Numer Algor
DOI 10.1007/s11075-017-0328-7

ORIGINAL PAPER

RankRev: a Matlab package for computing the numerical rank and updating/downdating

Tsung-Lin Lee[1] · **Tien-Yien Li**[2] · **Zhonggang Zeng**[3]

Received: 19 May 2016 / Accepted: 17 April 2017

Abstract The numerical rank determination frequently occurs in matrix computation when the conventional exact rank of a hidden matrix is desired to be recovered. This paper presents a Matlab package RANKREV that implements two efficient algorithms for computing the numerical rank and numerical subspaces of a matrix along with updating/downdating capabilities for making adjustment to the results when a row or column is inserted/deleted. The package and the underlying algorithms are accurate, reliable, and much more efficient than the singular value decomposition when the matrix is of low rank or low nullity.

Keywords Numerical rank · Updating · Downdating · Test suite · Matlab toolbox

1 Introduction

The rank-revealing problem arises widely in scientific computing and engineering applications, such as signal processing [16, 18], information retrieval [4], numerical polynomial algebra [19], etc. Some of those applications give rise to a large

✉ Tsung-Lin Lee
leetsung@math.nsysu.edu.tw

Tien-Yien Li
li@math.msu.edu

Zhonggang Zeng
zzeng@neiu.edu

1 Department of Applied Mathematics, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

2 Department of Mathematics, Michigan State University, East Lansing, MI 48824, USA

3 Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA

matrix whose rank or nullity is known to be small a priori. Although the Golub-Reinsch algorithm for the singular value decomposition (SVD) can be applied to calculate *all* singular values and hence determine the numerical rank, it becomes unnecessarily expensive in these situations. Furthermore, it is relatively difficult to update the SVD when the matrix is altered by inserting or deleting a few rows or columns. Several alternative methods have been proposed instead in the literature. Generally, they compute a gap-revealing factorization for estimating the singular values, such as rank-revealing QR decomposition (RRQR) [1–3], rank-revealing two-sided orthogonal decompositions (UTV or URV/ULV) [5–7], and rank-revealing LU decomposition (RRLU) [10].

Adaptive cross approximation (ACA) is capable of computing a low-rank factorization of a large matrix without computing a gap revealing factorization [11, 15]. This method only computes some entries, giving ACA a superior complexity. However, it is limited with respect to up- and downdating. Two novel rank-revealing strategies in [12] and [13] deal with large matrices having high rank (low nullity) and low rank, respectively. Rather than computing a gap-revealing factorization, the methods compute an orthonormal basis for the numerical subspace of the matrix directly. The resulting algorithms are quite efficient, and they only output those numerical subspaces that are usually in demand in applied problems.

After the numerical rank of a matrix has been calculated along with the orthonormal bases of some subspaces, a row or a column is often needed to be inserted into the matrix. It is desirable to make a minor adjustment to the available computing results accordingly, avoiding an unnecessary full rank-revealing from scratch. This process is called updating. It is called downdating if a row/column is deleted. For many applications, efficient rank updating and downdating algorithms play a very important role. An algorithm for updating UTV decomposition is proposed by G. W. Stewart in 1992, but for such a decomposition, the downdating problem seems somewhat difficult to carry out [7, 17]. The new high rank and low rank revealing algorithms generate the *kernel stacked QR factorization* and the *USV-plus decomposition*, respectively. The updating/downdating algorithms from these factorizations are quite straightforward. Furthermore, these algorithms are reliable and efficient as evidenced by the extensive numerical experiments [12, 13]. These rank-revealing algorithms and their accompanied updating/downdating algorithms are implemented in package RankRev.

The algorithms are outlined in Section 2. Section 3 shows the usage of three Matlab functions in RankRev. The numerical experiments will be in Section 4. In Section 5, we give a description of the test suite appended in this package.

2 Algorithms

The RankRev package includes the implementations of two rank-revealing algorithms as well as their updating and downdating procedures. In this section, we briefly describe those algorithms. We shall denote matrices by upper case letters such as A, B, R, and column vectors by lower case boldface letters like $\mathbf{u}, \mathbf{v}$, and $\mathbf{y}$. Notation

$(\cdot)^*$ stands for the Hermitian of a matrix or a vector, while notation $(\cdot)^\top$ denotes the transpose of a matrix or a vector.

We assume the target matrix A is of size $m \times n$ with the convention $m \geq n$. The *numerical rank* of matrix A with respect to threshold $\theta > 0$ is defined as the smallest rank of all matrices within a 2-norm distance θ of A [8, 9, 14]. Namely,

$$\operatorname{rank}_\theta(A) = \min_{\|B-A\|_2 \leq \theta} \left\{\operatorname{rank}(B)\right\}.$$

In terms of singular value decomposition (SVD), numerical rank is the number of singular values greater than the given threshold. However, there is no uniform threshold for all applications. The magnitude of threshold is usually assigned to reflect the noise level the matrices may have encountered in the applications.

2.1 Rank revealing

The *numerical range* $\mathcal{R}_\theta(A)$ of A is the subspace spanned by the left singular vectors associated with singular values greater than the threshold θ, and the *numerical row space* $\mathcal{R}_\theta(A^*)$ of A is the numerical range of A^*. When the numerical rank of a matrix is known to be small, the rank-revealing method constructs an orthonormal basis for its numerical range. The algorithm starts with the power iteration on AA^* to find a unit vector $\mathbf{u}_1$ in the numerical range. As shown in [12], by subtracting matrix $\mathbf{u}_1\mathbf{u}_1^*A$ from A, the new matrix $\tilde{A} = A - \mathbf{u}_1\mathbf{u}_1^*A$ has numerical rank one less than A's, and the numerical range of $\tilde{A}$ is contained in $\mathcal{R}_\theta(A)$ but orthogonal to $\mathbf{u}_1$. Therefore, we apply the power iteration on $\tilde{A}\tilde{A}^*$ to find a unit vector $\mathbf{u}_2 \in \mathcal{R}_\theta(A)$ with $\mathbf{u}_2 \perp \mathbf{u}_1$. The process can be continued recursively to calculate $\mathbf{u}_3, \mathbf{u}_4, \ldots$ until the orthonormal basis $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\}$ for $\mathcal{R}_\theta(A)$ is obtained along with the numerical rank r. The pseudo-code of the algorithm is given in Fig. 1.

