The transmission of digital data is one of the principal tasks in modern wireless communication. Classically, the communication channel consists of one transmitter and one receiver; however, due to the constantly increasing demand in higher transmission rates, the popularity of using several receivers and transmitters has been rapidly growing.
In this paper, we combine a number of fairly standard techniques from numerical linear algebra and probability to develop several (apparently novel) randomized schemes for the decoding of digital messages sent over a noisy multivariate Gaussian channel.
We use a popular mathematical model for such channels to illustrate the performance of our schemes via numerical experiments.

On decoding of digital data sent over a noisy MIMO channel

Andrei Osipov
Research Report YALEU/DCS/TR-1503
Yale University
January 6, 2015

Approved for public release: distribution is unlimited.
Keywords: *MIMO, decoding schemes, noisy Gaussian channels*

Contents

1 Introduction

The importance of wireless communication can hardly be overestimated; our life nowadays is unimaginable without it. For obvious reasons, the transmission of digital (as opposed to analog) data is one of the principal tasks in wireless communication. In the case when there is one transmitter and one receiver, the subject has been extensively studied for decades. More recently, however, as more and more data are collected and need to be rapidly transmitted, there has been an increasing demand in ways to boost communication performance; as a result, much research is being done on the use of multiple receivers/transmitters. This topic (generally referred to as multiple-input and multiple-output, or MIMO) is now at the frontier of the research in modern communications (see e.g. [2], [5], [7], [11], [12], [13], [14], [17], [19]).

The transmission of digital data over a MIMO channel is typically described by the following model (see e.g. [11], [19]). Suppose that $m > 0$ is an integer power of 2, and that C is a collection of m points in the complex plane (C is usually referred to as a "constellation"); each element of C corresponds to a unique binary word of length $\log_2(m)$.

Suppose also that there are n transmitters and n receivers, and that $X = C^m$ is the collection of all possible vectors in $\mathbb{C}^m$ whose coordinates belong to C (obviously, X contains m^n vectors). Suppose, in addition, that $\sigma > 0$ is a real number, and that H is an n by n complex matrix (the "channel matrix"). For any x in X, we define the random $n-$dimensional complex vector $y(x)$ via the formula

$$y(x) = H \cdot x + \sigma \cdot (z_1 + i \cdot z_2)\,, \tag{1}$$

where z_1, z_2 are independent standard normal random vectors, and $i = \sqrt{-1}$. Here x represents the transmitted message (that encodes $n \cdot \log_2(m)$ binary bits), $H \cdot x$ represents the received message in the absence of noise, and $y(x)$ represents the received message corrupted by Gaussian noise of component-wise standard deviation σ.

In this model, the decoding problem can be formulated as follows: suppose that x_{true} is the transmitted message, and that y is the received message (i.e. the observed value of $y(x_{\text{true}})$). Under the assumption that H and σ are known, one needs to find x_{best} in X such that

$$\|H \cdot x_{\text{best}} - y\| \leq \|H \cdot x - y\|, \tag{2}$$

for any x in X. In other words, x_{best} is the maximum likelihood estimate of x_{true}.

In modern applications, any practical decoding algorithm must be quite rapid. For instance, the peak speed requirement for 4G mobile telecommunication technology is 1 Gigabit per second, and the 100 Gb/s RF Backbone DARPA project aims at the rate of 100 Gigabits per second (using optical wireless communication). To complicate the matters, the channel matrix H needs to be frequently reevaluated (several times per second).

In principle, the decoding problem can be solved by brute force, i.e. by iteratively testing all x in X. However, X contains m^n vectors, which is a large number even for small values of m and n (e.g. $m = n = 8$); this makes the brute force approach impractical.

Another approach is based on the observation that, due to (24), $H \cdot x_{\text{best}}$ is the nearest neighbor of y in the collection

$$HX = \{H \cdot x \ : \ x \in X\} \tag{3}$$

of m^n complex vectors in $\mathbb{C}^n$; thus, a fast nearest neighbors algorithm might be used to find x_{best}. Such algorithms typically pre-process HX to obtain auxiliary data structures; then, a single y can be decoded reasonably fast (see e.g. [16]). However, in a typical modern application HX contains between 10^7 and 10^{10} vectors, which makes the initial pre-processing of HX unaffordable in terms of both memory requirements and CPU time (see e.g. [16]), especially considering that this calculation has to be redone every time that H changes. In other words, even the fastest *generic* nearest neighbor search in HX might not be fast enough.

Yet another approach is based on the observation that, as opposed to HX, the collection X does not depend on H; moreover, it has a special structure that allows for fast nearest neighbor searches *in* X. Thus, one can compute $\tilde{x}$ via the formula

$$\tilde{x} = H^{-1} \cdot y, \tag{4}$$

and look for x_{best} among several nearest neighbors of $\tilde{x}$ within X. Unfortunately, this way the baby gets often thrown out with the bath water: x_{best} will typically *not* be among any reasonable number of nearest neighbors of $\tilde{x}$ unless σ is significantly smaller than the minimal singular value λ_n of H. Suppose, for example, that $n = 8$, that C the so-called 8-PSK constellation of size $m = 8$ (equispaced points on a unit circle), that H is a randomly generated complex Gaussian matrix, and that $\sigma = 0.75$ (i.e. roughly twice the expected minimal singular value of such a matrix). Then, x_{best} will coincide with x_{true} in about half of all cases; yet, in more than 70% of these cases x_{best} will not be among even as many as 500 nearest neighbors of $\tilde{x}$ in X.

Some other decoding schemes and approaches can be found, for example, in [2], [11], [12], [13], [14] (see also references therein).

In this project, we develop several randomized decoding schemes. Our schemes are based on the observation that, in typical applications, the channel matrix H is not large (e.g. $n = 8$ or $n = 16$), and thus one can afford to evaluate its singular value decomposition (SVD). Then, we use the SVD of H to find x_{best} by a rapid randomized test-and-trial procedure. Our tentative algorithms demonstrate reasonably good performance in several test cases (see Section 5), even when $\sigma = O(\lambda_n)$. The schemes break down only when σ is so large that the estimate x_{best} defined via (24) differs from the transmitted message x_{true} with high probability (e.g. it is impossible to recover x_{true} without additional assumptions).

This paper is organized as follows. Section 2 contains mathematical preliminaries to be used in the rest of the paper. Section 3 contains a number of related theoretical results. In Section 4, we describe the decoding problem and present several decoding schemes. In Section 5, we illustrate the performance of our decoding schemes via numerical experiments. In Section 6, we present some conclusions and outline possible directions of future research.

2 Mathematical Preliminaries

In this section, we introduce notation and summarize several facts to be used in the rest of the paper.

2.1 Probability

In this section, we summarize some well known facts from the probability theory. These facts can be found, for example, in [1], [4], [6], [8].

Suppose that $x > 0$ is a real number. In agreement with the standard practice, we define the gamma function by the formula

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt. \tag{5}$$

The one-dimensional standard Gaussian distribution $N(0,1)$ with mean zero and standard deviation one is defined by its probability density function (pdf)

$$f_{N(0,1)}(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}, \quad -\infty < t < \infty. \tag{6}$$

Its cumulative distribution function (cdf) is given by the formula

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt = 1 - \frac{1}{2} \cdot \operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right), \tag{7}$$

where $\operatorname{erfc} : \mathbb{R} \to \mathbb{R}$ is the complementary error function defined via the formula

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt, \tag{8}$$

for all real x.

Suppose that $d > 0$ is a positive integer. We say that the random vector X has standard Gaussian d-dimensional distribution $N(0_d, I_d)$, if all of its coordinates are independent standard Gaussian random variables.

Suppose now that $X \sim N(0_d, I_d)$. Then $\|X\|^2$ has distribution[1] $\chi^2(d)$ with pdf

$$f_{\chi^2(d)}(t) = \frac{t^{d/2-1} \cdot e^{-t/2}}{2^{d/2} \cdot \Gamma(d/2)}, \quad t > 0, \tag{9}$$

where Γ denotes the gamma function defined via (5) above. In particular, if $n > 0$ is a positive integer, then

$$f_{\chi^2(2n)}(t) = \frac{t^{n-1} \cdot e^{-t/2}}{2^n \cdot (n-1)!}, \tag{10}$$

for all $t > 0$, and the corresponding cdf $F_{\chi^2(2n)}$ admits the form

$$F_{\chi^2(2n)}(x) = 1 - e^{-x/2} \cdot \sum_{k=0}^{n-1} \frac{x^k}{2^k \cdot k!}, \tag{11}$$

[1] Chi-square with d degrees of freedom.

for all $x > 0$.

The real-valued Gaussian random variable has a straightforward generalization to the complex plane. Suppose that $X, Y \sim N(0,1)$ are i.i.d. standard normal variables. We say that the random variable Z defined via the formula

$$Z = X + i \cdot Y \tag{12}$$

is a standard normal complex variable, and we denote its distribution by $N\mathbb{C}(0,1)$.[2] Similarly, if $d > 0$ is a positive integer, we say that the complex $d-$dimensional random vector Z_d has complex standard distribution $N\mathbb{C}(0_d, I_d)$ if all of its coordinates are i.i.d. standard normal complex variables. Needless to say, in this case

$$\|Z_d\|^2 \sim \chi^2(2 \cdot d). \tag{13}$$

The following well known theorem states the fact that Gaussian distribution is invariant under orthogonal transformations.

Theorem 1. *Suppose that $d > 0$ is a positive integer, that U is the d by d unitary matrix, and that the complex $d-$dimensional random vector Z_d has distribution $N\mathbb{C}(0_d, I_d)$. Then,*

$$U \cdot Z_d \sim N\mathbb{C}(0_d, I_d). \tag{14}$$

The following theorem describes some additional properties of complex normal random variables.

Theorem 2. *Suppose that $n > 0$ is a positive integer, and that $a_1, \ldots, a_n$ are complex numbers. Suppose also that $z_1, \ldots, z_n$ are i.i.d. standard normal complex random variables. Then,*

$$a_1 \cdot z_1 + \cdots + a_n \cdot z_n \sim N\mathbb{C}(0,1) \cdot \sqrt{|a_1|^2 + \cdots + |a_n|^2}. \tag{15}$$

2.2 Linear Algebra

The following widely known theorem can be found, in a more general form, in most standard books on linear algebra (see, for example, [9], [18]).

Theorem 3. *Suppose that $n > 0$ is a positive integer, and that H is an n by n complex matrix. Then, there exist non-negative real numbers $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ and n by n unitary matrices U and V such that*

$$H = U \cdot \Sigma \cdot V^*, \tag{16}$$

where V^* denotes the conjugate transpose of V, and Σ is the diagonal n by n matrix whose diagonal entries are defined via the formula

$$\Sigma_{i,i} = \sigma_i, \tag{17}$$

[2] in some sources, the same distribution might be denoted $N\mathbb{C}(0,2)$ or $N\mathbb{C}(0,2;0)$.

for every $i = 1, \ldots, n$. The columns $u_1, \ldots, u_n$ of U are referred to as left-singular vectors, the columns $v_1, \ldots, v_n$ of V are referred to as right-singular vectors, and $\sigma_1, \ldots, \sigma_n$ are called the singular values of H. The factorization (16) *is typically referred to as the singular value decomposition (SVD) of H.*

3 Analytical Apparatus

The purpose of this section is to provide the analytical apparatus to be used in the rest of the paper.

Theorem 4. *Suppose that $n > 0$ is a positive integer, that H is an n by n complex regular matrix, and that the matrices U, Σ, V constitute the SVD of H, as in* (16) *of Theorem 3. Suppose, in addition, that $w \sim N\mathbb{C}(0_d, I_d)$ is the standard normal complex random vector in $\mathbb{C}^d$. Then,*

$$H^{-1} \cdot w \sim \frac{z_1}{\sigma_1} \cdot v_1 + \cdots + \frac{z_n}{\sigma_n} \cdot v_n, \tag{18}$$

where the real numbers $\sigma_1, \ldots, \sigma_n$ are the singular values of H, the $n-$dimensional complex vectors $v_1, \ldots, v_n$ are the columns of V, and $z_1, \ldots, z_n$ are i.i.d. standard complex normal random variables, as in (12)*.*

Proof. Due to (16) in Theorem 3,

$$H^{-1} = V \cdot \Sigma^{-1} \cdot U^*, \tag{19}$$

where Σ^{-1} is the diagonal matrix whose diagonal entries are $\sigma_1^{-1}, \ldots, \sigma_n^{-1}$. We combine (14) in Theorem 1 with (19) to obtain (18). ■

Corollary 1. *Suppose, in addition to the hypothesis of Theorem 4, that $1 \leq i \leq n$ is an integer. Then,*

$$\left(H^{-1} \cdot w\right)(i) \sim z_1 \cdot \sqrt{\sum_{k=1}^{n-1} \frac{|v_k(i)|^2}{\sigma_k^2}} + z_n \cdot \frac{|v_n(i)|}{\sigma_n}, \tag{20}$$

where, for any complex vector v in $\mathbb{C}^n$, we denote by $v(i)$ its ith coordinate.

Proof. We combine Theorem 2 with (19) to obtain (20). ■

4 Numerical Algorithms

In this section, we describe several numerical algorithms for the decoding of a digital signal sent over a noisy MIMO channel, in the sense described in Section 4.1 below.

4.1 Principal Decoding Task

In this section, we provide a formal description of the principal decoding task used in the rest of this paper (see e.g. [2], [11], [12], [13], [14]).

Suppose that $n > 0$ and $m > 0$ are integers, that

$$C = \{c_1, \ldots, c_m\} \tag{21}$$

is a collection of m points in the complex plane (often referred to as "constellation" in the literature), that H is an n by n complex matrix, and that $\sigma > 0$ is a real number. Suppose also that X is the collection of m^n vectors in $\mathbb{C}^n$ defined via the formula

$$X = \left\{ (x(1), \ldots, x(n))^T \ : \ x(1), \ldots, x(n) \in C \right\}, \tag{22}$$

and that x_{true} is a vector in X (we think of x_{true} as of the unknown transmitted message). Suppose also that, for every x in X, the random complex $n-$dimensional vector $y(x)$ is defined via the formula

$$y(x) = H \cdot x + \sigma \cdot Z_n, \tag{23}$$

where $Z_n \sim N\mathbb{C}(0_d, I_d)$ is the standard normal $d-$dimensional complex random vector (see Section 2.1 above).

