ulm university universität
uulm

Utilization of Sparsity Awareness in the Simplex Algorithm

Masterthesis
of
Thomas Leichtle

Institute of Communications Engineering
Ulm University

May 2014

M/2014/HZ/02

ulm university universität
uulm

Institute of Communications Engineering

MASTER THESIS

Utilization of Sparsity Awareness in the Simplex Algorithm

Explanation

The principle of Compressed Sensing can be described as the problem of finding the sparsest solution to an underdetermined linear system of equations. In order to solve this generally intractable problem, several approaches have been proposed in the past. The well-known Basis Pursuit approach applies a convex ℓ_1-relaxation and transforms the underdetermined system of equations into a so called linear program. These special optimization problems can be solved efficiently by several algorithms, e.g. by the Simplex algorithm. However, due to the necessary convex relaxation, the found solution is not necessarily always the sparsest. Sparse solutions occur at so called degenerated corners in the Simplex algorithm.
The objective of this thesis is to implement the Simplex algorithm and to investigate the possibility of utilizing degenerated corners as indicator for sparse solutions. The resulting sparsity aware Simplex algorithm should obtain the desired sparse solution more often than the conventional convex relaxation approach. Within this thesis, it should be investigated by numerical evaluations to which extend the sparsity awareness can be utilized and at which computational costs.

Submission date: 20.05.2014

Candidate: Thomas Leichtle

Supervisor: Dipl.-Ing. Henning Zörlein

First Examiner: Prof. Dr.-Ing. Martin Bossert

Second Examiner: Prof. Dr. Uwe Schöning

Catalog No.: M/2014/HZ/02

I confirm, that the presented Masterthesis is an original work completed independently without inadmissible outside help.

Ulm, 20.05.2014

(Thomas Leichtle)

Contents

1. Introduction

The field of Compressed Sensing (CS) has become popular over the last decade. Current research focuses on several problems, but the main topics are the optimization of sensing matrices for different applications and the improvement of algorithms for reconstructing a sparse vector from a compressed one. This thesis regards the second.
A huge leap in CS was made, when [1] showed, that the so called ℓ_1-optimization for reconstruction leads for many matrices to the sparsest solution for which the direct search is not feasible. There are two reasons, why this increased the interest in the field of CS: They proved the equality, so a successful reconstruction is guaranteed, even if the provided bounds are bad compared to simulation results [2]. And second, the proposed method, Basis Pursuit, can be formulated as linear program. With the Simplex algorithm [3], an efficient solver for these is known since 1955 and so CS became applicable. Since then, many alternatives have been developed, e.g. the interior point method for linear programs, which is used as default setting in MATLABs linprog. But also other approaches, like heuristic algorithms performing greedy choices, e.g. OMP [2] or CoSaMP [4], have become popular.
The Simplex algorithm solves general linear optimization problems and has not been optimized for the application on sparse vectors in CS so far. It turns out that the solution of the sparse reconstruction problem is always located in a so called degenerated corner. The possibility to adapt the Simplex algorithm to recognize and specifically look for those corners, up to a certain extend, is provided. The range of this approach is investigated in this thesis.

In Chapter 2, the fundamental principles of CS are given. Afterwards, in Chapter 3, the Simplex algorithm and modifications to obtain a sparsity aware Simplex algorithm are described in detail. The analysis is provided in Chapter 4. In Chapter 5, some obstacles for implementation and possible countermeasures are presented. At the end, Chapter 6 concludes the work and outlines further investigations.

2. Principles of Compressed Sensing

2.1. The Problem Statement

Compressed Sensing is a very simple and efficient data acquisition protocol [5]. It reduces a N-dimensional data vector $\boldsymbol{x}$ to a smaller measurement vector $\boldsymbol{\mu}$ by projecting it onto a M-dimensional subspace with $M < N$

$$\boldsymbol{\Phi x} = \boldsymbol{\mu} \ . \tag{2.1}$$

On one hand, this data acquisition procedure is rather simple, compared to other methods. On the other hand it is complicated to reconstruct the original vector $\boldsymbol{x}$ from the compressed vector $\boldsymbol{\mu}$ and this is the main problem in CS.

The reconstruction is only possible under certain conditions. First, the original vector $\boldsymbol{x}$ must be sparse, which means most of its components are zero. *Sparsity* is one main keyword in the field of compressed sensing.

Definition 2.1.1 (Sparsity)
The sparsity k of a vector $\boldsymbol{x}$ is given by the number of elements x_i of $\boldsymbol{x}$ with $x_i \neq 0$. E.g. sparsity $\{\mathbf{0}\} = 0$ *and* sparsity $\{\begin{pmatrix}1 & 0 & 2 & 0 & 5\end{pmatrix}\} = 3$.

How sparse $\boldsymbol{x}$ needs to be, in order to recover it, depends also on the dimensions N and M. For good compression, it is desired that $M \ll N$, but the larger $\frac{N}{M}$ is, the harder the reconstruction task. The second important factor is the so called *sensing matrix* $\boldsymbol{\Phi}$. It is shown, that the reconstruction works differently well for different types of matrices. This is explained in the next section.

2.2. Suitable Sensing Matrices

2.2.1. Properties

There are several possible types of matrices which can be used as sensing matrices. To describe their quality for reconstruction, the following properties were introduced:

Null Space Property (NSP)

In order to reconstruct the $k-$sparse vector $\boldsymbol{x}$ from $\boldsymbol{\mu}$, it is necessary that for each two different vectors $\boldsymbol{x}$, $\boldsymbol{x'}$ with sparsity less or equal to k, there exists no common $\boldsymbol{\mu}$, i.e. $\boldsymbol{Ax} \neq \boldsymbol{Ax'}$. If $\boldsymbol{Ax} = \boldsymbol{Ax'}$, then $\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x'}) = \boldsymbol{0}$, where $(\boldsymbol{x} - \boldsymbol{x'})$ is at most $2k$-sparse.
The null space of $\boldsymbol{A}$ is denoted by

$$\mathcal{N}(\boldsymbol{A}) := \{\boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{Ax} = \boldsymbol{0}\} \ ,$$

and let

$$\Sigma_k := \{\boldsymbol{x} \in \mathbb{R}^N \ : \ \text{sparsity}\{\boldsymbol{x}\} \leq k\}$$

be the set with all vectors with sparsity $\leq k$.

Theorem 2.2.1
Given $\boldsymbol{A} \in \mathbb{R}^{M \text{x} N}$, the following statements are equivalent [6]:

1. *$\Sigma_{2k} \cap \mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{0}\}$, i.e. the null space does not contain vectors with sparsity $\leq 2k$ with exception of the $\boldsymbol{0}$ vector.*
2. *Choose any $2k$ columns from $\boldsymbol{A}$, then this matrix has a full rank of $2k$.*
3. *$\boldsymbol{x} \in \Sigma_k$, then $\boldsymbol{\mu} = \boldsymbol{Ax}$ is a unique 1:1 mapping from $\boldsymbol{x}$ to $\boldsymbol{\mu}$ and vice versa.*

This means a unique representation of a vector with sparsity less or equal to k occurs if and only if $\mathcal{N}(\boldsymbol{A})$ does not contain any vector with sparsity less or equal to $2k$. Unfortunately, it is NP-hard to check whether a matrix satisfies the NSP or not, and therefore, practically not possible.

Restricted Isometry Property (RIP)

Definition 2.2.1 (Restricted Isometry Constant [1])
The k-Restricted Isometry Constant (RIC) δ_k is the smallest number, such that

$$(1 - \delta_k)\|\boldsymbol{x}\|_{\ell_2}^2 \leq \|\boldsymbol{A}\boldsymbol{x}\|_{\ell_2}^2 \leq (1 + \delta_k)\|\boldsymbol{x}\|_{\ell_2}^2$$

holds for all $\boldsymbol{x}$ with sparsity less or equal to k.

This constant gives information how well a matrix $\boldsymbol{A}$ preserves the length of a vector $\boldsymbol{x}$ at the projection onto the subspace by $\boldsymbol{A}\boldsymbol{x}$. If δ_k is very small, the projected length is approximately as large as the original length, and therefore, the distance between any pair of k-sparse vectors is preserved.
If there exists a $\delta_k \in (0, 1)$ for a matrix $\boldsymbol{A}$, then this matrix satisfies the RIP of order k with δ_k.

Mutual Incoherence Property (MIP)

Another important property for successful reconstruction is the MIP.

Definition 2.2.2 (Coherence [7])
Let $\boldsymbol{A} \in \mathbb{R}^{M \mathrm{x} N}$ be a matrix with $\|\cdot\|_{\ell_2}$-normalized columns $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N$, i.e. $\|\boldsymbol{a}_i\|_{\ell_2} = 1$ for all $i \in \{1, \ldots, N\}$. The coherence of the matrix $\boldsymbol{A}$ is defined by

$$\eta(\boldsymbol{A}) := \max_{1 \leq i \neq j \leq N} |\langle \boldsymbol{a}_i, \boldsymbol{a}_j \rangle| \ ,$$

with the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$: $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{N} x_i y_i$.

It was shown that matrices with a small coherence guarantee a successful reconstruction by certain approaches [8].

2.2.2. Random Sensing Matrices

In [1], it is shown that certain random matrices fulfill the RIP with a very high probability. Random matrices based on two suitable distribution functions are mentioned below.

Bernoulli Sensing Matrix

Matrices with entries consisting of independent, identical distributed (iid) realizations of ± 1 Bernoulli random variables with probability $p = \frac{1}{2}$ can be used as sensing matrices and satisfy the RIP with high probability [9].

Gaussian Sensing Matrix

It is shown in [1] that if one draws the entries of a matrix $\mathbf{\Phi}$ iid Gaussian with zero mean and a variance $\frac{1}{p}$, then it satisfies the RIP 'with overwhelming probability'. Since Gaussian matrices are a popular choice for sensing matrices, they are used for evaluation and comparison in Chapter 4.

2.2.3. Deterministic and Optimized Sensing Matrices

Recently a lot of work has been done to optimize or construct matrices with respect to those properties mentioned above. Here, two examples are given.

BCH-based Matrices

The first class, proposed in [10], are sensing matrices created by using linear block codes, specifically BCH-codes. With this method, it is possible to create binary, bipolar or even ternary sensing matrices, that satisfy the RIP with a high order. The drawback of this procedure is the limitation to some fixed dimension ratios given by the construction method.

BASC-based Matrices

The second class is based on Best Antipodal Spherical Codes (BASC) and proposed in [11]. They minimize the coherence of the matrices. This optimizing procedure is applicable for all dimensions and that's the reason why BASC-based matrices are used within the scope of this thesis later on for comparison with Gaussian matrices.

2.3. Reconstruction

For the reconstruction task, a closer look on (2.1) is required.

Definition 2.3.1 (Reconstruction LSE)
Given a sparse vector $\boldsymbol{x} \in \mathbb{R}^N$ and a sensing matrix $\boldsymbol{\Phi} \in \mathbb{R}^{M \mathrm{x} N}$, the compressed (non-sparse) vector $\boldsymbol{\mu} \in \mathbb{R}^M$ is calculated by

$$\boldsymbol{\mu} = \boldsymbol{\Phi} \cdot \boldsymbol{x} \ .$$

As mentioned, it is desired to find an $\hat{\boldsymbol{x}}$, such that

$$\boldsymbol{\Phi} \cdot \hat{\boldsymbol{x}} = \boldsymbol{\mu} \ .$$

If $\boldsymbol{\Phi}$ satisfies the null space property (see Section 2.2.1), it follows $\hat{\boldsymbol{x}} = \boldsymbol{x}$. Ways to find a solution to this linear system of equations (LSE) are described in this section. A LSE can have in general three different cases:

1. **Overdetermined system:** If there are more equations than unknown variables, i.e. $M > N$, the system has no solution in general. Only if at most N rows are linear independent, the system can be reduced to one of the following cases.

2. **Square (exact) system:** The case with $M = N$ has in general (if $\boldsymbol{\Phi}$ has full rank) one unique solution. It can be calculated via the inverse matrix or by Gaussian elimination.

3. **Underdetermined system:** $M < N$, which is also the case in compressed sensing, leads to a system with infinite many solutions.

Having an underdetermined system of equations, it is still possible to reconstruct a single unique solution, but only with respect to some additional constraint. In many applications it is very common to calculate the Moore-Penrose pseudoinverse to resolve a unique $\hat{\boldsymbol{x}}$. This approach minimizes the ℓ_2-norm, and therefore, the energy of the solution, but there are also other possibilities.

Definition 2.3.2 (ℓ_p-Norms [8])
The (quasi) ℓ_p-norm or simply p-norm with $0 < p < \infty$ of a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$ is defined by

$$\|\boldsymbol{x}\|_{\ell_p} = \|\boldsymbol{x}\|_p := \left(\sum_{i=1}^{N} |x_i|^p \right)^{\frac{1}{p}}$$

and the ' ℓ_0-norm' is given by

$$\|\boldsymbol{x}\|_{\ell_0} := |\{i \ : \ x_i \neq 0\}| \ ,$$

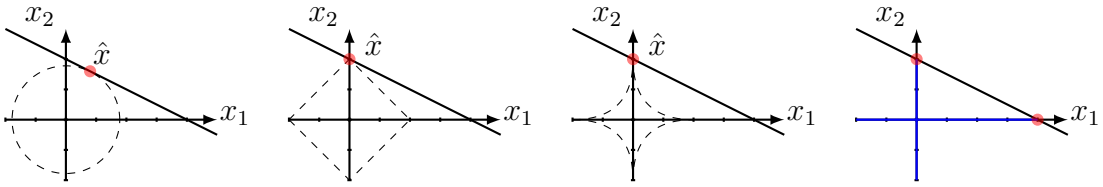
where $|\{\cdot\}|$ denotes the cardinality of the set. $\|\cdot\|_{\ell_p}$ is a norm for $1 \leq p < \infty$ and a quasi norm for $0 < p < 1$, which means it satisfies all norm axioms except the triangle equation. $\|\cdot\|_{\ell_0}$ is no norm, but is typically called one in the field of compressed sensing, and hence, the quotes.

An ℓ_p-ball in $\mathbb{R}^N$ is given by $B_p(r) = \{\boldsymbol{x} \in \mathbb{R}^N \ : \ \|\boldsymbol{x}\|_{\ell_p} \leq r\}$. For the search of the unique solution $\hat{\boldsymbol{x}}$ of an underdetermined system of equations with minimal $\|\hat{\boldsymbol{x}}\|_{\ell_p}$, one starts with $B_p(0)$ and inflates it by increasing r until one valid solution to $\boldsymbol{\Phi}\hat{\boldsymbol{x}} = \boldsymbol{\mu}$ is inside. In some pathological cases there even exist infinite many valid minimal solutions in respect to one $\|\cdot\|_{\ell_p}$.

Here a short example for $N = 2$ and $M = 1$. Choose $\boldsymbol{\Phi} = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\boldsymbol{x} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$

$$\boldsymbol{\mu} = \boldsymbol{\Phi}\boldsymbol{x} = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \end{pmatrix} \ .$$

First the problem is solved graphically by inflating ℓ_p-balls with $p \in \{2, 1, 0.5, 0\}$ as seen in Figure 2.1. The thick line is the constraint given by the matrix, the dotted curve and everything within is the ℓ_p-ball and the red dots mark the found solution. For the 'ℓ_0-norm' the ball consists of both axis marked blue. The solutions are: $p = 2 : \begin{pmatrix} 0.8 & 1.6 \end{pmatrix}^T$, $p = 1 : \begin{pmatrix} 0 & 2 \end{pmatrix}^T$, $p = 0.5 : \begin{pmatrix} 0 & 2 \end{pmatrix}^T$, '$\ell_0$-norm': $\begin{pmatrix} 0 & 2 \end{pmatrix}^T$.



(a) $p = 2$ (b) $p = 1$ (c) $p = 0.5$ (d) 'ℓ_0-norm'

Figure 2.1.: 2-dimensional example of how to find a unique solution of an undetermined system of equations by inflating ℓ_p-balls. The red dots mark the found solution.

It is possible, for such small dimensions, to retrieve the solutions from the plots. Also for $p = 2$, it can be calculated easily using the pseudo-invers as seen here:

$$\begin{aligned} \hat{\boldsymbol{x}} &= \boldsymbol{\Phi}^{\boldsymbol{T}} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\boldsymbol{T}}\right)^{-1} \cdot \boldsymbol{\mu} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \left(\begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right)^{-1} \cdot 4 \\ \hat{\boldsymbol{x}} &= \begin{pmatrix} 1 \\ 2 \end{pmatrix} \cdot \frac{1}{5} \cdot 4 = \begin{pmatrix} 0.8 \\ 1.6 \end{pmatrix} \end{aligned}$$

The problem is that for most cases, the ℓ_2-norm does not lead to the original vector $\boldsymbol{x}$. Thinking about the positions of the sparse solutions with $0-$sparse in the origin, $1-$sparse on the axis, and so on, one can conclude that the 'ℓ_0-ball' is optimal for the purpose of reconstruction. Taking a look at the definition of the ℓ_0-norm also explains why it is optimal, because it is the same as for the sparsity in Definition 2.1.1. So minimizing the ℓ_0-norm corresponds directly to the search for the sparsest solution.

For correct reconstruction, one has to solve the following:

$$\hat{\boldsymbol{x}} = \underset{\tilde{\boldsymbol{x}}}{\operatorname{argmin}} \|\tilde{\boldsymbol{x}}\|_{\ell_0} \quad \text{subject to} \quad \boldsymbol{\Phi}\tilde{\boldsymbol{x}} = \boldsymbol{\mu} \; . \tag{2.2}$$

Unfortunately, this problem is NP-hard as shown in [12]. This means, that exponentially many iteration steps in N and M would be needed in order to find the solution. One had to try every M-dimensional subset of columns of Φ and there are $\binom{N}{M}$ many. Since this is not feasible for larger examples, there exist different approaches, which are described below.

2.3.1. Greedy Algorithms

One possibility to tackle this optimization problem is utilizing greedy algorithms. A greedy algorithm is iterative and only based on local available information at each extension step. This means there has to be some weighting function to evaluate the quality of a (partial) solution [13].
Greedy algorithms start with an empty solution set. In each step, the element with the highest weight is added until no more changes are allowed. The canonical greedy algorithm looks as follows [13]:

- Sort elements by their weight in descending order: $e_1, \ldots, e_n$
- Set solution set $\mathcal{T} = \emptyset$
- For $k = 1$ to n do:
  If $\mathcal{T} \cup \{e_k\}$ is a valid (partial) solution, then $\mathcal{T} = \mathcal{T} \cup \{e_k\}$
- Return solution $\mathcal{T}$

It is known that the canonical greedy algorithm finds the optimal solution for a problem, if the underlying structure is a *matroid*.

Definition 2.3.3 (Matroid [13])
Given a finite set E and a set $\mathcal{U}$ of subsets of E. The algebraic structure $(E,\mathcal{U})$ is a matroid, if the following properties are fulfilled:

1. $\emptyset \in \mathcal{U}$
2. $A \subseteq B,\ B \in \mathcal{U} \Rightarrow A \in \mathcal{U}$
3. $A,\ B \in \mathcal{U},\ |A| < |B| \Rightarrow \exists x \in B \setminus A : A \cup \{x\} \in \mathcal{U}$

The first two properties create an independent system and the third is called the independent set exchange property.

Orthogonal Matching Pursuit

One famous greedy algorithm for solving (2.2) is the Orthogonal Matching Pursuit (OMP) [2]. It is used for comparison in Chapter 4 and is described in [7] as follows:

Input: sensing matrix $\boldsymbol{\Phi}$, measurement vector $\boldsymbol{\mu}$
Initialization: $S^0 = \emptyset$, $\boldsymbol{x}^0 = \mathbf{0}$
Iteration: repeat for k steps, where k is the expected sparsity

$$\begin{aligned} j_{n+1} &= \underset{j\in\{1,\dots,N\}}{\operatorname{argmax}} \{|(\boldsymbol{\Phi}^*(\boldsymbol{\mu} - \boldsymbol{\Phi}\boldsymbol{x}^n))_j|\} \\ S^{n+1} &= S^n \cup \{j_{n+1}\} \\ x^{n+1} &= \underset{\tilde{\boldsymbol{x}}\in\mathbb{R}^N}{\operatorname{argmin}} \left\{\|\boldsymbol{\mu} - \boldsymbol{\Phi}\tilde{\boldsymbol{x}}\|_2,\ \operatorname{support}(\tilde{\boldsymbol{x}}) \subset S^{n+1}\right\} \end{aligned}$$

Output: the k-sparse vector $\boldsymbol{x}$

The support($\boldsymbol{x}$) yields the positions of $\boldsymbol{x}$ with $x_i \neq 0$.
The first step finds the component of $\boldsymbol{x}$ that matches the measurements best. In the second step, this position is added to the support set S. At last, the vector $\boldsymbol{x}^{n+1}$ with one more component than $\boldsymbol{x}^n$ gets updated.