The main component of the algorithm is constituted with the power iteration on $\tilde{A}\tilde{A}^*$. In fact, the matrix $\tilde{A}$ is neither computed nor stored since it is not needed in explicit form. More precisely, after finding $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$, let $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k]$. The power iteration will be on

$$(A - UU^*A)(A - UU^*A)^* = (I - UU^*)AA^*(I - UU^*).$$

The first iteration vector $\mathbf{y}_1 = (I - UU^*)AA^*(I - UU^*)\mathbf{y}_0$. Since the columns of U form an orthonormal set, the second iteration vector

$$\begin{aligned}
\mathbf{y}_2 &= (I - UU^*)AA^*(I - UU^*)\mathbf{y}_1 \\
&= (I - UU^*)AA^*(I - UU^*)(I - UU^*)AA^*(I - UU^*)\mathbf{y}_0 \\
&= (I - UU^*)\left[AA^*(I - UU^*)\right]\left[AA^*(I - UU^*)\right]\mathbf{y}_0.
\end{aligned}$$

Therefore, this power iteration is equivalent to the power iteration on $AA^*(I - UU^*)$ combined with projection $(I - UU^*)$ on the final iterate. Such implementation reduces the number of flops from $8mn + 16mk + 4m$ to $8mn + 12mk + 3m$ for computing the second iteration vector. Similarly, the flops are reduced by $8mk + 2m$ for computing the third iteration vector. The code for rank-revealing is designed based on

```
PSEUDO-CODE   low rank-revealing algorithm
Input:  Matrix $A \in \mathbb{C}^{m\times n}$, numerical rank threshold $\theta > 0$
    • Initialize $\epsilon_m = \|A\|_\infty \epsilon_{machine}$ along with empty matrix $U$
    • for $k = 0, 1, \ldots, n$ do
        – generate  a random unit vector $\mathbf{z}_0$
        – for $j = 1, 2, \ldots$ do
            * set $\mathbf{p} = A^*[\mathbf{z}_{j-1} - U(U^*\mathbf{z}_{j-1})]$,  $\zeta_j = \|\mathbf{p}\|_2$,
              $\mathbf{x} = \mathbf{p}/\|\mathbf{p}\|_2$
            * set $\mathbf{z}_j = A\mathbf{x}$
            * if $\left(\frac{\theta}{\zeta_j}\right)^{2j-1} < \epsilon_m$ or $\zeta_j + \frac{|\zeta_j-\zeta_{j-1}|^2}{|\zeta_{j-1}-\zeta_{j-2}|-|\zeta_j-\zeta_{j-1}|} < \epsilon_m$
              (when $j > 4$) then break the $j$-loop, end if
          end do
        – set $\mathbf{y} = \mathbf{z}_j - U(U^*\mathbf{z}_j)$, $\mathbf{u} = \mathbf{y}/\|\mathbf{y}\|_2$
        – if $\zeta_j \le \theta$ then break the $k$-loop, end if
        – update $U = [U, \mathbf{u}]$,
      end do
    • (optional) find the skinny QR-factorization $QR = A^*U$.
  Output:  the numerical rank $r = k$
      (optional: $U$, $S = R^*$, $V = Q$ in USV-plus decomposition)
```

Fig. 1 Pseudo-code of `low rank-revealing algorithm`

the above algorithm with this modification that improves the efficiency in computing time and storage requirement.

As remarked, the algorithm only needs a unit vector in the numerical range and thus requires smaller number of power iteration steps in comparison with existing methods that require to compute singular vectors. The stopping criteria derived in [12]

$$\left(\frac{\theta}{\zeta_j}\right)^{2j-1} < \epsilon_m \text{ and } \zeta_j + \frac{\left|\zeta_j - \zeta_{j-1}\right|^2}{\left|\zeta_{j-1} - \zeta_{j-2}\right| - \left|\zeta_j - \zeta_{j-1}\right|} < \epsilon_m$$

ensure the distance between the vector and the numerical range is of order ϵ_m. Assume that r is the numerical rank of the matrix $A \in \mathbb{R}^{m\times n}$, and I_p is the average number of power iteration steps. The total flop count is approximately the sum of $4mnI_p$ flops in power iteration on AA^* ($k = 0$ loop) and $\sum_{k=1}^{r} \left[(4mn + 4mk + m)\, I_p + 4mk + m\right]$ flops in power iteration on $\tilde{A}\tilde{A}^*$ ($k = 1, 2, \ldots, r$ loops), which is $4mnI_p(r+1) + m(I_p+1)r(2r+3)$ in total. Hence, the complexity of the algorithm is of $O(mn)$ when r and I_p are far less than $\min\{m, n\}$.

The high rank-revealing algorithm begins with taking QR factorization of $A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$. Since matrices A and R share the same numerical kernel $\mathcal{K}_\theta(A)$, we use the inverse iteration on R^*R to find a unit vector $\mathbf{w}_1 \in \mathcal{K}_\theta(A)$. After $\mathbf{w}_1$ is obtained, the row vector $\tau\mathbf{w}_1^*$ is stacked on top of A to form a new matrix $\tilde{A} = \begin{bmatrix} \tau\mathbf{w}_1^* \\ A \end{bmatrix}$, where $\tau = \|A\|_\infty$. The numerical nullity, namely the dimension of numerical kernel, of this new matrix $\tilde{A}$ decreases by one, and any vector in $\mathcal{K}_\theta(\tilde{A})$ belongs to $\mathcal{K}_\theta(A)$ and is orthogonal to $\mathbf{w}_1$ [13]. Hence, we update the factorization $A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$ to obtain the QR factorization of $\tilde{A} = \tilde{Q}\begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}$, followed by the inverse iteration on $\tilde{R}^*\tilde{R}$ to find a unit vector $\mathbf{w}_2 \in \mathcal{K}_\theta(A)$ with $\mathbf{w}_2 \perp \mathbf{w}_1$. The process can be recursively continued by stacking row vector $\tau\mathbf{w}_2^*$ on top of $\tilde{A}$ until the numerical nullity of the new matrix reaches zero. When the recursive process terminates, the algorithm returns an orthonormal basis $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ for $\mathcal{K}_\theta(A)$, the numerical rank $r = n - k$, and a QR factorization of *the kernel stacked matrix* $\begin{bmatrix} \tau W^* \\ A \end{bmatrix}$, where $W = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k] \in \mathbb{C}^{n\times k}$. The pseudo-code of the algorithm is shown in Fig. 2. The computations are all on the order of $O(n^2)$ except the first QR factorization of A which costs $O(mn^2)$.

