

# A BLOCK BIDIAGONALIZATION METHOD FOR FIXED-ACCURACY LOW-RANK MATRIX APPROXIMATION\*

ERIC HALLMAN<sup>†</sup>

**Abstract.** We present randUBV, a randomized algorithm for matrix sketching based on the block Lanczos bidiagonalization process. Given a matrix  $\mathbf{A}$ , it produces a low-rank approximation of the form  $\mathbf{UBV}^T$ , where  $\mathbf{U}$  and  $\mathbf{V}$  have orthonormal columns in exact arithmetic and  $\mathbf{B}$  is block bidiagonal. In finite precision, the columns of both  $\mathbf{U}$  and  $\mathbf{V}$  will be close to orthonormal. Our algorithm is closely related to the randQB algorithms of Yu, Gu, and Li (2018) in that the entries of  $\mathbf{B}$  are incrementally generated and the Frobenius norm approximation error may be efficiently estimated. It is therefore suitable for the fixed-accuracy problem, and so is designed to terminate as soon as a user input error tolerance is reached. Numerical experiments suggest that the block Lanczos method is generally competitive with or superior to algorithms that use power iteration, even when  $\mathbf{A}$  has significant clusters of singular values.

**Key words.** randomized algorithm, low-rank matrix approximation, fixed-accuracy problem, block Lanczos

**AMS subject classifications.** 15A18, 15A23, 65F15, 65F30, 68W20

**1. Introduction.** In this paper we consider the problem of finding a quality low-rank approximation  $\tilde{\mathbf{A}}_r$  to a given matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , where we assume that  $m \geq n$ . In particular we consider the *fixed-accuracy* problem, where the desired truncation rank  $r$  is not known in advance, but we instead want to find the smallest possible  $r$  such that  $\|\mathbf{A} - \tilde{\mathbf{A}}_r\|_F < \tau$  for some tolerance  $\tau$ .

The optimal approximation can be found by computing and truncating the SVD of  $\mathbf{A}$ , but when  $\mathbf{A}$  is large this method may be impractically expensive. It is therefore increasingly common to use randomized techniques to find an approximation to the dominant subspace of  $\mathbf{A}$ : that is, to find a matrix  $\mathbf{Q} \in \mathbb{R}^{m \times r}$  with orthonormal columns so that [12]

$$(1.1) \quad \mathbf{A} \approx \mathbf{QB},$$

where  $\mathbf{B}$  is an  $r \times n$  matrix satisfying

$$(1.2) \quad \mathbf{B} = \mathbf{Q}^T \mathbf{A}.$$

Two variants on this basic approach are randomized subspace iteration and randomized block Lanczos. Algorithms 1.1 and 1.2 present prototype algorithms for each of these methods for the *fixed-rank* problem, where  $r$  is specified in advance.

---

**Algorithm 1.1** Randomized Subspace Iteration (randQB) [12, Alg. 4.3]

---

**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , rank  $r$ , integer  $\ell \geq r$ , power parameter  $p \geq 0$

**Output:**  $\mathbf{Q} \in \mathbb{R}^{m \times \ell}$  with orthonormal columns,  $\mathbf{B} \in \mathbb{R}^{\ell \times n}$

- 1: Draw a random standard Gaussian matrix  $\mathbf{\Omega} \in \mathbb{R}^{n \times \ell}$
  - 2: Form  $\mathbf{Y} = (\mathbf{AA}^T)^p \mathbf{A}\mathbf{\Omega}$
  - 3: Compute the QR factorization  $\mathbf{Y} = \mathbf{QR}$
  - 4:  $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$
- 

\* This research was supported in part by the National Science Foundation through grant DMS-1745654.

<sup>†</sup>North Carolina State University (erhallma@ncsu.edu, <https://erhallma.math.ncsu.edu/>).

---

**Algorithm 1.2** Randomized Block Lanczos [32, Alg. 1]

---

**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , block size  $b \geq 1$ , rank  $r$ , iterations  $q$  such that  $(q+1)b \geq r$

**Output:**  $\mathbf{Q} \in \mathbb{R}^{m \times (q+1)b}$  with orthonormal columns,  $\mathbf{B} \in \mathbb{R}^{(q+1)b \times n}$

- 1: Draw a random standard Gaussian matrix  $\mathbf{\Omega} \in \mathbb{R}^{n \times b}$
  - 2: Form  $\mathbf{Y} = [\mathbf{A}\mathbf{\Omega}, (\mathbf{A}\mathbf{A}^T)\mathbf{A}\mathbf{\Omega}, \dots, (\mathbf{A}\mathbf{A}^T)^q\mathbf{A}\mathbf{\Omega}]$
  - 3: Compute the QR factorization  $\mathbf{Y} = \mathbf{Q}\mathbf{R}$
  - 4:  $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$
- 

Extensions of these algorithms to the fixed-accuracy problem make use of the fact that the columns of  $\mathbf{Q}$  and rows of  $\mathbf{B}$  can be computed incrementally rather than all at once. The process can then be terminated once a user-specified error threshold has been reached, assuming the error can be efficiently computed or estimated. Algorithms for the fixed-accuracy problem are proposed in [12, 17], and more recently by Yu, Gu, and Li in [31]. One algorithm by the latter authors, `randQB_EI`, is currently the foundation for the MATLAB function `svdsketch` [18].

The algorithms cited above all rely on subspace iteration rather than the block Lanczos method, despite the fact that Krylov subspace methods are “the classical prescription for obtaining a partial SVD” [12], as with `svds` in MATLAB. One justification for the focus on subspace iteration is that convergence analysis is more complete. In particular, the block Lanczos method converges slowly when the spectrum of  $\mathbf{A}$  has a cluster larger than the block size  $b$ , and the convergence analysis becomes more complicated in this situation. In recent years, however, several works have improved the analysis for randomized block Lanczos. Analyzing Algorithm 1.2 for the case  $b \geq r$ , Musco and Musco [19] derive bounds on the approximation error that do not depend on the gaps between the singular values of  $\mathbf{A}$ . Yuan, Gu, and Li [32] derive results under the more general condition where  $\mathbf{A}$  has no singular values with multiplicity greater than  $b$ . Both papers focus mostly on theoretical results, but the latter authors make the following observation:

“A practical implementation of [Algorithm 1.2] should involve, at the very least, a reorganization of the computation to use the three-term recurrence and bidiagonalization [7], and reorthogonalization of the Lanczos vectors at each step using one of the numerous schemes that have been proposed [7, 21, 24].”

The goal of this paper is to provide a practical implementation of Algorithm 1.2, along with a method for efficiently estimating the Frobenius norm approximation error.

**1.1. Contributions.** Our main contribution is the algorithm `randUBV` (Algorithm 4.1), which uses the block Lanczos method to solve the fixed accuracy problem. It is for the most part a straightforward combination of the block Lanczos bidiagonalization process [8] shown in Algorithm 2.2 with a randomized starting matrix  $\mathbf{V}_1 = \mathbf{\Omega}$ . As such, it yields a factorization of the form  $\mathbf{UBV}^T$ , where  $\mathbf{U}$  and  $\mathbf{V}$  have orthonormal columns in exact arithmetic and  $\mathbf{B}$  is block bidiagonal. Our secondary contribution is Theorem 4.3, which establishes bounds on the accuracy of the Frobenius norm error estimate (2.6).

Our algorithm has two notable features that make it competitive with methods based on subspace iteration:

- It accepts block sizes smaller than the target rank. Contrary to what an exact arithmetic analysis would suggest, the block Lanczos method can find

multiple singular values of  $\mathbf{A}$  even when the multiplicity is greater than the block size  $b$ . Large clusters in the spectrum of  $\mathbf{A}$  are inconvenient, but not fatal.

We can therefore compare `randUBV` with adaptive methods such as `randQB_EI` when the two are run with the same block size. They will have the same cost per iteration when the latter algorithm is run with power parameter  $p = 0$ , and empirically `randUBV` converges faster. If `randQB_EI` instead uses  $p = 1$  or  $p = 2$  then `randUBV` empirically requires more iterations to converge, but each iteration costs significantly less.

- It uses *one-sided reorthogonalization*, wherein  $\mathbf{V}$  is reorthogonalized but  $\mathbf{U}$  is not. This technique was recommended in [25] for the single-vector case (i.e.,  $b = 1$ ), and leads to considerable cost savings when  $\mathbf{A}$  is sparse and  $m \gg n$ . If  $m \ll n$ , our algorithm should be run on  $\mathbf{A}^T$  instead. The matrix  $\mathbf{U}$  may slowly lose orthogonality in practice, but Theorem 4.3 shows that our error estimate (2.6) will still remain accurate.

For simplicity, we use full reorthogonalization on  $\mathbf{V}$  as opposed to more carefully targeted methods such as those discussed in [21, 24].

One other design choice merits discussion: *deflation* occurs when the blocks produced by the block Lanczos method are nearly rank-deficient and results in a reduction in the block size. In the event of deflation, we propose to augment the block Krylov space in order to keep the block column size constant. This will prevent the process from terminating early in extreme cases such as when  $\mathbf{A}$  is the identity matrix.

Numerical experiments on synthetic and real data suggest that `randUBV` generally compares favorably with `randQB` and its variants, at least on modestly sized problems.

**1.2. Outline.** The paper is organized as follows. In section 2, we review the background of QB algorithms for the fixed-accuracy problem as well as the block Lanczos method. In section 3 we discuss several implementation details including the choice of block size, deflation and augmentation, and one-sided reorthogonalization. We present our main algorithm in section 4 and establish the accuracy of the error indicator. Our numerical experiments are in section 5, and section 6 offers our concluding remarks and some avenues for future exploration.

**1.3. Notation.** Matrices, vectors, integers, and scalars will be respectively denoted by  $\mathbf{A}$ ,  $\mathbf{a}$ ,  $a$ , and  $\alpha$ . We use  $\|\mathbf{A}\|_F$  and  $\|\mathbf{A}\|_2$  for the Frobenius norm and operator norm, respectively, and  $\mathbf{I}$  for the identity matrix whose dimensions can be inferred from context. We use MATLAB notation for matrix indices: i.e.,  $\mathbf{A}(i, j)$  and  $\mathbf{A}(:, j)$  respectively represent the  $(i, j)$  element and the  $j$ -th column of  $\mathbf{A}$ .

