Low Rank Factorization Using Error Correcting Codes
APPM 5720 Final Project

James Folberth and Jessica Gronski

1 Introduction

In recent years, randomized techniques have gained a significant amount of attention for their simplicity and ability to efficiently generate low rank approximations of large matrices. We have discussed several randomized methods in class including the Subsampled Random Fourier and Hadamard Transforms (SRFT and SRHT, respectively). Ubaru et al. [6] propose using error correcting codes as sampling operators for the RSVD, much like the SRFT and SRHT. Our aim for the project is to implement their ideas and compare it to other sampling methods we saw in class.

2 Error Correcting Codes

In what follows, we give a brief overview of error correcting codes and the subspace embedding properties they satisfy. In communication systems, data are transmitted from a source (transmitter) to a destination (receiver) through physical channels. These channels are usually noisy, causing errors in the data received [6], and so error correcting codes were designed to help remedy this issue. BCH codes, named after Bose, Chaudhuri, and Hocquenghem, are of specific importance for their rich structure, ability to correct multiple bit errors and ease of decoding a transmitted message. They form a class of cyclic codes; if $\mathbf{c}$ is a codeword in the linear code space C, then $\tilde{c}$ is also in C where $\tilde{c}$ is a cyclic shift of $\mathbf{c}$.

A linear code C can be represented by the codeword length ℓ and message length r. For any two integers t and q, a BCH code over the binary field $\mathbb{F}_2$ has length $\ell = 2^q - 1$ and dimension $r = 2^q - 1 - tq$. In order to encode a message of r bits, we require 2^r unique code words and so C has matrix dimension $2^r \times \ell$. Another feature of BCH codes is that any two codewords maintain a minimum distance of at least $2t + 1$, where we define distance as the number of positions at which the codes differ. Moreover, we can define the *dual* of the code as a code of length $\ell' = 2^q - 1 = \ell$, dimension $r' = tq$ and minimum distance at least $2^{q-1} - (t-1)2^{q/2}$. The notion of a dual distance is of particular importance because a code matrix with dual distance greater than 4 will satisfy the Johnson-Lindenstrauss Transform (JLT) property and allow for an efficient subspace embedding. Therefore, for a fixed integer q (fixed codeword length), the length of the message that can be transmitted through a channel efficiently depends on the value of t.

3 Subsampled Code Matrix

Let A be an $m \times n$ matrix with approximate rank k. Similar to the randomized methods we've discussed in class, we would like to construct a lower dimensional subsampling matrix Ω such that $Y = A\Omega$ provides a "good" approximation for the column space of A. Ideally, the size of the subsampling matrix, i.e. the target rank, is much smaller than the dimension of the original matrix A, greatly reducing the cost of performing other factorizations like the QR or SVD on the subsampled matrix.

The question now is how do we construct an Ω from error correcting codes that still preserve the geometry of the original problem? We choose the length of the message vector $r \geq \lceil \log_2 n \rceil$ and the length of the codewords $l > k$, the target rank. Notice that l will depend on the dual distance of the code. We then consider the linear coding scheme as follows:

$$\Omega = \sqrt{\frac{2^r}{\ell}} DS\Phi,$$

where

- D is a random $n \times n$ diagonal matrix whose entries are independent random signs, i.e., random variables uniformly distributed on $\{\pm 1\}$.
- S is a uniformly random downsampler, an $n \times 2^r$ matrix whose n rows are randomly selected from a $2^r \times 2^r$ identity matrix.
- Φ is the $2^r \times \ell$ coding matrix, generated using an $[\ell, r]$-linear coding scheme, with a binary phase-shift keying (BPSK) mapping and scaled by $2^{-r/2}$ such that all columns have unit norm.

Recall that the codeword matrix C has 2^r codewords each of length ℓ, i.e. a set of 2^r vectors in $\{0, 1\}^\ell$. The BPSK mapping is defined as follows: given a codeword $\mathbf{c} \in C$, $\mathbf{c} \mapsto \phi \in \mathbb{R}$ by assigning $1 \to \frac{-1}{\sqrt{2^r}}$ and $0 \to \frac{1}{\sqrt{2^r}}$. This way, we transform the binary code matrix C to the code matrix $\Phi = (\phi_1^T, \dots \phi_{2^r}^T)^T$ [6]. Moreover, it can be shown that if the dual distance is greater than or equal to 3, Φ will have orthonormal columns. The scaling factor $\sqrt{\frac{2^r}{\ell}}$ enforces the rows of Ω to also have unit length.

