12-2014

Numerical decoding, Johnson-Lindenstrauss transforms, and linear codes

Yue Mao
Clemson University, yuem@g.clemson.edu

Numerical decoding, Johnson-Lindenstrauss transforms, and linear codes

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Mathematical Sciences

---

by
Yue Mao
December 2014

Accepted by:
Dr. Shuhong Gao, Committee Chair
Dr. Taufiquar Khan
Dr. Elena Dimitrova
Dr. Bruce Gao

Abstract

Many computational problems are related to the model $y = Ax + e$, including compressive sensing, coding theory, dimensionality reduction, etc. The related algorithms are extremely useful in practical applications for high performance computing, for example, digital communications, biological imaging and data streaming, etc. This thesis studies two important problems. One problem is related to efficient decoding for Reed-Solomon codes over complex numbers. In this case, A and y are given, and the goal is to find an efficient stable algorithm to compute x. This is related to magnetic resonance imaging (MRI). The other problem is related to fast algorithms for projecting vectors in high dimensional spaces into low dimensional spaces so that their pairwise distances are nearly preserved. In this case, x is given (in a high dimensional space) and one needs to find a matrix A with some nice properties. This is called Johnson-Lindenstrauss transforms.

On decoding Reed-Solomon (RS) codes, the thesis first briefly describes the current algorithms for decoding RS codes over finite fields, including the Sudan-Guruswami list decoding. However, almost all existing decoding algorithms for finite fields are not applicable over complex numbers since they are not numerically stable. The thesis proposes a new numerical algorithm based on Gao's gcd decoding algorithm. The algorithm can successfully correct burst errors over complex numbers.

On Johnson-Lindenstrauss (JL) transforms, Kane and Nelson recently gave a sparse construction based on linear codes. The thesis shows how to use explicit codes to perform Kane and Nelson's fast JL transforms. The thesis also improves Kane and Nelson's bound for projections in dimensions that are useful in practice.

Acknowledgments

Firstly, I am truly grateful to my advisor Shuhong Gao. It is hard to express how thankful I am for his unwavering support over the years of my Ph.D. studies. I am extremly fortune to have an advisor who is always encouraging, inspiring and patient with me.

I would like to thank my committee, Prof. Taufiquar Khan, Prof. Elena Dimitrova and Prof. Bruce Gao for their advice and support. I also acknowledge Prof. Zhengfeng Yang, Prof. Zhenbing Zeng and Prof. Huahua Chen for their work and discussions with me on my research.

I would also like to express my appreciation to Mingfu Zhu, who helped me in the first year of my Ph.D. studies, to Michael Dowling and Frank Volny for their inspiring ideas, and to Fiona Knoll for her useful comments on my thesis.

Lastly, special thanks to my parents Yihui Fan and Aidong Mao, and my wife Qi Meng for their support and endless love.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Compressive sensing

Compressive sensing (CS) or *compressed sampling* [12][16] is an emerging area in signal processing and information theory. It is connected to areas such as coding theory, machine learning, statistics and high dimensional geometry, etc. Traditionally, one has to sample at full rate called the Nyquist rate in order to reconstruct a signal. However, signals in my practical applications can be compressed, i.e., they can be approximated by a sparse representation over certain basis. For example, in JPEG2000, a typical image can be compressed using wavelet basis. From the theory of compressive sensing, one can sample a signal at a much lower rate, then reconstruct the signal. Therefore compressive sensing is useful in applications where it is desirable to reduce the number of samples. One of the most famous applications of compressive sensing is the magnetic resonance imaging (MRI) in medical imaging [31].

More precisely, we need some notations and terminations to define compressive sensing. Let $x \in \mathbb{R}^n$ be a column vector. For $1 \leq p < \infty$, the p-norm is defined to be

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Also, define

$$\|x\|_\infty = \max |x_i|,$$

and

$$\|x\|_0 := \#\{1 \le i \le n : x_i \neq 0\}.$$

A vector x is called *s-sparse* if it has s or fewer non-zero entries, i.e.,

$$\|x\|_0 \le s$$

and the integer s is called the *sparsity* of the vector x. A vector x is *sparse* if its sparsity is much less than its length. In practice, most signals are not exactly sparse, but their coefficients decay rapidly, we call these signals *approximately sparse*.

Compressive sensing is modeled by the following equation

$$y = Ax + e \tag{1.1}$$

where $x \in \mathbb{R}^n$ is a column vector representing a sparse signal, A is an $m \times n$ matrix over $\mathbb{R}$ called a *sensing matrix* or *measurement matrix*, y is a column vector representing $m(\ll n)$ samples of x and e represents some random sampling error. The main computational problem is to find the sparse vector x when A and y are given (assuming e has certain probability distribution distribution such as Gaussian). The problem is also called *sparse recovery*.

Consider the noiseless case when there is no noise, then $e = 0$. The problem is to recover a sparse vector x from

$$y = Ax. \tag{1.2}$$

In compressive sensing, m is much smaller than n. Hence (1.2) may have infinite many solutions. Hence, we need to find a sparse solution x. The difficulty is in find the support of the sparse vector x, i.e., the indices of non-zero entries of x,

$$T = supp(x) := \{i : x_i \neq 0\}.$$

Once T is known, it is easy to compute x. The reason is as follows. Consider an $m \times n$ matrix

$A = [a_1, a_2, ..., a_n]$. Suppose $supp(x) = T = \{i_1, i_2, ..., i_s\}$ where $1 \leq i_1 < i_2 < \cdots i_s \leq n$. Let

$$A_T = [a_{i_1}, a_{i_2}, ..., a_{i_s}] \quad \text{and} \quad x_T = [x_{i_1}, x_{i_2}, ..., x_{i_s}]^T$$

be the submatrix of A and subvector of x with indices restricted on T, respectively. Then

$$y = Ax = A_T x_T.$$

If the columns of A_T are linearly independent, then x_T can be uniquely determined. In the noiseless case, the recovery problem can be formulated as

$$\min \|z\|_0 \quad s.t. \quad y = Az. \tag{1.3}$$

In general, the problem (1.3) is NP-hard [34]. One approach to relax the above ℓ_0 minimization problem to the following ℓ_1 minimization problem

$$\min \|z\|_1 \quad s.t. \quad y = Az. \tag{1.4}$$

Note that (1.4) can be solved by a linear programming, hence can be solved in polynomial time. The main problem is when an optimal solution of (1.4) is also an optimal solution of (1.3). A major breakthrough is due to Tao and Candes [11] and Donoho [16], where they claim (1.3) can be relexed to (1.4) if the matrix A has restricted isometry property (RIP).

Definition 1.1.1 *Let A be an $m \times n$ matrix. Let s be a integer and $0 < \delta < 1$, if for any s-sparse vector $x \in \mathbb{R}^n$,*

$$(1 - \delta) \|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \delta) \|x\|_2^2, \tag{1.5}$$

then we say the matrix A satisfies RIP with parameters s and δ, denoted as $RIP(s, \delta)$.

Basically, a matrix with $RIP(s, \delta)$ means that the transform $x \to Ax$ approximately preserves the 2-norm of any s-sparse vector x with a relative error up to δ.

In most applications, noise (or error) is inevitable. The general signal recovery problem can be

$$\min \|z\|_0 \quad s.t. \quad \|y - Az\|_2 \leq \epsilon \tag{1.6}$$

for some real number $\epsilon > 0$. Then if the has RIP, one can find an approximate solution from

$$\min \|z\|_1 \quad s.t. \quad \|y - Az\|_2 \leq \epsilon. \tag{1.7}$$

It is not known how to efficiently check the condition (1.1.1) for any given matrix. In the literature, many researchers give probabilistic construction of A, so that a random matrix A satisfies (1.1.1) with high probability. For example, consider the following random matrices.

- A Gaussian measurement matrix A is defined as an $m \times n$ random matrix whose entries are independently sampled from the normal distribution with mean zero and variance $\frac{1}{m}$. If $m = \mathcal{O}(s \log n)$, then A has RIP with high probability [12].
- A binary measurement matrix A is defined as an $m \times n$ random matrix whose entries are independently sampled from the symmetric Bernoulli distribution $P(A_{ij} = \pm 1/\sqrt{m}) = 1/2$. If $m = \mathcal{O}(s \log n)$, then A has RIP with high probability [12].
- A Fourier measurement matrix A of size $m \times n$ is obtained as follows. First, select m rows uniformly at random from a full $n \times n$ Fourier matrix. Then, normalize it so that each column has a unit 2-norm. If $m = \mathcal{O}(s \log^4 n)$, then A satisfies RIP with high probability [37].

There are properties used in compressive sensing other than RIP. For example, instead of studying the image of A, Cohen et al. consider the null space property(NSP) [13]

$$\mathcal{N}(A) := \{h \in \mathbb{R}^n : Ah = 0\}.$$

Definition 1.1.2 *(Null space property (NSP)). Suppose A is an $m \times n$ matrix. Then the matrix A satisfies $NSP(s)$ if there exist a constant $C > 0$ such that*

$$\|h_T\|_2 \leq C \frac{\|h_{T^c}\|_1}{\sqrt{s}} \tag{1.8}$$

holds for all $h \in \mathcal{N}(A)$ and for all T such that $|T| \leq s$.

The NSP means the vectors in the null space of A should not be too concentrated on a small set of indices. For example, if a vector $h \in \mathcal{N}(A)$ is exactly s-sparse, then there exist index set T such that

$h_{T^c} = 0$, thus $h_T = 0$ by (1.8). Therefore if a matrix A satisfies NSP then the only vector in $\mathcal{N}(A)$ is $h = 0$. While RIP is a sufficient condition for sparse recovery, the null space property (NSP) is both sufficient and necessary. However, the approach using NSP only works for the noiseless sparse recovery [14] .

1.1.2 Linear codes

In this section, we give some preliminaries in coding theory. We focus on linear codes. Let $\mathbb{F}$ be any field. An (n, k) linear code over $\mathbb{F}$ is any linear subspace of $\mathbb{F}^n$ of dimension k. The ratio k/n is called the (information) rate of the code.

An (n, k) linear code $C \subset \mathbb{F}^n$ can be generated in two ways. One way is to represent codewords in C as linear combinations of k linearly independent vectors in $\mathbb{F}^n$. More precisely, let $g_1, \ldots, g_k \in \mathbb{F}^n$ be linearly independent (column) vectors, and let

$$G = (g_1, g_2, \ldots, g_k)$$

be the $n \times k$ matrix over $\mathbb{F}$ whose columns are the vectors g_i. A codeword in C is of the form

$$c = Gx = x_1 g_1 + \cdots + x_k g_k$$

where $x \in \mathbb{F}^k$. The matrix G is called a *generator matrix*. The vector $x = (x_1, \ldots, x_k)^{\mathrm{T}}$ corresponds to a message of length k. The corresponding codeword $c = Gx$ is the encoding of x. Another way is to view codewords as solutions of a linear system. Let H be any $(n - k) \times n$ matrix over $\mathbb{F}$ of rank $n - k$. Define

$$C = \{c \in \mathbb{F}^n : Hc = 0\}. \tag{1.9}$$

Then C is a subspace of $\mathbb{F}^n$ of dimension k. The matrix H is called a *parity check matrix for* C.

Suppose any codeword $c = Gx$ is transmitted over a noisy channel where G is a generator matrix and x is a message vector. Suppose a vector

$$b = c + e = Gx + e \tag{1.10}$$

is received where e is some error vector. The decoding problem is to recover the vector x from b.

Alternatively, if we multiply H on both sides of (1.10), we get

$$y := Hb = He. \tag{1.11}$$

Then the decoding problem is to find the error vector e from y. Note e is usually sparse in practice, therefore (1.11) is a sparse recovery problem in compressive sensing.

Let $\mathbb{F}$ be any alphabet set (finite or infinite). For any vector $x = (x_1, x_2, \cdots, x_n) \in \mathbb{F}^n$, the Hamming weight (or ℓ_0-norm) of x, denoted by $\|x\|_0$, is the number of nonzero entries in x, that is,

$$\|x\|_0 = \#\{1 \le i \le n : x_i \neq 0\}.$$

The Hamming distance of any two vectors $x, y \in \mathbb{F}^n$ is defined as

$$d(x, y) = \|x - y\|_0,$$

i.e., $d(x, y)$ is the number of positions where the vectors x and y differ. This distance satisfies the usual triangular inequality:

$$d(x, y) \le d(x, z) + d(z, y).$$

The minimum distance of a code $C \subset \mathbb{F}^n$ is defined as the minimum distance between all pairs of distinct codewords, i.e.,

$$d(C) = \min_{x,y \in C, x \neq y} d(x, y).$$

For a linear code $C \subset \mathbb{F}^n$,

$$d(C) = \min_{x \in C, x \neq 0} \|x\|_0.$$

Let $C \subset \mathbb{F}^n$ be any code with minimum distance d. Then, for each $y \in \mathbb{F}^n$, there is at most one codeword $x \in C$ such that $d(x, y) \le \lfloor (d-1)/2 \rfloor$. It means that the code C can correct any $\lfloor (d-1)/2 \rfloor$ or less errors. So the minimum distance of a code is an important parameter measuring the error correction capability of a code. For any (n, k) linear code C on $\mathbb{F}$ with minimum distance d we have

$$d \le n - k + 1.$$

This bound is called the *singleton bound*. A linear code with $d = n - k + 1$ is called a *maximum*

distance separable (MDS) code.

1.2 Main computational problems

In this thesis, we work on two computational problems: decoding Reed-Solomon (RS) codes and Johnson-Lindenstrauss (JL) transforms. The two problems are both related to compressive sensing and linear codes. We give brief introductions of them in this section.

1.2.1 Decoding Reed-Solomon (RS) codes

RS codes introduced by Irving Reed and Golomb Solomon in 1960 [36] are widely used in digital communications with well-known applications such as CD/DVD, digital television, ADSL and wireless communications, etc. The RS codes are defined as follows. Suppose $\mathbb{F}$ is a any field and $\mathbb{F}$ contains n distinct elements $\alpha_1, \alpha_2, ..., \alpha_n$. Let $k < n$. A message is a vector

$$f = (f_0, f_1, ..., f_{k-1})^T$$

where each f_i is in $\mathbb{F}$. Define the message polynomial

$$f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1} \in \mathbb{F}[x].$$

Then the codeword corresponding to f is a column vector

$$c = [c_1, c_2, \ldots, c_n]^T = [f(\alpha_1), f(\alpha_2), ..., f(\alpha_n)]^T \in \mathbb{F}^n. \tag{1.12}$$

The set of all such codewords form an (n, k)-RS code C. The minimum distance of an RS code is $d = n - k + 1$, thus RS codes are MDS codes. A generator matrix of the RS code is the *Vandermonde matrix*

$$G = \begin{bmatrix} \alpha_1^0 & \alpha_1^1 & \ldots & \alpha_1^{k-1} \\ \alpha_2^0 & \alpha_2^1 & \ldots & \alpha_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_n^0 & \alpha_n^1 & \ldots & \alpha_n^{k-1} \end{bmatrix} \tag{1.13}$$

and the encoding (1.13) is the same as

$$c = G \cdot f.$$

Suppose the codeword c is transmitted over a noisy channel and a corrupted vector

$$b = c + e$$

is received where $e = (e_1, e_2, ..., e_n)^T$ represents errors. The decoding problem for RS codes is to correct errors in e (or recover the message f) from the received vector b.

In the definition of RS codes, the field $\mathbb{F}$ can be also either finite field $\mathbb{F}_q$ where q is a prime power, or the field $\mathbb{C}$ of complex field. Let ω be an element of $\mathbb{F}$ of order n, i.e., $\omega^n = 1$ but $\omega^t \neq 1$ for any $1 \leq k < n$. For example, when $\mathbb{F} = \mathbb{C}$, we can let $\omega = e^{2\pi i/n}$ where $i = \sqrt{-1}$. Let

$$\alpha_j = \omega^{j-1}, j = 1, 2, \ldots, n$$

be the fixed points for evaluation. Then the generator matrix is a partial discrete Fourier matrix

$$G = \begin{bmatrix} \omega^{0\cdot 0} & \omega^{0\cdot 1} & \ldots & \omega^{0\cdot(k-1)} \\ \omega^{1\cdot 0} & \omega^{1\cdot 1} & \ldots & \omega^{1\cdot(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(n-1)\cdot 0} & \omega^{(n-1)\cdot 1} & \ldots & \omega^{(n-1)\cdot(k-1)} \end{bmatrix}$$

and the RS codeword is

$$c = G \cdot f.$$

Equivalently, consider the full discrete Fourier matrix

$$G' = \begin{bmatrix} \omega^{0\cdot 0} & \omega^{0\cdot 1} & \ldots & \omega^{0\cdot(n-1)} \\ \omega^{1\cdot 0} & \omega^{1\cdot 1} & \ldots & \omega^{1\cdot(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(n-1)\cdot 0} & \omega^{(n-1)\cdot 1} & \ldots & \omega^{(n-1)\cdot(n-1)} \end{bmatrix}$$

and a longer message vector

$$f' = [f_1, f_2, ..., f_k, 0, ..., 0]^T$$

with $n-k$ zeros appended. Then the RS codeword is

$$c = G' \cdot f'$$

We see that the encoding of such RS codes can be computed efficiently via fast Fourier transform (FFT).