Task. Suppose that x_{true} in X is the transmitted message, and that y_{obs} in $\mathbb{C}^n$ is the observed value of $y(x_{\text{true}})$. Given X, H, σ and y_{obs}, one needs to find x_{true}. More specifically, one needs to find x_{best} in X such that

$$\|H \cdot x_{\text{best}} - y_{\text{obs}}\| \leq \|H \cdot x - y_{\text{obs}}\|, \tag{24}$$

for any x in X. In other words, x_{best} is the maximum likelihood estimate of x_{true}.

Remark 1. *Obviously, for any fixed H and any x_{true} in X, the probability that x_{best} defined via (24) is equal to x_{true} depends on σ (more specifically, this probability decreases as σ increases). In particular, for noise of sufficiently large coordinate-wise standard deviation σ, the best likelihood estimate is unlikely to coincide with the transmitted message.*

In the view of Remark 1, we make the following observations.

1. If x_{best} defined via (24) coincides with x_{true}, a numerical scheme for decoding y_{obs} should be able to recover x_{true} (the task is well defined).

2. If x_{best} defined via (24) is different from x_{true}, then there is not enough information for recovering x_{true} (without additional assumptions). In that case, it might still be desirable for a decoding numerical scheme to recover x_{best}.

4.2 Running Example

For the sake of concreteness, we introduce the following example (by specifying typical values of parameters of the model from Section 4.1), to be used in the rest of this paper.

1. The dimensionality n of the channel matrix (the number of transmitters and receivers):

$$n = 6, 7, 8. \tag{25}$$

While the value $n = 8$ seems to be typical in many application, the schemes should handle values up to $n = 16$ or even $n = 32$.

2. The number m of constellation points:

$$m = 8. \tag{26}$$

While constellations of size $m = 8$ are frequently used in applications, many popular constellations contain $m = 16$ or $m = 32$ points, and these values should be kept in mind.

3. The constellation C:

$$C = \left\{ e^{i \cdot \pi \cdot k/4} \ : \ 0 \le k < 8 \right\} \tag{27}$$

(the so-called 8-PSK constellation). In other words, C consists of $m = 8$ points equally distributed on the unit circle.

4. The channel matrix H: in agreement with a common practice, we assume that the n^2 entries of H have been independently drawn from the standard normal complex distribution $N\mathbb{C}(0, 1)$. In other words,

$$H(i, j) \sim N\mathbb{C}(0, 1), \tag{28}$$

for every $i, j = 1, \ldots, n$.

In the rest of this section, we describe several decoding schemes. Some of them are well known; others are apparently novel. All of these schemes receive the observed value y_{obs} of $y(x_{\text{true}})$ as an input and look for x_{best} defined via (24).

4.3 Brute force algorithm

The brute force algorithm locates x_{best} by searching through all of X. In other words, it consists of the following steps:

Step 1. For every x in X, evaluate the real number $d(x)$ via the formula

$$d(x) = \|H \cdot x - y_{\text{obs}}\|. \tag{29}$$

Step 2. Define x_{best} via finding the minimum among all $d(x)$, i.e.

$$d\left(x_{\text{best}}\right) \le d(x), \tag{30}$$

for all x in X. Obviously, (30) is equivalent to (24).

Memory requirements. While the collection X is typically quite large (it contains m^n vectors – see (22)), all the vectors in X can be iteratively computed one by one in a straightforward way, thus obviating the need to pre-compute X and store it in memory (as far as the brute force algorithm is concerned). Hence, the memory requirements of the brute force algorithm are minimal in the sense that it requires only

$$M_{\text{brute}} = O\left(n^2 + m \cdot n\right) \tag{31}$$

memory words. In other words, practically speaking, the brute force algorithm does not require any memory to speak of (as m, n are typically very small: see Section 4.2).

Cost. The cost of the brute force algorithm is

$$C_{\text{brute}} = O\left(|X| \cdot n^2\right) = O\left(m^n \cdot n^2\right) \tag{32}$$

operations.

Success rate. The brute force algorithm always locates x_{best} and, in this sense, is optimal.

Remark 2. *Even for as small values of m and n as $m = n = 8$ (see Section 4.2), the brute force algorithm requires about $8^{10} \approx 10^9$ operations. Since in a typical application one needs to decode thousands or even millions messages per second, its high cost deems the brute force decoding algorithm impractical.*

Conclusion. While the brute force algorithm is useless in practical applications, it is a reasonable (albeit slow) testing tool, due to its 100% success rate.

We conclude by summarizing the principal input and output parameters of the brute force algorithm described above.

Calling sequence:

$$\text{brute}(y_{\text{obs}}; x_{\text{best}}). \tag{33}$$

Input parameters:

– y_{obs} in $\mathbb{C}^n$:
the received message (a noisy observation of $y(x_{\text{true}})$ defined via (23)).

Output parameters:

– x_{best} in X:
the maximal likelihood estimate of x_{true} (see (24)).

4.4 Nearest Neighbors in $H \cdot X$

Obviously, the vector $H \cdot x_{\text{best}}$ (see (24)) is simply the nearest neighbor of y_{obs} within the collection HX of vectors in $\mathbb{C}^n$ defined via the formula

$$HX = \{H \cdot x \ : \ x \in X\}. \tag{34}$$

This observation suggests using a fast nearest neighbors algorithm for computing x_{best}, such as, for example, the randomized approximate nearest neighbors algorithm (RANN) described in [15], [16] (needless to say, the straightforward nearest neighbors search is simply the algorithm from Section 4.3). Obviously, a nearest neighbor search in $\mathbb{C}^n$ is equivalent to that in $\mathbb{R}^{2n}$.

RANN consists of two steps briefly described below (the reader is referred to [15] for a more detailed description, and to [3] for another nearest neighbors algorithm):

Preprocessing (depends on H but not on y_{obs}). One constructs a tree-like structure on HX, recursively subdividing the points in HX along each of $2n$ randomly chosen real axes. The resulting "tree of boxes" consists of 2^{2n} boxes, each containing $(m/4)^n$ points on average. Typically, several random subdivisions are constructed (one can also further refine each tree by subdividing boxes that contain too many points).

Query for nearest neighbors of $y_{\mathbf{obs}}$. Once the auxiliary tree of boxes on HX has been constructed, the search for nearest neighbors of any given y_{obs} is done as follows. First, one locates the box that y_{obs} belongs to; then, the points in this box (and, possibly, in several boxes nearby) are inspected one by one.

Success rate. The success rate depends on such parameters as the number of trees used, their internal structure (e.g. the number of subdivisions in each tree, average number of points in a box), etc. In particular, there is a trade-off between probability of locating the nearest neighbor of y_{obs} and required CPU time. However, in this environment one can typically achieve success rate of 90% or higher without increasing CPU time too much; to that end, the parameters of the nearest neighbor search are best tuned empirically.

Memory requirements. Storing the data structures constructed in the pre-processing step requires

$$M_{\text{preprocess}} = O\left(m^n\right) \tag{35}$$

memory words per a single tree. In particular, even for such small values of m and n as $m = n = 8$ (see Section 4.2), the memory requirements can easily exceed several Gigabytes.

Cost. The cost of a single query for the nearest neighbor of y_{obs} is proportional to the depth of the tree and the number of points in a box; in particular, it will be of the order

$$C_{\text{query}} = O\left(n^2 \cdot \log_2(m)\right) \tag{36}$$

(see e.g. [15]). In other words, running time of a single query is likely to be reasonable (compare to (32), and see also Section 4.2). However, the cost of the pre-processing step will be at least

$$C_{\text{preproces}} = O\left(m^n\right) \tag{37}$$

operations. While this step does not need to be redone for each new y_{obs}, one still need to pre-process the data from scratch every time the channel matrix H changes. In particular, in any application where H is frequently re-evaluated, this creates an additional major obstacle to using such algorithms.

Conclusion. We strongly suspect that *any* generic nearest neighbors algorithm (used as a black box) will be impractical for this decoding problem, in terms of either memory requirements or CPU time (or both), simply due to the sheer amount of points in HX. If, in addition, the channel matrix H changes frequently, such approaches are likely to be completely unaffordable.

4.5 Nearest Neighbors in X

In Section 4.4 above, we discussed nearest neighbors searches within the collection HX defined via (34). In comparison, for any vector $y \in \mathbb{C}^n$ and integer $1 \leq k < n$, to find k nearest neighbors of y within X is a much easier task, for the following reasons:

1. Due to (22), for any x in X,

$$\|y - x\|^2 = |y(1) - x(1)|^2 + \cdots + |y(n) - x(n)|^2, \tag{38}$$

where $x(1), \ldots, x(n)$ are in C. Thus, the search for nearest neighbors is reduced to n one-dimensional searches. For example, to find the first nearest neighbor of y in X one simply needs to minimize $|y(j) - x(j)|$ separately for every $j = 1, \ldots, n$.

2. The collection X enjoys various symmetries that can be employed in nearest neighbor searches. For example, if C is defined via (27), and one has found the first k nearest neighbors of

$$x_0 = (1, \ldots, 1)^T \tag{39}$$

within X, then, for any x in X and every $1 \leq j \leq k$, the jth nearest neighbor of x within X is computed from that of x_0 by coordinate-wise rotation (in n operations). In other words, if x_j is the jth nearest neighbor of x_0, then the jth nearest neighbor $\hat{x}_j$ of $\hat{x}$ is defined via the formula

$$\hat{x}_j = (x_j(1) \cdot \hat{x}(1), \ldots, x_j(n) \cdot \hat{x}(n)) \,. \tag{40}$$

3. As opposed to HX, the collection X obviously does not depend on H, and thus any pre-computation on X needs to be done only once and for all.

These observations suggest the following algorithm for decoding y_{obs}:

Step 1. Select a positive integer $1 \leq k < m^n$.

Step 2. Compute the vector $\tilde{x}_{\text{obs}}$ in $\mathbb{C}^n$ via the formula

$$\tilde{x}_{\text{obs}} = H^{-1} \cdot y_{\text{obs}}. \tag{41}$$

Step 3. Find the k nearest neighbors $x_1, \ldots, x_k$ of $\tilde{x}_{\text{obs}}$ within X.

Step 4. For every $j = 1, \ldots, k$, evaluate d_j via the formula

$$d_j = \|H \cdot x_j - y_{\text{obs}}\|^2, \tag{42}$$

and find the index j_0 that corresponds to the minimal d_j.

Step 5. Return $x_{\text{nn}(k)}$ defined via the formula

$$x_{\text{nn}(k)} = x_{j_0} \tag{43}$$

(obviously, the subscript k refers to the number of nearest neighbors used to evaluate $x_{\text{nn}(k)}$).

The vector $x_{\text{nn}(k)}$ computed by this algorithm is an approximation to x_{best} defined via (24).

We summarize the principal input and output parameters of the algorithm described in this section as follows.

Calling sequence:

$$\text{nnx}(y_{\text{obs}}; k; \tilde{x}_{\text{obs}}, x_{\text{nn}(k)}). \tag{44}$$

Input parameters:

– y_{obs} in $\mathbb{C}^n$:
the received message (a noisy observation of $y(x_{\text{true}})$ defined via (23)).

– k (a positive integer):

the number of nearest neighbors to be used.

Output parameters:
– $\tilde{x}_{\text{obs}}$ in X:
the "naive" candidate for x_{true} (see (41)).
– $x_{\text{nn}(k)}$ in X:
the candidate for x_{true} found by nearest neighbors search in X (see (43)).

Memory requirements. The memory requirements for this algorithm are minimal; one might only need to store H and the list of k nearest neighbors of $\tilde{x}_{\text{obs}}$ in X, which requires

$$M_{\text{nn}} = O(k \cdot n + n^2) \tag{45}$$

memory words. Compared, for example, to (35), this is essentially negligible.

Cost. It costs $O(n^3)$ operations to invert H; however, if, for example, we have evaluated the SVD of H beforehand, the evaluation of $\tilde{x}_{\text{obs}}$ via (41) costs only $O(n^2)$ operations. The evaluation of $x_1, \ldots, x_k$ will typically require $O(n \cdot k)$ operations, and, for every j, the evaluation of d_j via (42) requires $O(n^2)$ operations; thus, the total cost of the algorithm is

$$C_{\text{nn}} = O\left(k \cdot n^2\right) \tag{46}$$

operations.

Success rate. Unfortunately, unless H is close to an orthogonal matrix or σ in (23) is small compared to the smallest singular value of H (see Theorem 3), the success rate of this scheme will typically be poor (loosely speaking, due to Theorem 4 – in other words, H^{-1} magnifies the noise by different factors in different directions). In the case of the model described in Section 4.2, the success rate of this scheme is investigated empirically in Section 5 below.

4.6 Randomized Decoder 1 (RaDe1)

In this section, we described a randomized decoding scheme based on Theorem 4 and Corollary 1. This scheme will be referred to as "Randomized Decoder 1", or "RaDe1".

4.6.1 Preliminary discussion

Suppose that U, Σ, V constitute the SVD of H (see Theorem 3). Suppose also that the random vector $y(x_{\text{true}})$ is defined via (23), and that the random vector $\tilde{x}$ in $\mathbb{C}^n$ is defined via the formula

$$\tilde{x} = H^{-1} \cdot y(x_{\text{true}}). \tag{47}$$

Due to the combination of (47) with Theorems 3, 4,

$$\tilde{x} \sim x_{\text{true}} = \sum_{k=1}^{n} z_k \cdot \frac{\sigma}{\sigma_k} \cdot v_k, \tag{48}$$

where $v_1, \ldots, v_n$ are the columns of V, $\sigma_1, \ldots, \sigma_n$ are the singular values of H, and $z_1, \ldots, z_n$ are i.i.d. standard complex normal variables.

Observation 1. The identity (48) admits the following interpretation: while the distribution of $y(x_{\text{true}})$ is radially symmetric about $H \cdot x_{\text{true}}$, the distribution of $\tilde{x}$ is not. More specifically, the "noise" in $\tilde{x}$ has the largest variance in the direction of v_n and the smallest variance in the direction of v_1. In other words, z_n has larger effect on how far $\tilde{x}$ is from x_{true} that z_1 does.

Suppose now that, for every $j = 1, \ldots, n$ and every $k = 1, \ldots, n$, we define the real numbers $s_k(j)$ and $S_k(j)$ via the formulae

$$s_k(j) = \frac{|v_k(j)|}{\sigma_k}, \tag{49}$$

$$S_k(j) = \sqrt{\sum_{l=1}^{k} s_l^2(j)}, \tag{50}$$

respectively. It follows from the combination of (49), (50) and Corollary 1 that

$$\tilde{x}(j) \sim x_{\text{true}}(j) + z \cdot \sigma \cdot S_{n-1}(j) + z_n \cdot \sigma \cdot s_n(j), \tag{51}$$

for every $j = 1, \ldots, n$, where z, z_n are i.i.d. $N\mathbb{C}(0,1)$ (obviously, $\tilde{x}(1), \ldots, \tilde{x}(n)$ are not independent of each other).

