

kappa_SQ: A Matlab package for randomized sampling of matrices with orthonormal columns

Thomas Wentworth, North Carolina State University
Ilse Ipsen, North Carolina State University

The kappa_SQ software package is designed to assist researchers working on randomized row sampling. The package contains a collection of Matlab functions along with a GUI that ties them all together and provides a platform for the user to perform experiments.

In particular, kappa_SQ is designed to do experiments related to the two-norm condition number of a sampled matrix, $\kappa(SQ)$, where S is a row sampling matrix and Q is a tall and skinny matrix with orthonormal columns. Via a simple GUI, kappa_SQ can generate test matrices, perform various types of row sampling, measure $\kappa(SQ)$, calculate bounds and produce high quality plots of the results. All of the important codes are written in separate Matlab function files in a standard format which makes it easy for a user to either use the codes by themselves or incorporate their own codes into the kappa_SQ package.

Categories and Subject Descriptors: G.4 [Mathematical Software]: Algorithms, Experimentation, Measurement, Standardization, Verification

General Terms: Algorithms, Experimentation, Measurement, Theory, Verification

Additional Key Words and Phrases: Randomized Algorithms, Sampling, Blendenpik, Package, Orthonormal, Leverage Scores, Coherence, Plot

ACM Reference Format:

Thomas Wentworth, and Ilse Ipsen, 2013. kappa_SQ: A Matlab package for randomized sampling of matrices with orthonormal columns. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 14 pages.
DOI: <http://dx.doi.org/10.1145/0000000.0000000>

1. INTRODUCTION

We wrote the kappa_SQ software package to assist us with our research on various algorithms for uniform row sampling [Ipsen and Wentworth 2013]. In our research, a $m \times n$ matrix Q with orthonormal columns and $m \geq n$ is sampled by a row sampling matrix S to create the $c \times n$ sampled matrix SQ . We then address the question, given $\eta > 0$, what is the probability that $\text{rank}(SQ) = n$ and the two-norm condition number $\kappa(SQ) = \|SQ\|_2 \|(SQ)^\dagger\|_2 \leq 1 + \eta$? This question is important due to its applications to randomized least squares solvers such as LSRN [Meng et al. 2011] and, in particular, the *Blendenpik* algorithm [Avron et al. 2010].

The *Blendenpik* algorithm uses randomized sampling to solve an overdetermined least-squares problem $\min_x \|Ax - b\|_2$ faster than LAPACK. It starts by finding the QR factorization, $Q_s R_s = SA$, of the randomly sampled matrix SA and then, if SA has

This work is supported by NSF CISE CCF NCSU and GRANT NUMBERS.

Authors' address: Mathematics Department, 2311 Stinson Drive, Box 8205, NC State University, Raleigh, NC 27695-8205

Authors' e-mails: Thomas Wentworth: thomas_wentworth@ncsu.edu, Ilse Ipsen: ipsen@ncsu.edu

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2010 ACM 1539-9087/2010/03-ART39 \$15.00

DOI: <http://dx.doi.org/10.1145/0000000.0000000>

full column rank, solves the preconditioned least squares problem $\min_z \|AR_s^{-1}z - b\|_2$ via LSQR. The solution to the original least squares problem can be found by solving a much smaller linear system with coefficient matrix R_s . The key to this method is that if $\kappa(AR_s^{-1}) \approx 1$, then LSQR will converge quickly.

The connection between our work, kappa.SQ and the *Blendenpik* algorithm is that if SA is full rank, then $\kappa(SQ) = \kappa(AR_s^{-1})$. This means that sampling rows from A is, conceptually, the same as sampling rows from Q and that $\kappa(AR_s^{-1})$ depends only on the column space of A (and the sampling matrix). Thus, it suffices to examine the behavior of $\kappa(SQ)$.

This code examines $\kappa(SQ)$ in two main ways. First, it can perform numerical experiments where $\kappa(SQ)$ is measured. And second, our code can plot bounds for $\kappa(SQ)$. In the literature, these bounds are often expressed in terms of two matrix properties that have been shown to be related to row sampling, leverage scores, and coherence.

Leverage scores were first introduced in 1978 by Hoaglin and Welsch [Hoaglin and Welsch 1978] to detect outliers when computing regression diagnostics. They give a measurement of the distribution of the elements in an orthonormal basis. The leverage scores of a matrix A are defined in terms of any orthonormal basis, Q , for the column space of A .

Definition 1.1. The leverage scores of the real $m \times n$ matrix A with $m \geq n$ are

$$\ell_j(A) = \ell_j(Q) \equiv \|e_j^T Q\|_2^2, \quad 1 \leq j \leq m.$$

Since leverage scores are simply row norms from matrices with orthonormal columns, the inequality $0 \leq \ell_i(A) \leq 1$ holds and $\sum_{i=1}^m \ell_i(A) = n$. If $\ell_i(A) = 1$ then the i 'th row contains all of the information for a particular column. On the other hand, if $\ell_i(A) = 0$, then the i 'th row of A is zero and contains no data. Thus, leverage scores give a quantification of the importance of each row with respect to sampling. We use leverage scores as an input to both generate test matrices and to bound the condition number of a sampled matrix. Our code for computing leverage scores is `leverageScores.m`.

In our work, coherence is simply the largest leverage score.

