k\times k} & \\ & \hat{U} \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ & & & \hat{\sigma}_{k+1} & & \\ & & & & \ddots & \\ & & & & & \hat{\sigma}_n \end{pmatrix} \begin{pmatrix} I_{k\times k} & \\ & \hat{V}^\top \end{pmatrix},
\end{aligned}$$

where P is a permutation matrix with $\hat{U}$ and $\hat{V}$ being orthogonal matrices in the SVD of

$$D = \begin{pmatrix} \varrho\rho_{k+1} & \cdots & \varrho\rho_n \\ \sigma_{k+1} & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix} = \hat{U} \begin{pmatrix} \hat{\sigma}_{k+1} & & \\ & \ddots & \\ & & \hat{\sigma}_n \end{pmatrix} \hat{V}^\top.$$

We claim that

$$\hat{\sigma}_{k+1} \geq \varrho \quad \text{and} \quad \hat{\sigma}_j \leq \sigma_{j-1}, \quad j = k+2, \ldots, n. \tag{5.3}$$

In fact, $\hat{\sigma}_{k+1}$ is the largest singular value of D which is larger than or equal to ϱ since

$$\hat{\sigma}_{k+1} = \max_{\mathbf{x}\in\mathbb{R}^{n-k},\, \|\mathbf{x}\|_2=1} \| D\mathbf{x} \|_2 \geq \left\| D \begin{pmatrix} \rho_{k+1} \\ \vdots \\ \rho_n \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \varrho \\ \rho_{k+1}\sigma_{k+1} \\ \vdots \\ \rho_n\sigma_n \end{pmatrix} \right\|_2 \geq \varrho.$$

On the other hand, let $\mathbf{y} = (0, \ldots, 0, y_{n-1}, y_n)^\top \in \mathbb{R}^{n-k}$ such that $\| \mathbf{y} \|_2 = 1$ and $y_{n-1}\rho_{n-1} + y_n\rho_n = 0$. Then

$$\hat{\sigma}_n = \min_{\mathbf{x}\in\mathbb{R}^{n-k},\, \|\mathbf{x}\|_2=1} \| D\mathbf{x} \|_2 \leq \| D\mathbf{y} \|_2 = \sqrt{(\sigma_{n-1}y_{n-1})^2 + (\sigma_n y_n)^2} \leq \sigma_{n-1}.$$

Denote the columns of $\hat{V}$ by $\hat{\mathbf{v}}_{k+1}, \ldots, \hat{\mathbf{v}}_n$. For any fixed $j \in \{k+1, \ldots, n-2\}$, let $\mathbf{z} = (0, \ldots, 0, z_j, \ldots, z_n)^\top$ with $\| \mathbf{z} \|_2 = 1$, where $\hat{\mathbf{v}}_l^\top \mathbf{z} = 0$ for $l = j+2, \ldots, n$, and $\Sigma_{i=j}^n \rho_i z_i = 0$. Then

$$\begin{aligned} \hat{\sigma}_{j+1} &= \min \left\{ \| D\mathbf{x} \|_2 \mid \|\mathbf{x}\|_2 = 1, \mathbf{x}^\top \hat{\mathbf{v}}_l = 0, l = j+2, \ldots, n \right\} \\ &\leq \| D\mathbf{z} \|_2 = \sqrt{(z_j\sigma_j)^2 + \cdots + (z_n\sigma_n)^2} \leq \sigma_j \end{aligned}$$

and inequalities (5.3) hold. Consequently, they lead to the validity of the inequalities in (5.2) with

$$\left\{\sigma'_1, \ldots, \sigma'_{k+1}\right\} = \left\{\sigma_1, \ldots, \sigma_k, \hat{\sigma}_{k+1}\right\}, \quad \sigma'_l = \hat{\sigma}_l, \quad l = k+2, \ldots, n. \qquad \square$$

In practice, we may choose $\varrho = \|A\|_\infty$. In applying iteration (3.2) on B, as the least squares solution of

$$\begin{pmatrix} \tau\mathbf{w}_2^\top \\ \varrho\mathbf{w}_1^\top \\ A \end{pmatrix} \mathbf{w}_2 = \begin{pmatrix} \tau \\ 0 \\ 0 \end{pmatrix},$$

$\mathbf{w}_2 \in \mathbf{W}$, is approximately orthogonal to $\mathbf{w}_1$. Continuing this process recursively, an orthonormal basis for $\mathbf{W}$ can be constructed.

As an important special case, if $\sigma_{n-1} \gg \sigma_n$, iteration (3.2) converges to $\mathbf{w} = \mathbf{v}_n$ and $\varsigma = \sigma_n$. In this case, stacking $\varrho\mathbf{v}_n^\top$ on top of A makes σ_{n-1} the smallest singular value of the resulting matrix.

COROLLARY 5.2. *Let σ be a singular value of A with associated singular vector $\mathbf{v}$. The matrix*

$$A_\rho = \begin{pmatrix} \rho\mathbf{v}^\top \\ A \end{pmatrix} \tag{5.4}$$

has the same singular values and corresponding singular vectors as those of A, except the singular value σ of A is replaced by the singular value $\sqrt{\rho^2 + \sigma^2}$ of A_ρ associated with the same singular vector $\mathbf{v}$.

Proof. For simplicity, let $\sigma = \sigma_n$ and $A = U\Sigma V^\top$ be the SVD of A. We have

$$\begin{pmatrix} 1 & \\ & U^\top \end{pmatrix} \begin{pmatrix} \rho \mathbf{v}^\top \\ A \end{pmatrix} V = \begin{pmatrix} \rho \mathbf{v}^\top V \\ U^\top A V \end{pmatrix} = \begin{pmatrix} 0 & \cdots & 0 & \rho \\ \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_{n-1} & \\ & & & \sigma \\ & & & \end{pmatrix}.$$

By applying a Givens transformation from the left on ρ and σ, it is clear that the singular value σ of A is replaced by the singular value $\sqrt{\rho^2 + \sigma^2}$ of A_ρ, while the associated singular vector remains the same. ∎

6. The overall algorithm. As mentioned before, the approxi-rank k of matrix A depends critically on the chosen threshold $\theta > 0$ for which singular values of A satisfy

$$\sigma_1 \geq \cdots \geq \sigma_k > \theta > \sigma_{k+1} \geq \cdots \geq \sigma_n. \tag{6.1}$$

There is no uniform threshold for all applications. The user must make a decision on the threshold $\theta > 0$ based on the nature of the application.

The approxi-rank gap $\gamma = \frac{\sigma_k}{\sigma_{k+1}}$ may be considered a condition number for this rank-revealing problem. If γ is large, say, 10^3, then every iterative step in (3.2) will improve the convergence by six digits because in Proposition 4.2 the sequences $\{\mathbf{x}_j\}$ and $\{\varsigma_j\}$ satisfy

$$\frac{\left\| \widehat{\mathbf{x}}_j \right\|_2}{\left\| \widecheck{\mathbf{x}}_j \right\|_2} \leq \left(10^{-3}\right)^{2j} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \widecheck{\mathbf{x}}_0 \right\|_2} \quad \text{and} \quad \sigma_n \leq \varsigma_j \leq \sigma_{k+1} + \left(10^{-3}\right)^{2j} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \widecheck{\mathbf{x}}_0 \right\|_2} \sigma_1.$$

After three iteration steps they become

$$\frac{\left\| \widehat{\mathbf{x}}_3 \right\|_2}{\left\| \widecheck{\mathbf{x}}_3 \right\|_2} \leq 10^{-18} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \widecheck{\mathbf{x}}_0 \right\|_2} \quad \text{and} \quad \sigma_n \leq \varsigma_3 \leq \sigma_{k+1} + 10^{-18} \frac{\left\| \widehat{\mathbf{x}}_0 \right\|_2}{\left\| \widecheck{\mathbf{x}}_0 \right\|_2} \sigma_1.$$

Since the machine epsilon of IEEE standard double precision is about 2.2×10^{-16}, in this case $\mathbf{x}_3$ is sufficiently accurate to be an approxi-null vector unless the randomly chosen initial vector $\mathbf{x}_0$ is almost orthogonal to the approxi-null space.

Let $(\sigma_1, \mathbf{v}_1), \ldots, (\sigma_n, \mathbf{v}_n)$ be the singular pairs of A with σ_j's satisfying (6.1). For an input threshold $\theta > 0$, our algorithm begins with calculating the smallest singular pair $(\hat{\sigma}_n, \hat{\mathbf{v}}_n)$. If $\hat{\sigma}_n > \theta$, A will be classified as being of full approxi-rank, and the process stops. Otherwise, the algorithm continues by calculating singular pairs $(\hat{\sigma}_{n-1}, \hat{\mathbf{v}}_{n-1})$, $(\hat{\sigma}_{n-2}, \hat{\mathbf{v}}_{n-2})$, ... Once we reach $\hat{\sigma}_k > \theta$, the process will be terminated with k being the approxi-rank of A and $\text{span}\{\hat{\mathbf{v}}_{k+1}, \ldots, \hat{\mathbf{v}}_n\}$ the approxi-null space. If the approxi-rank gap $\gamma = \frac{\hat{\sigma}_k}{\hat{\sigma}_{k+1}}$ is not as large, it may need more than three iteration steps in (3.2) for each singular value. The users can set the number of iteration steps based on the nature of the application. The overall algorithm RankRev is shown in a pseudocode in Figure 6.1.