In [13], the way for finding a unit vector in the numerical kernel of $\tilde{A} = \begin{bmatrix} \tau\mathbf{w}^* \\ A \end{bmatrix}$ is to use the Gauss-Newton iteration on solving the overdetermined system

$$\begin{bmatrix} \tau\mathbf{x}^* \\ A \end{bmatrix}\mathbf{x} = \begin{bmatrix} \tau \\ 0 \end{bmatrix}.$$

Namely,

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \begin{bmatrix} 2\tau\mathbf{x}_j^* \\ A \end{bmatrix}^+ \begin{bmatrix} \tau\mathbf{x}_j^*\mathbf{x}_j - \tau \\ A\mathbf{x}_j \end{bmatrix}.$$

In fact, it can be shown that this iteration is equivalent to the inverse iteration on $\tilde{A}^*\tilde{A}$. However, the flop count of inverse iteration is smaller than that of Gauss-Newton iteration. Hence, the inverse iteration is suggested.

2.2 Updating/downdating the USV-plus decomposition

Given a matrix A having numerical rank r with threshold $\theta > 0$ together with two matrices U and V whose columns form orthonormal bases for numerical range and numerical row space of A respectively, then the matrix $S \equiv U^*AV \in \mathbb{C}^{r\times r}$ has all singular values of A that are greater than θ, and the matrix $E \equiv A - USV^* \in \mathbb{C}^{m\times n}$ contains all other singular values of A. By rearranging, it yields the *USV-plus decomposition* of A,

$$A = U\,S\,V^* + E, \tag{1}$$

where USV^* and E are the *dominant part* and *noise part* of A, respectively. The low rank-revealing algorithm generates matrices U, S, and V in (1). To update/downdate

```
PSEUDO-CODE   high rank-revealing algorithm
Input:  Matrix $A \in \mathbb{C}^{m\times n}$,   numerical rank threshold $\theta > 0$
    • Compute the QR factorization of $A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$,  set $\tau = \|R\|_\infty$
    • Initialize  $r = n$  along with empty matrix $W$
    • for  $k = 1, 2, \ldots, n$  do
        – generate  a random unit vector  $\mathbf{w}$
        – for  $j = 1, 2, \ldots$  do
            ∗ forward elimination to solve $R^*\mathbf{x} = \mathbf{w}$ for $\mathbf{x}$,
              $\mathbf{u} = \mathbf{x}/\|\mathbf{x}\|_2$
            ∗ backward substitution to solve $R\mathbf{y} = \mathbf{u}$ for $\mathbf{y}$,
              $s_j = \|\mathbf{y}\|_2^{-1}$,   $\mathbf{w} = s_j\mathbf{y}$
            ∗ if  $s_j < \theta$  or  $\frac{|s_j - s_{j-1}|}{s_j} < \epsilon_m$ (when  $j > 1$)
              then break the  $j$-loop,  end if
          end do
        – if  $s_j > \theta$  then  break the  $k$-loop,  end if
        – set  $r = r - 1$
        – update  $W = [\mathbf{w}, W]$,  $R = \begin{bmatrix} \tau\mathbf{w}^* \\ R \end{bmatrix}$
        – Triangularize $R$ by Givens rotations
      end do
Output:   the numerical rank $r$
          (by-product:  numerical kernel  $W$)
```

Fig. 2 Pseudo-code of `high rank-revealing algorithm`

the USV-plus decomposition, we estimate the numerical range or numerical row space of the new matrix first, followed by one step refinement.

Suppose a USV-plus decomposition of matrix A in (1) is available and a row $\mathbf{a}^*$ is inserted into A forming a new matrix $\hat{A} \in \mathbb{C}^{(m+1)\times n}$. Let $\mathbf{y} = \mathbf{a} - VV^*\mathbf{a}$ be the projection of $\mathbf{a}$ on the numerical kernel of A. If $\|\mathbf{y}\|_2 \le \theta$, then $\mathbf{a}$ is close to the numerical row space $\mathcal{R}_\theta(A^*)$ of A, and the numerical row space of the new matrix is very close to $\mathcal{R}_\theta(A^*)$. Thus, the numerical rank of $\hat{A}$ does not increase. If $\|\mathbf{y}\|_2 > \theta$, then the new row $\mathbf{a}^*$ extends the numerical row space by one dimension. In this case, we take the augmented matrix $\hat{V} = \left[\, V \;\; \frac{\mathbf{y}}{\|\mathbf{y}\|_2} \,\right]$ as an approximation to the numerical row space of the new matrix $\hat{A}$. A refinement procedure [12] is applied, if needed, to ensure the accuracy of $\hat{V}$.

Let $\check{A}$ be the matrix resulting from deleting one row $\mathbf{r}^*$ in A. The numerical rank of $\check{A}$ may or may not decrease. Since $\mathbf{r}$ is in the row space of A, the numerical row space $\mathcal{R}_\theta(\check{A}^*)$ of $\check{A}$ is close to a subspace contained in the numerical row space of

A, which is the column space of V in (1). To obtain numerical row space $\mathcal{R}_\theta(\check{A}^*)$, we compute the "skinny" QR factorization of $\check{A}V = Q_1 R$, followed by the "skinny" LQ factorization of $Q_1^* \check{A} = L Q_2^*$, where the column space of Q_2 contains $\mathcal{R}_\theta(\check{A}^*)$. The smallest singular value σ_r of L reveals the numerical rank of $\check{A}$. If $\sigma_r > \theta$, the numerical rank of $\check{A}$ remains the same and $Q_1 L Q_2^*$ is the dominant part of $\check{A}$. Otherwise, the numerical rank decreases by one and we shall construct orthogonal matrices G and $\tilde{G}$ such that $G^* R \tilde{G} = \begin{bmatrix} \check{S} & 0 \\ 0 & \sigma_r \end{bmatrix}$ for extracting σ_r from R. By removing the last columns from $Q_1 G$ and $V\tilde{G}$ respectively, we obtain the numerical range $\check{U}$ and the numerical row space $\check{V}$ of $\check{A}$, resulting in the dominant part $\check{U}\check{S}\check{V}^*$ of $\check{A}$.