Alternatively, the RS codes can be also represented using the parity check matrix. It is easy to check the following partial discrete Fourier matrix formed by consecutive rows of the full discrete Fourier matrix G'

$$H = \begin{bmatrix} \omega^{0\cdot k} & \omega^{1\cdot k} & \dots & \omega^{(n-1)\cdot k} \\ \omega^{0\cdot (k+1)} & \omega^{1\cdot (k+1)} & \dots & \omega^{(n-1)\cdot (k+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{0\cdot (n-1)} & \omega^{1\cdot (n-1)} & \dots & \omega^{(n-1)\cdot (n-1)} \end{bmatrix} \tag{1.14}$$

is the parity check matrix for RS codes since

$$HG = 0, \quad \text{and} \quad rank(H) + rank(G) = n.$$

Then for any RS codeword $c \in C$ we have

$$Hc = 0.$$

Suppose the received vector is

$$b = c + e \tag{1.15}$$

and if we multiply the parity check matrix H on both sides of (1.15), we get

$$y := Hb = He \tag{1.16}$$

where the vector y is called *syndromes*. We see the problem of decoding RS codes becomes recovering the error vector e from y. Since there are only few errors during transmission, the number errors $\|e\|_0$ is usually small. Therefore (1.16) has the same form as the sparse recovery problem (1.1) in compressive sensing. In compressive sensing, a general problem is to design a measurement matrix and give an efficient algorithm to recover the sparse vector.

1.2.2 Johnson-Lindenstrauss (JL) transforms

The Johnson-Lindenstrauss (JL) transform is a key tool for dimension reduction techniques and it is useful in approximate nearest neighbor (ANN) search [2], compressive sensing [16, 11], computational geometry [24], etc. In many applications, one is given a collection of vectors in a vector space of high dimension, and the JL transform is designed to embed these vectors into a space of lower dimension such that all distances between pairs of vectors are nearly preserved. The following lemma was first proved by Johnson and Lindstauss [26] in 1984.

Lemma 1.2.1 *([26]). Let n and p be any positive integers, $0 < \epsilon < \frac{1}{2}$ and $m = \mathcal{O}(\epsilon^{-2} \log p)$. For any vectors $x_1, x_2, ..., x_p$ in $\mathbb{R}^n$, there exists a transform matrix $A \in \mathbb{R}^{m \times n}$ such that*

$$(1-\epsilon) \|x_i - x_j\|_2^2 \leq \|Ax_i - Ax_j\|_2^2 \leq (1+\epsilon) \|x_i - x_j\|_2^2 \tag{1.17}$$

for all pairs of i and j.

Random matrices can be used to prove the above Lemma 1.2.1. The following 1.2.2 is a "randomized" verision of the JL lemma.

Lemma 1.2.2 *([26]). Let n be any positive integer, $0 < \epsilon, \delta < \frac{1}{2}$ and $m = \mathcal{O}(\epsilon^{-2} \log \frac{1}{\delta})$. Then, there exists a probabilistic distribution on $A \in \mathbb{R}^{m \times n}$, such that for any vector $x \in \mathbb{R}^n$,*

$$P\left[(1-\epsilon) \|x\|_2^2 \leq \|Ax\|_2^2 \leq (1+\epsilon) \|x\|_2^2\right] \geq 1 - \delta. \tag{1.18}$$

Both Lemma 1.2.1 and Lemma 1.2.2 are referred as JL lemmas in the literature.

One may notice that (1.17) looks similar to the RIP (Definition 1.1.1) in compressive sensing. Indeed, if the probability distribution of the matrix A satisfies the following *concentration inequality*, then a random matrix from this distribution satisfies RIP with high probability,

$$P\left[(1-\epsilon) \|x\|_2^2 \leq \|Ax\|_2^2 \leq (1+\epsilon) \|x\|_2^2\right] \geq 1 - e^{-nc(\epsilon)} \tag{1.19}$$

where $c(\epsilon)$ is a constant depending only on ϵ, and the probability distribution is over the random matrix A [5]. Conversely, if a matrix Φ satisfies RIP, then the matrix $A = \Phi D$ satisfies the condition (1.17) for a set of vectors $\{x_1, x_,..., x_p\}$ with high probability, where D is a diagonal matrix with i.i.d. uniform ± 1 signs[30].

The major open problem is to give explicit construction of A so that (1.17) holds and computing Ax is fast. There is a vast literature on this topic. We give a brief survey as follows.

Indyk and Motwani [25] uses i.i.d entries to construction the matrix A. The $m \times n$ matrix A with i.i.d. entries

$$A_{ij} \sim \mathcal{N}(0, \frac{1}{m})$$

satisfies the condition (1.18) with high probability.

In 2003, Achlioptas uses binary entries to construct A [1]. The $m \times n$ matrix A with i.i.d. entries

$$A_{ij} := \begin{cases} +\frac{1}{\sqrt{m}}, & \text{with probability } \frac{1}{2}; \\ -\frac{1}{\sqrt{m}}, & \text{with probability } \frac{1}{2}. \end{cases}$$

satisfies the condition (1.18) with high probability. The motivation of this construction is to make random projections easier to use in practice. Indeed, flipping coins is much easier than generating numbers from a Gaussian distribution. In addition, Achlioptas shows the matrix A can be made relatetively sparse. The $m \times n$ matrix A with i.i.d. entries

$$A_{ij} := \begin{cases} +\sqrt{\frac{3}{m}}, & \text{with probability } \frac{1}{6}; \\ 0, & \text{with probability } \frac{2}{3}; \\ -\sqrt{\frac{3}{m}}, & \text{with probability } \frac{1}{6}; \end{cases}$$

satisfies the condition (1.18) with high probability. From this construction, 2/3-fraction of A's entries are zero. The speed of computing Ax with a sparse matrix A is much faster comparing his dense construction.

The speed of computing Ax in [1] is not optimal. Then Ailon and Chazelle give a fast JL transforms (FJLT) [2] using sparse matrix that improves [1]. Let $q = \Theta(\epsilon^3 \log^3 pn^{-1}) < 1$. Define D as a $n \times n$ diagonal matrix whose diagonal entries are i.i.d. over $\{\pm 1\}$ uniformly. Define H as either a $n \times n$ Walsh Hadamard matrix or a $n \times n$ discrete Fourier matrix. Define P as a $m \times n$ matrix with entries

$$P_{ij} = \begin{cases} \sim \mathcal{N}(0; q^{-1}), & \text{with probability } q; \\ 0, & \text{with probability } 1 - q. \end{cases}$$

Then $A = PHD$ satisfies the condition (1.17) for a set of vectors $\{x_1, x_2, ..., x_p\}$.

In the meantime, another sparse construction is given by Matousek [33]. Let $\epsilon \in (0, 1/2)$ and $\eta \in [1/\sqrt{n}, 1]$. Let $q = C_0\eta^2 \log(\frac{1}{\delta})$ for sufficiently large constant C_0. Let $m = C_1 \log(\frac{1}{\delta})/\epsilon^2$ for a sufficiently large C_1. Define A as a $m \times n$ matrix with entries

$$A_{ij} = \begin{cases} +\frac{1}{\sqrt{q}}, & \text{with probability } q/2; \\ -\frac{1}{\sqrt{q}}, & \text{with probability } q/2; \\ 0, & \text{with probability } 1-q. \end{cases}$$

Then the matrix A satisfies the condition (1.18) with high probability.

Later on, Ailon and Chazelle gives a further improvement [3] by using BCH codes. Let $\delta > 0$ be some arbitrarily small constant and let n be any integer and $m \leq n^{1/2-\delta}$. Let B_0 be a $m \times n$ normalized binary dual BCH codes of designed distance 5 (detailed construction can be found in [32]). Let B be normalized from B so that each column of B has unit 2-norm. Let $D, D^{(1)}, D^{(2)}, ...$ be a sequence of random ± 1 diagonal matrices. Let H be the $n \times d$ Walsh-Hadamard matrix. Then the matrix

$$A = (BD)(HD^{(1)})(HD^{(2)}) \cdots (HD^{(r)})$$

for $r = \lceil \frac{1}{2\delta} \rceil$ satisfiesthe condition (1.18) with high probability.

Chapter 2

Numerical decoding of RS codes

In Section 2.1, we first review sparse recovery approaches in compressive sensing for a general measurement matrix.

Then in Section 2.2, we study an equivalent problem called *sparse polynomial interpolation*. A sparse polynomial is a polynomial given the number of terms is much smaller than the polynomial degree. We show the two problems are equivalent if we use certain evaluation points, and we review the major approaches for sparse polynomial interpolation.

In Section 2.3, we review tradition algorithms for decoding RS codes over finite fields. We first review Gao's algorithm [17] for RS codes with the number of errors below half the minimum distance, and then Guruwami-Sudan's algorithm [39, 22] for RS codes with the number of errors beyond half the minimum distance.

In Section 2.4, we propose a numerical algorithm to decode RS codes over complex numbers. We give our theoretical proofs, numerical algorithm and experimental results.

2.1 Sparse recovery algorithms for general matrices

There are many papers studying efficient algorithms on sparse reovery. In this section, we give a brief survey of some of the main algorithms.

Definition 2.1.1 *(Best s-sparse approximation). We sort the entries of the vector* $x = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ *(in absolute values) as* $|x_{i_1}| \geq |x_{i_2}| \geq \cdots \geq |x_{i_n}|$. *Let* $T = \{i_1, i_2, ..., i_s\}$. *The best s-sparse ap-*

proximation of x is a vector $x^* \in \mathbb{R}^n$ with

$$x_j^* = \begin{cases} x_j, & j \in T \\ 0, & j \notin T. \end{cases}$$

Theorem 2.1.2 *([9]). Suppose that a matrix A satisfies $RIP(2s, \delta)$ with $\delta < \sqrt{2}-1$ and we obtain measurements of the form $y = Ax$ from any signal x. Then the solution $\hat{x}$ to*

$$\min \|z\|_1 \quad s.t. \quad y = Az \tag{2.1}$$

obeys

$$\|\hat{x} - x\|_2 \leq c_1 \frac{\|x - x^*\|_1}{\sqrt{s}}, \tag{2.2}$$

where x^* is the best s-sparse approximation of x and $c_1 = 2\frac{1-(1-\sqrt{2})\delta}{1-(1+\sqrt{2})\delta}$.

Theorem 2.1.3 *(Theorem 1.2 of [9]). Suppose that a matrix A satisfies $RIP(2s, \delta)$ with $\delta < \sqrt{2}-1$ and we obtain measurements of the form $y = Ax + e$ from any signal x where the error $\|e\|_2 \leq \epsilon$. Then the solution $\hat{x}$ to*

$$\min \|z\|_1 \quad s.t. \quad \|y - Az\|_2 < \epsilon \tag{2.3}$$

obeys

$$\|\hat{x} - x\|_2 \leq c_1 \frac{\|x - x^*\|_1}{\sqrt{s}} + c_2\epsilon, \tag{2.4}$$

where x^* is the best s-sparse approximation of x, $c_1 = 2\frac{1-(1-\sqrt{2})\delta}{1-(\sqrt{2}+1)\delta}$ and $c_2 = 4\frac{\sqrt{1+\delta}}{1-(1+\sqrt{2})\delta}$.

Theorem 2.1.4 *([10]). Suppose that a matrix A satisfies $RIP(2s, \delta)$ with $\delta < \sqrt{2}-1$ and we obtain measurements of the form $y = Ax + e$ from any signal x where the error $\left\|A^T e\right\|_\infty \leq \lambda$. Then the solution $\hat{x}$ to*

$$\min \|z\|_1 \quad s.t. \quad \left\|A^T(y - Az)\right\|_\infty \leq \lambda \tag{2.5}$$

obeys

$$\|\hat{x} - x\|_2 \leq c_1 \frac{\|x - x^*\|_1}{\sqrt{s}} + c_3\sqrt{s}\lambda, \tag{2.6}$$

where x^* is the best s-sparse approximation of x, $c_1 = 2\frac{1-(1-\sqrt{2})\delta}{1-(\sqrt{2}+1)\delta}$ and $c_3 = \frac{4\sqrt{2}}{1-(1+\sqrt{2})\delta}$.

In compressive sensing, an alternative to l_1-minimization algorithms is called greedy algorithms. Greedy algorithms find the the non-zero entries by iterations. The Algorithm 1 called compressive sampling matching pursuit(CoSaMP) [35] is the first algorithm to use RIP and can stably solve sparse recovery problems.

Algorithm 1 Compressive Sampling Matching Pursuit(CoSaMP)

Input: A, y, s
Output: A s-sparse approximation $\hat{x}$ of the target signal x
Initialization:
1: Set $T^0 = \emptyset$
2: Set $y^0 = y$
Iterations: During iteration ℓ, do
1: $\tilde{T}^\ell = T^{\ell-1} \cup \{2s \text{ indices of largest magnitude entries of } A^* y^{\ell-1}\}$
2:$x^\ell = A^\dagger_{\tilde{T}^\ell} y$
3:$T^\ell = \{s \text{ indices of largest magnitude entries of } x^\ell\}$
4:$y^\ell = y - A_{T^\ell} \tilde{x}_{T^\ell}$
5:if $\left\|y^\ell\right\|_2 = 0$ then
6: return $\hat{x}$ defined by $\hat{x}_{\{1,\dots,n\}-T^\ell} = 0$ and $\hat{x}_{T^\ell} = x^\ell_{T^\ell}$
7:else
8: Perform iteration $\ell + 1$
9:end if

Theorem 2.1.5 *([35]). Suppose that a matrix A satisfies $RIP(4s, \delta)$ with $\delta < 0.1$ and we obtain measurements of the form $y = Ax + e$ from any signal x where the error $\|e\|_2 \leq \epsilon$. Then at iteration ℓ, the CoSaMP algorithm recovers an approximate solution x^ℓ that obeys*

$$\left\|x^\ell - x\right\|_2 \leq 2^{-\ell} \|x^*\|_2 + 20 \left(\|x - x^*\|_2 + \frac{\|x - x^*\|_1}{\sqrt{s}} + \epsilon \right). \tag{2.7}$$

where x^* is the best s-sparse approximation of x.

The Algorithm 2 called iterative hard thresholding (IHT) [8] improves CoSaMP and it is one of the simplest greedy algorithms in compressive sensing.

Theorem 2.1.6 *([8]). Suppose that a matrix A satisfies $RIP(3s, \delta)$ with $\delta < \frac{1}{\sqrt{32}}$ and we obtain measurements of the form $y = Ax + e$ from any signal x where the error $\|e\|_2 \leq \epsilon$. Then at iteration ℓ, the IHT algorithm recovers an approximate solution x^ℓ that obeys*

$$\left\|x^\ell - x\right\|_2 \leq 2^{-\ell} \|x^*\|_2 + 6 \left(\|x - x^*\|_2 + \frac{\|x - x^*\|_1}{\sqrt{s}} + \epsilon \right). \tag{2.8}$$

Algorithm 2 Iterative Hard Thresholding (IHT)

Input: $A, y, w \in (0,1), s$
Output: A s-sparse approximation $\hat{x}$ of the target signal x
Initialization:
1: Set $x^0 = 0$
2: Set $T^0 = \emptyset$
3: Set $y^0 = y$
Iteration: During iteration ℓ, do
1:$x^\ell = x^{\ell-1}_{T^{\ell-1}} + wA^* y^{\ell-1}$
2:$T^\ell = \{s \text{ indices of largest magnitude entries of } x^\ell\}$
3:$y^\ell = y - A_{T^\ell} x^\ell_{T^\ell}$
4:if $\left\|y^\ell\right\|_2 = 0$ then
5: return $\hat{x}$ defined by $\hat{x}_{\{1,\ldots,n\}-T^\ell} = 0$ and $\hat{x}_{T^\ell} = x^\ell_{T^\ell}$
6:else
7: Perform iteration $\ell + 1$
8:end if

In this chapter, we study the problem of decoding RS codes. We first recall the definition of RS codes here. Suppose $\mathbb{F}$ is a any field. A message is a vector

$$f = (f_0, f_1, ..., f_{k-1})^T$$

where each f_i is an element in $\mathbb{F}$. Form the message polynomial

$$f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1} \in \mathbb{F}[x]$$

and evaluate $f(x)$ at $n(> k)$ distinct points $\alpha_1, \alpha_2, ..., \alpha_n \in \mathbb{F}$. Then an RS codeword is

$$c = [c_1, c_2, \ldots, c_n]^T = [f(\alpha_1), f(\alpha_2), ..., f(\alpha_n)]^T.$$

Suppose the codeword c is transmitted over a noisy channel and a corrupted vector

$$b = c + e$$

is received where $e = (e_1, e_2, ..., e_n)^T$ represents errors. The problem of decoding RS codes is to correct errors in e (or recover the message f) from the received vector b.

2.2 Sparse polynomial interpolation

Interpolation of a sparse polynomial from its values is a classic problem in symbolic computing and it recently becomes interesting in numeric computing. Let $\mathbb{F}$ be any field. The problem of interpolating an univariate sparse polynomial is defined as follows. Suppose we have an unknown s-sparse univariate polynomial $f \in \mathbb{F}[x]$, that is

$$f(x) = \sum_{1 \le j \le s} c_j x^{d_j}.$$

where the coefficients $c_1, .., c_s \in \mathbb{F}$, the exponents $0 \le d_1 < d_2 < \cdots < d_s < n$. Assume $s \ll n$, thus f is a sparse polynomial. Evaluating at our own choice of $\nu_1, \nu_2, ..., \nu_m \in \mathbb{C}$ and get $y_1 = f(\nu_1), y_2 = f(\nu_2), ..., y_m = f(\nu_m)$. The problem is that given $y_1, y_2, ..., y_m$, determine the coefficients $c_1, .., c_s \in \mathbb{F}$ and the exponents $0 \le d_1 < d_2 < \cdots < d_s < n$. The values of $f(x)$ evaluated at $\nu_1, ..., \nu_m$ can be represented as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \nu_1^{d_1} & \nu_1^{d_2} & \dots & \nu_1^{d_s} \\ \nu_2^{d_1} & \nu_2^{d_2} & \dots & \nu_2^{d_s} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_m^{d_1} & \nu_m^{d_2} & \dots & \nu_m^{d_s} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{bmatrix}. \tag{2.9}$$

Define an n-dimensional column vector $x = (x_0, x_1, ..., x_{n-1})^T$ such that

$$x_i = \begin{cases} c_j, & i = d_j, 1 \le j \le s; \\ 0, & \text{otherwise.} \end{cases} \tag{2.10}$$

Then (2.9) becomes

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \nu_1^0 & \nu_1^1 & \dots & \nu_1^{n-1} \\ \nu_2^0 & \nu_2^1 & \dots & \nu_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_m^0 & \nu_m^1 & \dots & \nu_m^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.11}$$

where the above $m \times n$ matrix is a Vandermonde matrix. We see (2.11) is a sparse recovery problem defined in Section 1.1.1 where the measurement matrix is a Vandermonde matrix.