For the cost analysis of our algorithm we use the same notation as in [17, 31]:  $C_{\text{mul}}$  and  $C_{\text{qr}}$  will represent constants so that the cost of multiplying two dense matrices of sizes  $m \times n$  and  $n \times l$  is taken to be  $C_{\text{mul}}mnl$  and the cost of computing the QR factorization of an  $m \times n$  matrix with  $m \geq n$  is taken to be  $C_{\text{qr}}mn^2$ , or  $C_{\text{qrcp}}mn^2$  if column pivoting is used.

**2. Background.** In this section we review the fixed-accuracy QB factorization algorithm `randQB_EI` and the block Lanczos bidiagonalization process.

**2.1. A fixed-accuracy QB algorithm.** In order to extend Algorithm 1.1 to the fixed-accuracy problem, Yu, Gu, and Li [31] make use of two key ideas. First, for a given block size  $b \leq \ell$  the matrix  $\mathbf{\Omega}$  can be generated  $b$  columns at a time rather than all at once, allowing the resulting factors  $\mathbf{Q}$  and  $\mathbf{B}$  to be generated incrementally.

Second, since  $\mathbf{Q}$  has orthonormal columns and  $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$ , it follows [31, Thm. 1] that

$$(2.1) \quad \|\mathbf{A} - \mathbf{QB}\|_F^2 = \|\mathbf{A} - \mathbf{QQ}^T \mathbf{A}\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{QQ}^T \mathbf{A}\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{B}\|_F^2.$$

As long as the columns of  $\mathbf{Q}$  are kept close to orthonormal, the Frobenius norm error can be efficiently estimated at each step simply by updating  $\|\mathbf{B}\|_F$ . It is therefore possible to compute the low-rank factorization  $\mathbf{QB}$  and cheaply estimate its error without ever forming the error matrix  $\mathbf{A} - \mathbf{QB}$  explicitly. Algorithm `randQB_EI` incorporates both of these ideas, the second of which is particularly useful when  $\mathbf{A}$  is sparse.

Algorithm 2.1 presents code for `randQB_EI`, which in exact arithmetic will output the same  $\mathbf{QB}$  factorization as `randQB` when run to the same rank. It is noted in [12] that a stable implementation of Algorithm 1.1 should include a reorthogonalization step after each application of  $\mathbf{A}$  or  $\mathbf{A}^T$ . The reorthogonalization step in Line 10 provides further stability.

---

**Algorithm 2.1** Blocked `randQB` algorithm (`randQB_EI`) [31, Alg. 2]

---

**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , block size  $b \geq 1$ , power parameter  $p \geq 0$ , tolerance  $\tau$

**Output:**  $\mathbf{Q} \in \mathbb{R}^{m \times \ell}$ ,  $\mathbf{B} \in \mathbb{R}^{\ell \times n}$ , such that  $\|\mathbf{A} - \mathbf{QB}\|_F < \tau$

```

1:  $\mathbf{Q} = []$ ,  $\mathbf{B} = []$ 
2:  $E = \|\mathbf{A}\|_F^2$  (Approximate costs)
3: for  $k = 1, 2, 3, \dots$  do
4:   Draw a random standard Gaussian matrix  $\mathbf{\Omega}_k \in \mathbb{R}^{n \times b}$ 
5:    $\mathbf{Q}_k = \text{qr}(\mathbf{A}\mathbf{\Omega}_k - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_k))$   $C_{\text{mul}}mnb + (k-1)C_{\text{mul}}(m+n)b^2 + C_{\text{qr}}mb^2$ 
6:   for  $j = 1 : p$  do
7:      $\tilde{\mathbf{Q}}_k = \text{qr}(\mathbf{A}^T \mathbf{Q}_k - \mathbf{B}^T (\mathbf{Q}^T \mathbf{Q}_k))$   $-\text{''}- + \text{''}- + C_{\text{qr}}nb^2$ 
8:      $\mathbf{Q}_k = \text{qr}(\mathbf{A}\tilde{\mathbf{Q}}_k - \mathbf{Q}(\mathbf{B}\tilde{\mathbf{Q}}_k))$   $-\text{''}- + \text{''}- + C_{\text{qr}}mb^2$ 
9:   end for
10:   $\mathbf{Q}_k = \text{qr}(\mathbf{Q}_k - \mathbf{Q}(\mathbf{Q}^T \mathbf{Q}_k))$   $2(k-1)C_{\text{mul}}mb^2 + C_{\text{qr}}mb^2$ 
11:   $\mathbf{B}_k = \mathbf{Q}_k^T \mathbf{A}$   $C_{\text{mul}}mnb$ 
12:   $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_k]$ 
13:   $\mathbf{B} = [\mathbf{B}^T, \mathbf{B}_k^T]^T$ 
14:   $E = E - \|\mathbf{B}_k\|_F^2$ 
15:  if  $E < \tau^2$  then stop
16: end for

```

---

Suppose that we stop Algorithm 2.1 after  $t$  iterations, and set  $\ell = tb$ . The runtime of `randQB_EI` can then be approximated as

$$(2.2) \quad T_{\text{randQB\_EI}} \approx 2C_{\text{mul}}mnl + \frac{1}{2}C_{\text{mul}}(3m+n)\ell^2 + \frac{2}{t}C_{\text{qr}}m\ell^2 + p \left( 2C_{\text{mul}}mnl + C_{\text{mul}}(m+n)\ell^2 + \frac{1}{t}C_{\text{qr}}(m+n)\ell^2 \right),$$

where the cost increases more or less proportionally to  $p+1$ . By comparison, the cost of the fixed-rank prototype algorithm `randQB` can be approximated as

$$(2.3) \quad T_{\text{randQB}} \approx 2(p+1)C_{\text{mul}}mnl + C_{\text{qr}}m\ell^2.$$

**2.2. Block Lanczos bidiagonalization.** Here we describe a block Lanczos method for reducing a matrix to block bidiagonal form. Since this method generalizes the single-vector algorithm by Golub and Kahan [6] commonly known as the Golub-Kahan-Lanczos process, we will abbreviate it as `bGKL`.

The **bGKL** process was introduced by Golub, Luk, and Overton [8] to find the largest singular values and associated singular vectors of a large and sparse matrix. Since then, it has been applied to both least squares problems [14, 27] and total least squares problems [2, 13] with multiple right-hand sides.

The process takes a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and matrix  $\mathbf{V}_1 \in \mathbb{R}^{n \times b}$  with orthonormal columns, and after  $k$  steps produces the orthonormal bases  $\mathbf{U}_{(k)} = [\mathbf{U}_1, \dots, \mathbf{U}_k]$  and  $\mathbf{V}_{(k+1)} = [\mathbf{V}_1, \dots, \mathbf{V}_{k+1}]$  satisfying

$$\begin{aligned} \text{Span}\{\mathbf{U}_{(k)}\} &= \text{Span}\{\mathbf{A}\mathbf{V}_1, \mathbf{A}(\mathbf{A}^T\mathbf{A})\mathbf{V}_1, \dots, \mathbf{A}(\mathbf{A}^T\mathbf{A})^{k-1}\mathbf{V}_1\}, \\ \text{Span}\{\mathbf{V}_{(k+1)}\} &= \text{Span}\{\mathbf{V}_1, (\mathbf{A}^T\mathbf{A})\mathbf{V}_1, \dots, (\mathbf{A}^T\mathbf{A})^k\mathbf{V}_1\}. \end{aligned}$$

Furthermore, it produces the  $kb \times (k+1)b$  block bidiagonal matrix

$$(2.4) \quad \mathbf{B}_k = \begin{bmatrix} \mathbf{R}_1 & \mathbf{L}_2 & & & \\ & \mathbf{R}_2 & \ddots & & \\ & & \ddots & \mathbf{L}_k & \\ & & & \mathbf{R}_k & \mathbf{L}_{k+1} \end{bmatrix}$$

so that at each step of the process the relations

$$(2.5) \quad \mathbf{A}\mathbf{V}_{(k)} = \mathbf{U}_{(k)}\mathbf{B}_k(:, 1 : kb) \quad \text{and} \quad \mathbf{A}^T\mathbf{U}_{(k)} = \mathbf{V}_{(k+1)}\mathbf{B}_k^T$$

are satisfied. Assuming no loss of rank, the blocks  $\{\mathbf{R}_i\}_{i=1}^k$  or  $\{\mathbf{L}_i\}_{i=1}^{k+1}$  are respectively  $b \times b$  upper and lower triangular.

---

**Algorithm 2.2** Block Lanczos bidiagonalization process (**bGKL**) [8]

---

**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , matrix  $\mathbf{V}_1 \in \mathbb{R}^{n \times b}$  with orthonormal columns

- 1:  $\mathbf{U}_0 = \mathbf{0}; \mathbf{L}_1 = \mathbf{0}$  (Approximate costs)
  - 2: **for**  $k = 1, 2, \dots$  **do**
  - 3:    $\mathbf{U}_k\mathbf{R}_k = \text{qr}(\mathbf{A}\mathbf{V}_k - \mathbf{U}_{k-1}\mathbf{L}_k)$   $C_{\text{mul}}mnb + \frac{1}{2}C_{\text{mul}}mb^2 + C_{\text{qr}}mb^2$
  - 4:    $\mathbf{V}_{k+1}\mathbf{L}_{k+1}^T = \text{qr}(\mathbf{A}^T\mathbf{U}_k - \mathbf{V}_k\mathbf{R}_k^T)$   $C_{\text{mul}}mnb + \frac{1}{2}C_{\text{mul}}nb^2 + C_{\text{qr}}nb^2$
  - 5: **end for**
- 

The basic outline of the process is given in Algorithm 2.2, where the costs assume no loss of rank in the blocks  $\{\mathbf{R}_i\}_{i=1}^k$  or  $\{\mathbf{L}_i\}_{i=1}^{k+1}$ . We note that the original algorithm in [8] is organized so that  $\mathbf{B}_k$  is square at the end of each iteration. Our current presentation more directly mimics the **QB** factorization, since  $\mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T = \mathbf{U}_{(k)}\mathbf{U}_{(k)}^T\mathbf{A}$  by the second relation in (2.5). It follows that in exact arithmetic the identity

$$(2.6) \quad \|\mathbf{A} - \mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{B}_k\|_F^2$$

will hold, and so the **bGKL** process can be readily adapted to find a fixed-accuracy approximation to  $\mathbf{A}$ .