4 Prototype Algorithm

The RSVD algorithm provided by Ubaru et al. in [6] :

Algorithm 1 Prototype Algorithm

Input: An $m \times n$ matrix A, a target rank k.
Output: Rank-k factors U, Σ and V in an approximate SVD $A \approx U\Sigma V^T$.

1. Form an n subsampled code matrix Ω, as described above using an $[\ell, r]$-linear coding scheme, where $\ell > k$ and $r \geq \lceil \log_2 n \rceil$.
2. Form the $m \times \ell$ sample matrix $Y = A\Omega$.
3. Form an $m \times \ell$ orthogonal matrix Q such that $Y = QR$.
4. Form the $\ell \times n$ matrix $B = Q^T A$.
5. Compute the SVD of the small matrix $B = \hat{U}\Sigma V^T$.
6. Form the matrix $U = Q\hat{U}$.

A drawback to the algorithm is that it requires the singular values of the input matrix to decay rapidly for a good approximation. Similar to other randomized techniques, the Prototype Algorithm applies a unitary transform to reduce the coherence of an input matrix A; this in turn allows for an effective subsampling. Moreover, dual BCH code matrices share similar structured properties to the Subsampled Random Hadamard Transform. A key advantage of using structured matrices (SRHT or SRFT) in randomized sketching algorithms is that we can compute the matrix-matrix products $Y = A\Omega$ in $O(mn \log_2 \ell)$ time exploiting the structure of Hadamard/Fourier matrices. For a general dense input matrix in RAM, the total computational cost of Algorithm 1 using Subsampled Code Matrices is $O(mn \log_2 \ell + (m+n)k^2)$ [6].

4.1 Subsampled Code Matrices, Johnson Lindenstrauss Transform and Subspace Embedding

Much of this subsection is taken directly from [6]. For more details or proofs of what follows, please refer to Ubaru et. al [6].

In order to satisfy the $(1+\epsilon)$ error bound, we require SCMs to satisfy the two key properties: JLT and subspace embedding. The first property is a result given by Clarkson and Woodruff in [2]. They show that error correcting dual BCH codes with with more than 2 error correcting ability will satisfy the JLT property. This is achieved when the dual distance of the BCH code is greater than 4. In order to show that SCMs satisfy the subspace embedding property, we use the following lemma:

Lemma 1. *Let $0 < \epsilon, \delta < 1$ and f be some function. If $\Omega \in \mathbb{R}^{n \times \ell}$ satisfies a JLT (ϵ, δ, d) with $\ell = O(k \log(k/\epsilon)/\epsilon^2 . f(\delta))$, then for any orthonormal matrix $V \in \mathbb{R}^{n \times k}, n \geq k$ we have*

$$\boldsymbol{Pr}(||V^T \Omega \Omega^T V - I||_2 \leq \epsilon) \geq 1 - \delta.$$

Thus, any sampling matrix Ω satisfying JLT, that is with dual distance greater than 4, and having length $\ell = O(k \log(k/\epsilon)/\epsilon^2)$ satisfies the subspace embedding property. This result shows that SCM matrices can preserve the geometry of the top k-singular vectors of the input matrix A.

5 RSVD Experiments

We implemented sampling with dual BCH code matrices using various tools from the MATLAB Communications System Toolbox [5]. The toolbox provides a primal BCH encoder in MATLAB script. The main operation in encoding is simulating a shift register, which is vectorized. The toolbox also provides a function, `bchgenpoly`, which generates a vector over $\mathbb{F}_2$ that represents the generating polynomial, $F_p(x)$, for the primal BCH code. The dual generating polynomial is

$$F_d(x) = \frac{x^l + 1}{F_p(x)},$$

where the remainder is 0 [4]. Polynomial division over $\mathbb{F}_2$ is (thankfully) implemented in `deconv`, which is provided in the Communications System Toolbox. Once we have the dual generating polynomial, we may borrow the shift register from the primal BCH encoder, and we have a functioning dual BCH encoder.

To compute the sampling matrix Ω, we draw n Rademacher variables to form D, and draw the indices of the subsampling matrix S. We then compute a $n \times r$ message matrix with each row a binary vector corresponding to the decimal index from S. Finally, we compute the $n \times \ell$ subsampled code matrix $S\Phi$ by encoding the message matrix with the dual BCH encoder. To be clean, we are forming the subsampled code matrix, and use BLAS to apply it to the data matrix A. We will see in the next section a method of efficiently sampling A with the SRHT.

