Normalized power iterations for the computation of SVD

Per-Gunnar Martinsson
Department of Applied Mathematics
University of Colorado
Boulder, Co.
Per-gunnar.Martinsson@Colorado.edu

Arthur Szlam
Courant Institute of Mathematical Sciences
New York, N.Y.
aszlam@courant.nyu.edu

Mark Tygert
Courant Institute of Mathematical Sciences
New York, N.Y.
tygert@courant.nyu.edu

1 A randomized algorithm for low rank matrix aproximation

We are interested in finding an approximation $U\Sigma V^T$ to the $m \times n$ matrix A, where U is an $m \times k$ orthogonal matrix, Σ is a $k \times k$ diagonal matrix, V is an $n \times k$ orthognal matrix, and k is a user set positive integer less than m; usually $k \ll m$. Recently there have been several papers which have approached this problem by determining the approximate range space of A by applying it to a set of random vectors. The full SVD is then computed on A's approximate range, giving algorithms that run in $O(mkn)$ time; see [5, 3, 6]. The purpose of this short note is to discuss a particular variant of these algorithms which is a good choice when a high quality (as measured by operator norm) low rank approximation of a matrix is desired, and memory is a limiting quantity.

The algorithm takes as input nonnegative integers q and l such that $m > l > k$ (usually l is slightly bigger than k, say $k + 10$), and then goes as follows:

1. Form an $m \times l$ Gaussian random matrix P_0.
2. Using a QR decomposition, form an $n \times l$ matrix Q_0 such that
$$Q_0 R_0 = (P_0^T A)^T,$$
the columns of Q_0 are orthonormal, and R_0 is upper triangular.
3. Iterate for $j = 0$ to $j = q - 1$
$$P_{j+1} R_{2j+1} = A Q_j \tag{1}$$
$$Q_{j+1} R_{2j+2} = (P_{j+1}^T A)^T, \tag{2}$$
where the left side of each equation is the QR decomposition of the right side.
4. Form the full SVD $\tilde{U}\tilde{\Sigma}\tilde{V}^T = AQ_q$
5. Set U to be the first k columns of $\tilde{U}$, Σ to be the $k \times k$ upper left corner of $\tilde{\Sigma}$, and V to be the first k columns of $Q_q\tilde{V}$.

This is essentially the normalized power iterations described in [2, Sec 7.3.2]. In exact arithmetic, if A has rank greater than l, the subspace found by this method is the same as the subspace found

by the unnormalized power iterations in [3, 6]. This can be seen expanding out Q_j using equations 1 and 2: $Q_0 = A^T P_0 R_0^{-1}$, then

$$P_1 = AA^T P_0 R_0^{-1} R_1^{-1},$$
$$Q_1 = (A^T A) A^T P_0 R_0^{-1} R_1^{-1} R_2^{-1},$$
$$P_2 = (AA^T)^2 P_0 R_0^{-1} R_1^{-1} R_2^{-1} R_3^{-1},$$
$$Q_2 = (A^T A)^2 A^T P_0 R_0^{-1} R_1^{-1} R_2^{-1} R_3^{-1} R_4^{-1},$$

and in general,

$$Q_j = (A^T A)^j A^T P_0 \left(\prod_{s=0}^{2j} R_s^{-1} \right).$$

These are also the same subspaces constructed by [7], which interprets solving

$$B_{j+1} = \text{argmin}_{B \in \mathbb{R}^{m \times l}} ||BC_j - A||_{\text{F}}^2 \tag{3}$$
$$C_{j+1} = \text{argmin}_{C \in \mathbb{R}^{l \times n}} ||B_{j+1} C - A||_{\text{F}}^2$$

as an instance of the EM algortithm; in that work each of these subproblems is solved via an explicit pseudoinverse.

All of these methods (in the case of exact arithmetic) thus satisfy the bounds described in [3, 6]. For example, we have[1]

$$\mathbb{E}||A - AQ_q Q_q^T|| \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{e\sqrt{k+p}}{p} \cdot \sqrt{m-k} \right]^{1/(2q+1)} \sigma_{k+1}, \tag{4}$$

where the matrix norms are taken in the sense of the operator norm, σ_{k+1} is the $k+1$th singular value of A (and thus the optimal possible error for a rank k approximation), and we denote $p = l - k$. However, these methods are not the same with respect to roundoff error. In particular, the unnormalized power iterations can produce poor results as q increases. In [6] this problem is resolved by storing all of the intermediate powers of A against B_0. However, this can be an issue if $m \times l$ is large with respect to your memory budget. In the next section, we will see experiments suggesting that the normalized power iterations are numerically more stable than the unnormalized power iterations, but with nearly the same memory requirements, at the cost of an increase in computation time.

2 Experiments

All experiments were run in matlab 7.7 for linux. The machine used has two 3Ghz core duo processors and 8 Gb of ram.

2.1 Random dense matrices

For each $m \in \{512, 1024, 2048, 4096\}$, we fix a number $p \in \{10^{-2}, 10^{-4}, ..., 10^{-14}\}$, and generate matrices $A = A_{m,p}$ via the formula

$$A = U \Sigma V^T,$$

where U is the left singular vectors of an $m \times m$ Gaussian matrix, V is the left singular vectors of an $2m \times 2m$ Gaussian matrix, and Σ is $m \times 2m$ matrix with $\Sigma(s,t) = 0$ if $s \neq t$, and

$$\Sigma_{s,s} = p^{\lfloor s/2 \rfloor / 5}$$

for $s \leq 10$, and

$$\Sigma_{s,s} = p \cdot \frac{m-j}{m-11}$$

for $11 \leq s \leq m$.