Pseudocode RankRev:
input: Matrix $A \in \mathbb{R}^{m\times n}$, threshold $\theta > 0$
output: approxi-rank k, orthonormal basis $\{\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n\}$
for the approxi-null space.

```
Compute the QR decomposition A = QR
Initialize B = R,   τ = ||A||_∞
For k = n, n−1, ..., 1 do
    generate a random unit vector x_0
    for  j = 0, 1, 2  do
        D = [ 2τ x_j^T ; B ],  b = [ τ x_j^T x_j − τ ; B x_j ]
        Hessenberg QR decomposition D = QR
        backward substitution to solve Rz = Q^T b for z
        x_{j+1} = x_j − z,   ς = ||Rx_{j+1}||_2 ||x_{j+1}||_2^{-1}
        if ς < θ then
            w_k = x_{j+1} / ||x_{j+1}||_2
            break j-loop
        end if
    end do
    if ς > θ then
        break k-loop
    else
        Hessenberg QR decomposition [τ w_k^T ;  B] = QR
        update B = R
    end if
end do
```

FIG. 6.1. *Pseudocode of RankRev.*

Practically, the iteration (3.2) is carried out by finding a least squares solution $\Delta\mathbf{x}$ $(= \mathbf{x}_{j+1} - \mathbf{x}_j)$ to the linear system

$$\begin{pmatrix} 2\tau\mathbf{x}_j^\top \\ A \end{pmatrix} \Delta\mathbf{x} = -\begin{pmatrix} \tau\left(\mathbf{x}_j^\top\mathbf{x}_j - 1\right) \\ A\mathbf{x}_j \end{pmatrix} \tag{6.2}$$

at the jth stage. To avoid unnecessary QR decomposition of the full matrix at each step, we may calculate the QR decomposition of A *before* the iteration and update the QR decomposition at each step.

With QR factorization $A = Q\binom{R}{0}$, finding the least squares solution to (6.2) is equivalent to solving the least squares problem of

$$\begin{pmatrix} 2\tau\mathbf{x}_j^\top \\ R \end{pmatrix} \Delta\mathbf{x} = -\begin{pmatrix} \tau\mathbf{x}_j^\top\mathbf{x}_j - \tau \\ R\mathbf{x}_j \end{pmatrix} \tag{6.3}$$

for $\Delta\mathbf{x} = \mathbf{x}_{j+1} - \mathbf{x}_j$, in which one uses the QR decomposition of the upper Hessenberg matrix $\binom{2\tau\mathbf{x}_j^\top}{R}$. Updating the QR factorization of an n-column upper Hessenberg matrix requires n Givens transformations which cost $O(n^2)$ flops in total. After QR

updating, solving (6.3) for its least squares solution requires a total of $O(n^2)$ flops in backward substitutions.

The final QR factorization of $\binom{2\tau\mathbf{x}_j^\top}{R}$ can be used as the QR decomposition of the matrix B in (5.1) with $\rho = 2\tau$ and $\mathbf{w} = \mathbf{x}_j$. The computations are all on the order of $O(n^2)$ except the first QR factorization of A which costs $O(mn^2)$. Actually, on many occasions the QR decomposition of A had already been calculated for other purposes.

7. Numerical experiments and comparisons. Our rank-revealing algorithm is implemented as a MATLAB module RankRev that is electronically available from the authors upon request. Here we compare its efficiency, robustness, and accuracy with the MATLAB built-in SVD function as well as hurv in UTV Tools implemented by Fierro, Hansen, and Hansen [6]. The package UTV Tools is perhaps the only published comprehensive rank-revealing package with updating/downdating capabilities. All tests are carried out in MATLAB 6.1 on a Dell personal computer with a Pentium 4 CPU of 1.8 Mhz, 768 Mb of memory, and machine precision $\varepsilon \approx 10^{-16}$.

The main objective of our code RankRev is to calculate the approxi-rank and the approxi-null space of a matrix A that has a low approxi-nullity (equivalently, A is close to being of full approxi-rank) within a user-specified threshold. If the given matrix A is of approxi-rank about $n/2$, the full SVD can be more efficient. For low approxi-rank (i.e., high approxi-nullity) cases, UTV Tools function lurv and SVDPACK based on Lanczos method [1] are efficient options.

When $A \in \mathbb{R}^{m\times n}$ has an approxi-rank k within threshold $\theta > 0$, then A is often considered to be under a perturbation of a "noise" matrix E with $\|E\|_2 \le \theta$ such that $A - E$ has exact rank k. The 2-norm of E is often referred to as noise level. Usually, we consider a perturbation magnitude near machine precision, say, 1.0e-12, a low noise level, perturbation near 1, say, 1.0e-3, a high noise level, and the median noise level is around 1.0e-8.

7.1. Type 1: Low approxi-nullity, median noise level, small gap. Matrices for this test are of size $2n\times n$ with approxi-nullity fixed at 10 within threshold 10^{-8}. The singular values range from ε to $\|A\|_2 = 20$ with approxi-rank gap $\frac{\sigma_{n-10}}{\sigma_{n-9}} = 10^3$. Each matrix A is constructed using those specified singular values to form a diagonal matrix Σ and by setting $A = U\Sigma V^\top$ with randomly generated orthogonal matrices U and V with proper sizes. We use this type of matrix to test the efficiency and accuracy of RankRev compared with SVD and hurv for increasing n.

All three algorithms output accurate approxi-ranks. Table 7.1 lists the times and errors in executing SVD, hurv, and RankRev. The time measures are in seconds, and the error measures the distances of the computed approxi-null spaces to the spaces spanned by the right singular vectors associated with the ten smallest singular values. The results show that our RankRev is at least as efficient as hurv with significantly higher accuracy. When matrix sizes are in the thousands, both hurv

Table 7.1
Results for Type 1 *matrices.*

	Matrix sizes							
	400×200		800×400		1600×800		3200×1600	
	time	error	time	error	time	error	time	error
SVD	0.67	1e-15	5.6	2e-15	43.6	1e-15	1166.9	2e-15
hurv	1.41	1e-06	3.4	2e-06	11.6	1e-05	79.2	7e-06
RankRev	1.23	2e-09	3.3	2e-09	11.3	2e-09	48.8	2e-09

TABLE 7.2

Results for Type 2 *matrices. The computed approxi-ranks in parentheses are inaccurate results from* HURV.

	Matrix column size n	100	200	300	400	500
	Approxi-rank	50	100	150	200	250
HURV	Computed approxi-rank	50	100	150	(234)	(294)
	Approxi-null space error	1e-10	1e-05	3e-05	—	—
RANKREV	Computed approxi-rank	50	100	150	200	250
	Approxi-null space error	3e-10	6e-10	3e-08	5e-08	6e-08

TABLE 7.3

The accuracy measures on Type 3 *matrices without refinement. Due to the fixed size of the test matrices, the execution time is close to a constant for each code. We therefore list only the average time in the parentheses next to the code name.*

	Approxi-rank gaps γ					
Code (time)	10^6	10^5	10^4	10^3	10^2	10^1
HURV (4.86)	7.4e-11	2.7e-09	3.4e-08	1.6e-06	1.1e-04	1.8e-02
RANKREV (4.58)	7.4e-11	2.2e-10	6.3e-10	2.0e-09	6.9e-08	2.6e-04

and RANKREV are more than ten times faster than standard SVD even with the interpretation overhead in MATLAB codes.

7.2. Type 2: Median approxi-nullity, median noise level, small gap. Matrices used for this test are of $2n \times n$ with approxi-rank $\frac{n}{2}$ within threshold 10^{-8}. They are constructed in the same way as Type 1 above except for different singular values. The singular values range from ε to $\|A\|_2 = 20$ with approxi-rank gap $\gamma = 10^3$. While computing approxi-ranks of matrices of this sort is not the main goal of either RANKREV or HURV; we simply use them to test the robustness of both codes since both algorithms must recursively deflate $\frac{n}{2}$ times here. As shown in Table 7.2, the approxi-null space accuracy for HURV seems to deteriorate when n increases and it fails to provide accurate approxi-ranks for $n = 400$ and $n = 500$ even when its refinement option is activated.[1] In contrast, our code RANKREV always outputs accurate approxi-ranks and tiny errors in computed approxi-null spaces.