Definition 1.2 (*Definition 3.1 in [Avron et al. 2010], Definition 1.2 in [Candès and Recht 2009]*). The coherence of A is

$$\mu(A) \equiv \max_{1 \leq j \leq m} \ell_j(Q) = \max_{1 \leq j \leq m} \|e_j^T Q\|_2^2.$$

Although coherence contains far less information about a matrix than the leverage scores, it can still be useful in bounding the condition number of a sampled matrix (see Bound 1) and may be easier to estimate than leverage scores. Due to the properties of leverage scores, the inequality $n/m \leq \mu(A) \leq 1$ holds, and if $\mu(A) \approx n/m$, then $\ell_i(A) \approx n/m$. Our code for computing coherence is `coherence.m`.

2. KAPPA_SQ DESIGN

Kappa_SQ was designed to perform all of the computations from [Ipsen and Wentworth 2013] and output paper-ready plots. It can assist researchers in the following ways.

First, the GUI for kappa_SQ has been designed to assist the user set-up, perform and plot experiments on $\kappa(SQ)$. There are two types of experiments, the computation of (possibly probabilistic) bounds on $\kappa(SQ)$ and the computation of $\kappa(SQ)$ for a given or generated test matrix Q . The GUI has also been coded to allow a user to easily incorporate their own codes by simply placing a properly formatted Matlab function in the “boundsAndAlgorithms” directory.

Second, kappa_SQ includes a collection of codes for various algorithms and bounds pertaining to the field of randomized row sampling. These codes are all written as Matlab function files that can be used on their own or with the kappa_SQ GUI. The codes include functions for row sampling, test matrix generation, leverage score distribution generation and functions to compute bounds for $\kappa(SQ)$. The included codes are outlined in section 2.2.

The kappa_SQ codes can be broken up into two main groups, the GUI and “Algorithm Codes.” Below, we describe these codes and their functions.

2.1. kappa_SQ GUI

The kappa_SQ GUI is designed to produce plots of both numerical experiments, where $\kappa(SQ)$ is actually measured, and of bounds on $\kappa(SQ)$. To perform a numerical experiment, kappa_SQ will perform sampling on a matrix and then measure the condition number of the sampled matrix, $\kappa(SQ)$. Occasionally, the sampled matrix, SQ , is not full rank. This event is termed a “failure” event and kappa_SQ also keeps track of these.

When kappa_SQ has completed its computations, it will output plots like those shown in Figures 1 and 2. For the moment, do not worry about the specifics of each plot other than the following. The triangles in figure 1 show the measured condition number of the sampled matrix, $\kappa(SQ)$, and the line plots a bound on $\kappa(SQ)$. For the sections of the domain where a line is not plotted, the bound does not apply. All of the bounds included with $\kappa(SQ)$ are probabilistic bounds and therefore only hold with probability at least $1 - \delta$. Therefore, at least $100(1 - \delta)\%$ of the measured $\kappa(SQ)$ should be below the line. In Figure 2, the “failure rate,” the percent of numerical experiments that resulted in a failure event, is plotted. Despite the fact that for many experiments the failure rate will be 0% for most values, it is still an important quantity because figure 1 only plots the “good” events (where SQ has full column rank).

When a user executes kappaSQ.m, he or she is presented with the kappa_SQ GUI (see Figure 3). It is broken up into sections; the first two correspond to inputs the user must provide to produce a plot and the last two are for plot editing and to display important information. Below we describe these sections and, in the process, how to use the GUI to produce plots.

“Step 1: Select Bounds and/or Numerical Experiments.” This section allows the user to select what he or she would like to plot. The first listbox contains possible bounds that the user can plot. The second listbox contains various sampling methods. In kappa_SQ, numerical experiments are first defined in terms of what sampling algorithm the user would like to use. By selecting a sampling method, the user is telling kappa_SQ that he or she wishes to do a numerical experiment with that sampling method. Selecting multiple sampling methods will perform multiple experiments.

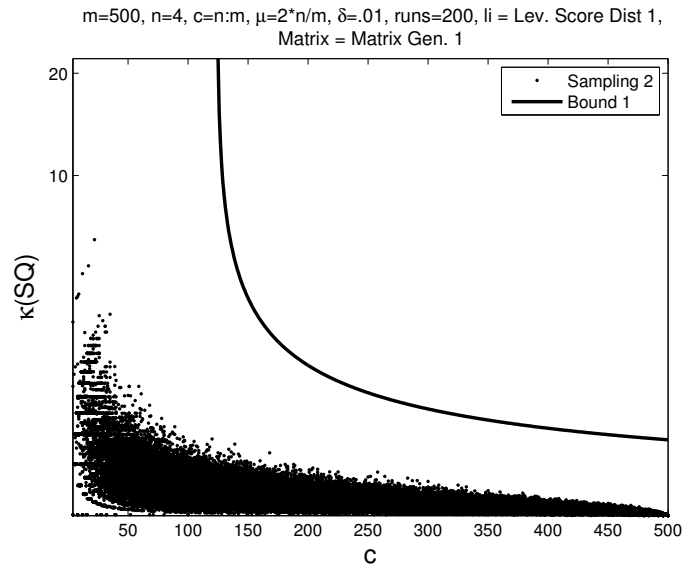


Fig. 1. In this plot we show the results of a numerical experiment (triangles) and a bound on $\kappa(SQ)$ (line) that holds with probability $1 - \delta$.