[1] The reader may notice that Q_q is rank l rather than rank k. The authors of [3] have communicated to us that the same bound holds with the QQ^T in the left of the inequality replaced by $U\Sigma V^T$; but this has not yet been published. Also note that the analogous bound to 4 in [6] is in terms of $U\Sigma V^T$, although there the bound is taken with high probability, and there are constants that depend on the "high".

(a) Normalized power iterations

	m=512	m=1024	m=2048	m=4096
p=1.0e-2	0.011	0.014	0.016	0.018
p=1.0e-4	1e.0-4	1.0e-4	1.0e-4	01.03e-4
p=1.0e-6	1e.0-06	1.0e-06	1.0e-06	1.0e-06
p=1.0e-8	1e.0-08	1.0e-08	1.0e-08	1.0e-08
p=1.0e-10	1e.0-10	1.0e-10	1.0e-10	1.0e-10
p=1.0e-12	1e.0-12	1.0e-12	1.0e-12	1.0e-12
p=1.0e-14	1.01e-14	1.0e-14	1.01e-14	1.0e-14

(b) Unnormalized power iterations

	m=512	m=1024	m=2048	m=4096
p=1e-2	0.011	0.014	0.016	0.018
p=1e-4	1.0e-4	1.0e-4	1.0e-4	1.0e-4
p=1e-6	1.18e-06	1.01e-06	1 e-06	1.05e-06
p=1e-8	4.55e-07	4.30e-07	4.0e-07	3.98e-07
p=1e-10	9.98e-07	9.98e-07	9.99e-07	9.99e-07
p=1e-12	2.56e-07	2.29e-07	1.69e-07	1.76e-07
p=1e-14	2.36e-06	2.33e-06	2.28e-06	2.22e-06

Table 1: Error on random dense matrices as in section 2.1, processed using unnormalized power iterations. Here $q = 1$, $p = 4$, and $k = 10$.

	m=512	m=1024	m=2048	m=4096
normalized	.02	.07	.27	1.08
unnormalized	.02	.07	.27	1.07

Table 2: Timings in seconds for the computation of the approximate svds of dense random matrices. The top row is using normalized power iterations, and the bottom row is unnormalized power iterations; here $q = 2$.

(a) Normalized power iterations

	p=1e-2	p=1e-4	p=1e-6	p=1e-8	p=1e-10	p=1e-12	p=1e-14
q=1	0.025	2.0 e-4	1.0 e-6	1.0 e-8	1.0 e-10	1.0 e-12	4.3 e-14
q=2	0.014	1.0 e-04	1.0 e-6	1.0 e-8	1.0 e-10	1.0 e-12	1.9 e-13
q=3	0.01	1.0 e-04	1.0 e-6	1.0 e-8	1.0 e-10	1.0 e-12	2.0 e-13
q=4	0.01	1.0 e-04	1.0 e-6	1.0 e-8	1.0 e-10	1.0 e-12	1.8 e-13
q=5	0.01	1.0 e-04	1.0 e-6	1.0 e-8	1.0 e-10	1.0 e-12	1.7 e-13

(b) Unnormalized power iterations

	p=1e-2	p=1e-4	p=1e-6	p=1e-8	p=1e-10	p=1e-12	p=1e-14
q=1	0.025	2.0 e-4	1.0 e-6	4.0 e-7	1.0 e-6	1.1 e-7	1.9 e-6
q=2	0.014	2.0 e-4	2.5 e-4	1.5 e-4	1.0 e-4	1.6 e-5	3.8 e-6
q=3	0.01	3.5 e-3	3.2 e-3	6.3 e-4	1.0 e-4	3.0 e-3	1.5 e-3
q=4	0.01	3.9 e-3	4.0 e-3	6.3 e-4	0.01	4.0 e-3	1.5 e-3
q=5	0.025	0.025	4.0 e-3	0.024	0.01	4.0 e-3	1.6 e-3

Table 3: Error on random DCT matrices constructed as in section 2.2. Here $m = 512^2$, $n = 2 \cdot 512^2$, $p = 4$, and $k = 10$. We use the power method from [1, 4] with 400 iterations to estimate the norm of the error.

2.2 Random DCT matrices

We fix $m = 512^2$ and pick a number $p \in \{10^{-2}, 10^{-4}, ..., 10^{-14}\}$, and generate matrices $A = A_{m,p}$ via the formula

$$A = U\Sigma P V^T,$$

where V^T computes a discrete cosine transform of length $2m$, P is a random permutation matrix, U computes an inverse discrete cosine transform, and Σ is $m \times 2m$ matrix with $\Sigma(s,t) = 0$ if $s \neq t$, and

$$\Sigma_{s,s} = p^{\lfloor s/2 \rfloor/5}$$

for $s \leq 10$, and

$$\Sigma_{s,s} = p \cdot \frac{m-j}{m-11}$$

for $11 \leq s \leq m$.