For the rest part of this section, we always let $\mathbb{F}$ be any finite field (or $\mathbb{C}$) and let ω be a primitive element (or a primitive root of unity $e^{\frac{2\pi i}{n}}, i = \sqrt{-1}$).

The problem decoding an (n,k)-RS codes C evaluated at $\alpha_1 = \omega^0, \alpha_2 = \omega^0, ..., \alpha_n = \omega^{n-1}$ is identical to the problem of interpolating a degree-n sparse polynomial over $\mathbb{F}$ evaluated at $\nu_1 = \omega^k, \nu_2 = \omega^{k+1}, ..., \nu_{n-k} = \omega^{n-1}$. Indeed by (1.14), for any RS codeword $c \in C$ we have $Hc = 0$ where

$$H = \begin{bmatrix} \omega^{0\cdot k} & \omega^{1\cdot k} & \dots & \omega^{(n-1)\cdot k} \\ \omega^{0\cdot (k+1)} & \omega^{1\cdot (k+1)} & \dots & \omega^{(n-1)\cdot (k+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{0\cdot (n-1)} & \omega^{1\cdot (n-1)} & \dots & \omega^{(n-1)\cdot (n-1)} \end{bmatrix}.$$

is a parity check matrix. Suppose the received codeword is $b = c + x$ where x is a sparse error vector, then

$$y = Hb = Hx. \tag{2.12}$$

is the vector of syndromes. If we let x be defined defined in (2.10) and let $\nu_1 = \omega^k, \nu_2 = \omega^{k+1}, ..., \nu_{n-k} = \omega^{n-1}$, then (2.12) is identical to (2.11). Therefore, with such particular evaluations, the problem of decoding RS codes can be converted to a problem of interpolating a sparse polynomial .

Early algorithms usually assume exact arithmetic, e.g. [6, 21, 27, 28]. $m = 2s$ are required, as there are $2s$ unknowns: $d_1, d_2, ..., d_s$ and $c_1, c_2, .., c_s$. The numerical version of the sparse polynomial interpolation problem has become interesting since the numerical algorithm [19] given by Giesbrecht, Labahn and Lee.

Before we review approaches of sparse polynomial interpolation, we give some remarks on the following multivariate sparse polynomial interpolation. Let $\mathbb{F}$ be any field. Suppose we have an unknown s-sparse multivariate polynomial $f \in \mathbb{F}[x_1, x_2, ..., x_\ell]$, that is

$$f(x_1, x_2, ..., x_\ell) = \sum_{1 \le j \le s} c_j x_1^{d_{j_1}} x_2^{d_{j_2}} \cdots x_\ell^{d_{j_\ell}}.$$

where the coefficients $c_1, .., c_s \in \mathbb{F}$ and the exponents $0 \le d_{j_1} < d_{j_2} < \cdots < d_{j_\ell} < n$ for $1 \le j \le s$. Assume $s \ll n$ so that f is a sparse polynomial. Evaluating at our own choice of $\nu_1, \nu_2, ..., \nu_m \in \mathbb{F}^\ell$ and get $y_1 = f(\nu_1), y_2 = f(\nu_2), ..., y_m = f(\nu_m)$. The problem is that given $y_1, y_2, ..., y_m$ determine the coefficients $c_1, .., c_s \in \mathbb{F}$ and the exponents $0 \le d_{j_1} < d_{j_2} < ... < d_{j_\ell} < n$ for $1 \le j \le s$. By the discussion in [19], this problem can be converted from the multivariate case to the univariate

case by using the Chinese reminder theorem. Thus we focus on the interpolating univariate sparse polynomials.

2.2.1 Prony's method

Prony's method uses $m = 2s$ points to evaluate the sparse polynomial $f(x)$, say $\nu_i = \omega^{i-1}, 1 \leq i \leq 2s$. Then

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{2s} \end{bmatrix} = \begin{bmatrix} \omega^{0\cdot d_1} & \omega^{0\cdot d_2} & \dots & \omega^{0\cdot d_s} \\ \omega^{1\cdot d_1} & \omega^{1\cdot d_2} & \dots & \omega^{1\cdot d_s} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(2s-1)\cdot d_1} & \omega^{(2s-1)\cdot d_2} & \dots & \omega^{(2s-1)\cdot d_s} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{bmatrix}.$$

Define the polynomial $\Lambda(z) = \prod_{j=1}^{s}(z - \omega^{d_j}) = z^s + \lambda_{s-1}z^{s-1} + \cdots + \lambda_1 z + \lambda_0$, then its coefficients $\lambda_0, \lambda_1, ..., \lambda_{s-1}$ satisfy

$$\begin{bmatrix} y_0 & y_1 & \dots & y_{s-1} \\ y_1 & y_2 & \dots & y_s \\ \vdots & \vdots & \ddots & \vdots \\ y_{s-1} & y_s & \dots & y_{2s-2} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{s-1} \end{bmatrix} = - \begin{bmatrix} y_s \\ y_{s+1} \\ \vdots \\ y_{2s-1} \end{bmatrix}$$

where the matrix in the above equation is a Hankel matrix. proof...

The polynomial $\Lambda(z)$ is called the *error locator polynomial*. By the lemma, since the roots of Λ are $\omega^{b_1}, ..., \omega^{b_s}$, the exponents $0 \leq d_1 < ... < d_s < n$ can be determined by taking logarithms. In [19], the author commented that the problem of finding roots of a high degree polynomial is numerically unstable. However, we want to point out that an alternative way is to plug all possible solutions $\omega^0, \omega^1, ..., \omega^{n-1}$ in $f(x)$ and pick s smallest ones from $|f(\omega^i)|, 0 \leq i \leq n-1$.

2.2.2 Generalized eigenvalues method

Given $m \geq 2s$ evaluations $y_1, y_2, ..., y_m$, define two Hankel matrices as follows

$$H_0 = \begin{bmatrix} y_0 & y_1 & \cdots & y_{s-1} \\ y_1 & y_2 & \cdots & y_s \\ \vdots & \vdots & \ddots & \vdots \\ y_{m-s-1} & y_{m-s} & \cdots & y_{m-2} \end{bmatrix}, \quad H_1 = \begin{bmatrix} y_1 & y_2 & \cdots & y_s \\ y_2 & y_3 & \cdots & y_{s+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m-s} & y_{m-s+1} & \cdots & y_{m-1} \end{bmatrix}. \tag{2.13}$$

Consider the equation

$$H_1 v = \lambda H_0 v \tag{2.14}$$

where λ is a real number and v is a column vector. The equation (2.14) is called a genernalized eigenvalues problem and any solution pair (λ, v) is called generalized eigenvalues and generalized eigenvectors, respectively. Define

$$b_j = \omega^{d_j}, j = 1, 2, ..., s.$$

Then H_0 and H_1 have the following decomposition

$$H_0 = \begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_s \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{m-s-1} & b_2^{m-s-1} & \dots & b_s^{m-s-1} \end{bmatrix} \begin{bmatrix} c_1 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_s \end{bmatrix} \begin{bmatrix} 1 & b_1 & \dots & b_1^{m-s-1} \\ 1 & b_2 & \dots & b_2^{m-s-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_s & \dots & b_s^{m-s-1} \end{bmatrix}$$

$$H_1 = \begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_s \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{m-s-1} & \dots & \dots & b_s^{m-s-1} \end{bmatrix} \begin{bmatrix} b_1 c_1 & 0 & \dots & 0 \\ 0 & b_2 c_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_s c_s \end{bmatrix} \begin{bmatrix} 1 & b_1 & \dots & b_1^{m-s-1} \\ 1 & b_2 & \dots & b_2^{m-s-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_s & \dots & b_s^{m-s-1} \end{bmatrix}.$$

Then for any $j = 1, 2, ..., s$, b_j is a generalized eigenvalue to (2.14) as $H_1 - b_j H_0$ is singular.

Once the generalized eigenvalues $b_1, b_2, ..., b_s$ are recovered, the exponents can be computed as

$$d_j = \log_\omega b_j, \quad j = 1, 2, ..., s.$$

And computing $c_1, c_2, ..., c_s$ is trial from

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_s \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{m-1} & b_2^{m-1} & \dots & b_s^{m-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{bmatrix}$$

because $m > s$.

When $\mathbb{F}$ is a finite field, then the sparse polynomial can be recovered by above generalized eigenvalues method. When $\mathbb{F} = \mathbb{C}$, we need to discuss the stability. By [19], the stability of the generalized eigenvalues relies on the condition number of the Vandermonde matrix

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_s \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{m-s-1} & b_2^{m-s-1} & \dots & b_s^{m-s-1} \end{bmatrix}.$$

It is known that the Vandermonde matrix V is bad conditioned if the exponents $\{d_1, d_2, ..., d_s\}$ (mod n) are clustered. For example, if $d_i = 0$ and $d_j = n - 1$, then d_i and d_j are clustered in the sense of modulo n. So [19] suggests to evaluate $f(x)$ at random roots of unity $(\omega^r)^j, j = 0, 1, ..., m-1$ where $2 \leq r \leq n - 1$ is a random integer. The idea is to map the original exponents $d_1, d_2, ..., d_s$ to $rd_1, rd_2, ..., rd_s$, then $rd_1, rd_2, ..., rd_s \pmod{n}$ are not clustered with high probability.

Another important question is how to determine the sparsity s (recall that we assume s is known previously). Suppose from a s-sparse signal x we have measured $y_1, y_2, \dots, y_m$ which are not noisy. Let $1 \leq j \leq \lfloor \frac{m}{2} \rfloor$. We consider the rank of following square Hankel matrix,

$$H^{[j]} = \begin{bmatrix} y_1 & y_2 & \cdots & y_j \\ y_2 & y_3 & \cdots & y_{j+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_j & y_{j+1} & \cdots & y_{2j-1} \end{bmatrix}.$$

If $j \geq k$, then $H^{[j]}$ has the following decomposition

$$H^{[j]} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_s \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{j-1} & b_2^{j-1} & \dots & b_s^{j-1} \end{bmatrix} \begin{bmatrix} c_1 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_s \end{bmatrix} \begin{bmatrix} 1 & b_1 & \dots & b_1^{j-1} \\ 1 & b_2 & \dots & b_2^{j-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_s & \dots & b_s^{j-1} \end{bmatrix}$$

Hence, if $j \geq s$, the rank of $H^{[j]}$ is equal to the sparsity s. If $j < s$, then the rank of $H^{[j]}$ is less than s. Therefore, we determine the sparsity by computing the ranks of $H^{[1]}, H^{[2]}, \dots$ until they remain a finite integer s. Then integer s is the sparsity.

2.3 Decoding over finite fields

Suppose $\mathbb{F}_q$ is a finite filed with q elements where q is a prime power. A message is a vector $[f_0, f_1, ..., f_{k-1}]^T$ where each f_i is an element in $\mathbb{F}_q$. Form the message polynomial $f(x) = f_0 + f_1 x + \cdots + f_{k-1}x^{k-1} \in \mathbb{F}_q[x]$ and evaluate $f(x)$ at $n(> k)$ distinct points $\alpha_1, \alpha_2, ..., \alpha_n \in \mathbb{F}_q$. Then an RS codeword is

$$c = [c_1, c_2, \dots, c_n]^T = [f(\alpha_1), f(\alpha_2), ..., f(\alpha_n)]^T. \tag{2.15}$$

The set of all such codewords form an (n, k)-RS code C, and $|C| = q^k$ since different message polynomials give different codewords. Suppose a codeword c is sent over a noisy channel, and the received vector is $b = [b_1, \dots, b_n] \in \mathbb{F}_q^n$, where $b_i = c_i + e_i$ for some $e_i \in \mathbb{F}_q$, $1 \leq i \leq n$. d

2.3.1 Gao's GCD decoding algorithm

Let $t = \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{n-k}{2} \rfloor$, and assume that there are at most t errors. We show how to reconstruct $f(x)$.

Lemma 2.3.1 *Suppose we have a nonzero polynomial $u(x)y - v(x) \in \mathbb{F}[x, y]$, with $\deg u(x) \leq \frac{n-k}{2}$ and $\deg v(x) \leq \frac{n+k}{2} - 1$, that vanishes at all the points (a_i, b_i), $1 \leq i \leq n$, i.e., $u(a_i)b_i = v(a_i)$. Then*

$$f(x) = \frac{v(x)}{u(x)}.$$

To find the polynomial $u(x)y + v(x)$, we just need to solve the linear system

$$u(a_i)b_i - v(a_i) = 0, \quad 1 \le i \le n,$$

where $u(x) = u_0 + u_1x + \cdots + u_tx^t, v(x) = v_0 + v_1x + \cdots + v_mx^m \in \mathbb{F}[x]$ with $m = t + k - 1$. The coefficients $u_i, v_j \in \mathbb{F}$ are unknowns, and the linear system can be rewritten as

$$\begin{bmatrix} b_1 & b_1a_1 & \cdots & b_1a_1^t \\ b_2 & b_2a_2 & \cdots & b_2a_2^t \\ \vdots & \vdots & & \vdots \\ b_n & b_na_n & \cdots & b_na_n^t \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_t \end{bmatrix} = \begin{bmatrix} 1 & a_1 & \cdots & a_1^m \\ 1 & a_2 & \cdots & a_2^m \\ \vdots & \vdots & & \vdots \\ 1 & a_n & \cdots & a_n^m \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_m \end{bmatrix}. \tag{2.16}$$

This system has n equations and $m+t+2 \ge n-1$ unknowns $u_0, \ldots, u_t$ and $v_0, \ldots, v_m$. In principle, this linear system can be solved by Gaussian elimination using $\mathcal{O}(n^3)$ many operations in $\mathbb{F}$. There always is a nonzero solution. If the quotient $u(x)/v(x)$ is a polynomial of degree $\le k-1$, then it is the correct message polynomial $f(x)$; otherwise, the decoding fails.

When $\mathbb{F}$ is a finite field, this gives a cubic time algorithm for decoding the Reed-Solomon codes. In fact, there are faster algorithms to find the desired polynomial $u(x)y + v(x)$. Welch-Berlekamp [7] algorithm finds the polynomial $u(x)y+v(x)$ incrementally. The algorithm of Shiozaki [38], rediscovered by Gao [17], computes $u(x)y + v(x)$ by interpolation and partial gcd. These two algorithms use $\mathcal{O}(n^2)$ many operations in $\mathbb{F}$. By using fast algorithms for interpolation and gcd, one can get a decoding algorithm that uses only $\mathcal{O}(n\log^2(n)\log\log(n))$ many operations in $\mathbb{F}$. When $\mathbb{F}$ is the field of real or complex numbers, however, none of the algorithms mentioned above works due to their numerical instability. Also, the linear system is ill-conditioned, and it is still a challenge to find a good numerical algorithm for solving such a system.

We now present Gao's algorithm mentioned above. Recall that the extended Euclidean algorithm when applied to polynomials r_0, r_1 starts with $u_0 = 1, u_1 = 0, v_0 = 0, v_1 = 1$, and then performs a sequence of long divisions, producing

$$r_{i-1} = q_ir_i + r_{i+1}$$

Algorithm 3 Gao's Algorithm

Input: $b = (b_1, \ldots, b_n) \in \mathbb{F}^n$, $g_0(x) = \prod_{i=1}^{n}(x - a_i)$.
Output: $f(x)$ or failure.
Step 1: Interpolation. Find $g_1(x) \in \mathbb{F}[x]$ of degree $\leq n-1$ such that $g_1(a_i) = b_i$, $1 \leq i \leq n$.
Step 2: Partial gcd. Apply extended gcd method to $g_0(x)$ and $g_1(x)$ to find u, v, g so that

$$u(x)g_0(x) + v(x)g_1(x) = g(x)$$

and $\deg v(x) \leq \frac{n+k}{2}$
Step 3: Long division. Check if $\frac{g(x)}{v(x)}$ is a polynomial of degree $\leq k-1$. If it is, output this polynomial; otherwise output "failure".

with $\deg(r_{i+1}) < \deg(r_i)$, for $i = 1, 2, \ldots, m$, where $r_i \neq 0$ for $1 \leq i \leq m$, $r_{m+1} = 0$ and $\gcd(r_0, r_1) = r_m$, and also

$$u_{i+1} = u_{i-1} - q_i u_i, \qquad v_{i+1} = v_{i-1} - q_i v_i, \tag{2.17}$$

$1 \leq i \leq m$, with

$$r_i = u_i r_0 + v_i r_1, \tag{2.18}$$

for $0 \leq i \leq m+1$.

Lemma 2.3.2 *Let r_0 and r_1 be nonzero polynomials in $\mathbb{F}[x]$, and suppose the extended Euclidean algorithm performs the computation as described above. Then*

$$u_{m+1} = (-1)^{m+1}\frac{r_1}{r_m}, \qquad v_{m+1} = (-1)^m \frac{r_0}{r_m}.$$

Lemma 2.3.3 *Let $g_0(x) = w_0(x) \cdot r_0(x) + \epsilon_0(x)$ and $g_1(x) = w_0(x) \cdot r_1(x) + \epsilon_1(x)$, with $\gcd(r_0(x), r_1(x)) = 1$ and*

$$\deg(r_i(x)) \leq t, \qquad \deg(\epsilon_i(x)) \leq \ell, \qquad i = 1, 2.$$

Suppose d_0 satisfies

$$\deg(w_0(x)) \geq d_0 > \ell + t.$$

Apply the extended Euclidean algorithm to $g_0(x)$ and $g_1(x)$, and stop whenever the remainder $g(x)$ has degree $< d_0$. Suppose that at termination we have

$$u(x)g_0(x) + v(x)g_1(x) = g(x).$$

Then

$$u(x) = -\alpha r_1(x), \qquad v(x) = \alpha r_0(x)$$

for some nonzero $\alpha \in \mathbb{F}$.