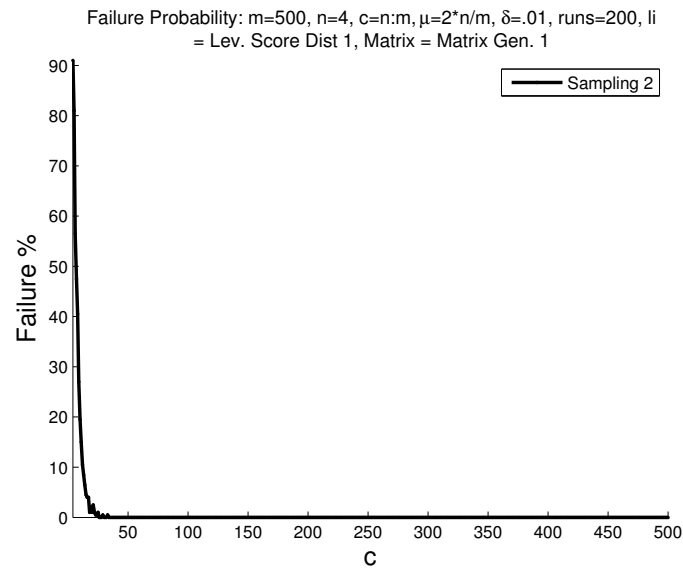


Fig. 2. In this plot we show the failure rate of a numerical experiment on $\kappa(SQ)$.

“Step 2: Matrix Properties / Parameters.” This section allows the user to provide the required inputs. Only inputs that are required will be visible. As an example, if the user chose to plot Bound 1, then this section would ask for the user to provide values for m, n, μ, δ and c as those values are required to compute Bound 1. In addition, either c or μ must be a vector and whichever is a vector will be placed on the x -axis.

Of particular interest in this section are the “Matrix Generation” and “li” (ℓ_i) inputs. When a test matrix is required (ex: when running a numerical experiment), kappa_SQ will generate a matrix using the algorithm specified in this listbox. Similarly, when a leverage score distribution is required, kappa_SQ will generate one by the method specified in the “li” listbox.

“Plot Button.” Once the user has completed steps 1 and 2, he or she may click the plot button to run and plot the experiment.

“Help!” This button will open the kappa_SQ help file. This file includes a FAQ section and a list of included functions.

Adv. Features: “Batch Features.” This section can be viewed by clicking on the “Adv. Features” button. After performing steps 1 and 2, the user can instead add the current experiment to a batch of jobs to be run later in serial. This is particularly useful if the chosen experiments require a long time to run, or if the user has many experiments to run. In addition, if the user keeps all of their experiments defined in a batch file, they can easily repeat all of the experiments for their work.

Adv. Features: “Other Features.” This section contains a button called “Beautify Plots” which will open the plot editing window shown in Figure 4. This window assists the user with modifying many of the common plot settings and creating a script that will apply these settings to future plots. In addition, the plot editing window will generate a command which will apply these settings without the GUI. KappaSQ can be set to run this command for all future plots by checking the “beautify command” checkbox and entering the command. Thus, the user only needs to set up their plots once. Finally, this section also has an option to plot a standard confidence interval for the failure probability.

Below we will discuss the various included codes that the kappa_SQ GUI uses in the above sections to produce plots.

2.2. Algorithm Codes

In this section, we describe the various algorithms and bounds that are included in the kappa_SQ package. These algorithms can be broken up into four main groups, bounds, matrix generation, sampling methods and leverage score distributions.

2.2.1. Sampling methods. We include four different row sampling methods from the literature, Sampling without Replacement, Sampling with Replacement, Bernoulli Sampling, and Sampling Proportional to Leverage Scores. In our recent paper [Ipsen and Wentworth 2013], the first three of these sampling methods are described by constructing a sampling matrix S such that SQ is the sampled matrix. In the kappa_SQ package, we instead code these algorithms to compute $B \equiv SQ$ directly. Each sampling method inputs the initial matrix, Q , and the desired (or desired expected) number of rows to be sampled, c , and outputs the sampled matrix, SQ .

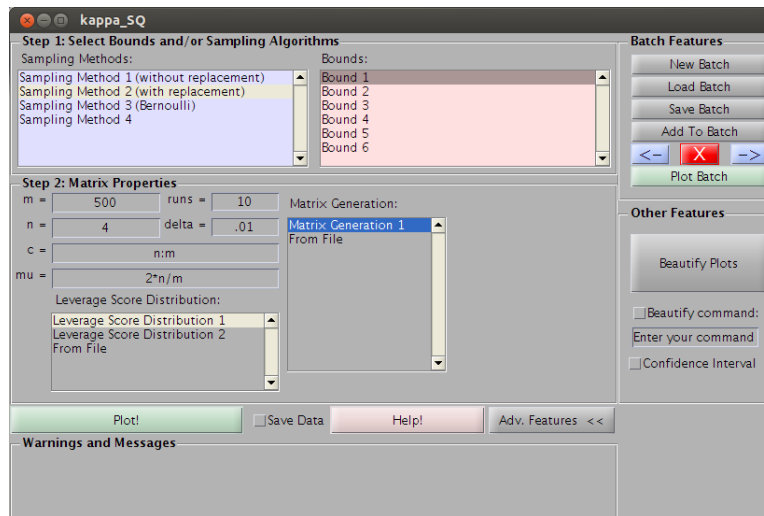


Fig. 3. kappaSQ GUI with advanced features shown.