Suppose that we stop the process after  $t$  iterations and set  $\ell = tb$ . The runtime of the **bGKL** process can then be approximated as

$$(2.7) \quad T_{\text{bGKL}} \approx 2C_{\text{mul}}mnl + \frac{1}{2t}C_{\text{mul}}(m+n)\ell^2 + \frac{1}{t}C_{\text{qr}}(m+n)\ell^2.$$

At this point, it is not fair to compare this cost to the cost of (2.2) because we have not yet accounted for the cost of reorthogonalization in **bGKL**, which is necessary for stability. Nonetheless, it suggests that we may be able to obtain an algorithm based on **bGKL** that costs no more per iteration than **randQB\_EI** with power parameter  $p = 0$ .

**3. Implementation details.** In this section we discuss how to handle several important issues in the implementation of our fixed-accuracy algorithm. The first concerns the difficulty the Lanczos method encounters when  $\mathbf{A}$  has large singular value clusters. The second is the matter of ensuring that the columns of  $\mathbf{U}_{(k)}$  and  $\mathbf{V}_{(k)}$  remain close to orthonormal, and the third is the use of deflation and augmentation when the blocks  $\mathbf{R}_k$  or  $\mathbf{L}_k$  are rank-deficient.

**3.1. Block size and singular value clusters.** It is known that if  $\mathbf{A}$  has a singular value with multiplicity greater than the block size  $b$ , then in exact arithmetic the block Lanczos process will recover at most  $b$  of those singular values. More generally, if the spectrum of  $\mathbf{A}$  has a cluster of size greater than  $b$  then the approximate singular vectors recovered by the Lanczos process may converge slowly. This behavior stands in stark contrast to that of blocked subspace iteration methods such as `randQB_EI`, whose outputs do not in exact arithmetic depend on  $b$ .

For the first situation—singular values with multiplicity greater than  $b$ —classical results tend to examine a restricted problem. Saad [23] notes that the Lanczos process would simply behave as though it were being performed on a restricted matrix  $\mathbf{A}|_S$  whose singular values<sup>1</sup> had multiplicity at most  $b$ . There is therefore “no loss of generality” in assuming that the singular values of  $\mathbf{A}$  have multiplicity bounded by  $b$  for the purpose of analyzing convergence rates. Other more recent works restrict their attention to the case where the cluster size is bounded by  $b$  [16], or where  $b$  is greater than or equal to the target rank  $r$  [19, 28, 4].

The analysis of Yuan, Gu, and Li [32] makes an important advancement by allowing for cluster sizes (though not multiplicity) greater than  $b$ , and showing that even within a large cluster the recovered singular values will converge superlinearly in the number of Lanczos iterations. Their numerical experiments on real-world data suggest that smaller block sizes generally lead to faster convergence with respect to the number of flops expended.

As it turns out, even singular values with multiplicity greater than  $b$  are not fatal to the Lanczos process. Parlett [21] notes that since “rounding errors introduce components in all directions”, even repeated singular vectors<sup>2</sup> will eventually be found. Simon and Zha [25] add the caveat that the singular vectors will not converge in consecutive order: the Lanczos process will likely find several smaller singular values of  $\mathbf{A}$  before it finds copies of the larger repeated ones. What we should expect in practice is that a singular value of multiplicity greater than  $b$  (or a cluster of comparable size) will delay convergence, but not prevent it entirely.

Thus in spite of complications in the *analysis* of the block Lanczos method, using a smaller block size can be quite effective in practice. Even when  $\mathbf{A}$  has clusters larger than the block size, we can obtain a good approximation simply by increasing the number of Lanczos iterations. Our numerical experiments support this notion: although we can construct synthetic examples for which `randUBV` is inferior to methods that use subspace iteration, our algorithm performs quite well on a real-world example with large clusters.

**3.1.1. Adaptive block size.** An alternate method for dealing with clusters is offered in [30] and explored further in [1, 33]: instead of keeping the block size constant, we may periodically augment the block Krylov space with new vectors in

<sup>1</sup>Strictly speaking, Saad’s analysis is for block Lanczos tridiagonalization applied to a symmetric matrix as opposed to Lanczos bidiagonalization applied a rectangular matrix. Our focus is on bidiagonalization, but the two processes are closely related.

<sup>2</sup>See footnote 1.

order to better approximate clusters. The rough idea would be to monitor the singular values of  $\mathbf{B}_k$ , and to increase the block size  $b$  so that it remains larger than the largest cluster in  $\mathbf{B}_k$ . For the sake of keeping the implementation of our algorithm simple, we leave this extension for future exploration.

**3.2. One-sided reorthogonalization.** In exact arithmetic, the matrices  $\mathbf{U}_{(k)}$  and  $\mathbf{V}_{(k)}$  will have orthonormal columns. In practice, they will quickly lose orthogonality due to roundoff error, and so we must take additional steps to mitigate this loss of orthogonality.

For the single-vector case  $b = 1$ , Simon and Zha [25] observe that it may suffice to reorthogonalize only one of  $\mathbf{U}_{(k)}$  or  $\mathbf{V}_{(k)}$  in order to obtain a good low-rank approximation. They suggest that if the columns of  $\mathbf{V}_{(k)}$  alone are kept close to orthonormal, then  $\mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T$  will remain a good approximation to  $\mathbf{A}$  regardless of the orthogonality of  $\mathbf{U}_{(k)}$ . Separately, experiments by Fong and Saunders [5] in the context of least-squares problems suggest that keeping  $\mathbf{V}_{(k)}$  orthonormal to machine precision  $\epsilon_{\text{mach}}$  might be enough to keep  $\mathbf{U}_{(k)}$  orthonormal to at least  $\mathcal{O}(\sqrt{\epsilon_{\text{mach}}})$ , at least until the least-squares solver reaches a relative backward error of  $\sqrt{\epsilon_{\text{mach}}}$ . For the sake of computational efficiency, we therefore choose to explicitly reorthogonalize  $\mathbf{V}_{(k)}$  but not  $\mathbf{U}_{(k)}$  (assuming that  $m \geq n$ ).

Reorthogonalization can take up a significant portion of the runtime of our algorithm, particularly if  $\mathbf{A}$  is sparse. However, it is known for the Lanczos process that orthogonality is lost only in the direction of singular vectors that have already converged [20]. Thus in a high-quality implementation, it should be possible to save time by orthogonalizing each block  $\mathbf{V}_k$  against a smaller carefully chosen set of vectors obtained from  $\mathbf{V}_{(k-1)}$  (see [21, 10, 24] for a few such proposals). In our implementation, we use full reorthogonalization for simplicity. We note that even if  $\mathbf{A}$  is square, full reorthogonalization will cost no more than the equivalent step in `randQB_EI` (line 10 of Algorithm 2.1).

**3.3. Deflation.** In practice, the block Lanczos process may yield blocks  $\mathbf{R}_k$  or  $\mathbf{L}_k$  that are rank-deficient or nearly so. Here and with other block Krylov methods, it is typical to reduce the block size  $b$  in response so that  $\mathbf{R}_k$  and  $\mathbf{L}_k$  retain full row rank and column rank, respectively. This process is known as *deflation*. For more background, we refer the reader to the survey paper by Gutknecht [11] and the references therein.

In the context of solving systems with multiple right-hand sides, Gutknecht stresses that deflation is highly desirable. Indeed, when solving a system such as  $\mathbf{A}\mathbf{X} = \mathbf{B}$ , it is precisely the dimension reduction resulting from deflation that gives block methods an advantage over methods that solve each right hand side separately. In this context, deflation might occur if  $\mathbf{B}$  is itself rank-deficient, or if  $\mathbf{B}$  has some notable rank structure in relation to the matrix  $\mathbf{A}$ . When running block Lanczos with a randomly chosen starting matrix  $\mathbf{V}_1$  (i.e.,  $\mathbf{V}_1 = \text{qr}(\mathbf{\Omega})$  and  $\mathbf{\Omega}$  is a standard Gaussian matrix), we do not expect deflation to occur frequently since  $\mathbf{\Omega}$  is not likely to have any notable structure with respect to  $\mathbf{A}$ . Nonetheless, a reliable implementation should be prepared for the possibility, and so we examine the details here.

Björck [2] proposes computing the QR factorizations in lines 3–4 of Algorithm 2.2 using Householder reflections without column pivoting. The resulting matrix  $\mathbf{B}_k$  will be not just block bidiagonal, but a banded matrix whose effective bandwidth begins at  $b$  and decreases with each deflation. Hnětynková et al. [13] refer to  $\mathbf{B}_k$  as a *b-wedge shaped matrix*. If the effective bandwidth decreases to zero, the bidiagonalization process will terminate.

**Algorithm 3.1** Deflated QR (def1QR)**Input:**  $\mathbf{X} \in \mathbb{R}^{m \times n}$ , deflation tolerance  $\delta$ **Output:**  $\mathbf{Q} \in \mathbb{R}^{m \times s}$  with orthonormal columns,  $\mathbf{R} \in \mathbb{R}^{s \times n}$ , rank  $s$ 

- 1: Compute the pivoted QR factorization  $\mathbf{X}\mathbf{\Pi} = \widehat{\mathbf{Q}}\widehat{\mathbf{R}}$
- 2: Find the largest  $s$  such that  $|\widehat{\mathbf{R}}(s, s)| \geq \delta$
- 3:  $\mathbf{R} = \widehat{\mathbf{R}}(1 : s, :)\mathbf{\Pi}^T$
- 4:  $\mathbf{Q} = \widehat{\mathbf{Q}}(:, 1 : s)$

We propose to instead use QR with column pivoting, which is slower and less elegant but simpler to implement in terms of readily available subroutines. The procedure is outlined in Algorithm 3.1, where the deflation tolerance  $\delta$  is presumably somewhat larger than  $\epsilon_{\text{mach}}\|\mathbf{A}\|_2$ . Lines 3–4 of Algorithm 2.2 would use this modified routine in place of unpivoted QR, and as Björck [2] notes the recurrence in those lines will still work in the presence of deflation.

**3.4. Augmentation.** When using block Lanczos to solve systems of linear equations, deflation can be highly beneficial. In the context of matrix sketching, it is less desirable. Consider an extreme example where the columns of  $\mathbf{V}_1$  are right singular vectors of  $\mathbf{A}$ : the Lanczos process will terminate after a single iteration, returning an approximation of the form  $\mathbf{A} \approx \mathbf{U}_1\mathbf{\Sigma}\mathbf{V}_1^T$ . Termination at this point would yield accurate singular vectors, but the factorization may not approximate  $\mathbf{A}$  to within the desired error tolerance.