Theorem 2.3.4 *Suppose the received vector* $b = [b_1, \ldots, b_n]$ *has distance at most* $\frac{d-1}{2}$ *from a codeword* $c = [c_1, \ldots, c_n]$ *defined by a polynomial* $f(x) \in \mathbb{F}[x]$. *Then Algorithm 3 returns* $f(x)$. *If* b *has distance greater than* $\frac{d-1}{2}$ *to every codeword, than the algorithm returns "failure".*

2.3.2 Guruswami-Sudan's list decoding algorithm

Suppose C is an (n, k, d) linear code over $\mathbb{F}_q$. Given $t > 0$ and a vector $b \in \mathbb{F}_q^n$, find all codewords $c \in C$ such that $d(c, b) \leq t$. For $t \leq \frac{d-1}{2}$, there is at most one codeword in the list.

For $t > \frac{d-1}{2}$, there exist $b \in \mathbb{F}_q^n$ such that the list contains at least two codewords. If t is not too large, we expect that the list is not too big, but this is not well understood even for special codes, including RS codes. The list-decoding problem is hard in general.

Recall that for an (n, k, d) RS code where $d = (n - k + 1)$, we can decode up to $d/2$ errors, where $d = n-k+1$. The basic idea is to find a polynomial $Q(x, y) = u(x)y-v(x)$ so that $Q(a_i, b_i) = 0$ for $1 \leq i \leq n$, where $a_1, \ldots, a_n \in \mathbb{F}$ are distinct and used to define the code, and $(b_1, \ldots, b_n) \in \mathbb{F}^n$ is the received vector. Then hope that

$$f(x) = \frac{v(x)}{u(x)}$$

is the polynomial used to define the codeword. This means that we hope $(y - f(x))|Q(x, y)$, which is equivalent to $Q(x, f(x)) = 0$ (for any polynomial $Q(x, y)$).

In 1997, Sudan realized that this method can be generalized to decode beyond $d/2$. Sudan considered a polynomial $Q(x, y)$ of smallest weighted degree so that

$$Q(a_i, b_i) = 0, \quad 1 \leq i \leq n \tag{2.19}$$

By weighted degree we mean that y has degree $k - 1$ and x has degree 1, hence $x^i y^j$ has degree $i + j(k - 1)$. If Q has degree D, then $Q(x, f(x))$ has degree $\leq D$.

For (2.19) to have a non-zero solution $Q(x, y)$ we need the number of coefficients of $Q(x, y)$ to be more than n. Also, we need $(y - f(x))|Q(x, y)$ if $f(x)$ agrees with b_i's for at least t i's. This leads to a good decoding algorithm if the rate $R = k/n$ is small ($R < 1/2$).

Guruswami and Sudan [22] improved Sudan's idea to make it work for RS codes of all rates (and to a greater decoding radius). They consider (2.19) and require that $Q(x, y)$ vanishes at (a_i, b_i) with higher multiplicity. Recall that for a univariate polynomial $g(x) \in \mathbb{F}[x]$ an element $a \in \mathbb{F}$ is called a root of multiplicity r of $g(x)$ if $(x-a)^r | g(x)$. In other words, the Taylor expansion of $g(x+a)$ is of the form

$$g(x + a) = c_r x^r + c_{r+1} x^{r+1} + \cdots$$

without terms of degree $< r$.

Similarly, we say a point $(a, b) \in \mathbb{F}^2$ is a zero of $Q(x, y)$ with multiplicity r if the Taylor expansion of $Q(x + a, b + y)$ has no terms of degree $< r$. Note that there are $(r + 1)r/2$ terms of degree $< r$.

Algorithm 4 Guruswami and Sudan Algorithm

Input: $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{F}^n$.
Output: all $f(x) \in \mathbb{F}[x]$ of degree $\leq k - 1$ so that $f(a_i) = b_i$ for at least $n - \sqrt{(k(k-1))}$ i's.
Step 1: Find a non-zero polynomial $Q(x, y)$ of weighted degree D (defined later) so that $Q(x, y)$ vanishes at (a_i, b_i) with multiplicity r (defined later) for $a \leq i \leq n$.
Step 2: Find all $f(x) \in \mathbb{F}[x]$ of degree $\leq k - 1$ so that $y - f(x)$ divides $Q(x, y)$ and $f(x)$ agrees with b_i's for at least t (defined later) positions.

Lemma 2.3.5 *If*

$$n\frac{(r + 1)r}{2} < \frac{D(D + 2)}{2(k - 1)} \tag{2.20}$$

then $Q(x, y)$ in Step 1 exists.

Proof. In Step 1, for each point (a_i, b_i) there are $r(r + 1)/2$ linear equations for the coefficients of $Q(x, y)$. Hence, the total number is equations is $nr(r + 1)/2$. The number of coefficients in $Q(x, y)$ is the number of pairs (i, j) so that $i + j(k - 1) \leq D$. Note that $j \leq \frac{D}{k-1}$. Let $\ell = \lfloor \frac{D}{k-1} \rfloor$. The number of such pairs is

$$\begin{aligned}
\sum_{j=0}^{\ell} (D - j(k - 1) + 1) &= D(\ell + 1) - (k - 1)\frac{(\ell + 1)\ell}{2} + (\ell + 1) \\
&= \frac{\ell + 1}{2} [2D - (k - 1)\ell + 2] \\
&\geq \frac{\ell + 1}{2}(D + 2) \\
&\geq \frac{D(D + 2)}{2(k - 1)}.
\end{aligned}$$

So we need

$$n\frac{(r+1)r}{2} < \frac{D(D+2)}{2(k-1)}$$

If we take $D = \lceil \sqrt{(k-1)nr(r+1)} \rceil$, then (2.20) is satisfied.

Lemma 2.3.6 *Suppose $Q(x,y)$ vanishes at (a_i, b_i) with multiplicity r. Then, for $f(x) \in \mathbb{F}[x]$ with $f(a_i) = b_i$, we have $(x-a_i)^r | Q(x, f(x))$.*

Proof. Consider the Taylor expansion of $Q(x,y)$ at (a_i, b_i),

$$Q(x,y) = \sum_{u+v \geq r} c_{uv}(x-a_i)^u (y-b_i)^v,$$

as $Q(x,y)$ vanishes at (a_i, b_i) with multiplicity r. Then

$$Q(x, f(x)) = \sum_{u+v \geq r} c_{uv}(x-a_i)^u (f(x)-b_i)^v.$$

Since $f(a_i) = b_i$, $(x-a_i) | (f(x)-b_i)$. Hence $(x-a_i)^r$ divides each term, and thus

$$(x-a_i)^r | Q(x, f(x)).$$

Corollary 2.3.7 *If $f(a_i) = b_i$ for t i's, then $Q(x, f(x))$ has at least tr zeros, counting multiplicities.*

Theorem 2.3.8 *Suppose $t > D/r$ and $Q(x,y)$ has $(1, k-1)$-weighted degree $\leq D$. Then for any $f(x) \in \mathbb{F}[x]$ of degree $k-1$ such that $f(a_i) = b_i$ for at least t of the indices i we have $(y - f(x)) | Q(x,y)$.*

Proof. Note that $Q(x, f(x))$ has degree $\leq D$ but has $tr > D$ zeros. So $Q(x, f(x)) = 0$, and hence $(y - f(x)) | Q(x,y)$.

So we have the following choice of parameters:

$$D = \lceil \sqrt{(k-1)nr(r+1)} \rceil$$

$$t = \left\lceil \sqrt{(k-1)n(1-\frac{1}{r})} \right\rceil$$

$$r = 2kn$$

Any $f(x)$ from the output list gives a codeword with Hamming distance at most t to $[b_1, \ldots, b_n]$. Note that

$$\frac{n-t}{n} = 1 - \sqrt{\frac{k-1}{n}}.$$

This is much better than $d/2$, which satisfies

$$\frac{d/2}{n} = \frac{n-k+1}{2n} = \frac{1}{2}\left(1 - \frac{k-1}{n}\right) = 1 - \frac{1}{2}\left(1 + \frac{k-1}{n}\right).$$

2.4 Decoding over complex numbers

In this section, we assume the integers n and k are even, and we define (n, k) RS codes over complex numebrs as follows. Let $q \geq 2$ be a prime power. Define $\mathcal{S} = \{s_0, s_1, \ldots, s_{q-1}\}$ as a set of q distinct complex numebrs. A message of length k is a vector $(f_0, f_1, ..., f_{k-1})^T$ where each f_i is an element in $\mathcal{S}$. Form the message polynomial $f(x) = f_0 + f_1 x + \cdots + f_{k-1}x^{k-1} \in \mathbb{C}[x]$ and evaluate $f(x)$ at $n(> k)$ distinct points $\alpha_1, \alpha_2, ..., \alpha_n \in \mathcal{S}$. Then an RS codeword is

$$c = [c_1, c_2, \ldots, c_n]^T = [f(\alpha_1), f(\alpha_2), ..., f(\alpha_n)]^T.$$

The set of all such codewords form an (n, k)-RS code C, and $|C| = q^k$ since different message polynomials give different codewords. The set $\mathcal{S}$ has to be chosen carefully and is often called a *constellation diagram*. For instance, the set $\mathcal{S}$ of q evenly distributed complex numebrs on the unit circle, i.e. $\mathcal{S} = \{e^{j \cdot 2\pi i/q}, j = 0, 1, \ldots, q-1\}$, is used for q-phase shift keying (q-PSK). In particular, we use

$$\alpha_j = \omega^j, \qquad j = 1, 2, \ldots, n \tag{2.21}$$

as our points of evaluations.

2.4.1 Gaussian channel

Definition 2.4.1 *(Real Gaussian channel). Suppose we send information over a channel with additive white Gaussian noise. Then the output is*

$$Y = X + Z$$

where X is the channel input, Y is the channel output and Z is a Gaussian with mean zero and variance σ^2, i.e., $Z \sim \mathcal{N}(0, \sigma^2)$.

Definition 2.4.2 *(Power Constraint). If the input codeword $(x_1, x_2, ..., x_n)$, it is assumed that the average power is constrained so that*

$$\frac{1}{n}\sum_{i=1}^{n} x_i^2 \leq P \tag{2.22}$$

and P is called the power constraint *of the channel.*

We calculate the probability of error for binary transmission. Suppose that we send either $-\sqrt{P}$ or $\sqrt{P}$ over the channel. The receiver determine the the result based on the signs of the received signal, then the error probability is

$$\begin{aligned}
P_e &= \frac{1}{2}P(Y < 0|X = \sqrt{P}) + \frac{1}{2}P(Y > 0|X = -\sqrt{P}) \\
&= \frac{1}{2}P(Z < -\sqrt{P}|X = \sqrt{P}) + \frac{1}{2}P(Z > \sqrt{P}|X = -\sqrt{P}) \\
&= \frac{1}{2}P(Z < -\sqrt{P}) + \frac{1}{2}P(Z > \sqrt{P}) \\
&= P(Z > \sqrt{P}) \\
&= \int_{\sqrt{P}}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} dx \\
&= 1 - \Phi(\sqrt{\frac{P}{\sigma^2}})
\end{aligned}$$

where $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$.

Let $I(X;Y)$ denote the mutual information of X and Y, then the *capacity* of a channel is defined as $C = \max_P I(X;Y)$. It is known that the capacity of a Gaussian channel with power constraint P and noise variance σ^2 is

$$C = \frac{1}{2}\log(1 + \frac{P}{\sigma^2}) \quad \text{bits per transmission.} \tag{2.23}$$

Theorem 2.4.3 (Channel Coding Theorem) *The Shannon theorem states that given a noisy channel with channel capacity C and information transmitted at a rate R, then if $R < C$ there exist codes that allow the probability of error at the receiver to be made arbitrarily small. This means that, theoretically, it is possible to transmit information nearly without error at any rate below a limiting rate, C.*

We make some remarks on modulations. In digital communications, the traditional way to send a RS codeword is to modulate it to binary bits (there could be also other modulation methods), and then send binary bits over a noisy channel. The real valued noise is added to each bit, and we receive a real valued vector that has be rounded back to bits. Such rounding step is called a hard-decision. The rounded bits have to be demodulated back to a vector over a finite field, in order to apply the decoding algorithms. Instead of modulating and sending binary bits over a noisy channel, we send complex valued codewords directly. The details of how to send complex numebrs over a noisy channel is not discussed are not discussed in this thesis. We compare the procedures of RS codes over finite fields and complex numebrs.

	Modulation for RS codes over finite fields
Input:	A message sequence $f \in \mathbb{F}_q^k$ where $q = 2^p$.
Output:	A decoded message sequence $f \in \mathbb{F}_q^k$.
Step 1.	Encode f to a codeword $c \in \mathbb{F}_q^n$.
Step 2.	Modulate c to $c_1 \in \{\pm 1\}^{pn}$.
Step 3.	Send c_1 over a noisy channel and receive a real valued vector $b_1 = c_1 + e_1 \in \mathbb{R}^{pn}$.
Step 4.	Round each entry of b_1 to $\{\pm 1\}$ and get $b_2 \in \{\pm 1\}^{pn}$
Step 5.	Demodulate b_2 to $b = c + e \in \mathbb{F}_q^n$.
Step 6.	Apply the decoding algorithm (over finite fields) to b and output $f \in \mathbb{F}_q^k$.

	No modulation for RS codes over complex numbers
Input:	A message sequence $f \in S^k$ where $S = \{s_0, s_1, ..., s_{q-1}\}$ is a set of distinct complex numebrs.
Output:	A decoded message sequence $f \in S^k$.
Step 1.	Encode f to a codeword $c \in \mathbb{C}^n$.
Step 2.	Send c over a noisy channel and receive a complex valued vector $b = c + e \in \mathbb{C}^n$.
Step 3.	Apply the decoding algorithm (over complex numbers) to b, say we get $\tilde{f} \in \mathbb{C}^k$.
Step 4.	Round each entry of $\tilde{f}$ to S and output $f \in S^k$.

2.4.2 Decoding for exact coefficients

Suppose a codeword $c = (c_1, c_2, \ldots, c_n)^T$ is transmitted over an often noisy channel and we receive a vector $b = (b_1, b_2, ..., b_n)^T$ with

$$b = c + e \tag{2.24}$$

where $e = (e_1, e_2, ..., e_n)^T$ represents the errors from the channel.

Throughout the rest part of this chapter, we assume k and n (the length of the message and the codeword) are both even. Let

$$\ell := \{1 \le i \le n : e_i = b_i - c_i \neq 0\} \quad \text{and} \quad t := \frac{n-k}{2}. \tag{2.25}$$

Recall that over finite fields, RS codes can be decoded uniquely by Gao's algorithm [17] if $\ell \le t$, and can be decoded (not uniquely) by Sudan's list decoding algorithm [39] if $\ell > t$.

We consider the problem of decoding RS codes over complex numbers and assume $\ell \le t$, i.e., most coefficients of $b_1, ..., b_n$ are received exactly. Define $h(x) \in \mathbb{C}[x]$ to be a polynomial of degree n,

$$h(x) = \prod_{j=1}^{n} (x - \alpha_j). \tag{2.26}$$

$h(x)$ can be pre-computed prior to encoding because it does not depend on the message $(f_0, f_1, ..., f_{k-1})$. Define another polynomial $g(x) \in \mathbb{C}[x]$ of degree $\le n-1$ such that

$$g(\alpha_j) = b_j, \quad j = 1, 2, \ldots, n \tag{2.27}$$

$g(x)$ is unique and it can be computed by Lagrange interpolation. We state and prove the following crucial theorem.

Theorem 2.4.4 *For any (n, k) RS code over complex numbers, suppose the received vector has $\ell \le t = \frac{n-k}{2}$ errors. Define $t = \frac{n-k}{2}$ and suppose the number of errors $\|e\|_0 = \ell \le t$. For any pair of polynomials ($u(x)$ and $v(x)$ in $\mathbb{C}[x]$, define $r(x) = u(x)g(x) + v(x)h(x)$ where $g(x)$ and $h(x)$ are*

defined in (2.27) and (2.26). If

$$\begin{aligned} \deg u(x) &\leq t, && (2.28)\\ \deg v(x) &\leq t-1, && (2.29)\\ \deg r(x) &= \deg[u(x)g(x)+v(x)h(x)] \leq k+t-1, && (2.30) \end{aligned}$$

then the message polynomial $f(x) = \frac{r(x)}{u(x)}$. *Further more, let* $Z = \{(u(x), v(x)) : u(x), v(x) \in \mathbb{C}[x]$ *satisfy* $(2.28)-(2.30)\}$, *then* $\dim_{\mathbb{C}} Z = t-\ell+1$.