By performing exactly k iterations, the resulting vector is at most k-sparse. But since it could happen that a wrong support is chosen in an iteration, more iterations are used often. If the expected sparsity is unknown, $\|\boldsymbol{x}^{n+1} - \boldsymbol{x}^n\|_2 < \varepsilon$ can be used as termination condition.

There are examples for which the algorithm fails, but even for cases where success is not guaranteed by the RIP, the results are usually good.

2.3.2. Convex Relaxations

Another way to tackle (2.2) is provided by convex relaxations. As seen above in Chapter 2.3.2, out of the presented ℓ_p-norms, only the ℓ_2-norm gives a wrong solution for the example. And since the problem is not solvable for the ℓ_0-norm, which would be perfect for the reconstruction problem, there is the question why one should not use one of the other norms. Indeed, this is possible. Instead of solving (2.2),

$$\hat{\boldsymbol{x}} = \underset{\tilde{\boldsymbol{x}}}{\operatorname{argmin}} \|\tilde{\boldsymbol{x}}\|_{\ell_1} \quad \text{subject to} \quad \boldsymbol{\Phi}\tilde{\boldsymbol{x}} = \boldsymbol{\mu} \tag{2.3}$$

is calculated. This is called *Basis Pursuit* [14]. As mentioned before, the ℓ_0-norm minimization has combinatorical complexity. Actually, all p-norms with $0 < p < 1$ are not convex and NP-hard to solve [15]. The ℓ_1-norm is the convex norm with the smallest p and can thus efficiently be solved.

Definition 2.3.4 (Convex Set [7])
A subset $K \subset \mathbb{R}^N$ is called convex, if for all $\boldsymbol{x}$, $\boldsymbol{y} \in K$, the line segment connecting $\boldsymbol{x}$ and $\boldsymbol{y}$ is entirely contained in K, that is,

$$t\boldsymbol{x} + (1-t)\boldsymbol{y} \in K \quad \forall t \in [0,1] \, .$$

In Figure 2.2 can be seen that the ℓ_2- and ℓ_1-balls are convex, because for every two points of the set, their connecting line would also belong to the set. Whereas for the $\ell_{0.5}$- and ℓ_0-balls, the red lines do not fulfill this criteria.

(a) ℓ_2-ball $B_2(r)$: convex (b) ℓ_1-ball $B_1(r)$: convex

(c) $B_{0.5}(r)$: nonconvex (d) 'ℓ_0-ball': nonconvex

Figure 2.2.: Multiple ℓ_p-balls as examples for convex and nonconvex sets. Each red line connects two points of a set. Since this connection is not contained in $B_{0.5}(r)$ and $B_0(r)$, they are nonconvex.

In general, (2.2) and (2.3) do not give the same result. But it is shown in [1] that if the RIP (see Section 2.2.1) is fulfilled such that $\delta_k + \delta_{2k} + \delta_{3k} < 1$ holds, the results are the same. This bound for a guaranteed successful reconstruction has been tightened multiple times, e.g. in [16] and [17].

The proof that Basis Pursuit guarantees a successful reconstruction, even if the theoretical bounds provided by the RIP are pessimistic [2], is only one reason why it became popular in the early times of Compressed Sensing. The other is the possibility to translate it into *Linear Programs* for which well-studied solving techniques are available. Linear programs are explained in the next paragraph.

Linear Program

Definition 2.3.5 (Linear program in canonical form)
Given $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{R}^N$, $c_0 \in \mathbb{R}$, $\boldsymbol{b} \in \mathbb{R}^M$ *with* $\boldsymbol{b} \geq \boldsymbol{0}$. *Find the maximum for the objective function*

$$z = c_0 + \boldsymbol{c}^T \boldsymbol{x} \quad \textit{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0}\ ,$$

where the inequalities are meant componentwise.

Any linear optimization problem with linear constraints can be brought into this canonical form. Here are possible differences and how to transform them:

- **Minimization instead of Maximization:**
  Minimize $\hat{z}(\boldsymbol{x}) \quad \Leftrightarrow \quad$ Maximize $z(\boldsymbol{x}) = -\hat{z}(\boldsymbol{x})$
- **Greater-Equal Constraints:**
  $\boldsymbol{a}^T \cdot \boldsymbol{x} \geq s \quad \Leftrightarrow \quad -\boldsymbol{a}^T \cdot \boldsymbol{x} \leq -s$
- **Exact-Equality Constraints:**
  $\boldsymbol{a}^T \cdot \boldsymbol{x} = s \quad \Leftrightarrow \quad -\boldsymbol{a}^T \cdot \boldsymbol{x} \leq -s\ ,\ \boldsymbol{a}^T \cdot \boldsymbol{x} \leq s$
- **Negativity Constraints ($\boldsymbol{x'} \leq \boldsymbol{0}$):**
  $\boldsymbol{x'} \leq \boldsymbol{0} \quad \Leftrightarrow \quad \boldsymbol{x} = -\boldsymbol{x'},\ \boldsymbol{x} \geq \boldsymbol{0}$
- **Real-valued Variables ($x_i \in \mathbb{R}$):**
  $x_i \in \mathbb{R} \quad \Leftrightarrow \quad x_i = x_i' - x_i''\ ,\ x_i', x_i'' \geq 0$

There are some downsides with this procedure. Each exact-equality-constraint gets translated into two less-equal-constraints, and therefore, the number of those con-

straints gets doubled. And also the substitution for each real-valued variable increases the size of the problem by doubling the number of those variables.

The Basis Pursuit optimization problem

$$\hat{\boldsymbol{x}} = \underset{\tilde{\boldsymbol{x}}}{\operatorname{argmin}} \|\tilde{\boldsymbol{x}}\|_{\ell_1} \quad \text{subject to} \quad \boldsymbol{\Phi}\tilde{\boldsymbol{x}} = \boldsymbol{\mu}, \;\; \tilde{x}_i \in \mathbb{R}$$

can now be rewritten as a linear program in canonical form as

$$\max \boldsymbol{c}^T x \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \,,$$

with $\boldsymbol{c}^T = \begin{pmatrix} -1 & \dots & -1 \end{pmatrix}$, $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{\Phi} & -\boldsymbol{\Phi} \\ -\boldsymbol{\Phi} & \boldsymbol{\Phi} \end{pmatrix}$, $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{x''} \end{pmatrix}$ and $\boldsymbol{b} = \begin{pmatrix} \boldsymbol{\mu} \\ -\boldsymbol{\mu} \end{pmatrix}$.

This problem is convex, because its solution space is a polytope.

Definition 2.3.6 (Polytope [8])
The intersection of a finite amount of half-spaces is called a polytope. Since each half-space is convex, the polytope, as their intersection, is also convex.

Now one can apply a solver for linear programs in canonical form, such as the *Interior Point Method* or the *Simplex Algorithm.*

3. Simplex Algorithm

3.1. Introduction to Linear Optimization

The Simplex algorithm can be utilized to solve linear programs. It is used later on in this thesis and therefore described here in more detail in the style of [8]. It has become a famous method for linear optimization since it has was proposed in 1955 [3]. For further description, a linear problem in the following form is assumed:

$$\text{maximize } \boldsymbol{c}\boldsymbol{x} + c_0 \quad \text{subject to} \quad \boldsymbol{A}_{le}\boldsymbol{x} \leq \boldsymbol{b}_{le},\ \boldsymbol{A}_{ge}\boldsymbol{x} \geq \boldsymbol{b}_{ge},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}, \tag{3.1}$$

with $\boldsymbol{x} \geq 0$, $\boldsymbol{c} \in \mathbb{R}^N$, $c_0 \in \mathbb{R}$, $\boldsymbol{A}_{le} \in \mathbb{R}^{M_{le}\text{x}N}$, $\boldsymbol{A}_{ge} \in \mathbb{R}^{M_{ge}\text{x}N}$, $\boldsymbol{A}_{eq} \in \mathbb{R}^{M_{eq}\text{x}N}$, $\boldsymbol{b}_{le} \in \mathbb{R}^{M_{le}}$, $\boldsymbol{b}_{ge} \in \mathbb{R}^{M_{ge}}$, $\boldsymbol{b}_{eq} \in \mathbb{R}^{M_{eq}}$ and $M_{le} + M_{ge} + M_{eq} = M$. The inequality constraints are meant componentwise.

3.1.1. Introductory Graphical Approach

The graphical approach is only applicable for small dimensions $N \leq 3$. For each constraint, a $(N-1)$-dimensional hyperplane, that satisfies the equality, is drawn. For less-equal- or greater-equal-constraints the valid solution space below or above the hyperplane is marked. Then the hyperplane for the objective function is drawn and translated upwards until the least possible number of solutions > 0 is available. The remaining solutions are optimal.
In general, there is one possible solution left, but there are cases where the objective function and the bounding constraint have a $(N-i)$-dimensional subspace with $1 \leq i < N$ in common. Only in these pathological cases, there exist infinite many optimal solutions.

Example 3.1.1 (Graphical Optimization)

$$\text{maximize } z = 2x_1 + x_2 \ ,$$

$$\begin{aligned} \text{subject to} \quad x_1 &\geq 2, \\ x_2 &\geq 3, \\ -x_1 + 2x_2 &\geq 2, \\ x_1 + 2x_2 &\leq 12 \ . \end{aligned}$$

The result for Example 3.1.1, using the graphical approach, is presented in Figure 3.1. First, the blue constraint lines are drawn and afterwards the intersection of the respective solution spaces is shaded. At last the black constraint line is plotted through the origin and shifted upwards until it intersects with the shaded area in only one point, which is the solution. It is marked with a red dot and has the coordinates $\boldsymbol{x} = (5, 3.5)$ with the respective value $z(\boldsymbol{x}) = 2 \cdot 5 + 3.5 = 13.5$ of the objective function.

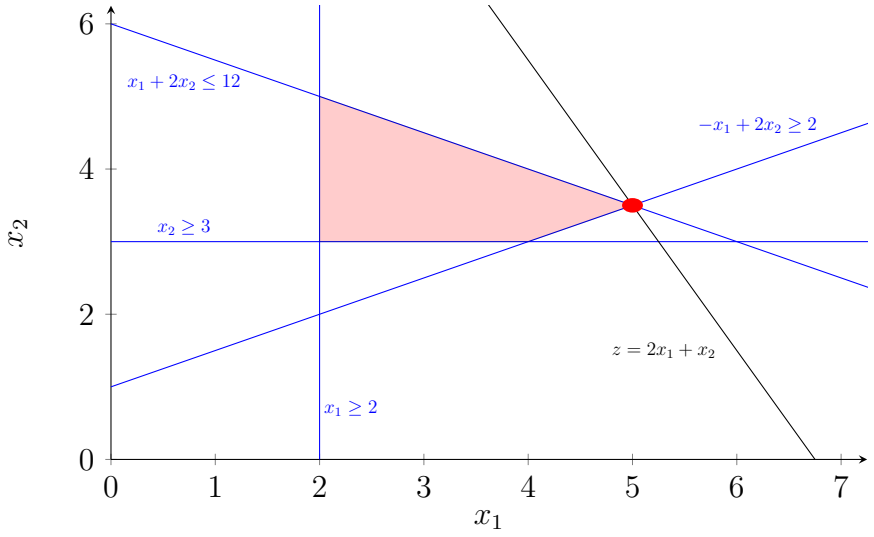


Figure 3.1.: Example 3.1.1 solved with the graphical approach. The blue lines are the constraints, the shaded area is the valid solution space, the black line represents the objective function and the red dot marks the found solution.

The division of $\mathbb{R}^N$ into two half-spaces is noticeable for each constraint, as mentioned before in Section 2.3.2. Also the convexity of the intersection can be observed for this example. Given this approach, it is obvious that the optimal solution is always contained in a corner.

Definition 3.1.1 (Corner, Degenerated Corner)
A point p of a polytope $P \subseteq R^N$ is a corner, if p is the intersection of at least N hyperplanes. If it's the intersection of more than N hyperplanes, it is called a degenerated corner.

For visualization, an example of a degenerated corner in $\mathbb{R}^3$ is shown in Figure 3.2. The two left corners are not degenerated, because they are the intersection of $N = 3$ planes in $\mathbb{R}^3$. Only the top of the pyramid is degenerated.

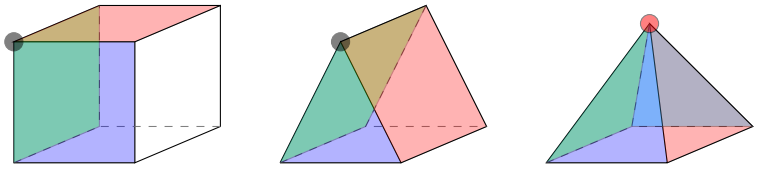


Figure 3.2.: Example for a degenerated corner (right) in $\mathbb{R}^3$.

The graphical approach is only applicable for $N \leq 3$, as mentioned before. Therefore, the preparation for a more general, arithmetical version of the solving procedure, which can be used for all $N > 0$, is given in the next section.

3.1.2. Arithmetical Approach

Transformation into a Simplex Tableau

To apply the Simplex algorithm, the problem has to be transformed into a Simplex tableau.

Definition 3.1.2 (Simplex Tableau)
The Simplex tableau for a problem as given in (3.1) *is:*

$$\left(\begin{array}{ccccc|c} \boldsymbol{A}_{le} & \boldsymbol{I}_{slack} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{b}_{le} \\ \boldsymbol{A}_{ge} & \mathbf{0} & -\boldsymbol{I}_{slack} & \boldsymbol{I}_{artif.} & \mathbf{0} & \boldsymbol{b}_{ge} \\ \boldsymbol{A}_{eq} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_{artif.} & \boldsymbol{b}_{eq} \\ \hline \boldsymbol{c} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -c_0 \\ \hline \boldsymbol{\Sigma_1} & \boldsymbol{\Sigma_2} & \boldsymbol{\Sigma_3} & \mathbf{0} & \mathbf{0} & \boldsymbol{\Sigma}_0 \end{array}\right).$$

It is an extended matrix of coefficients for (3.1)*. For each less-equal- or greater-equal-constraint, a slack variable contained in $\boldsymbol{I}_{slack}$ is added, and for each greater-equal- or equal-constraint, an artificial variable is introduced in $\boldsymbol{I}_{artificial}$ of the Sim-*

plex tableau. The $\boldsymbol{\Sigma}_i$ are the sums of the respective entries of the columns above, which contain an artificial variable. The new solution vector, composed of the old vector and the additional variables, is denoted as $\boldsymbol{x}_{ext}$. In this thesis, it is often abbreviated as $\boldsymbol{x}$, as long as they can be distinguished from the context.

In order to illustrate the role of the slack variables, Example 3.1.2 containing only less-equal-constraints is provided.

Example 3.1.2 (Slack Variables)

$$\text{maximize } z = x_1 + 2x_2 \ ,$$

$$\begin{aligned} \text{subject to} \quad x_1 + x_2 &\leq 4, \\ x_2 &\leq 2, \\ x_1, x_2 &\geq 0 \ . \end{aligned}$$

Rewriting Example 3.1.2 in matrix notation results in:

$$\text{maximize } \boldsymbol{cx} = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \ ,$$

$$\begin{aligned} \text{subject to} \quad \boldsymbol{A}_{le}\boldsymbol{x} &\leq \boldsymbol{b}_{le} \\ \Rightarrow \quad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &\leq \begin{pmatrix} 4 \\ 2 \end{pmatrix} \ . \end{aligned}$$

The other parts, $\boldsymbol{A}_{ge}$, $\boldsymbol{b}_{ge}$, $\boldsymbol{A}_{eq}$ and $\boldsymbol{b}_{eq}$ are empty, because no greater-equal- or exact-equality-constraints are given.

The corresponding Simplex tableau is

$$\left(\begin{array}{cc|c} \boldsymbol{A}_{le} & \boldsymbol{I}_{slack} & \boldsymbol{b}_{le} \\ \hline \boldsymbol{c} & \boldsymbol{0} & 0 \end{array}\right) = \left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 4 \\ 0 & 1 & 0 & 1 & 2 \\ \hline 1 & 2 & 0 & 0 & 0 \end{array}\right) \ .$$

The first two columns correspond to the basic variables x_1 and x_2, while the next two columns, consisting of unit vectors, represent the slack variables $\boldsymbol{I}_{slack}$. Due to the fact that Example 3.1.2 contains only less-equal-constraints, no articial variables are introduced, and hence, the last row, composed of the $\boldsymbol{\Sigma}_i$, can be ignored.

In Figure 3.3a, the solution space for Example 3.1.2 is shown. The blue labeled lines represent the less-equal-constraints, while the coordinate axes correspond to the non-negativity constraints. Since the optimum lies always in a corner [8] as

seen before in Example 3.1.1, the corners are investigated in more detail. From Figure 3.3a, the corners $P_0 = (0,0)$, $P_1 = (4,0)$, $P_2 = (2,2)$ and $P_3 = (0,2)$ are retrieved.
Inserting the coordinates of $P_0 = (0,0)$ into the extended constraints of Example 3.1.2 yields:

$$\begin{aligned} 0+0+x_3 &= 4 \quad \Rightarrow x_3 = 4\,, \\ 0+x_4 &= 2 \quad \Rightarrow x_4 = 2\,, \\ 0+2\cdot 0 &= z \quad \Rightarrow z = 0\,. \end{aligned}$$

Because x_3 and x_4 can be interpreted as the *gap* between the investigated point and the respective constraint (x_3 for the first and x_4 for the second), the constraints are not saturated in P_0. The value $x_3 = 4$ means there is a gap of 4 for the constraint $x_1 + x_2 \leq 4$ and $x_4 = 2$ means there is 2 left till the equality of $x_2 \leq 2$. This can be seen in Figure 3.3b. The objective function $z(\boldsymbol{x}) = z\,((0,0)) = 0$ has its minimum. The next point $P_1 = (4,0)$ leads to:

$$\begin{aligned} 4+0+x_3 &= 4 \quad \Rightarrow x_3 = 0\,, \\ 0+x_4 &= 2 \quad \Rightarrow x_4 = 2\,, \\ 4+2\cdot 0 &= z \quad \Rightarrow z = 4\,. \end{aligned}$$

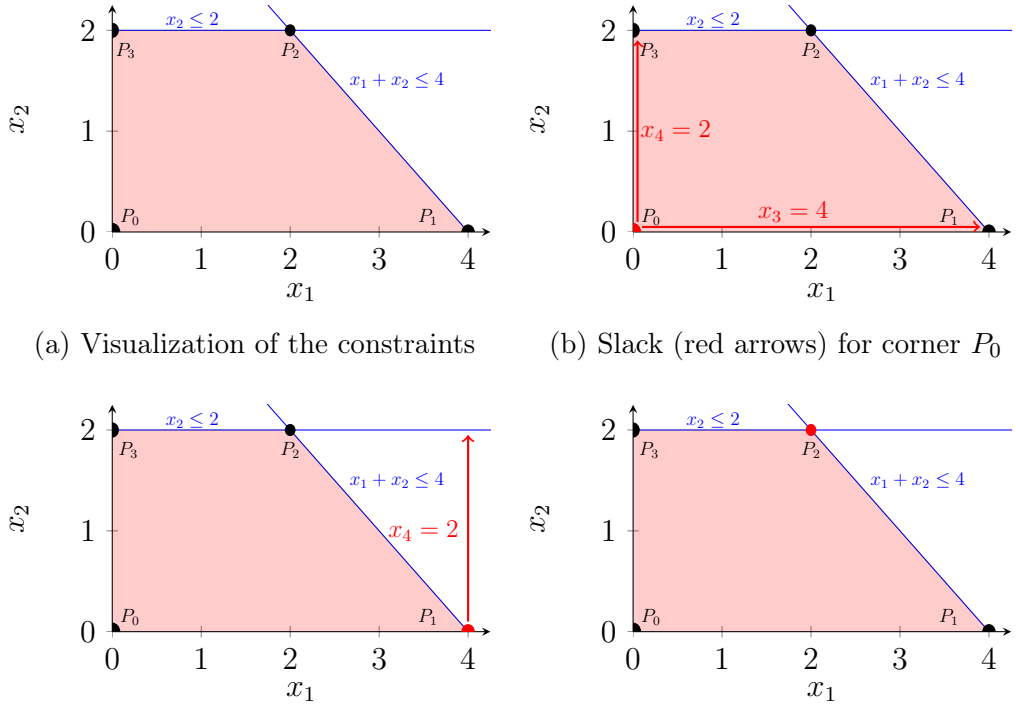
With $x_3 = 0$, the limit of the first constraint is reached, but $x_4 = 2$ means the second is not fulfilled with equality as seen in Figure 3.3c. For $P_3 = (0,2)$, it is vice versa. At both points, $z(\boldsymbol{x}) = 4$ holds, which already increased the value compared to P_0. At last, $P_2 = (2,2)$ gives:

$$\begin{aligned} 2+2+x_3 &= 4 \quad \Rightarrow x_3 = 0\,, \\ 2+x_4 &= 2 \quad \Rightarrow x_4 = 0\,, \\ 2+2\cdot 2 &= z \quad \Rightarrow z = 6\,. \end{aligned}$$

Here, both constraints are fulfilled with equality (Figure 3.3d) and the objective function reaches its maximum with $z((2,2)) = 6$.