2.3 20 Newsgroups

We work on the 20-newsgroups dataset (available from `http://people.csail.mit.edu/jrennie/20Newsgroups/`), consisting of 18477 documents from one of 20 newsgroups represented by its tf-idf normalized term-document matrix A, with a vocabulary of the 60000 most common words. That is: we form the matrix A_0 whos (s,t) entry records the number of times

(a) Normalized power iterations

	k=10	k=20	k=30	k=40
$m = 2^{16}$	5.3	9.6	15.4	21.5
$m = 2^{17}$	13.4	23.9	42.9	55.8
$m = 2^{18}$	34.5	67.9	101.8	134.1
$m = 2^{19}$	77.5	148.6	230.1	268.1

(b) Unnormalized power iterations

	k=10	k=20	k=30	k=40
$m = 2^{16}$	4.2	7.3	11.0	14.6
$m = 2^{17}$	11.0	18.6	35.7	43.9
$m = 2^{18}$	30.3	55.5	83.1	105.8
$m = 2^{19}$	68.6	121.8	178.7	220.7

Table 4: Timings in seconds for the computation of the approximate svds of random DCT matrices using normalized and unnormalized power iterations; here q=2.

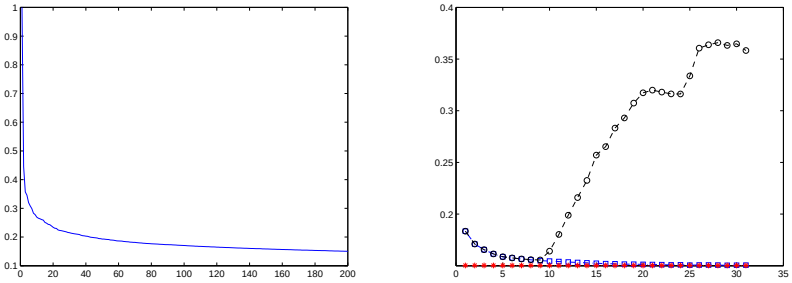


Figure 1: On the left: The first 200 singular values of the 20 newsgroups data set. On the right: operator norm error on the 20 newsgroups dataset. Here $k = 200$, $p = 10$. The baseline is the error from the matlab function svds; the blue squares are the error of the normalized power iterations, and the errors for the unnormalized power iterations are shown with black circles . The x axis is iterations q, and the y axis is error. We use the power method from [1, 4] with 400 iterations to estimate the norm of the error.

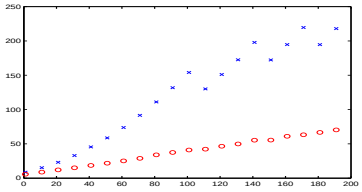


Figure 2: Time taken to compute 20 iterations of normalized (blue) and unnormalized (red) power iterations on the 20 newsgroups with various values of k. The x axis is k and the y axis is time in seconds. The sawtooth in the larger k is completely dependent on matlab's qr from its precompiled LAPACK library.

document s contains word t. Each row of A_0 is divided by its maximum value to get the matrix A_1. Each column t of A_1 is multiplied by $\log(60000/\text{freq}_t)$ to obtain A_2, where freq_t is the number of times the word t appears in the corpus. Each row of A_2 is l_2 normalized to get A_3. Finally, to run the comparison against the unnormalized power iterations, we divide A_3 by its largest singular value (as computed by the matlab function svds) to obtain the matrix A. The singular values of A start from 1 and decay quite slowly after the first few- see Figure 2. Nevertheless, as can be seen in figure 1 the error rapidly decays to the optimum. On the other hand, after several iterations, the error of the unnormalized iterations starts to increase.

References

[1] Vladimir Rokhlin Franco Woolfe, Edo Liberty and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Appl. Comp. Harm. Anal.*, 25(3):335–366, 2008.

[2] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd edition, October 1996.

[3] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Sep 2009.

[4] Jacek Kuczynski and Henryk Wozniakowski. Estimating the largest eigenvalues by the power and lanczos methods with a random start. *SIAM J. Math. Anal.*, 4(13):1094–1122, 1992.

[5] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the approximation of matrices. Technical report, Yale CS research report YALEU/DCS/RR-1361, 2006.

[6] V. Rokhlin, A. Szlam, and M. Tygert. A randomized algorithm for principal component analysis. *SIAM. J. Matrix Anal. & Appl.*, 31:1100–1124, 2009.

[7] Sam T. Roweis. EM algorithms for PCA and SPCA. In *NIPS*, 1997.

A Additional experiments

A.1 Faces

We use images from the labeled faces in the wild dataset available at `http://vis-www.cs.umass.edu/lfw/`. We use the resize each of the centered images to 64 by 64, obtaining a data set A_1 of size 4096×13233. We normalize A_1 by its first singular value, calculated by the matlab function svds, to get A. The singular values of A are shown in figure 3. As can be seen in figure 3 the operator norm error of the normalized power iterations rapidly tends to the optimal error; the unnormalized power iterations initially give the same results but become unstable. Here we again set $p = 10$. In figure 5 we show the first 50 left singular vectors from A computed by 20 normalized power iterations.