7.3. Type 3. Decreasing gaps, fixed size, low approxi-nullity, median noise level. Matrices A_j, $j = 6, 5, \ldots, 2, 1$, used in this test are of size 1000×500 with an approxi-nullity fixed at 10 within the same threshold 10^{-8}. The singular values range from ε to $\|A_j\|_2 = 20$. However, the approxi-rank gaps are set at 10^j for $j = 6, 5, \ldots, 2, 1$, respectively.

Table 7.3 lists the accuracy measures on computed approxi-null spaces with decreasing approxi-rank gaps. While the accuracy in computing the approxi-null spaces of both RANKREV and HURV deteriorate when the approxi-rank gap diminishes, our code RANKREV achieves a higher accuracy level with slightly faster speed. When tighter accuracy on the approxi-null space is required in application, while UTV Tools has its own refinement strategy [6, 11], we may simply set tighter criteria for stopping the Gauss–Newton iteration. Table 7.4 shows both codes are about equally accurate with their refinements.

[1] In a recent correspondence, the authors of UTV Tools indicated that the source of the problem leading to those failures has been identified and will be dealt with in future releases.

TABLE 7.4
The accuracy measures on Type 3 matrices with refinement.

Code (time)	Approxi-rank gaps					
	10^6	10^5	10^4	10^3	10^2	10^1
HURV (9.74)	7.4e-11	2.2e-10	6.3e-10	2.0e-09	6.9e-09	1.6e-08
RANKREV (7.82)	7.4e-11	2.2e-10	6.3e-10	2.0e-09	6.9e-09	1.8e-08

TABLE 7.5
Results for Type 4 matrices.

	Matrix sizes							
	400×200		800×400		1600×800		3200×1600	
	time	error	time	error	time	error	time	error
HURV	1.55	8.3e-05	3.42	3.2e-03	15.8	1.4e-04	71.9	2.6e-03
RANKREV	1.27	8.5e-11	3.05	6.8e-11	12.9	4.0e-10	48.5	1.6e-10

TABLE 7.6
Results for Type 5 matrices.

	Matrix sizes							
	400×200		800×400		1600×800		3200×1600	
	time	error	time	error	time	error	time	error
HURV	1.40	1.2e-15	3.47	9.0e-16	23.7	1.1e-15	—	failed
RANKREV	1.16	2.0e-14	3.11	4.1e-14	18.9	9.5e-14	53.7	6.5e-13

7.4. Type 4. High noise level, low approxi-nullity, small gap. The series of matrices used in this test are of $2n \times n$ with singular values in the interval $[1, 2]$ except ten small singular values in the interval $[0, 10^{-2}]$. Those matrices are used to test the accuracy and robustness of the rank-revealing computation in the presence of high noise level.

The results exhibited in Table 7.5 show the significant advantage of RANKREV over HURV in accuracy without refinement. If both codes activate the refinement option, however, HURV achieves slightly higher accuracy (2.9e-15) over RANKREV (4.3e-14), while RANKREV is slightly faster in speed by about 15%.

7.5. Type 5. Near-zero noise level, low approxi-nullity, large gap. This series of test matrices has singular values in the interval $[1, 2]$, except for the smallest ten, which are in the magnitude of machine precision. Table 7.6 shows that HURV consistently achieves slightly higher accuracy when the approxi-ranks are correctly determined, while our code maintains the advantage in efficiency. Nonetheless, HURV did not report an accurate approxi-rank for the 3200×1600 matrix. (The error appears to be machine-dependent. The authors of HURV are currently investigating it.)

8. Updating and downdating. For $A \in \mathbb{R}^{m \times n}$, the algorithm RANKREV in Figure 6.1 produces an approxi-rank k, a matrix $W = [\mathbf{w}_{k+1}, \mathbf{w}_{k+2}, \ldots, \mathbf{w}_n]$ whose columns form an orthonormal basis of the approxi-null space $\mathbf{W}$ of A, and a QR decomposition

$$\begin{pmatrix} \tau W^\top \\ A \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{8.1}$$

When a row/column is inserted in A, the determination of a new set of k, W, Q, and R of the new matrix using the information already available is called *updating*. It is called *downdating* if a row/column is deleted from A instead.

One of the main motivations in seeking alternatives to SVD in determining approxi-ranks is its difficulties in updating and downdating. The UTV decomposition possesses good updating capabilities, but its downdating seems somewhat complicated. In contrast, both updating and downdating in our method are straightforward and also quite stable and efficient.

In elaborating our procedure for updating and downdating, we shall repeatedly use the following QR downdating strategy [8, section 12.5.3].

We wish to compute the QR decomposition of the submatrix $\hat{B}$ in

$$B = \begin{bmatrix} \mathbf{b}^\top \\ \hat{B} \end{bmatrix}_{\begin{smallmatrix}1\\ m-1\end{smallmatrix}} = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times n},$$

where the QR decomposition of B is available as given above. Let $\mathbf{q}^\top$ be the first row of Q and $G_1, \ldots, G_{m-1}$ be Givens rotations such that

$$G_1 \cdots G_{m-1}\mathbf{q} = \mathbf{e}_1.$$

Notice that

$$H = G_1 \cdots G_{m-1} \begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{bmatrix} \mathbf{v}^\top \\ \hat{R} \\ 0 \end{bmatrix}_{\begin{smallmatrix}1\\ n\\ m-n-1\end{smallmatrix}}$$

is upper Hessenberg and

$$QG_{m-1}^\top \cdots G_1^\top = \begin{bmatrix} 1 & \\ & \hat{Q} \end{bmatrix}_{\begin{smallmatrix}1\\ m-1\end{smallmatrix}},$$

where $\hat{Q}$ is orthogonal. Thus, for $G = G_1 \cdots G_{m-1}$,

$$\begin{pmatrix} \mathbf{b}^\top \\ \hat{B} \end{pmatrix} = \left[QG^\top\right] \left[G \begin{pmatrix} R \\ 0 \end{pmatrix} \right] = \begin{bmatrix} 1 & \\ & \hat{Q} \end{bmatrix} \left[\begin{pmatrix} \mathbf{b}^\top \\ \hat{R} \\ 0 \end{pmatrix} \right], \tag{8.2}$$

and therefore

$$\hat{B} = \hat{Q} \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}.$$

This QR downdating process requires $O(n^2)$ flops.

8.1. Column updating. For $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{R}^{m\times n}$ and $\mathbf{a}_{n+1} \in \mathbb{R}^m$, let $\hat{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{a}_{n+1})$. Clearly the approxi-null space $\hat{\mathbf{W}}$ of $\hat{A}$ contains $\{\hat{\mathbf{w}}_{k+1} \ldots, \hat{\mathbf{w}}_n\}$, where

$$\hat{\mathbf{w}}_j = \begin{pmatrix} \mathbf{w}_j \\ 0 \end{pmatrix}, \quad j = k+1, \ldots, n.$$

Those $\hat{\mathbf{w}}_j$'s remain orthonormal. The approxi-rank of $\hat{A}$ is either k or $k+1$. Only when it stays at k, the orthonormal basis of $\hat{\mathbf{W}}$ contains an additional vector which is the only approxi-null vector of the matrix

$$\breve{A} = \begin{pmatrix} \tau\hat{W}^\top \\ \hat{A} \end{pmatrix} = \begin{pmatrix} \tau W^\top & \mathbf{0} \\ A & \mathbf{a}_{n+1} \end{pmatrix}, \tag{8.3}$$

where $\hat{W} = [\hat{\mathbf{w}}_{k+1}, \ldots, \hat{\mathbf{w}}_n]$. For the QR decomposition

$$\begin{pmatrix} \tau W^\top \\ A \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

in (8.1), let

$$Q^\top \breve{A} = Q^\top \begin{pmatrix} \tau W^\top & \mathbf{0} \\ A & \mathbf{a}_{n+1} \end{pmatrix} = \begin{pmatrix} R & \mathbf{d}_1 \\ 0 & \mathbf{d}_2 \end{pmatrix}$$

and H be the Householder transformation satisfying

$$H\mathbf{d}_2 = (\zeta, 0, \ldots, 0)^\top.$$

Then

$$\breve{A} = \begin{pmatrix} \tau W^\top & \mathbf{0} \\ A & \mathbf{a}_{n+1} \end{pmatrix} = \left[Q \begin{pmatrix} I_{n\times n} & 0 \\ 0 & H^\top \end{pmatrix} \right] \begin{bmatrix} \begin{pmatrix} R & \mathbf{d}_1 \\ 0 & \zeta \end{pmatrix} \\ \begin{matrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{matrix} \end{bmatrix} = \tilde{Q} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}, \tag{8.4}$$

and we may obtain the possible additional approxi-null vector by

$$\begin{aligned} &\text{solving } R\mathbf{x} = -\mathbf{d}_1 \quad \text{for } \mathbf{x} \in \mathbb{R}^n \\ &\text{and setting } \mathbf{y} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}, \qquad \hat{\mathbf{w}}_{n+1} = \frac{1}{\|\mathbf{y}\|_2}\,\mathbf{y}. \end{aligned} \tag{8.5}$$

Clearly,

$$\hat{W}^\top \hat{\mathbf{w}}_{n+1} = 0 \quad \text{and} \quad \left\| \hat{A}\hat{\mathbf{w}}_{n+1} \right\|_2 = \frac{|\zeta|}{\|\mathbf{y}\|_2}.$$

When $\frac{|\zeta|}{\|\mathbf{y}\|_2}$ is below the threshold θ, $\hat{\mathbf{w}}_{n+1}$ becomes the additional approxi-null vector and $\{\hat{\mathbf{w}}_{k+1}, \ldots, \hat{\mathbf{w}}_{n+1}\}$ constitutes an orthonormal basis for the approxi-null space $\hat{\mathbf{W}}$ of $\hat{A}$.