As mentioned before, we do not expect deflation to occur frequently if  $\mathbf{V}_1$  is chosen randomly. However, if we do not make any further adjustments for deflation our algorithm would fail to converge on cases as simple as  $\mathbf{A} = \mathbf{I}$ . In order to make our method more robust, we will replace any deflated vectors with new randomly drawn ones in order to keep the block column size constant. Similar augmentation techniques have been proposed to prevent breakdown in the case of the nonsymmetric Lanczos process [29] and GMRES [22].

More specifically, if Algorithm 3.1 returns a factorization  $\mathbf{V}_k\mathbf{L}_k^T$  with rank less than  $b$ , we generate a standard Gaussian matrix  $\mathbf{\Omega}_k$  so that  $[\mathbf{V}_k, \mathbf{\Omega}_k]$  has  $b$  columns. We then orthogonalize  $\mathbf{\Omega}_k$  against  $\mathbf{V}_k$  and  $\mathbf{V}_{(k-1)}$ , obtaining  $\mathbf{V}'_k$ . The resulting matrix  $[\mathbf{V}_k, \mathbf{V}'_k]$  is then used in place of  $\mathbf{V}_k$  in the next step of the Lanczos process.

In keeping with the spirit of one-sided reorthogonalization, we do not augment  $\mathbf{U}_k$  if a block  $\mathbf{R}_k$  is found to be rank deficient. This will allow us to avoid accessing the matrix  $\mathbf{U}_{(k-1)}$  while the block Lanczos process is running. As a consequence, the blocks of  $\mathbf{B}_k$  will each have  $b$  columns, but some may have fewer than  $b$  rows.

We observe that in the presence of augmentation, the space  $\text{Span}\{\mathbf{V}_{(k)}\}$  will not be a block Krylov space, but will instead be the sum of multiple block Krylov spaces with different dimensions. As of the time of writing we are not aware of any convergence results for this more general case.

**4. Fixed-accuracy algorithm.** Algorithm 4.1 presents code for randUBV. Ignoring the augmentation step in line 16, the cost is more or less equal to the cost of bGKL plus the cost of reorthogonalizing  $\mathbf{V}_{k+1}$  in Line 11. Thus if we stop the process after  $t$  iterations and set  $\ell = tb$ , the total cost is approximately

$$(4.1) \quad T_{\text{randUBV}} \approx 2C_{\text{mul}}mnl + C_{\text{mul}}n\ell^2 + \frac{1}{2t}C_{\text{mul}}(m+n)\ell^2 + \frac{1}{t}C_{\text{qr}}(m+n)\ell^2.$$



Comparing this quantity to (2.2), we see that `randUBV` requires fewer floating points operations than `randQB_EI` when run for the same number of iterations, even when the latter is run with power parameter  $p = 0$ . In particular, the cost of one-sided reorthogonalization is only  $\mathcal{O}(n\ell^2)$  while the stabilization steps in lines 5 and 10 of `randQB_EI` cost  $\mathcal{O}((m+n)\ell^2)$ . We can therefore expect that if  $\mathbf{A}$  is sparse and  $m \gg n$ , `randUBV` may run significantly faster.

---

**Algorithm 4.1** Blocked Bidiagonalization algorithm (`randUBV`)

---

**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , block size  $b$ , relative error  $\tau$ , deflation tolerance  $\delta$

**Output:**  $\mathbf{U}, \mathbf{B}, \mathbf{V}$ , such that  $\|\mathbf{A} - \mathbf{UBV}^T\|_F < \tau$

```

1:  $E = \|\mathbf{A}\|_F^2$  (Approximate costs)
2: Draw a random standard Gaussian matrix  $\mathbf{\Omega} \in \mathbb{R}^{n \times b}$ 
3:  $\mathbf{V}_1 = \text{qr}(\mathbf{\Omega})$   $C_{\text{qr}}nb^2$ 
4:  $\mathbf{U}_1 = \mathbf{0}; \mathbf{L}_1 = \mathbf{0}$ 
5:  $\mathbf{V} = \mathbf{V}_1; \mathbf{U} = \mathbf{U}_1$ 
6: for  $k = 1, 2, 3, \dots$  do
7:    $[\mathbf{U}_k, \mathbf{R}_k] = \text{def1QR}(\mathbf{AV}_k - \mathbf{U}_{k-1}\mathbf{L}_k, \delta)$   $C_{\text{mul}}mnb + \frac{1}{2}C_{\text{mul}}mb^2 + C_{\text{qr}}mb^2$ 
8:    $\mathbf{U} = [\mathbf{U}, \mathbf{U}_k]$ 
9:    $E = E - \|\mathbf{R}_k\|_F^2$ 
10:   $\mathbf{V}_{k+1} = \mathbf{A}^T\mathbf{U}_k - \mathbf{V}_k\mathbf{R}_k^T$   $C_{\text{mul}}mnb + \frac{1}{2}C_{\text{mul}}nb^2$ 
11:   $\mathbf{V}_{k+1} = \mathbf{V}_{k+1} - \mathbf{V}(\mathbf{V}^T\mathbf{V}_{k+1})$   $2kC_{\text{mul}}nb^2$ 
12:   $[\mathbf{V}_{k+1}, \mathbf{L}_{k+1}, s] = \text{def1QR}(\mathbf{V}_{k+1}, \delta)$   $C_{\text{qr}}nb^2$ 
13:   $\mathbf{V} = [\mathbf{V}, \mathbf{V}_{k+1}]$ 
14:  if  $s < b$  then
15:    Draw a random standard Gaussian matrix  $\mathbf{\Omega}_k \in \mathbb{R}^{n \times (b-s)}$ 
16:     $\mathbf{V}'_{k+1} = \text{qr}(\mathbf{\Omega}_k - \mathbf{V}(\mathbf{V}^T\mathbf{\Omega}_k))$   $2kC_{\text{mul}}nb(b-s) + C_{\text{qr}}n(b-s)^2$ 
17:     $\mathbf{V} = [\mathbf{V}, \mathbf{V}'_{k+1}]$ 
18:  end if
19:   $E = E - \|\mathbf{L}_{k+1}\|_F^2$ 
20:  if  $E < \tau^2\|\mathbf{A}\|_F^2$  then stop
21: end for

```

---

Since our focus is on the fixed-accuracy algorithm, however, different algorithms (and for `randQB_EI`, different power parameters  $p$ ) will converge after different numbers of iterations. We must therefore consider not just the cost per iteration, but how quickly the approximations converge. We discuss this matter further along with the numerical experiments in section 5.

**4.1. Approximation accuracy.** It is noted in [31] that due to cancellation, the computed value of  $E = \|\mathbf{A}\|_F^2 - \|\mathbf{B}\|_F^2$  may be inaccurate when  $E$  is very small. In order to estimate the error  $E$  to within a relative tolerance of  $\gamma$  (say,  $\gamma = 1\%$ ), the authors suggest that the absolute accuracy tolerance  $\tau$  for the QB factorization should be set large enough to satisfy

$$(4.2) \quad \tau > \sqrt{E} \geq \sqrt{\frac{4\epsilon_{\text{mach}}}{\gamma}} \|\mathbf{A}\|_F,$$

where  $\epsilon_{\text{mach}}$  is the machine precision. In short, the proposed method of error estimation cannot reliably estimate a relative error below  $2\sqrt{\epsilon_{\text{mach}}}$ .

We provide a similar analysis in order to account for deflation and loss of orthogonality of  $\mathbf{U}_{(k)}$ . In particular, we show that the error estimate can remain accurate

even as  $\mathbf{U}_{(k)}$  loses orthogonality in practice. To that end, we define the *local loss of orthogonality* of a matrix as follows:

DEFINITION 4.1. *Given a matrix  $\mathbf{U}_{(k)} = [\mathbf{U}_1, \dots, \mathbf{U}_k]$ , the local loss of orthogonality of  $\mathbf{U}_{(k)}$  is defined as*

$$\varepsilon_k = \max \left\{ \max_{1 \leq i \leq k} \|\mathbf{U}_i^T \mathbf{U}_i - \mathbf{I}\|_2, \max_{2 \leq i \leq k} \|\mathbf{U}_{i-1}^T \mathbf{U}_i\|_2 \right\}$$

The main idea is that we do not require  $\|\mathbf{U}_{(k)}^T \mathbf{U}_{(k)} - \mathbf{I}\|_2$  to be small. Instead, we need only the milder condition that adjacent blocks be close to orthogonal. This idea bears some resemblance to the work [26], which uses local recurrence formulas to show that certain error estimates for the conjugate gradient method remain accurate in a finite precision setting.

LEMMA 4.2. *Consider the matrix  $\mathbf{U}_{(k)} = [\mathbf{U}_1, \dots, \mathbf{U}_k]$ , and let  $\varepsilon_k$  denote the local loss of orthogonality of  $\mathbf{U}_{(k)}$ . Let  $\mathbf{B}_k$  be a block upper bidiagonal matrix whose blocks are partitioned conformally with those of  $\mathbf{U}_{(k)}$ . Then*

$$\|\mathbf{U}_{(k)} \mathbf{B}_k\|_F^2 = (1 + \theta) \|\mathbf{B}_k\|_F^2, \quad |\theta| \leq 2\varepsilon_k.$$

*Proof.* We will find the squared Frobenius norm of  $\mathbf{U}_{(k)} \mathbf{B}_k$  one block column at a time, and use the fact that since  $\mathbf{B}_k$  is block bidiagonal, each block column in the product uses at most two adjacent blocks of  $\mathbf{U}_{(k)}$ .