For the first experiment, we compare various sampling methods for the RSVD with oversampling parameter $p = 5$ and $q = 0$ power iterations. We use the matrix `LOCAL_fast_decay(100,140,200)`, which was used frequently in our homeworks. Figure 1 shows the spectral and Frobenius norms for the various sampling methods. We can see that sampling with dual BCH code matrices are competitive with Gaussian matrices and the SRFT. If we use $q = 1$ or more power iterations, we are not able to resolve any differences between the sampling methods.

As a second experiment with the RSVD, we attempt to recreate one result from [6]. They selected the matrix `Kohonen.mat`, an adjacency matrix for a directed graph, from the UFL sparse matrix collection [3]. Figure 2 shows the spectral and Frobenius norm errors, again with oversampling $p = 5$ and $q = 0$ power iterations. Notice that the singular values of `Kohonen.mat` do *not* decay quickly, and so the RSVD is not accurate, regardless of sampling. Nevertheless, dual BCH codes performed slightly better than the SRFT and Gaussian matrices.

One downside to sampling with dual BCH code matrices is the restriction of ℓ to $\ell = 2^q - 1$. If we have a target rank of $k = 100$, we must pick $p = 27$ which gets us to the nearest admissible code length of $\ell = 127$. This does add a bit of work, but is offset by the existence of a fast multiplication via the SRHT, discussed in the following section.

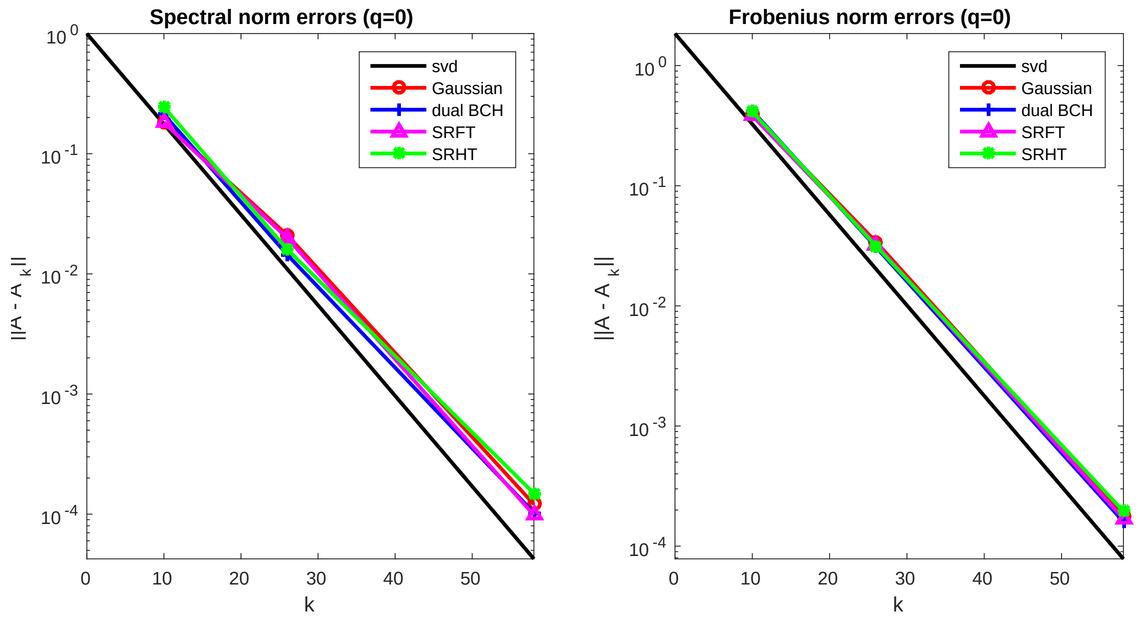

Figure 1: RSVD with various sampling methods on `LOCAL_fast_decay(100,140,200)`.