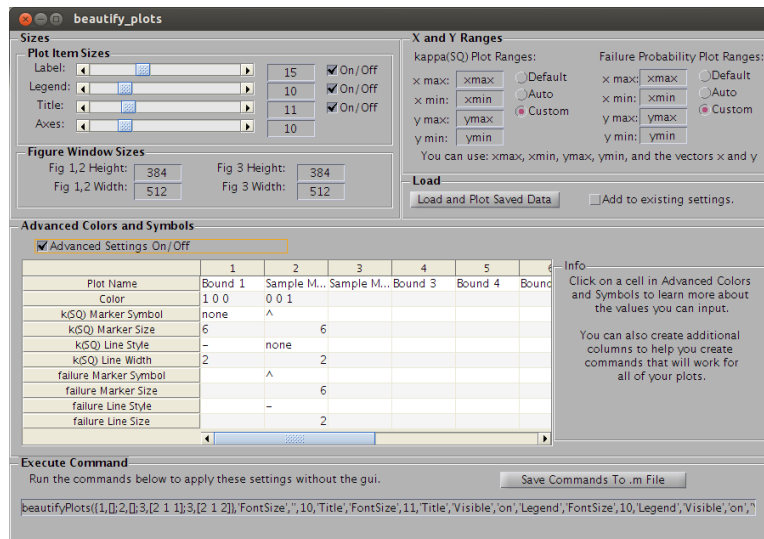


Fig. 4. Beautify Plots GUI.

Sampling Method 1, Sampling Without Replacement. This algorithm samples exactly the desired number of rows such that no row is sampled more than once by sampling uniformly from the $m!/(m-c)!$ possible permutations of c rows. We implement this by first randomly permuting all of the rows and then sampling the first c rows. In the algorithm below we use the term random permutations. A permutation π_1, \dots, π_m of the integers $1, \dots, m$ is a *random permutation*, if it is equally likely to be one of $m!$ possible permutations [Mitzenmacher and Upfal 2006, pages 41 and 48]. The matlab command `randperm(m)` will generate a random permutation of the integers $1, \dots, m$. Our code for this sampling method is `Sample_randperm.m`.

Sampling Method 1: Sampling Without Replacement, [Gittens and Tropp 2011; Gross and Nesme 2010]

Input: A $m \times n$ matrix Q and an integer c , such that $1 \leq c \leq m$.

Output: A $c \times m$ sampled matrix B .

$v = \text{randperm}(m)$;

$s = v(1 : c)$;

$B = \sqrt{m/c} Q(s, :)$;

Sampling Method 2, Sampling With Replacement (Exactly(c)). This algorithm samples exactly the desired number of rows with a uniform probability distribution and with replacement. Our code for implementing this sampling method is `Sample_exactlyC.m`.

Sampling Method 2: Sampling With Replacement, [Drineas et al. 2006; Drineas et al. 2011]

Input: A $m \times n$ matrix Q and an integer c , such that $1 \leq c \leq m$.

Output: A $c \times m$ sampled matrix B .

Let π_1, \dots, π_c be integers uniformly sampled from $\{1, \dots, m\}$ with replacement;

$s = [\pi_1, \dots, \pi_c]$;

$B = \sqrt{m/c} Q(s, :)$;

Sampling Method 3, Bernoulli Sampling. In this sampling method, each row is either sampled, with probability c/m , or not sampled with probability $1 - c/m$. Thus, whether or not each row is sampled is an independent Bernoulli trial and the expected total number of rows sampled is c . Our implementation of this algorithm differs slightly from [Ipsen and Wentworth 2013]. In our paper, rows that are not sampled are set to 0, while in our code they are removed. Removing the zero rows is more memory efficient, avoids unnecessary matrix-matrix multiplications and does not affect $\kappa(SQ)$. Our code for this algorithm is `Sample_bernoulli.m`.

Sampling Method 3: Bernoulli Sampling, [Avron et al. 2010; Gittens and Tropp 2011; Gross and Nesme 2010]

Input: A $m \times n$ matrix Q and an integer c , such that $1 \leq c \leq m$.

Output: A $c \times m$ sampled matrix B .

Let π be a $m \times 1$ vector of m independent realizations of a boolean random variable with success probability c/m ;

Let s be a $\hat{c} \times 1$ vector containing the indices where $\pi_i = 1$, where $1 \leq i \leq m$ and $\hat{c} =$ the number of nonzero entries in π ;

$B = \sqrt{m/c} Q(s, :)$;

Sampling Method 4, Sampling Proportional to Leverage Scores. In this sampling method, c rows are sampled with probability $\ell_i(Q)/n$ with replacement. Our code for implementing this sampling method is `Sample_leverageScores.m`

Sampling Method 4: Sampling Proportional to Leverage Scores

Input: A $m \times n$ matrix Q and an integer c , such that $1 \leq c \leq m$ and the leverage scores $\ell(Q)$.

Output: A $c \times m$ sampled matrix B .