Let  $\{\mathbf{R}_i\}_{i=1}^k$  denote the blocks on the main block diagonal of  $\mathbf{B}_k$ , and let  $\{\mathbf{L}_i\}_{i=2}^{k+1}$  denote the off-diagonal blocks. Then for  $2 \leq i \leq k$ , the squared Frobenius norm of the  $i$ -th block column of  $\mathbf{U}_{(k)} \mathbf{B}_k$  is given by

$$(4.3) \quad \|\mathbf{U}_{i-1} \mathbf{L}_i + \mathbf{U}_i \mathbf{R}_i\|_F^2 = \|\mathbf{U}_{i-1} \mathbf{L}_i\|_F^2 + \|\mathbf{U}_i \mathbf{R}_i\|_F^2 + 2 \operatorname{tr}(\mathbf{R}_i^T \mathbf{U}_i^T \mathbf{U}_{i-1} \mathbf{L}_i).$$

Examining the first term, it can be seen that

$$\begin{aligned} \|\mathbf{U}_{i-1} \mathbf{L}_i\|_F^2 &= \operatorname{tr}(\mathbf{L}_i^T \mathbf{U}_{i-1}^T \mathbf{U}_{i-1} \mathbf{L}_i) \\ &= \operatorname{tr}(\mathbf{L}_i^T (\mathbf{U}_{i-1}^T \mathbf{U}_{i-1} - \mathbf{I}) \mathbf{L}_i) + \operatorname{tr}(\mathbf{L}_i^T \mathbf{L}_i) \\ &= (1 + \theta_1) \|\mathbf{L}_i\|_F^2, \end{aligned}$$

where  $|\theta_1| \leq \varepsilon_k$ . A similar result applies to the term  $\|\mathbf{U}_i \mathbf{R}_i\|_F^2$ . As for the final term, we find that

$$\begin{aligned} 2 |\operatorname{tr} \mathbf{R}_i^T \mathbf{U}_i^T \mathbf{U}_{i-1} \mathbf{L}_i| &\leq 2 \|\mathbf{U}_i^T \mathbf{U}_{i-1}\|_2 \|\mathbf{R}_i\|_F \|\mathbf{L}_i\|_F \\ &\leq 2\varepsilon_k \|\mathbf{R}_i\|_F \|\mathbf{L}_i\|_F, \\ &\leq \varepsilon_k (\|\mathbf{R}_i\|_F^2 + \|\mathbf{L}_i\|_F^2). \end{aligned}$$

By adding these expressions back together we arrive at the bound

$$(4.4) \quad \|\mathbf{U}_{i-1} \mathbf{L}_i + \mathbf{U}_i \mathbf{R}_i\|_F^2 = (1 + \theta) (\|\mathbf{R}_i\|_F^2 + \|\mathbf{L}_i\|_F^2), \quad |\theta| \leq 2\varepsilon_k,$$

so the desired relative error bound holds for each block column (the first and last columns may be checked separately). The main claim then follows by summing over the block columns.  $\square$

Next, we observe that with one-sided reorthogonalization of  $\mathbf{V}_{(k)}$  and in the absence of deflation, the *first* relation in (2.5) will remain accurate to machine precision regardless of the orthogonality of  $\mathbf{U}_{(k)}$  (as noted in [25], the second relation will not). In the presence of deflation, the first relation must be amended slightly. We rewrite it as

$$(4.5) \quad \mathbf{A}\mathbf{V}_{(k)} = \mathbf{U}_{(k)}\mathbf{B}'_k + \mathbf{D}_k,$$

where  $\mathbf{B}'_k$  is shorthand for  $\mathbf{B}_k(:, 1 : kb)$  and  $\mathbf{D}_k$  is a matrix accounting for all deflations in  $\mathbf{U}_{(k)}$ . Assuming the column pivoting in Algorithm 3.1 selects at each step the column with the largest 2-norm, it can be verified that  $\|\mathbf{D}_k\|_F \leq \delta\sqrt{d}$ , where  $\delta$  is the deflation tolerance and  $d$  is the total number of columns that have been removed from  $\mathbf{U}_{(k)}$  through deflation.

We now show that the error estimate  $E = \|\mathbf{A}\|_F^2 - \|\mathbf{B}_k\|_F^2$  will remain accurate up to terms involving the deflation tolerance and the local loss of orthogonality in  $\mathbf{U}_{(k)}$ . The proof makes the simplifying assumptions that  $\mathbf{V}_{(k+1)}$  has orthonormal columns and that there is no rounding error term in (4.5), but accounting for both of these effects will change the bound (4.6) by at most  $\mathcal{O}(\epsilon_{\text{mach}}\|\mathbf{A}\|_F^2)$ . The proof also ignores the effect of cancellation in the computation of  $E$ , so as with [31] we cannot expect to reliably estimate a relative error below  $\sqrt{\epsilon_{\text{mach}}}$ .

**THEOREM 4.3.** *Given a matrix  $\mathbf{A}$ , let  $\mathbf{U}_{(k+1)}$ ,  $\mathbf{B}'_{k+1}$ , and  $\mathbf{V}_{(k+1)}$  be as produced by Algorithm 4.1 with deflation tolerance  $\delta$ . Let  $\epsilon_{k+1}$  denote the local loss of orthogonality of  $\mathbf{U}_{(k+1)}$ . Assume that  $\mathbf{V}_{(k+1)}$  has orthonormal columns. Assume that (4.5) holds exactly at each iteration, and let  $d$  be the number of columns removed from  $\mathbf{U}_{(k+1)}$  due to deflation. Finally, let  $E = \|\mathbf{A}\|_F^2 - \|\mathbf{B}\|_F^2$ . Then*

$$(4.6) \quad \|\mathbf{A} - \mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T\|_F^2 \leq E + 4\epsilon_{k+1}\|\mathbf{A}\|_F^2 + 2\delta\sqrt{d}(1 + 2\epsilon_{k+1})\|\mathbf{A}\|_F.$$

*Proof.* First, by assuming the columns of  $\mathbf{V}_{(k+1)}$  are orthonormal we find that

$$(4.7) \quad \|\mathbf{A} - \mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T\|_F^2 = \|\mathbf{A}\|_F^2 + \|\mathbf{U}_{(k)}\mathbf{B}_k\|_F^2 - 2\text{tr}(\mathbf{A}\mathbf{V}_{(k+1)}\mathbf{B}_k^T\mathbf{U}_{(k)}^T).$$

By assuming that (4.5) holds exactly at each step, we also get the identity

$$\mathbf{A}\mathbf{V}_{(k+1)} = \mathbf{U}_{(k+1)}\mathbf{B}'_{k+1} + \mathbf{D}_{k+1} = \mathbf{U}_{(k)}\mathbf{B}_k + [\mathbf{0}, \mathbf{U}_{k+1}\mathbf{R}_{k+1}] + \mathbf{D}_{k+1},$$

where  $\|\mathbf{D}_{k+1}\|_F \leq \delta\sqrt{d}$ . It follows that

$$(4.8) \quad \text{tr}(\mathbf{A}\mathbf{V}_{(k+1)}\mathbf{B}_k^T\mathbf{U}_{(k)}^T) = \|\mathbf{U}_{(k)}\mathbf{B}_k\|_F^2 + \text{tr}(\mathbf{U}_k^T\mathbf{U}_{k+1}\mathbf{R}_{k+1}\mathbf{L}_{k+1}^T) + \text{tr}(\mathbf{D}_{k+1}\mathbf{B}_k^T\mathbf{U}_{(k)}^T).$$

From the definition of  $\epsilon_{k+1}$  we have

$$(4.9) \quad |\text{tr}(\mathbf{U}_k^T\mathbf{U}_{k+1}\mathbf{R}_{k+1}\mathbf{L}_{k+1}^T)| \leq \|\mathbf{U}_k^T\mathbf{U}_{k+1}\|_2\|\mathbf{R}_{k+1}\|_F\|\mathbf{L}_{k+1}\|_F \leq \epsilon_{k+1}\|\mathbf{A}\|_F^2,$$

and since  $\|\mathbf{D}_{k+1}\|_F \leq \delta\sqrt{d}$  we also have

$$(4.10) \quad \left| \text{tr}(\mathbf{D}_{k+1}\mathbf{B}_k^T\mathbf{U}_{(k)}^T) \right| \leq \|\mathbf{D}_{k+1}\|_F\|\mathbf{U}_{(k)}\mathbf{B}_k\|_F \leq \delta\sqrt{d}\|\mathbf{U}_{(k)}\mathbf{B}_k\|_F.$$

Lemma 4.2 gives us bounds on  $\|\mathbf{U}_{(k)}\mathbf{B}_k\|_F^2$ , so by returning to (4.7) and using (4.8), (4.9), and (4.10), we conclude that

$$\begin{aligned} \|\mathbf{A} - \mathbf{U}_{(k)}\mathbf{B}_k\mathbf{V}_{(k+1)}^T\|_F^2 &= \|\mathbf{A}\|_F^2 + \|\mathbf{U}_{(k)}\mathbf{B}_k\|_F^2 - 2\text{tr}(\mathbf{A}\mathbf{V}_{(k+1)}\mathbf{B}_k^T\mathbf{U}_{(k)}^T) \\ &\leq \|\mathbf{A}\|_F^2 - \|\mathbf{U}_{(k)}\mathbf{B}_k\|_F^2 + 2\epsilon_{k+1}\|\mathbf{A}\|_F^2 + 2\delta\sqrt{d}\|\mathbf{U}_{(k)}\mathbf{B}_k\|_F \\ &\leq E + 4\epsilon_{k+1}\|\mathbf{A}\|_F^2 + 2\delta\sqrt{d}(1 + 2\epsilon_{k+1})\|\mathbf{A}\|_F. \quad \square \end{aligned}$$

Thus as long as *local* orthogonality is maintained for  $\mathbf{U}_{(k)}$  and as long as the number of deflations is not too large, we can expect  $E$  to remain an accurate estimate of the Frobenius norm approximation error, at least when the error tolerance is not too small.

**4.2. Postprocessing of  $\mathbf{B}$ .** Recall that our original goal for the fixed-accuracy problem was not just to find a factorization that satisfies the bound  $\|\mathbf{A} - \mathbf{UBV}^T\|_F < \tau$ , but to find the factorization with the smallest rank that does so. In order to accomplish this, we may compute the SVD of  $\mathbf{B}$  as  $\mathbf{B} = \hat{\mathbf{U}}\hat{\Sigma}\hat{\mathbf{V}}^T$ , truncate it to the smallest rank  $r$  such that  $\|\mathbf{A} - \hat{\mathbf{U}}_r\hat{\Sigma}_r\hat{\mathbf{V}}_r^T\|_F < \tau$ , then approximate the left and right singular vectors of  $\mathbf{A}$  by  $\mathbf{U}\hat{\mathbf{U}}_r$  and  $\mathbf{V}\hat{\mathbf{V}}_r$ . It should be noted that since  $\mathbf{B}$  is a block bidiagonal matrix, its SVD can in theory be computed more efficiently than if  $\mathbf{B}$  were dense. Algorithms for computing the SVD typically first reduce the matrix to bidiagonal form [6], and  $\mathbf{B}$  can be efficiently reduced to this form using band reduction techniques as in [15].