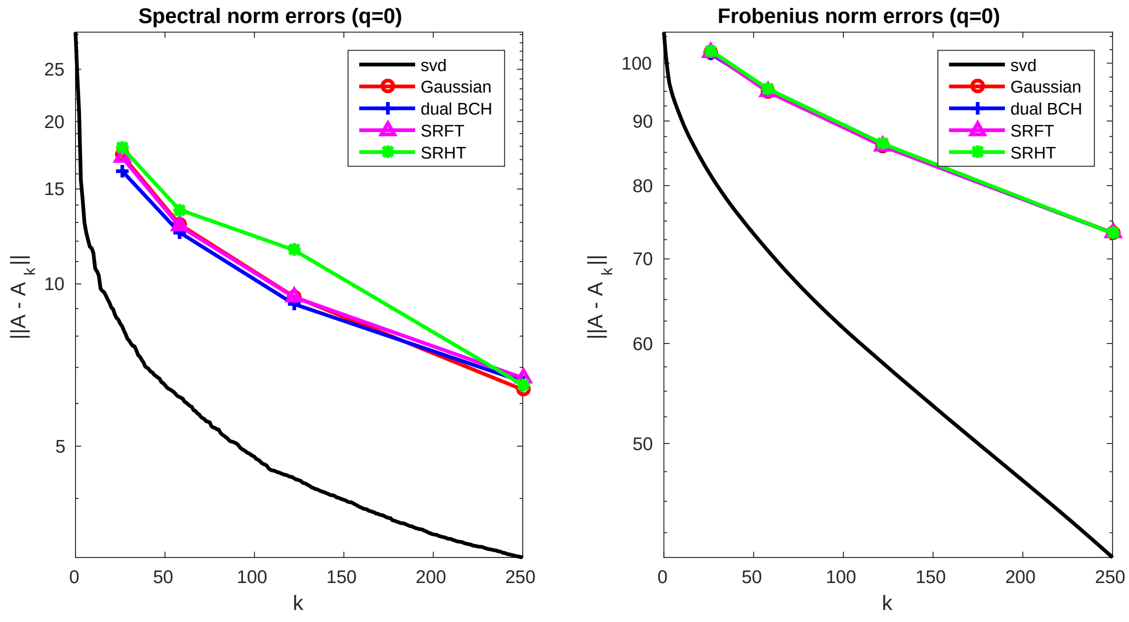

Figure 2: RSVD with various sampling methods on `Kohonen.mat` from [3].

6 An efficient SRHT

The naturally ordered (Walsh-)Hadamard transform is a structured orthogonal transform inspired by the DFT and Haar wavelet transforms. Neglecting normalization, it can be defined recursively for sizes $d = 2^r$ as

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_d = \begin{bmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{bmatrix}. \tag{1}$$

Using this recursion [1], we also have a fast algorithm for computing the transform of a vector (or matrix, column-wise). Notice that each Hadamard matrix H_d has ± 1 entries. Recall that after the BPSK mapping, the primal and dual BCH code matrices also have ± 1 entries. In [1] it was shown that every row of the $2^r \times \ell$ code matrix Φ is equal to some row of the $2^r \times 2^r$ Hadamard matrix, up to some normalization. Therefore, if we do some offline preprocessing to determine the proper row subsampling for the Hadamard transform, we can use the subsampled Hadamard transform to compute sampling with subsampled dual BCH code matrices and so sampling with dual BCH codes enjoys a fast transform, just like the SRFT and SRHT.

The recursive definition of the naturally ordered Hadamard transform (1) leads naturally to an $\mathcal{O}(d \log d)$ implementation, by computing the full length d transform and then subsampling to, say, k indexes. However, we can do better, although it seems that no one has implemented and made publicly available a smarter transform.

Ailon and Liberty pointed out that by including the subsampling matrix in the recursion, we can reduce the complexity to $\mathcal{O}(d \log k)$, which should yield an improved runtime when $k \ll d$ [1]. Let S be a row subsampling matrix, taking in d rows and keeping only k rows. We can define the SRHT recursively as

$$SH_d x = \begin{bmatrix} S_1 & S_2 \end{bmatrix} \begin{bmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = S_1 H_{d/2}(x_1 + x_2) + S_2 H_{d/2}(x_1 - x_2), \tag{2}$$

where x_1 and x_2 are the first and second halves of x; S_1 and S_2 sample k_1 and k_2 rows, respectively; and $k_1 + k_2 = k$. The algorithm proceeds by computing $x_1 + x_2$ and $x_1 - x_2$, and then recursing to compute $S_1 H_{d/2}$ and $S_2 H_{d/2}$. At some point, say $d = 8$, we directly compute the full 8×8 transform and subsample. We then proceed back "up" the recursion. In [1], it was shown that this algorithm has complexity $\mathcal{O}(d \log k)$. Recall that this means that the complexity is asymptotically bounded above by a constant times $d \log k$, not that the complexity is asymptotically a constant times $d \log k$.