Let π_1, \dots, π_c be integers sampled from $\{1, \dots, m\}$ with probabilities $\{\ell_1(Q)/n, \dots, \ell_m(Q)/n\}$ and replacement;

$s = [\pi_1, \dots, \pi_c]$;

$B = \sqrt{m/c} Q(s, :)$;

2.2.2. Bounds. We include codes for the two probabilistic bounds for $\kappa(SQ)$ from our recent paper [Ipsen and Wentworth 2013] and four other weaker bounds that were included in the first version of our paper [Ipsen and Wentworth 2012].

Bound 1, Coherence based bound. This bound is expressed in terms of coherence and comes from a matrix Chernoff concentration inequality [Tropp 2011, Corollary 5.2]. It applies to all of the sampling methods 2.2.1 except for Sampling Proportional to Leverage Scores (Sampling Method 4). Our code for this bound is `Bound_muBound.m`.

BOUND 1 ([IPSEN AND WENTWORTH 2013, THEOREM 4.1]). *Let Q be a real $m \times n$ matrix with $Q^T Q = I_n$ and coherence μ . Let SQ be a sampling matrix produced by Algorithms 1, 2, or 3 with $n \leq c \leq m$. For $0 < \epsilon < 1$ and $f(x) \equiv e^x(1+x)^{-(1+x)}$ define*

$$\delta \equiv n \left(f(-\epsilon)^{c/(m\mu)} + f(\epsilon)^{c/(m\mu)} \right). \quad (1)$$

If $\delta < 1$, then with probability at least $1 - \delta$ we have $\text{rank}(SQ) = n$ and

$$\kappa(SQ) \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}}.$$

Bound 2, Leverage score based bound. This bound is based on leverage scores and only applies to sampling without replacement (Sampling Method 1). It is based on a matrix Bernstein concentration inequality [Recht 2011, Theorem 4][Ipsen and Wentworth 2013, Theorem 8.2]. Our code for this bound is `Bound_leverageScoresBound.m`.

BOUND 2 ([IPSEN AND WENTWORTH 2013, THEOREM 5.2]). *Let Q be a $m \times n$ real matrix with $Q^T Q = I_n$, leverage scores $\ell_j(Q)$, $1 \leq j \leq m$, and coherence μ . Let L be a diagonal matrix such that $L_{j,j} = \ell_j(Q)$. Let S be a sampling matrix produced by Algorithm 2 with $n \leq c \leq m$. For $0 < \epsilon < 1$ set*

$$\delta \equiv 2n \exp \left(-\frac{3}{2} \frac{c\epsilon^2}{m(3\|Q^T L Q\|_2 + \epsilon\mu)} \right).$$

If $\delta < 1$, then with probability at least $1 - \delta$ we have $\text{rank}(SQ) = n$ and

$$\kappa(SQ) \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}}.$$

Bound 3, Weaker coherence based bound. This bound is based on a probabilistic two-norm bound for a Monte Carlo matrix multiplication algorithm that samples according to Sampling Method 2 [Drineas et al. 2011, Theorem 4]. Our code for this bound is `weakerBound_1.m`.

BOUND 3 ([IPSEN AND WENTWORTH 2012, THEOREM 3.2]). *Given $0 < \epsilon < 1$ and $0 < \delta < 1$, let Q be a $m \times n$ real matrix with $Q^T Q = I_n$ and coherence μ . Let c be an*

integer so that

$$\min \left\{ n, \zeta \ln \left(\zeta / \sqrt{\delta} \right) \right\} \leq c \leq m, \quad \text{where} \quad \zeta \equiv \frac{96m\mu}{\epsilon^2}.$$

If S is a $c \times m$ matrix produced by Sampling Method 2 with uniform probabilities $p_k = 1/m$, $1 \leq k \leq m$, then with probability at least $1 - \delta$, we have $\text{rank}(SQ) = \text{rank}(M_s) = n$ and

$$\kappa(SQ) = \kappa(AR_s^{-1}) \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}}.$$

Bound 4, Weaker coherence based bound. This bound is based on a special case of the noncommutative Bernstein inequality [Recht 2011, Theorem 4] and applies to sampling Sampling Method 2. Our code for this bound is weakerBound_3.m.

BOUND 4 ([IPSEN AND WENTWORTH 2012, COROLLARY 3.10]). Given $c \geq n$ and $0 < \delta < 1$, let Q be a $m \times n$ real matrix with $Q^T Q = I_n$ and coherence μ . Let $\rho \equiv \frac{2}{3} \ln(2n/\delta)$ and

$$\epsilon_1 \equiv \frac{\mu m}{2c} \left(\rho + \sqrt{\frac{12c\rho}{m\mu} + \rho^2} \right).$$

Let S be a $m \times m$ matrix produced by Algorithm 2. If $\epsilon_1 < 1$ then with probability at least $1 - \delta$, we have $\text{rank}(SQ) = n$ and

$$\kappa(SQ) \leq \sqrt{\frac{1+\epsilon_1}{1-\epsilon_1}}.$$

Bound 5, Weaker coherence based bound. This bound is based on a Frobenius norm bound for a Monte Carlo matrix multiplication algorithm that samples according to Sampling Method 2 and applies to sampling Sampling Method 2. Our code for this bound is weakerBound_4.m.