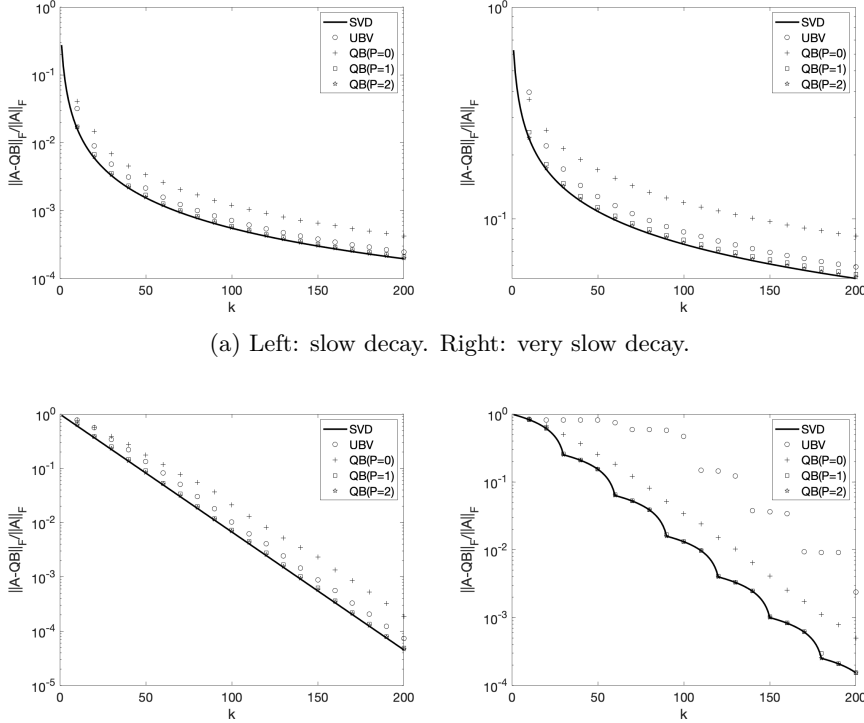
This postprocessing step takes on additional importance when dealing with the block Lanczos method rather than subspace iteration. Where subspace iteration will yield a matrix  $\mathbf{B}$  whose singular values are all decent approximations of the top singular values of  $\mathbf{A}$ , the factor  $\mathbf{B}$  produced by the Lanczos method will contain approximations to the *smallest* singular values of  $\mathbf{A}$  as well [9]. It is therefore possible that the matrix  $\mathbf{B}$  produced by `randUBV` can be truncated significantly without diminishing the quality of the approximation.

In fact, if one has the goal of obtaining a factorization whose rank is as small as possible, we recommend setting the stopping tolerance  $\tau_{\text{stop}}$  slightly smaller than the desired approximation tolerance  $\tau_{\text{err}}$  (or similarly, running the algorithm for a few more iterations after the approximation tolerance has already been satisfied). Doing so will may significantly reduce the rank  $r$  of the truncated SVD, which will in turn pay dividends by reducing the cost of computing  $\mathbf{U}\hat{\mathbf{U}}_r$  and  $\mathbf{V}\hat{\mathbf{V}}_r$ .

**5. Numerical experiments.** Here we report the results of numerical experiments on synthetic and real test cases. We run four sets of experiments in order to examine the following:

1. The rate of convergence by iteration. We use synthetic matrices whose spectra decay at different rates, and compare `randUBV` with `randQB_EI` using power iterations  $p = 0, 1, 2$  for the latter.
2. The effect of sparsity and truncation rank on reorthogonalization costs.
3. The effect of block size on the time and number of iterations required for convergence.
4. The effect of choosing a smaller stopping tolerance  $\tau_{\text{stop}} < \tau_{\text{err}}$  on the quality of the approximation.

All experiments were carried out in MATLAB 2020b on a 4-core Intel Core 7 with 32GB RAM.



(a) Left: slow decay. Right: very slow decay.

(b) Left: fast decay. Right: singular values have multiplicity greater than the block size.

Fig. 5.1: Convergence rate by iteration. In all cases but the last, `randUBV` requires fewer iterations for convergence than `randQB_EI` with  $p = 0$  but more than `randQB_EI` with  $p = 1$ .

**5.1. Convergence rate by iteration.** For our first set of test cases we created matrices of size  $2000 \times 2000$  with the form  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ , where  $\mathbf{U}$  and  $\mathbf{V}$  were formed by orthogonalizing standard Gaussian matrices and  $\mathbf{\Sigma}$  was set in the following manner:

- (Matrix 1) Slow decay, in which  $\sigma_j = 1/j^2$  for  $1 \leq j \leq 2000$ .
- (Matrix 2) Very slow decay, in which  $\sigma_j = 1/j$  for  $1 \leq j \leq 2000$ .
- (Matrix 3) Fast decay, in which  $\sigma_j = \exp(-j/20)$  for  $1 \leq j \leq 2000$ .
- (Matrix 4) Step function decay, in which  $\sigma_j = 10^{-0.6(\lceil j/30 \rceil - 1)}$  for  $1 \leq j \leq 2000$ . Each singular value of  $\mathbf{A}$  (except for the smallest) has multiplicity 30.

In all four cases, we ran the sketching algorithms to a maximum rank  $k = 200$  using block size  $b = 10$ . The deflation tolerance was set at  $\delta = 10^{-12} \sqrt{\|\mathbf{A}\|_1 \|\mathbf{A}\|_\infty}$ , but we did not encounter deflation in any of these cases.

Results are shown in Figure 5.1. In the first three test cases, the approximation error for `randUBV` was smaller than that of `randQB_EI` (with power parameter  $p = 0$ ) for every iteration after the first. It lagged somewhat behind `randQB_EI` with  $p = 1$  or  $p = 2$ , both of which were quite close to optimal. In the final case, where the singular values of  $\mathbf{A}$  were chosen to have multiplicity larger than the block size, `randUBV` lagged significantly behind even `randQB_EI` with  $p = 0$ . We note that algorithm `randUBV` did nonetheless converge, which would not have been possible in exact arithmetic.

Finally, we offer a snapshot of the singular values of  $\mathbf{B}_{200}$  after the algorithms have terminated. Results for test cases 1 and 4 are shown in Figure 5.2. We note that the leading singular values returned by `randUBV` are more accurate than those returned by `randQB_EI` with  $p = 0$  and comparable to the cases  $p = 1$  or  $p = 2$ . The smallest singular values for `randUBV` are much smaller than their `randQB` counterparts, which appears to be undesirable but has a bit of a silver lining: it suggests that the rank of  $\mathbf{B}_k$  can be truncated without losing much approximation accuracy.

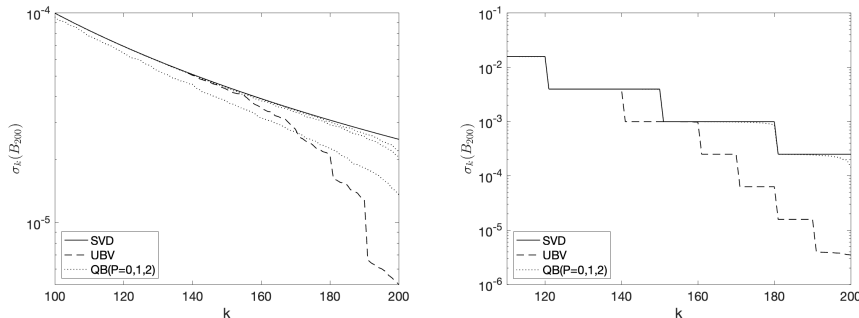


Fig. 5.2: Singular values of  $\mathbf{B}_k$  after termination. Left: slow decay. Right: step function decay.

**5.2. Reorthogonalization costs.** For our second set of test cases we generated random sparse matrices as  $\mathbf{A} = \text{sprand}(m, n, d)$  with  $n = 4000$  columns and varying numbers of rows  $m$  and densities  $d$ . We then approximated  $\mathbf{A}$  to a variable rank  $k$  using `randUBV` and `randQB_EI` with  $p = 0$ . We tested three different variations:

- Number of rows  $m$  varying from 8000 to 40000, rank  $k = 600$ , and  $d = 0.8\%$  nonzeros.
- Number of rows  $m = 24000$ , rank  $k$  varying from 200 to 1000, and  $d = 0.8\%$  nonzeros.
- Number of rows  $m = 24000$ , rank  $k = 600$ , and nonzeros varying from  $d = 0.4\%$  to  $d = 2\%$ .

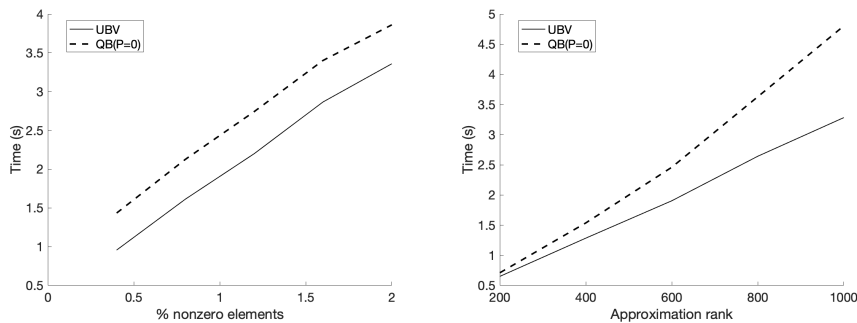


Fig. 5.3: Effects of sparsity (left) and approximation rank (right) on run time.

Results for the second and third cases are shown in Figure 5.3, which confirm our general expectations: for a rectangular matrix with  $m > n$ , if the matrix is sparse or

the approximation rank large then reorthogonalization will take up a larger proportion of the overall cost. Consequently, `randUBV` will gain a competitive advantage over `randQB_EI` due to the fact that it uses one-sided reorthogonalization. This effect will be more pronounced the larger  $m$  is compared to  $n$ , although we found that changing  $m$  alone did not have much effect on the relative runtimes of the two algorithms.