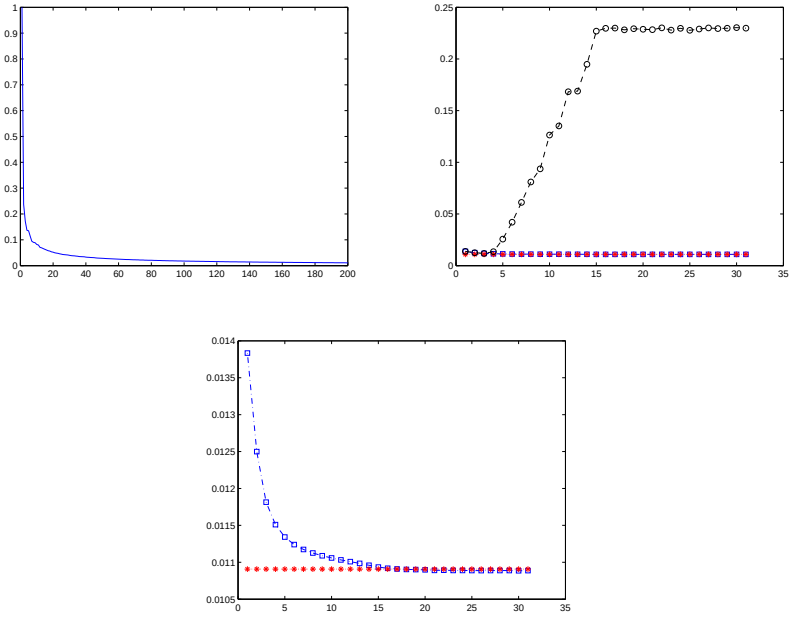


Figure 3: On the left: The first 200 singular values of the faces data set. On the right: operator norm error on the faces dataset. Here $k = 200$, $p = 10$. The baseline is the error from the matlab function svds; the blue squares are the error of the normalized power iterations, and the errors for the unnormalized power iterations are shown with black circles . The x axis is iterations q, and the y axis is error. On the bottom: we display the same graph as the right but with the unnormalized power iterations removed and the y axis rescaled. We use the power method from [1, 4] with 400 iterations to estimate the norm of the error.

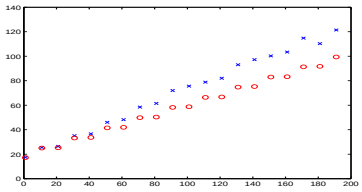


Figure 4: Time taken to compute 20 iterations of normalized (blue) and unnormalized (red) power iterations on the faces data set with various values of k. The x axis is k and the y axis is time in seconds.

Figure 5: The first 50 left singular vectors of the faces data set.

A.2 Some comparisons against other types of power iterations

We compare five techniques for constructing a basis matrix Q for the approximate range of A. In all that follows, Ω is an $n \times \ell$ Gaussian random matrix.

a. The unnormalized power method
b. The normalized power method.
c. The power method with the "extended sample matrix":
$$[Q, \cdot] = \mathtt{qr}([(A\,A^*)^q\,A\,\Omega,\ \ldots,\ (A\,A^*)\,A\,\Omega,\ A\,\Omega],\ 0).$$
Note that here Q has $q\,\ell$ columns.
d. Q consists of the ℓ leading left singular vectors of the "extended sample matrix":
$$[U, D, V] = \mathtt{svd}([(A\,A^*)^q\,A\,\Omega,\ \ldots,\ (A\,A^*)\,A\,\Omega,\ A\,\Omega]), \qquad Q = U(:, 1\!:\!\ell).$$
e. Normalized power method with an "extended sample matrix":
$$[Q, \cdot] = \mathtt{qr}([A\,\Omega,\ A\,Q_1,\ A\,Q_2,\ \ldots,\ A\,Q_{q-2},\ A\,Q_{q-1}],\ 0),$$
where $\{Q_j\}_{j=1}^{q-1}$ are defined as in method (b).

Examples:

To compare the errors, we considered four different test matrices. Each matrix is $n \times n$, with $n = 100$ or $n = 1000$.

Example 1 $A = U\,\Sigma\,U$ where U is a random $n \times n$ unitary matrix, and Σ is a prescribed diagonal matrix. The first ten singular values decay linearly down to a fixed level of 10^{-10}; all the remaining singular values are 10^{-10}.

Example 2 $A = U\,\Sigma\,U$ where U is a random $m \times m$ unitary matrix, and Σ is a prescribed diagonal matrix. The singular values are specified by $\Sigma(k,k) = 10^{-2(1-(k)^{-1.1})}$

Example 3 A is a matrix from potential theory (sort of). Set
$$x = [0, 1 - 1/2, 1 - 1/3, ..., 1 - 1/1000],$$
y to be the vector of length 1000 with every entry equal to 1. Then set
$$A = \log(|x^T y - y^T (y + x)|).$$

Example 4 $A = B\,D\,B^*$ where B is an $n \times n$ Gaussian matrix, and D is a diagonal matrix whose diagonal values are specified by $D(k,k) = k^{-3}$.

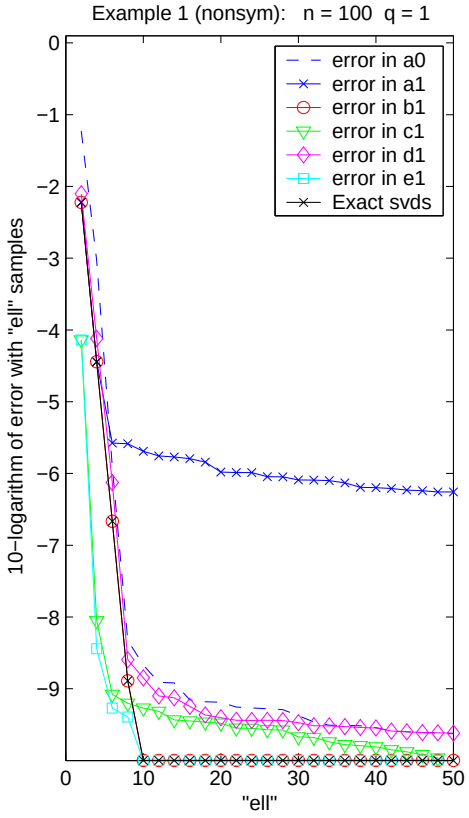
All matrices are normalized so that $||A|| = \sigma_1 = 1$.