BOUND 5 ([IPSEN AND WENTWORTH 2012, THEOREM 3.5]). Given $0 < \delta < 1$ and $c \geq n$, let Q be a $m \times n$ real matrix with $Q^T Q = I_n$ and coherence μ . Let

$$\epsilon_2 \equiv \sqrt{\frac{mn\mu}{c}} + m\mu\sqrt{\frac{8\log(1/\delta)}{c}}.$$

Let S be a $c \times m$ matrix produced by Algorithm 2 with uniform probabilities $p_k = 1/m$, $1 \leq k \leq m$. If $\epsilon_2 < 1$, then with probability at least $1 - \delta$, we have $\text{rank}(SQ) = n$ and

$$\kappa(SQ) \leq \sqrt{\frac{1+\epsilon_2}{1-\epsilon_2}}.$$

Bound 6, Weaker coherence based bound. This bound is again based on the noncommutative Bernstein inequality in [Recht 2011, Theorem 4] and applies to sampling Sampling Method 3. Our code for this bound is weakerBound_6.m.

BOUND 6 ([IPSEN AND WENTWORTH 2012, COROLLARY 4.3]). Given $m \geq n$, $0 < \gamma < 1$ and $0 < \delta < 1$, let Q be a $m \times n$ real matrix with $Q^T Q = I_n$ and coherence μ . Let $\rho \equiv \frac{2}{3} \ln(2n/\delta)$ and

$$\hat{\epsilon}_3 \equiv \frac{\mu}{2} \left(\phi\rho + \sqrt{\frac{1-\gamma}{\gamma} 12m\rho + \phi^2\rho^2} \right), \quad \phi = \begin{cases} 1 & \text{if } \gamma \geq 1 - \gamma \\ \frac{1-\gamma}{\gamma} & \text{if } 1 - \gamma > \gamma \end{cases}$$

Let S be a $m \times m$ matrix produced by Algorithm 3. If $\hat{\epsilon}_3 < 1$ then with probability at least $1 - \delta$, we have $\text{rank}(SQ) = n$ and

$$\kappa(SQ) \leq \sqrt{\frac{1 + \hat{\epsilon}_3}{1 - \hat{\epsilon}_3}}.$$

2.2.3. Leverage score distribution. We include code for two functions which define leverage score distributions.

Leverage Score Distribution 1, Good leverage score distribution. The first function is designed to be an ideal case for row sampling. It outputs a leverage score distribution with one leverage score is set equal to the coherence and the remaining leverage scores all identical. Thus, most rows are equally “important” and uniform row sampling should work well. The code for this algorithm is `liDist_oneBig.m`.

Leverage Score Distribution 1: *Good leverage score distribution [Ipsen and Wentworth 2013, Algorithm 6.2]*

Input: Integers m and n such that $m \geq n \geq 1$, and desired coherence μ .

Output: A $m \times 1$ vector, ℓ of leverage scores such that $\max \ell = \mu$.

$\ell = [\mu; \text{ones}(m - 1, 1)(n - m\mu)/(m - 1)];$

Leverage Score Distribution 2, Bad leverage score distribution. The second function is designed to be a particularly bad case for row sampling. It outputs a leverage score distribution with the maximal number of entries set equal to the coherence and at most one additional non-zero entry. Matrices with this leverage score distribution will have the maximal number of zero rows for the given coherence. Zero rows are bad for uniform row sampling since zero rows contain no information. The code for this algorithm is `liDist_manyBig.m`.

Leverage Score Distribution 2: *Bad leverage score distribution [Ipsen and Wentworth 2013, Algorithm 6.3]*

Input: Integers m and n such that $m \geq n \geq 1$, and desired coherence μ .

Output: A $m \times 1$ vector, ℓ of leverage scores such that $\max \ell = \mu$.

$\tilde{m} = \lfloor n/\mu \rfloor;$

if $\tilde{m} < m$ **then**

$\ell = [\mu \text{ones}(\tilde{m}, 1); n - \tilde{m}\mu, \text{zeros}(m - \tilde{m}, 1)];$

else

$\ell = \mu \text{ones}(\tilde{m}, 1)$

end

2.2.4. Test matrix generation. Included in `kappa.SQ`, we provide code for a deterministic matrix generation algorithm.

Matrix Generation 1, Deterministic matrix generation algorithm. This algorithm inputs the desired matrix dimensions and leverage scores and outputs a test matrix, Q , with these properties. To compute Q , the algorithm applies $m - 1$ Givens rotations to the matrix $Q = [I_n \quad \text{zeros}(n, m - n)]^T$. Each Givens rotation alters the leverage scores of two rows such that at least one of the two rows has the desired

leverage score. The reason that only $m - 1$ Givens rotations are required is that the leverage scores sum to n and thus the final leverage score is determined by the other $m - 1$ leverage scores. We note here that this algorithm is a transposed version of [Dhillon et al. 2005, Algorithm 3] and that the Givens rotations are computed from numerically stable expressions [Dhillon et al. 2005, section 3.1]. The code for this algorithm is `mtxGen.li.m`.

Matrix Generation Algorithm 1: Matrix Generation 1

Input: Integers m and n such that $m \geq n \geq 1$, and a $m \times 1$ vector l of the desired leverage scores.