Proof. Define the error locator polynomial

$$w(x) = \prod_{1\leq j\leq n, e_j\neq 0} (x-\alpha_j) \tag{2.31}$$

and

$$w_0(x) = \prod_{1\leq j\leq n, e_j=0} (x-\alpha_j), \tag{2.32}$$

where $\deg w(x) = \ell$ and $\deg w_0(x) = n-\ell$ since $\|e\|_0 = \ell$, then

$$h(x) = w(x)w_0(x). \tag{2.33}$$

Define $\bar{w}(x) \in F[x]$ the unique polynomial of degree at most $\ell-1$ such that

$$\bar{w}(\alpha_j) = \frac{b_j - f(\alpha_j)}{w_0(\alpha_j)} \quad \text{for all } 1\leq j\leq n \text{ with } e_j \neq 0, \tag{2.34}$$

then

$$g(x) = w_0(x)\bar{w}(x) + f(x), \tag{2.35}$$

since the polynomials on both sides have degree $\leq n-1$ and have the same value at w_j, for $1\leq j\leq n$. Plug (2.35) and (2.33) in $r(x)$, we get

$$\begin{aligned} r(x) &= u(x)g(x)+v(x)h(x)\\ &= u(x)[w_0(x)\bar{w}(x)+f(x)] + v(x)[w(x)w_0(x)]\\ &= w_0(x)\big(u(x)\bar{w}(x)+v(x)w(x)\big) + u(x)f(x) \end{aligned}$$

We must have

$$u(x)\bar{w}(x) + v(x)w(x) = 0 \tag{2.36}$$

because

$$\begin{aligned} \deg r(x) &\leq k + t - 1, \\ \deg u(x)f(x) &= \deg u(x) + \deg f(x) \leq t + k - 1, \\ \deg w_0(x) &= n - \ell \geq n - t > k + t - 1, \end{aligned}$$

and therefore, $r(x) = u(x)f(x)$.

Now we prove the second part of the theorem. Note $\gcd(\bar{w}(x), w(x)) = 1$ by their definitions, the equation (2.36) implies that $w(x)|u(x)$ and $\bar{w}(x)|v(x)$, hence

$$u(x) = w(x)d(x), \qquad v(x) = -\bar{w}(x)d(x), \tag{2.37}$$

for some polynomial $d(x)$ of degree $\leq t - \ell$. Conversely, it is easy to check that, for any $d(x)$ with $\deg d(x) \leq t - \ell$, each pair of $(u(x), v(x))$ in (2.37) satisfy the conditions (2.28)-(2.30). Therefore, $\dim_{\mathbb{C}} Z = 1 + \deg d(x) = t - \ell + 1$. □

Gao proves a similar theorem for RS codes over finite fields in [17]. By the above theorem, we see the main problem here is to find $u(x)$ and $v(x)$ satisfying conditions (2.28)-(2.30). Gao's algorithm uses the extended Euclidean algorithm (EEA) to find the polynomial pairs $(u(x), v(x))$ with strictly decreasing degrees. However, EEA is known to be numerically stable, as it computes $u(x)$ and $r(x)$ recursively. Instead, we introduce another way to find $u(x)$ and $v(x)$ satisfying conditions (2.28)-(2.30), by using so called Sylvester matrix (defined later on).

Since we have the following degree constrains,

$$\begin{aligned} \deg g(x) &\leq n - 1, \\ \deg h(x) &= n, \\ \deg u(x) &\leq t \quad \text{(condition (2.28))}, \\ \deg v(x) &\leq t - 1 \quad \text{(condition (2.29))}, \\ \deg r(x) &\leq k + t - 1 \quad \text{(condition (2.30))}, \end{aligned}$$

we may denote

$$g(x) = \sum_{j=0}^{n-1} g_j x^j, \qquad h(x) = \sum_{j=0}^{n} h_j x^j,$$

$$u(x) = \sum_{j=0}^{t} u_j x^j, \qquad v(x) = \sum_{j=0}^{t-1} v_j x^j, \qquad r(x) = \sum_{j=0}^{n+t-1} r_j x^j. \tag{2.38}$$

where we allow $g(x)$, $u(x)$, $v(x)$ and $r(x)$ to have leading zero coefficients. Then $r(x) = u(x)g(x) + v(x)h(x)$ can be represented as

$$\begin{bmatrix} r_{n+t-1} \\ r_{n+t-2} \\ \\ \vdots \\ \\ \\ r_{k+t} \\ r_{k+t-1} \\ \vdots \\ r_t \\ r_{t-1} \\ \vdots \\ r_0 \end{bmatrix} = \begin{bmatrix} g_{n-1} & & & & & h_n & & \\ & g_{n-1} & & & & & \ddots & \\ \vdots & & \ddots & & & \vdots & & h_n \\ & \vdots & & g_{n-1} & & & & \vdots \\ & & & \vdots & & & & \\ g_k & g_{k+1} & \cdots & g_{k+t} & h_{k+1} & \cdots & h_{k+t} & \\ g_{k-1} & g_k & \cdots & g_{k+t-1} & h_k & \cdots & h_{k+t-1} & \\ \vdots & \vdots & & & \vdots & & & \\ g_0 & & & \vdots & & & \vdots & \\ & g_0 & & & h_0 & & & \\ & & \ddots & & & \ddots & & \\ & & & g_0 & & & h_0 & \end{bmatrix} \cdot \begin{bmatrix} u_t \\ \\ \\ \vdots \\ \\ \\ u_0 \\ v_{t-1} \\ \\ \\ \vdots \\ \\ v_0 \end{bmatrix}. \tag{2.39}$$

The above matrix formed by coefficients of $g(x)$ and $h(x)$ is called the Sylvester matrix of $g(x)$ and $h(x)$. Denote the upper as

$$M = \begin{bmatrix} g_{n-1} & & & & h_n & & \\ & g_{n-1} & & & & \ddots & \\ \vdots & & \ddots & & \vdots & & h_n \\ & \vdots & & g_{n-1} & & & \vdots \\ & & & \vdots & & & \\ g_k & g_{k+1} & \cdots & g_{k+t} & h_{k+1} & \cdots & h_{k+t} \end{bmatrix}, \tag{2.40}$$

and we call this $(n-k)\times(n-k+1)$ matrix M as a partial Sylvester matrix. By $\deg r(x) \le k+t-1$ (condition (2.30)), we must have

$$r_{n+t-1} = r_{n+t-2} = \cdots = r_{k+t} = 0,$$

i.e., $z = (u_t, u_{t-1}, \ldots, u_0, v_{t-1}, v_{t-2}, \ldots, v_0)^T$ is a solution to

$$Mz = 0. \tag{2.41}$$

Conversely, from each solution z to (2.41) we can construct $(u(x), v(x))$ and the pair satisfy the conditions (2.28)-(2.30). Therefore, the null space of M

$$Null(M) = \{z \in \mathbb{C}^{n-k+1} : Mz = 0\}$$

gives all valid $(u(x), v(x))$ pairs. Note the matrix M has $n-k+1 = 2t+1$ rows and by Theorem 2.4.4,

$$Rank[Null(M)] = t - \ell + 1 \quad \text{and} \quad Rank[M] = t + \ell. \tag{2.42}$$

2.4.3 Numerical decoding algorithm

In practice, when a codeword entry c_i is transmitted over an often noisy channel, we never receive an exact $b_i = c_i$. So in this section, we assume a more practical error model

$$b = c + e + \eta \tag{2.43}$$

where $\ell := \{1 \le i \le n : e_i = b_i - c_i \ne 0\} \le t = \frac{n-k}{2}$ and $\eta = (\eta_1, ..., \eta_n)^T$ is a vector of small random noise. As a result, the matrix M (constructed from b) is perturbed. The first question is how to stably compute a solution z to $Mz = 0$ if the matrix M is perturbed. Also, the polynomials $r(x)$ and $u(x)$ (constructed from z) are perturbed. Note $f(x)$ is computed from the long division $\frac{r(x)}{u(x)}$ and the coefficients of $f(x)$ are in $\mathcal{S} = \{s_0, s_1, ..., s_{q-1}\}$. The next question is how to stably compute the long division $\frac{r(x)}{u(x)}$ to produce a polynomial with coefficients in $\mathcal{S}$?

To answer the first question, we consider the SVD of M, say

$$M = PSQ^* \quad \text{or} \quad MQ = PS.$$

where $P = [p_1, p_2, ..., p_{n-k}] \in \mathbb{C}^{(n-k)\times(n-k)}$ and $Q = [q_1, q_2, ..., q_{n-k+1}] \in \mathbb{C}^{(n-k+1)\times(n-k+1)}$ are both unitary matrices, and $S \in \mathbb{R}^{(n-k)\times(n-k+1)}$ have singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n-k} \geq 0$ in the diagonal. By (2.42), the approximate rank of M is $t + \ell$. Also it is reasonable that $\sigma_1 \geq \sigma_2, ..., \sigma_{t+\ell} \gg \sigma_{t+\ell}$ and $\sigma_{t+\ell}, \sigma_{t+\ell+1}, ..., \sigma_{n-k}$ are very close to zero. From (2.4.3), we have

$$Mq_i = \begin{cases} \sigma_i p_i, & 1 \leq i \leq t+\ell; \\ \sigma_i p_i \approx 0, & t+\ell+1 \leq i \leq n-k; \\ 0, & i = n-k+1. \end{cases}$$

Therefore, the last column of Q is automatically a good solution to $Mz = 0$.

Now we answer the second question. If $r(x)$ and $u(x)$ are exact, then the long division $f(x) = \frac{r(x)}{u(x)}$ is exact with coefficients in $\mathcal{S} = \{s_0, s_1, ..., s_{q-1}\}$. However, both $r(x)$ and $u(x)$ are perturbed, the following approximate long division problem is considered.

A naïve way is to use the use long division and then round the resulting polynomial to the closest polynomial in $\mathcal{S}[x]$. However, it turns out to be unstable as direct long division compute coefficients sequentially. (It is similar to the reason why EEA is unstable.) Also, it seems unlikely to find the optimal solution $\underset{f}{argmin} \left\| f(x)u(x) - r(x) \right\|_2$, as the exhaustive search takes q^k trials. What we do is to consider the Toeplitz matrix

$$r = \begin{bmatrix} r_{k+t-1} \\ r_{k+t-2} \\ \\ \vdots \\ \\ \\ r_1 \\ r_0 \end{bmatrix} \approx \begin{bmatrix} u_t & & & \\ u_{t-1} & u_t & & \\ \vdots & \vdots & \ddots & \\ u_0 & u_1 & \ddots & u_t \\ & u_0 & \ddots & \vdots \\ & & \ddots & \vdots \\ & & & u_0 \end{bmatrix} \begin{bmatrix} f_{k-1} \\ f_{k-2} \\ \\ \vdots \\ \\ \\ f_1 \\ f_0 \end{bmatrix} = Uf \tag{2.44}$$

where U is the Toeplitz matrix of $u(x)$. We describe our modified long division in Algorithm 5 and

summarize our main Algorithm 6.

Algorithm 5 Modified long division.

Input: $r(x) \in \mathbb{C}[x]$ of degree $k+t-1$, $u(x) \in \mathbb{C}[x]$ of degree t, $\mathcal{S} = \{s_0, s_1, \ldots, s_{q-1}\}$.
Output: $f(x) \in \mathcal{S}[x]$ of degree $k-1$.
Step 1. Construct the Toeplitz matrix U and vector r as in (2.44).
Step 2. Compute the least squares solution $\tilde{f} = U^{\dagger} r$.
Step 3. Round $\tilde{f}$ to the closest vector in $\mathcal{S}^{k-1}$, say $f = (f_{k-1}, ..., f_0)$. Output $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{k-1} x^{k-1}$.

Algorithm 6 Decoding RS codes over complex numbers.

Input: $b = c + e \in \mathbb{C}^n$, $\alpha_1, ..., \alpha_n \in \mathbb{C}$ and $\mathcal{S} = \{s_0, s_1, ..., s_{q-1}\}$.
Output: A message polynomial $f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1} \in \mathcal{S}[x]$.
Step 0. Pre-compute the polynomial $h(x)$ as in (2.26).
Step 1. Compute $g(x)$ as in (2.27).
Step 2. Construct the partial Sylvester matrix M from $g(x)$ and $h(x)$ as in (2.40).
Step 3. Compute the SVD of $M = PSQ^*$.
Step 4. Set $z =$ the last column of Q, as a solution to $Mz = 0$.
Step 5. Apply the modified long division Algorithm 5 to complete $\frac{r(x)}{u(x)}$, output the result $f(x)$.

2.4.4 Experiments

In this section, we give the following two examples to show the SVD of the partial Sylvester matrix and the modified long division. Also, we provide numerical experiments on the performance of our main algorithm (Algorithm 6). All computation in this section is run in MATLAB 2011a with default IEEE floating point precision with 16 decimal digits, although we display only 4 decimal digits in two examples.

Example 2.4.5 (SVD of the partial Sylveter matirx) *Let* $q = 2, n = 16, k = 8$ *and* $\mathcal{S} = \{\pm 1\}$. *We generate the random message polynomial:*

$$f(x) = x^7 - x^6 + x^5 - x^4 - x^3 - x^2 + x + 1$$

and compute the RS codeword c *by plugging* n*-th roots of unity* $\omega_j = e^{j\frac{2\pi i}{n}}, j = 1, 2, ..., 16$ *in the message polynomial* $f(x)$. *The received codeword is* $b = c + e$ *where the error vector* e *has* $\ell = 3$

burst errors as follows

$$\begin{aligned} e \;=\; & (-0.0085+0.0002i, \quad 0.0206-0.0094i, \quad 0.0058+0.0080i, \quad 0.0098+0.0025i, \\ & -0.0075-0.0021i, \quad 0.5711+0.3958i, \quad -0.0019-0.0019i, \quad 0.4194+0.4578i, \\ & -0.0020-0.0020i, \quad 0.0050-0.0059i, \quad -0.0145-0.0069i, \quad -0.0025-0.0082i, \\ & -0.0058-0.0038i, \quad -0.0112-0.0142i, \quad 0.6559-0.2500i, \quad 0.0020+0.0037i). \end{aligned}$$

The partial Sylvester matrix M is constructed and ts singular values turn out to be

$$(\sigma_1, \ldots, \sigma_8) = (1.0874, \quad 1.0599, \quad 1.0345, \quad 1.0005, \quad 0.5571, \quad 0.4907, \quad 0.2794, \quad 0.0073).$$

Note that only σ_8 is significantly smaller than the other singular values and it is close to zero, we may conclude that the approximate rank of M is 7. Alternatively, we can determine the approximate rank of M by computing the ratios of consecutive singular values:

$$(\frac{\sigma_1}{\sigma_2}, \frac{\sigma_2}{\sigma_3}, \ldots, \frac{\sigma_7}{\sigma_8}) = (1.0259, \quad 1.0246, \quad 1.0340, \quad 1.7958, \quad 1.1354, \quad 1.7561, \quad 38.5245).$$

We pick a threshold $\theta = 10$ (pick different values of θ in different problems), then the largest ratio $\frac{\sigma_7}{\sigma_8} > \theta$ and it is significantly larger than others. We conclude that the approximate rank of M is 7 and the number of burst errors is $\ell = 7 - 4 = 3$. □

Example 2.4.6 ***(Modified long division)*** *Let $q = 4, n = 16, k = 8$ and $\mathcal{S} = \{\pm 1, \pm i\}$. Suppose we are given*

$$\begin{aligned} u(x) = \; & (-0.1865+0.4009i)x^4 + (-0.1974+0.2252i)x^3 + (0.1707-0.5692i)x^2 \\ & +(0.0580-0.0499i)x + (0.4761+0.3178i) \end{aligned}$$

and

$$\begin{aligned} r(x) = \quad & (-0.3778 + 0.3227i)x^{11} + (-0.6186 + 0.0385i)x^{10} + (-0.2270 - 0.3770i)x^9 \\ & +(-0.0476 + 0.1001i)x^8 + (0.3521 + 0.0619i)x^7 + (-0.3078 + 0.7086i)x^6 \\ & +(0.1448 - 0.5716i)x^5 + (-0.3028 + 0.1988i)x^4 + (0.8377 + 0.3953i)x^3 \\ & +(0.3181 + 0.7768i)x^2 + (-0.2719 + 0.5114i)x + (-0.3291 + 0.3831i), \end{aligned}$$

and we want to compute $f(x)$ with coefficients in $\mathcal{S}$ from the long division $\frac{r(x)}{u(x)}$. The true message polynomial is

$$f(x) = x^7 + ix^6 + x^5 + ix^4 + x^3 + ix^2 + ix + i.$$

The modified long division (Algorithm 3) computes the message polynomial correctly as

$$f_{mld}(x) = x^7 + ix^6 + x^5 + ix^4 + x^3 + ix^2 + ix + i.$$

However, the direct long division

$$\begin{aligned} f_{dld}(x) \quad = \quad & (1.0222 + 0.4668i)x^7 + (0.0933 + 0.7340i)x^6 + (0.9721 + 0.7540i)x^5 \\ & +(0.1555 + 0.4321i)x^4 + (0.3738 + 1.0618i)x^3 + (1.1376 - 0.0987i)x^2 \\ & +(-2.5293 + 2.6110i)x + (3.7307 - 0.8290i) \end{aligned}$$

is far away from $f(x)$, and $f_{dld}(x)$ is rounded to

$$x^7 + ix^6 + x^5 + ix^4 + ix^3 + x^2 + ix + 1.$$

with two coefficients in error. □

Experiment 2.4.7 ***(Performance of Algorithm 6 with burst errors)*** *We use $q = 2, \mathcal{S} = \{\pm 1\}$ or $q = 4, \mathcal{S} = \{\pm 1, \pm i\}$ as our constellations. We generate a random message sequence $(f_0, f_1, ..., f_{k-1}) \in \mathcal{S}^k$ and get our RS codeword $(c_1, c_2, ...c_n)$ by plugging roots of unities $\omega_j = e^{j \cdot 2\pi i/n}$ into the message polynomial $f(x) = f_0 + f_1 x + ... + f_{k-1}x^{k-1}$. Since $(c_1, c_2, ..., c_n)$ is the DFT of*

$(f_0, f_1, ..., f_{k-1}, 0, ..., 0)^T$ *(k zeros), we know the mean "power" of codeword entries is*

$$P = \sqrt{\frac{1}{n}\sum_{j=1}^{n} |c_j|^2} = \sqrt{k}.$$

Assumes the error vector $e \in \mathbb{C}^n$ *has* ℓ *burst errors and* $n - \ell$ *tiny errors where* $\ell < \frac{n-k}{2}$. *We generate a random integer* ℓ *from* $\{0, 1, ..., \frac{n-k}{2}\}$ *and a random* $T \subset \{1, 2, ...n\}$ *with* $|T| = \ell$. *For* $\delta = 0.05, 0.1, 0.15, 0.2, 0.25$, *generate the error vector* e *as*

$$e_j \sim \begin{cases} \sqrt{k} \cdot uniform(0.5, 1), & j \in T; \\ \sqrt{k} \cdot uniform(0, \delta), & j \notin T. \end{cases}$$