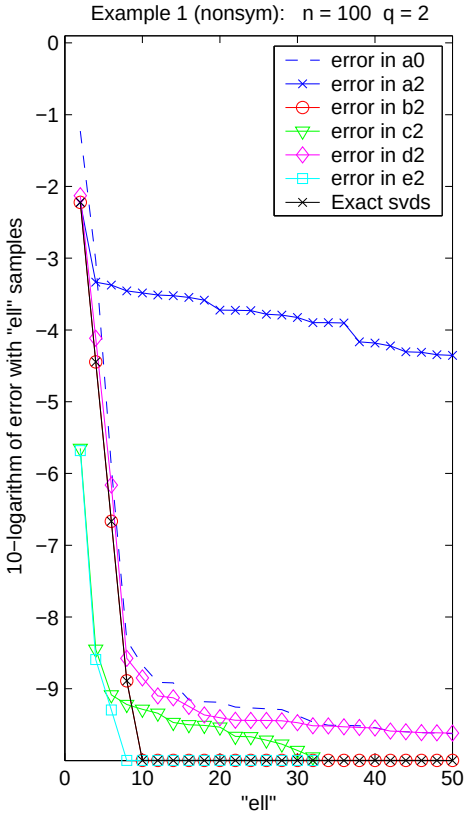
Numerical results:

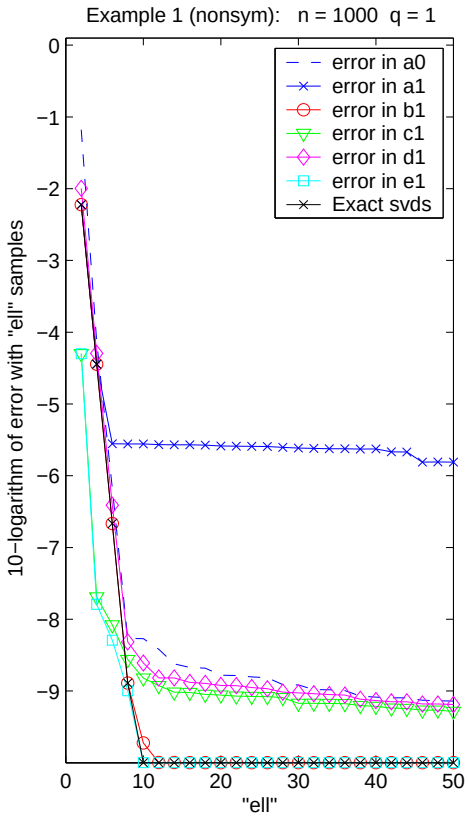
The following graphs plot $\log_{10}(e)$ versus the number of columns ℓ in the random matrix, where where

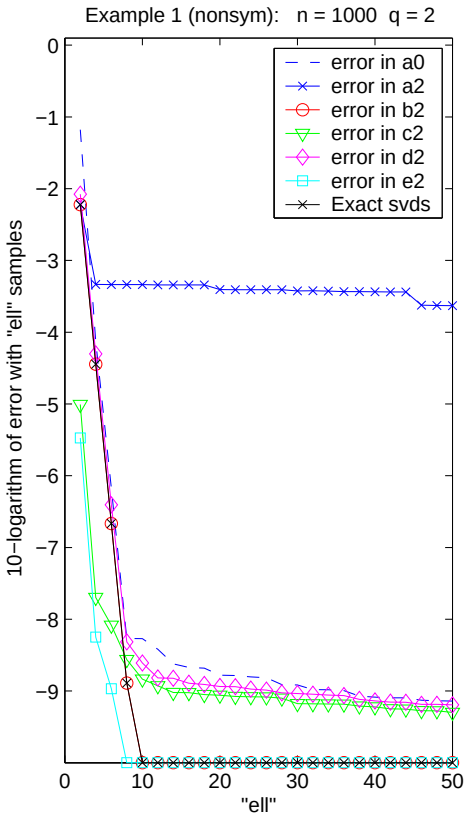
$$e = e_\ell = ||A - Q\,Q^*\,A||.$$

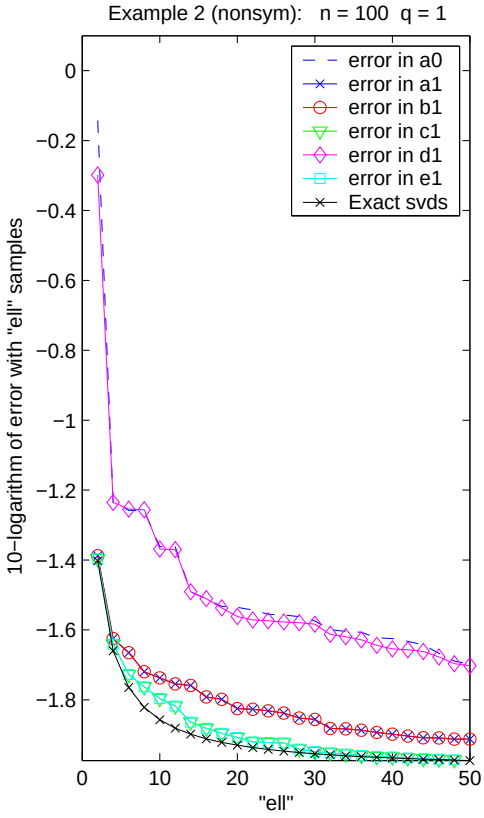
The exact singular values are plotted with a solid black line and crosses. Note that $\sigma_{\ell+1}$ represents a lower bound for the approximation error achievable using ℓ basis vectors. (Method C often outperforms this bound, which is fine since Method C uses $q\,\ell$ basis vectors.) The error for the "basic" scheme with $q = 0$ is plotted with a dashed line. (Notice that the methods A/B/C/D/E are identical when $q = 0$.) In the legend for each graph, the methods are abbreviated. For instance "c2" means "Method C" with $q = 2$.