The coefficients $S_{n-1}(j)$ and $s_n(j)$ in (51) are determined by H. In the case when the entries of H have been drawn independently from $N\mathbb{C}(0,1)$ (see Section 4.2), we can make the following observation.

Observation 2. Suppose that $6 \leq n \leq 32$ is an integer, that the entries of H are i.i.d. $N\mathbb{C}(0,1)$, and that the real random variable $r = r(H)$ is defined via the formula

$$r(H) = \frac{s_n(j)}{S_{n-1}(j)}, \tag{52}$$

where, for any H, $1 \leq j \leq n$ is the integer such that $S_{n-1}(j) \leq S_{n-1}(i)$ for every $i = 1, \ldots, n$. Then,

$$0.25 \leq \mathbb{E}\left[S_{n-1}(j)\right] \leq 0.3, \tag{53}$$

and also

$$4.5 \leq \mathbb{E}\left[r(H)\right] \leq 6. \tag{54}$$

In other words, the coefficient in front of z_n in (51) is about 5 times larger (on average) than that in front of z.

4.6.2 RaDe1: informal description

The discussion in Section 4.6.1 above leads to the following decoding scheme (based primarily on (51) and Observation 2 above).

We select $1 \leq j \leq n$ such that $S_{n-1}(j)$ in (51) is minimal. Then, for every $i = 1, \ldots, m$, we assume that

$$x_{\text{true}}(j) = c_i, \tag{55}$$

and proceed as follows. Under the assumption (55), the difference $\tilde{x}(j) - x_{\text{true}}(j)$ is a sum of two independent complex normal variables with different variances (see (51)). We sample one of them; more specifically, we draw the complex number $\tilde{z}$ from the distribution $N\mathbb{C}(0, 1)$. The assumptions (55) and $z = \tilde{z}$ determine the value of z_n in (51). However, we observe that z_n is "responsible" for the largest component of the noise in $\tilde{x}$, due to (48). Thus, we can reduce the noise by subtracting $z_n \cdot \sigma \cdot \sigma_n^{-1} \cdot v_n$ from the observed value of $\tilde{x}$. Next, we look for the nearest neighbor x_i of the "improved" $\tilde{x}$ in X (a simple task; see Section 4.5). We observe that, provided that the assumption (55) is correct and the noise in $\tilde{x}$ has been reduced, x_i is more likely to be equal to x_{true} than if we were to use the nearest neighbor search in X for the original $\tilde{x}$.

To validate the assumption (55), we make several observations.

1. For each $i = 1, \ldots, m$, the assumption (55) yields x_i. We can simply select the best x_i out of $x_1, \ldots, x_m$ (the one for which $\|H \cdot x_i - y_{\text{obs}}\|$ is the smallest).

2. If $x_i = x_{\text{true}}$, then we can determine the noise from (23); in particular, we can compute the square of the Euclidean norm of the noise. However, this quantity must be an observed value of a $\chi^2(2n)$ random variable (multiplied by σ^2), whose cdf is defined via (11); we use it to estimate our "confidence" in the statement that $x_i = x_{\text{true}}$.

3. The value of z_n must be an observed value of a standard normal complex variable. In particular, $|z_n|^2 \sim \chi^2(2)$, and we can reject the assumption (55) if $|z_n|^2$ is too large.

Finally, we observe that the scheme described above is randomized. The apparent downside is that, even when the assumption (55) is correct for a particular value of i, there is a finite probability that $x_i \neq x_{\text{best}}$ (see (24)). However, the obvious advantage of randomization is that the scheme can be iterated (and the probability of failure will decrease with the number of iterations; see, however, Remark 4 in Section 5.4).

4.6.3 Basic RaDe1: detailed description

This section contains a detailed description of the decoding scheme described in Section 4.6.2 above.

Precomputation. Suppose that H is the n by n channel matrix.

Step 1. Evaluate the SVD (e.g. the matrices U, Σ, V) of H (see Theorem 3).
Step 2. For every $k = 1, \ldots, n$, evaluate $s_n(k)$ and $S_{n-1}(k)$ via (49), (50), respectively.
Step 3. Find $1 \leq j \leq n$ such that $S_{n-1}(j) \leq S_{n-1}(k)$ for every $k = 1, \ldots, n$.

Decoding. Suppose that x_{true} in X is the (unknown) transmitted message, that $\sigma > 0$ is the component-wise variation of the noise in (23), and that y_{obs} is the observed value of

$y(x_{\text{true}})$ (see (23)).

For every $i = 1, \ldots, m$ proceed as follows:

Step 1. Sample the complex number $\hat{z} \sim N\mathbb{C}(0, 1)$.
Step 2. Evaluate $\tilde{x}_{\text{obs}}$ in $\mathbb{C}^n$ via (41).
Step 3. Evaluate $\hat{z}_n$ via the formula

$$\hat{z}_n = \frac{\tilde{x}_{\text{obs}}(j) - c_i - \hat{z} \cdot \sigma \cdot S_{n-1}(j)}{v_n(j)}, \tag{56}$$

where v_n in $\mathbb{C}^n$ is the nth column of V.
Step 4. Evaluate $\tilde{x}_i$ in $\mathbb{C}^n$ via the formula

$$\tilde{x}_i = \tilde{x}_{\text{obs}} - \hat{z}_n \cdot v_n. \tag{57}$$

Step 5. Find the nearest neighbor x_i of $\tilde{x}_i$ in X.
Step 6. Evaluate w_i in $\mathbb{C}^n$ via the formula

$$w_i = y_{\text{obs}} - H \cdot x_i. \tag{58}$$

Step 7. Evaluate the real numbers r_i and χ_i via the formulae

$$r_i = \frac{|w_i(1)|^2 + \cdots + |w_i(n)|^2}{\sigma^2} \tag{59}$$

and

$$\chi_i = 1 - F_{\chi^2(2n)}(r_i), \tag{60}$$

respectively, where $F_{\chi^2(2n)}$ is the cdf of the χ^2_{2n}-distribution (see (11)).

Thus, for each $i = 1, \ldots, m$, the scheme produces $x_i \in X$ and the real numbers r_i and χ_i. Suppose that i_0 is the index of the smallest r_i (and thus largest χ_i) among $r_1, \ldots, r_n$. The algorithm returns the vector x_{RaDe1} in X defined via the formula

$$x_{\text{RaDe1}} = x_{i_0}. \tag{61}$$

Observation. Under the assumption that $x_{\text{true}} = x_i$, the vector w_i defined via (58) is an observed value of the random vector having the distribution $\sigma \cdot N\mathbb{C}(0_d, I_d)$ (see (23)). Consequently, r_i is an observed value from the χ^2_{2n} distribution, and χ_i is the probability that a χ^2_{2n} random variable attains values larger than r_i (we refer to χ_i as the "confidence").

As a conclusion, we summarize the principal input and output parameters of the decoding scheme described in this section.

Calling sequence:

$$\text{RaDe1\_search}\,(y_{\text{obs}}; x_{\text{RaDe1}}, r, \chi)\,. \tag{62}$$

Input parameters:

– $y_{\rm obs}$ in $\mathbb{C}^n$:
the received message (a noisy observation of $y(x_{\rm true})$ defined via (23)).

Output parameters:
– $x_{\rm RaDe1}$ in X:
the candidate for the transmitted message (see (61)).
– r (a positive real number):
the squared norm of the noise (provided that $x_{\rm true} = x_{\rm RaDe1}$), see (59).
– χ (a real number between 0 and 1):
the confidence in $x_{\rm RaDe1}$ (see (60)).

4.6.4 RaDe1: full algorithm

In this section, we describe a decoding algorithm whose basic step is the scheme described above (see Sections 4.6.2, 4.6.3).

Additional parameters.
– min_RaDe1: a positive integer (minimal number of iterations)
– max_RaDe1 $\geq$ min_RaDe1: a positive integer (maximal number of iterations)
– $\chi_{\rm thresh}$: a real number between 0 and 1 (a confidence threshold)

Description.
1. For each $1 \leq j <$ min_RaDe1, call RaDe1_search $(y_{\rm obs}; x_j, r_j, \chi_j)$.
2. Among all (x_j, r_j, χ_j), select the triplet

$$(x_{\rm RaDe1}, r_{\rm RaDe1}, \chi_{\rm RaDe1}) = (x_i, r_i, \chi_i) \tag{63}$$

that corresponds to the smallest r_i.
3. For each min_RaDe1 $\leq j \leq$ max_RaDe1:
3a. call RaDe1_search $(y_{\rm obs}; x_j, r_j, \chi_j)$.
3b. if $r_j < r_{\rm RaDe1}$, set

$$(x_{\rm RaDe1}, r_{\rm RaDe1}, \chi_{\rm RaDe1}) = (x_j, r_j, \chi_j). \tag{64}$$

3c. if $\chi_{\rm RaDe1} > \chi_{\rm thresh}$, stop.

Comment. The algorithm conducts at least min_RaDe1 iterations of the basic scheme. Then, if the confidence in the best guess is high enough (compared to $\chi_{\rm thresh}$), the algorithm stops. Otherwise, the basic scheme is called iteratively until the confidence is high enough *or* the total number of iterations reaches max_RaDe1.

To conclude, we summarize the principal input and output parameters of the algorithm described in this section.

Calling sequence:

$$\text{RaDe1\_all}\,(y_{\rm obs}; \text{min\_RaDe1}, \text{max\_RaDe1}, \chi_{\rm thresh}; x_{\rm RaDe1}, r_{\rm RaDe1}, \chi_{\rm RaDe1})\,. \tag{65}$$

Input parameters:
– $y_{\rm obs}$ in $\mathbb{C}^n$ (see Section 4.6.3).
– min_RaDe1: see above.

– max_RaDe1: see above.
– $0 < \chi_{\text{thresh}} < 1$: see above.

Output parameters:
– x_{RaDe1} in X (see (63)).
– r_{RaDe1} (a positive real number): see (63).
– χ_{RaDe1} (a real number between 0 and 1): see (63).

4.6.5 RaDe1: cost and memory requirements

In this section, we describe the cost and memory requirements of the basic decoding scheme described in Sections 4.6.2, 4.6.3 above.

Memory requirements. The memory requirements of the scheme are or the order

$$M_{\text{RaDe1}} = O\left(m \cdot n + n^2\right) \tag{66}$$

memory words - in other words, absolutely minimal (compare for example to (35); see also Section 4.2 for typical values of m and n).

Cost. The precomputation step (see Section 4.6.3) requires $O(n^3)$ operations, and should not be re-done until H changes.

On the other hand, each decoding step requires

$$C_{\text{RaDe1}} = O\left(m \cdot n^2\right) \tag{67}$$

operations. Therefore, the full algorithm described in Section 4.6.4 requires between

$$C_{\text{RaDe1,best}} = O\left(\text{min\_RaDe1} \cdot m \cdot n^2\right) \tag{68}$$

and

$$C_{\text{RaDe1,worst}} = O\left(\text{max\_RaDe1} \cdot m \cdot n^2\right) \tag{69}$$

operations.

Success rate. Obviously, the success rate of RaDe1 depends on various parameters of the problems (e.g. H, σ, X) as well as on the parameters of the algorithm (see Section 4.6.4). In addition, since the scheme is randomized, for each set of conditions, there is a certain probability of failure (that decreases with the number of iterations of the basic step). Obviously, there is a trade-off between the probability of success and number of iterations.

The performance of the scheme is demonstrated via several experiments in Section 5.

4.7 Randomized Decoder 2 (RaDe2)

In this section, we describe yet another decoding scheme. This scheme is closely related to RaDe1 from Section 4.6, and can be viewed as a generalization of the latter. This scheme will be referred to as "Randomized Decoder 2", or "RaDe2".

4.7.1 Preliminary discussion

The decoding scheme RaDe1 from Section 4.6 is based on the equation (51). Loosely speaking, one fixes the coordinate j, "guesses" the value of $x_{\text{true}}(j)$, samples z from $N\mathbb{C}(0,1)$, and determines the value of z_n from (51). Then, the noise in $\tilde{x}$ is reduced by subtracting its (estimated) largest component in the direction of v_n (see (48)).

In this section, we describe an algorithm that uses two (rather than one) coordinates. The resulting scheme is more accurate than RaDe1 (see Section 4.6); however, it is also more computationally expensive.

Along the lines of Section 4.6.1, we make the following observation. Suppose that $1 \leq j_1, j_2 \leq n$ correspond to the two smallest values of $S_{n-2}(1), \ldots, S_{n-2}(n)$ defined via (50). Due to the combination of (48), (49), (50),

$$\tilde{x}(j_1) \sim x_{\text{true}}(j_1) + u \cdot \sigma \cdot S_{n-2}(j_1) + z_{n-1} \cdot \sigma \cdot s_{n-1}(j_1) + z_n \cdot \sigma \cdot s_n(j_1), \tag{70}$$

$$\tilde{x}(j_2) \sim x_{\text{true}}(j_2) + w \cdot \sigma \cdot S_{n-2}(j_2) + z_{n-1} \cdot \sigma \cdot s_{n-1}(j_2) + z_n \cdot \sigma \cdot s_n(j_2), \tag{71}$$

where $u, w \sim N\mathbb{C}(0,1)$ (however, u and w are not independent of each other), while $z_{n-1}, z_n \sim N\mathbb{C}(0,1)$ are independent of each other and of u, w.

Obviously, due to (50), the coefficients of u, w in (71) are even smaller than the coefficient of z in (51) (see Section 4.6, in particular Observation 2).

4.7.2 RaDe2: informal description

The discussion in Section 4.7.1 above leads to the following decoding scheme (somewhat similar to that from Section 4.6).

We select $1 \leq j_1, j_2 \leq n$ such that $S_{n-2}(j_1), S_{n-2}(j_2)$, respectively, are the smallest among $S_{n-2}(1), \ldots, S_{n-2}(n)$. Then, for every *pair* of indices $1 \leq i_1, i_2 \leq m$, we assume that

$$x_{\text{true}}(j_1) = c_{i_1}, \tag{72}$$

$$x_{\text{true}}(j_2) = c_{i_2}, \tag{73}$$

and proceed as follows. Under the assumption (73), the difference $\tilde{x}(j) - x_{\text{true}}(j)$ for each $j = j_1, j_2$ is a sum of three independent complex normal variables with different variances (see (71)). We sample $\hat{u}, \hat{w}$ in (71) by drawing i.i.d. $z_1, \ldots, z_{n-2} \sim N\mathbb{C}(0,1)$ and using (48). Then, we determine the values of z_{n-1}, z_n in (71) under the assumption (73) and $u = \hat{u}, w = \hat{w}$ by solving the corresponding two by two linear system. We reduce the noise in $\tilde{x}$ by subtracting $z_{n-1} \cdot \sigma \cdot \sigma_{n-1}^{-1} \cdot v_{n-1}$ and $z_n \cdot \sigma \cdot \sigma_n^{-1} \cdot v_n$ from the observed value of $\tilde{x}$ (see (48)).