Output: A $m \times n$ matrix Q with orthonormal columns and the desired leverage scores.

```

 $Q = [I_n \text{ zeros}(n, m - n)]^T$ ;
 $[l, I] = \text{sort}(l)$ ; % Sort and store original order;
 $i = m - n$ ;
 $j = m - n + 1$ ;
for  $dummyVar = 1 : m - 1$  do
    if  $|l_i - \|e_i^T Q\|_2| < |l_j - \|e_j^T Q\|_2|$  then
        Rotate rows  $i$  and  $j$  of  $Q$  so that  $\|e_i^T Q\|_2^2 = l_i$ ;
         $i = i - 1$ ;
    else
        Rotate rows  $i$  and  $j$  of  $Q$  so that  $\|e_j^T Q\|_2^2 = l_j$ ;
         $j = j + 1$ ;
    end
end
 $Q(I, :) = Q$ ; % Undo sorting;

```

2.3. Other Functions

We also include two simple functions to assist with choosing nice, aesthetically pleasing, ranges for c and μ named `logPoints.m` and `logPointsDouble.m`, respectively. These functions produce ranges that are more heavily weighted towards the smaller end of the desired range. Most of the interesting action in the final plots occurs near smaller c or μ values and, in addition, larger values of c are more computationally expensive. We describe these functions in the `kappa_SQ` help file which can be accessed by pressing the help button in the gui (see Section 2.1).

3. EXAMPLES

Here we show a few examples of some of the ways that the `kappa_SQ` GUI can be used.

Example 1: In this example, we show how to perform a basic experiment with the GUI. In this experiment we compare Bound 1 to a numerical experiment with sampling Sampling Method 2. Since Bound 1 applies to this sampling method, the results should show that at least $100(1 - \delta)\%$ of the measured $\kappa(SQ)$ are less than the bound. In order for `kappa_SQ` to run a numerical experiment, it must have a test matrix to work on. Here, we will chose to generate a test matrix with Sampling Method 1 and leverage scores defined by Sampling Method 1.

To set up the experiment, start by selecting the desired bound and sampling method. Then, move on to step 2 and input the following values, $m=500$, $n=4$ $c=n:m$,

$\mu=2n/m$, $\text{runs}=10$, and $\delta=.01$. For the “Matrix Generation” listbox, select “Matrix Generation 1” (Sampling Method 1). This will cause the leverage score listbox to appear since this matrix generation algorithm requires a leverage score distribution. Select “Leverage Score Distribution 1” (Sampling Method 1) for the leverage score distribution. In figure 3 we show how the GUI should at this point. When ready, click the plot button to begin the experiment. We show the resulting plot in figure 5.

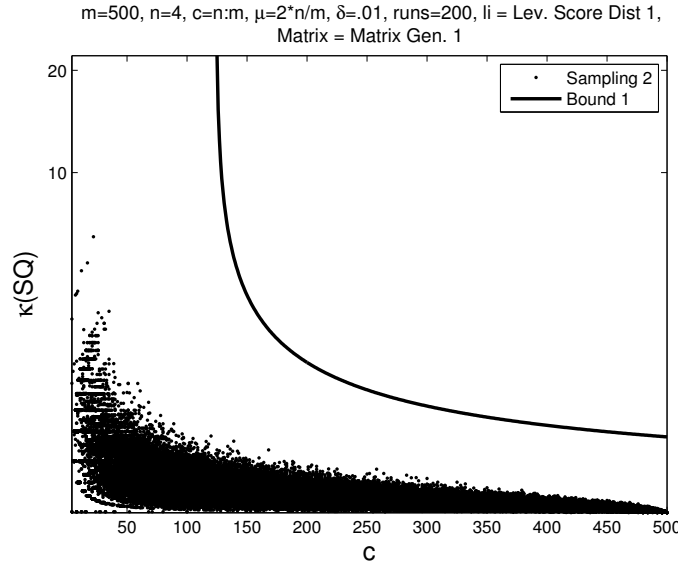


Fig. 5. Resulting plots produced after clicking on the plot-button shown in Figure ?? . The solid line shows Bound 1 and the triangles show the results of the numerical experiments with sampling Sampling Method 2 and a matrix generated by Sampling Method 1.

Here Q is a matrix generated by Algorithm 1 with orthonormal columns, $m = 10,000$, $n = 4$, coherence $\mu = 20n/m$. Left panel: Horizontal coordinate axes represent amounts of sampling $n \leq c \leq 10,000$. Vertical coordinate axes represent condition numbers $\kappa(SQ)$; the maximum is 10. Right panels: Horizontal coordinate axes represent amounts of sampling that give rise to numerically rank deficient matrices SQ . Vertical coordinate axes represent percentage of numerically rank deficient matrices.

Example 2: Here we show how to set up a kappa_SQ batch to perform multiple experiments in serial. To create a new batch, first click the “Adv. Features” button to expand the GUI and then click on the “New Batch” button to start a new batch. Next, set up an experiment by performing steps 1 and 2 as described in the first example. Then, instead of clicking the plot button, click the “Add to Batch” button. This will add the current experiment to the batch. Repeat this process for the remaining experiments. The user may use the arrow buttons to navigate and the “X” button to delete previously entered experiments. When ready, click the “Save Batch” button to save the current experiments to a file and then the “Run Batch” button to have kappa_SQ run all of the experiments. Plot images will be saved automatically with a file name based on the batch file name and their job number.