This provides already the principal idea of the Simplex algorithm. It starts in one corner of the solution space and travels from corner to corner by selecting a neighbor with a higher value of the objective function until no better corner is available.

In the previous simple example, there are only non-negativity- and less-equal-constraints. Therefore, $\boldsymbol{x} = \mathbf{0}$ is used as starting point, since it is contained in the solution space for this example. However, this is not necessarily the case for equal- or greater-equal-constraints. For those cases a *Two-Phase-Method* is used. In the first phase, the challenge is to find a valid corner of the polytop (solution) and then the second phase is for optimization.



(a) Visualization of the constraints

(b) Slack (red arrows) for corner P_0

(c) Slack (red arrow) for corner P_1

(d) All slack variables are zero in P_2

Figure 3.3.: Visualization of the solution space (shaded red) for Example 3.1.2. The blue lines and the axes are the constraints, while the dots mark the corners.

To achieve the goal of the first phase, a secondary objective function $z'(\boldsymbol{x})$ is created whenever greater-equal- or exact-equality-constraints are given. This function is represented in the Simplex tableau by the $\boldsymbol{\Sigma}s$ from Definition 3.1.2 in the last row.

In the following, the procedure is illustrated for Example 3.1.1. First, the equations with the additional variables and the corresponding Simplex tableau are given:

$$\begin{array}{rcl} x_1 + 2x_2 + x_3 &=& 12 \\ x_1 - x_4 + x_7 &=& 2 \\ x_2 - x_5 + x_8 &=& 3 \\ -x_1 + 2x_2 - x_6 + x_9 &=& 2 \\ 2x_1 + x_2 &=& z \end{array} \quad \left(\begin{array}{ccccccccc|c} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 12 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 3 \\ -1 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 2 \\ \hline 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 3 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 7 \end{array}\right) . \tag{3.2}$$

$$\underbrace{}_{\text{original}} \quad \underbrace{}_{\text{slack}} \quad \underbrace{}_{\text{artificial}}$$

For the less-equal-constraints, the slack variables measure the space between a point and the respective constraint from 'below', as seen in Figure 3.3b for Example 3.1.2. The minus signs in front of the slack variables for the greater-equal-constraints induce that this slack is measured from 'above' the equality.
To reach the valid solution space, artificial variables are introduced into the tableau. They describe, like the slack variables, the 'gap' between the current point and the equality of the constraints. But instead of indicating how much space is left to improve the solution, their value represents the distance to the valid solution space. Therefore, in order to get a valid solution, all artificial variables have to be zero. Because they are all positive, it can be rephrased as 'the sum over all artificial variables needs to be zero'. Applying this to the example yields:

$$\begin{aligned} x_7 &= -x_1 + x_4 + 2 \,, \\ x_8 &= -x_2 + x_5 + 3 \,, \\ x_9 &= x_1 - 2x_2 + x_6 + 2 \,, \\ \Rightarrow x_7 + x_8 + x_9 &= -3x_2 + x_4 + x_5 + x_6 + 7 = 0 = z'(\boldsymbol{x}) \,, \end{aligned}$$

which corresponds to the last row of the Simplex tableau and is the secondary objective function. As soon as its maximum value $z'(\boldsymbol{x}) = 0$ is reached, a valid corner is found. Since all artificial variables are zero, they can be removed from the tableau together with the secondary objective function.

The Simplex Step

For a given Simplex tableau, the corresponding corner can be extracted with the help of columns containing unit vectors. Only those are used to determine the vector $\boldsymbol{x}$ corresponding to a point. The elements of $\boldsymbol{b}$ corresponding to the ones of the unit vectors give the non-zero elements of a corner. In tableau (3.2) it is $\boldsymbol{x} = (0,\ 0,\ 12,\ 0,\ 0,\ 0,\ 2,\ 3,\ 2)$.

For the further description, the following Simplex tableau notation is used:

$$\left(\begin{array}{ccc|c} a_{1,1} & \dots & a_{1,N+M} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{M,1} & \dots & a_{M,N+M} & b_M \\ \hline \alpha_1 & \dots & \alpha_{N+M} & \alpha \end{array}\right) . \tag{3.3}$$

Let $\boldsymbol{p} \in \mathbb{R}^{N+M}$ be the corresponding corner of the tableau, then the value of the objective function is $z(\boldsymbol{p}) = -\alpha$.

In order to perform a Simplex step, a column j with positive element α_j is chosen. This column is called *pivot column*. In tableau (3.2) for Example 3.1.1, only the second column with $\alpha_2 = 3$ is possible. To prevent negative values on the right side of the tableau, and therefore negative components of $\boldsymbol{x}$ after the step, one has to look for the *pivot row*. This is done with the minimal ratio test.

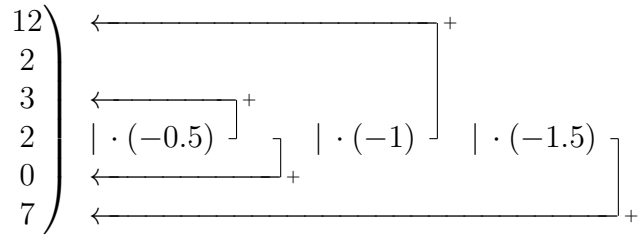
Definition 3.1.3 (Minimal Ratio Test)
For a pivot column j the associated pivot row i is chosen in the following way:

$$i = \underset{i'}{\operatorname{argmin}} \left\{ \frac{b'_i}{a_{i',j}} \geq 0 \right\} .$$

This ensures that the valid solution space is not left by a Simplex step.

Applying this to tableau (3.2) with the pivot column $j = 2$ results in the pivot row $i = \operatorname{argmin}\{\frac{12}{2}, \frac{2}{0}, \frac{3}{1}, \frac{2}{2}\} = 4$ and gives the pivot element $a_{4,2} = 2$. Creating the unit vector in column $j = 2$ by proceeding as for Gaussian elimination:

$$\left(\begin{array}{ccccccccc|c} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 12 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 3 \\ -1 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 2 \\ \hline 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 3 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 7 \end{array}\right)$$



Those operations lead to the following Simplex tableau

$$\left(\begin{array}{cccccccccc} 2 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 10 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0.5 & 0 & 0 & 0 & -1 & 0.5 & 0 & 1 & -0.5 & 2 \\ -0.5 & 1 & 0 & 0 & 0 & -0.5 & 0 & 0 & 0.5 & 1 \\ \hline 2.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & -0.5 & -1 \\ \hline 1.5 & 0 & 0 & -1 & -1 & 0.5 & 0 & 0 & -1.5 & 4 \end{array}\right),$$

with the corresponding corner $\boldsymbol{x} = (0,\ 1,\ 10,\ 0,\ 0,\ 0,\ 2,\ 2,\ 0)$. For point $(0, 1)$ in Figure 3.1, the constraint $-x_1 + 2x_2 \geq 2$ is satisfied which corresponds to artificial variable $x_9 = 0$. Since $x_8 = x_7 = 2 \neq 0$, the procedure has to be continued. Choosing $a_{2,1} = 1$ as pivot element yields:

$$\left(\begin{array}{cccccccccc} 0 & 0 & 1 & 2 & 0 & 1 & -2 & 0 & -1 & 6 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0.5 & -1 & 0.5 & -0.5 & 1 & -0.5 & 1 \\ 0 & 1 & 0 & -0.5 & 0 & -0.5 & 0.5 & 0 & 0.5 & 2 \\ \hline 0 & 0 & 0 & 2.5 & 0 & 0.5 & -2.5 & 0 & -0.5 & -6 \\ \hline 0 & 0 & 0 & 0.5 & -1 & 0.5 & -1.5 & 0 & -1.5 & 1 \end{array}\right)$$

$$\boldsymbol{x} = (2,\ 2,\ 6,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0)\ .$$

This corner is still not optimal, because the value of the secondary objective function is $z'(\boldsymbol{x}) = -1 \neq 0$. That value can be extracted from the bottom right corner of the tableau, but its sign has to be inverted, as mentioned before.

Definition 3.1.4 (Optimality Criteria)
Given a Simplex tableau as in (3.3) *for a corner* $\boldsymbol{p} \in \mathbb{R}^{N+M}$*, then the value of the objective function is* $z(\boldsymbol{p}) = -\alpha$ *and can only be improved until* $\alpha_1, \ldots, \alpha_{N+M} \leq 0$*. Then,* $-\alpha$ *is the maximum.*

Performing one more Simplex step with pivot element $a_{3,4} = 0.5$:

$$\left(\begin{array}{cccccccccc} 0 & 0 & 1 & 0 & 4 & -1 & 0 & -4 & 1 & 2 \\ 1 & 0 & 0 & 0 & -2 & 1 & 0 & 2 & -1 & 4 \\ 0 & 0 & 0 & 1 & -2 & 1 & -1 & 2 & -1 & 2 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 3 \\ \hline 0 & 0 & 0 & 0 & 5 & -2 & 0 & -5 & 2 & -11 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 \end{array}\right)$$

$$\boldsymbol{x} = (4,\ 3,\ 2,\ 2,\ 0,\ 0,\ 0,\ 0,\ 0)\ .$$

This corner is optimal, because $\forall j : \alpha_j \leq 0$ and the secondary objective function has its maximum $z'(\boldsymbol{x}) = 0$. Therefore, the first phase is finished.

For the second phase, one further Simplex step with the reduced tableau has to be calculated. The columns corresponding to the artificial variables and the last row for the secondary objective function are not needed anymore, and therefore, dropped. Then, a last step with the pivot element $a_{1,5}$ is done:

$$\left(\begin{array}{cccccc|c} 0 & 0 & 1 & 0 & 4 & -1 & 2 \\ 1 & 0 & 0 & 0 & -2 & 1 & 4 \\ 0 & 0 & 0 & 1 & -2 & 1 & 2 \\ 0 & 1 & 0 & 0 & -1 & 0 & 3 \\ \hline 0 & 0 & 0 & 0 & 5 & -2 & -11 \end{array}\right) \rightarrow \left(\begin{array}{cccccc|c} 0 & 0 & 0.25 & 0 & 1 & -0.25 & 0.5 \\ 1 & 0 & 0.5 & 0 & 0 & 0.5 & 5 \\ 0 & 0 & 0.5 & 1 & 0 & 0.5 & 3 \\ 0 & 1 & 0.25 & 0 & 0 & -0.25 & 3.5 \\ \hline 0 & 0 & -1.25 & 0 & 0 & -0.75 & -13.5 \end{array}\right).$$

This contains the corner $\boldsymbol{x} = (5,\ 3.5,\ 0,\ 3,\ 0.5,\ 0)$ with the optimum $z(\boldsymbol{x}) = 13.5$.

In order to complete the Simplex algorithm, a rule for the choice of the pivot column is needed. So far, the first positive α_j has been chosen, which is called the *rule of Bland*.

Definition 3.1.5 (Rule of Bland)
For each Simplex step, choose the first column with a positive α_j, and hence, with the lowest index j as pivot column. Using this rule ensures that the Simplex algorithm ends after a finite number of steps. This prevents cycling even in degenerated corners.

The downside of the rule of Bland is, that it usually requires far more iterations than other pivot strategies. Another strategy is the *greedy choice* which always chooses the column with the largest α_j. But unlike the rule of Bland, this strategy might result in a degenerated corner, where the algorithm could be caught in a cycle.

3.1.3. The Simplex Algorithm

Each **Simplex phase** for a Simplex tableau as seen in (3.3) consists of:

DO

1. Choose the **pivot column** j with $\alpha_j > 0$, e.g. by one of the following rules: **greedy choice** or **rule of Bland**
2. Choose the **pivot row** i for column j according to the **Minimal Ratio Test** as $i = \underset{i' \in \{1,\ldots,M\}}{\operatorname{argmin}} \left\{ \frac{b'_i}{a_{i',j}} \geq 0 \right\}$

   $\rightarrow$ if $\forall i' \in \{1, \ldots, M\} : \frac{b'_i}{a_{i',j}} < 0$, then the solution space is unbounded and there exists no optimum! Therefore, the algorithm can be stopped with an error message.
3. Transform pivot column j into an **unit vector** with 1 in row i by performing one Gaussian elimination step

UNTIL (all $\alpha_j \leq 0$)

The whole procedure for the Simplex algorithm is summarized as a flowchart in Figure 3.4.

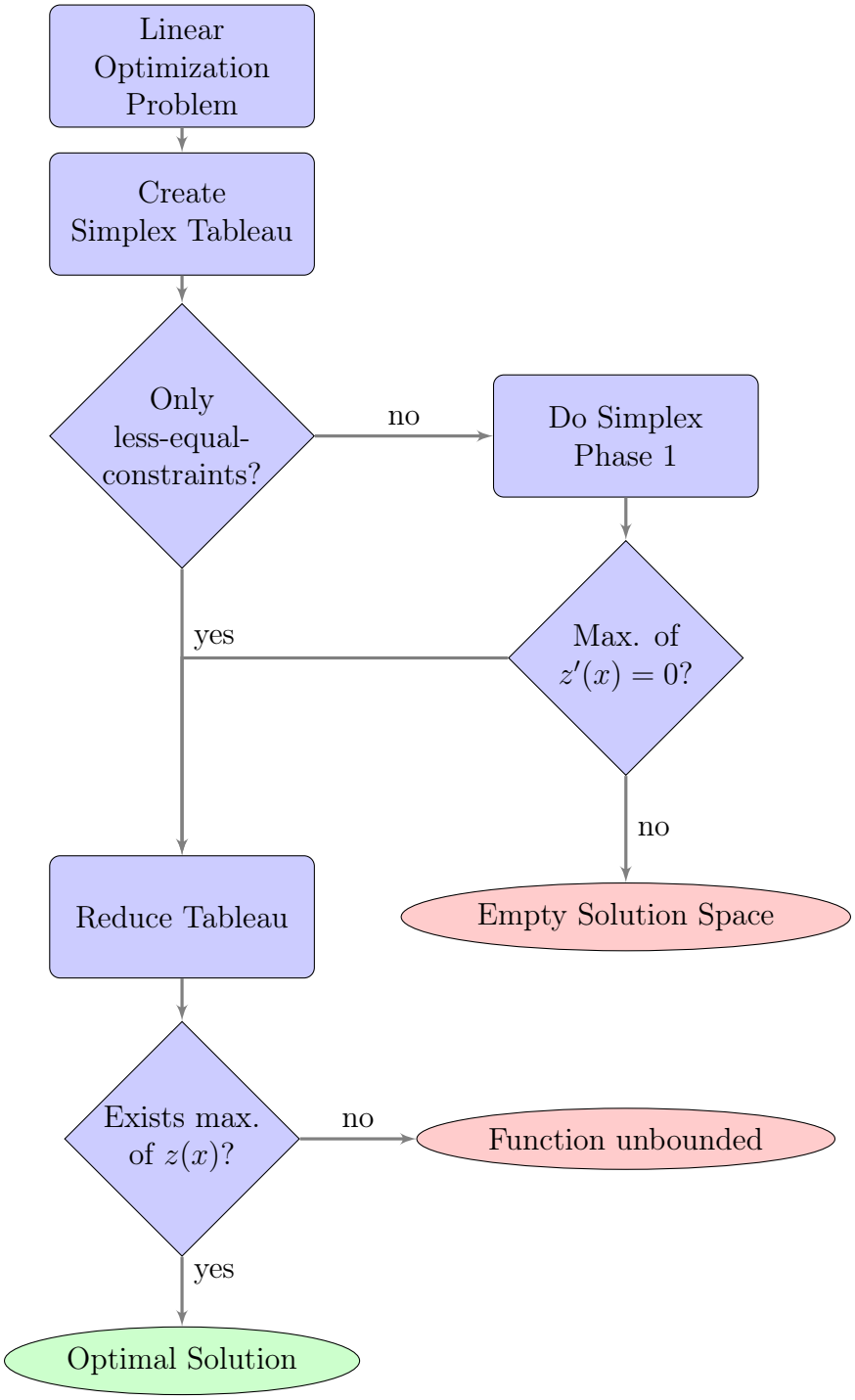


Figure 3.4.: Flowchart of the Simplex algorithm with objective functions $z'(x)$ for the first and $z(x)$ for the second phase [8].

3.2. Consequences for Basis Pursuit

As discussed before, Basis Pursuit corresponds to ℓ_1-minimization:

$$\hat{\boldsymbol{x}} = \underset{\tilde{\boldsymbol{x}}}{\operatorname{argmin}} \|\tilde{\boldsymbol{x}}\|_{\ell_1} \quad \text{subject to} \quad \boldsymbol{\Phi}\tilde{\boldsymbol{x}} = \boldsymbol{\mu} \; .$$

For the application of Basis Pursuit, a few modifications to the Simplex algorithm have to be done. Since there are only equality-constraints, the Simplex tableau looks as follows:

$$\left(\begin{array}{cc|c} \boldsymbol{\Phi}_{eq} & \boldsymbol{I}_{\text{artif.}} & \boldsymbol{\mu}_{eq} \\ \hline \boldsymbol{c} & \boldsymbol{0} & -c_0 \\ \hline \boldsymbol{\Sigma_1} & \boldsymbol{0} & \boldsymbol{\Sigma}_0 \end{array}\right) \; .$$

This means there are no slack-variables but M artificial variables, one for each constraint. Translating the ℓ_1-minimization into an objective function, because of the non-negativity-constraints $\boldsymbol{x} \geq \boldsymbol{0}$, yields:

$$\underset{\tilde{\boldsymbol{x}}}{\operatorname{argmin}} \|\tilde{\boldsymbol{x}}\|_{\ell_1} = \min \boldsymbol{c} \cdot \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \; ,$$

with $\boldsymbol{c} = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}$ and $c_0 = 0$.

Since the Basis Pursuit does not contain non-negativity constraints, a transformation has to be applied. Each variable x is substituted by two variables x' and x'' such that $x = x' - x''$ with $x', x'' \geq 0$. Therefore, $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{\Phi} & -\boldsymbol{\Phi} \end{pmatrix}$, $\boldsymbol{x}^T = \begin{pmatrix} \boldsymbol{x}' & \boldsymbol{x}'' \end{pmatrix}$ and $\boldsymbol{b} = \boldsymbol{\mu}$. In the previous examples, this step was left out for simplicity so far, since $\boldsymbol{x} \geq \boldsymbol{0}$.