For further updating or downdating, if needed, we also update the QR decomposition in (8.1):

$$\begin{pmatrix} \tau\hat{\mathbf{w}}_{n+1}^\top \\ \tau\hat{W}^\top \\ \hat{A} \end{pmatrix} = \begin{pmatrix} 1 & \\ & \tilde{Q} \end{pmatrix} \begin{bmatrix} \tau\hat{\mathbf{w}}_{n+1}^\top \\ \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix} \end{bmatrix} = \hat{Q} \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}. \tag{8.6}$$

Computing $\hat{Q}$ and $\hat{R}$ requires $O(n^2)$ additional flops since $\tilde{R}$ is already upper-triangular.

If the new column is inserted between the $(l-1)$th and the lth column of A where $l < n$, we may first append the new column as the last (i.e., the $(n+1)$th) column and complete the computation described above. Then by switching its lth and $(n+1)$th components for each approxi-null vector $\hat{\mathbf{w}}_j$, $j = k+1, \ldots, n+1$, we obtain an orthonormal basis for the approxi-null space of $\hat{A}$, the new matrix with a new lth column inserted.

For further updating and/or downdating, the QR decomposition in (8.6) needs to be revised. We illustrate the process for $n = 4$ and $l = 2$ as

$$\begin{pmatrix} \tau\hat{W}^\top \\ \hat{A} \end{pmatrix} = \hat{Q} \begin{pmatrix} + & \times & + & + & + \\ & \times & + & + & + \\ & \times & & + & + \\ & \times & & & + \\ & \times & & & \end{pmatrix} = \hat{Q}G_1^\top \begin{pmatrix} + & \times & + & + & + \\ & \times & + & + & + \\ & \times & & + & + \\ & * & & & * \\ & 0 & & & * \end{pmatrix}$$

$$= \hat{Q}G_1^\top G_2^\top \begin{pmatrix} + & \times & + & + & + \\ & \times & + & + & + \\ & * & & * & * \\ & 0 & & * & * \\ & 0 & & & * \end{pmatrix} = \hat{Q}G_1^\top G_2^\top G_3^\top \begin{pmatrix} + & \times & + & + & + \\ & * & * & * & * \\ & 0 & * & * & * \\ & 0 & & * & * \\ & 0 & & & * \end{pmatrix}$$

$$= \check{Q} \begin{pmatrix} \check{R} \\ 0 \end{pmatrix}, \tag{8.7}$$

where the G_j's are the Givens rotations. The new $\check{Q}$ and $\check{R}$ are then available for further use.

We summarize the column updating process as follows.

- Input: matrix A, approxi-rank k, scaling factor τ, rank threshold θ, orthonormal basis for the approxi-null space $\mathbf{W}$, the QR decomposition Q and R as in (8.1), a new column $\mathbf{a}_{n+1}$, and its location l to be inserted.
- Append $\hat{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{a}_{n+1})$, form $\check{A}$ as in (8.3).
- Update the QR decomposition $\tilde{Q}$ and $\tilde{R}$ of $\check{A}$ as in (8.4).
- Calculate $\hat{\mathbf{w}}_{n+1}$ as in (8.5), and obtain the residual $\frac{|\zeta|}{\|\mathbf{y}\|_2}$.
- If the residual $\frac{|\zeta|}{\|\mathbf{y}\|_2} > \theta$, then
  - Set $k = k+1$, $\hat{W} = \left[\hat{\mathbf{w}}_{k+1}, \ldots, \hat{\mathbf{w}}_n\right]$, $\hat{Q} = \tilde{Q}$, $\hat{R} = \tilde{R}$

  else
  - Calculate $\hat{Q}$ and $\hat{R}$ as in (8.6), set $\hat{W} = \left[\hat{\mathbf{w}}_{k+1}, \ldots, \hat{\mathbf{w}}_n, \hat{\mathbf{w}}_{n+1}\right]$

  end if
- If $l \neq n+1$, then
  - Swap the lth and the $(n+1)$th components of every approxi-null vector as columns of $\hat{W}$
  - Calculate $\check{Q}$ and $\check{R}$ as in (8.7), set as $\hat{Q}$ and $\hat{R}$, respectively.

end if

- Output updated k, $\hat{W}$, $\hat{Q}$, $\hat{R}$.

While the only significant cost of updating is solving an upper-triangular system $R\mathbf{x} = -\mathbf{d}_1$ in (8.5) with $n^2 + O(n)$ flops when $\hat{Q}$ and $\hat{R}$ are not needed, the cost stays at $O(mn + n^2)$ with output $\hat{Q}$ and $\hat{R}$.

8.2. Column downdating. Let $\tilde{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_{l-1}, \mathbf{a}_{l+1}, \ldots, \mathbf{a}_n)$ where the lth colum $\mathbf{a}_l$ of A is deleted and $\tilde{\mathbf{W}}$ be its approxi-null space. If the approxi-nullity of A, or the dimension $n-k$ of its approxi-null space $\mathbf{W}$, is zero, then the approxi-nullity of $\tilde{A}$ remains zero, requiring no further computations. We therefore assume $n-k \geq 1$ and write

$$W = [\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n] = \begin{pmatrix} w_{1,k+1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,k+1} & \cdots & w_{n,n} \end{pmatrix} \in \mathbb{R}^{n \times (n-k)}.$$

Let $H \in \mathbb{R}^{(n-k)\times(n-k)}$ be the Householder transformation satisfying

$$H \begin{pmatrix} w_{l,k+1} \\ w_{l,k+2} \\ \vdots \\ w_{l,n} \end{pmatrix} = \begin{pmatrix} \eta \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{8.8}$$

This yields

$$WH^\top = [\hat{\mathbf{w}}_{k+1}, \ldots, \hat{\mathbf{w}}_n] = \begin{pmatrix} * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \\ \eta & 0 & \cdots & 0 \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & \cdots & * & * \end{pmatrix} \longleftarrow l\text{th row.}$$

Because

$$\left(WH^\top\right)^\top \left(WH^\top\right) = H\left(W^\top W\right)H^\top = H^\top I_{n-k} H = I_{n-k},$$

the columns of $WH^\top$ also form an orthonormal basis for $\mathbf{W}$. By removing the lth component of $\hat{\mathbf{w}}_j$ for each $j = k+1, \ldots, n$, we obtain a set of vectors $\tilde{\mathbf{w}}_{k+1}, \ldots, \tilde{\mathbf{w}}_n$ satisfying

$$\tilde{A}\tilde{\mathbf{w}}_{k+1} = A\hat{\mathbf{w}}_{k+1} - \eta\,\mathbf{a}_l, \qquad \tilde{A}\tilde{\mathbf{w}}_j = A\hat{\mathbf{w}}_j, \quad j = k+2, \ldots, n.$$

Apparently, $\{\tilde{\mathbf{w}}_{k+2}, \ldots, \tilde{\mathbf{w}}_n\}$ is a subset of an orthonormal basis for $\tilde{\mathbf{W}}$, and the magnitude of $\| \tilde{A}\tilde{\mathbf{w}}_{k+1} \|_2$ determines the possible existence of an additional approxi-null vector: when it is small enough, the normalization of $\tilde{\mathbf{w}}_{k+1}$ completes $\{\tilde{\mathbf{w}}_{k+1}, \ldots, \tilde{\mathbf{w}}_n\}$ as an orthonormal basis for $\tilde{W}$.