We apply Algorithm 6 to decode RS codes and repeat it for 10000 trials, then we output the total number of wrong codewords and wrong bits in Table 2.1 ($q = 2$*) and Table 2.2 (*$q = 4$*). Overall the Algorithm 6 is stable and it can successfully decode RS codes over complex numbers. For* $q = 2$, *Algorithm 6 succeeds almost every time. The result for* $q = 2$ *is slightly worse than the result for* $q = 2$. *This makes sense, when rounding all coefficients of* $\tilde{f}$ *to the closest points in* $\mathcal{S}$, *more choices of* $\mathcal{S}$ *increase the chance of mistakes. Also, we see there are about one or two wrong bits in each wrong codeword in average. That suggests even one codeword has one or two bits wrong, the other entries are still correctly decoded.*

Experiment 2.4.8 ***(Performance of Algorithm 6 with complex Gaussian noise)*** *The setup in this experiment is the same as the previous experiment except that we assume the error vector* $e \in \mathbb{C}^n$ *follows a complex Gaussian distribution. Recall the mean "power" of codeword entries is* $\sqrt{k}$, *we generate the error vector* e *as*

$$\begin{aligned} Real(e_j) &\sim \sqrt{k} \cdot \mathcal{N}(0, \sigma^2/2), \quad j = 1, 2, ..., n \\ Imag(e_j) &\sim \sqrt{k} \cdot \mathcal{N}(0, \sigma^2/2), \quad j = 1, 2, ..., n \end{aligned}$$

where $Real(e_j)$ *and* $Imag(e_j)$ *denote the real and the imaginary parts of* e_j, *and they are generated independently. In digital communications, the signal-to-noise-ratio (SNR) can be calculated as*

$$SNR = \frac{P_{signal}}{P_{noise}} = \frac{1}{\sigma}.$$

and it is usually represented in decibel (dB),

$$SNR_{dB} = 10\log_{10} SNR = 10\log_{10} \frac{1}{\sigma}.$$

The values of SNR_{dB} *are set to range from* 2 *to* 8*. We apply Algorithm 6 to decode RS codes and repeat it for 10000 trials, then we output the total number of wrong codewords and wrong bits in Table 2.3 (*$q = 2$*) and Table 2.4 (*$q = 4$*). Again, the result for* $q = 4$ *is slightly worse than the result for* $q = 2$*, and there are about one or two wrong bits in each wrong codeword in average. One important observation we want to remark is, the bit error ratio (BER)* $\frac{\#wrong\ bits}{k}$ *does not decrease as* n *increases. Unfortunately, this does not match with the channel coding theorem, which stats that if the bit rate is under the channel capacity, then there exists a code such that the bit error probability goes to zero as* n *increases. The reason may be significant increasement of computation complexity for computing SVD as* n *increases.*

n	4	4	4	4	4	4	4	4	4	4
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	0	0
#wrong bits	0	0	0	0	0	0	0	0	0	0
n	8	8	8	8	8	8	8	8	8	8
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	0	0
#wrong bits	0	0	0	0	0	0	0	0	0	0
n	16	16	16	16	16	16	16	16	16	16
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	0	0
#wrong bits	0	0	0	0	0	0	0	0	0	0
n	32	32	32	32	32	32	32	32	32	32
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	1	0
#wrong bits	0	0	0	0	0	0	0	0	1	0
n	64	64	64	64	64	64	64	64	64	64
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	0	1
#wrong bits	0	0	0	0	0	0	0	0	0	1
n	128	128	128	128	128	128	128	128	128	128
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	1	0	0	0	1	0
#wrong bits	0	0	0	0	1	0	0	0	1	0

Table 2.1: The performance of Algorithm 6 under burst error model for $q = 2$. The length and rate of the code are chosen from $4, 8, 16, 32, 64, 128$ and $0.5, 0.75$, respectively. The parameter δ ranges from 0.05 to 0.25. The table lists the number of wrong codewords and wrong bits for 10000 trials.

n	4	4	4	4	4	4	4	4	4	4
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	0	0	0	0	0	0	0
#wrong bits	0	0	0	0	0	0	0	0	0	0
n	8	8	8	8	8	8	8	8	8	8
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	0	0	5	4	0	0	0	0	2
#wrong bits	0	0	0	6	4	0	0	0	0	2
n	16	16	16	16	16	16	16	16	16	16
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	2	4	7	9	5	3	2	8	7
#wrong bits	0	2	4	7	9	8	7	3	8	9
n	32	32	32	32	32	32	32	32	32	32
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	1	3	4	8	26	0	6	4	7	15
#wrong bits	1	3	4	8	28	0	9	4	7	15
n	64	64	64	64	64	64	64	64	64	64
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	0	3	11	15	52	0	0	7	22	44
#wrong bits	0	3	12	17	54	0	0	7	22	47
n	128	128	128	128	128	128	128	128	128	128
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
δ	0.05	0.1	0.15	0.2	0.25	0.05	0.1	0.15	0.2	0.25
#wrong codewords	3	3	24	53	93	4	6	12	34	82
#wrong bits	3	3	24	56	100	4	6	12	36	130

Table 2.2: The performance of Algorithm 6 under burst error model for $q = 4$. The length and rate of the code are chosen from $4, 8, 16, 32, 64, 128$ and $0.5, 0.75$, respectively. The parameter δ ranges from 0.05 to 0.25. The table lists the number of wrong codewords and wrong bits for 10000 trials.

n	4	4	4	4	4	4	4	4	4	4
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	272	68	5	0	0	579	168	16	0	0
#wrong bits	272	69	5	0	0	591	169	16	0	0
n	8	8	8	8	8	8	8	8	8	8
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	625	127	14	0	0	1755	577	115	8	0
#wrong bits	651	127	14	0	0	1923	611	116	8	0
n	16	16	16	16	16	16	16	16	16	16
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	1309	331	52	6	0	3285	1151	236	23	3
#wrong bits	1424	342	53	6	0	3979	1248	247	23	3
n	32	32	32	32	32	32	32	32	32	32
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	2528	730	97	9	1	5454	2227	514	54	2
#wrong bits	2969	769	99	9	1	7941	2622	551	55	8
n	64	64	64	64	64	64	64	64	64	64
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	4573	1515	256	31	1	7933	4126	1048	127	13
#wrong bits	6361	1679	264	33	1	16466	5496	1146	128	13
n	128	128	128	128	128	128	128	128	128	128
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	7196	2870	581	65	4	9592	6592	2027	311	31
#wrong bits	13308	3535	622	69	4	33809	11458	2388	328	34

Table 2.3: The performance of Algorithm 6 under complex Gaussian model for $q = 2$. The length and rate of the code are chosen from $4, 8, 16, 32, 64, 128$ and $0.5, 0.75$, respectively. The table lists the number of wrong codewords and wrong bits for 10000 trials.

n	4	4	4	4	4	4	4	4	4	4
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	2257	964	318	54	1	3709	1909	620	126	10
#wrong bits	2429	997	331	54	1	4270	2061	631	126	10
n	8	8	8	8	8	8	8	8	8	8
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	3955	2017	649	98	11	6849	4441	2003	551	69
#wrong bits	4821	2248	671	100	11	10908	5879	2285	594	70
n	16	16	16	16	16	16	16	16	16	16
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	6590	3870	1403	256	28	9006	6961	3739	1123	152
#wrong bits	10270	4924	1533	259	28	21667	11756	4810	1226	160
n	32	32	32	32	32	32	32	32	32	32
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	8938	6169	2652	609	73	9899	8954	6027	2207	350
#wrong bits	21240	9726	3192	642	76	43636	23785	9692	2632	379
n	64	64	64	64	64	64	64	64	64	64
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	9864	8699	4778	1296	184	9999	9914	8498	4001	734
#wrong bits	43487	20710	6907	1469	191	88430	48512	20036	5492	799
n	128	128	128	128	128	128	128	128	128	128
rate	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75
SNR_{dB}	2	4	6	8	10	2	4	6	8	10
#wrong codewords	9997	9805	7456	2524	409	10000	10000	9752	6487	1542
#wrong bits	88941	42287	14553	3150	443	179516	98487	40890	11554	1767

Table 2.4: The performance of Algorithm 6 under complex Gaussian model for $q = 4$. The length and rate of the code are chosen from $4, 8, 16, 32, 64, 128$ and $0.5, 0.75$, respectively. The table lists the number of wrong codewords and wrong bits for 10000 trials.

Chapter 3

JL transforms via algebraic geometry (AG) codes

In [29], Kane and Nelson gave construction of the JL transform matrix A using error correction codes. In this chapter, we first describe their construction in Section 3.1. We give a better and explicit bound in Section 3.2 and give proofs in Section 3.3. In Section 3.4, we give a explict construction of the JL transform matrix A using algebraic geometry codes.

3.1 Code-based construction

In this section, we describe the code-based construction [29] by Kane and Nelson. Suppose $C \subset \{1, 2, ..., q\}^s$ is any code of length s and minimum distance d. Suppose

$$|C| \geq n. \tag{3.1}$$

We list any n different codewords $c_1, c_2, ..., c_n \in C$ as an $s \times n$ matrix

$$[c_1, c_2, ..., c_n] = [c_{rj}]_{s\times n} = \begin{bmatrix} c_{11} & \cdots & \cdots & c_{1n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ c_{s1} & \cdots & \cdots & c_{sn} \end{bmatrix}$$

where $c_{rj} \in \{1, 2, ..., q\}, 1 \leq r \leq s, 1 \leq j \leq n$. For $1 \leq i \leq q$, define $E(i) \in \{0, 1\}^q$ to be the i-th unit vector (a column vector). For $1 \leq r \leq s, 1 \leq j \leq n$, let v_{rj} be i.i.d ± 1 random integers where

$$P(v_{rj} = 1) = P(v_{rj} = -1) = 0.5.$$

Let $m = qs$. Define an $m \times n$ matrix A by replacing each entry c_{rj} by the vector $\frac{v_{rj}}{\sqrt{s}} E(c_{rj})$ for $1 \leq r \leq s, 1 \leq j \leq n$, i.e.,

$$A = \frac{1}{\sqrt{s}} \begin{bmatrix} v_{11}E(c_{11}) & \cdots & \cdots & v_{1n}E(c_{1n}) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ v_{s1}E(c_{s1}) & \cdots & \cdots & v_{sn}E(c_{sn}) \end{bmatrix} \tag{3.2}$$

Now consider the error term $\|Ax\|_2^2 - 1$ in the JL lemma where x is any given vector with $\|x\|_2 = 1$. For $1 \leq i, j \leq n$ and $1 \leq r \leq s$, define an indicator random variable η_{ijr} as

$$\eta_{ijr} = \begin{cases} 1, & c_{ri} = c_{rj}, i \neq j; \\ 0, & c_{ri} \neq c_{rj}, i \neq j; \\ 0, & i = j. \end{cases} \tag{3.3}$$

Let

$$N = sn. \tag{3.4}$$

Define an $N \times N$ block-diagonal matrix $B = [b_{ij}]_{N \times N}$ as

$$B = diag(B_1, B_2, \ldots, B_s) \tag{3.5}$$

where each $B_r = [b_{ij}^{(r)}]_{n \times n}$ is defined as

$$b_{ij}^{(r)} = x_i x_j \eta_{ijr}/s, \quad 1 \leq i, j \leq n, \quad 1 \leq r \leq s.$$

This matrix B depends on the code C and the vector x, thus B is not random. From $v_{rj} \in \{-1, 1\}, 1 \leq r \leq s, 1 \leq j \leq n$, we define a column vector of length N as

$$v = [v_{11}, v_{21}, ..., v_{s1}, ..., v_{1n}, v_{2n}, ..., v_{sn}]^T$$

The following lemma indicates that the error term $\|Ax\|_2^2 - 1$ in the JL lemma can be expressed as a quadratic form of the random vector v.

Lemma 3.1.1 *Let A, B and v be defined as above. Then*

$$\|Ax\|_2^2 - 1 = v^T B v.$$

Proof. We use a_i and a_j to denote the ith and jth columns of A. Then

$$\begin{aligned}
\|Ax\|_2^2 - 1 &= \sum_{1 \leq i \leq n} x_i a_i^T \sum_{1 \leq j \leq n} a_j x_j - 1 \\
&= \sum_{1 \leq i,j \leq n} x_i x_j a_i^T a_j - 1 \\
&= \sum_{\substack{1 \leq i,j \leq n \\ i \neq j}} x_i x_j a_i^T a_j + \sum_{1 \leq i \leq n} x_i^2 - 1 \\
&= \sum_{\substack{1 \leq i,j \leq n \\ i \neq j}} x_i x_j a_i^T a_j \quad (\text{because } \|x\|_2 = 1)
\end{aligned}$$

By the definition of A, the inner product of two distinct columns a_i and a_j $(i \neq j)$ is

$$\begin{aligned}
a_i^T a_j &= \sum_{r=1}^{s} \frac{v_{ri} E(c_{ri})}{\sqrt{s}} \cdot \frac{v_{rj} E(c_{rj})}{\sqrt{s}} \\
&= \sum_{r=1}^{s} \frac{v_{ri} v_{rj} E(c_{ri}) E(c_{rj})}{s} \\
&= \sum_{r=1}^{s} \frac{v_{ri} v_{rj} \eta_{ijr}}{s} \quad (\text{by the definition of } \eta_{ijr} \text{ and } E(i))
\end{aligned}$$

Then

$$\begin{aligned}
\|Ax\|_2^2 - 1 &= \sum_{\substack{1\le i,j\le n\\ i\neq j}} x_i x_j \sum_{r=1}^{s} \frac{v_{ri}v_{rj}\eta_{ijr}}{s} \\
&= \sum_{r=1}^{s} \sum_{\substack{1\le i,j\le n\\ i\neq j}} \frac{x_i x_j \eta_{ijr}}{s} v_{ri}v_{rj} \\
&= \sum_{r=1}^{s} \sum_{1\le i,j\le n} \frac{x_i x_j \eta_{ijr}}{s} v_{ri}v_{rj} \quad (\text{because } \eta_{iir} = 0) \\
&= v^T B v.
\end{aligned}$$

□

3.2 A better and explicit bound

Throughout the rest of this chapter, we always use ℓ to denote a positive even integer. By Lemma 3.1.1 and Markov's inequality,

$$P[|\,\|Ax\|_2^2 - 1| > \epsilon] = P[|v^T B v| > \epsilon] < \epsilon^{-\ell} \cdot E[(v^T B v)^\ell]. \tag{3.6}$$

Thus in order to prove the JL lemma, the main problem is to give a good upper bound of $E[(v^T B v)^\ell]$. If one can prove $E[(v^T B v)^\ell] < \beta^\ell$ for some $\beta < \epsilon$, then $P[|\,\|Ax\|_2^2 - 1| > \epsilon] < (\epsilon^{-1}\beta)^\ell$. When ℓ increases, the probability can be artibarily small.

In Kane and Nelson's proof for their code-based construction, they use the Hanson-Wright inequality to bound $E[(v^T B v)^\ell]$ (Theorem 6, [29]). Recall the 2-norm $\|B\|_2$ is defined as the largest singular value of B, and the Frobenius norm is defined as $\|B\|_F = \sqrt{\sum_{i,j} b_{ij}^2}$. Then by the following Hanson-Wright inequality, $E[(v^T B v)^\ell]$ can be bounded via the 2-norm and the Frobenius norm of B.

Lemma 3.2.1 *(Hanson-Wright inequality [23]) For any symmetric matrix G and random uniform ± 1 Bernoulli vector v,*

$$E[|v^T G v|^\ell] \le (\hat{C} \cdot \max\{\sqrt{\ell}\,\|G\|_F, \ell\,\|G\|_2\})^\ell,$$

where $\|\cdot\|_F$ is the Frobenius norm, $\|\cdot\|_2$ is the 2-norm and $\hat{C}$ is a constant does not depend on ℓ.

In [15], Kane and Nelson reproved the Hanson Wright inequality and provide the constant $\hat{C} = 64$. For the matrix B defined in (3.5), they give upperbounds $\|B\|_F \leq \frac{\sqrt{s-d}}{s}$ (Lemma 7, [29]) and $\|B\|_2 \leq \frac{1}{s}$ (Lemma 8, [29]), where s is the code length and d is the minimum distance of the code. Therefore their upperbounds on $E[(v^T Bv)^\ell]$ is

$$E[(v^T Bv)^\ell] \leq \left(64 \cdot \max\left\{\frac{\sqrt{\ell(s-d)}}{s}, \frac{\ell}{s}\right\}\right)^\ell. \tag{3.7}$$

The constant 64 may be too large for some applications. One of our main acontributions is to give a better uppperbound than (3.7). In fact, we will prove the following lemma in Section 3.3.