Example 3: Here we show how the built in plot editing tools can be used to expedite plot editing, how to create a script that will apply these settings to future plots and set up kappa_SQ to run that script after every experiment. To start, run an experiment

as described in example 1. Then, click the "Adv. Features" button to expand the GUI and then click on the "Beautify Plots" button. This will open the plot editing window. (See figure 4).

This window allows the user to easily edit many different plot settings. While using this window, any changes will instantly be applied to the plots, so we suggest positioning the plots and the GUI window such that they can all be seen. When done editing, press the button labeled "Save Commands To .m File" to create a .m file that will apply these settings to future plots. To have kappa_SQ apply these plot settings automatically to all new plots, write the command for this file in the box labeled "Enter your command here" in the main GUI window and check the "Beautify Command" checkbox. (See figure 3).

4. CONCLUSIONS

The kappa_SQ package is designed to assist researchers examine the behavior of $\kappa(SQ)$. The package includes codes for generating matrices with specific leverage score distributions, generating a two specific leverage score distributions, four types of row sampling methods and computing bounds on $\kappa(SQ)$. These codes can be used on their own, or with the kappa_SQ GUI which is capable of setting up and running numerical experiments, computing bounds, and producing quality plots with the help of custom plot-editing tools. In addition, the GUI has been designed to detect properly formatted Matlab function files which allows the user to incorporate their own codes into the GUI.

REFERENCES

- H. Avron, P. Maymounkov, and S. Toledo. 2010. Blendenpik: Supercharging Lapack's least-squares solver. *SIAM J. Sci. Comput.* 32, 3 (2010), 1217–1236.
- E. J. Candès and B. Recht. 2009. Exact Matrix Completion via Convex Optimization. *Found. Comput. Math.* 9 (2009), 717–772.
- I. S. Dhillon, R. W. Heath, M. A. Sustik, and J. A. Tropp. 2005. Generalized finite algorithms for constructing Hermitian matrices with prescribed diagonal and spectrum. *SIAM J. Matrix Anal. Appl.* 27, 1 (2005), 61–71.
- P. Drineas, R. Kannan, and M. W. Mahoney. 2006. Fast Monte Carlo Algorithms for Matrices. I. Approximating Matrix Multiplication. *SIAM J. Comput.* 36, 1 (2006), 132–157.
- P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. 2011. Faster Least Squares Approximation. *Numer. Math.* 117 (2011), 219–249.
- A. Gittens and J. A. Tropp. 2011. Tail Bounds for All Eigenvalues of a Sum of Random Matrices. (2011). arXiv:1104.4513.
- D. Gross and V. Nesme. 2010. Note on Sampling without Replacement from a Finite Collection of Matrices. (2010). arXiv:1001.2738.
- D. C. Hoaglin and R. E. Welsch. 1978. The Hat Matrix in Regression and ANOVA. *Amer. Statist.* 32, 1 (1978), 17–22.
- Ilse C. F. Ipsen and Thomas Wentworth. 2012. The Effect of Coherence on Sampling from Matrices with Orthonormal Columns, and Preconditioned Least Squares Problems. (March 2012). <http://arxiv.org/abs/1203.4809v1>; <http://arxiv.org/pdf/1203.4809v1>
- Ilse C. F. Ipsen and Thomas Wentworth. 2013. The Effect of Coherence on Sampling from Matrices with Orthonormal Columns, and Preconditioned Least Squares Problems. (May 2013). <http://arxiv.org/abs/1203.4809v2>; <http://arxiv.org/pdf/1203.4809v2>
- X. Meng, M. A. Saunders, and M. W. Mahoney. 2011. LSRN: A Parallel Iterative Solver for Strongly Over- or Under-determined Systems. (2011). arXiv:1109.5981v1.
- M. Mitzenmacher and E. Upfal. 2006. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York.
- B. Recht. 2011. A simpler Approach to Matrix Completion. *J. Machine Learning* 12 (2011), 3413–3430.

J. A. Tropp. 2011. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.* (2011), 1–46.

5. APPENDIX

5.1. Notation

We use the following notation through out this paper.

- m, c and n are integers such that $m \geq c \geq n > 0$.
- $\|\cdot\|_2$ denotes the standard 2-norm.
- A^T denotes the transpose of A .
- e_i denotes the canonical vector with a 1 in the i^{th} position and zeros everywhere else.
- A is a $m \times n$ full column rank matrix.
- Q is a $m \times n$ matrix with orthonormal columns that span the column space of A .
- S is a $c \times n$ random row sampling matrix.
- $\kappa(A) \equiv \|A\|_2 \|A^\dagger\|_2$ denotes the two-norm condition number of a $m \times n$ full column rank matrix A , where A^\dagger is the Moore-Penrose inverse.
- $I_k = (e_1 \dots e_k)$ denotes the $k \times k$ identity matrix.
- $\text{ones}(m \times n)$ denotes the $m \times n$ matrix of all ones.
- $\text{zeros}(m \times n)$ denotes the $m \times n$ matrix of all zeros.
- $\mu(A)$ denotes the coherence of A .
- $\ell(A)$ denotes the $m \times 1$ vector containing the leverage scores of A , and $\ell_i(A)$ denotes the i^{th} leverage score of A .
- L is a diagonal matrix with the leverage score of Q on the diagonal.
- δ is a number such that $0 < \delta < 1$ and is referred to as the failure probability.