Next, we look for the nearest neighbor x_{i_1,i_2} of the "improved" $\tilde{x}$ in X (see Section 4.5). We observe that, provided that the assumption (73) is correct and also the noise in $\tilde{x}$ has indeed been reduced, x_{i_1,i_2} is more likely to be equal to x_{true} than if we were to use the nearest neighbor search in X for the original $\tilde{x}$.

To validate the assumption (55), we make several observations.

1. For each $1 \leq i_1, i_2 \leq m$, the assumption (73) yields x_{i_1,i_2}. We can simply select the best one out of the m^2 possible x_{i_1,i_2} (actually, we do not even need to examine all m^2 possibilities; see Section 4.7.3 below).

2. If $x_{i_1,i_2} = x_{\text{true}}$, then we can determine the noise from (23); in particular, we can compute the square of the Euclidean norm of the noise. However, this quantity must be an observed value of a $\chi^2(2n)$ random variable (multiplied by σ^2), whose cdf is defined via (11); we use it to estimate our "confidence" in the statement that $x_{i_1,i_2} = x_{\text{true}}$.

3. The value of $|z_{n-1}|^2 + |z_n|^2$ must be an observed value of a $\chi^2(4)$ random variable, and thus we can reject the assumption (73) *before* evaluating x_{i_1,i_2} if $|z_n|^2$ is too large.

4.7.3 Basic RaDe2: detailed description

This section contains a detailed description of the decoding scheme described in Section 4.7.2 above.

Compared to the scheme described in Section 4.6.3, the procedure below accepts an additional input parameter: namely, the "early exit confidence" χ_{stop} (a real number between 0 and 1: see (80) below).

Precomputation. Suppose that H is the n by n channel matrix.

Step 1. Evaluate the SVD (e.g. the matrices U, Σ, V) of H (see Theorem 3).

Step 2. For every $k = 1, \ldots, n$, evaluate $s_n(k), s_{n-1}(k)$ and $S_{n-2}(k)$ via (49), (50), respectively.

Step 3. Find $1 \leq j_1, j_2 \leq n$ such that $S_{n-2}(j_1), S_{n-2}(j_2)$ are the two smallest values among $S_{n-2}(1), \ldots, S_{n-2}(n)$.

Decoding. Suppose that x_{true} in X is the (unknown) transmitted message, that $\sigma > 0$ is the component-wise variation of the noise in (23), and that y_{obs} is the observed value of $y(x_{\text{true}})$ (see (23)).

For every $i_1 = 1, \ldots, m$ and $i_2 = 1, \ldots, m$, proceed as follows:

Step 1. Sample the $n-2$ complex numbers $\hat{z_1}, \ldots, \hat{z_{n-2}}$ independently from $N\mathbb{C}(0,1)$.

Step 2. Evaluate $\hat{u}, \hat{w}$ via the formulae

$$\hat{u} = \sum_{k=1}^{n-2} \hat{z}_k \cdot \frac{\sigma}{\sigma_k} \cdot v_k(j_1), \tag{74}$$

$$\hat{w} = \sum_{k=1}^{n-2} \hat{z}_k \cdot \frac{\sigma}{\sigma_k} \cdot v_k(j_2), \tag{75}$$

where v_k in $\mathbb{C}^n$ is the kth column of V for every $k = 1, \ldots, n$.

Step 3. Evaluate $\tilde{x}_{\text{obs}}$ in $\mathbb{C}^n$ via (41).

Step 4. Evaluate $\hat{z}_{n-1}, \hat{z}_n$ via solving the two by two linear system

$$z_{n-1} \cdot \frac{\sigma \cdot v_{n-1}(j_1)}{\sigma_{n-1}} + z_n \cdot \frac{\sigma \cdot v_n(j_1)}{\sigma_n} = \tilde{x}_{\text{obs}}(j_1) - c_{i_1} - \hat{u}, \tag{76}$$

$$z_{n-1} \cdot \frac{\sigma \cdot v_{n-1}(j_2)}{\sigma_{n-1}} + z_n \cdot \frac{\sigma \cdot v_n(j_2)}{\sigma_n} = \tilde{x}_{\text{obs}}(j_2) - c_{i_2} - \hat{w}, \tag{77}$$

in the unknowns z_{n-1}, z_n.

Step 5. Evaluate the real numbers r and χ via the formulae

$$r = |\hat{z}_{n-1}|^2 + |\hat{z}_n|^2, \tag{78}$$

$$\chi = 1 - F_{\chi^2(4)}(r), \tag{79}$$

respectively, where $F_{\chi^2(4)}$ is the cdf of the distribution $\chi^2(4)$ defined via the formula (11).

Step 6. If

$$\chi < \chi_{\text{stop}}, \tag{80}$$

skip the rest of the steps for these values of i_1, i_2.

Comment. Roughly speaking, if χ is too small, it means that $\hat{z}_{n-1}$ and $\hat{z}_n$ are unlikely to be observed values of two i.i.d. $N\mathbb{C}(0,1)$ random variables; in other words, the assumption (73) is likely to be wrong for these values of i_1, i_2.

Step 7. Evaluate $\tilde{x}_{i_1,i_2}$ in $\mathbb{C}^n$ via the formula

$$\tilde{x}_{i_1,i_2} = \tilde{x}_{\text{obs}} - \hat{z}_{n-1} \cdot \frac{\sigma}{\sigma_{n-1}} \cdot v_{n-1} - \hat{z}_n \cdot \frac{\sigma}{\sigma_n} \cdot v_n \tag{81}$$

Step 8. Find the nearest neighbor x_{i_1,i_2} of $\tilde{x}_{i_1,i_2}$ in X.

Step 9. Evaluate w_{i_1,i_2} in $\mathbb{C}^n$ via the formula

$$w_{i_1,i_2} = y_{\text{obs}} - H \cdot x_{i_1,i_2}. \tag{82}$$

Step 10. Evaluate the real numbers r_{i_1,i_2} and χ_{i_1,i_2} via the formulae

$$r_{i_1,i_2} = \frac{|w_{i_1,i_2}(1)|^2 + \cdots + |w_{i_1,i_2}(n)|^2}{\sigma^2} \tag{83}$$

and

$$\chi_{i_1,i_2} = 1 - F_{\chi^2(2n)}(r_{i_1,i_2}), \tag{84}$$

respectively, where $F_{\chi^2(2n)}$ is the cdf of the χ^2_{2n}-distribution (see (11)).

Thus, for every pair (i_1, i_2) for which Steps 7-10 were performed, the scheme produces $x_{i_1,i_2} \in X$ and the real numbers r_{i_1,i_2} and χ_{i_1,i_2}. Suppose that, among these, the triplet

$$(x_{\text{RaDe2}}, r_{\text{RaDe2}}, \chi_{\text{RaDe2}}) = (x_{i_1,i_2}, r_{i_1,i_2}, \chi_{i_1,i_2}) \tag{85}$$

corresponds to the smallest r_{i_1,i_2} (equivalently, to the largest χ_{i_1,i_2}).

As a conclusion, we summarize the principal input and output parameters of the decoding scheme described in this section.

Calling sequence:

$$\text{RaDe2\_search}\,(y_{\text{obs}}; \chi_{\text{stop}}; x_{\text{RaDe2}}, r, \chi)\,. \tag{86}$$

Input parameters:

– y_{obs} in $\mathbb{C}^n$:

the received message (a noisy observation of $y(x_{\text{true}})$ defined via (23)).
– χ_{stop} (a real number between 0 and 1):
the confidence threshold used in Step 6 (see (80)).

Output parameters:
– x_{RaDe2} in X:
the candidate for the transmitted message (see (85)).
– r (a positive real number):
the squared norm of the noise (provided that $x_{\text{true}} = x_{\text{RaDe2}}$), see (83).
– χ (a real number between 0 and 1):
the confidence in x_{RaDe2} (see (84)).

4.7.4 RaDe2: full algorithm

In this section, we describe a decoding algorithm whose basic step is the scheme described above (see Sections 4.7.2, 4.7.3). This algorithm is closely related to RaDe1 (see Section 4.6.4), except that the basic step of the latter (see Section 4.6.3) is replaced with the one from Section 4.7.3.

Additional parameters.
– min_RaDe2: a positive integer (minimal number of iterations)
– max_RaDe2 $\geq$ min_RaDe2: a positive integer (maximal number of iteration)
– χ_{thresh}: a real number between 0 and 1 (a confidence threshold)
– χ_{stop}: a real number between 0 and 1 (the early exit confidence; see (80)).

Description.
1. For each $1 \leq j <$ min_RaDe2, call RaDe2_search $(y_{\text{obs}}; \chi_{\text{stop}}; x_j, r_j, \chi_j)$.
2. Among all (x_j, r_j, χ_j), select the triplet

$$(x_{\text{RaDe2}}, r_{\text{RaDe2}}, \chi_{\text{RaDe2}}) = (x_i, r_i, \chi_i) \tag{87}$$

that corresponds to the smallest r_i.
3. For each min_RaDe2 $\leq j \leq$ max_RaDe2:
3a. call RaDe2_search $(y_{\text{obs}}; \chi_{\text{stop}}; x_j, r_j, \chi_j)$.
3b. if $r_j < r_{\text{RaDe2}}$, set

$$(x_{\text{RaDe2}}, r_{\text{RaDe2}}, \chi_{\text{RaDe2}}) = (x_j, r_j, \chi_j). \tag{88}$$

3c. if $\chi_{\text{RaDe2}} > \chi_{\text{thresh}}$, stop.

Comment. The algorithm conducts at least min_RaDe2 iterations of the basic scheme. Then, if the confidence in the best guess is high enough (compared to χ_{thresh}), the algorithm stops. Otherwise, the basic scheme is called iteratively until the confidence is high enough *or* the total number of iterations reaches max_RaDe2.

To conclude, we summarize the principal input and output parameters of the algorithm described in this section.

Calling sequence:

$$\text{RaDe2\_all}\,(y_{\text{obs}}; \text{min\_RaDe2}, \text{max\_RaDe2}, \chi_{\text{thresh}}, \chi_{\text{stop}}; x_{\text{RaDe2}}, r_{\text{RaDe2}}, \chi_{\text{RaDe2}})\,. \tag{89}$$

Input parameters:
– y_{obs} in $\mathbb{C}^n$ (see Section 4.7.3).
– min_RaDe2: see above.
– max_RaDe2: see above.
– $0 < \chi_{\text{thresh}} < 1$: see above.
– $0 < \chi_{\text{stop}} < 1$: the confidence threshold used in (80).

Output parameters:
– x_{RaDe2} in X (see (87)).
– r_{RaDe2} (a positive real number): see (87).
– χ_{RaDe2} (a real number between 0 and 1): see (87).

4.7.5 RaDe2: cost and memory requirements

In this section, we describe the cost and memory requirements of the basic scheme described in Sections 4.7.2, 4.7.3 above.

Memory requirements. The memory requirements of the scheme are or the order

$$M_{\text{RaDe2}} = O\left(m^2 \cdot n + n^2\right) \tag{90}$$

memory words - in other words, absolutely minimal (compare for example to (35); see also Section 4.2 for typical values of m and n).

Cost. The precomputation step (see Section 4.6.3) requires $O(n^3)$ operations, and should not be re-done until H changes.

On the other hand, each decoding step requires

$$C_{\text{RaDe2}} = O\left(\alpha \cdot m^2 \cdot n^2\right) \tag{91}$$

operations, where $0 < \alpha < 1$ is the proportion of pairs (i_1, i_2) for which Steps 7-10 in Section 4.7.3 are performed (e.g. for which χ defined via (79) is greater than χ_{stop}). Therefore, the full algorithm described in Section 4.7.4 requires between

$$C_{\text{RaDe2,best}} = O\left(\text{min\_RaDe2} \cdot m^2 \cdot n^2\right) \tag{92}$$

and

$$C_{\text{RaDe2,worst}} = O\left(\text{max\_RaDe2} \cdot m^2 \cdot n^2\right) \tag{93}$$

operations.

Observation. By comparing (91) to (67) in Section 4.6.5, we observe that the basic step of RaDe2 (see Section 4.7.3) is typically somewhat slower than the basic step of RaDe1 (see Section 4.6.3) roughly by a factor of $\alpha \cdot m$, where $0 < \alpha < 1$ is a real number (see (91) above). Obviously, α depends, among other things, on the parameter χ_{stop} (see Section 4.7.3).

Success rate. Obviously, the success rate of RaDe2 depends on various parameters of the problems (e.g. H, σ, X) as well as on the parameters of the algorithm (see Section 4.7.4). In addition, since the scheme is randomized, for each set of conditions, there is a certain probability of failure (that decreases with the number of iterations of the basic step; see also

Remark 5 in Section 5.5). Obviously, there is a trade-off between the probability of success and number of iterations. In addition, compared to RaDe1 from Section 4.6.4, RaDe2 is, in general, slower, but has a higher success rate (see Section 5 below).

Also, in Section 5 we demonstrate the performance of the scheme via several experiments.

4.8 Supercharging

In this section, we describe a procedure that should not be used on its own, but rather as an additional step after the algorithms from previous sections, with the goal to improve their output at a moderate computational cost.

4.8.1 Supercharging: informal description

Either of the algorithms RaDe1, RaDe2 (see Sections 4.6, 4.7) computes a "candidate" x for x_{best} defined via (24); in addition, it evaluates the confidence χ in this candidate (see (62), (86)). This candidate can be improved (e.g. replaced by a better candidate) by a procedure that we call "supercharging".

The idea behind supercharging is based on the observation that even if x is not equal to x_{best}, the latter can still be among several nearest neighbors of x in X. In other words, we fix the integer k_1 of nearest neighbors that we wish to inspect and find k_1 nearest neighbors $x_1, \dots, x_{k_1}$ of x in X (see Section 4.5). Among these, we find the one that minimizes the distance between $H \cdot x_i$ and y_{obs}.