To downdate the QR decomposition in (8.1) for further updating/downdating, since

$$\begin{pmatrix} \tau H\, W^\top \\ A \end{pmatrix} = \left[\begin{pmatrix} H & \\ & I \end{pmatrix} Q \right] \begin{pmatrix} R \\ 0 \end{pmatrix},$$

we have

$$\begin{pmatrix} \tau \tilde{W}^\top \\ \tilde{A} \end{pmatrix} = \left[\begin{pmatrix} H & \\ & I \end{pmatrix} Q \right] \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} = \left[\begin{pmatrix} H & \\ & I \end{pmatrix} Q G_l^\top \cdots G_{n-1}^\top \right] \begin{pmatrix} \check{R} \\ 0 \end{pmatrix}, \tag{8.9}$$

where $\hat{R}$ is obtained from R by deleting its lth column and $G_l, \cdots G_{n-1}$ are the Givens rotations that transform $\hat{R}$ into upper-triangular $\check{R}$. Applying the QR downdating technique in (8.2) yields

$$\begin{pmatrix} \tau \tilde{\mathbf{w}}_{k+2}^\top \\ \vdots \\ \tau \tilde{\mathbf{w}}_n^\top \\ \tilde{A} \end{pmatrix} = \tilde{Q} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}. \tag{8.10}$$

The column downdating process stops here if $\tilde{\mathbf{w}}_{k+1}$ is not an approxi-null vector. Otherwise, we will stack $\tau \tilde{\mathbf{w}}_{k+1}^\top$ as the top row and update the QR decomposition in (8.10):

$$\begin{aligned} \begin{pmatrix} \tau \tilde{\mathbf{w}}_{k+1}^\top \\ \tau \tilde{\mathbf{w}}_{k+2}^\top \\ \vdots \\ \tau \tilde{\mathbf{w}}_n^\top \\ \tilde{A} \end{pmatrix} &= \begin{pmatrix} 1 & \\ & \tilde{Q} \end{pmatrix} \begin{pmatrix} \tau \tilde{\mathbf{w}}_n \\ \tilde{R} \\ 0 \end{pmatrix} \\ &= \left[\begin{pmatrix} 1 & \\ & \tilde{Q} \end{pmatrix} \begin{pmatrix} G^\top & \\ & I \end{pmatrix} \right] \begin{pmatrix} \breve{R} \\ 0 \end{pmatrix} = \breve{Q} \begin{pmatrix} \breve{R} \\ 0 \end{pmatrix}, \end{aligned} \tag{8.11}$$

where G is a product of $n-1$ Givens rotations that transforms the upper Hessenberg matrix $\begin{pmatrix} \tau \tilde{\mathbf{w}}_{k+1} \\ \tilde{R} \end{pmatrix}$ into upper triangular form $\breve{R}$.

The column downdating algorithm can be summarized as follows:

- Input: matrix A, approxi-rank k, scaling factor τ, threshold θ, orthonormal basis $\{\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n\}$ for the approxi-null space $\mathbf{W}$, the QR decomposition Q and R as in (8.1), a column index l indicating the lth column of A is to be deleted.
- Form $W = [\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n]$ and the Householder transformation H in (8.8).
- Set $\hat{W} = W\, H^\top$ and η as in (8.8).
- Get $\tilde{W} = [\tilde{\mathbf{w}}_{k+1}, \ldots, \tilde{\mathbf{w}}_n]$ by deleting the lth row of $\hat{W}$ and normalizing the first column afterward.
- Form the QR decomposition (8.9).
- Apply the QR downdating process (8.2) on (8.9) to obtain $\tilde{Q}$ and $\tilde{R}$ in (8.10).
- If $\| \tilde{A} \tilde{\mathbf{w}}_{k+1} \|_2 > \theta$, then

– Output k, $\tilde{W} = [\tilde{\mathbf{w}}_{k+2}, \ldots, \tilde{\mathbf{w}}_n]$, $\tilde{Q}$, $\tilde{R}$, the approxi-rank stays at k.

else

– Update the QR decomposition as in (8.11).
– Output $k = k-1$, $\tilde{W} = [\tilde{\mathbf{w}}_{k+1}, \ldots, \tilde{\mathbf{w}}_n]$, $\breve{Q}$, $\breve{R}$, the approxi-rank reduces by one.

end if

It requires $O(n^2)$ flops to carry out the column downdating process.

8.3. Row updating. Inserting a row $\mathbf{b}^\top$ into A for a new matrix $\hat{A}$, the approxi-rank of $\hat{A}$ will remain the same unless the approxi-rank k of A is less than n. In such cases, it is clear that the approxi-null space $\hat{\mathbf{W}}$ of $\hat{A}$ is a subset of the approxi-null space $\mathbf{W}$ of A, and they are equal if $\mathbf{b}$ is approximately orthogonal to $\mathbf{W}$. Namely, $\mathbf{W} = \hat{\mathbf{W}}$ if $\| W^\top \mathbf{b} \|_2 \le \theta$, where $W = [\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n] \in \mathbb{R}^{n\times(n-k)}$, whose columns form an orthogonal basis of $\mathbf{W}$. When $\| W^\top \mathbf{b} \|_2 > \theta$, the approxi-rank of the new matrix $\hat{A}$ becomes $k+1$. To find an orthonormal basis of $\hat{\mathbf{W}}$ in this case, we first let $\mathbf{y} = W^\top \mathbf{b} \in \mathbb{R}^{n-k}$ and $H \in \mathbb{R}^{(n-k)\times(n-k)}$ be the Householder transformation such that $H\mathbf{y} = (\|\mathbf{y}\|_2, 0, \ldots, 0)^\top$. Denoting $H = [\mathbf{y}_{k+1}, \ldots, \mathbf{y}_n]$, we have $\{\mathbf{y}\}^\perp = \text{span}\{\mathbf{y}_{k+2}, \ldots, \mathbf{y}_n\}$. For $E = [\mathbf{y}_{k+2}, \ldots, \mathbf{y}_n] \in \mathbb{R}^{(n-k)\times(n-k-1)}$, let $WE = [\hat{\mathbf{w}}_{k+2}, \ldots, \hat{\mathbf{w}}_n] \in \mathbb{R}^{n\times(n-k-1)}$. The columns $\{\hat{\mathbf{w}}_{k+2}, \ldots, \hat{\mathbf{w}}_n\}$ form an orthonormal basis for $\hat{\mathbf{W}}$ because

$$(WE)^\top (WE) = E^\top \left(W^\top W\right) E = I_{(n-k-1)\times(n-k-1)},$$

and for $j = k+2, \ldots, n$, $\| \hat{A}\hat{\mathbf{w}}_j \|_2 = \| A\hat{\mathbf{w}}_j \|_2$ since $\mathbf{b}^\top W E = \mathbf{y}^\top E = 0$.

To update the QR decomposition in (8.1), we apply the QR downdating strategy in (8.2) on

$$\begin{pmatrix} \tau H W^\top \\ A \end{pmatrix} = \left[\begin{pmatrix} H & \\ & I \end{pmatrix} Q \right] \begin{pmatrix} R \\ 0 \end{pmatrix} = \breve{Q} \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{8.12}$$

to delete its first row, yielding

$$\begin{pmatrix} \tau E^\top W^\top \\ A \end{pmatrix} = \hat{Q} \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}. \tag{8.13}$$

When inserting a new row $\mathbf{b}^\top$ into A, let P be the permutation matrix that swaps the new row to the top. It follows that

$$\begin{pmatrix} \tau E^\top W^\top \\ \hat{A} \end{pmatrix} = P \begin{pmatrix} \mathbf{b}^\top \\ \tau E^\top W^\top \\ A \end{pmatrix} = P \begin{pmatrix} 1 & \\ & \breve{Q} \end{pmatrix} \begin{pmatrix} \mathbf{b}^\top \\ \hat{R} \\ 0 \end{pmatrix} = \tilde{Q} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}, \tag{8.14}$$

where

$$\tilde{Q} = P \begin{pmatrix} 1 & \\ & \breve{Q} \end{pmatrix} G^\top, \qquad G \begin{pmatrix} \mathbf{b}^\top \\ \hat{R} \\ 0 \end{pmatrix} = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix},$$

and G is the product of n Givens rotations.

Our row-updating algorithm can be summarized as follows:

- Input: matrix A, approxi-rank k, scaling factor τ, threshold θ, orthonormal basis $\{\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n\}$ for the approxi-null space $\mathbf{W}$, the QR decomposition Q and R as in (8.1), a new row $\mathbf{b}^\top$, and the row index l indicating $\mathbf{b}^\top$ will be inserted above the lth row of A.
- Form $W = [\mathbf{w}_{k+1}, \ldots, \mathbf{w}_n]$.
- If $\| W^\top \mathbf{b} \|_2 < \theta$, then
  - Update the QR decomposition (8.1) for inserting $\mathbf{b}^\top$
  - Output k, W and the updated Q, R.

  else
  - Construct the Householder transformation H such that $H(W^\top \mathbf{b}) = (*, 0, \ldots, 0)^\top$.
  - Use H to get $\breve{Q}$ as in (8.12).
  - Downdate the QR decomposition (8.12) to obtain $\hat{Q}$ and $\hat{R}$ in (8.13)
  - Insert $\mathbf{b}^\top$ into A and update the QR decomposition (8.13) to obtain $\tilde{Q}$ and $\tilde{R}$ in (8.14)
  - Output $k = k + 1$, $\tilde{W} = WE$, $\tilde{Q}$, $\tilde{R}$, the approxi-rank increases by one.

  end if

8.4. Row downdating. Let $\breve{A}$ be the matrix obtained from A by deleting its lth row $\mathbf{r}^\top$. For a proper permutation matrix P, we have, from (8.1),

$$\begin{pmatrix} \mathbf{r}^\top \\ \tau W^\top \\ \breve{A} \end{pmatrix} = P \begin{pmatrix} \tau W^\top \\ A \end{pmatrix} = [PQ] \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{8.15}$$

Applying the QR downdating algorithm (8.2) on this QR decomposition yields

$$\tilde{A} = \begin{pmatrix} \tau W^\top \\ \breve{A} \end{pmatrix} = \breve{Q} \begin{pmatrix} \breve{R} \\ 0 \end{pmatrix}. \tag{8.16}$$

Obviously, the approxi-null space $\hat{\mathbf{W}}$ of $\hat{A}$ contains the approxi-null space $\mathbf{W}$ of A. For the possible emergence of an extra approxi-null vector of $\hat{A}$, we may apply the Gauss–Newton iteration (3.2) on the matrix $\breve{R}$ to calculate the singular vector. As explained earlier, if this singular vector is indeed an extra approxi-null vector, it is orthogonal to columns of W and forms an orthonormal basis for $\hat{\mathbf{W}}$ along with columns of W. We omit the pseudocode since the process is a straightforward application of QR downdating algorithm.