The column updating/downdating can be computed in a similar way.

2.3 Updating/downdating the kernel stacked QR factorization

Given a matrix A having numerical nullity k with respect to the threshold $\theta > 0$, and a *kernel matrix* $W \in \mathbb{C}^{n\times k}$ whose columns form an orthonormal basis for the numerical kernel of A, then all the singular values of the *kernel stacked matrix* $\begin{bmatrix} \tau W^* \\ A \end{bmatrix} \in \mathbb{C}^{(m+k)\times n}$ are greater than θ if $\tau > \theta$. The high rank-revealing algorithm generates a kernel matrix W and the QR factorization of the kernel stacked matrix:

$$Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} \tau W^* \\ A \end{bmatrix}, \quad \text{where } \tau > \theta, \tag{2}$$

which we call the *kernel stacked QR factorization* of A. For updating/downdating, we find a new kernel matrix in the first place, followed by applying the QR-updating and QR-downdating algorithms [9]. These algorithms take $O(n^2)$ flops to compute the QR factorization of the new matrix obtained by inserting/deleting a row (or column).

When the kernel stacked QR factorization (2) of A is available, the updating and downdating algorithms for high rank revealing in RankRev are outlined as follows.

Row Updating Let $\hat{A}$ be the matrix resulting from inserting a row $\mathbf{r}^*$ into A. If $\|W^*\mathbf{r}\|_2 \le \theta$, then $\mathbf{r}$ is very close to the numerical row space of A. Hence, the numerical nullity and the kernel matrix do not change. The QR factorization of the new kernel stacked matrix $\begin{bmatrix} \tau W^* \\ \hat{A} \end{bmatrix} = \tilde{Q} \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}$ can be obtained by updating (2) directly. If $\|W^*\mathbf{r}\|_2 > \theta$, then $\mathbf{r}$ annihilates the numerical kernel by one dimension. Let H be the Householder transformation with $H(W^*\mathbf{r}) = \|W^*\mathbf{r}\|_2\,\mathbf{e}_1$. Deleting the first column of WH^* yields the kernel matrix $\hat{W}$ of $\hat{A}$. The QR factorization of the new kernel stacked matrix $\begin{bmatrix} \tau \hat{W}^* \\ \hat{A} \end{bmatrix}$ can be obtained by downdating the QR factorization

$$\begin{bmatrix} \tau H W^* \\ \hat{A} \end{bmatrix} = \hat{Q} \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix} \quad \text{where } \hat{Q} = \begin{bmatrix} H & \\ & I \end{bmatrix} \tilde{Q}.$$

Row downdating Let $\check{A}$ be the matrix resulting from deleting a row from A. The algorithm begins with the QR-downdating algorithm on (2) to obtain the QR factorization

$$\begin{bmatrix} \tau W^* \\ \check{A} \end{bmatrix} = \check{Q} \begin{bmatrix} \check{R} \\ 0 \end{bmatrix}, \tag{3}$$

followed by finding a unit singular vector $\mathbf{w}_n$ of $\check{R}$ associated with the smallest singular value σ_n. If $\sigma_n > \theta$, then (3) is already the desired QR factorization. Otherwise, the numerical nullity of $\check{A}$ increases by one and $\check{W} = \begin{bmatrix} \mathbf{w}_n & W \end{bmatrix}$ is a kernel matrix of $\check{A}$. Thus, the new kernel stacked matrix $\begin{bmatrix} \tau \check{W}^* \\ \check{A} \end{bmatrix}$ has one more row on top of $\begin{bmatrix} \tau W^* \\ \check{A} \end{bmatrix}$. By applying the QR-updating algorithm on (3), the kernel stacked QR factorization of new matrix $\check{A}$ can be achieved.

Column updating and downdating procedures can be naturally derived for high rank cases [12].

3 The usage of RankRev package

The package RANKREV, available on-line at Netlib (http://www.netlib.org/numeralgo/) as the na46 package, is entirely written in the Matlab programming language and distributed as a compressed archive file. After uncompressing the archive file, the directory `RankRev` will be created with all the needed files inside. User should add the directory to the Matlab search path.

The directory consists of the following Matlab functions.

`NumericalRank`	computing the numerical rank of a matrix.
`NumericalRankUpdate`	updating the numerical rank of an inserted matrix.
`NumericalRankDowndate`	downdating the numerical rank of a deleted matrix.

These functions are in the form of M-files with generic name *function*.**m**, where *function* is the name of the function. The syntax description of each function can be accessed by the Matlab command "help *function*".

3.1 NumericalRank

The simple call of `NumericalRank` for computing the numerical rank of matrix A by the high rank-revealing algorithm is

```
>> r = NumericalRank(A);
```

The full-featured call sequence of `NumericalRank` is

```
>> [r,Basis,C] = NumericalRank(A,tol,HL);
```

Beside the numerical rank `r` and target matrix `A`, other input/output items are optional as listed below.

(i) Optional input items

`tol`: the rank decision threshold. Its default value is $\sqrt{n}\|A\|_1 \times eps$, where *eps* is the machine epsilon.

`HL`: a string parameter to switch between running high rank-revealing algorithm (`'high rank'`) and running low rank-revealing algorithm (`'low rank'`). The default value is `'high rank'`.

(ii) Optional output items

`Basis`: an orthonormal basis. If `HL` is set to be `'high rank'`, it is a basis for numerical kernel; otherwise, it is a basis for numerical range.

`C`: the Matlab cell array containing the data required by updating/downdating. The details of each cell are listed in Table 1.