Example 3.2.1 (Basis Pursuit)

$$\boldsymbol{\Phi} = \begin{pmatrix} -9 & 8 & -8 & -7 \\ -3 & 5 & -6 & -7 \end{pmatrix} \; , \qquad \boldsymbol{x} = \begin{pmatrix} 0 & 8 & 0 & 0 \end{pmatrix}^T \; ,$$

$$\boldsymbol{\Phi} \cdot \boldsymbol{x} = \boldsymbol{\mu} = \begin{pmatrix} 64 \\ 40 \end{pmatrix} \; .$$

Creating the Simplex tableau using $\boldsymbol{\Phi}$ and $\boldsymbol{\mu}$ for reconstruction:

$$\left(\begin{array}{cccccccccc|c} -9 & 8 & -8 & -7 & 9 & -8 & 8 & 7 & 1 & 0 & 64 \\ -3 & 5 & -6 & -7 & 3 & -5 & 6 & 7 & 0 & 1 & 40 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ \hline -12 & 13 & -14 & -14 & 12 & -13 & 14 & 14 & 0 & 0 & 104 \end{array}\right) \; .$$

Using rule of Bland results in column two for pivoting. Since $\frac{64}{8} = \frac{40}{5} = 8$, there are two possible choices for the pivot element. According to rule of Bland, the first one is used and results in:

$$\left(\begin{array}{cccccccccc|c} -\frac{9}{8} & 1 & -1 & -\frac{7}{8} & \frac{9}{8} & -1 & 1 & \frac{7}{8} & \frac{1}{8} & 0 & 8 \\ \frac{21}{8} & 0 & -1 & -\frac{21}{8} & -\frac{21}{8} & 0 & 1 & \frac{21}{8} & -\frac{5}{8} & 1 & 0 \\ \hline -\frac{17}{8} & 0 & -2 & -\frac{15}{8} & \frac{1}{8} & -2 & 0 & -\frac{1}{8} & \frac{1}{8} & 0 & 8 \\ \hline \frac{21}{8} & 0 & -1 & -\frac{21}{8} & -\frac{21}{8} & 0 & 1 & \frac{21}{8} & -\frac{13}{8} & 0 & 0 \end{array}\right).$$

This gives already the correct sparse solution

$$\hat{\boldsymbol{x}} = \begin{pmatrix} 0 & 8 & 0 & 0 \end{pmatrix}^T \quad \text{from} \quad \hat{\boldsymbol{x}}_{\text{ext}} = \begin{pmatrix} 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T,$$

what is indicated by the same ratio at the search for the pivot row. But since the termination condition is not satisfied yet, the algorithm continues with the first column and second row as pivot element.

$$\left(\begin{array}{cccccccccc|c} 0 & 1 & -\frac{10}{7} & -2 & 0 & -1 & \frac{10}{7} & 2 & -\frac{1}{7} & \frac{3}{7} & 8 \\ 1 & 0 & -\frac{8}{21} & -1 & -1 & 0 & \frac{8}{21} & 1 & -\frac{5}{21} & \frac{8}{21} & 0 \\ \hline 0 & 0 & -\frac{59}{21} & -4 & -2 & -2 & \frac{17}{21} & 2 & -\frac{8}{21} & \frac{17}{21} & 8 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \end{array}\right).$$

The first phase is finished and the tableau still contains the sparse solution after its reduction. Further Simplex steps are performed until $\forall j : \alpha_j \leq 0$:

$$\overset{\text{reduction}}{\Rightarrow} \left(\begin{array}{cccccccc|c} 0 & 1 & -\frac{10}{7} & -2 & 0 & -1 & \frac{10}{7} & 2 & 8 \\ 1 & 0 & -\frac{8}{21} & -1 & -1 & 0 & \frac{8}{21} & 1 & 0 \\ \hline 0 & 0 & -\frac{59}{21} & -4 & -2 & -2 & \frac{17}{21} & 2 & 8 \end{array}\right)$$

$$\overset{\text{pivot element: } a_{2,7}}{\Longrightarrow} \left(\begin{array}{cccccccc|c} -\frac{15}{4} & 1 & 0 & \frac{7}{4} & \frac{15}{4} & -1 & 0 & -\frac{7}{4} & 8 \\ \frac{21}{8} & 0 & -1 & -\frac{21}{8} & -\frac{21}{8} & 0 & 1 & \frac{21}{8} & 0 \\ \hline -\frac{17}{8} & 0 & -2 & -\frac{15}{8} & \frac{1}{8} & -2 & 0 & -\frac{1}{8} & 8 \end{array}\right)$$

$$\overset{\text{pivot element: } a_{1,5}}{\Longrightarrow} \left(\begin{array}{cccccccc|c} -1 & \frac{4}{15} & 0 & \frac{7}{15} & 1 & -\frac{4}{15} & 0 & -\frac{7}{15} & \frac{32}{15} \\ 0 & \frac{7}{10} & -1 & \frac{7}{5} & 0 & -\frac{7}{10} & 1 & \frac{7}{5} & \frac{28}{5} \\ \hline -2 & -\frac{1}{30} & -2 & -\frac{29}{15} & 0 & -\frac{59}{30} & 0 & -\frac{1}{15} & \frac{116}{15} \end{array}\right)$$

Now, the solution to the linear program is obtained with

$$\hat{\boldsymbol{x}}_{\text{ext}} = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{32}{15} & 0 & \frac{28}{5} & 0 \end{pmatrix}^T ,$$

and

$$\hat{\boldsymbol{x}} = \begin{pmatrix} x_1 - x_5 & x_2 - x_6 & x_3 - x_7 & x_4 - x_8 \end{pmatrix}^T = \begin{pmatrix} -\frac{32}{15} & 0 & -\frac{28}{5} & 0 \end{pmatrix} .$$

with $\|\hat{\boldsymbol{x}}\|_{\ell_1} = \frac{116}{15}$. This is the correct result for the ℓ_1-minimization, but not the desired sparse solution. It can be verified by calculating $\boldsymbol{\Phi} \cdot \hat{\boldsymbol{x}} = \boldsymbol{\mu}$. But as seen in the example, the algorithm already had found the sparse solution. It is the one with a zero on the right side of the tableau. That zero also led to a degenerated corner, with more zeros than needed for a usual corner in the solution space.
Recognizing that the sparse solution always has to be in a degenerated corner serves as the basic idea for the proposed sparsity aware Simplex algorithm. The question is how to find those corners and what modifications to the Simplex algorithm are needed. This is described in the next section.

3.3. Sparsity Aware Simplex Algorithm

As seen in the example, one reaches a degenerated corner after a Simplex step, if there are equal ratios at the search for the pivot row for multiple rows. However, it is not known apriori which column contains such values with the highest probability. Deciding between the rule of Bland, which is usually used to prevent reaching degenerated corners and the greedy choice, the second is more promising. It has also the advantage that it needs in average fewer Simplex steps than the rule of Bland to reach the optimum.

The Sparsity Aware Simplex algorithm (SAS) is identical to the Simplex algorithm from Section 3.1.3 till the end of the first phase:

- Create the Simplex tableau
- Perform the first phase of the Simplex algorithm to maximize $z'(x)$ until $\alpha_i \geq 0$
- Reduce the Simplex tableau

Afterwards, it is adapted and modified to different variants.

Here are four versions of the proposed sparsity aware Simplex algorithm. Each one is a trade-off between increased success rate and computational complexity. They differ on the amount of columns checked per step for equal ratios, and therefore, degenerated corners. In principle, each variant adds a certain complexity per step, but because they terminate when the sparse solution is reached, they potentially reduce the number of steps. For the description, a tableau as in (3.3) is assumed.

3.3.1. Variant 1: Pivot Column Only

The first version checks only the pivot column for identical ratios. This means the pivot column p is taken greedy, i.e. with $\forall j : \alpha_p \geq \alpha_j$. Within this, the ratios are calculated. If equal ratios are found, the first corresponding rows is chosen for the next Simplex step instead of the one satisfying the minimal ratio test. After the execution of the step, a degenerated corner is reached and the algorithm terminates. If there are not equal ratios in column p, the algorithm continues as usual with the minimal ratio test and the Gaussian elimination step. This makes sure that even if the sparse solution is not found, the solution of the default Simplex algorithm is found, and thus, the success rate is at least equal.

The only additional expenses for this version are the checks for equal ratios, since they are even calculated for the usual Simplex algorithm. This can be done by sorting those M values, subtract the vector with a shifted version and then looking for a zero. This is insignificant compared to the other operations done in the algorithm.

3.3.2. Variant 2: Positive Columns

The next version investigates all columns j with $\alpha_j \geq 0$. If it does not find equal ratios in any column, it continues with the one of largest α_j and performs the minimal ratio test. This approach seems useful because all columns with $\alpha_j > 0$ lead to a better solution in respect to $\|\cdot\|_{\ell_1}$ and one expects the optimal $\|\cdot\|_{\ell_0}$-solution to be close to it.

The increase in complexity can be estimated as calculating M ratios, sorting and vector subtraction for each column with $\alpha_j \geq 0$. This number is approximately half the number of columns as rough guess. The possible gain for the success rate is explained by the additional corners which are considered.

3.3.3. Variant 3: All Columns

The third variant checks all columns at each Simplex step. Since it considers the most possible corners on its way to the ℓ_1-optimum, it should have the highest success rate, but also the highest complexity per step. It has to check every N (or even $2N$ for $x_i \in \mathbb{R}$) columns for equal ratios.

3.3.4. Variant 4: ℓ_1-Neighbors

Assuming that the ℓ_0-solution is close to the ℓ_1-optimal corner. Starting from the ℓ_1-solution obtained with the default Simplex algorithm (or also one of the previous variants), all neighbors of this found corner can be checked for equal ratios. This might improve the success rate with relatively small computational effort. It is investigated in Section 4.1.

Example for Basis Pursuit with Sparsity Awareness

Applying each of the proposed variants on the trivial Example 3.2.1 leads to the sparse solution, since the first phase is exactly the same as seen above. Each variant, with exception of Variant 4, stops after recognizing the degenerated corner. But even this finds the sparse solution by searching around the ℓ_1-optimum. For this trivial example the actual sparse reconstruction problem is now successfully solved and since the number of required Simplex steps is reduced, the calculation time is also improved.

4. Evaluation and Comparison

4.1. Exploration of the Solution Space

4.1.1. Number of Corners

The first analysis considers the number of corners in the solution space and how many of them merge into a degenerated corner depending on the dimension of the space and the sparsity [18]. This is required to estimate the difficulty of finding the sparse solution.
Given two subspaces $\mathcal{A}_1, \mathcal{A}_2 \subset \mathbb{R}^N$ with $\dim(\mathcal{A}_1) = p$ and $\dim(\mathcal{A}_2) = q$ in general position, i.e. the coefficients of their matrix representation are chosen randomly. Then, $\mathcal{A}_1$ and $\mathcal{A}_2$ intersect in a subspace of dimension

$$\dim(\mathcal{A}_1 \cap \mathcal{A}_2) = \dim(\mathcal{A}_1) + \dim(\mathcal{A}_2) - N$$

with probability 1 [19, p. 9]. This can be explained by the observation that there are just finite many cases for a different dimension of the intersection, but infinite many cases for this dimension. Two low-dimensional examples of common problems are povided:

Example 1: Intersection of two planes in $\mathbb{R}^3$:
$\dim(\mathcal{A}_1) = \dim(\mathcal{A}_2) = 2 \;\Rightarrow\; \dim(\mathcal{A}_1 \cap \mathcal{A}_2) = 2+2-3 = 1$, i.e. the typical solution space for this constellation is a line.

Example 2: Intersection of two lines in $\mathbb{R}^3$:
$\dim(\mathcal{A}_1) = \dim(\mathcal{A}_2) = 1 \;\Rightarrow\; \dim(\mathcal{A}_1 \cap \mathcal{A}_2) = 1+1-3 = -1$, i.e. they are typically skew and have no solution in common.

There are M random hyperplanes described by $\mathbf{\Phi}$ from (2.1), each with dimension $N-1$, and therefore, the solution space $\mathcal{S}(N, M)$ is a $(N-M)$-dimensional subspace of $\mathbb{R}^N$. Considering a k-sparse solution, which is known to be contained in a k-dimensional subspace $\mathcal{V}$, results in an intersection of dimension

$$\dim(\mathcal{S} \cap \mathcal{V}) = N - M + k - N = k - M \; .$$

From this, it can be derived that there are no solutions with sparsity $k < M$ for M hyperplanes in general position. It can also be seen, that a typical solution is M-sparse, and therefore, there are $\binom{N}{M}$ possibilities to choose those entries. By this, the uniqueness of the sufficiently k-sparse solution (for $k < M$), is shown. Due to the sparsity, it is always in a degenerated corner.

Comparing the hyperplanes in general position to system (2.1), after multiplication with a sparse vector $\boldsymbol{x}$, shows that some of the corners merge into a degenerated corner. In consequence, the next step is to calculate the number of those corners. This is analyzed combinatorically in [18], which states that these are $\binom{N-k}{M-k}$ corners. The line of thought is the following: Because $\boldsymbol{x}$ is a fixed k-sparse vector, contained in a thereby defined k-dimensional subspace, k values are appointed. Knowing the corners are usually M-sparse leaves $M-k$ positions open, and thus, there are $\binom{N-k}{M-k}$ many possibilities.

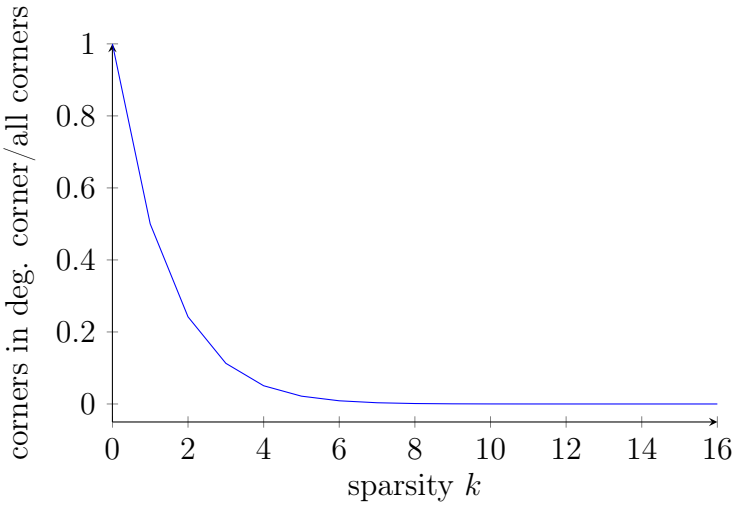


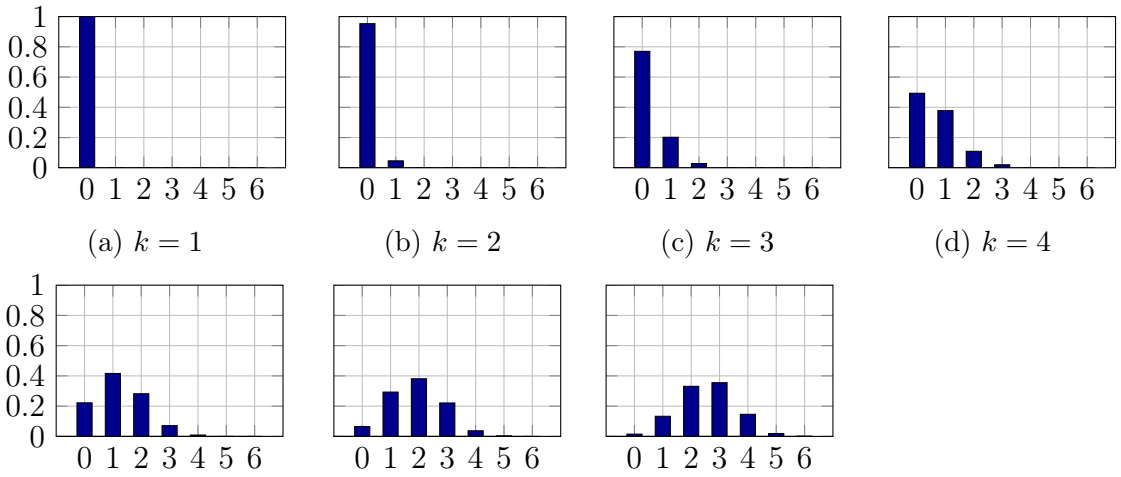
Figure 4.1.: Ratio of corners in the degenerated corner $\binom{N-k}{M-k}$ over all corners $\binom{N}{M}$ for $N = 32$, $M = 16$.

As seen in Figure 4.1, a lot of corners merge into the degenerated corner for small dimensions. For those cases, it is easy to find the sparse solution. Regarding a randomly selected corner, there is a high probability that it is part of the degenerated corner. On the other hand, for large k, there are a lot of corners, but only very few are part of the sparse solution. In these cases, it is like looking for the needle in a haystack. Thus, random search strategies are not efficient for larger values of k.

4.1.2. Distance from ℓ_1- to ℓ_0-Solutions

Since the SAS relies on the 'closeness' of the ℓ_1- to the ℓ_0-solution, the question of their distance distribution arises. This is investigated statistically in this section. The main problem of the exploration is the huge number of corners that should be regarded. Actually, the problem is to reconstruct the sparse vector directly, e.g. by 'ℓ_0-norm minimization' and this is practically not possible, because it is NP-hard as already mentioned. Therefore, the approach is the following: First, the ℓ_1-solution is calculated with the default Simplex algorithm and checked if it is already the sparse solution. If not, all neighbors of this corner are regarded. Next, each of their neighbors is checked again, and so on. Since there are approximately N^d neighbors at distance d, very small dimensions are chosen for the statistical analysis.

For the first simulation, Gaussian sensing matrices of size $N = 16$, $M = 8$ are selected. The results can be seen in Figure 4.2. The percentage of solutions with a distance $d = 0$ corresponds directly to the success rate of Basis Pursuit (default Simplex algorithm). By checking all neighbors of the ℓ_1-solution, as performed by some variants of the SAS, the success rate can be increased by the corners with distance $d = 1$. The complexity for this is small, and thus, it is profitable for small k, relatively to N and M. If k is large, the success rate and also the additional gain decreases. Here, applying this approach leads to a success rate $> 80\%$ up to $k = 4$, while the Basis Pursuit falls even for $k = 3$ below that rate.

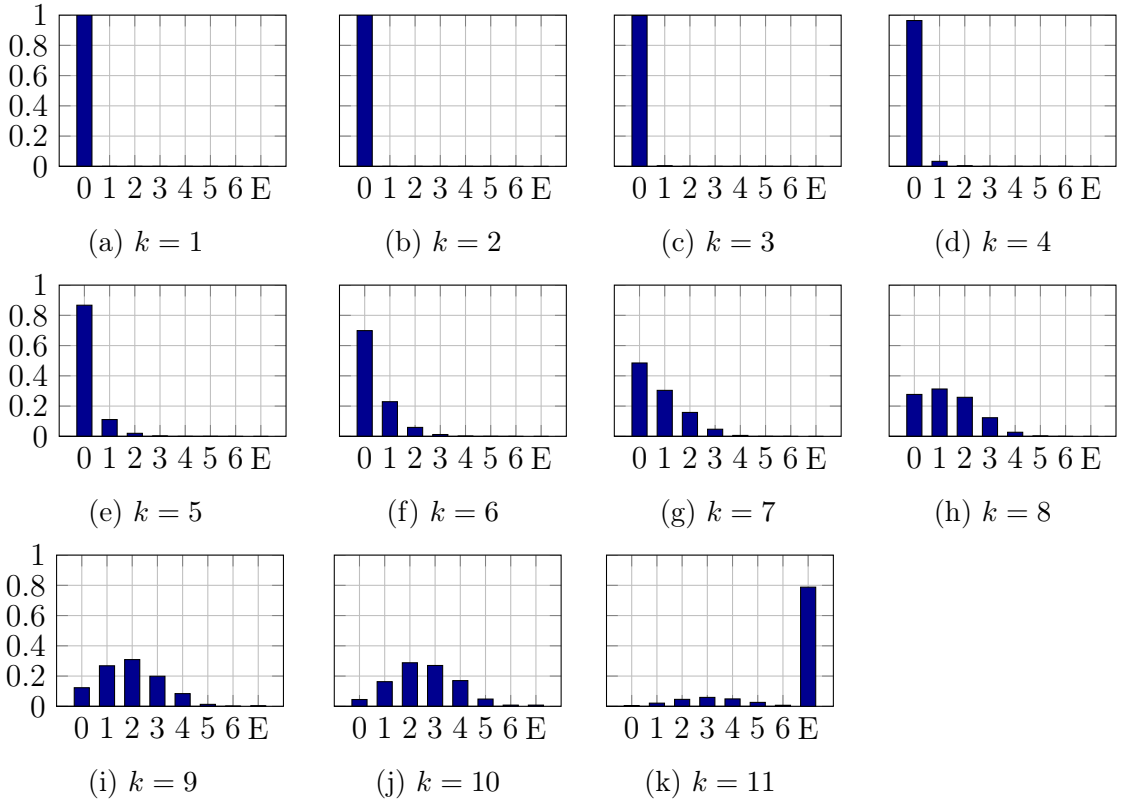


(a) $k = 1$ (b) $k = 2$ (c) $k = 3$ (d) $k = 4$

(e) $k = 5$ (f) $k = 6$ (g) $k = 7$

Figure 4.2.: Histograms for the distance from the ℓ_1-solution to the sparse solution in Simplex steps. The dimensions are $N = 16$, $M = 8$. The plot is created with 1000 runs with different Gaussian sensing matrices and vectors for each sparsity k.