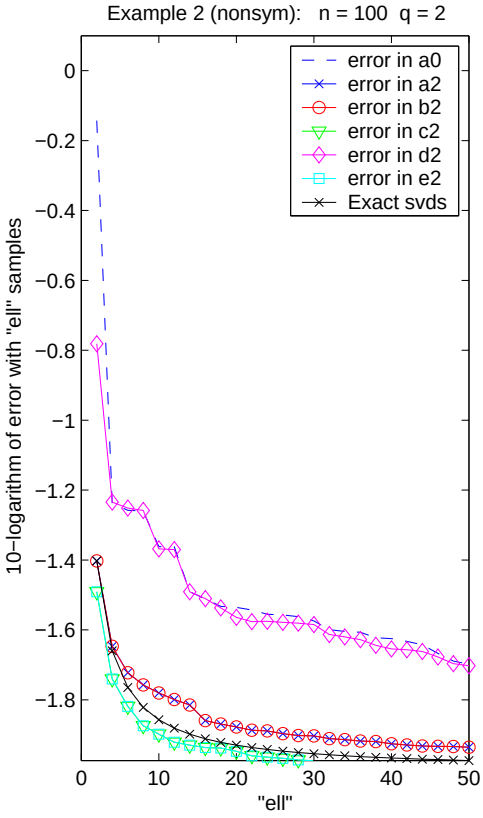

Example 1 (nonsym): n = 100 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"

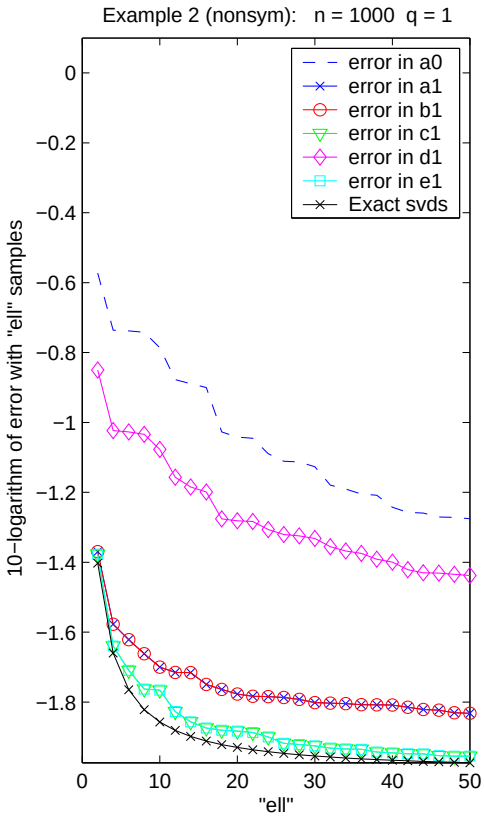

Example 1 (nonsym): n = 100 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"

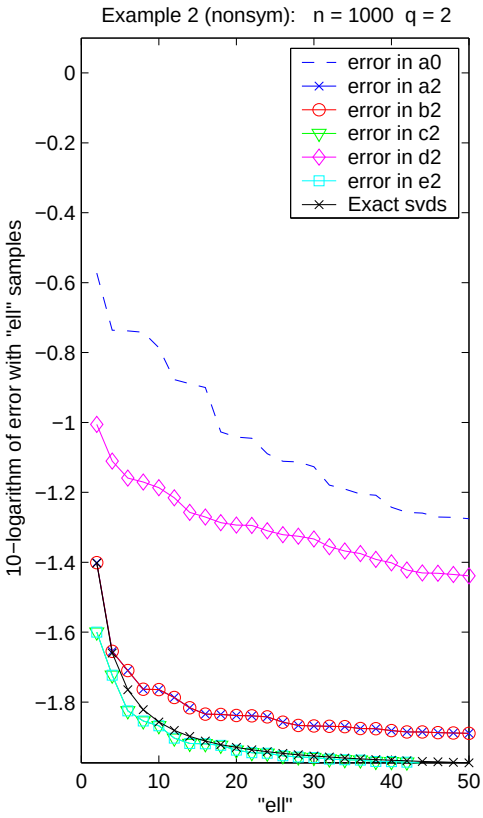

Example 1 (nonsym): n = 1000 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"

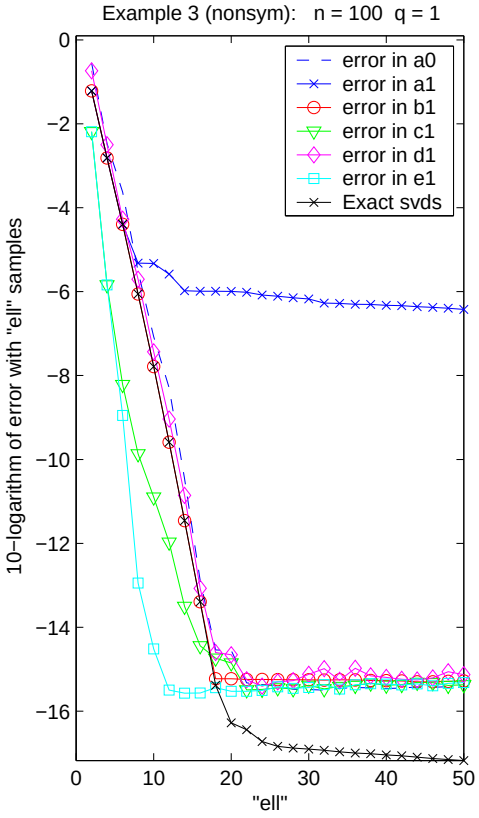

Example 1 (nonsym): n = 1000 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"