Taking matrix

$$A = \begin{bmatrix} 1/3 & 1/5 & 1/7 \\ 1/3 & 2/5 & 3/7 \\ 2/3 & 2/5 & 2/7 \\ 2/3 & 4/5 & 6/7 \\ 2/3 & 3/5 & 4/7 \end{bmatrix} \tag{4}$$

with exact rank 2 as an example, one can simply call the Matlab command

```
>> [r,Basis] = NumericalRank(A,1e-12)
```

Table 1 Matlab cell array `C`

	High rank-revealing algorithm
C{1,1}	The numerical rank of A
C{2,1}	Matrix whose columns form an orthonormal kernel basis
C{3,1}	The Q in the QR decomposition of the kernel stacked matrix
C{4,1}	The R in the QR decomposition of the kernel stacked matrix
C{5,1}	Scaling factor in the kernel stacked matrix
C{6,1}	The rank decision threshold
	Low rank-revealing algorithm
C{1,1}	The numerical rank of A
C{2,1}	The U in the USV+E decomposition of A
C{3,1}	The V in the USV+E decomposition of A
C{4,1}	The S in the USV+E decomposition of A
C{5,1}	The rank decision threshold

to calculate its numerical rank within threshold 10^{-12} and an orthonormal basis for the numerical kernel by using the high rank-revealing algorithm. The screen shall display

```
r =
    2
Basis =
   0.23866718525272
   -0.79555728417573
   0.55689009892301
```

The full-featured Matlab command call to compute the numerical rank of A within threshold 10^{-8} as well as all data for updating/downdating problems by using the low rank-revealing algorithm is as follows:

```
>> [r,Basis,C] = NumericalRank(A,1e-8,'low rank')
r =
    2
Basis =
   0.19354591669367  0.36601714380583
   0.32864011800731 -0.25184170477646
   0.38709183338734  0.73203428761166
   0.65728023601462 -0.50368340955292
   0.52218603470098  0.11417543902937
C =
    [                    2]  'rank'
               [5x2 double]  'U :range'
               [3x2 double]  'V : row space'
               [2x2 double]  'S'
    [1.000000000000000e-008]  'tol'
```

Here, two columns of `Basis` form an orthonormal basis for the numerical range. Note that all data in the cell array `C` are needed for updating and downdating as demonstrated in the next subsection.

3.2 NumericalRankUpdate/NumericalRankDowndate

The full-featured call sequence of `NumericalRankUpdate` is

```
>> [r,Basis,C] = NumericalRankUpdate(A,pth,vec,C,RC);
```

that computes the numerical rank and the orthonormal basis of the updated matrix obtained by inserting row/column vector `vec` at the `pth` row/column of matrix A. `RC` is the switch parameter between running row updating algorithm (`'row'`) and running column updating algorithm (`'column'`). The fourth parameter `C` on the right-hand side contains data for updating matrix A; the third argument on the left contains data for the next updating/downdating computation.

Consider the following two matrices as examples:

$$A_1 = \begin{bmatrix} -1/3 & -1/5 & -1/7 \\ 1/3 & 1/5 & 1/7 \\ 1/3 & 2/5 & 3/7 \\ 2/3 & 2/5 & 2/7 \\ 2/3 & 4/5 & 6/7 \\ 2/3 & 3/5 & 4/7 \end{bmatrix} \begin{matrix} \longleftarrow \\ \\ \\ \\ \\ \\ \end{matrix} \quad \text{and} \quad A_3 = \begin{bmatrix} 1/3 & 1/5 & 1/7 \\ 1/3 & 2/5 & 3/7 \\ -1/3 & -1/5 & -1/7 \\ 2/3 & 2/5 & 2/7 \\ 2/3 & 4/5 & 6/7 \\ 2/3 & 3/5 & 4/7 \end{bmatrix}, \longleftarrow$$

where A_1 and A_3 are constructed by inserting vector $\left[-1/3 \ -1/5 \ -1/7 \right]^\top$ at the top and into the third row of A in (4), respectively. Hence, both are of rank 2. For calculating the numerical rank of matrix A_1, the user could execute the following commands if the numerical rank of A has been computed and the output cell C is obtained as shown in Section 3.1.

```
>> vec = [-1/3 -1/5 -1/7];
>> pth = 1;
>> [r,Basis,C] = NumericalRankUpdate(A,pth,vec,C,'row')
r =
    2
Basis =
  -0.19080645640117  0.33156350765768
   0.19080645640117 -0.33156350765768
   0.32203931262166  0.25888682554978
   0.38161291280233 -0.66312701531536
   0.64407862524332  0.51777365109955
   0.51284576902283 -0.07267668210790
C =
    [                    2]  'rank'
              [6x2 double]  'U : range'
              [3x2 double]  'V : row space'
              [2x2 double]  'S'
    [1.000000000000000e-008]  'tol'
```

The commands for calculating the numerical rank of A_3 are the same as above, except that pth should be set as 3.

The full-featured call sequence of NumericalRankDowndate is

```
>> [r,Basis,C] = NumericalRankDowndate(A,pth,C,RC);
```

that computes the numerical rank and numerical range of the new matrix resulting from deleting the pth row/column vector of matrix A. RC is the string parameter to switch between running row downdating algorithm ('row') and running column downdating algorithm ('column'). The third parameter C on the right-hand side contains data for downdating matrix A; the third argument on the left contains data for the next downdating/updating computation.