Remark. As mentioned in [6], row downdating may be difficult and complex for UTV decomposition: "... [W]e want to emphasize that numerically stable UTV downdating algorithms have become very complex, and the computational overhead can become quite large, especially when the exact rank decreases. It may be worth to consider whether recomputation of the ULV decomposition ... is to be preferred." In comparison, row downdating in our algorithm seems quite straightforward.

8.5. Numerical results on updating and downdating. Our updating and downdating algorithms have been thoroughly tested for all circumstances of inserting/deleting rows or columns. Since UTV Tools [6] contains only row updating and row downdating modules, we shall restrict our comparisons with UTV Tools to those situations only. The results of our method on column updating and downdating are quite similar.

TABLE 8.1
Comparisons on random row updating with changing approxi-ranks.

		Number of random rows inserted						
		1	2	3	···	8	9	10
Time (seconds)	URV_UP	0.66	0.64	0.63	···	0.67	0.67	0.49
	ROWUP	1.00	0.98	0.95	···	0.98	0.98	0.98
Approxi-null space accuracy	URV_UP	1e-8	1e-8	1e-8	···	3e-8	1e-8	0.0
	ROWUP	3e-9	2e-8	2e-8	···	4e-8	2e-9	0.0

The two modules in UTV Tools for row updating and row downdating are URV_UP and URV_DW, respectively. The updating module URV_UP works on inserting a row at the bottom, and the downdating module URV_DW applies to deleting the top row. Row inserting/deleting may or may not change the approxi-rank. Our tests show that there seems to be a significant difference in performance for both modules of UTV Tools in rank invariant and rank altering cases.

All tests in this section are conducted on the same computer listed in section 7. Both URV_UP and URV_DW are set to use their default control parameters, while our codes ROWUP and ROWDOWN are set to optimize the speed.

8.5.1. Row updating with changing approxi-ranks. The test matrix is initially a 1000×500 matrix having an approxi-nullity 10 with threshold 10^{-8}. The approxi-rank gap is $\gamma = 10^4$. After executing our RANKREV and HURV on this initial matrix separately, a random vector is inserted at the bottom in each updating step. Therefore, every update results in an increase in the approxi-rank by one. Both URV_UP in UTV Tools and our ROWUP have no difficulty identifying the increasing approxi-ranks with nearly identical accuracy in the updated approxi-null space. As shown in Table 8.1, URV_UP is considerably faster than our ROWUP in this case.

8.5.2. Row updating without changing approxi-ranks. When no changes in the approxi-rank occur for row updating, the code URV_UP in UTV Tools seems to have difficulties in identifying the approxi-ranks during the recursive updating, especially when the approxi-rank gap is not large enough. Even when the gap is large, URV_UP is still prone to miscalculating the approxi-rank at certain points. In contrast, our code ROWUP always outputs accurate approxi-ranks in all occasions and runs more than twice as fast.

Table 8.2 shows this event in a typical example. The initial matrix has the same features as the one in section 8.5.1 except the approxi-rank gap γ is increased to 10^6 since URV_UP fails too soon for the gap 10^4. A sequence of rows consisting of linear combinations of the existing rows are inserted at the bottom one at a time. The approxi-rank should stay at 490. However, after certain steps in the recursive updating, URV_UP outputs inaccurate approxi-ranks.

8.5.3. Row downdating without changing approxi-ranks. When deleting a row does not change the approxi-rank, our code ROWDOWN and its counterpart URV_DW in UTV Tools show similar performance in both efficiency and accuracy. The test starts by constructing an initial matrix $A \in \mathbb{R}^{1000\times 500}$ with the same features as in the initial matrix in section 8.5.1. Then 20 rows that are linear combinations of the existing rows of A are generated and stacked on top of A. Deleting those rows one by one does not alter the approxi-rank. Table 8.3 shows the results.

TABLE 8.2
Comparisons on random row updating without changing approxi-ranks. Data in parentheses indicate inaccurate computation.

		Number of linearly dependent rows inserted							
		1	2	···	5	6	7	···	10
Time (seconds)	URV_UP	1.09	1.14	···	1.11	0.69	0.69	···	1.11
	ROWUP	0.39	0.50	···	0.39	0.48	0.59	···	0.42
Approxi-null space error	URV_UP	2e-6	4e-6	···	3e-6	(0.15)	(0.06)	···	(0.14)
	ROWUP	1e-9	1e-9	···	2e-9	2e-9	3e-9	···	3e-9
Approxi-rank output	URV_UP	490	490	···	490	(491)	(492)	···	(492)
	ROWUP	490	490	···	490	490	490	···	490

TABLE 8.3
Comparisons on random row downdating without changing approxi-ranks.

		Number of linear dependent rows deleted						
		1	2	3	···	8	9	10
Time (seconds)	URV_DW	0.75	0.73	0.78	···	0.75	0.72	0.72
	ROWDOWN	0.78	0.78	0.89	···	0.76	0.70	0.70
Approxi-null space error	URV_DW	6e-8	1e-7	1e-7	···	4e-7	4e-7	4e-7
	ROWDOWN	6e-8	1e-7	1e-7	···	4e-7	4e-7	4e-7

8.5.4. Row downdating with decreasing approxi-ranks. As mentioned in [6], UTV decomposition may have difficulties in downdating especially when it reduces the approxi-ranks. This phenomenon does occur in the experiment we conducted below. We downdate a matrix of 1030×500 obtained by stacking 30 random rows on top of a matrix A of size 1000×500 with an approxi-nullity 30 within a threshold of 10^{-8}. The approxi-rank gap is set at a relatively large threshold 10^6. During the test, the 30 random rows are deleted one-by-one and both URV_DW and ROWDOWN are used to downdate the approxi-rank and the approxi-null space. The approxi-rank should decrease by one at every downdating step.

Table 8.4 shows that when downdating the approxi-rank accurately as in steps 1 to 15, both URV_DW and ROWDOWN exhibit similar efficiency and accuracy. At step 16, URV_DW miscalculates the approxi-rank by one and this error is carried on in remaining downdating steps, whereas our code ROWDOWN always produces the correct approxi-rank.

It is not clear whether the inaccurate outputs of UTV Tools in those difficult tests in both sections 7 and 8.5 are inherent in the UTV decomposition or the results of coding errors. They are under investigation by the authors of UTV Tools.

9. Applications.

9.1. Computing polynomial GCD. A new method for computing the GCD of univariate polynomials plays a key role in establishing a novel algorithm that accurately calculates polynomial roots and their multiplicities without using multiprecision arithmetic even if the polynomial is inexactly given [19]. This root-finding method is implemented in the MATLAB package MULTROOT [20]. Our rank-revealing method and recursive column updating constitute indispensable components in the new GCD finder and the root finder.

For any polynomial $h(x) = h_0x^k + h_1x^{k-1} + \cdots + h_k$, its coefficient vector is

TABLE 8.4
Comparisons on random row downdating with changing approxi-ranks. Data in parentheses indicate inaccurate computation.