For comparison with larger dimensions N and M with the same ratio $\frac{N}{M}$, a look onto the histograms for $N = 32$ and $M = 16$ is given in Figure 4.3. In Figures 4.2 and 4.3, one can see that the ℓ_1- and the ℓ_0-solution are very close to each other for a small value of the sparsity k. The distance is measured in Simplex steps and increases with larger k. The smaller the distance d, the easier the sparse solution can be found. But for larger k it gets considerably harder, since the number of possible solutions grows exponentially in d. For Figure 4.3 and $k \geq 9$, there are cases where the algorithm does not find the sparse solution and exits premature due to limited heap space or runtime limitation. Those cases are marked with E.



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$ (d) $k = 4$

(e) $k = 5$ (f) $k = 6$ (g) $k = 7$ (h) $k = 8$

(i) $k = 9$ (j) $k = 10$ (k) $k = 11$

Figure 4.3.: Histograms for the distance from the $\|\cdot\|_{l_1}$-solution to the sparse solution in Simplex steps. The dimensions are $N = 32$, $M = 16$. The plot is created with 1000 runs with different Gaussian sensing matrices and vectors for each sparsity k. E is the result if the algorithm does not find the sparse solution due to heap space or runtime limitations.

4.2. Comparison of the Reconstruction Algorithms

The algorithms from Section 3.3 are statistically analyzed. The used combinations are summarized in Table 4.1.

Label	Algorithm / Combinations
Ⓐ	Default Simplex
Ⓑ	Default Simplex with SAS Variant 4
Ⓒ	SAS Variant 1
Ⓓ	SAS Variant 1 with SAS Variant 4
Ⓔ	SAS Variant 2 (contains Variant 1)
Ⓕ	SAS Variant 2 with SAS Variant 4
Ⓖ	SAS Variant 3 (contains Variant 2)
Ⓗ	SAS Variant 3 with SAS Variant 4

Table 4.1.: Labeling for the different combinations of modifications to the algorithm.

4.2.1. Success Rate Comparison

The first simulation investigates the behavior of the algorithms from Table 4.1 for Gaussian sensing matrices with the dimensions $N = 32$ and $M = 16$. The results can be seen in Figure 4.4. It is created by averaging over 1000 runs for each sparsity $k \in \{1, \dots, M\}$ with different matrices, sparse vectors and permutations for the positions of the non-zero values. A reconstruction is declared successful, if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise. This ε is necessary due to numerical imprecision and the componentwise check has the advantage over the often used $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_{\ell_2}$, that it varies less for different dimensions.
Investigating the variants without checking the neighbors of the ℓ_1-solution, represented as solid lines, reveals an expected behavior. Ⓐ (default Simplex) has the worst success rate and Ⓒ (SAS pivot) improved the rate without noteworthy increase in complexity. Ⓔ (SAS positive) has an even higher gain, but not as large as Ⓖ (SAS all).
A remarkable improvement just by checking the neighbors of the ℓ_1-solution is observed regarding the dashed lines. Ⓑ is already better than Ⓔ while having a lower complexity. All variants benefit from considering the neighbors at the end, but the more complex the modifications, the less the potential benefit is. This can be explained with the higher number of neighbors already checked on the path to the

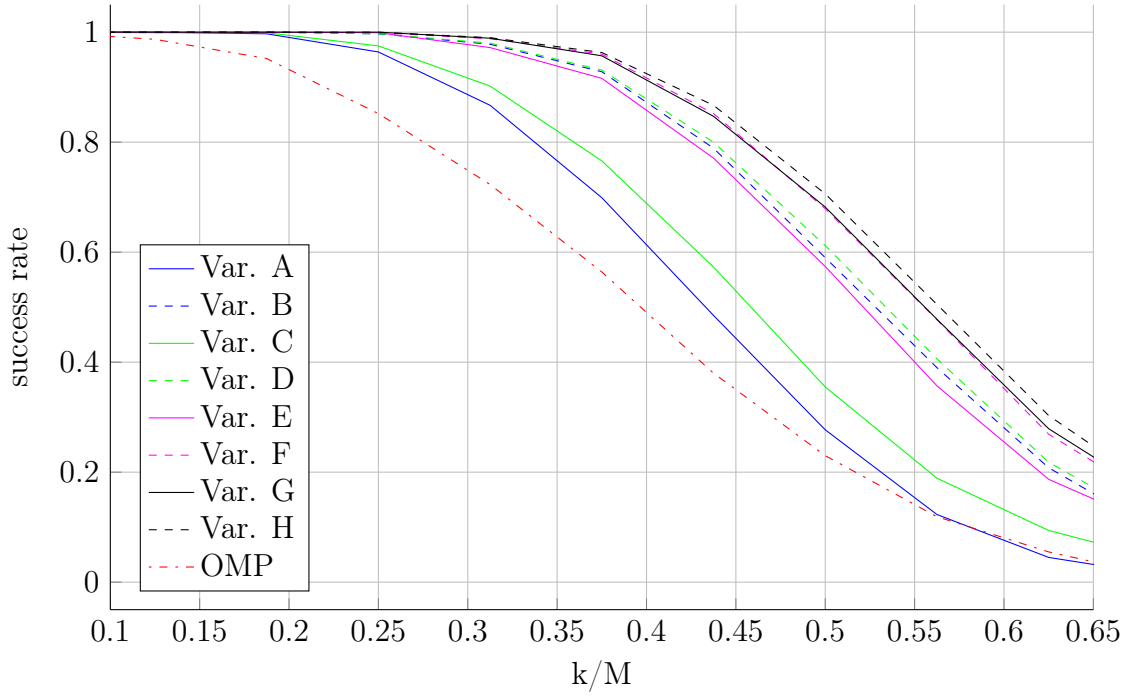


Figure 4.4.: Comparison of the success rate for the different algorithms and variants from Table 4.1 using Gaussian matrices with dimensions $N = 32$, $M = 16$. The rate is averaged over 1000 runs for each k with different matrices and Gaussian values for the sparse vector $\boldsymbol{x}$ with a success if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.

ℓ_1-solution. Ⓗ has the best success rate, but also the highest worst case complexity. Depending on the application, one might prefer Ⓓ because of the improved success rate without a significant increase in computing time.
The bad behavior of the OMP algorithm is explainable. One reason are the chosen dimensions. As seen later on, they have a noticeable influence on the performance. The other reason is that the used OMP is not optimized for those simulations, i.e. it uses an $\varepsilon \leq 10^{-7}$ termination condition and does not exploit the sparsity.

The next simulation, with results in Figure 4.5, is done for the same variants. The difference is the change of the $\frac{N}{M}$-ratio from 2 to 5, and therefore, $N = 80$, $M = 16$. Considering a higher ratio, which corresponds to a higher compression rate, one expects the success rates to drop. This can be confirmed by comparing Figure 4.4 and Figure 4.5. The performance of all algorithms decreases, but differently. The OMP is in this case for success rates $< 85\%$ better than Ⓐ and Ⓒ. The ranking of the variants of the Simplex algorithm among themselves stays the same.
Comparing the two figures, another observation is the faster 'phase-transition' from a 100% success rate to 0%. The edge gets sharper with increasing N.

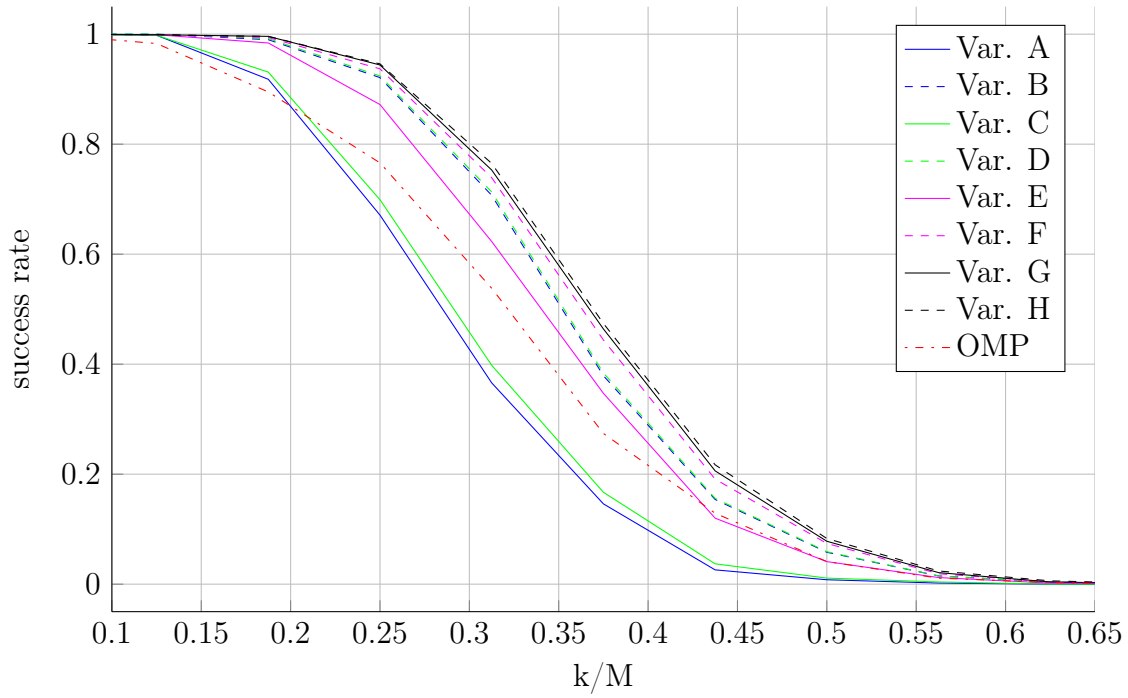
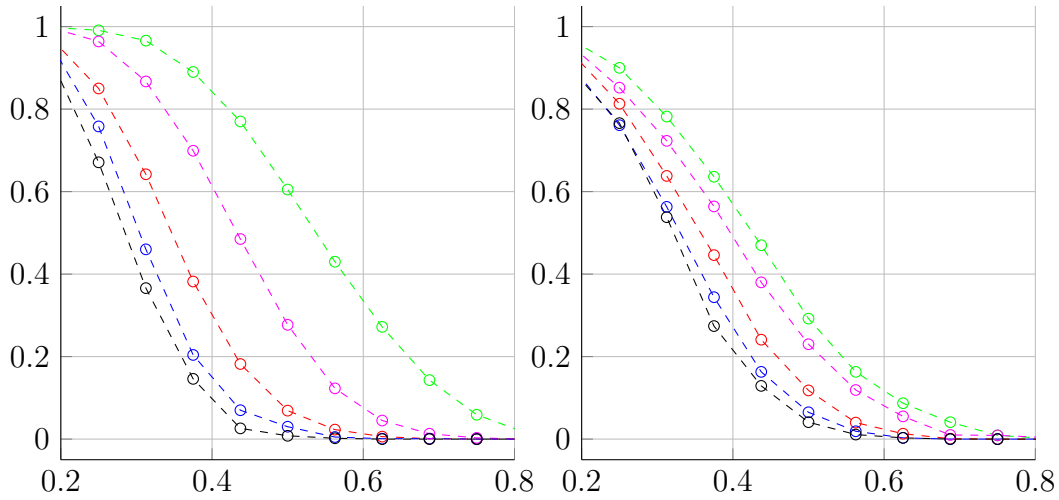


Figure 4.5.: Comparison of the success rate for the different algorithms and variants from Table 4.1 using Gaussian matrices with dimensions $N = 80$, $M = 16$. The rate is averaged over 1000 runs for each k with different matrices and Gaussian values for the sparse vector $\boldsymbol{x}$ with a success if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.

The curves for $M = 16$ with $\frac{N}{M} \in \{3, 4\}$, and respectively $N \in \{48, 64\}$, as intermediate steps can be found in the Appendix in Figures A.1a and A.1b.

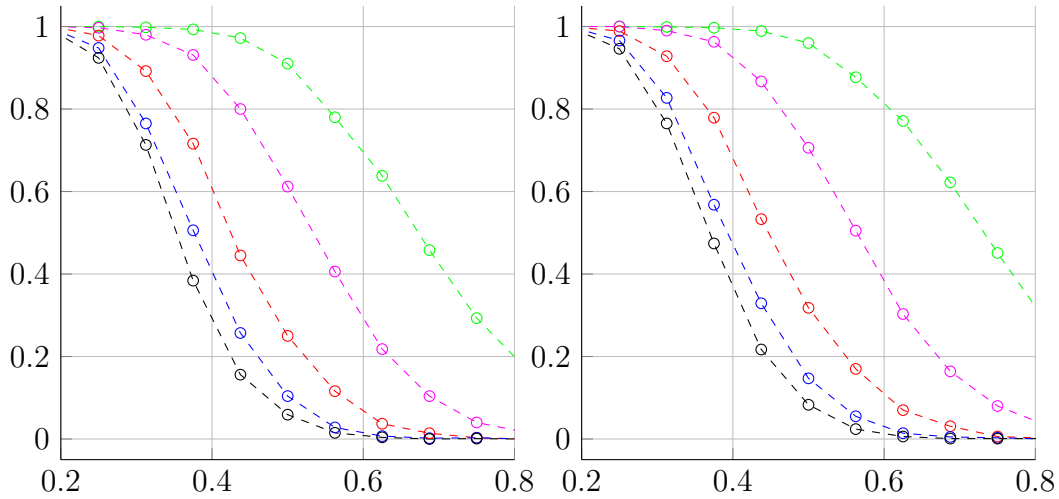
The behavior for varying $\frac{N}{M}$ is shown in Figure 4.6. By comparing the default Simplex algorithm and the OMP in Figures 4.6a and 4.6b, it can be recognized, that the rate of both algorithms decreases for larger $\frac{N}{M}$, but less for the OMP. Examining Figures 4.6c and 4.6d exhibits similar rates, especially for larger $\frac{N}{M}$. This indicates, that most of the improvement (over Ⓐ in Figure 4.6a) is due to the 'check the neighbors of the ℓ_1-solution'-strategy, that both variants have in common and not due to the search of the solution as neighbors of the path.

For further comparison, another type of sensing matrices is considered in the next section.



(a) Variant A

(b) OMP



(c) Variant D

(d) Variant H

Figure 4.6.: Success rate over k/M for $M = 16$ with $N/M = 1.5$ (green), $N/M = 2$ (magenta), $N/M = 3$ (red), $N/M = 4$ (blue) and $N/M = 5$ (**black**) averaged for 1000 runs with Gaussian matrices.

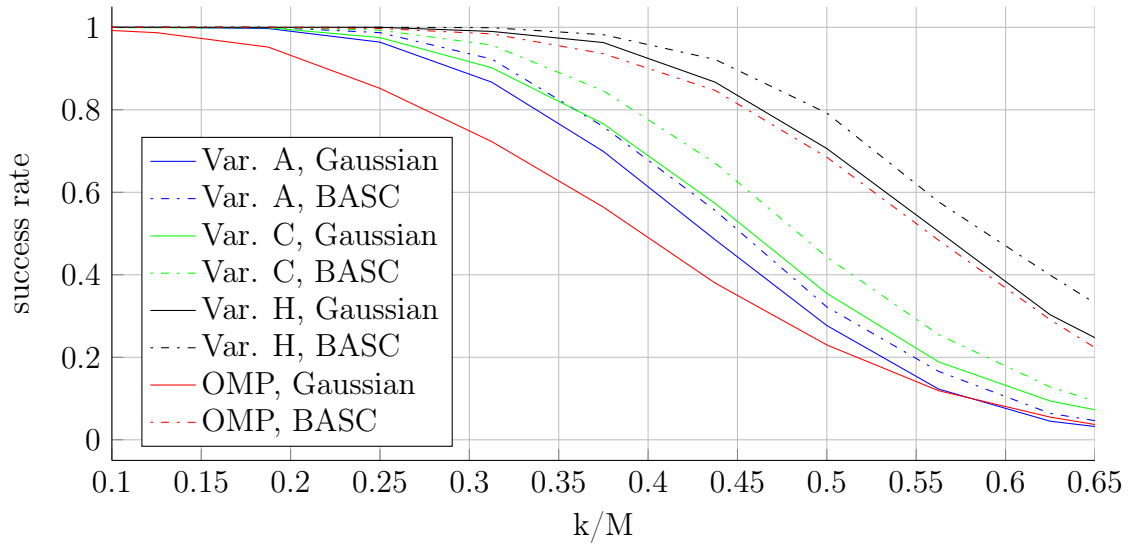


Figure 4.7.: Influence of BASC matrices on several algorithms from Table 4.1. For each $k \in \{1, \ldots, M\}$ with $M = 16$ and $N = 32$, the rate is averaged over 1000 runs. The reconstruction is successful if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| < 10^{-14}$ componentwise.

4.2.2. Influence of Coherence Optimized Sensing Matrices

In this section, the influence of coherence optimized sensing matrices on the algorithms is investigated. The idea is, that a lower coherence (as defined in Section 2.2.1) guarantees a successful reconstruction even for larger values of k for fixed N, M. The reconstruction rates in the provided simulations are way above those guaranteed. Whether the algorithms benefit nevertheless is analyzed now.
BASC matrices [11] have low coherence and are used for this purpose. As seen in Figure 4.7, the success rate for BASC matrices is better than for Gaussian sensing matrices. The gain for all Simplex based variants is approximately the same, while the OMP takes a leap. The other variants behave the same and can be found in the Appendix in Sections A.2 and A.3. The dimensions for the simulations are the same as for Figure 4.4: $N = 32$, $M = 16$. As observed in the figures, the algorithms do improve by utilizing coherence optimized sensing matrices. The Simplex based algorithms gain only a little bit, but the OMP improves more, because its choice of the element, which is added in each iteration, depends directly on the coherence. The lower the coherence, the less similar the columns of the sensing matrix, and hence, the better the reconstruction rate. This behavior is described in more detail in [20].