After removing the second row of A, we have a new matrix

$$A_2 = \begin{bmatrix} 1/3 & 1/5 & 1/7 \\ 2/3 & 2/5 & 2/7 \\ 2/3 & 4/5 & 6/7 \\ 2/3 & 3/5 & 4/7 \end{bmatrix}. \quad \longrightarrow$$

With all downdating information of A reserved in Matlab cell array `C`, the downdating calculation for the numerical rank and the numerical range of A_2 can be carried out by a simple call of `NumericalRankDowndate`:

```
>> pth = 2;
>> [r,Basis,C] = NumericalRankDowndate(A,pth,C,'row')
r =
     2
Basis =
  -0.20502747002880 -0.36107828740737
  -0.41005494005759 -0.72215657481474
  -0.69577544637511  0.58607400140091
  -0.55291519321635 -0.06804128670691
C =
    [                     2]  'rank'
               [4x2 double]  'U : range'
               [3x2 double]  'V : row space'
               [2x2 double]  'S'
    [1.000000000000000e-008]  'tol'
```

4 The numerical experiments

In this section, we present some numerical results of RankRev package compared with the Matlab built-in `svd` function and UTV Tools implemented by Fierro, Hansen and Hansen[7]. The package UTV Tools is perhaps the only published comprehensive rank-revealing package with updating/downdating capabilities. All the computations are performed with Matlab R2013b (Linux 64-bit version), on a 4 cores computer (Intel Core i7 2600) with 8 Gbyte RAM, running CentOS 7.0/Linux x86_64.

We use the commonly defined distance between two subspaces [9] to measure the subspace error: Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two k-dimensional subspaces in $\mathbb{R}^n$ with $0 < k < n$ and let columns of matrices $W \in \mathbb{R}^{n\times k}$ and $Z \in \mathbb{R}^{n\times(n-k)}$ form orthonormal bases of $\mathcal{S}_1$ and $\mathcal{S}_2^{\perp}$ respectively. Then the distance between $\mathcal{S}_1$ and $\mathcal{S}_2$ is $\left\| W^{\top} Z \right\|_2$.

4.1 The low rank revealing algorithm

The low rank revealing algorithm in `NumericalRank.m` is to calculate the numerical rank and the numerical range of a matrix whose rank is small. There are two existing Matlab functions having the same purpose in the UTV tools: `lurv` and `lulv`. The functions `lurv` and `lulv` have several parameters including an option to switch between the power iteration and Lanczos method for estimating singular vectors. In the tests, all the other parameters are set as default.

The test matrix is of size 3200×1600 in the form of $A = U\Sigma V^\top$ with numerical rank $k = 10$ within $\theta = 10^{-8}$. The singular values in the diagonal of Σ are distributed geometrically in two sets: $1 = \sigma_1 \geq \cdots \geq \sigma_{10} = 10^{-7}$ and $10^{-9} = \sigma_{11} \geq \cdots \geq \sigma_{1600} = 10^{-15}$. The orthogonal matrices $U \in \mathbb{R}^{3200\times 3200}$ and $V \in \mathbb{R}^{1600\times 1600}$ are randomly generated.

In Table 2, the "range error" represents the distance of the computed numerical range to the exact numerical range. $\|I - \tilde{U}^\top \tilde{U}\|_2$ measures the orthogonality of the columns for the computed numerical range $\tilde{U}$. The results show that all algorithms accurately compute numerical ranks and numerical ranges with orthonormal bases, while `RankRev` takes less running time.

4.2 The high rank revealing algorithm

The high rank revealing algorithm in `NumericalRank.m` is to calculate the numerical rank and the numerical kernel of a matrix whose rank is close to full. For the same purpose, there are four existing Matlab functions in the UTV tools : `hurv`, `hurv_a`, `hulv` and `hulv_a`. These four functions compute orthogonal decompositions of the general form $A = UTV^\top$, where U and V are orthogonal and T is triangular. The functions `hurv` and `hulv` use the generalized LINPACK condition estimator for approximating singular vectors, while `hurv_a` and `hulv_a` estimate singular vector via inverse iteration. All parameters are set as default in the following experiments.

The test matrix A is of size 3200×1600 with numerical rank fixed at $k = 1590$ within threshold $\theta = 10^{-8}$. The matrix is constructed by setting $A = U\Sigma V^\top$ with randomly generated orthogonal matrices $U \in \mathbb{R}^{3200\times 3200}$ and $V \in \mathbb{R}^{1600\times 1600}$. The singular values in the diagonal matrix Σ are distributed geometrically in two sets: $1 = \sigma_1 \geq \cdots \geq \sigma_{1590} = 10^{-7}$ and $10^{-9} = \sigma_{1591} \geq \cdots \geq \sigma_{1600} = 10^{-15}$. We use this type of matrix to test the efficiency of algorithms as well as the accuracy and orthogonality of computed numerical kernel.

Table 2 The numerical results for low rank revealing algorithms

	`svd`	`RankRev`	`lurv`		`lulv`	
			Power	Lanczos	Power	Lanczos
Time (s)	2.84	0.27	3.67	3.73	3.86	3.69
Computed rank	–	10	10	10	10	10
Range error	2.60e−10	2.15e−10	2.60e−10	2.60e−10	2.32e−10	7.32e−10
$\|I - \tilde{U}^\top \tilde{U}\|_2$	3.23e−15	3.80e−15	3.91e−15	3.87e−15	1.51e−13	1.73e−13

In Table 3, the "kernel error" represents the distance of the computed numerical kernel to the exact numerical kernel $V(:, 1591 : 1600)$. On the bottom row of the table, $\|I - \tilde{K}^\top \tilde{K}\|_2$ measures the orthogonality of matrix $\tilde{K}$ whose columns are the basis of computed numerical kernel. The results show that `hurv` miscalculates numerical rank while other codes compute numerical ranks correctly and their computed kernel bases are very "orthonormal."

4.3 The low rank approximation

There are many scientific computing problems where a low rank approximation of a matrix is of interest. Let $U \in \mathbb{R}^{m\times r}$ be a matrix whose columns span the numerical range of matrix $A \in \mathbb{R}^{m\times n}$ with dimension $r \ll n$. Then, the matrix $UU^\top A$ is a low rank approximation to A which truncates those terms with small singular values $\sigma_{r+1}, \ldots, \sigma_n$ from the SVD of $A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^\top + \cdots + \sigma_n \mathbf{u}_n \mathbf{v}_n^\top$ such that $\|A - UU^\top A\|_2 = \sigma_{r+1}$. Although the Golub-Reinsch algorithm is able to compute the singular value decomposition accurately, computing all singular values and singular vectors becomes unnecessarily expensive. Moreover, the exact numerical rank may not be essential while a low rank approximation $\tilde{A}$ is desired such that the distance $\|A - \tilde{A}\|_2$ is close to or less then the prescribed threshold θ. In this situation, the low rank revealing algorithm in `NumericalRank.m` can be applied to compute the numerical range of a low rank approximation.