		Number of linearly independent rows deleted						
		1	···	15	16	17	···	30
Approxi-null space error	URV_DW	9e-10	···	6e-10	(0.89)	(0.89)	···	(0.87)
	ROWDOWN	9e-10	···	2e-9	3e-9	3e-9	···	6e-8
Approxi-rank output	URV_DW	499	···	485	(485)	(484)	···	(471)
	ROWDOWN	499	···	485	484	483	···	470
Time (seconds)	URV_DW	1.06	···	1.17	—	—	···	—
	ROWDOWN	0.95	···	1.03	1.03	1.05	···	1.03

denoted by $\mathbf{h} = (h_0, h_1, \ldots, h_k)^\top$. Let $p(x)$ and $q(x)$ be polynomials of degrees n and m, respectively. Write

$$p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_n,$$
$$q(x) = q_0 x^m + q_1 x^{m-1} + \cdots + q_m,$$

with $n \geq m$. A polynomial $u(x)$ is a GCD of $p(x)$ and $q(x)$, denoted by $\mathrm{GCD}\,(p, q)$, if there are polynomials $v(x)$ and $w(x)$ such that

$$u(x)v(x) = p(x), \qquad u(x)w(x) = q(x), \tag{9.1}$$

where $v(x)$ and $w(x)$ share no common roots (or, equivalently, no common factors). The $(n + r) \times (n - m + 2r)$ matrix

$$S_r(p,q) = \begin{pmatrix} \overbrace{\begin{matrix} p_0 & & \\ p_1 & \ddots & \\ \vdots & \ddots & p_0 \\ p_n & & p_1 \\ & \ddots & \vdots \\ & & p_n \end{matrix}}^{r} & \overbrace{\begin{matrix} q_0 & & \\ q_1 & \ddots & \\ \vdots & \ddots & q_0 \\ q_m & & q_1 \\ & \ddots & \vdots \\ & & q_m \end{matrix}}^{n-m+r} \end{pmatrix}, \quad n \geq m,\ r = 1, 2, \ldots, m,$$

is called the rth *Sylvester subresultant matrix* of p and q. The following results are well known [14, 19]:

- $S_m(p, q)$ is of rank $n + m - 2l + 1$ if and only if $\deg(\mathrm{GCD}\,(p, q)) = l \geq 1$.
- $S_r(p, q)$ has full rank if and only if $\deg(\mathrm{GCD}\,(p, q)) \leq m - r$.
- If $S_{r-1}(p, q)$ is of full rank and $S_r(p, q)$ is rank deficient, then the null space of $S_r(p, q)$ is spanned by the vector $\begin{pmatrix} \mathbf{w} \\ -\mathbf{v} \end{pmatrix}$ whose components $\mathbf{v} \in \mathbb{R}^{n-m+r}$ and $\mathbf{w} \in \mathbb{R}^r$ are coefficient vectors of $v(x)$ and $w(x)$ in (9.1), respectively. Also, $\deg(\mathrm{GCD}\,(p, q)) = m - r + 1$.

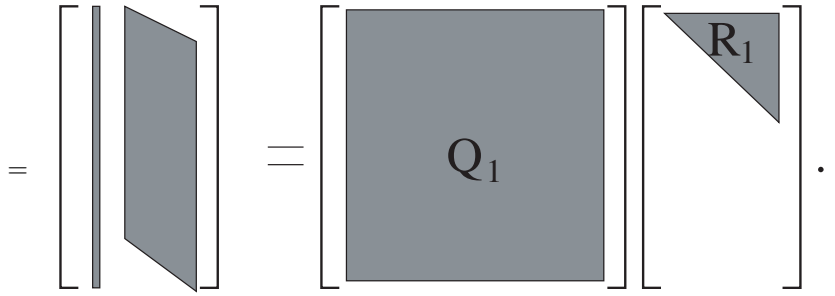
From those results, one may calculate $\mathrm{GCD}\,(p, q)$ by finding the first approxi-rank deficient Sylvester matrix $S_r(p, q)$ in the sequence

$$S_1(p,q), S_2(p,q), \ldots, S_m(p,q)$$

and a (single vector) basis of the approxi-null space. A GCD finder constructed in this way can be illustrated in the following process.

First, we form $S_1(p,q)$, set the first permutation $P_1 = I_{(n-m+2)\times(n-m+2)}$, and calculate its QR decomposition

$$T_1 = S_1(p,q)P_1$$

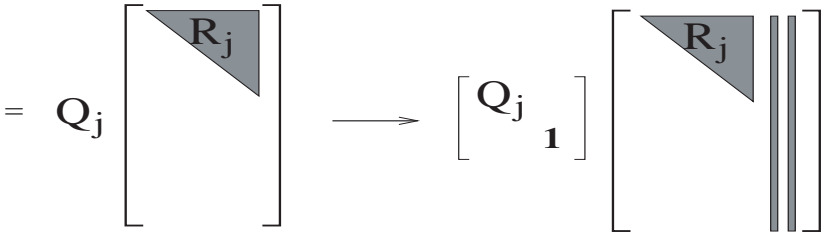


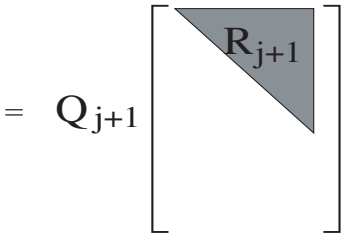
If $S_1(p,q)$ is approxi-rank deficient, then GCD $(p,q) = q$. The process needs to continue only if $S_1(p,q)$ is of full approxi-rank. In general, if $S_j(p,q)$ is of full approxi-rank with its pivoted QR decomposition $T_j = S_j(p,q)P_j = Q_jR_j$ being available, we attach one zero row to the bottom of T_j and add two columns

$$\begin{bmatrix} Q_j^\top & \\ & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ q_0 \\ \vdots \\ q_m \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} Q_j^\top & \\ & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ p_0 \\ \vdots \\ p_n \end{bmatrix}$$

to the right of the resulting matrix to form T_{j+1}. With a proper permutation matrix P_{j+1}, we have $T_{j+1}P_{j+1}^\top = S_{j+1}(p,q)$. Therefore,

$$T_j = S_j(p,q)P_j$$





$$= S_{j+1}(p,q)P_{j+1} = T_{j+1}.$$

Updating the QR decomposition of $T_{j+1} = S_{j+1}(p,q)P_{j+1}$ requires only $O(n+m)$ additional flops. We apply the iteration (3.2) on R_{j+1} for an approxi-null vector. If R_{j+1} (or, equivalently, $S_{j+1}(p,q)$) remains in full approxi-rank, the process continues to $j+2$ in a similar way. It stops at the (column permuted) kth Sylvester resultant matrix $T_k = S_k(p,q)P_k$, the first to be approxi-rank deficient.

It can be shown that the null space of T_k is of dimension one with a single null vector $\mathbf{z} \in \mathbb{R}^{n-m+2k}$ in its basis. Let

$$\begin{pmatrix} \mathbf{w} \\ -\mathbf{v} \end{pmatrix} = P_k^\top \mathbf{z} \quad \text{with } \mathbf{w} \in \mathbb{R}^k \text{ and } \mathbf{v} \in \mathbb{R}^{n-m+k}.$$

Then $\mathbf{v}$ and $\mathbf{w}$ are coefficient vectors of $v(x)$ and $w(x)$ satisfying (9.1). Now $u(x) = \mathrm{GCD}\,(p,q)$ is the quotient of $p(x)$ and $v(x)$. However, it is numerically unstable to use polynomial synthetic division $p(x) \div v(x)$ for finding $u(x)$ [19]. Instead, we use the "least squares division" [19] which solves the coefficient vector $\mathbf{u}$ of $u(x)$ as a least squares solution to

$$\begin{pmatrix} v_0 & & \\ v_1 & \ddots & \\ \vdots & \ddots & v_0 \\ v_s & & v_1 \\ & \ddots & \vdots \\ & & v_s \end{pmatrix} \mathbf{u} = \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ \\ \vdots \\ \vdots \\ p_n \end{pmatrix}, \quad \mathbf{u} \in \mathbb{R}^{m-k+2},\ s = n-m+k-1. \tag{9.2}$$

The procedure listed in Figure 9.1 illustrates the calculation of $\deg(\mathrm{GCD}\,(p,q))$ and the coefficients of $u(x)$, $v(x)$, and $w(x)$ in (9.1). To achieve highest attainable accuracy, the Gauss–Newton iteration on a quadratic system based on (9.1) can be applied to refine the GCD [19].

9.2. Nonisolated solutions to a polynomial system. When a numerical solution $\mathbf{x}_0$ of a system of polynomial equations

$$P(\mathbf{x}) = \big(p_1(\mathbf{x}), \ldots, p_n(\mathbf{x})\big) = 0, \quad \text{where } \mathbf{x} = (x_1, \ldots, x_n)^\top \in \mathbb{C}^n,$$

is obtained, we wish to identify whether $\mathbf{x}_0$ is an isolated solution of $P(\mathbf{x}) = 0$. While in the previous sections we mainly focused our attention on the development of our method and algorithm in the real vector space $\mathbb{R}^n$, the entire content remains valid in $\mathbb{C}^n$ with proper adjustments.

If the Jacobian of $P(\mathbf{x})$, denoted by $P_x(\mathbf{x})$, at $\mathbf{x}_0$ allows no small (relative to $\|P_x(\mathbf{x}_0)\|_\infty$) singular values, $\mathbf{x}_0$ is of course an isolated solution. When our rank-revealing algorithm is applied to $P_x(\mathbf{x}_0)$ and the result shows it admits very small singular values, $\mathbf{x}_0$ may lie on a solution component of $P(\mathbf{x}) = 0$ with positive dimension or it may still be an isolated zero with multiplicity ≥ 2. Our strategy to distinguish those cases is given below.