4.8.2 Supercharging: detailed description

Suppose that y_{obs} in $\mathbb{C}^n$ is the observed value of x_{true} (see Section 4.1), that x in X is a candidate for x_{best} evaluated by either RaDe1 or RaDe2 (see Sections 4.6, 4.7), that r is the (normalized) squared norm of the noise, and that χ is the confidence in x (see (63), (87)). Suppose also that $k_1 > 0$ is the number of nearest neighbors of x that we want to inspect. Supercharging consists of the following steps.

Step 1. Evaluate the k_1 nearest neighbors $x_1, \dots, x_{k_1}$ of x in X (see Section 4.5).

Step 2. For every $i = 1, \dots, k_1$, evaluate the real number r_i via the formula

$$r_i = \frac{\|H \cdot x_i - y_{\text{obs}}\|^2}{\sigma^2}. \tag{94}$$

Step 3. Find the minimum r_i among $r_1, \dots, r_{k_1}$.

Step 4. If $r_i < r$, evaluate x_{super} in X and the real numbers $r_{\text{super}}, \chi_{\text{super}}$ via the formulae

$$x_{\text{super}} = x_i, \tag{95}$$

$$r_{\text{super}} = r_i, \tag{96}$$

$$\chi_{\text{super}} = 1 - F_{\chi^2(2n)}(r_i), \tag{97}$$

where $F_{\chi^2(2n)}$ is the cdf of the $\chi^2(2n)$ distribution defined via (11).

As a conclusion, we summarize the principal input and output parameters of supercharging.

Calling sequence:

$$\text{super}\,(y_{\text{obs}}; k_1, x, r, \chi; x_{\text{super}}, r_{\text{super}}, \chi_{\text{super}})\,. \tag{98}$$

Input parameters:

– y_{obs} in $\mathbb{C}^n$:
the received message (a noisy observation of $y(x_{\text{true}})$ defined via (23)).
– k_1 (a positive integer):
the number of nearest neighbors to use.
– x in X:
the candidate for x_{best} (see (62), (86)).
– r (a positive real number):
the normalized squared norm of the noise (see (62), (86)).
– χ (a real number between 0 and 1):
the confidence in x (see (62), (86)).

Output parameters:

– x_{super} in X:
the improved candidate (see (95)).
– r_{super} (a positive real number):
the corresponding normalized squared norm of the noise (see (96)).
– χ_{super} (a real number between 0 and 1):
the confidence in x_{super} (see (97)).

4.8.3 Supercharging: cost and memory requirements

The memory requirements and number of operations of supercharging are given, respectively, by the formulae (45), (46) in Section 4.5 (obviously, with k being replaced by k_1).

However, often this cost can be reduced roughly by the factor of n as follows. Suppose, for the sake of concreteness, that the constellation C is given via (27) in Section 4.2. Then, every x in X has $2 \cdot n$ nearest neighbors $x_1, \ldots, x_{2n}$ in X all of which are at exactly the same distance from x. Each of $x_1, \ldots, x_{2n}$ differs from x at precisely one coordinate; in other words, for every $i = 1, \ldots, 2n$, the difference $x - x_i$ has only one non-zero coordinate. In particular, each of r_i defined via (94) can be evaluated in $O(n)$ rather than $O(n^2)$ operations. In other words, if, for example,

$$k_1 = 2 \cdot n, \tag{99}$$

then the total cost of supercharging is

$$C_{\text{super}}(2 \cdot n) = O(n^2) \tag{100}$$

(rather than $O(n^3)$) operations.

5 Numerical Results

In this section, we illustrate the performance of the schemes from Section 4 via several numerical experiments. All the calculations were implemented in FORTRAN (the Lahey 95 LINUX version), and were carried out in double precision, on a standard laptop computer with DualCore CPU 2.53 GHz and 2.9GB RAM.

5.1 Experiments: basic structure

All the experiments are build around the model described in Section 4.1. In addition, we choose our parameters based on the running example from Section 4.2.

Each experiment consists of five stages described below.

5.1.1 Stage 1: preparation

On this stage, we perform operations that do not depend on the channel matrix H, let alone the transmitted (or received) message.

We start by selecting the number of transmitters and receivers n (see (25)), the constellation size m (see (26)) and the constellation C (see (27)). In particular, the constellation C is the same in all experiments, as is the collection X of all possible messages (see (22)).

Next, we select the integer $k > 0$ and find the list of k nearest neighbors of x_0 (see (39)) in X by brute force. For any x in X, this list allows us to compute its k nearest neighbors in $O(k \cdot n)$ operations (see Section 4.5, in particular (40)).

5.1.2 Stage 2: H and related quantities

On this stage, we form the n by n complex matrix H and carry out some computations that depend only on H.

First, we generate H according to (28) (i.e. by drawing n^2 independent samples from $N\mathbb{C}(0,1)$ and using them as entries of H).

Then, we compute the singular value decomposition of H (e.g. the matrices U, Σ, V in the notation of Theorem 3). We use Σ and V to evaluate the real numbers $S_{n-2}(1), \ldots, S_{n-2}(n)$, $S_{n-1}(1), \ldots, S_{n-1}(n)$ (see (50)) and $s_{n-1}(1), \ldots, s_{n-1}(n)$, $s_n(1), \ldots, s_n(n)$ (see (49)). Also, we find the integer $1 \leq j \leq n$ that corresponds to the minimal $S_{n-1}(j)$ among $S_{n-1}(1), \ldots, S_{n-1}(n)$ (see the precomputation step in Section 4.6.3). In addition, we find the integers $1 \leq j_1, j_2 \leq n$ such that $S_{n-2}(j_1)$ and $S_{n-2}(j_2)$ are the two smallest values among $S_{n-2}(1), \ldots, S_{n-2}(n)$ (see the precomputation step in Section 4.7.3).

Finally, we select the real number $\sigma > 0$ (roughly of the same order of magnitude that the smallest singular value σ_n of H: see Theorem 3 and Remark 1).

In Table 1 we summarize the input and output parameters of all computations described in Sections 5.1.1, 5.1.2.

5.1.3 Stage 3: generation of messages

During this stage, we choose, more or less arbitrarily, the integer $L > 0$, and generate L points $x_{\text{true},1}, \ldots, x_{\text{true},L}$ in X by taking L independent samples from the uniform distribu-

Variable	Details
n	dimensionality of received/transmitted message (25)
m	size of constellation (26)
C	the complex constellation (27)
X	the collection of all possible messages (22)
k	the number of nearest neighbors
$\hat{x}_1, \ldots, \hat{x}_k$	the nearest neighbors of x_0 (39) in X
H	the n by n complex channel matrix (28)
U, Σ, V	the SVD of H (see Theorem 3)
$v_1, \ldots, v_n$	the columns of V (vectors in $\mathbb{C}^n$)
$\sigma_1, \ldots, \sigma_n$	the singular values of H (positive real numbers)
$s_{n-1}(1), \ldots, s_{n-1}(n)$	noise deviations, see (49)
$s_n(1), \ldots, s_n(n)$	noise deviations, see (49)
$S_{n-2}(1), \ldots, S_{n-2}(n)$	noise deviations, see (50)
$S_{n-1}(1), \ldots, S_{n-1}(n)$	noise deviations, see (50)
j	the index of the minimal $S_{n-1}(j)$
j_1, j_2	the indices of the two smallest $S_{n-2}(j_1), S_{n-2}(j_2)$
σ	the deviation of noise in (23)

Table 1: *Basic experiment: list of input variables*

tion on X. (In the language of the model from Section 4.1, each $x_{\text{true},i}$ represents a message to be transmitted).

For each $i = 1, \ldots, L$, we generate $y_{\text{obs},i}$ from $x_{\text{true},i}$ according to (23) as follows: we sample a complex normal standard random vector $\hat{z}_i$ in $\mathbb{C}^n$ (i.e. from the distribution $N\mathbb{C}(0_n, I_n)$) and evaluate $y_{\text{obs},i}$ via the formula

$$y_{\text{obs},i} = H \cdot x_{\text{true},i} + \sigma \cdot \hat{z}_i, \tag{101}$$

where σ is that from Table 1. Needless to say, all $\hat{z}_i$'s are sampled independently of each other. In the language of the model from Section 4.1, each $y_{\text{obs},i}$ represents a received message corrupted by Gaussian noise of coordinate-wise standard deviation σ.

Our ultimate goal is to decode each received message $y_{\text{true},i}$ by computing the maximal likelihood estimate $x_{\text{best},i}$ to $x_{\text{true},i}$, in the sense of (24). We do so by means of several numerical schemes from Section 4. Some of these schemes have a finite probability of failure (e.g. they compute a candidate for $x_{\text{best},i}$ which is equal to the latter with probability less than one). Obviously, none of the schemes cannot use either $x_{\text{true},i}$ or $\hat{z}_i$ (e.g. both the original message and the noise are assumed to be unknown). In other words, each scheme is allowed to access only $y_{\text{obs},i}$ as well as the variables from Table 1.

5.1.4 Stage 4: decoding

On this stage, we decode each of the messages $y_{\text{obs},1}, \ldots, y_{\text{obs},L}$ by means of several schemes from Section 4. More specifically, we use the following schemes:

1. Brute force (see Section 4.3).

2. Nearest neighbors search in X (see Section 4.5).

3. Randomized Decoder 1 (see Section 4.6).

4. Randomized Decoder 2 (see Section 4.7).

Optionally, we improve the results RaDe1 (or RaDe2) by supercharging (see Section 4.8).

In addition to y_{obs_i} and the variables from Table 1, RaDe1 and RaDe2 also receive several "tuning" parameters listed in Table 2 below (see Sections 4.6.4, 4.7.4 for details).

Parameter	Details
χ_{thresh}	the threshold confidence (see Section 4.6.3, 4.7.3)
min_RaDe1	minimal number of iterations of RaDe1 (see Section 4.6.4)
max_RaDe1	maximal number of iterations of RaDe1 (see Section 4.6.4)
χ_{stop}	the early exit confidence (see (80) in Section 4.7.3)
min_RaDe2	minimal number of iterations of RaDe2 (see Section 4.7.4)
max_RaDe2	maximal number of iterations of RaDe2 (see Section 4.7.4)
isuper	whether to perform supercharging (1) or not (0)
k_1	the number of nearest neighbors in supercharging (see (99)).

Table 2: *Basic experiment: list of "tuning" parameters*

For every $i = 1, \ldots, L$, we proceed as follows:

1. Evaluate $x_{\mathrm{best},i}$ from $y_{\mathrm{obs},i}$ by brute force (see Section 4.3).

Comment. This is the "correct answer" (see, however, Remark 1 in Section 4.3).

2. Evaluate $\tilde{x}_{\mathrm{obs},i}$ from $y_{\mathrm{obs},i}$ via (41).

3. Evaluate $x_{\mathrm{nn}(k),i}$ from $y_{\mathrm{obs},i}$ via (43) in Section 4.5 by using k nearest neighbors.

4. Evaluate $x_{\mathrm{RaDe1},i}$ from $y_{\mathrm{obs},i}$ via RaDe1 search (see Section 4.6.4).

5. Evaluate $x_{\mathrm{RaDe2},i}$ from $y_{\mathrm{obs},i}$ via RaDe2 search (see Section 4.7.4).

6. If isuper $= 1$, evaluate $x_{\mathrm{super},i}$ from the best of $x_{\mathrm{RaDe1},i}, x_{\mathrm{RaDe2},i}$ by supercharging (see (103), (104) below, and also Section 4.8).

In other words, for every $i = 1, \ldots, L$, we execute

$$\begin{aligned}
&\mathrm{brute}(y_{\mathrm{obs},i}; x_{\mathrm{best},i})\\
&\mathrm{nnx}(y_{\mathrm{obs},i}; k; \tilde{x}_{\mathrm{obs},i}, x_{\mathrm{nn}(k),i})\\
&\mathrm{RaDe1_all}\,(y_{\mathrm{obs},i}; \mathrm{min_RaDe1}, \mathrm{max_RaDe1}, \chi_{\mathrm{thresh}}; x_{\mathrm{RaDe1},i}, r_{\mathrm{RaDe1},i}, \chi_{\mathrm{RaDe1},i})\\
&\mathrm{RaDe2_all}\,(y_{\mathrm{obs},i}; \mathrm{min_RaDe2}, \mathrm{max_RaDe2}, \chi_{\mathrm{thresh}}, \chi_{\mathrm{stop}}; x_{\mathrm{RaDe2},i}, r_{\mathrm{RaDe2},i}, \chi_{\mathrm{RaDe2},i})
\end{aligned} \tag{102}$$

(see (33), (44), (65), (89)). In addition, if isuper $= 1$, we define x_i in X and the real numbers r_i, χ_i via the formula

$$(x_i, r_i, \chi_i) = \begin{cases} (x_{\mathrm{RaDe1},i}, r_{\mathrm{RaDe1},i}, \chi_{\mathrm{RaDe1},i}) & \text{if } r_{\mathrm{RaDe1},i} < r_{\mathrm{RaDe2},i},\\ (x_{\mathrm{RaDe2},i}, r_{\mathrm{RaDe2},i}, \chi_{\mathrm{RaDe2},i}) & \text{otherwise} \end{cases} \tag{103}$$

and execute

$$\mathrm{super}\,(y_{\mathrm{obs},i}; k_1, x_i, r_i, \chi_i; x_{\mathrm{super},i}, r_{\mathrm{super},i}, \chi_{\mathrm{super},i}) \tag{104}$$

(see (98)).

5.1.5 Stage 5: evaluation of statistics

In Section 5.1.4 above, we describe the decoding of a single received message by means of several schemes. In this section, we describe a way to compare the performance of these schemes via the evaluation of various statistics (essentially, via Monte Carlo simulations). Briefly speaking, we evaluate how much time, on average, it took for each algorithm to compute its output, and in what proportion of cases the output was correct.

Each statistic is evaluated for every $i = 1, \dots, L$. In addition, we also average over several matrices H (corresponding to the same set of parameters).

5.2 Experiment 1

Description. In this experiment, we proceed as follows. For each $n = 6, 7, 8$, we generate five n by n complex matrices H (see Section 5.1.2 above). For each such matrix and each $\sigma = 0.25, 0.5, 0.75, 1, 1.25$, we generate $L = 1000$ messages x_{true} in X, and for each such x_{true} we generate y_{obs} as described in Section 5.1.3. For each such y_{obs}, we evaluate the maximal likelihood estimate x_{best} of x_{true} via the brute force algorithm from Section 4.3 (see also Section 5.1.4).

Thus, for each $n = 6, 7, 8$ and each $\sigma = 0.25, 0.5, 0.75, 1, 1.25$, we obtain 5,000 pairs $(x_{\text{true}}, v_{\text{best}})$. We evaluate the proportion of cases when x_{true} is equal to x_{best}. The results of this experiment are displayed in Table 3.