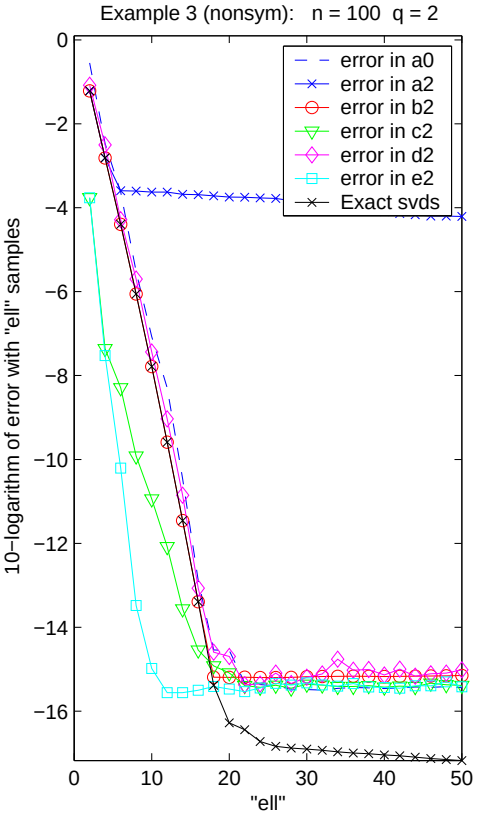

Example 2 (nonsym): n = 100 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"

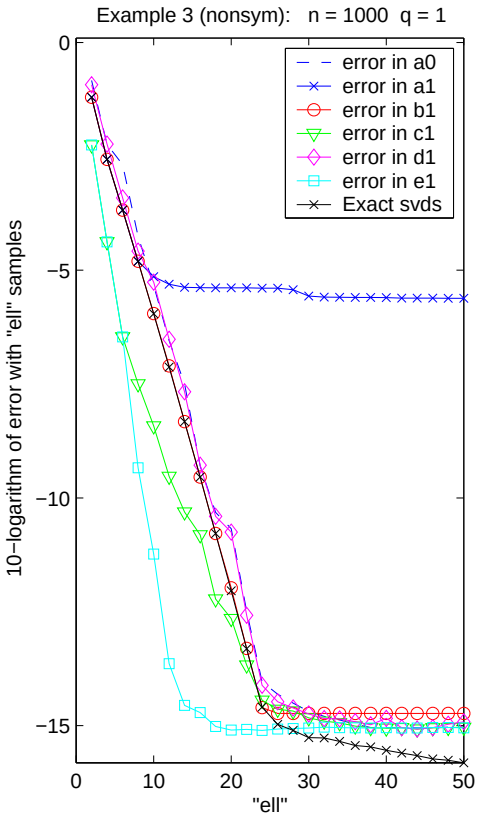

Example 2 (nonsym): n = 100 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"

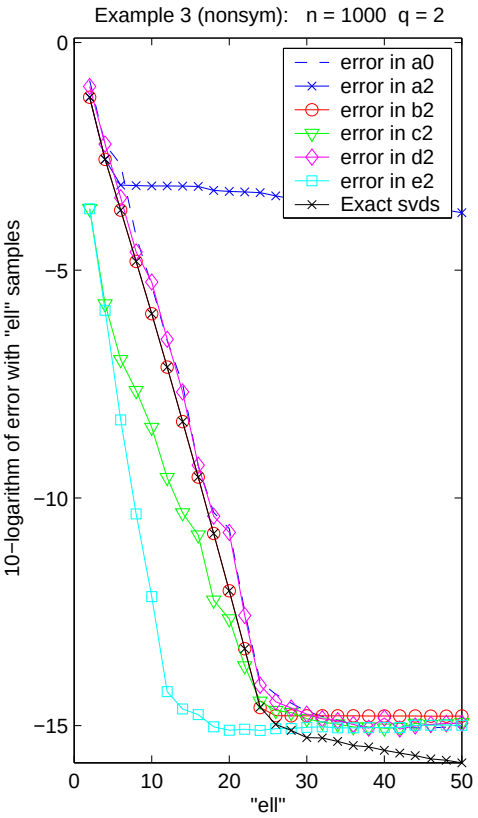

Example 2 (nonsym): n = 1000 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"