If $P_x(\mathbf{x}_0)$ permits only one singular value that appears tiny and if $\mathbf{x}_0$ is not an isolated solution, then $\mathbf{x}_0$ must lie on a one- (complex) dimensional solution component M of $P(\mathbf{x}) = 0$. We will begin to identify this path to a substantial length by a path following scheme developed in [9]. If this attempt fails, no such solution component M may exist and $\mathbf{x}_0$ will be classified as an isolated solution of $P(\mathbf{x}) = 0$.

```
Pseudocode GCD:
 input:  coefficient vectors for p(x), q(x)
 output:  d = deg(GCD (p, q)),
             coefficients of v(x) and w(x) in (9.1)
    QR decomposition QR = S_1(p, q)
    For j = 1, 2, ..., m do
        Gauss--Newton iteration (3.2) on R, get ϱ and x
        if ϱ is small enough, then
            extract coefficients of v(x) and w(x) from x
            solve (9.2) for the coefficients of u(x)
            exit
        else
            if j ≤ m then
                update Q_{j+1}R_{j+1} = S_{j+1}(p, q)P_{j+1}
            else
                deg(GCD (p, q)) = 0, v(x) = p(x), w(x) = q(x)
            end if
        end if
    end do
```

FIG. 9.1. *Pseudocode of GCD.*

When $P_x(\mathbf{x}_0)$ has $k > 1$ very small singular values as a result of our rank-revealing algorithm, we augment $P(\mathbf{x}) = 0$ with $k-1$ generic hyperplanes

$$\mathbf{a}_j^H(\mathbf{x} - \mathbf{x}_0) = 0, \quad j = 1, \ldots, k-1,$$

at $\mathbf{x}_0$. The enlarged system

$$\widehat{P}(\mathbf{x}) = \begin{cases} P(\mathbf{x}) = 0, \\ \mathbf{a}_1^H(\mathbf{x} - \mathbf{x}_0) = 0 \\ \vdots \\ \mathbf{a}_{k-1}^H(\mathbf{x} - \mathbf{x}_0) = 0 \end{cases} \tag{9.3}$$

will produce a one-dimensional component $\widehat{M}$ of $\widehat{P}(\mathbf{x}) = 0$ if the solution component M of $P(\mathbf{x}) = 0$ to which $\mathbf{x}_0$ belongs is of dimension k. Thus, the assertion that $\dim(M) = k$ is valid only if we can identify $\widehat{M}$ by following this path to a satisfactory length. If the path following cannot be carried out successfully, such a component $\widehat{M}$ may not exist. We will then remove hyperplane $\mathbf{a}_{k-1}^H(\mathbf{x} - \mathbf{x}_0) = 0$ in (9.3) and restart our effort to identify the one-dimensional component $\widehat{\widehat{M}}$ produced by

$$\widehat{\widehat{P}}(\mathbf{x}) = \begin{cases} P(\mathbf{x}) = 0, \\ \mathbf{a}_1^H(\mathbf{x} - \mathbf{x}_0) = 0 \\ \vdots \\ \mathbf{a}_{k-2}^H(\mathbf{x} - \mathbf{x}_0) = 0. \end{cases} \tag{9.4}$$

The existence of such a component $\widehat{\widehat{M}}$ of $\widehat{\widehat{P}} = 0$ implies the solution component M of $P(\mathbf{x}) = 0$ containing $\mathbf{x}_0$ is of dimension $k-1$. If it fails, the process may be continued in the same manner and the dimension of M will ultimately (very soon in practice) be determined. Of course, when $\dim(M) = 0$, $\mathbf{x}_0$ is an isolated zero even though $P_x(\mathbf{x}_0)$ may have very small singular values from our rank-revealing algorithm.

Example (see [15]). Consider the polynomial system $P(\mathbf{x}) = (p_1(\mathbf{x}), p_2(\mathbf{x}), p_3(\mathbf{x}))$, $\mathbf{x} = (u, v, w) \in \mathbb{C}^3$, where

$$\begin{aligned} p_1(\mathbf{x}) &= (v - u^2) \quad \cdot \quad (u^2 + v^2 + w^2 - 1)(u - 0.5), \\ p_2(\mathbf{x}) &= (w - u^3)(u^2 + v^2 + w^2 - 1)(v - 0.5), \\ p_3(\mathbf{x}) &= (v - u^2)(w - u^3)(u^2 + v^2 + w^2 - 1)(w - 0.5). \end{aligned}$$

Obviously, the solution set of $P(\mathbf{x}) = 0$ consists of

1. a two-dimensional component $u^2 + v^2 + w^2 = 1$;
2. four one-dimensional components
   (a) line $u = 0.5$, $v = (0.5)^3$;
   (b) line $u = \sqrt{0.5}$, $v = 0.5$;
   (c) line $u = -\sqrt{0.5}$, $v = 0.5$;
   (d) twisted cubic $v = u^2$, $w = u^3$;
3. one isolated solution $(u, v, w) = (0.5, 0.5, 0.5)$.

When the polyhedral homotopy continuation method [10] was used to solve $P(\mathbf{x}) = 0$, 129 numerical solutions were obtained. We applied our method to all those solutions, and the result shows

- 112 of them lie on the two-dimensional component,
- 16 of them lie on one-dimensional components (four on line 2a, four on line 2b, four on line 2c, four on line 2d),
- one isolated solution.

When we classified a solution $\mathbf{x}_0$ that is lying on a two-dimensional component of $P(\mathbf{x}) = 0$, for instance, we substituted $\mathbf{x}_0$ into $u^2 + v^2 + w^2 = 1$ to verify the accuracy of our identification, and the results were all accurate.

Acknowledgments. The authors wish to thank R. D. Fierro, P. C. Hansen, and P. S. K. Hansen for making UTV Tools freely available. In particular, the second author is grateful to P. C. Hansen for his helpful e-correspondence.

REFERENCES

[1] M. W. Berry, *Large scale sparse singular value computation*, Internat. J. Supercomput. Appl., 6 (1992), pp. 13–49.
[2] C. H. Bischof and G. Quintana-Orti, *Algorithm* 782: *Codes for rank-revealing QR factorizations of dense matrices*, ACM Trans. Math. Software, 24 (1998), pp. 254–257.
[3] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
[4] T. R. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.
[5] F. Deprettere, *SVD and Signal Processing, Algorithms, Applications, and Architectures*, North–Holland, Amsterdam, 1988.
[6] R. D. Fierro, P. C. Hansen, and P. S. K. Hansen, *UTV tools: MATLAB templates for rank-revealing UTV decompositions*, Numer. Algorithms, 20 (1999), pp. 165–194.
[7] G. H. Golub, V. Klema, and G. W. Stewart, *Rank Degeneracy and Least Squares Problems*, Tech. rep. TR 456, University of Maryland, Baltimore, MD, 1976.
[8] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[9] Y. C. Kuo and T. Y. Li, *Determining Whether a Zero of a Polynomial System is Isolated*, preprint, 2003.

[10] T. Y. Li, *Numerical solution of multivariate polynomial systems by homotopy continuation methods*, in Acta Numerica, Acta Numer. 6, Cambridge University Press, Cambridge, UK, 199, pp. 399–436.
[11] R. Mathias and G. W. Stewart, *A block QR algorithm and the singular value decomposition*, Linear Algebra Appl., 182 (1993), pp. 91–100.
[12] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math. Oxford Ser. (2), 11 (1960), pp. 50–59.
[13] M. Moonen and B. De Moor, *SVD and Signal Processing,* III*, Algorithms, Applications, and Architectures*, Elsevier, Amsterdam, 1995.
[14] D. Rupprecht, *An algorithm for computing certified approximate GCD of n univariate polynomials*, J. Pure Appl. Algebra, 139 (1999), pp. 255–284.
[15] A. J. Sommese, J. Verschelde, and C. W. Wampler, *Numerical decomposition of the solution sets of polynomial systems into irreducible components*, SIAM J. Numer. Anal., 38 (2001), pp. 2022–2046.
[16] G. W. Stewart, *UTV decompositions*, in Numerical Analysis 1993, D. F. Griffith and G. A. Watson, eds., Pitman Res. Notes Math. Ser. 303, Longman, Harlow, UK 1994, pp. 225–236.
[17] G. W. Stewart, *Matrix Algorithms: Basic Decompositions*, SIAM, Philadelphia, 1998.
[18] R. Vaccaro, *SVD and Signal Processing,* II*, Algorithms, Applications, and Architectures*, Elsevier, Amsterdam, 1991.
[19] Z. Zeng, *Computing multiple roots of inexact polynomials*, Math. Comp., 74 (2005), pp. 869–903.
[20] Z. Zeng, *Algorithm* 835: *MultRoot—a MATLAB package for computing polynomial roots and multiplicities*, ACM Trans. Math. Software, 30 (2004), pp. 218–236.