We call a direct, natural implementation of either "algorithm" (1) or (2) a reference implementation, as we will compare the runtime of any optimized implementation to the reference implementation. In Figures 3, 4, 5, we plot the runtimes of various implementations of the SRHT. We implemented a reference simple SRHT and smarter SRHT in MATLAB and Julia. We also wrote an optimized C implementation that uses SSE2 intrinsics, which were quite pleasant to use and gave about a 20% improvement in runtime. Perhaps writing a fully iterative implementation instead of a stack-abusing recursive implementation would improve the runtime; we speculate that this wouldn't be too difficult, but would certainly introduce a countably infinite number of bugs into our code. Our implementations, as well as our dual BCH sampling code, are available online at `https://github.com/jamesfolberth/fast_methods_big_data_project`.

In each figure, we compute compute the transform for a few values of the transform length d and vary the number of samples k. The simple SRHT computes the full length d transform and then subsamples, so its runtime is nominally constant with respect to the amount of subsampling k. The "better" SRHT, which is an implementation of (2), doesn't grow linearly with $\log k$ (and so is not strictly $\sim Cd \log k$), but does appear to be $\mathcal{O}(d \log k)$. At any rate, for k at least 10 times smaller than d, the smarter SRHT is faster than the simple SRHT. If k is very large, then the smarter SRHT has too much overhead work, and the simple SRHT is faster, which is reasonable.

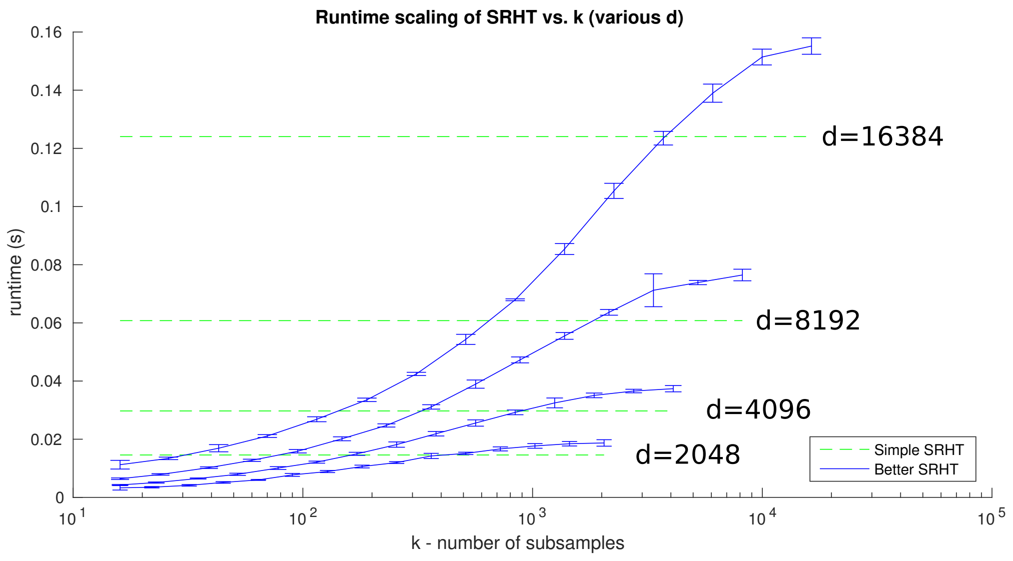

Figure 3: MATLAB reference implementation of the naïve SRHT and the smarter SRHT (2).