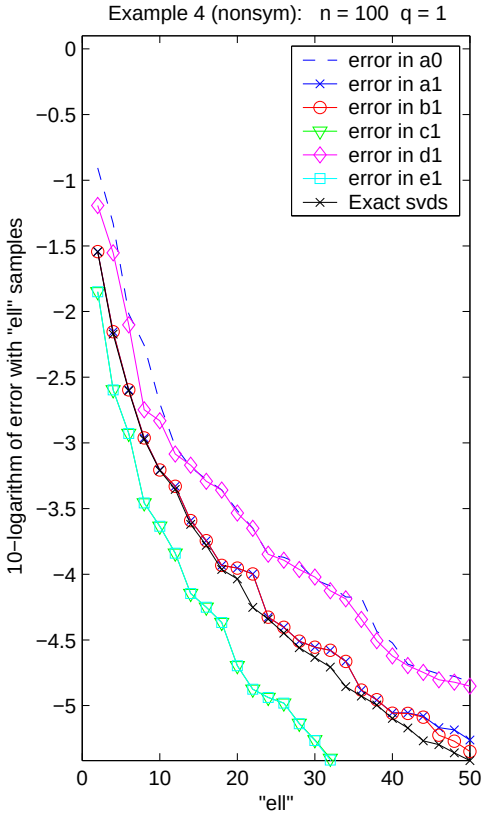

Example 2 (nonsym): n = 1000 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"

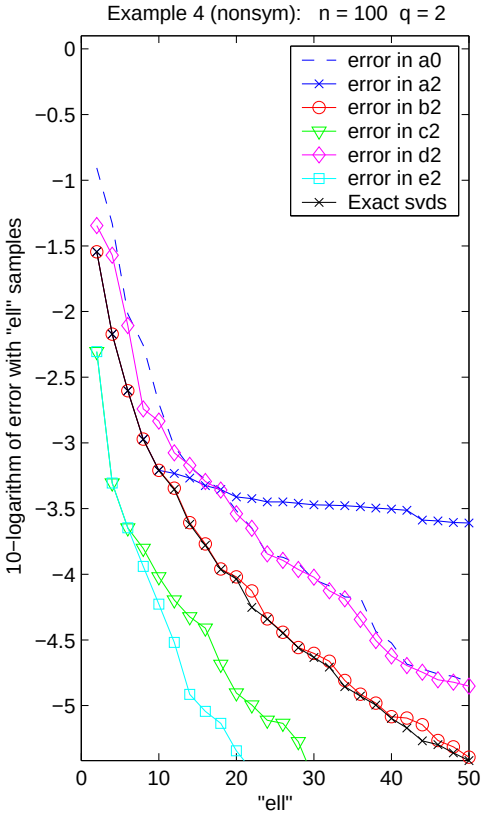

Example 3 (nonsym): n = 100 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"

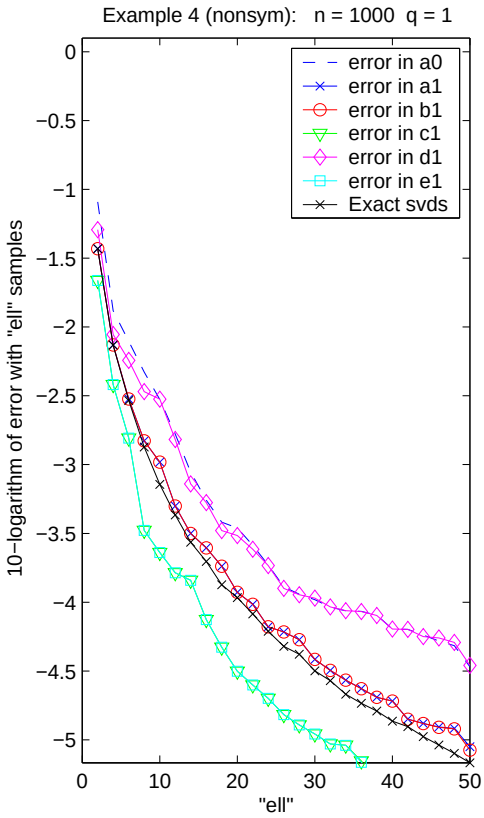

Example 3 (nonsym): n = 100 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"

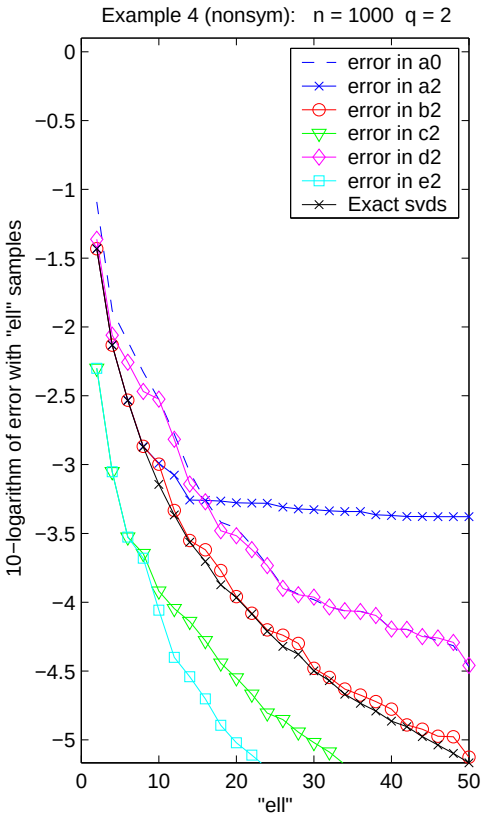

Example 3 (nonsym): n = 1000 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"


Example 3 (nonsym): n = 1000 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"


Example 4 (nonsym): n = 100 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"


Example 4 (nonsym): n = 100 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"


Example 4 (nonsym): n = 1000 q = 1
error in a0
error in a1
error in b1
error in c1
error in d1
error in e1
Exact svds
10−logarithm of error with "ell" samples
"ell"


Example 4 (nonsym): n = 1000 q = 2
error in a0
error in a2
error in b2
error in c2
error in d2
error in e2
Exact svds
10−logarithm of error with "ell" samples
"ell"