4.3. Evaluation for Constant N/M-Ratio

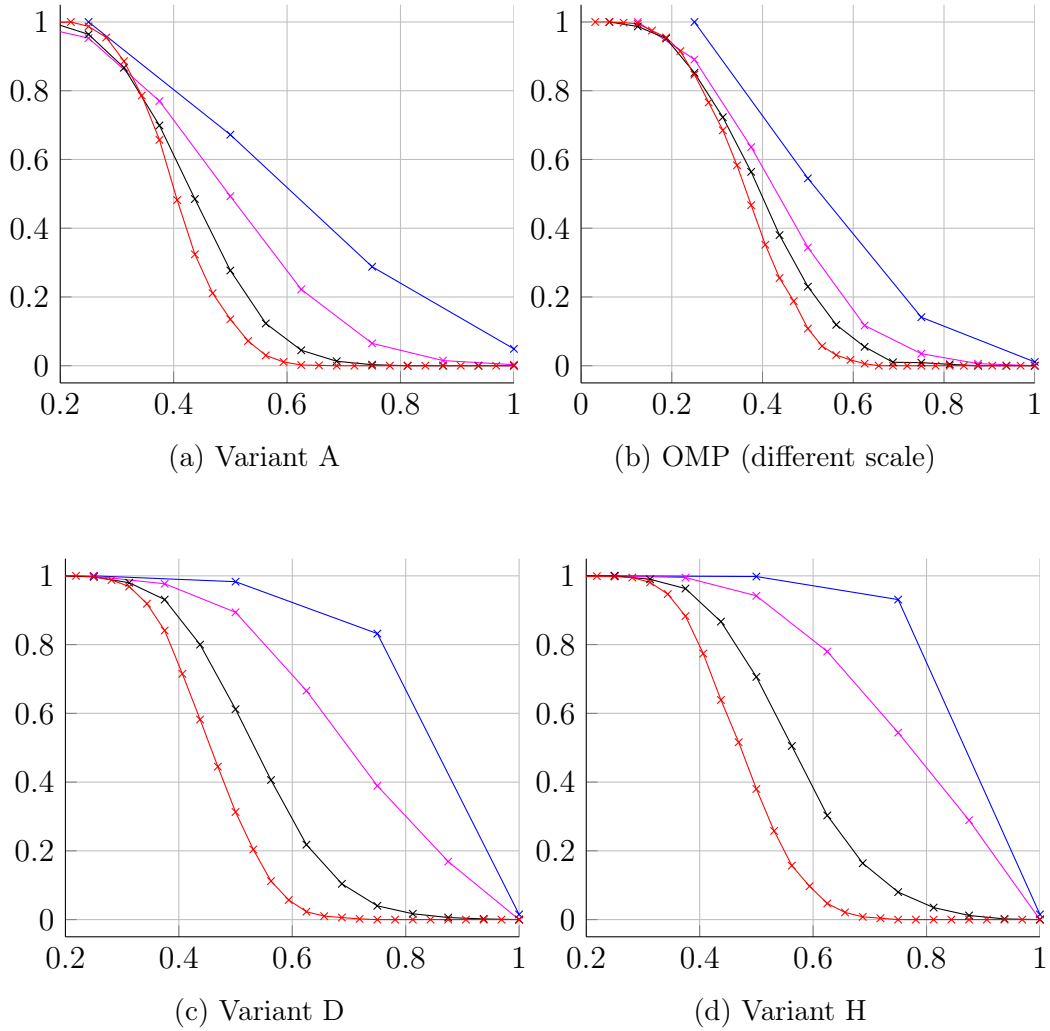
Assuming a fixed $\frac{N}{M}$-ratio, the next investigation is performed regarding the change of the success rate over dimensions. The results of the simulations can be seen in Figure 4.8.

For Ⓐ, the default Simplex algorithm, the rate increases for growing M and small $\frac{k}{M}$ (≤ 0.3), while for larger $\frac{k}{M}$ the rate decreases as seen in Figure 4.8a. Again, the 'phase-transition' becomes sharper, as seen before in Figure 4.5. With increasing dimensions, the rate becomes close to a step function.

The high reconstruction rate of all algorithms in Figure 4.8 for small values of M and relatively large $\frac{k}{M}$ ($0.5 \leq \frac{k}{M} \leq 0.8$) is due to the small number of corners caused by the small M. Using the formulas from Section 4.1.1, there are only $\binom{N}{M} = \binom{8}{4} = 70$ corners, whereof $\binom{N-k}{M-k} = \binom{5}{1} = 5$ are degenerated for $\frac{k}{M} = 0.75$. Even randomly choosing a corner would have a success rate of about 7%.
For small dimensions, it is also likely that the ℓ_0-solution is close to the ℓ_1-solution as described before in Section 4.1.2 and shown in Figure 4.2. Since Ⓓ and Ⓗ regard the neighbors of the ℓ_1-optimal corner, and therefore, retrieve not just solutions with distance $d = 0$, but also $d = 1$ for sure, they have an extraordinary high success rate for those cases.

In addition, Ⓓ and Ⓗ perform almost alike (Figures 4.8c & 4.8d) and show for small M more gain due to their modification compared to Ⓐ. This can be explained by the fast growing number of corners for larger dimensions and the related smaller probability to find the degenerated corner. The considerably higher complexity of Ⓗ, because of the investigation of the neighbors on the path to the ℓ_1-solution, does only lead to marginally better reconstruction rate than for Ⓓ. I.e., depending on the application, one would prefer Ⓓ over Ⓗ. Again, as mentioned before in Section 4.2.1, this shows that the search for the sparse solution on the path makes up only a small part of the improvement compared to the gain by looking at the neighbors of the ℓ_1-solution. This, together with the histograms in Figure 4.3 and the behavior in Figures 4.8c and 4.8d suggests, that for larger dimensions N and M, only very few, or even no gain at all, can be expected.

The OMP, shown in Figure 4.8b, behaves similar to Ⓐ, but has a worse success rate for this setting.



(a) Variant A (b) OMP (different scale)

(c) Variant D (d) Variant H

Figure 4.8.: Success rate over k/M for $N/M = 2$ with $M = 4$ (blue), $M = 8$ (magenta), $M = 16$ (**black**), $M = 32$ (red) averaged for 1000 runs with Gaussian matrices.

4.4. Runtime Analysis

At last, a runtime comparison for the different variants of the SAS is performed. The runtime is measured in Simplex steps applied until their successful or unsuccessful termination.

Boxplots, or sometimes also called *box-and-whisker diagrams*, are utilized in this section for comparison. Therefore, a short recapitulation is provided. Assuming a data set Y consisting of N values sorted in ascending order $y_1 \leq y_2 \leq \cdots \leq y_N$, then the parts of the corresponding boxplot represent:

- The filled circle is the **median** of Y, i.e. the element $y_{\frac{N+1}{2}}$. If N is even, the median is calculated as the mean of the two middle values. It is denoted as the second quartile q_2.
- The filled box contains all data points from **25th- to the 75th-percentile**, i.e. the $\frac{N}{2}$ data points in the middle of the set. The 25th-percentile is the first quartile q_1 and the 75th is the third q_3.
- The dash contained in the box represents the **mean** of Y, but is sometimes hidden by the circle for the median.
- The dotted lines outside the box, limited by the dashes, extend the box by 1.5 times the interquartile range (IQR) to each direction. The IQR is the difference of the 75th- and the 25th-percentile: $IQR = q_3 - q_1$. The two dashes enclose all values y_i with $\boldsymbol{q_1 - 1.5 \cdot IQR \leq y_i \leq q_3 + 1.5 \cdot IQR}$.
- The gray dots represent the **outliers** not contained in the other ranges.

The advantage of boxplots over diagrams showing only the mean of a set is the additional information, e.g. about the distribution or, at least, the spread of the data points.

4.4.1. Steps in the First Simplex Phase

In the first part, an analysis of the behavior in the first Simplex phase is performed. The number of steps is the same for all variants, because they utilize all the same strategy. Hence, the analysis is performed over different $\frac{N}{M}$-ratios and growing M for $\frac{N}{M} = 2$.

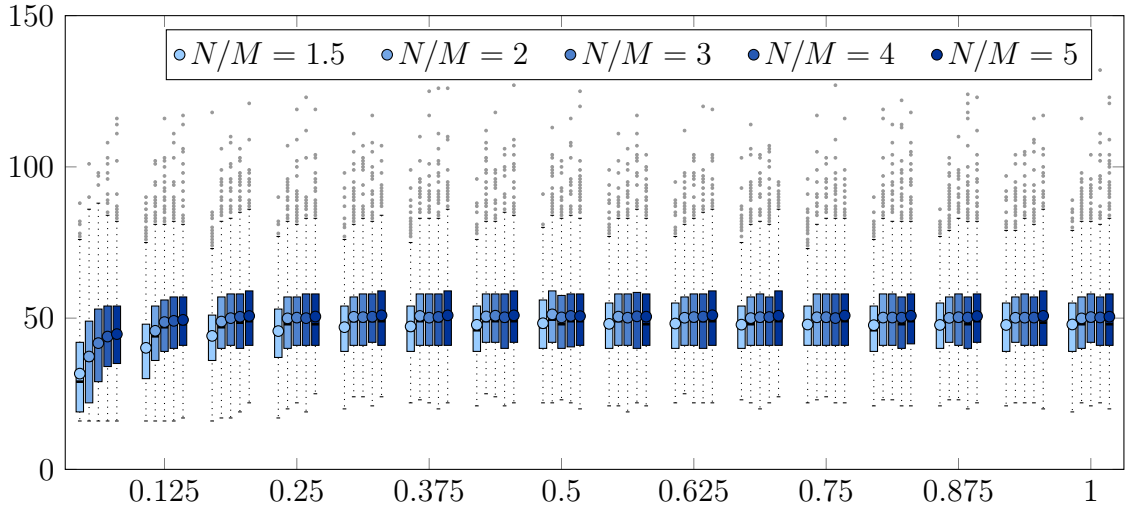
Figure 4.9a shows the Simplex steps needed to finish the first phase, i.e. to find a valid corner of the solution space. The number of steps varies over $\frac{N}{M}$ for very

small $\frac{k}{M}$, whereas for larger $\frac{k}{M}$ the median and the distribution, represented by the circle and the position of the box, stays almost the same. So for constant M, the number of required steps is approximately the same. This indicates a direct correlation of iterations and the number of artificial variables M. It might be related to the goal of the first phase to eliminate them. According to this, the number of steps should increase for larger M, which is confirmed by Figure 4.9b.
That there are other influences as well, can be retrieved from Figure 4.10. It shows the required iterations for the first phase utilizing BASC matrices. One observes small changes for varying $\frac{N}{M}$, but relatively constant distributions over $\frac{k}{M}$. This behavior might be due to the quality of the matrices and the setting of the simulation. In Figure 4.9a, 1000 different Gaussian sensing matrices are used, while for Figure 4.10 it is one for each $\frac{N}{M}$. This one is chosen as the best, i.e. with the lowest coherence, gathered from 10 runs of a matrix optimization procedure. It would be out of the scope of this thesis to investigate the matrix properties causing this, but it may be attractive for future research. But as said, it shows, that there are influences on the number of required steps besides M.

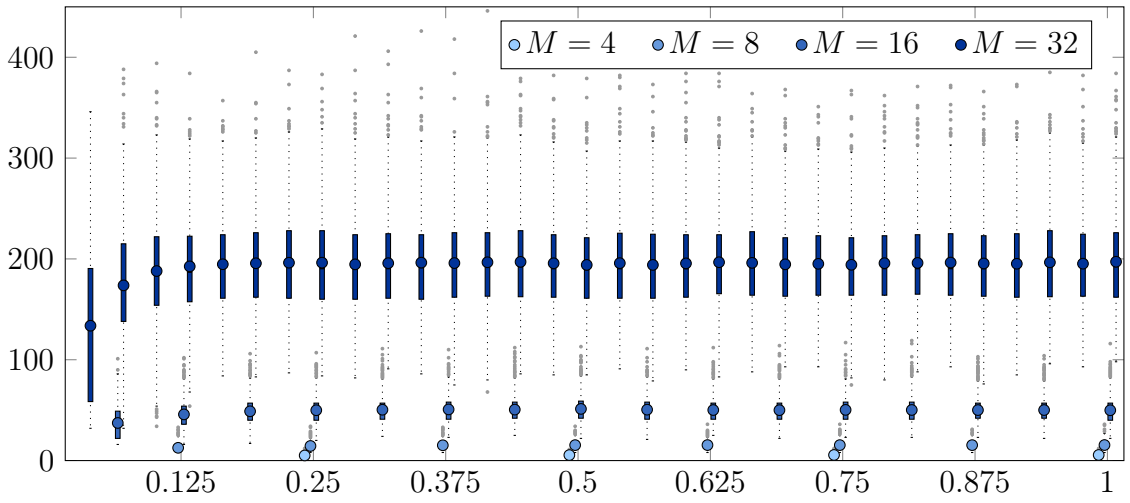
In Figure 4.9b, there are only few data points for small M, since $k \in \mathbb{N} \leq M$, but the indicated behavior is the same for all plotted values of M. For very small $\frac{k}{M}$, up to about 0.13, fewer steps are needed, while there's virtually no difference in the median or the variance for larger $\frac{k}{M}$.
More interesting is not just the growing average of the iterations, but also the increasing spread for larger dimensions as indicated by the bigger boxes.

4.4.2. Steps in the Second Simplex Phase

In the second part, the required iterations for the second Simplex phase are evaluated by simulation. The identical zero values for $\frac{k}{M} = 1$ in Figure 4.11 are conspicuous. As mentioned before, every non-degenerated corner is M-sparse. Because the sparsity $k = M$ is committed to the SAS variants, they regard every corner as degenerated and stop after the first Phase. But those values of k are not interesting since there is no reconstruction possible. The uniqueness is only given for $k < M$ as derived in Section 4.1.1.
Also for large $\frac{k}{M}$, all variants require approximately the same number of steps. In some cases, even the outliers are the same. This shows that the sparse solution is not found as a neighbor on the path, because Ⓒ only regards the visited corners. Therefore, all algorithms proceed until the ℓ_1-solution is found. Again, the rate for a success reconstruction of the sparse vector in those cases tends to zero, and therefore, they are not important.
The most interesting cases for small $\frac{k}{M}$, possessing a high success rate (see Figure 4.4), the number of steps varies most. Ⓒ needs more steps than Ⓔ and Ⓖ.



(a) $M = 16$, varying N over k/M.



(b) $N/M = 2$, varying M (and N) over k/M.

Figure 4.9.: Boxplot of the number of required Simplex steps for the first Simplex phase using Gaussian sensing matrices. For each of the 1000 runs per $\frac{k}{M}$ and $\frac{N}{M}$ different matrices, sparse vectors and permutations for the positions of the non-zeros are generated.

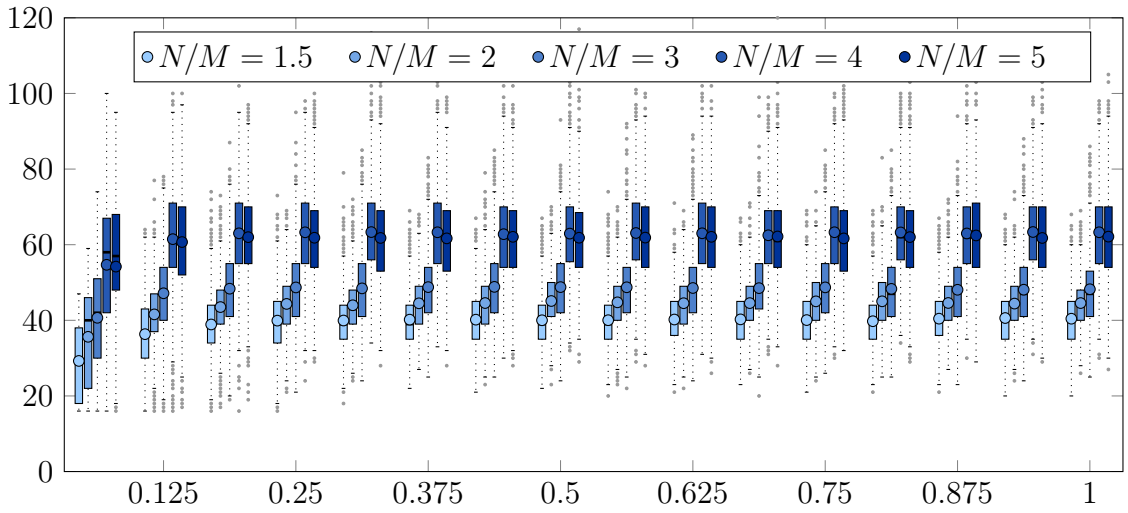


Figure 4.10.: Boxplot of required iterations over k/M in the first phase by the different variants (see Table 4.1) of the algorithms utilizing BASC matrices for $M = 16$.

Those two variants find the degenerated corner early as a neighbor on their way to the ℓ_1-solution, Ⓒ directly on the way, but closer to the end. Since Ⓔ and Ⓖ require almost the same number of steps, one can conclude for small $\frac{k}{M}$, it is likely for degenerated corners to occur in a positive direction. Where positive means $\alpha_i > 0$ in the last row of the corresponding Simplex tableau (ref. to (3.3)).
The default Simplex is not included in this comparison, because it utilizes the rule of Bland, and therefore, requires way more steps as seen in the Appendix in Section A.4.2. Figures for different dimensions can be found there too.

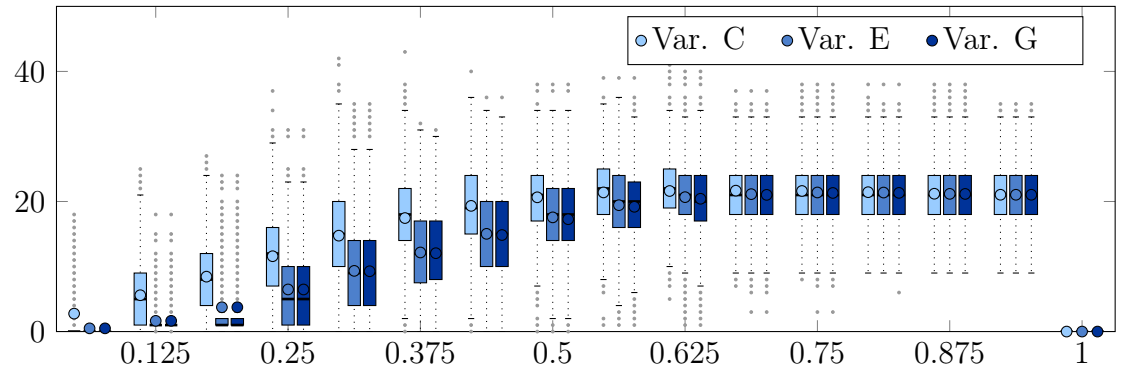


Figure 4.11.: Number of Simplex steps needed by the different variants for the second phase with $N = 32$, $M = 16$ over k/M.

5. Implementation Issues

The whole code for this thesis was written in MATLAB [21]. It is not especially optimized to any criteria. During implementation, several obstacles had to be considered. This chapter gives an overview to those details in order to simplify the reproducibility and to show the used methods.

5.1. Numerical Problems

A major problem for implementation is the numerical imprecision due to the number representation. In theory, the sensing process $\boldsymbol{\Phi x} = \boldsymbol{\mu}$ and the reconstruction of the sparse solution are defined and calculated over $\mathbb{R}$, but MATLAB uses *double* floating point numbers. They have only a precision of approximately up to 10^{-16} [21]. By applying arithmetical operations, this inaccuracy accumulates, and therefore, the more operations are performed, the larger is the difference to the proper value [22].

This behavior needs to be taken into account for comparisons in the algorithms. There are many steps where it has to be checked whether a value is zero or not, or if two ratios are equal. Every time a ε-neighborhood with $\varepsilon > 0$ needs to be defined. However, a good value for this ε, in order to get the correct results, is not known so far. E.g. reconstructing a vector from a Simplex tableau of a degenerated corner results in a lot of components that should be zero but have an amplitude of about 10^{-15}. For verification whether it is a sparse solution or not, one has to adjust ε. If it is chosen too small, the algorithm might continue the calculation, leave the degenerated corner and ends up finding a wrong reconstruction. If chosen too large, a wrong corner with components with a small amplitude might be treated as a degenerated corner and the algorithm also ends up finding a wrong solution.
The fact that more operations lead to higher inaccuracy and cause a precision loss is the core problem. It is possible to choose the corresponding ε larger, but this results sometimes in failures when two different values are treated as if they are equal. In this thesis, ε is chosen empirically. The results for small dimensions up to $M = 32$ and $N = 128$ are satisfying. But for larger dimensions, the probability grows, that at least one component of a vector falls below this ε-threshold and is treated as zero, even if it is not. Because the SAS usually stops when it finds a g-sparse corner

with $g < M$, the indicated imprecision degrades the success rate. To prevent this, one can adapt the SAS to stop only if a g-sparse corner with $g < M - B$, $B > 0$, is found. Thus, solutions with $k \geq M - B$ could not be retrieved anymore. Since the reconstruction rate for those cases is very bad (see Figure 4.5) and the algorithms are commonly used for small values of k, this trade-off is justifiable.

The especial sensitivity to imprecision of the ratio test for choosing the pivot row is noticeable during implementation. Since it is important, particularly for the SAS, an additional strategy is utilized. Instead of taking one ε, two values ε_1 and ε_2 with $\varepsilon_1 > \varepsilon_2$ are chosen empirically. If $|r_1 - r_2| < \varepsilon_1$ for two ratios r_1, r_2 in the test, then a Simplex step, with one of the corresponding rows as pivot row, is performed and checked whether a degenerated corner has been reached. If that's the case, the algorithm ends successfully. Else, *backtracking* is done, which means the last Simplex step is withdrawn. Then ε_1 is reduced, as long as $\varepsilon_1 > \varepsilon_2$, and the next step takes the pivot row provided by the minimal ratio test. The algorithm continues until either a degenerated corner or the ℓ_1-solution is found.
This improves the rate and reduces the importance of a well-adapted ε.