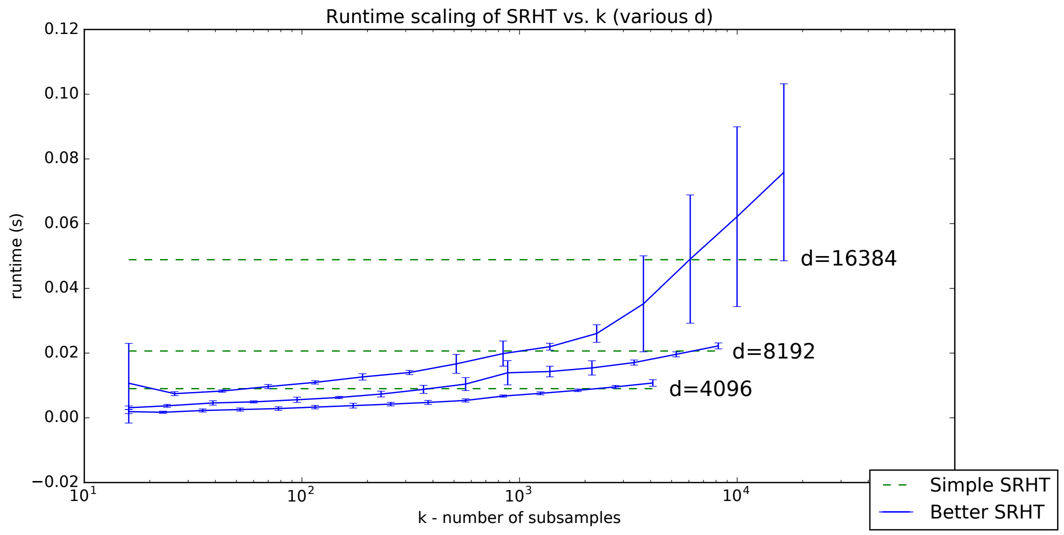

Figure 4: Julia reference implementation of the naïve SRHT and the smarter SRHT (2).

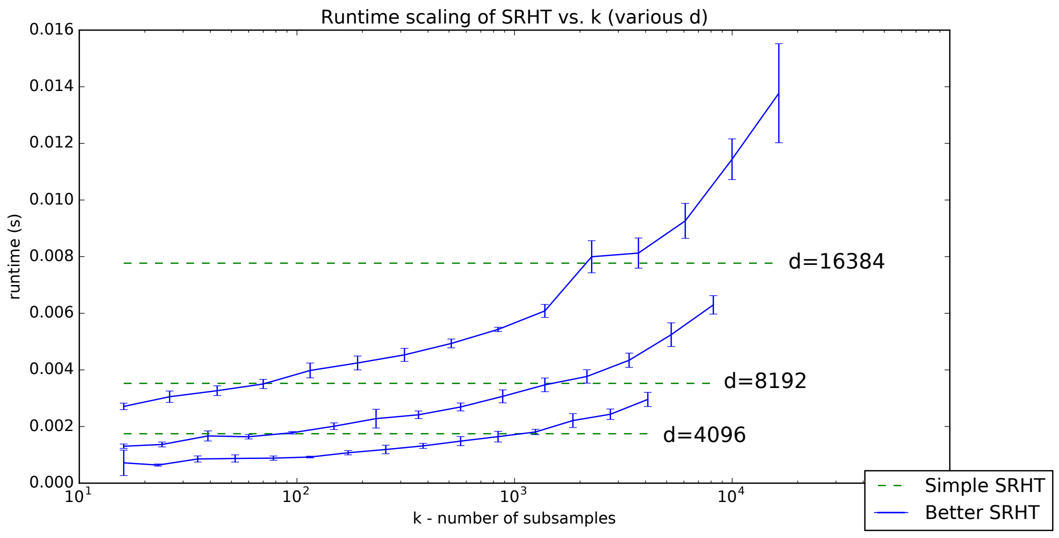

Figure 5: Optimized C implementations of the full Hadamard transform (1) and smarter SRHT (2). The optimized full Hadamard transform was used for the “Simple SRHT”.

7 Discussion

Ubaru et al. experimented with using subsampled dual BCH code matrices as sampling matrices for the RSVD [6]. In addition to having nice theoretical properties, dual BCH code matrices admit a fast multiplication operation via the SRHT and enjoy performance on par with Gaussian and SRFT sampling operators.

We also implemented Ailon and Liberty’s SRHT algorithm [1], which has complexity $\mathcal{O}(d \log k)$. To our knowledge, this is the first publicly available implementation of the SRHT that has this complexity, instead of the simple SRHT, which has complexity $\mathcal{O}(d \log d)$. Time permitting, we may clean up, document, and publish our SRHT code under an open source license for others to use. To achieve fast multiplication of dual BCH code matrices, one can do some offline preprocessing and use the SRHT for the multiplication.

References

[1] N. Ailon and E. Liberty, *Fast dimension reduction using rademacher series on dual bch codes*, Discrete & Computational Geometry, 42 (2009), pp. 615–630.

[2] K. L. Clarkson and D. P. Woodruff, *Numerical linear algebra in the streaming model*, in Proceedings of the forty-first annual ACM symposium on Theory of computing, ACM, 2009, pp. 205–214.

[3] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), p. 1.

[4] J. I. Hall, *Notes on coding theory*, FreeTechBooks. com, 2003.

[5] The MathWorks, Inc., *MATLAB and Communications System Toolbox Release R2015b*, Natick, Massachusetts, 2015.

[6] S. Ubaru, A. Mazumdar, and Y. Saad, *Low rank approximation using error correcting coding matrices*, in Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 2015, pp. 702–710.