Here, we consider the test matrices $A \in \mathbb{R}^{n\times n}$ with no significant gap for $n = 200, 400, 800, 1600$. The matrices are constructed by setting $A = U\Sigma V^\top$ with randomly generated orthogonal matrices $U, V \in \mathbb{R}^{n\times n}$. The singular values in the diagonal matrix Σ are distributed geometrically $1 = \sigma_1 \geq \cdots \geq \sigma_n = 10^{-15}$. Namely, $\sigma_j = (10^{-15/(n-1)})^{j-1}$ for $j = 1, 2, \ldots, n$.

Table 4 lists the numerical ranks and the gap ratios for the threshold $\theta = 10^{-3}$, and presents the numerical results for computing the low rank approximation. $\|A - \tilde{U}\tilde{U}^\top A\|_2$ indicates the distance between the test matrix and its low rank approximation $\tilde{U}\tilde{U}^\top A$, where $\tilde{U} \in \mathbb{R}^{m\times\tilde{r}}$ is the computed numerical range. The results show that the distances are nearby the threshold 10^{-3}, while the algorithm is limited in computing the exact numerical rank for matrices which have no significant rank gap.

Table 3 The numerical results for high rank revealing algorithms

	`svd`	`RankRev`	`UTVtools`			
			`hurv`	`hurv_a`	`hulv`	`hulv_a`
Time (s)	2.83	1.85	4.73	3.51	(–)	3.24
Computed rank	–	1590	1590	1590	(1600)	1590
Kernel error	1.19e−10	5.31e−08	2.61e−4	1.11e−10	(–)	3.26e-10
$\|I - \tilde{K}^\top \tilde{K}\|_2$	7.68e−15	6.66e−15	1.17e−15	1.34e−15	(–)	1.00e−15

Data in parentheses indicate incorrect computation

This experiment can be replicated by running the Matlab script `Demo_Nogap.m` in the testsuite directory.

5 The test suite

The test suite in package RANKREV contains a series of Matlab functions, in the format of M-files, that generate test matrices and their corresponding thresholds for testing rank identification and accuracy of computed subspaces. The matrices in this series are of size $2n \times n$ and have low rank or low nullity. Analogical to the test matrices in Chan and Hansen [3], the singular values are distributed *geometrically* in two sets: $1 = \sigma_1 \geq \cdots \geq \sigma_r$ and $\sigma_{r+1} \geq \cdots \geq \sigma_n = 10^{-15}$.

If matrix A is of numerical rank r within threshold $\theta > 0$, then there is a "noise" matrix E with $\|E\|_2 \leq \theta$ where $A - E$ has exact rank r. Relative perturbation $\|E\|_2 \,/\, \|A\|_2$ is often referred to as the *noise level*. Usually, we regard the magnitude of relative perturbation near machine precision, say 10^{-12}, as a low noise level, relative perturbation near 1.0, say 10^{-3}, a high noise level, and the median noise level is around 10^{-8}. The generic file name is in the format `xxnois`**.m** – the first letter `x` indicates a **h**igh/**l**ow rank matrix it will generate; the second letter indicates the **h**igh/**m**edian/**l**ow noise level that the matrix is contaminated with.

The full-featured call sequence of Matlab function `xxnois` is

```
>> [A,tol,Y] = xxnois(n);
```

Besides test matrix `A` and default threshold `tol`, all other items are optional. The input `n` determines the size of `2n×n` test matrix. The output `Y` represents the orthogonal complement of numerical kernel (for `hxnois.m`) or the orthogonal complement of numerical range (for `lxnois.m`). When the algorithm finishes computing the numerical rank, users can call the Matlab command `norm(Z'*Y)` to evaluate the subspace error, where `Z` is the computed numerical kernel/range. The detailed description of each function can be accessed by the Matlab command " `help xxnois` ".

The test suite also contains Matlab functions that generate matrices for testing updating/ downdating algorithms. All circumstances of inserting/deleting rows or columns are included. Users can take the scripts `Demo_Update.m` and `Demo_Downdate.m` as templates for testing other updating/downdating algorithms. Similarly, the Matlab script `Demo_Rank.m`, demonstrating the usage of calling `xxnois` and `NumericalRank`, can serve as a template for testing other rank-revealing algorithms.

In many rank-revealing problems in scientific or industrial applications, the matrices encountered may not have large gaps between significant larger and very small singular values, or the matrices are modeled by empirical data. In these cases, the exact numerical rank may not be essential. The test suite has two Matlab functions to generate such matrices: `imagemx` transfers a graphics file to an image matrix, and `cranfield` outputs a term-by-document matrix from the standard document collection CRANFIELD at Cornell SMART system. The entry of an image matrix represents the level of color intensity at a certain pixel, and each entry of the term-by-document matrix represents the frequency of a term appears in a document. These entries are empirical data.

Rather than testing the correctness of computed numerical ranks for these empirical matrices, we test the accuracy of the computed low-rank approximations, which seems to be essential in the application. For example, the first test in the script `Demo_Cranfield.m` intends to find a low-rank approximation to the term-by-document matrix by truncating those singular values less than 12% of the largest singular value σ_1. Therefore, the threshold is set to be $0.12 \times \sigma_1$. The accuracy of the approximation is evaluated by estimating how close the removed noise level is to 12%. In `Demo_Imagemx.m`, the similar demonstration is applied to image matrices, and the resolution comparison between the original image and the low-rank approximation image will be illustrated on Matlab figures.