5.2. Highly Degenerated Corners

A highly degenerated corner is a corner in the solution space with $x_i = 0$ for almost all components x_i. They are untypical for most applications of the Simplex algorithm and lead to problems during implementation.

5.2.1. Secondary Objective Function

Usually, for non-sparse linear optimization one tries to evade degenerated corners. For sparse reconstruction, they are desired due to the containing solution, but only in the second Simplex phase. There, the algorithm stops successfully after reaching a degenerated corner. But as mentioned in Chapter 3, the goal of the first phase is to reach a valid corner of the solution space. Because the Simplex tableau contains artificial variables for this purpose, a degenerated corner in the first phase is not already a solution for the sparse reconstruction problem. In general, the end of the first phase is recognized due to fact, that the secondary objective function reaches its maximum $z' = 0$. In highly degenerated cases, the value $z' = 0$ is reached often, even if columns of the artificial variables do still consist of unit vectors. The problem here is, that after the reduction of the tableau, the application of further Simplex steps might lead to corners outside of the valid solution space. To prevent this, the

criteria to stop the first Simplex phase needs also to contain $\forall j : \alpha_j \leq 0$, which is already utilized in Section 3.1.3.

5.2.2. Anti-Cycling Strategy

Due to numerical imprecision and the extraordinary amount of zeros in those highly degenerated cases, the algorithm often starts cycling. Even the rule of Bland fails in combination with the minimal ratio test. The problem is, that these zeros also lead to ratios of value zero, which should be regarded for not leaving the valid solution space, but do not improve the objective function. Therefore, another strategy is proposed:

As preparation, let be $\mathcal{U}$ be the set of columns consisting of unit vectors (ignoring the row for the objective function). One looks for $\mathcal{U}' \subset \mathcal{U}$ corresponding to the artificial variables. Since $x_i \in \mathbb{R}$ is considered, those are the columns $i > 2N$. All rows of $\mathcal{U}'$ containing ones form the set $\mathcal{R}$.

Choose pivot column c by rule of Bland.

Case 1: If all ratios of the minimal ratio test of column c are unequal to zero, use the thereby given pivot element and continue as usual with the Simplex step.

Case 2: If at least one ratio equals zero, do case-by-case analysis:

- **Case (a):** If the minimal ratio test, applied to the rows with $b_i > 0$, returns a pivot row $r \in \mathcal{R}$, then $a_{r,c}$ is the pivot element.
- **Case (b):** Else, sort the coefficients of column c $a_{i,c}$ descending and check them consecutively if $i \in \mathcal{R}$. If yes, $a_{i,c}$ is the pivot element.
- **Case (c):** If no pivot element is chosen so far, check if $\mathcal{R}$ is empty.
  - **Case (i):** If $\mathcal{R} \neq \emptyset : \forall i \ \alpha_i > 0$: Look for a row providing a positive ratio at any $r \in \mathcal{R}$ for pivot.
  - **Case (ii):** If $\mathcal{R} = \emptyset : \forall i \ \alpha_i > 0$: Look for any positive ratio for pivot.

The strategy works as follows: As long as no zeros occur, the behavior is as usual (see Chapter 3). When zeros appear, the goal is to eliminate the unit vectors in the columns of the artificial variables, so they are set to zero at the extraction of the solution from the tableau. This can be achieved by choosing rows of $\mathcal{R}$, which are the rows of the unit vector columns for the artificial variables that contain ones. If

there's no such row in the column c, given by the rule of Bland, then one has to take another column. If $\mathcal{R}$ is empty, the artificial variables are already eliminated and the remaining goal is to maximize the secondary objective function with any valid pivot element, that doesn't lead to leaving the solution space.

Due to the structure of the sparse reconstruction problem, the solvability of the first phase is known. Hence, a more simple strategy could have been given, but the proposed one is more general and possibly also covers cases of imprecision.

5.3. Hashtables for Histograms

For the calculation of the histograms, another program has been developed. The main problem is the huge number of corners that needs to be regarded. The procedure is to start with the ℓ_1-solution and do a breadth-first search. But since the graph is cyclic, a lot of corners are regarded multiple times. To handle this exponential growth of possible neighbors, already checked solutions need to be marked. The enumeration of the corners is not feasible and not effective. Therefore, an approach utilizing hashtables is applied. Each regarded corner is checked whether it is sparse or not and if it is already contained in the hashtable. For this purpose, the hashvalue of the corresponding vector $\boldsymbol{x}$ is chosen as key element in the table. If the key does not exist so far, it is added to the table and the Simplex tableaus of all neighbors of the regarded corner are added into a queue (together with the current depth). If the key already exists, the corner is skipped and the next one is taken from the queue. As soon as the unique sparse corner is found, the algorithms stops and returns the calculated distance.
The previously mentioned numerical effects harden the problem, because small differences give a completely different hash. To prevent this, every component of $\boldsymbol{x}$ is rounded to six decimal places before hashing.

Using this approach, it is possible to determine the distances for small N, M as seen in Figures 4.2 and 4.3 but only up to a certain sparsity $\frac{k}{M}$. Afterwards the distances get too large, resulting in too many corners to check. These lead to a heap space exception due to an overflowing queue, limited by provided system resources.

6. Conclusion and Future Research

In this thesis, the Simplex algorithm is explained in detail and several modifications for the application in sparse vector recovery are proposed. Those are compared among themselves and to the popular OMP as reference.

To estimate the expected improvement due to the modifications more accurately, a statistical analysis of the solution space is performed. The distance from the ℓ_1- to the ℓ_0-solution, for varying k, is visualized in histograms for two different dimensions. It turns out that the solutions are very close to each other for small values of k, but the distribution shifts for larger k.
The simulation operating on Gaussian sensing matrices shows a differently large gain for all variants in comparison to the default Simplex algorithm. For small values of N and M, this gain is higher and it decreases for larger dimensions.
Utilizing coherence optimized sensing matrices, here BASC-based [11], raises the reconstruction rate even further.

Furthermore, the runtime of the algorithms is investigated by statistical analysis. The number of Simplex steps in the two different Simplex phases is measured for changing parameters N, M, k. It turns out that the number of steps in the first phase depends almost just on M, i.e. the number of equations. It is the same for all variants. Whereas for the second Simplex phase, the number of required steps differs for the variants, especially for small $\frac{k}{M}$. All of them need fewer steps than the default Simplex algorithm utilizing the rule of Bland, but the more complex variants reduce the number even further.

At last, obstacles for implementation are discussed: numerical problems, the behavior for highly degenerated corners and the huge number of corners. Most of them are due to the special, highly sparse optimization problem. Applied countermeasurements are presented.

More research could be done regarding several questions. Is it possible to avoid occurring imprecision by using a number representation as fractions for reconstruc-

tion? Therefore, the sensing matrices have to be adapted and the elimination steps should be calculated over $\mathbb{Q}$. This would also need the components of the sparse vector to be rational numbers, but might simplify the reconstruction. Additionally, this might lead to some noise resistance and its quality should be determinable thereby.

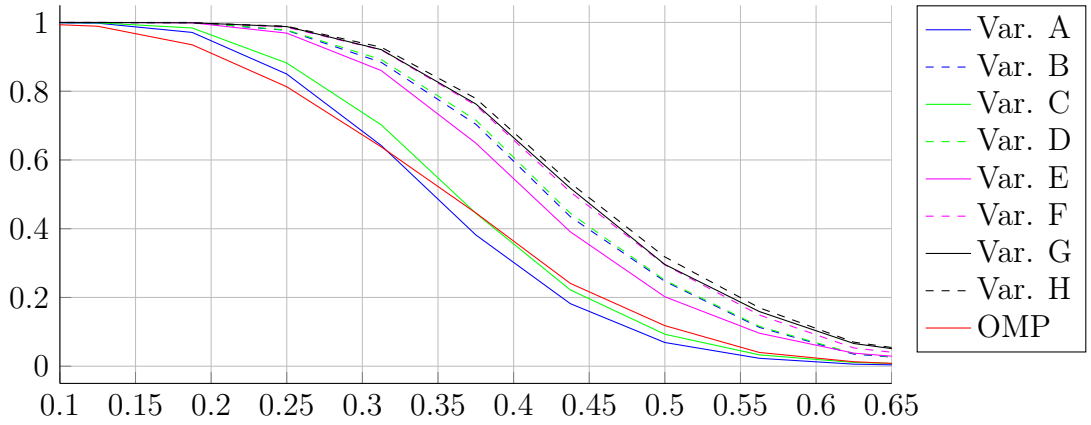
The next question rises due to the fact, that the whole investigation in this thesis is done without noise taken into account. Hence, the next step should be the analysis of the behavior for those cases. For this purpose, an additional ℓ_2-minimization could be introduced in the style of [23], where this is performed using a Simplex extension.

In addition, a more precise estimation of the distribution of the distance from the ℓ_0- and ℓ_1-optimal corner would be desired. Thus, better adapted strategies could be developed.

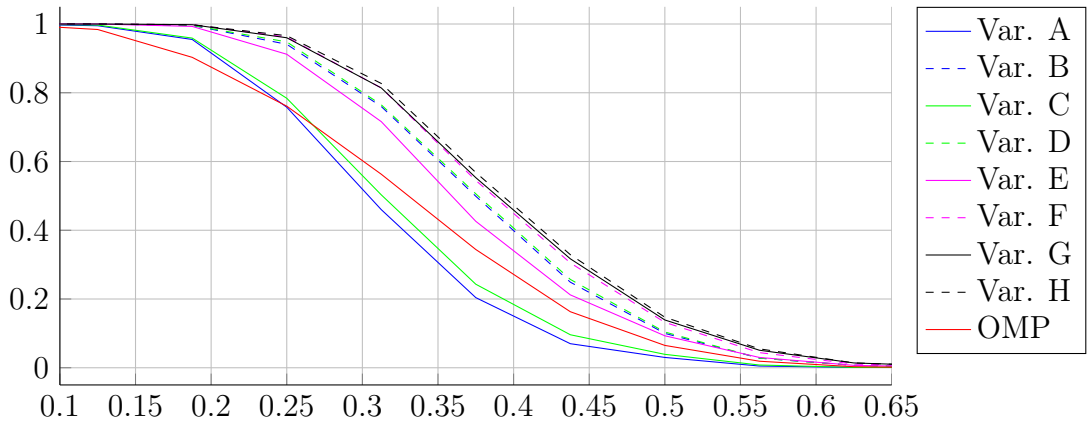
And last but not least, some research on the complexity of the proposed variants of the sparsity aware Simplex algorithm needs to be performed. Not just a theoretical analysis regarding required floating point operations would be nice, but also some approaches to reduce their number. E.g., the number of variables doubles for $x_i \in \mathbb{R}$ instead of $x_i \geq 0$. Maybe there is a way to evade this step, and therefore, the increased complexity.

A. Appendix

A.1. Success Rates for Gaussian Matrices

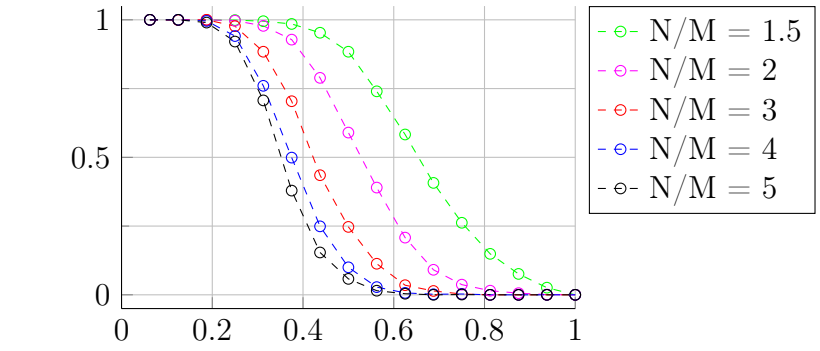


(a) $N = 48$, $M = 16 \leftrightarrow N/M = 3$

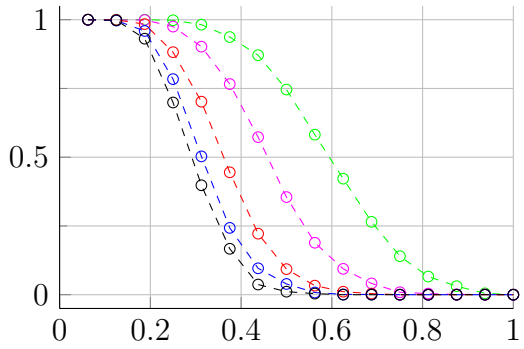


(b) $N = 64$, $M = 16 \leftrightarrow N/M = 4$

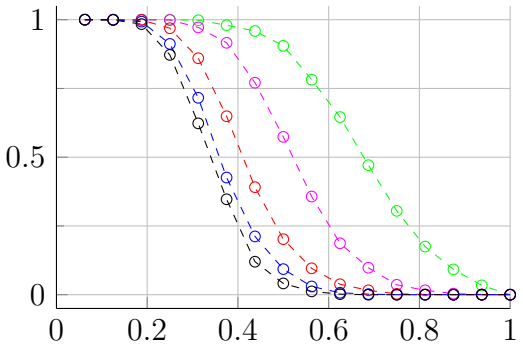
Figure A.1.: Comparison of the success rate over k/M for the algorithms from Table 4.1 for Gaussian matrices averaged over 1000 runs for each k with success if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.



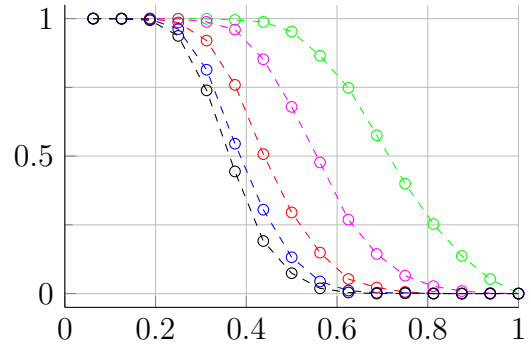
(a) Variant B



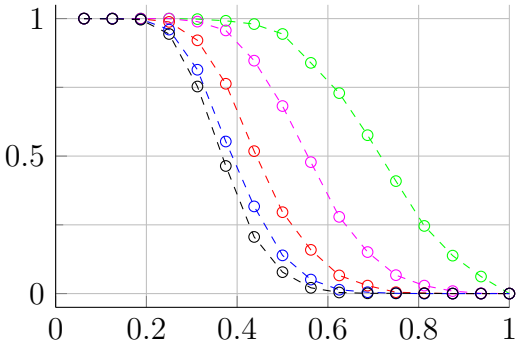
(b) Variant C



(c) Variant E

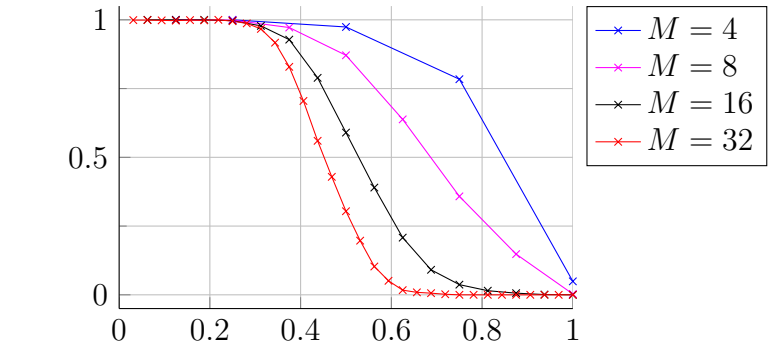


(d) Variant F

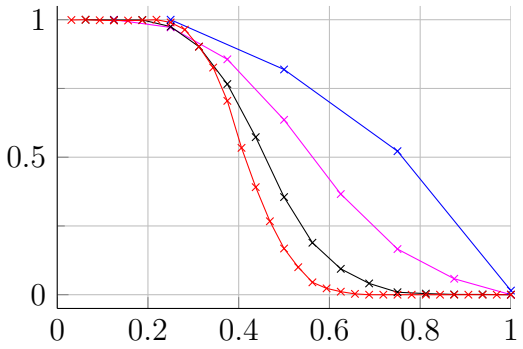


(e) Variant G

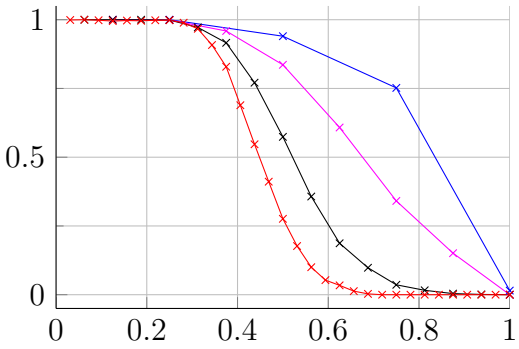
Figure A.2.: Success rate over k/M for the remaining variants of the analysis from Section 4.2.1. The value $M = 16$ is fixed, N varies. The reconstruction is successful, if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.



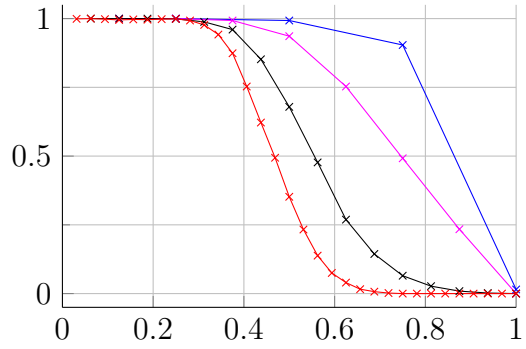
(a) Variant B



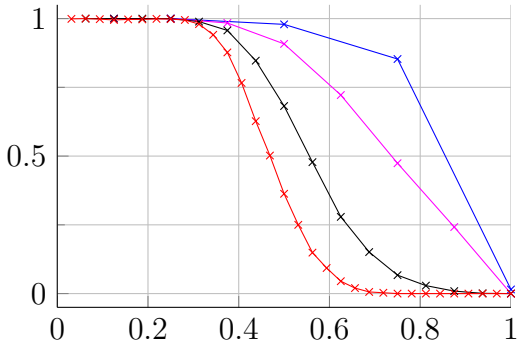
(b) Variant C



(c) Variant E



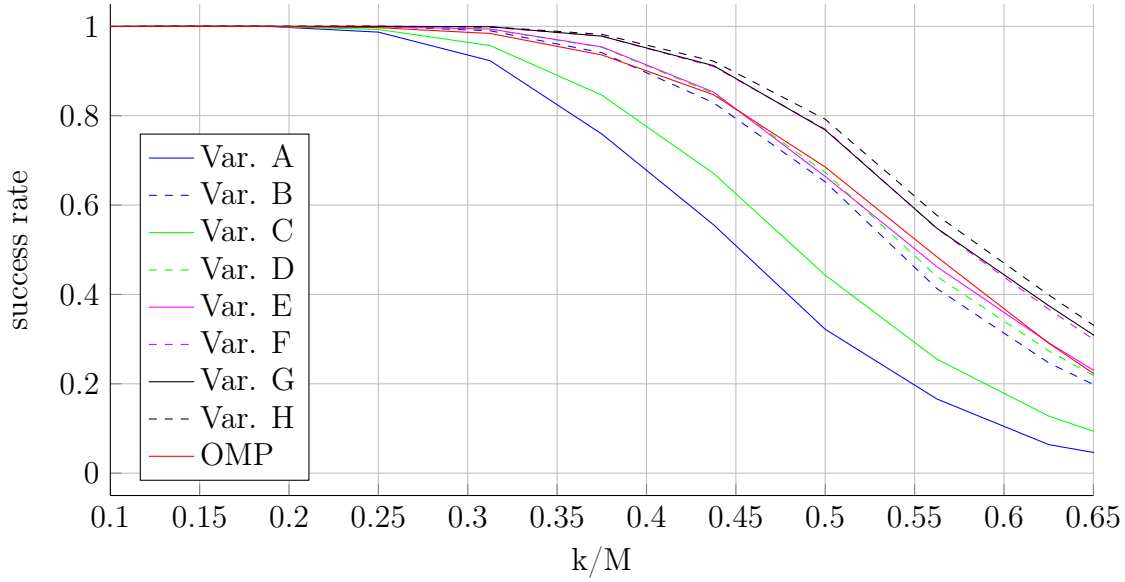
(d) Variant F



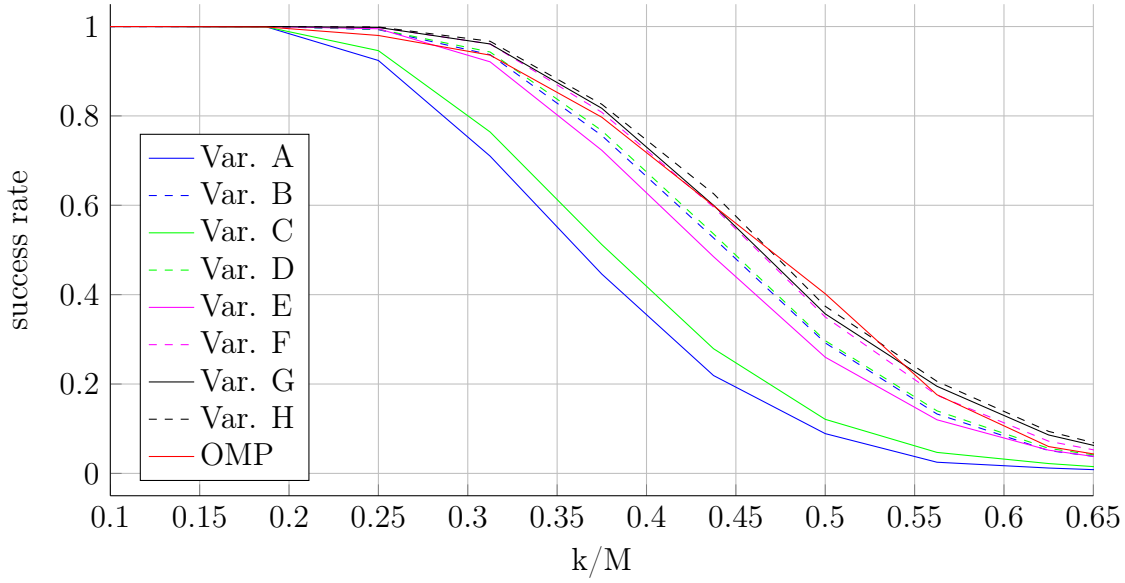
(e) Variant G

Figure A.3.: Success rate over k/M for the remaining variants of the analysis from Section 4.2.1. The value $N/M = 2$ is fixed, N and M vary. The reconstruction is successful, if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.