**5.3. Block size.** For our third set of test cases, we examine how the choice of block size affects the time and number of iterations required for convergence. We use one synthetic matrix and two real ones: the synthetic matrix is a  $4000 \times 4000$  matrix whose singular values decrease according to the step function  $\sigma_j = 10^{-0.1(\lceil j/30 \rceil - 1)}$ . Thus each singular value except for the last has multiplicity 30.

The first real matrix is a dense  $3168 \times 4752$  matrix, representing the grayscale image of a spruce pine. The second, `lp_cre_b`, comes from a linear programming problem from the SuiteSparse collection [3], and is a  $9648 \times 77137$  sparse matrix with 260,785 nonzero elements and at most 9 nonzero elements per column. This second matrix has several sizeable clusters of singular values: for example,  $\sigma_{268} \approx 71.10$  and  $\sigma_{383} \approx 70.77$ . The median relative gap  $(\sigma_k - \sigma_{k+1})/\sigma_{k+1}$  among the first 800 singular values is about  $8.6 \times 10^{-5}$ , and the smallest relative gap is about  $2.3 \times 10^{-8}$ . Prior to running the sketching algorithms, both matrices were transposed in order to have more rows than columns.

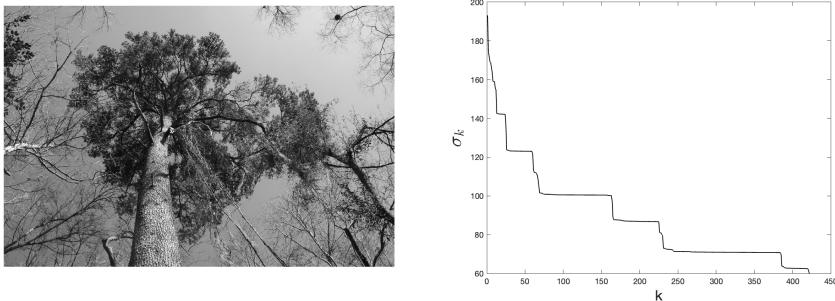


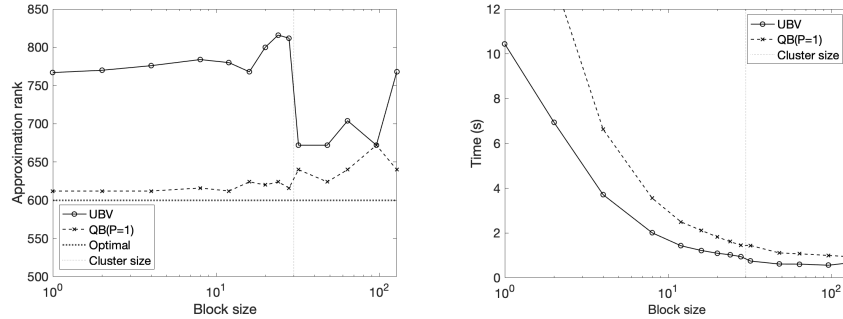
Fig. 5.4: Left: image of *pinus glabra*. Right: leading singular values of `lp_cre_b`.

We compare `randUBV` to `randQB_EI` with power parameter  $p = 1$ . For both algorithms we approximate the synthetic matrix to a relative error  $\tau_{\text{err}} = 0.01$ , the grayscale image to a relative error  $\tau_{\text{err}} = 0.1$ , and the SuiteSparse matrix to a relative error  $\tau_{\text{err}} = 0.5$ .

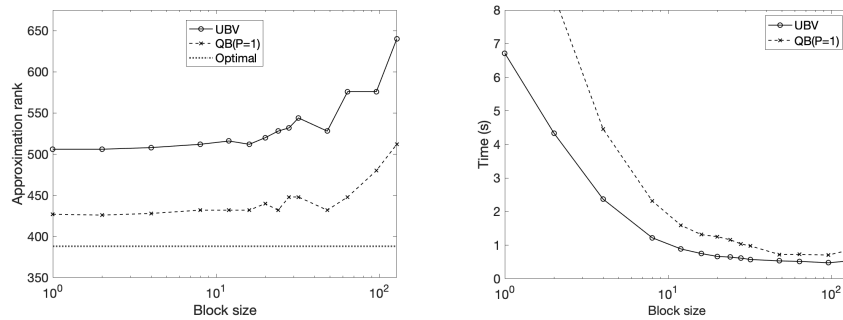
Results are shown in Figure 5.5. The behavior of `randQB_EI` was fairly straightforward: using larger block sizes was more efficient, at least up to the point where the block size was large enough to waste computation by computing  $\mathbf{Q}$  and  $\mathbf{B}$  to a larger rank than necessary. This makes sense because larger block sizes offer more opportunities for using BLAS 3 operations and parallelization. Relatedly, we note that MATLAB’s `svdsketch` function adaptively increases the block size in order to accelerate convergence.

The behavior of `randUBV` was very similar to that of `randQB_EI` on the grayscale image, but less so on the other two cases. For the synthetic matrix whose singular values were distributed according to a step function, increasing  $b$  from just below the cluster size to just above it led to a sharp drop in both the time and number of

iterations required. On the matrix `lp_cre_b`, the optimal block size was near  $b = 10$  even though the approximation rank was close to constant over all block sizes tested. We speculate that the reason for this is that `lp_cre_b` is both sparse and rectangular, so dense QR operations are a significant portion of the cost of the algorithm. Looking back to the cost of `randUBV` as shown in (4.1), we note that using a smaller block size reduces the cost of performing QR operations on  $U$ .



(a) Step function decay.



(b) Grayscale image.

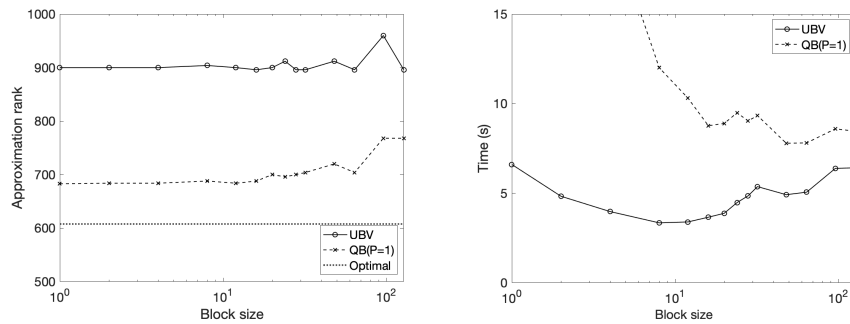
(c) SuiteSparse matrix `lp_cre_b`.

Fig. 5.5: Effect of block size the time and number of iterations required for convergence.



**5.4. Stopping tolerance.** In our final set of experiments we examined the effect of choosing a stopping tolerance  $\tau_{\text{stop}}$  smaller than the desired approximation error tolerance  $\tau_{\text{err}}$ , with the conjecture that doing so would allow **randUBV** to attain significantly better compression rates. We used **randQB\_EI** with  $p = 0, 1, 2$  as a reference for comparison.

The procedure went as follows: in the first step, each sketching algorithm was run until the Frobenius norm approximation error dropped below a set tolerance  $\tau_{\text{stop}}$ . In the second step, the SVD of  $\mathbf{B}$  was then computed and truncated as  $\mathbf{B}_r = \mathbf{U}_r \Sigma_r \mathbf{V}_r^T$  to the smallest rank such that  $\|\mathbf{A} - \mathbf{B}_r\|_F \leq \tau_{\text{err}} \|\mathbf{A}\|_F$ , and the singular vectors of  $\mathbf{A}$  computed as  $\mathbf{U}\mathbf{U}_r$  and  $\mathbf{V}\mathbf{V}_r$  (or as  $\mathbf{Q}\mathbf{U}_r$  for **randQB\_EI**). The time required for each of these two stages was recorded using **tic** and **toc**.

Method	$\tau_{\text{stop}}$	$t_{\text{fac}}$	$t_{\text{svd}}$	$t_{\text{total}}$	$k$	$r$
SVD	–	–	13.52	13.52	–	388
UBV	0.1	0.68	0.08	0.76	520	439
UBV	0.09	0.87	0.11	0.98	600	392
QB(P=0)	0.1	1.22	0.21	1.44	700	663
QB(P=1)	0.1	1.12	0.09	1.22	440	420
QB(P=2)	0.1	1.55	0.08	1.63	420	398

Fig. 5.6: Results for image data with approximation tolerance  $\tau_{\text{err}} = 0.1$ .

**5.4.1. Image data.** For the image data, we ran all algorithms to a relative error of  $\tau_{\text{stop}} = \tau_{\text{err}} = 0.1$  with block size  $b = 20$ , and for **randUBV** additionally considered the stricter stopping tolerance  $\tau_{\text{stop}} = 0.09$ .

Results are shown in Figure 5.6, with all time reported in seconds. There,  $t_{\text{fac}}$  is the time required for the QB or UBV factorization,  $t_{\text{svd}}$  is the time required to compute the SVD of  $\mathbf{B}$  and the new singular vectors of  $\mathbf{A}$ , and  $t_{\text{total}} = t_{\text{fac}} + t_{\text{svd}}$ . Finally,  $k$  is the rank at which the algorithm was terminated, and  $r$  the rank to which  $\mathbf{B}$  was truncated. The first line represents the time required to directly compute the SVD of  $\mathbf{A}$  and the optimal truncation rank.

We observe that **randUBV** ran faster than **randQB\_EI** regardless of the value of the power parameter  $p$ . Even though it required more iterations to converge than **randQB\_EI** with  $p = 1$  or  $p = 2$ , it required fewer matrix-vector products with  $\mathbf{A}$  or  $\mathbf{A}^T$  per iteration. Furthermore, running **randUBV** to a stopping tolerance that was slightly smaller than the truncation tolerance took somewhat longer but resulted in nearly optimal compression, even superior to subspace iteration with  $p = 2$ .

**5.4.2. SuiteSparse data.** For the matrix `1p_cre_b` from the SuiteSparse collection, we ran two trials. In the first, we ran all algorithms to the rather modest relative error of  $\tau_{\text{stop}} = \tau_{\text{err}} = 0.5$ , and for **randUBV** considered the stricter stopping tolerance  $\tau_{\text{stop}} = 0.45$ . In the second, we ran the algorithms to the stricter relative error of  $\tau_{\text{stop}} = \tau_{\text{err}} = 0.15$ , and for **randUBV** additionally considered  $\tau_{\text{stop}} = 0.14$ . We used block size  $b = 50$  for both trials.