The test suite also includes a Matlab function `syl_gen` that generates the Sylvester matrices with large rank gaps. Given polynomials f and g of positive degree n, write them in the form

$$\begin{aligned} f &= a_0x^n + a_1x^{n-1} + \cdots + a_n, \quad a_0 \neq 0,\\ g &= b_0x^n + b_1x^{n-1} + \cdots + b_n, \quad b_0 \neq 0. \end{aligned}$$

Then the Sylvester matrix of f and g is

$$\begin{pmatrix} \overbrace{\begin{matrix} a_0 & & \\ a_1 & a_0 & \\ \vdots & a_1 & \ddots \\ & \vdots & \ddots & a_0 \\ a_n & & & a_1 \\ & a_n & & \vdots \\ & & \ddots & \\ & & & a_n \end{matrix}}^{n \text{ columns}} & \overbrace{\begin{matrix} b_0 & & \\ b_1 & b_0 & \\ \vdots & b_1 & \ddots \\ & \vdots & \ddots & b_0 \\ b_n & & & b_1 \\ & b_n & & \vdots \\ & & \ddots & \\ & & & b_n \end{matrix}}^{n \text{ columns}} \end{pmatrix}.$$

Note that the rank of the Sylvester matrix is $2n-d$ if and only if the greatest common divisor (GCD) of f and g is of degree d. These matrices arise in numerical GCD computation, multiple roots computation [19], etc.

The function `syl_gen` is designed to generate the Sylvester matrix of two polynomials whose degrees are the same. And, the user can assign the degree of their GCD or impose perturbation to the coefficients. All input items are as follows:

`n`:	the degree of polynomials.
`d`:	the degree of numerical GCD (default: `d` = 10).
`err`:	the perturbation of coefficients (default: `err` = 0.0).

`syl_gen` outputs the following items

Table 4 The results for computing the low rank approximation

	Matrix sizes			
	200×200	400×400	800×800	1600×1600
`numerical rank` (r)	40	80	160	320
σ_r	1.14895e−03	1.07170e−03	1.03519e−03	1.01743e−03
σ_{r+1}	9.65883e−04	9.82836e−04	9.91392e−04	9.95689e−04
`gap ratio` (σ_r/σ_{r+1})	1.18953	1.09042	1.04418	1.02184
`computed rank` $(\tilde{r})$	39,40,41	79,80,81	$157 \sim 162$	$316 \sim 322$
$\left\Vert I - \tilde{U}^\top \tilde{U} \right\Vert_2$	1.77e−15	2.51e−15	4.17e−15	6.14e−15
$\left\Vert A - \tilde{U}\tilde{U}^\top A \right\Vert_2$	9.76e−04	1.09e−03	1.01e−03	1.02e−03

r is the numerical rank within $\theta = 10^{-3}$. $\tilde{U} \in \mathbb{R}^{m \times \tilde{r}}$ is the computed numerical range

`A`:	the 2n-by-2n Sylvester matrix.
`Y`:	the orthogonal complement of the numerical kernel.
`tol`:	the suggested threshold.

If the imposed perturbation `err` is smaller than 10^{-3}, the output matrix `A` will have a significant rank gap within the threshold `tol`. The script `Demo_Sylvester.m` demonstrates the call of Matlab functions `syl_gen` and `NumericalRank`.

Acknowledgements The authors are grateful to the anonymous referees for their comments and suggestions.

Research of Tsung-Lin Lee was supported in part by the Taiwan MOST Grant 105-2115-M-110-005. Research of Tien-Yien Li was supported in part by NSF under Grant DMS 11-15587. Research of Zhonggang Zeng was supported in part by NSF under Grant DMS 1620337.

References

1. Bischof, C.H., Quintana-Orti, G.: Algorithm 782: Codes for rank-revealing QR factorizations of dense matrices. ACM Trans. Math. Software **24**, 254–257 (1998)
2. Chan, T.R.: Rank revealing QR factorizations. Linear Algebra Appl. **88/89**, 67–82 (1987)
3. Chan, T.R., Hansen, P.C.: Low-rank revealing QR factorizations. Numer. Linear Algebra Appl. **1**, 33–44 (1994)
4. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Amer. Soc. Inform. Sci. **41**, 391–407 (1990)
5. Fierro, R.D., Bunch, J.R.: Bounding the subspaces from rank revealing two-sided orthogonal decompositions. SIAM J. Matrix Anal. Appl. **16**(3), 743–759 (1995)
6. Fierro, R.D., Hansen, P.C.: Low-rank revealing UTV decompositions. Numer. Algorithms **15**, 37–55 (1997)
7. Fierro, R.D., Hansen, P.C., Hansen, P.S.K.: UTV tools: MATLAB templates for rank-revealing UTV decompositions. Numer. Algorithms **20**, 165–194 (1999)
8. Golub, G.H., Klema, V., Stewart, G.W.: Rank degeneracy and least squares problems. Tech. rep. TR 456. University of Maryland, Baltimore (1976)
9. Golub, G.H., Van Loan, C.F. Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)

10. Hwang, T.M., Lin, W.W., Yang, E.K.: Rank-revealing LU Factorization. Linear Algebra Appl. **175**, 115–141 (1992)
11. Kurz, S., Rain, O., Rjasanow, S.: The adaptive cross approximation technique for the 3D boundary element method. IEEE Trans. Magnetics **38**(2), 421–424 (2002)
12. Lee, T.L., Li, T.Y., Zeng, Z.: A rank-revealing method with updating, downdating, and applications. Part II. SIAM J. Matrix Anal. Appl. **31**, 503–525 (2009)
13. Li, T.Y., Zeng, Z.: A rank-revealing method with updating, downdating, and applications. SIAM J. Matrix Anal. Appl. **26**, 918–946 (2005)
14. Mirsky, L.: Symmetric gauge functions and unitarily invariant norms. Quart. J. Math. Oxford Ser. (2) **11**, 50–59 (1960)
15. Rjasanow, S.: Adaptive cross approximation of dense matrices. In: International Association Boundary Element Methods Conference, IABEM, pp. 28–30 (2002)
16. Stewart, G.W.: An updating algorithm for subspace tracking. IEEE Trans. Signal Processing **40**, 1535–1541 (1992)
17. Stewart, G.W.: Updating a rank-revealing ULV decompositions. SIAM J. Matrix Anal. Appl. **14**, 494–499 (1993)
18. Vaccaro, R.: SVD and Signal Processing, II, Algorithms, Applications, and Architectures. Elsevier, Amsterdam (1991)
19. Zeng, Z.: Computing multiple roots of inexact polynomials. Math. Comp. **74**, 869–903 (2005)