	σ				
n	0.25	0.5	0.75	1	1.25
6	0.99840E+00	0.83740E+00	0.44200E+00	0.17900E+00	0.74800E-01
7	0.99940E+00	0.87960E+00	0.49780E+00	0.20440E+00	0.77200E-01
8	0.10000E+01	0.90900E+00	0.45600E+00	0.15220E+00	0.49000E-01

Table 3: *Average proportion of $x_{true} = x_{best}$, five channel matrices H per each n. The number of messages is $L = 1,000$ (per matrix). See Section 5.2.*

Observations. Several observations can be made from Table 3.

1. For each n, the average proportion of $x_{\text{best}} = x_{\text{in}}$ decreases monotonically with σ, as expected (the larger σ is the more likely x_{best} is to be different from x_{true}). For example, for $\sigma = 0.25$ this proportion is above 0.99 for all $n = 6, 7, 8$, while for $\sigma = 1.25$ it might be as low as 0.05 (for $n = 8$). See also Remark 1 in Section 4.1.

2. In the view of the previous observation, for each n the values of σ vary from that corresponding to a relatively "easy" decoding task ($\sigma = 0.25$) to that corresponding to a "difficult but possible" decoding task ($\sigma = 1.25$), in the sense of Section 4.1.

Motivated by Observation 2, we will select the same values of σ in all numerical experiments below, to investigate how the performance of various numerical schemes for the solution of the decoding task depends on the standard deviation of noise and dimensionality of the matrix.

5.3 Experiment 2

In this experiment, we investigate the performance of the nearest neighbors search in X (see Section 4.5).

Description. For each $n = 6, 7, 8$ and each $\sigma = 0.25, 0.5, 0.75, 1, 1.25$, we proceed as in Experiment 1 from Section 5.2. Then, for each received message y_{obs} and each $k = 1, 2n+1, 2n^2+1, n^3, n^4, n^5+1$, we evaluate the estimate $x_{\text{nn}(k)}$ of x_{true} via the algorithm from Section 4.5 (see (44)). Then, among all cases when $x_{\text{true}} = x_{\text{best}}$, we evaluate the proportion of cases when the estimate $x_{\text{nn}(k)}$ is correct; in other words, we compute the number $\text{prop}_{\text{nn}}(n, \sigma, k)$ defined via the formula

$$\text{prop}_{\text{nn}}(n, \sigma, k) = \frac{\#\left\{x_{\text{best}} = x_{\text{true}} = x_{\text{nn}(k)}\right\}}{\#\left\{x_{\text{best}} = x_{\text{true}}\right\}}. \tag{105}$$

As in Experiment 1 from Section 5.2, to evaluate each such proportion we use 5,000 messages (five matrices per n, $L = 1,000$ messages per matrix).

Remark 3. *According to* (105)*, in this experiment we evaluate the performance of the scheme only on those messages x_{true} for which the maximal likelihood estimate x_{best} coincides with x_{true}.*

Tables. The results of Experiment 2 are displayed in Tables 4–6. These tables correspond, respectively, to $n = 6, 7, 8$, and contain $\text{prop}_{\text{nn}}(n, \sigma, k)$ defined via (105) above (in each table, rows correspond to the values of σ and columns correspond to the values of k).

In addition, for each n and k, we measure the CPU time required by this scheme to evaluate $x_{\text{nn}(k)}$ for $L = 1,000$ messages (obviously, this CPU time essentially does not depend on σ: see Section 4.5). These CPU times are listed in the last five columns of Table 7. Each row of this table corresponds to $n = 6, 7, 8$, while the last five columns of this table correspond to $k = 2n, 2n^2, n^3, n^4, n^5$, respectively.

	k					
σ	1	$2n+1$	$2n^2+1$	n^3	n^4	n^5+1
0.25	0.659E+00	0.825E+00	0.906E+00	0.939E+00	0.973E+00	0.991E+00
0.5	0.242E+00	0.458E+00	0.645E+00	0.741E+00	0.852E+00	0.922E+00
0.75	0.108E+00	0.281E+00	0.447E+00	0.559E+00	0.717E+00	0.862E+00
1	0.693E-01	0.194E+00	0.362E+00	0.468E+00	0.660E+00	0.812E+00
1.25	0.374E-01	0.144E+00	0.286E+00	0.385E+00	0.596E+00	0.805E+00

Table 4: *Success rate of nearest neighbors search in X (see* (105) *in Section 5.3). Corresponds to Experiment 2 with $n = 6$.*

Observations. We make the following observations from Tables 4–6.

1. As expected, for each n and σ the performance of the scheme improves as k increases (obviously, this improvement comes at the cost of additional CPU time). For example, for $n = 8$ and $\sigma = 0.75$ the proportion of correct "guesses" grows from about 2% to about 58% as k increases from 1 to n^5.

	k					
σ	1	$2n+1$	$2n^2+1$	n^3	n^4	n^5+1
0.25	0.603E+00	0.794E+00	0.897E+00	0.926E+00	0.971E+00	0.993E+00
0.5	0.226E+00	0.418E+00	0.568E+00	0.658E+00	0.808E+00	0.917E+00
0.75	0.996E-01	0.236E+00	0.374E+00	0.483E+00	0.648E+00	0.796E+00
1	0.489E-01	0.139E+00	0.262E+00	0.359E+00	0.524E+00	0.679E+00
1.25	0.311E-01	0.907E-01	0.184E+00	0.251E+00	0.425E+00	0.598E+00

Table 5: *Success rate of nearest neighbors search in X (see* (105) *in Section 5.3). Corresponds to Experiment 2 with $n = 7$.*

	k					
σ	1	$2n+1$	$2n^2+1$	n^3	n^4	n^5+1
0.25	0.430E+00	0.569E+00	0.655E+00	0.699E+00	0.765E+00	0.821E+00
0.5	0.112E+00	0.244E+00	0.369E+00	0.450E+00	0.566E+00	0.666E+00
0.75	0.298E-01	0.101E+00	0.197E+00	0.287E+00	0.442E+00	0.575E+00
1	0.920E-02	0.420E-01	0.125E+00	0.192E+00	0.311E+00	0.466E+00
1.25	0.816E-02	0.245E-01	0.571E-01	0.106E+00	0.196E+00	0.363E+00

Table 6: *Success rate of nearest neighbors search in X (see* (105) *in Section 5.3). Corresponds to Experiment 2 with $n = 8$.*

			k				
n	RaDe1	RaDe2	$2n$	$2n^2$	n^3	n^4	n^5
6	0.46E-1	0.14E+0	0.20E-1	0.75E-1	0.22E+0	0.13E+1	0.76E+1
7	0.50E-1	0.12E+0	0.26E-1	0.14E+0	0.40E+0	0.27E+1	0.19E+2
8	0.54E-1	0.15E+0	0.33E-1	0.27E+0	0.80E+0	0.56E+1	0.45E+2

Table 7: *CPU time of various decoding schemes (in seconds). The number of messages is $L = 1000$. For RaDe1 and RaDe2, we used $\sigma = 0.75$.*

2. As expected, for each n and k the performance of the scheme deteriorates as σ increases (noise of a larger standard deviation makes the decoding task more difficult).

3. Typically, for each σ and comparable values of k, the performance of the scheme deteriorates as n increases.

4. When $n = 8$ and $\sigma = 1, 1.25$, the scheme returns the correct answer in less than half of all cases for all values of k in Table 6. In other words, when, for example, $\sigma = 1$, in more than half of all cases x_{true} is not among as many as $32,000$ nearest neighbors of $H^{-1} \cdot y_{\text{obs}}$ in X, even when $x_{\text{true}} = x_{\text{best}}$ (see (41) in Section 4.5).

Some additional observations can be made from Table 7.

5. Naturally, the CPU time of nearest neighbor search in X scales roughly as one would expect from (46) in Section 4.5.

6. Typically, even a noticeable increase in the CPU time (e.g. taking a larger k) results in a fairly modest improvement in performance. For example, when $n = 8$ and $\sigma = 1$, using $k = 8^4$ nearest neighbors to decode the message results in about 31% of correct guesses (and it takes about 6 seconds per 1000 messages). Increasing the number of nearest neighbors by a factor of eight allows one to determine about half as many additional messages correctly (e.g. about 46% overall), while the CPU time goes up to about 45 seconds.

5.4 Experiment 3

In this experiment, we investigate the performance of Randomized Decoder 1 (see Section 4.6).

Description. For each $n = 6, 7, 8$ and each $\sigma = 0.25, 0.5, 0.75, 1, 1.25$, we proceed as in Experiment 1 (see Section 5.2). Then, for each received message y_{obs} and each $T = 1, \ldots, 7$, we evaluate the estimate $x_{\text{RaDe1}}(T)$ of x_{true} from y_{obs} via RaDe1 (see Section 4.6), with the following parameters (see (62) in Section 4.6.4, and also Table 2 in Section 5.1.3):

$$\begin{aligned} &\text{min\_RaDe1} = T, \\ &\text{max\_RaDe1} = T, \\ &\chi_{\text{thresh}} : \text{not applicable} \end{aligned} \tag{106}$$

(the parameter χ_{thresh} is not applicable since min_RaDe1 = max_RaDe1 = T, e.g. the number of basic iterations is always exactly T). Then, we define the integer k_1 via the formula

$$k_1 = 2 \cdot n^2. \tag{107}$$

and evaluate the estimate $x_{\text{super}}(T)$ of x_{true} from y_{obs} via RaDe1 (where the parameters are defined via (106)) followed by supercharging (see Section 4.8, in particular (98)); during the supercharging step, k_1 nearest neighbors are used. In other words, $x_{\text{RaDe1}}(T)$ and $x_{\text{super}}(T)$ are obtained from y_{obs} via calling

$$\begin{aligned} &\text{RaDe1\_search}\,(y_{\text{obs}}; x_{\text{RaDe1}}(T), r, \chi) \\ &\text{RaDe1\_search}\,(y_{\text{obs}}; x, r, \chi) \\ &\text{super}\,(y_{\text{obs}}; k_1, x, r, \chi; x_{\text{super}}(T), r_{\text{super}}, \chi_{\text{super}}) \end{aligned} \tag{108}$$

(see (62), (98)), where the parameters are defined via (106) and (107).

Thus, for each n and σ, we have 5,000 messages y_{obs} (five matrices per n, $L = 1,000$ messages per matrix, as in Experiments 1,2 from Sections 5.2, 5.3); for each such message, we obtain 14 estimates $x_{\text{RaDe1}}(1), \dots, x_{\text{RaDe1}}(7)$ and $x_{\text{super}}(1), \dots, x_{\text{super}}(7)$ of x_{true}. For each such estimate, we evaluate the proportion of the cases in which this estimate is correct (provided that $x_{\text{true}} = x_{\text{best}}$, see Section 5.2). In other words, we compute $\text{prop}_{\text{RaDe1}}(n, \sigma, T)$ and $\text{prop}_{\text{super}}(n, \sigma, T)$ via the formulae

$$\text{prop}_{\text{RaDe1}}(n, \sigma, T) = \frac{\#\left\{x_{\text{best}} = x_{\text{true}} = x_{\text{RaDe1}(T)}\right\}}{\#\left\{x_{\text{best}} = x_{\text{true}}\right\}} \tag{109}$$

and

$$\text{prop}_{\text{super}}(n, \sigma, T) = \frac{\#\left\{x_{\text{best}} = x_{\text{true}} = x_{\text{super}(T)}\right\}}{\#\left\{x_{\text{best}} = x_{\text{true}}\right\}}, \tag{110}$$

respectively (see also Remark 3 in Section 5.3).

Tables. The results of Experiment 3 are displayed in Table 8. In this table, the rows correspond to all pairs of $n = 6, 7, 8$ and $T = 1, \dots, 7$, and the columns correspond to $\sigma = 0.25, 0.5, 0.75, 1, 1.25$. For each n, σ, T, the table contains two entries that appear one under the other: $\text{prop}_{\text{RaDe1}}(n, \sigma, T)$ (above) and $\text{prop}_{\text{super}}(n, \sigma, T)$ (below); see (109), (110). For example, for $n = 6$, $T = 1$ and $\sigma = 0.5$, these proportions are equal, respectively, to 0.211 and 0.411.

In addition, for each $n = 6, 7, 8$ and for $\sigma = 0.75$, we measure the CPU time required to process $L = 1,000$ messages by a single iteration of this algorithm. These CPU times are listed in the second column of Table 7.

Observations. The following observations can be made from Table 8 and some additional experiments by the authors.

1. Not surprisingly, for each n and σ the performance of the scheme improves as the number of iterations T increases (obviously, this improvement comes at the cost of additional CPU time). See, however, Remark 4 below.

2. As expected, for each n, σ and T, the performance of the scheme with supercharging is better than without supercharging (again, at the cost of additional CPU time).

3. The effects of supercharging are generally rather negligible when the proportion of "correct guesses" is already high. For example, when $n = 8$, $\sigma = 0.25$ and $T = 7$, supercharging improves the success rate from 93.1% to 97.4%. On the other hand, the improvement might be quite noticeable when the success rate is relatively low. For example, when $n = 8$, $\sigma = 0.75$ and $T = 7$, supercharging improves the success rate from 30% to 49%.

4. As expected, for each n and T the performance of the scheme deteriorates as σ increases (noise of a larger standard deviation makes the decoding task more difficult).