Method	$\tau_{\text{stop}}$	$t_{\text{fac}}$	$t_{\text{svd}}$	$t_{\text{total}}$	$k$	$r$
SVD	–	–	–	–	–	608
UBV	0.5	4.69	0.93	5.62	900	747
UBV	0.45	5.68	0.99	6.67	1050	627
QB(P=0)	0.5	8.33	8.32	16.66	1150	1123
QB(P=1)	0.5	5.16	3.69	8.85	700	676
QB(P=2)	0.5	6.54	3.21	9.75	650	627

Fig. 5.7: Results for `lp_cre_b` with approximation tolerance  $\tau_{\text{err}} = 0.5$ .

Method	$\tau_{\text{stop}}$	$t_{\text{fac}}$	$t_{\text{svd}}$	$t_{\text{total}}$	$k$	$r$
SVD	–	–	–	–	–	2082
UBV	0.15	21.28	12.15	33.43	2600	2293
UBV	0.14	24.09	13.88	37.98	2700	2150
QB(P=0)	0.15	72.36	63.25	135.61	3600	3505
QB(P=1)	0.15	38.05	22.91	60.97	2150	2147
QB(P=2)	0.15	48.00	21.59	69.59	2100	2100

Fig. 5.8: Results for `lp_cre_b` with approximation tolerance  $\tau_{\text{err}} = 0.15$ .

Results are shown in Figures 5.7 and 5.8, with all time reported in seconds. Due to the size of the matrix  $\mathbf{A}$ , we did not attempt to compute its SVD directly but instead found the optimal truncation rank using the precomputed singular values available online [3].

Once again, `randUBV` ran faster than its subspace-iteration-based counterpart, and using a slightly smaller stopping tolerance  $\tau_{\text{stop}}$  improved the compression ratio without significantly increasing the runtime. The iteration  $k$  at which `randUBV` terminated was significantly smaller than it was for `randQB_EI` with  $p = 0$ , but significantly larger than for `randQB_EI` with  $p = 1$  or  $p = 2$  (perhaps in part due to the singular value clusters).

It should be noted that the matrix  $\mathbf{A}$  in question is quite sparse with only about 0.03% of its entries nonzero, and fairly skinny with  $m \approx 8n$ . It is therefore worth exploring whether `randQB_EI` might save time on reorthogonalization costs if performed on  $\mathbf{A}^T$  instead. We re-ran the experiment for  $\tau_{\text{err}} = 0.15$ , and found that while the factorization time  $t_{\text{fac}}$  did not change much, the second step  $t_{\text{svd}}$  took around twice as long due to the matrix  $\mathbf{B}$  being  $k \times m$  rather than  $k \times n$ .

**6. Conclusions.** We have proposed a randomized algorithm `randUBV` that takes a matrix  $\mathbf{A}$  and uses block Lanczos bidiagonalization to find an approximation of the form  $\mathbf{UBV}^T$ , where  $\mathbf{U}$  and  $\mathbf{V}$  each have orthonormal columns in exact arithmetic and  $\mathbf{B}$  is a block bidiagonal matrix. For square matrices it costs approximately the same per iteration as `randQB`-type methods run with power parameter  $p = 0$  while having better convergence properties. On rectangular matrices, it exploits one-sided reorthogonalization to run faster without much degrading the accuracy of the error estimator. Numerical experiments suggest that `randUBV` is generally competitive with existing `randUBV`-type methods, at least as long as the problem is not so large that it becomes important to minimize the number of passes over  $\mathbf{A}$ .

A few avenues for future exploration are suggested. First and most importantly, roundoff error allows block Lanczos methods to handle repeated singular values, which they would be unable to do in exact arithmetic. This fact has been known for decades, but we are not currently aware of any rigorous convergence bounds that account for finite precision. Second, reinflation or any more general method for adaptively changing the block size  $b$  will make the span of  $\mathbf{V}$  a sum of Krylov spaces of different dimensions. We are not aware of any convergence results that cover this more general setting.

It is also worth exploring just how much the block Lanczos method benefits from oversampling. We have observed that running `randUBV` for a few more iterations than necessary can result in near-optimal compression, but it would be worthwhile to turn the convergence results of e.g. [32] into practical guidance on how many more iterations are necessary.

Finally, the behavior of  $\mathbf{U}$  when using one-sided reorthogonalization merits further study. We generally found that when using a larger stopping tolerance  $\tau$  the columns of  $\mathbf{U}$  remained closer to orthonormal. It would be highly desirable to obtain a rigorous result establishing that one-sided reorthogonalization is safe as long as only a rough approximation is required, but we leave this goal for a future work.

MATLAB code is available at <https://github.com/erhallma/randUBV>, including our main algorithm `randUBV` as well as code used to reproduce the figures used in this paper.

**Acknowledgments.** The author would like to thank Ilse Ipsen and Arvind Saibaba for their helpful comments on an earlier draft of this paper.

#### REFERENCES

- [1] Z. BAI, D. DAY, AND Q. YE, *ABLE: an adaptive block Lanczos method for non-hermitian eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 1060–1082.
- [2] A. BJÖRCK, *Block bidiagonal decomposition and least square problems*, Perspectives in numerical Analysis, Helsinki, (2008).
- [3] T. A. DAVIS AND Y. HU, *The University of Florida sparse matrix collection*, 38 (2011), <https://doi.org/10.1145/2049662.2049663>.
- [4] P. DRINEAS, I. C. IPSEN, E.-M. KONTOPOULOU, AND M. MAGDON-ISMAIL, *Structural convergence results for approximation of dominant subspaces from block Krylov spaces*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 567–586.
- [5] D. C.-L. FONG AND M. SAUNDERS, *LSMR: An iterative algorithm for sparse least-squares problems*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2950–2971.
- [6] G. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2 (1965), pp. 205–224.
- [7] G. GOLUB, R. UNDERWOOD, AND J. WILKINSON, *The Lanczos algorithm for the symmetric  $Ax = \lambda Bx$  problem*, tech. report, Report STAN-CS-72-270, Department of Computer Science, Stanford U. Stanford . . . , 1972.
- [8] G. H. GOLUB, F. T. LUK, AND M. L. OVERTON, *A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix*, ACM Transactions on Mathematical Software (TOMS), 7 (1981), pp. 149–169.
- [9] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 4th ed., 2013.
- [10] J. F. GRGAR, *Analyses of the Lanczos Algorithm and of the Approximation Problem in Richardson’s Method.*, PhD thesis, University of Illinois at Urbana-Champaign, 1982.
- [11] M. H. GUTKNECHT, *Block Krylov space methods for linear systems with multiple right-hand sides: an introduction*, 2006.
- [12] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53

- (2011), pp. 217–288.
- [13] I. HNĚTYNKOVÁ, M. PLEŠINGER, AND Z. STRAKOŠ, *Band generalization of the Golub–Kahan bidiagonalization, generalized Jacobi matrices, and the core problem*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 417–434.
  - [14] S. KARIMI AND F. TOUTOUNIAN, *The block least squares method for solving nonsymmetric linear systems with multiple right-hand sides*, Applied Mathematics and Computation, 177 (2006), pp. 852–862.
  - [15] L. KAUFMAN, *Band reduction algorithms revisited*, ACM Transactions on Mathematical Software (TOMS), 26 (2000), pp. 551–567.
  - [16] R.-C. LI AND L.-H. ZHANG, *Convergence of the block Lanczos method for eigenvalue clusters*, Numerische Mathematik, 131 (2015), pp. 83–113.
  - [17] P.-G. MARTINSSON AND S. VORONIN, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM Journal on Scientific Computing, 38 (2016), pp. S485–S507.
  - [18] MATLAB, *version 9.9.0 (R2020b)*, The MathWorks Inc., Natick, Massachusetts, 2020.
  - [19] C. MUSCO AND C. MUSCO, *Randomized block Krylov methods for stronger and faster approximate singular value decomposition*, in Advances in Neural Information Processing Systems, 2015, pp. 1396–1404.
  - [20] C. C. PAIGE, *The computation of eigenvalues and eigenvectors of very large sparse matrices.*, PhD thesis, University of London, 1971.
  - [21] B. N. PARLETT AND D. S. SCOTT, *The Lanczos algorithm with selective orthogonalization*, Mathematics of computation, 33 (1979), pp. 217–238.
  - [22] L. REICHEL AND Q. YE, *Breakdown-free GMRES for singular systems*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 1001–1021.
  - [23] Y. SAAD, *On the rates of convergence of the Lanczos and the block-Lanczos methods*, SIAM Journal on Numerical Analysis, 17 (1980), pp. 687–706.
  - [24] H. D. SIMON, *The Lanczos algorithm with partial reorthogonalization*, Mathematics of computation, 42 (1984), pp. 115–142.
  - [25] H. D. SIMON AND H. ZHA, *Low-rank matrix approximation using the Lanczos bidiagonalization process with applications*, SIAM Journal on Scientific Computing, 21 (2000), pp. 2257–2274.
  - [26] Z. STRAKOŠ AND P. TICHÝ, *On error estimation in the conjugate gradient method and why it works in finite precision computations.*, ETNA. Electronic Transactions on Numerical Analysis [electronic only], 13 (2002), pp. 56–80.
  - [27] F. TOUTOUNIAN AND M. MOJARRAB, *The block LSMR method: a novel efficient algorithm for solving non-symmetric linear systems with multiple right-hand sides*, Iranian Journal of Science and Technology (Sciences), 39 (2015), pp. 69–78.
  - [28] S. WANG, Z. ZHANG, AND T. ZHANG, *Improved analyses of the randomized power method and block Lanczos method*, arXiv preprint arXiv:1508.06429, (2015).
  - [29] Q. YE, *A breakdown-free variation of the nonsymmetric Lanczos algorithms*, Mathematics of Computation, 62 (1994), pp. 179–207.
  - [30] Q. YE, *An adaptive block Lanczos algorithm*, Numerical Algorithms, 12 (1996), pp. 97–110.
  - [31] W. YU, Y. GU, AND Y. LI, *Efficient randomized algorithms for the fixed-precision low-rank matrix approximation*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1339–1359.
  - [32] Q. YUAN, M. GU, AND B. LI, *Superlinear convergence of randomized block Lanczos algorithm*, in 2018 IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 1404–1409.
  - [33] Y. ZHOU AND Y. SAAD, *Block Krylov–Schur method for large symmetric eigenvalue problems*, Numerical Algorithms, 47 (2008), pp. 341–359.