Lemma 3.2.2 *Let the $N \times N$ matrix B be defined as above from a code of length s and minimum distance d. Let v be the Bernoulli vector defined above. For any positive and even integer ℓ,*

$$E[(v^T Bv)^\ell] \leq (2\ell - 1)!! \left(\frac{\sqrt{s-d)}}{s}\right)^\ell. \tag{3.8}$$

The proof of Lemma 3.2.2 is not trivial. We will prove it in Section 3.3. Let $C_\ell = ((2\ell-1)!!)^{\frac{1}{\ell}}/\sqrt{\ell}$ be a constant dependent only on ℓ, then

$$E[(v^T Bv)^\ell] \leq \left(C_\ell \cdot \frac{\sqrt{\ell(s-d)}}{s}\right)^\ell.$$

We compare our new bound (3.8) with (3.7). Asymptotically,

$$\lim_{\ell\to\infty} C_\ell = \lim_{\ell\to\infty} \frac{((2\ell-1)!!)^{\frac{1}{\ell}}}{\sqrt{\ell}} \sim \frac{2\sqrt{\ell}}{e}.$$

Although C_ℓ is not bounded, it increases slowly

$$C_2 = 1.22, C_4 = 1.60, C_8 = 2.17, C_{16} = 3.01, C_{32} = 4.21, C_{64} = 5.92, C_{128} = 8.35, ...$$

It can be shown that $C_\ell < 64$ for $\ell \leq 7564$. Therefore,

$$\left(C_\ell \cdot \frac{\sqrt{\ell(s-d)}}{s}\right)^\ell \leq \left(64 \cdot \max\left\{\frac{\sqrt{\ell(s-d)}}{s}, \frac{\ell}{s}\right\}\right)^\ell \quad \text{for } \ell \leq 7564. \tag{3.9}$$

We will see it is enough for JL projections in dimensions of practical applications. In the following,

we state our main theorem.

Theorem 3.2.3 *Let C be any code with the code length s, minimum distance d and $|C| \geq n$. Suppose a sparse random matrix A is defined from C, as discussed in Section 3.1. For any $\epsilon > 0$ and even integer $\ell > 1$, suppose*

$$\frac{s^2}{s-d} \geq 4\epsilon^{-2} \cdot [(2\ell-1)!!]^{\frac{1}{\ell}}, \tag{3.10}$$

Then, for any $\delta = 0.5^{\ell}$ and $x \in \mathbb{R}^n$ with $\|x\|_2 = 1$, we have $P[|\,\|Ax\|_2^2 - 1| > \epsilon] < \delta$ where the probability is over the distribution of the random matrix A.

Proof. By Lemma 3.2.2 and (3.6), the theorem immediately follows. □

From Theorem 3.2.3, we see the integer ℓ is used to control the error probability $\delta = 0.5^{\ell}$. In most applications, it may be sufficient to have $\ell \leq 7564$ (required in (3.9)) since the probability $(0.5)^{7564}$ is small enough.

An important question is, what codes with length s and minimum distance d can satisfy the inequality (3.10)? In Section 3.4, we answer this question by considering algebraic geometry (AG) codes.

3.3 Proof of Lemma 3.2.2

We prove Lemma 3.2.2 in this section. First. recall the matrix $B = [b_{ij}]$ is an $N \times N$ matrix and the vector v of length N contains ± 1 uniform Bernoulli random variables. We can expand terms of $E[(v^T B v)^{\ell}]$ as

$$\begin{aligned} E[(v^T B v)^{\ell}] &= E\left[\left(\sum_{1 \leq i,j \leq N} b_{ij} v_i v_j\right)^{\ell}\right] \\ &= E\left[\sum_{(i_1,\ldots,i_{2\ell}) \in \{1,\ldots,N\}^{2\ell}} (b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}) \cdot (v_{i_1} v_{i_2} \cdots v_{i_{2\ell-1}} v_{i_{2\ell}})\right] \\ &= \sum_{(i_1,\ldots,i_{2\ell}) \in \{1,\ldots,N\}^{2\ell}} (b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}) \cdot E[v_{i_1} v_{i_2} \cdots v_{i_{2\ell-1}} v_{i_{2\ell}}]. \end{aligned} \tag{3.11}$$

where the summation is over all vectors $(i_1, \ldots, i_{2\ell}) \in \{1, 2, ..., N\}^{2\ell}$ and the estimation is over all i.i.d. ± 1 uniform Bernoulli random variables $v_{i_1}, \ldots, v_{i_{2\ell}}$. One may observe that $E[v_{i_1} v_{i_2} \cdots v_{i_{2\ell-1}} v_{i_{2\ell}}]$

is nonzero only if each index in the vector $(i_1, ..., i_{2\ell})$ occur an even number of times. Such vector $(i_1, ..., i_{2\ell})$ produces an partition of $\{1, ..., 2\ell\}$ where the size of each subset is even.

Definition 3.3.1 *Denote the partition of* $\{1, 2, ..., 2\ell\}$ *implied by any vector* $(i_1, i_2, \ldots, i_{2\ell}) \in \{1, 2, ..., N\}^{2\ell}$ *as* $\pi(i_1, i_2, ..., i_{2\ell})$. *A partition of* $\{1, 2, ..., 2\ell\}$ *is even if the size of every subset is even. In particular, a partition of* $\{1, 2, ..., 2\ell\}$ *is called a* 2*-partition if the size of every subset is exactly* 2. *Conversely, for a partition* P *of* $\{1, 2, ..., 2\ell\}$, *the set of all vectors in* $\{1, 2, ..., N\}^{2\ell}$ *that generated by the partition* $(i_1, i_2, \ldots, i_{2\ell})$ *is denoted as* $\tau(P)$.

Example 3.3.2 $\pi(i_1, i_3, i_2, i_2, i_2, i_3, i_2, i_1) = \{1, 8\} \cup \{3, 4, 5, 7\} \cup \{2, 6\}$ *is an even partition of* $\{1, 2, ..., 8\}$. *Conversely, the indices set generated by* $\{1, 8\} \cup \{3, 4, 5, 7\} \cup \{2, 6\}$ *is*

$$\tau(\{1, 8\} \cup \{3, 4, 5, 7\} \cup \{2, 6\}) = \{(i_1, i_3, i_2, i_2, i_2, i_3, i_2, i_1) : i_1, i_2, i_3 = 1, 2, ..., N\}.$$

Since v_{i_1}, v_{i_2} *and* v_{i_3} *are i.i.d.* ± 1 *Bernoulli random variables,*

$$E\left[v_{i_1} v_{i_3} v_{i_2} v_{i_2} v_{i_2} v_{i_3} v_{i_2} v_{i_1}\right] = E\left[v_{i_1}^2 v_{i_2}^4 v_{i_3}^2\right] = 1$$

Example 3.3.3 $\pi(1, 3, 2, 2, 2, 3, 2, 4) = \{1\} \cup \{8\} \cup \{3, 4, 5, 7\} \cup \{2, 6\}$ *is not an even partition of* $\{1, 2, ..., 8\}$. *Since* v_{i_1}, v_{i_2} v_{i_3} *and* v_{i_4} *are i.i.d.* ± 1 *Bernoulli random variables, we have*

$$E\left[v_{i_1} v_{i_3} v_{i_2} v_{i_2} v_{i_2} v_{i_3} v_{i_2} v_{i_4}\right] = E\left[v_{i_1} v_{i_2}^4 v_{i_3}^2 v_{i_4}\right] = E\left[v_{i_1} v_{i_4}\right] = 0.$$

From above two examples, we observe that

$$E[v_{i_1} v_{i_2} \cdots v_{i_{2\ell-1}} v_{i_{2\ell}}] = \begin{cases} 1, & \text{if } \pi(i_1, i_2, \ldots, i_{2\ell-1}, i_{2\ell}) \text{ is even;} \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, the summation in (3.11) can be restricted to all vectors $(i_1, i_2, \ldots, i_{2\ell-1}, i_{2\ell}) \in \{1, ..., N\}^{2\ell}$ that correspond to even partitions of $\{1, 2, ..., 2\ell\}$, that is,

$$E[(v^T B v)^\ell] = \sum_{\pi(i_1, ..., i_{2\ell}) \text{ is even}} b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}. \tag{3.12}$$

The above summation is over all vectors in

$$I_{even} = \{(i_1, \ldots, i_{2\ell}) \in \{1, 2, ..., N\}^{2\ell} : \pi(i_1, i_2, \ldots, i_{2\ell-1}, i_{2\ell}) \text{ is even}\}.$$

While summing (3.12) over I_{even} is hard, we consider splitting I_{even} into smaller subsets and summing (3.12) over these smaller subsets. Indeed, the sum over cyclic vectors $(i_1, i_2, i_2, i_3, ..., i_{t-1}, i_t, i_t, i_1)$ can be bounded. The result is presented in the follow lemma.

Lemma 3.3.4 *For any positive integer t with $2 \le t \le \ell$, we have*

$$\sum_{1 \le i_1, i_2, ..., i_t \le N} |b_{i_1 i_2} b_{i_2 i_3} \cdots b_{i_{t-1} i_t} b_{i_t i_1}| \le \frac{s-d}{s^t}.$$

Proof. By the definition of B, $|b_{i_1 i_2} b_{i_2 i_3} \cdots b_{i_{t-1} i_t} b_{i_t i_1}|$ is non-zero only if i_1-th, i_2-th, ... ,i_t-th columns are in the same block.

$$\begin{aligned}
& \sum_{1 \le i_1, i_2, ..., i_t \le N} |b_{i_1 i_2} b_{i_2 i_3} \cdots b_{i_{t-1} i_t} b_{i_t i_1}| \\
= & \sum_{r=1}^{s} \sum_{1 \le j_1, j_2, ..., j_t \le n} \left| \frac{x_{j_1} x_{j_2} \eta_{j_1 j_2 r}}{s} \cdot \frac{x_{j_2} x_{j_3} \eta_{j_2 j_3 r}}{s} \cdots \frac{x_{j_t} x_{j_1} \eta_{j_t j_1 r}}{s} \right| \quad (N = sn) \\
= & \frac{1}{s^t} \sum_{1 \le j_1, j_2, ..., j_t \le n} x_{j_1}^2 x_{j_2}^2 \cdots x_{j_t}^2 \sum_{r=1}^{s} \eta_{j_1 j_2 r} \eta_{j_2 j_3 r} \cdots \eta_{j_t j_1 r} \\
\le & \frac{1}{s^t} \sum_{1 \le j_1, j_2, ..., j_t \le n} x_{j_1}^2 x_{j_2}^2 \cdots x_{j_t}^2 \sum_{r=1}^{s} \frac{\eta_{j_1 j_2 r} + \eta_{j_2 j_3 r} + \cdots + \eta_{j_t j_1 r}}{t} \\
\le & \frac{1}{s^t} \sum_{1 \le j_1, j_2, ..., j_t \le n} x_{j_1}^2 x_{j_2}^2 \cdots x_{j_t}^2 \frac{\sum_{r=1}^{s} \eta_{j_1 j_2 r} + \sum_{r=1}^{s} \eta_{j_2 j_3 r} + \cdots + \sum_{r=1}^{s} \eta_{j_t j_1 r}}{t}
\end{aligned}$$

By the definition of η_{ijr} in (3.3), we have

$$\sum_{r=1}^{s} \eta_{ijr} \le s - d,$$

then it follows that

$$\begin{aligned}
& \sum_{1\le i_1,i_2,\ldots,i_t\le sn} |b_{i_1i_2}b_{i_2i_3}\cdots b_{i_{t-1}i_t}b_{i_ti_1}| \\
\le\ & \frac{1}{s^t}\sum_{1\le j_1,j_2,\ldots,j_t\le n} x_{j_1}^2x_{j_2}^2\cdots x_{j_t}^2\,\frac{(s-d)+(s-d)+\cdots(s-d)}{t} \\
=\ & \frac{s-d}{s^t}\sum_{1\le j_1,j_2,\ldots,j_t\le n} x_{j_1}^2x_{j_2}^2\cdots x_{j_t}^2 \\
=\ & \frac{s-d}{s^t}\,\|x\|_2^{2t} \\
=\ & \frac{s-d}{s^t}.
\end{aligned}$$

□

A more special case is that $\pi(i_1,i_2,...,i_{2\ell})$ is a 2-partition of $\{1,2,...,2\ell\}$ (see Definition 3.3.1). That is, any index in $\{i_1,i_2,...,i_{2\ell}\}$ occurs exactly twice.

Lemma 3.3.5 *Let P be any 2-partition of $\{1,2,...,2\ell\}$, then*

$$\sum_{(i_1,\ldots,i_{2\ell})\in\tau(P)} |b_{i_1i_2}\cdots b_{i_{2\ell-1}i_{2\ell}}| \le \left(\frac{\sqrt{s-d}}{s}\right)^{\ell}$$

Proof. Consider any $(i_1,i_2,\ldots,i_{2\ell})\in\tau(P)$. Since the diagonal entries $b_{ii}=0$ for all i, we may assume $i_1\neq i_2, i_3\neq i_4, ..., i_{2\ell-1}\neq i_{2\ell}$. Consider an undirected graph (V,E) as follows. Define the vertices $V=\{i_1,i_2,\ldots,i_{2\ell}\}$ (with repetitions), then $|V|=\ell$ since every number in $(i_1,i_2,\ldots,i_{2\ell})$ occurs exactly twice. Define the edges $E=\{(i_{2j-1},i_{2j}) : 1\le j\le\ell\}$. Every vertex in this graph has a degree of 2, thus it is a 2-regular graph. By graph theory, this graph consists of disconnected cycles only, say h cycles of sizes $t_1,t_2,\ldots,t_h\ge 2$ with $t_1+t_2+\cdots+t_h=\ell$. By Lemma 3.3.4,

$$\sum_{(i_1,i_2,\ldots,i_{2\ell-1},i_{2\ell})\in\tau(P)} |b_{i_1i_2}b_{i_3i_4}\cdots b_{i_{2\ell-1}i_{2\ell}}| \le \frac{s-d}{s^{t_1}}\frac{s-d}{s^{t_2}}\cdots\frac{s-d}{s^{t_h}} = \frac{(s-d)^h}{s^\ell}.$$

Note there are $h\le\frac{\ell}{2}$ cycles, with the equality holds when every cycle has a size of 2, it follows that

$$\sum_{(i_1,i_2,\ldots,i_{2\ell-1},i_{2\ell})\in\tau(P)} |b_{i_1i_2}b_{i_3i_4}\cdots b_{i_{2\ell-1}i_{2\ell}}| \le \left(\frac{\sqrt{s-d}}{s}\right)^{\ell}.$$

□

The following lemma indicates that we can sum over all $\tau(P)$ generated by a 2-partition P of $\{1, 2, ..., 2\ell\}$, then sum over all possible P.

Lemma 3.3.6

$$E[(v^T Bv)^\ell] \leq \sum_{P \text{ is a 2-partition of } \{1,2,...,2\ell\}} \left(\sum_{(i_1,...,i_{2\ell}) \in \tau(P)} |b_{i_1 i_2} b_{i_3 i_4} \cdots b_{i_{2\ell-1} i_{2\ell}}| \right)$$

Proof.

$$\begin{aligned} E[(v^T Bv)^\ell] &= \sum_{\pi(i_1,...,i_{2\ell}) \text{ is even}} b_{i_1 i_2} b_{i_3 i_4} \cdots b_{i_{2\ell-1} i_{2\ell}} \\ &\leq \sum_{\pi(i_1,...,i_{2\ell}) \text{ is even}} |b_{i_1 i_2} b_{i_3 i_4} \cdots b_{i_{2\ell-1} i_{2\ell}}| \\ &\leq \sum_{P \text{ is a 2-partition of } \{1,2,...,2\ell\}} \left[\sum_{(i_1,...,i_{2\ell}) \in \tau(P)} |b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}| \right]. \end{aligned} \tag{3.13}$$

The last inequality holds because if $\pi(i_1, \ldots, i_{2\ell})$ is even, then $(i_1, \ldots, i_{2\ell})$ must be in $\tau(P)$ for some 2-partition P. □

Now, having above lemmas, we are ready to prove the main Lemma 3.2.2 as follows.

Proof. (of Lemma 3.2.2) By Lemma 3.3.5, the estimate $E[(v^T Bv)^\ell]$ can be bounded as

$$E[(v^T Bv)^\ell] \leq \sum_{P \text{ is a 2-partition of } \{1,...,2\ell\}} \left[\sum_{(i_1,...,i_{2\ell}) \in \tau(P)} |b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}| \right].$$

Suppose $\mathcal{P} := \{P \text{ is a 2-partition of } \{1, ..., 2\ell\}\}$. We count $\mathcal{P}$ as follows. Consider any $P \in \mathcal{P}$, say $P = \{i_1, i_2\} \cup \cdots \cup \{i_{2\ell-1}, i_{2\ell}\} = \{1, ..., 2\ell\}$. Fixing any i_1, there are $2\ell - 1$ choices for i_2. Fixing any i_3, there are $2\ell - 3$ choices for i_4. Keeping this fashion, we get $|\mathcal{P}| = (2\ell - 1)!!$. By Lemma 3.3.6,

$$\sum_{(i_1,...,i_{2\ell}) \in \tau(P)} |b_{i_1 i_2} \cdots b_{i_{2\ell-1} i_{2\ell}}| \leq \left(\frac{\sqrt{s-d}}{s} \right)^\ell.$$

Therefore,

$$E[(v^T Bv)^\ell] \leq (2\ell - 1)!! \cdot \left(\frac{\sqrt{s-d}}{s} \right)^\ell.$$

□

We remark that the constant $|\mathcal{P}|$ can be slightly reduced, but the new constant is complicated. Recall $b_{i_{2t-1}i_{2t}} = 0$ if $i_{2t-1} = i_{2t}$ for some $1 \le t \le \ell$, since it is on the diagonal of B. Thus this case does not have to be counted. Let $\mathcal{P}' := \{P \text{ is a 2-partition of } \{1, ..., 2\ell\} : 2t-1 \neq 2t \text{ for } 1 \le t \le \ell\}$ be a subset of $\mathcal{P}$ and

$$|\mathcal{P}'| = (2\ell-1)!! - \binom{\ell}{1}(2\ell-3)!! + \binom{\ell}{2}(2\ell-5)!! - \cdots + (-1)^{\ell-1}\binom{\ell}{\ell-1}1!! + (-1)^{\ell}.$$

3.4 Explicit construction via AG codes

3.4.1 AG codes from the Garcia-Stichtenoth tower

Consider a linear (s, k, d) code C over $\mathbb{F}_q$ where s, k and d denote the length, dimension and minimum distance of the code, respectively. Define the transmission rate $R(C) = \frac{k}{s}$ and the relative minimum distance $\delta(C) = \frac{d}{s}$. Let

$$V_q = \{(\delta(C), R(C)) \in [0,1]^2 | C \text{ is a code over } \mathbb{F}_q\}$$

and let U_q be the set of limit points of V_q. Then there is a continuous function $\alpha_q : [0,1] \to [0,1]$ such that

$$U_q = \{(\delta, R) | 0 \le \delta \le 1, 0 \le R \le \alpha_q(\delta)\}$$

The following Gilbert-Varshamov(GV) bound is commonly used to give performance of long codes. Let $H_q = \delta \log_q(q-1) - \delta \log_q \delta - (1-\delta)\log_q(1-\delta)$, then $\alpha_q(\delta)$ has the lower bound

$$\alpha_q(\delta) \ge 1 - H_q(\delta) \quad (0 \le \delta \le 1 - 1/q).$$

Although it can be proved that there exist long codes that meet the GV bound, no explicit description were given. In 1980, Goppa [20] introduced algebraic geometry (AG) codes based on algebraic geometry curves. The performance of AG code depends the ratio $\frac{N}{g}$ where N is the number of rational points of the curve and g is called *genus* of the curse. An AG code is good if the ratio $\frac{N}{g}$ is large and its upperbound is call the Drinfeld-Vladut (DV) bound

$$\limsup_{g \to \infty} \frac{N}{g} \le \sqrt{q} - 1.$$

In 1982, Tsfasman, Vladut and Zink proved the existence of AG codes that exceed the GV bound, by introducing the modular curves. In 1995, Garcia and Stichtenoth (GS) provided an algorithm to generate such modular of AG codes using towers.