Remark 4. *Additional numerical experiments seem to indicate that, for sufficiently large noise, the success rate of RaDe1 does not approach 100% as the number of iterations increases to some reasonable value (e.g. $T = 100$), but seems to be stuck at some intermediate value (e.g. 70%). This phenomenon is likely to be related to the fact that RaDe1 is not fully*

		σ				
n	min_RaDe1	0.25	0.5	0.75	1	1.25
6	1	0.587E+00	0.211E+00	0.103E+00	0.737E-01	0.588E-01
		0.749E+00	0.411E+00	0.268E+00	0.219E+00	0.168E+00
6	2	0.737E+00	0.342E+00	0.185E+00	0.128E+00	0.802E-01
		0.843E+00	0.549E+00	0.381E+00	0.316E+00	0.222E+00
6	3	0.791E+00	0.427E+00	0.267E+00	0.207E+00	0.142E+00
		0.879E+00	0.620E+00	0.465E+00	0.404E+00	0.337E+00
6	4	0.836E+00	0.478E+00	0.297E+00	0.212E+00	0.160E+00
		0.904E+00	0.667E+00	0.502E+00	0.426E+00	0.353E+00
6	5	0.858E+00	0.509E+00	0.343E+00	0.258E+00	0.198E+00
		0.922E+00	0.688E+00	0.545E+00	0.472E+00	0.422E+00
6	6	0.878E+00	0.554E+00	0.355E+00	0.296E+00	0.222E+00
		0.932E+00	0.719E+00	0.558E+00	0.473E+00	0.409E+00
6	7	0.892E+00	0.582E+00	0.374E+00	0.317E+00	0.225E+00
		0.940E+00	0.735E+00	0.574E+00	0.494E+00	0.404E+00
7	1	0.678E+00	0.310E+00	0.155E+00	0.900E-01	0.622E-01
		0.839E+00	0.526E+00	0.334E+00	0.227E+00	0.174E+00
7	2	0.793E+00	0.419E+00	0.242E+00	0.155E+00	0.137E+00
		0.902E+00	0.631E+00	0.465E+00	0.325E+00	0.267E+00
7	3	0.844E+00	0.487E+00	0.293E+00	0.182E+00	0.137E+00
		0.929E+00	0.689E+00	0.514E+00	0.366E+00	0.324E+00
7	4	0.875E+00	0.541E+00	0.348E+00	0.245E+00	0.192E+00
		0.942E+00	0.731E+00	0.562E+00	0.444E+00	0.373E+00
7	5	0.903E+00	0.576E+00	0.364E+00	0.261E+00	0.199E+00
		0.955E+00	0.756E+00	0.575E+00	0.446E+00	0.376E+00
7	6	0.910E+00	0.600E+00	0.400E+00	0.273E+00	0.210E+00
		0.959E+00	0.771E+00	0.605E+00	0.465E+00	0.376E+00
7	7	0.924E+00	0.624E+00	0.426E+00	0.307E+00	0.220E+00
		0.964E+00	0.790E+00	0.619E+00	0.513E+00	0.412E+00
8	1	0.757E+00	0.288E+00	0.129E+00	0.631E-01	0.367E-01
		0.900E+00	0.514E+00	0.296E+00	0.184E+00	0.147E+00
8	2	0.845E+00	0.393E+00	0.186E+00	0.105E+00	0.653E-01
		0.941E+00	0.617E+00	0.376E+00	0.247E+00	0.188E+00
8	3	0.883E+00	0.446E+00	0.233E+00	0.122E+00	0.898E-01
		0.959E+00	0.658E+00	0.425E+00	0.263E+00	0.216E+00
8	4	0.902E+00	0.475E+00	0.250E+00	0.154E+00	0.114E+00
		0.965E+00	0.683E+00	0.457E+00	0.302E+00	0.257E+00
8	5	0.910E+00	0.508E+00	0.278E+00	0.152E+00	0.102E+00
		0.965E+00	0.697E+00	0.475E+00	0.327E+00	0.282E+00
8	6	0.920E+00	0.539E+00	0.290E+00	0.173E+00	0.114E+00
		0.968E+00	0.719E+00	0.486E+00	0.357E+00	0.278E+00
8	7	0.931E+00	0.546E+00	0.300E+00	0.184E+00	0.114E+00
		0.974E+00	0.729E+00	0.493E+00	0.350E+00	0.294E+00

Table 8: *Success rate of RaDe1, with and without supercharging ($k_1 = 2n^2$). Corresponds to Experiment 3 in Section 5.4.*

randomized, i.e. the parameter j from Table 1 (see also Section 4.6.2) is fixed (and is determined by H). To get rid of this undesirable feature, in the future we will investigate the possibility of choosing j at random, probably taking into account the values $S_{n-1}(j)$ (see (50) *in Section 4.6.1, and also Table 1).*

Some additional observations can be made from Table 7.

5. A single iteration of RaDe1 (see Section 4.6.3) is faster than the nearest search in X with $k = 2 \cdot n^2$ (see Section 4.5) by a factor of about 1.6 for $n = 6$ and by a factor of about 5 for $n = 8$. Needless to say, for larger values of k the difference in CPU times is even more significant. For example, for $n = 8$ and $k = n^5$ the nearest neighbors search in X is about 800 times slower.

6. Since the CPU time required by a a nearest neighbors search in X with $k = 2 \cdot n^2$ is of the same order of magnitude as that required by a single iteration of RaDe1, it might make sense to perform fewer iterations of the basic scheme followed by a supercharging rather than performing more iterations of the basic scheme without supercharging (see also Table 8).

5.5 Experiment 4

In this experiment, we investigate the performance of Randomized Decoder 2 (see Section 4.7).

Description. This experiment is similar to that described in Section 5.4, with the difference that instead of RaDe1 (see Section 4.6) we use RaDe2 (see Section 4.7). More specifically, for each $n = 6, 7, 8$ and each $\sigma = 0.25, 0.5, 0.75, 1, 1.25$, we proceed as in Experiment 1 from Section 5.2. Then, for each received message y_{obs} and each $T = 1, \ldots, 7$, we evaluate the estimate $x_{\text{RaDe2}}(T)$ of x_{true} from y_{obs} via RaDe2, with the following parameters (see (80), (86) in Section 4.7.4, and also Table 2 in Section 5.1.3):

$$\begin{aligned} &\text{min\_RaDe2} = T, \\ &\text{max\_RaDe2} = T, \\ &\chi_{\text{stop}} = 10^{-3}, \\ &\chi_{\text{thresh}} : \text{not applicable} \end{aligned} \tag{111}$$

(the parameter χ_{thresh} is not applicable since min_RaDe2 = max_RaDe2 = T, e.g. the number of basic iterations is always exactly T). Then, we define the integer k_1 via (107) above, and evaluate the estimate $x_{\text{super}}(T)$ of x_{true} from y_{obs} via RaDe2 (where the parameters are defined via (111)) followed by supercharging (see Section 4.8, in particular (98)); during the supercharging step, k_1 nearest neighbors are used. In other words, $x_{\text{RaDe2}}(T)$ and $x_{\text{super}}(T)$ are obtained from y_{obs} via calling

$$\begin{aligned} &\text{RaDe2\_search}\,(y_{\text{obs}}; x_{\text{RaDe2}}(T), r, \chi) \\ &\text{RaDe2\_search}\,(y_{\text{obs}}; x, r, \chi) \\ &\text{super}\,(y_{\text{obs}}; k_1, x, r, \chi; x_{\text{super}}(T), r_{\text{super}}, \chi_{\text{super}}) \end{aligned} \tag{112}$$

(see (86), (98)), where the parameters are defined via (111) and (107).

Thus, for each n and σ, we have 5,000 messages y_{obs} (five matrices per n, $L = 1,000$ messages per matrix, as in Experiments 1,2 from Sections 5.2, 5.3, 5.4); for each such message, we obtain 14 estimates $x_{\text{RaDe2}}(1), \ldots, x_{\text{RaDe2}}(7)$ and $x_{\text{super}}(1), \ldots, x_{\text{super}}(7)$ of x_{true}. For each such estimate, we evaluate the proportion of the cases in which this estimate is correct (provided that $x_{\text{true}} = x_{\text{best}}$, see Section 5.2). In other words, we compute $\text{prop}_{\text{RaDe2}}(n, \sigma, T)$ and $\text{prop}_{\text{super}}(n, \sigma, T)$ via the formulae

$$\text{prop}_{\text{RaDe2}}(n, \sigma, T) = \frac{\#\left\{x_{\text{best}} = x_{\text{true}} = x_{\text{RaDe2}(T)}\right\}}{\#\left\{x_{\text{best}} = x_{\text{true}}\right\}} \tag{113}$$

and

$$\text{prop}_{\text{super}}(n, \sigma, T) = \frac{\#\left\{x_{\text{best}} = x_{\text{true}} = x_{\text{super}(T)}\right\}}{\#\left\{x_{\text{best}} = x_{\text{true}}\right\}}, \tag{114}$$

respectively (see also Remark 3 in Section 5.3).

Tables. The results of Experiment 4 are displayed in Table 9, whose structure is similar to that of Table 8. More specifically, in this table, the rows correspond to all pairs of $n = 6, 7, 8$ and $T = 1, \ldots, 7$, and the columns correspond to $\sigma = 0.25, 0.5, 0.75, 1, 1.25$. For each n, σ, T, the table contains two entries that appear one under the other: $\text{prop}_{\text{RaDe2}}(n, \sigma, T)$ (above) and $\text{prop}_{\text{super}}(n, \sigma, T)$ (below); see (113), (114). For example, for $n = 6$, $T = 1$ and $\sigma = 0.5$, these proportions are equal, respectively, to 0.533 and 0.775.

In addition, for each $n = 6, 7, 8$ and for $\sigma = 0.75$, we measure the CPU time required to process $L = 1,000$ messages by a single iteration of this algorithm. These CPU times are listed in the third column of Table 7.

Observations. The following observations can be made from Table 9 and some additional experiments by the authors. These observations are somewhat similar to those from Section 5.4.

1. Not surprisingly, for each n and σ the performance of the scheme improves as the number of iterations T increases (obviously, this improvement comes at the cost of additional CPU time). See, however, Remark 5 below.

2. As expected, for each n, σ and T, the performance of the scheme with supercharging is better than without supercharging (again, at the cost of additional CPU time).

3. The effects of supercharging are generally rather negligible when the proportion of "correct guesses" is already high. For example, when $n = 8$, $\sigma = 0.25$ and $T = 3$, supercharging improves the success rate from 93.5% to 96.6%. On the other hand, the improvement might be quite noticeable when the success rate is relatively low. For example, when $n = 8$, $\sigma = 0.75$ and $T = 2$, supercharging improves the success rate from 31.4% to 50.6%.

4. As expected, for each n and T the performance of RaDe2 deteriorates as σ increases (noise of a larger standard deviation makes the decoding task more difficult).

Remark 5. *Additional numerical experiments seem to indicate that, for sufficiently large noise, the success rate of RaDe2 does not approach 100% as the number of iterations increases to some reasonable value (e.g. $T = 100$), but seems to be stuck at some intermediate value (e.g. 70%). In addition, this intermediate value is larger that the related value of RaDe1 (see Remark 4 in Section 5.4). This phenomenon is likely to be related to the*

		σ				
n	min_RaDe2	0.25	0.5	0.75	1	1.25
6	1	0.969E+00	0.752E+00	0.533E+00	0.429E+00	0.302E+00
		0.987E+00	0.902E+00	0.775E+00	0.696E+00	0.612E+00
6	2	0.989E+00	0.871E+00	0.713E+00	0.609E+00	0.529E+00
		0.997E+00	0.955E+00	0.870E+00	0.804E+00	0.735E+00
6	3	0.995E+00	0.912E+00	0.798E+00	0.697E+00	0.642E+00
		0.998E+00	0.965E+00	0.915E+00	0.849E+00	0.821E+00
6	4	0.996E+00	0.938E+00	0.841E+00	0.771E+00	0.709E+00
		0.998E+00	0.978E+00	0.930E+00	0.892E+00	0.866E+00
6	5	0.998E+00	0.952E+00	0.871E+00	0.807E+00	0.781E+00
		0.100E+01	0.984E+00	0.948E+00	0.926E+00	0.874E+00
6	6	0.998E+00	0.962E+00	0.892E+00	0.839E+00	0.807E+00
		0.999E+00	0.985E+00	0.960E+00	0.930E+00	0.912E+00
6	7	0.998E+00	0.968E+00	0.909E+00	0.860E+00	0.834E+00
		0.100E+01	0.988E+00	0.962E+00	0.949E+00	0.914E+00
7	1	0.838E+00	0.536E+00	0.349E+00	0.262E+00	0.202E+00
		0.895E+00	0.714E+00	0.562E+00	0.448E+00	0.446E+00
7	2	0.918E+00	0.657E+00	0.488E+00	0.387E+00	0.334E+00
		0.948E+00	0.804E+00	0.673E+00	0.578E+00	0.510E+00
7	3	0.954E+00	0.727E+00	0.556E+00	0.440E+00	0.396E+00
		0.973E+00	0.854E+00	0.735E+00	0.630E+00	0.570E+00
7	4	0.963E+00	0.784E+00	0.598E+00	0.494E+00	0.415E+00
		0.978E+00	0.887E+00	0.757E+00	0.688E+00	0.624E+00
7	5	0.976E+00	0.802E+00	0.648E+00	0.535E+00	0.456E+00
		0.985E+00	0.896E+00	0.785E+00	0.690E+00	0.655E+00
7	6	0.981E+00	0.834E+00	0.674E+00	0.564E+00	0.503E+00
		0.989E+00	0.913E+00	0.818E+00	0.735E+00	0.699E+00
7	7	0.982E+00	0.859E+00	0.705E+00	0.619E+00	0.516E+00
		0.988E+00	0.934E+00	0.837E+00	0.748E+00	0.692E+00
8	1	0.764E+00	0.370E+00	0.229E+00	0.158E+00	0.155E+00
		0.863E+00	0.554E+00	0.393E+00	0.281E+00	0.224E+00
8	2	0.888E+00	0.494E+00	0.314E+00	0.242E+00	0.200E+00
		0.941E+00	0.699E+00	0.506E+00	0.427E+00	0.347E+00
8	3	0.935E+00	0.586E+00	0.400E+00	0.298E+00	0.249E+00
		0.966E+00	0.760E+00	0.588E+00	0.480E+00	0.400E+00
8	4	0.958E+00	0.652E+00	0.420E+00	0.334E+00	0.290E+00
		0.980E+00	0.813E+00	0.626E+00	0.523E+00	0.465E+00
8	5	0.970E+00	0.704E+00	0.461E+00	0.363E+00	0.306E+00
		0.982E+00	0.841E+00	0.665E+00	0.568E+00	0.527E+00
8	6	0.977E+00	0.732E+00	0.512E+00	0.393E+00	0.302E+00
		0.989E+00	0.859E+00	0.697E+00	0.598E+00	0.502E+00
8	7	0.977E+00	0.759E+00	0.545E+00	0.417E+00	0.339E+00
		0.992E+00	0.873E+00	0.717E+00	0.622E+00	0.518E+00

Table 9: *Success rate of RaDe2, with and without supercharging ($k_1 = 2n^2$). Corresponds to Experiment 4 in Section 5.5.*

fact that RaDe2 is not fully randomized, i.e. the parameters j_1, j_2 from Table 1 (see also Section 4.7.2) are fixed (and are determined by H). To get rid of this undesirable feature, in the future we will investigate the possibility of choosing j_1, j_2 at random, probably taking into account the values $S_{n-2}(j_1), S_{n-2}(j_2)$ (see (50) in Section 4.6.1, and also Table 1).

Some additional observations can be made from Table 7.

5. A single iteration of RaDe2 (see Section 4.7.3) is typically somewhat slower than a single iteration of RaDe1 (by a factor between 2.4 and 3.1 for $n = 6, 7, 8$ and $\sigma = 0.75$). Obviously, this is expected from (67), (91).

6. Since the CPU time required by a a nearest neighbors search in X with $k = 2 \cdot n^2$ is of the same order of magnitude as that required by a single iteration of RaDe2, it might make sense to perform fewer iterations of the basic scheme followed by a supercharging rather than performing more iterations of the basic scheme without supercharging (see also Table 9).

5.6 Experiment 5

In this experiment, we compare the performance of the three decoding schemes: nearest neighbors search in X, RaDe1 and RaDe2 (see Sections 4.5, 4.6, 4.7, respectively). Roughly speaking, the purpose of this experiment is to determine empirically how long it takes for each of the schemes to achieve a certain success rate (for several choices of n and noise deviation σ). In this sense, this experiment can be viewed simply as a comparison of the results of Experiments 2,3,4 above. For the sake of completeness, however, we provide a detailed description below.

For each $n = 6, 7, 8$ and each $\sigma = 0.25, 0.75, 1.25$, we proceed as follows. First, we repeat Experiment 1 from Section 5.2; thus, for each of the 5,000 randomly selected transmitted messages x_{true} in X, we obtain a received message y_{obs} in $\mathbb{C}^n$ and a maximum likelihood estimate x_{best} of x_{true} (see Section 5.2). As in Experiments 2,3,4 above, to evaluate the performance of a decoding scheme we use only those messages for which $x_{\text{best}} = x_{\text{true}}$ (see Remark 3 in Section 5.3).

Then we choose, more or less arbitrarily, the desired success rate $0 < p < 1$, and, for each of the schemes from Sections 4.5, 4.6, 4.7, we choose the parameters in such a way that the success rate of the scheme is roughly equal to p (see e.g. (105) in Section 5.3, (109), (110) in Section 5.4, (113), (114) in Section 5.5). More specifically:

for the nearest neighbors search in X, we select the appropriate number of nearest neighbors k (see Section 4.5);

for RaDe1, we select the number of iterations T, the number of nearest neighbors k_1 for supercharging (zero if no supercharging; see also (107)), and use the parameters from (106);

for RaDe2, select the number of iterations T, the number of nearest neighbors k_1 for supercharging (zero if no supercharging; see also (107)), and use the parameters from (111).

For each of the schemes from Sections 4.5, 4.6, 4.7, we measure the proportion of correct guesses ($\text{prop}_{\text{nn}}(n, \sigma)$, $\text{prop}_{\text{RaDe1}}(n, \sigma)$, $\text{prop}_{\text{RaDe2}}(n, \sigma)$, respectively; see (105), (109), (110), (113), (114)). In addition, for each scheme, we measure the CPU times required to decode 1,000 messages (see e.g. Table 7).

Tables. The results of this experiment are displayed in Tables 10 – 18. Each of this tables corresponds to a certain choice of n and σ, and has the following structure. The columns correspond to the decoding schemes from Sections 4.5, 4.6, 4.7, respectively (i.e. nearest neighbors search in X, RaDe1 and RaDe2). In the first row, we list the scheme's parameters (k for the nearest neighbors search in X, number of iterations T and supercharging parameter k_1 for each of RaDe1 and RaDe2). The second row contains the proportion of correctly decoded messages (provided that $x_{\text{true}} = x_{\text{best}}$), i.e. $\text{prop}_{\text{nn}}(n, \sigma)$, $\text{prop}_{\text{RaDe1}}(n, \sigma)$, $\text{prop}_{\text{RaDe2}}(n, \sigma)$, respectively (by design, we expect these proportions to be roughly the same across all columns). The third row contains the CPU time (in seconds) required to decode 1,000 messages; these times are used to compare the performance of the schemes (the faster the better).

For example, for $n = 6$ and $\sigma = 0.25$ (see Table 10), it takes the scheme from Section 4.5 (with $k = 2n^2 + 1$) to achieve the success rate of 91% in 0.075 seconds per 1,000 messages; on the other hand, one iteration of RaDe2 (without supercharging) achieves the success rate of 97% in 0.052 seconds per 1,000 messages.

	NN in X	RaDe1	RaDe2
parameters	$k = 2n^2 + 1$	4 iterations, $k_1 = 2n$	1 iteration, $k_1 = 0$
proportion	0.91E+0	0.90E+0	0.97E+0
CPU time	0.75E-1	0.20E+0	0.52E-1

Table 10: *Corresponds to Experiment 5 with $n = 6$, $\sigma = 0.25$ (see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = 2n^2 + 1$	2 iterations, $k_1 = 2n$	2 iterations, $k_1 = 0$
proportion	0.90E+0	0.90E+0	0.92E+0
CPU time	0.14E+0	0.11E+0	0.10E+0

Table 11: *Corresponds to Experiment 5 with $n = 7$, $\sigma = 0.25$ (see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^5 + 1$	2 iterations, $k_1 = 0$	2 iterations, $k_1 = 0$
proportion	0.82E+0	0.85E+0	0.88E+0
CPU time	0.45E+2	0.10E+0	0.13E+0

Table 12: *Corresponds to Experiment 5 with $n = 8$, $\sigma = 0.25$ (see Section 5.6).*

Observations. The following observations can be made from Tables 10 – 18 and some additional experiments by the authors. In all of these observations, by "better performance" we mean "takes less CPU time to achieve a similar success rate". All the times are in seconds per 1,000 messages.

1. When the noise is relatively small ($\sigma = 0.25$), the performance of RaDe1 is similar to that of RaDe2 (moreover, even a single iteration of each scheme takes roughly the same

	NN in X	RaDe1	RaDe2
parameters	$k = n^4$	20 iterations, $k_1 = 2n^2$	3 iterations, $k_1 = 2n^2$
proportion	0.72E+0	0.70E+0	0.71E+0
CPU time	0.13E+1	0.10E+1	0.27E+0

Table 13: *Corresponds to Experiment 5 with* $n = 6$, $\sigma = 0.75$ *(see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^4$	9 iterations, $k_1 = 2n^2$	2 iterations, $k_1 = 2n^2$
proportion	0.65E+0	0.65E+0	0.67E+0
CPU time	0.27E+1	0.58E+0	0.35E+0

Table 14: *Corresponds to Experiment 5 with* $n = 7$, $\sigma = 0.75$ *(see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^5 + 1$	20 iterations, $k_1 = 2n^2$	3 iterations, $k_1 = 0$
proportion	0.58E+0	0.57E+0	0.59E+0
CPU time	0.45E+2	0.13E+1	0.61E+0

Table 15: *Corresponds to Experiment 5 with* $n = 8$, $\sigma = 0.75$ *(see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^4$	25 iterations, $k_1 = 2n^2$	1 iteration, $k_1 = 2n^2$
proportion	0.60E+0	0.59E+0	0.61E+0
CPU time	0.13E+1	0.12E+1	0.30E+0

Table 16: *Corresponds to Experiment 5 with* $n = 6$, $\sigma = 1.25$ *(see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^5 + 1$	50 iterations, $k_1 = 2n^2$	4 iterations, $k_1 = 2n^2$
proportion	0.60E+0	0.59E+0	0.63E+0
CPU time	0.19E+2	0.26E+1	0.88E+0

Table 17: *Corresponds to Experiment 5 with* $n = 7$, $\sigma = 1.25$ *(see Section 5.6).*

	NN in X	RaDe1	RaDe2
parameters	$k = n^5 + 1$	20 iterations, $k_1 = 2n^2$	3 iterations, $k_1 = 2n^2$
proportion	0.36E+0	0.36E+0	0.38E+0
CPU time	0.45E+2	0.13E+1	0.82E+0

Table 18: *Corresponds to Experiment 5 with* $n = 8$, $\sigma = 1.25$ *(see Section 5.6).*

time; compare to Table 7 that corresponds to $\sigma = 0.75$, and see also (67), (91)). For example, for $n = 8$, it takes RaDe1 0.1 seconds to achieve the success rate of 85%, while it takes RaDe2 0.13 seconds to achieve the success rate of 88% (see Table 12).

2. Even for $\sigma = 0.25$, the performance of the nearest neighbors search in X strongly depends on n (for similar success rates). For example, when $n = 6$ or $n = 7$, this scheme achieves the success rate of about 90% in time comparable to that of RaDe1 and RaDe2 (see Tables 10, 11; in these, $k = 2n^2 + 1$). On the other hand, when $n = 8$, the nearest neighbors search in X requires already $k = n^5$ to achieve the success rate of 82%; subsequently, it is slower than both RaDe1 and RaDe2 schemes by a factor of about 40 (see Table 12).

3. When $\sigma = 0.75$, RaDe2 outperforms RaDe1 somewhat. More specifically, it achieves a slightly higher success rate (71% vs. $70n = 8$) faster (by a factor of 3.7 for $n = 6$, by a factor of 1.7 for $n = 7$, and by a factor of 2.1 for $n = 8$). In this case, while a single iteration of RaDe2 is slower than a single iteration of RaDe1 (see Table 7), the latter requires more iterations to achieve a similar success rate (see Tables 13 – 15).

4. When $\sigma = 0.75$, the nearest neighbors search in X is noticeably slower than RaDe2 (when the success rates are about 70%, 65% and 60% for $n = 6, 7, 8$, respectively). More specifically, it is slower by factors of 4.8, 7.7 and 74 for $n = 6, 7, 8$, respectively (see Tables 13 – 15).

5. When the noise is large ($\sigma = 1.25$), both RaDe1 and RaDe2 typically perform somewhat better with supercharging than without it. In addition, in all experiments, RaDe2 outperforms RaDe1 (by factors of about 4,3 and 1.5 with success rates of about 60%, 60% and 37% for $n = 6, 7, 8$, respectively).

6. For $\sigma = 1.25$, the algorithm from Section 4.5 requires a relatively large number k of nearest neighbors to achieve a success rate similar to that of RaDe1 and RaDe2, in all experiments (see Tables 13 – 15). Consequently, the nearest neighbors search in X is slower than, say, RaDe2, by a factor of 4.3, 22 and 55 with success rates about 60%, 60% and 37% for $n = 6, 7, 8$, respectively.

We conclude that the nearest neighbors search in X typically underperforms compared to both RaDe1 and RaDe2 decoding schemes (i.e. takes more time to achieve a similar success rate); moreover, for $n = 8$ the difference can be quite noticeable. On the other hand, RaDe2 typically outperforms RaDe1.

6 Conclusions and Future Research

In this paper, we presented several schemes for decoding of digital messages sent over a noisy multivariate Gaussian channel, and illustrated their performance via numerical experiments.

Needless to say, there is always a gap between a prototype dealing with a single mathematical model and the multitude of applications; we are looking forward to collaborating with researchers and engineers in both industry and academia to shrink this gap in the case of our decoding schemes.

Some future directions of our research in this area are related to the fact that our schemes need to be extensively tested in a demanding industrial environment on various choices of C, H, σ, etc.; this will lead to further improvements and modifications. Other developments are related to the fact that the decoding problem described above admits certain variations

(frequently encountered in applications). For example, often the coordinates of the transmitted message x are not independent of each other (e.g. there is a certain redundancy, as is in case of parity checks etc.). In other words, the set of possible message is not all of X but rather a subset of X (the reader is referred, for example, to [2], [5], [11], [12], [13], [14], [17], for further information on the subject). In addition, certain messages might be more likely to be transmitted than others (based, for instance, on long-term statistics). Obviously, our schemes will need to be modified to take all such additional information into account.

7 Acknowledgments

The author would like to thank Linda Ness and Joe Liberty from Applied Communication Sciences for numerous helpful discussions.

References

[1] M. Abramowitz, I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, Dover Publications, 1964.

[2] S. M. Alamouti, *A simple transmitter diversity scheme for wireless communications*, IEEE J. Select. Areas Commun., Vol. 16, pp. 1451–1458 (Oct 1998).

[3] Arya, S., D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, *An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions*, Journal of the ACM, vol. 45, no. 6, pp. 891-923 (1998).

[4] P. Billingsley, *Probability and Measure*, Wiley, NY (1986).

[5] U. Erez, M. D. Trott, G. W. Wornell, *Rateless Coding for Gaussian Channels*, IEEE Trans. on Information Theory, Vol. 58, No. 10, pp. 530–547 (Feb 2012).

[6] W. Feller, *An Introduction to Probability Theory and Its Applications*, Volume II, Second edition, Wiley, NY (1957).

[7] G. J. Foschini, *Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas*, Bell Labs, Tech. J., Vol. 1, No. 2, pp. 41–59 (1996).

[8] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, Second edition, Oxford University Press (1992).

[9] G. Golub, C. V. Loan, *Matrix Computations*, Second Edition, Johns Hopkins University Press, Baltimore, 1989.

[10] N. Halko, P. G. Martinsson, J. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, Vol 53. No. 2, pp. 217–288 (2011).

[11] B. Hochwald, S. ten Brink, *Achieving Near-Capacity on a Multiple-Antenna Channel*, IEEE Transactions on communication, Vol. 51, No. 3 (March 2003).

[12] J. Jalden, B. Ottersten, *On the complexity of sphere decoding in digital communications*, IEEE Trans. Signal Proc., Vol. 53, No. 4, pp. 1474–1484 (April 2005).

[13] J. C. Koshy, J. C. Liberty, T. R. Hoerning, *A new low-complexity demapper for high-performance iterative MIMO: information-theoretic and ber analyses*, Telcordia Technologies, Applied Research, Red Bank, NJ.

[14] J. C. Koshy, J. C. Liberty, T. R. Hoerning, *Iterative MIMO detector using a group-wise approach*, Telcordia Technologies, Applied Research, Red Bank, NJ.

[15] Peter W. Jones, Andrei Osipov, Vladimir Rokhlin, *A Randomized Approximate Nearest Neighbors Algorithm*, Applied and Computational Harmonic Analysis, July 2012.

[16] Peter W. Jones, Andrei Osipov, Vladimir Rokhlin, *Randomized Approximate Nearest Neighbors Algorithm*, *Proceedings of the National Academy of Sciences*, PNAS 2011 108 (38) 15679-15686

[17] D. L. Romero, N. B. Chang, A. R. Margetts, *Practical non-binary rateless codes for wireless channels*, IEEE (2013).

[18] G. Strang, *Introduction to Linear Algebra (3rd ed.)*, Wellesley-Cambridge Press, 1998.

[19] I. E. Telatar, *Capacity of multi-antenna Gaussian channels*, Eur. Trans. Telecommun., vol. 10, pp. 585–595 (Nov 1999).