Definition 3.4.1 *(Garcia-Stichtenoth tower [18]). Let $\mathcal{F} := (F_0, F_1, ...)$ denote the tower of extensions of rational function field $F_0 = \mathbb{F}_{q^2}(x_0)$ where q is a prime power. For $i \geq 1$ let $F_i := F_{i-1}(x_i)$ where*

$$x_i^q + x_i = \frac{x_{i-1}^q}{x_{i-1}^{q-1} + 1}.$$

Let $u \geq 1$ be an integer. Denote the genus of F_u as $g(F_u)$, and the number of places of degree one of F_u as $N(F_u)$. Then from [4],

$$g(F_u) = (q^{\lfloor \frac{u+1}{2} \rfloor} - 1)(q^{\lceil \frac{u+1}{2} \rceil} - 1) = \begin{cases} (q^{\frac{u+1}{2}} - 1)^2, & \text{u is odd;} \\ (q^{\frac{u}{2}} - 1)(q^{\frac{u+2}{2}} - 1), & \text{u is is even.} \end{cases} \tag{3.14}$$

and

$$N(F_u) \geq q^u(q^2 - q).$$

From the GS-tower, one can construct an linear AG code as described in the following theorem.

Theorem 3.4.2 *[4] Let $u \geq 1$ be some integer and $q \geq 2$ be a prime power. Let $s = q^u(q^2 - q)$, $g = (q^{\lfloor \frac{u+1}{2} \rfloor} - 1)(q^{\lceil \frac{u+1}{2} \rceil} - 1)$. Suppose $k < s$ is an integer and $d = s - k - g$. One can explicitly construct an linear (s, k, d) AG code over $\mathbb{F}_{q^2}$ with code length s, dimension k and minimum distance d.*

3.4.2 Code parameters

In this section, we discuss on code parameters. Suppose we are given an (s, k, d)-AG code over $\mathbb{F}_{q^2}$ from the GS tower, where $q \geq 2$ is a prime power. From Theorem 3.4.2, we know

$$s = q^u(q^2 - q), \quad d = s - k - g, \quad g = (q^{\lfloor \frac{u+1}{2} \rfloor} - 1)(q^{\lceil \frac{u+1}{2} \rceil} - 1) \tag{3.15}$$

where $u \geq 1$ is an integer. The condition (3.10) of Theorem 3.2.3 becomes

$$\frac{s^2}{s-d} = \frac{(q^{u+2} - q^u)^2}{k + (q^{\lfloor \frac{u+1}{2} \rfloor} - 1)(q^{\lceil \frac{u+1}{2} \rceil} - 1)} \geq 4\epsilon^{-2} \cdot [(2\ell - 1)!!]^{\frac{1}{\ell}}. \tag{3.16}$$

The right hand side of (3.10) depends on ϵ and ℓ where ϵ controls the tolerance on $|\,\|Ax\|_2^2 - 1|$ and ℓ controls the error probability $\delta = 2^{-\ell}$. The left hand side of (3.10) depends on parameters s and d of AG codes. The Table 3.1 presents some parameters for JL transforms that are useful in practice.

q	u	k	d	g	$s = q^u(q^2 - q)$	$m = s \cdot q^2$	$n = (q^2)^k$	ϵ	$\delta = 0.5^\ell$
2	6	15	8	105	128	512	1.07×10^{09}	0.59	0.5^{16}
2	7	15	16	225	256	1024	1.07×10^{09}	0.42	0.5^{16}
2	7	15	16	225	256	1024	1.07×10^{09}	0.59	0.5^{32}
2	8	16	31	465	512	2048	4.29×10^{09}	0.30	0.5^{16}
2	8	16	31	465	512	2048	4.29×10^{09}	0.42	0.5^{32}
2	8	16	31	465	512	2048	4.29×10^{09}	0.59	0.5^{64}
2	9	16	47	961	1024	4096	4.29×10^{09}	0.21	0.5^{16}
2	9	16	47	961	1024	4096	4.29×10^{09}	0.30	0.5^{32}
2	9	16	47	961	1024	4096	4.29×10^{09}	0.42	0.5^{64}
2	9	16	47	961	1024	4096	4.29×10^{09}	0.59	0.5^{128}
2	10	17	78	1953	2048	8192	1.72×10^{10}	0.15	0.5^{16}
2	10	17	78	1953	2048	8192	1.72×10^{10}	0.21	0.5^{32}
2	10	17	78	1953	2048	8192	1.72×10^{10}	0.30	0.5^{64}
2	10	17	78	1953	2048	8192	1.72×10^{10}	0.42	0.5^{128}
2	10	17	78	1953	2048	8192	1.72×10^{10}	0.60	0.5^{256}
3	3	10	88	64	162	1458	3.49×10^{09}	0.37	0.5^{16}
3	3	10	88	64	162	1458	3.49×10^{09}	0.52	0.5^{32}
3	4	11	267	208	486	4374	3.14×10^{10}	0.21	0.5^{16}
3	4	11	267	208	486	4374	3.14×10^{10}	0.30	0.5^{32}
3	4	11	267	208	486	4374	3.14×10^{10}	0.42	0.5^{64}
3	4	11	267	208	486	4374	3.14×10^{10}	0.59	0.5^{128}
4	1	8	31	9	48	768	4.29×10^{09}	0.60	0.5^{16}
4	2	8	139	45	192	3072	4.29×10^{09}	0.26	0.5^{16}
4	2	8	139	45	192	3072	4.29×10^{09}	0.37	0.5^{32}
4	2	8	139	45	192	3072	4.29×10^{09}	0.52	0.5^{64}
5	1	7	77	16	100	2500	6.10×10^{09}	0.33	0.5^{16}
5	1	7	77	16	100	2500	6.10×10^{09}	0.47	0.5^{32}

Table 3.1: Parameters for practical JL transforms matrix $A \in \mathbb{R}^{m \times n}$ that can projects any unit vector x such that $|\,\|Ax\|_2^2 - 1| < \epsilon$ with probability $1 - \delta$. The construction of A is based on an AG code from the u-th level GS-tower over $\mathbb{F}_{q^2}$ where q is a prime power. The parameters s, k, d and g correspond to the length, dimension, minimum distance and genus of the AG code, respectively. The matrix A is a sparse matrix and the number s is also the column sparsity of A. The parameter k is chosen such that $n = (q^2)^k$ is at least $m \times 10^6$ and ϵ is computed from (3.16).

In practice, the error probability δ should be as small as possible. For example, if $\delta = 0.5^{64}$ and there are $p = 2^{20}$ vectors, then JL transforms can project these vectors and nearly preserve their pairwise distances with at most $\delta' = \delta \cdot p^2/2 = 0.5^{15}$. Then such JL transforms may be useful. In Table 3.1, the error tolerance ϵ is relatively large comparing to δ. For example, if $\epsilon = 0.21$, it means a JL transform can projects any unit vector x such that $|\,\|Ax\|_2^2 - 1| < 0.21$ or $0.9 < \|Ax\|_2 < 1.1$.

It is possible some applications require a smaller ϵ. In this case, one can choose proper (ϵ, δ) pairs from (3.16).

Recall that the random JL transform matrix A has $m = q^2 s$ rows and $n = (q^2)^k$ columns. We choose k such that $n \geq m \times 10^6$, i.e., $k = \lceil \log_{q^2} (m \times 10^6) \rceil$. In practice, m is the size of stored vectors, thus $m < 10000$ may be useful. Therefore, the parameters q and u should be as small as possible.

Chapter 4

Conclusion and future problems

For RS codes, this thesis has proposed a new numerical decoding algorithm that is stable over complex numbers. From the noisy channel coding theorem, there exists a sequence of codes with increasing lengths whose bit error probabilities (i.e. the number of wrong bits/total number of transferred bits) go to zero. It is open to give a decoding algorithms with this property. However, our algorithm for RS codes does not achievement this goal. The error probability remains a constant as the code length increases. The main reason may be the sensitivity of computing of SVD of a large matrix.

For JL transforms, this thesis has improved the error bound from previous code-based construction by Kane and Nelson. Also, the thesis has given an explicit construction via AG codes with explicit parameters. It would be better for practical applications if one can reduce the code length (or the column sparsity of the transform matrix). This may be possible if the error probability bound is improved further.

Appendices

Appendix A Matlab codes for numerical decoding of RS codes

```
% This is the main procedure.
clc;clear;
trials=10000;
tic
result=zeros(0,8);
for q=[2,4]
for n=[8,16,32,64,128]
for rate=[0.5,0.75]
for sigma=[000.1,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
    k=n*rate;
    t=floor((n-k)/2);
    root=transpose(exp(2*pi*1i/n).^[1:n]);
    h=[1;zeros(n-1,1);-1];
    wrongsymbols=0;
    wrongcodes=0;
    for tt=1:trials
        m=randi([0,q-1],k,1);
        f=(exp(2*pi*1i/q)).^m;
        c=polyval(f,root);

        %generate gausian error
        e_matrix=normrnd(0,sigma,n,2);
        e=(e_matrix(:,1)+1i*e_matrix(:,2))/sqrt(2);



        %generate small error+cata error
        %a1=delta*rand(n,1);
        %phi1=2*pi*rand(n,1);
        %e=a1.*exp(phi1*1i);
```

```
        %ell=randi([0,t]);
        %e_indices=sort(randsample(n,ell));
        %a2=0.5*(rand(ell,1)+1);
        %phi2=2*pi*rand(ell,1);
        %e_values=a2.*exp(phi2*1i);
        %e(e_indices)=e_values;



        b=c+sqrt(k)*e;
        g=transpose(polyfit(root,b,n-1));
        M=sylv(g,h,k,t,1);
        [U,S,V]=svd(M);
        z=V(:,size(V,2)); %last column of V
        u=z(1:t+1);
        v=z(t+2:2*t+1);
        r=conv(g,u)+conv(h,v);
        r=r(length(r)-(k+t)+1:length(r));
        t=(length(z)-1)/2;
        G=toep(u,k);


        f_rec=lscov(G,r); %or f_rec=pinv(G)*r;
        m_rec=(angle(f_rec))/(2*pi/q);
        m_rec=round(m_rec);
        m_rec=mod(m_rec,q);
        wrongsymbols=wrongsymbols+nnz(m-m_rec);
        wrongcodes=wrongcodes+(norm(m-m_rec)>0);
    end
    ser=wrongsymbols/(n*rate*trials);
    result=[result;[q,n,rate,sigma,trials,wrongcodes,wrongsymbols,ser]];
end
end
```

```
end
end

toc
dlmwrite('E:\result.txt', result, 'newline', 'pc', 'delimiter', '\t','-append');




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%transmit a column complex vector
%keep d decimal places
% trunc.m
%transmit a column complex vector
%keep d decimal places

function [y]=trunc(x,d)
n=length(x);
y=zeros(n,1);
for i=1:n
    real_y=round(real(x(i))*10^(d))/(10^d);
    imag_y=round(imag(x(i))*10^(d))/(10^d);
    y(i)=real_y+1i*imag_y;
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%input: column vector u of length (t+1), shift k times
%output:(k+t)*k toeplitz matrix G
%such that, for any column vector f of length k, G*f=conv(u,f)

function [G]=toep(u,k)
```

```
t=length(u)-1;
u=flipud(u);
u=transpose(u);
G=zeros(k+t,k+2*t);
for j=1:k+t
G(j,j:t+j)=u;
end
G=G(:,t+1:k+t);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate full/partial Sylvester matrix
% Input:
% n-dimensional coefficients column vector g of g(x) in descending order
% (n+1)-dimensional coefficients column vector of h(x) in descending order
% t
% Output:
% if the input partial=1 then output
%(n-k)*(2t+1) dimensional sylvester matrix M
% otherwise output full sylvester matrix
function [M]=sylv(g,h,k,t,partial)
n=length(g);
M1=zeros(n+t,t+1);
M2=zeros(n+t,t);
for j=0:t
    M1(1+j:n+j,1+j)=g;
end
for j=0:(t-1)
    M2(1+j:n+1+j,1+j)=h;
end
M=[M1,M2];
```

```
if partial==1
M=M(1:n-k,:);
end
end
```

Bibliography

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671 – 687, 2003. Special Issue on {PODS} 2001.

[2] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput*, pages 302–322, 2009.

[3] Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual bch codes. Technical report, 2007.

[4] Ilia Aleshnikov, P. Vijay Kumar, Kenneth W. Shum, and Henning Stichtenoth. On the splitting of places in a tower of function fields meeting the drinfeld-vladut bound. *IEEE Transactions on Information Theory*, 47(4):1613–1619, 2001.

[5] Richard Baraniuk, Mark Davenport, Ronald Devore, and Michael Wakin. A simple proof of the restricted isometry property for random matrices. *Constr. Approx*, 2008, 2007.

[6] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 301–309, New York, NY, USA, 1988. ACM.

[7] E. R. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470.

[8] Thomas Blumensath and Mike E. Davies. Iterative hard thresholding for compressed sensing. *CoRR*, abs/0805.0510, 2008.

[9] E. Candes. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9-10):589–592, May 2008.

[10] Emmanuel Candes and Terence Tao. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 12 2007.

[11] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[12] Emmanuel J. Candés and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[13] Albert Cohen, Wolfgang Dahmen, and Ronald Devore. Compressed sensing and best k-term approximation. *J. Amer. Math. Soc*, pages 211–231, 2009.

[14] A. d'Aspremont and L. El Ghaoui. Testing the nullspace property using semidefinite programming. *Math. Progr.*, February 2010. Special issue on machine learning.

[15] Ilias Diakonikolas, Daniel M. Kane, and Jelani Nelson. Bounded independence fools degree-2 threshold functions. *CoRR*, abs/0911.3389, 2009.

[16] David L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.

[17] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In *Communications, Information and Network Security, V.Bhargava, H.V.Poor, V.Tarokh, and S.Yoon*, pages 55–68. Kluwer, 2002.

[18] A. Garcia and H. Stichtenoth. On the Asymptotic Behaviour of Some Towers of Function Fields over Finite Fields. *Journal of Number Theory*, pages 248–273, December 1996.

[19] Mark Giesbrecht, George Labahn, and Wen shin Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. In Barry M. Trager, editor, *ISSAC*, pages 116–123. ACM, 2006.

[20] V. D. Goppa. Codes on algebraic curves. *Dokl. Akad. Nauk SSSR*, 259(6):1289–1290, 1981.

[21] Dima Yu. Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in nc. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:166–172, 1987.

[22] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[23] David Lee Hanson and Farroll Tim Wright. A bound on tail probabilities for quadratic forms in independent random variables. *Ann. Math. Statist.*, 42(3):1079–1083, 1971.

[24] P. Indyk. *High-dimensional computational geometry.* PhD thesis, Stanford University, 2000.

[25] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[26] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[27] E. Kaltofen, Y. N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 135–139, New York, NY, USA, 1990. ACM.

[28] Erich Kaltofen and Wen shin Lee. Early termination in sparse interpolation algorithms. *Journal of Symbolic Computation*, 36(3C4):365 – 400, 2003. {ISSAC} 2002.

[29] Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, January 2014.

[30] Felix Krahmer and Rachel Ward. New and improved johnson-lindenstrauss embeddings via the restricted isometry property. *CoRR*, abs/1009.0744, 2010.

[31] Michael Lustig, David L. Donoho, Juan M. Santos, and John M. Pauly. Compressed sensing mri. In *IEEE SIGNAL PROCESSING MAGAZINE*, 2007.

[32] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-holland Publishing Company, 2nd edition, 1978.

[33] Jirí Matousek. On variants of the johnson-lindenstrauss lemma. *Random Struct. Algorithms*, 33(2):142–156, 2008.

[34] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, April 1995.

[35] Deanna Needell and Joel A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Commun. ACM*, 53(12):93–100, December 2010.

[36] Irving Reed and Golomb Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(2):300–304, 06/1960 1960.

[37] Mark Rudelson. Sparse reconstruction by convex relaxation: Fourier and gaussian measurements. In *CISS 2006 (40th Annual Conference on Information Sciences and Systems*, 2006.

[38] Akira Shiozaki. Decoding of redundant residue polynomial codes using euclid's algorithm. *IEEE Transactions on Information Theory*, 34(5):1351–1354, 1988.

[39] Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997.