A.2. Success Rates for BASC Matrices

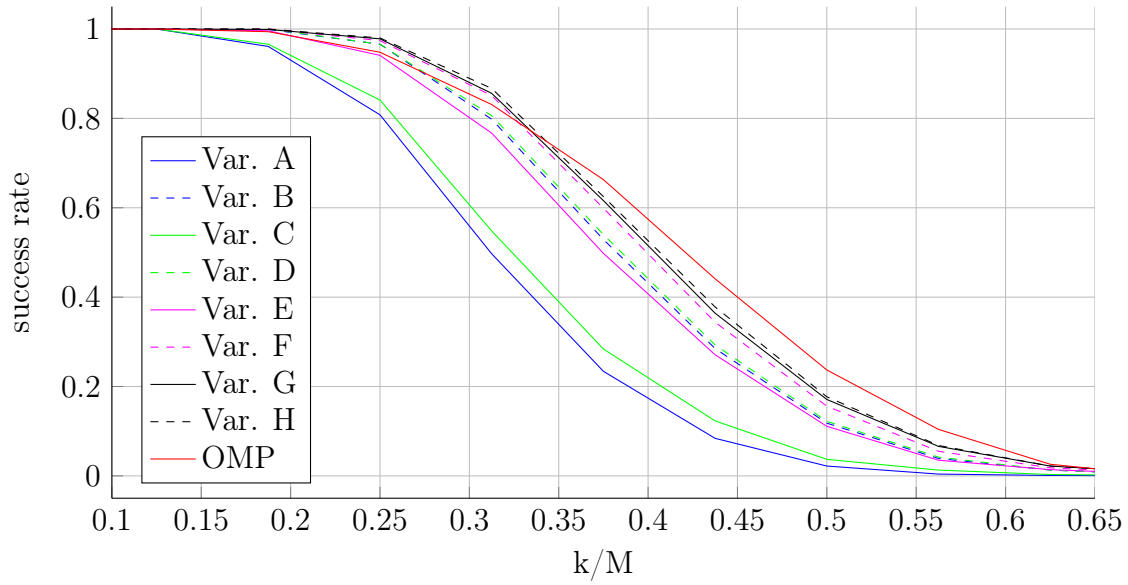


(a) $N = 32$, $M = 16 \leftrightarrow N/M = 2$

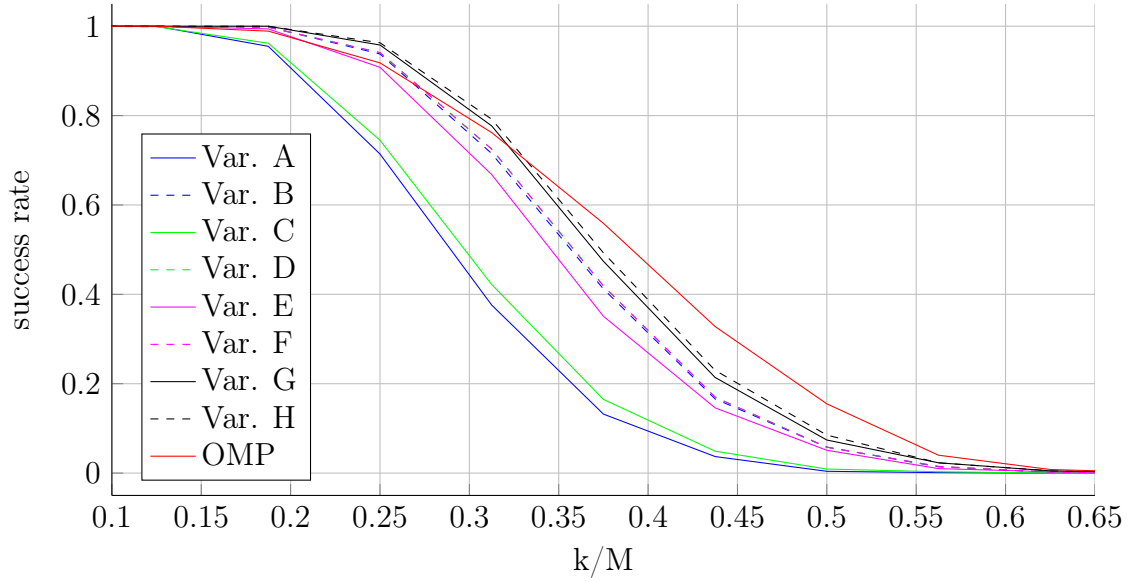


(b) $N = 48$, $M = 16 \leftrightarrow N/M = 3$

Figure A.4.: Comparison of the success rate over k/M for the algorithms from Table 4.1 for BASC matrices averaged over 1000 runs for each k with success if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.

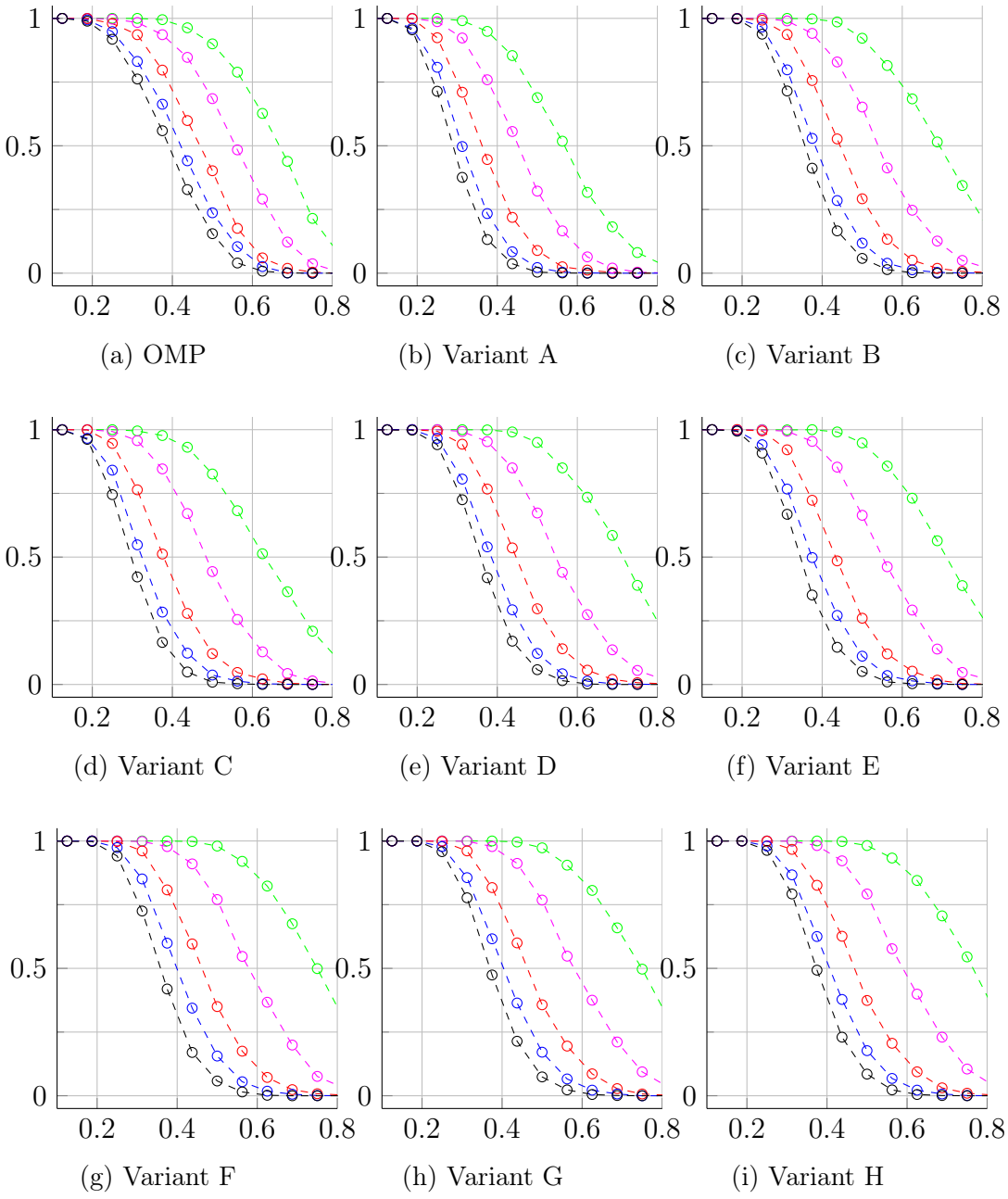


(a) $N = 64$, $M = 16 \leftrightarrow N/M = 4$



(b) $N = 80$, $M = 16 \leftrightarrow N/M = 5$

Figure A.5.: Comparison of the success rate over k/M for the algorithms from Table 4.1 for BASC matrices averaged over 1000 runs for each k with success if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| \leq 10^{-14}$ componentwise.

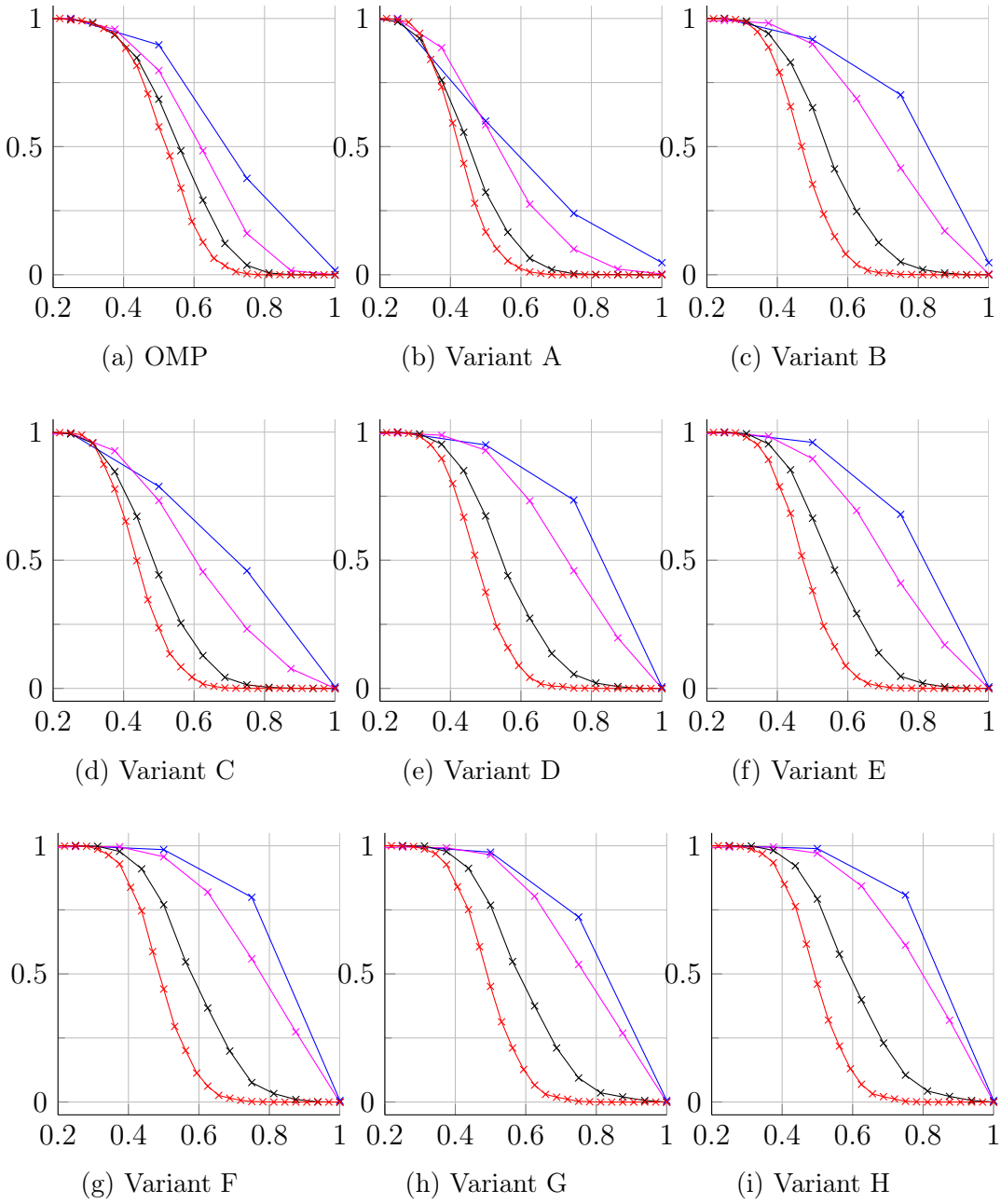


(a) OMP (b) Variant A (c) Variant B

(d) Variant C (e) Variant D (f) Variant E

(g) Variant F (h) Variant G (i) Variant H

Figure A.6.: Success rate over k/M for $M = 16$ with $N/M = 1.5$ (green), $N/M = 2$ (magenta), $N/M = 3$ (red), $N/M = 4$ (blue) and $N/M = 5$ (**black**) averaged for 1000 runs with BASC matrices for comparison to Figures A.2 and 4.6 analyzed in Section 4.2.1.



(a) OMP (b) Variant A (c) Variant B

(d) Variant C (e) Variant D (f) Variant E

(g) Variant F (h) Variant G (i) Variant H

Figure A.7.: Success rate over k/M for $N/M = 2$ with $M = 4$ (blue), $M = 8$ (magenta), $M = 16$ (**black**), $M = 32$ (red) averaged for 1000 runs with BASC matrices for comparison to Figures A.3 and 4.8 analyzed in Section 4.3.

A.3. Comparison of Success Rate for BASC and Gaussian Sensing Matrices

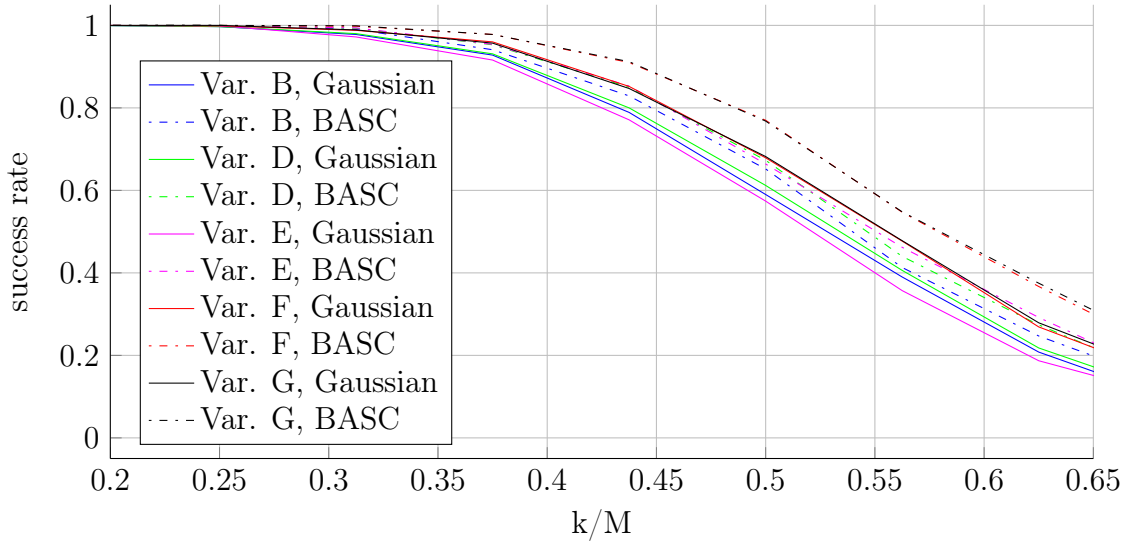


Figure A.8.: Influence of BASC matrices on the remaining variants of the analysis from Section 4.2.2 for $N = 32$, $M = 16$. The reconstruction is successful if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| < 10^{-14}$ componentwise.

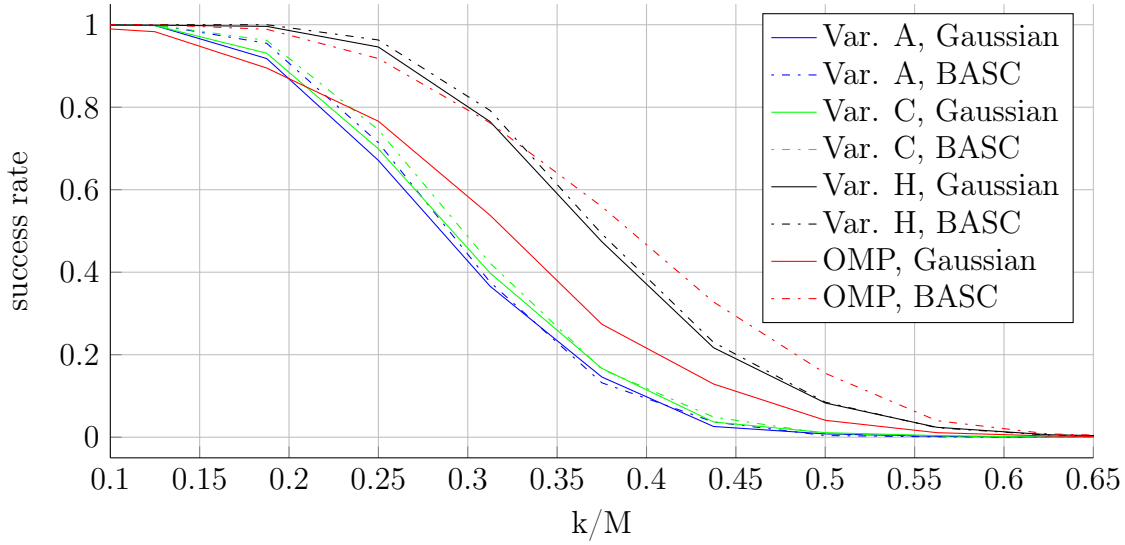
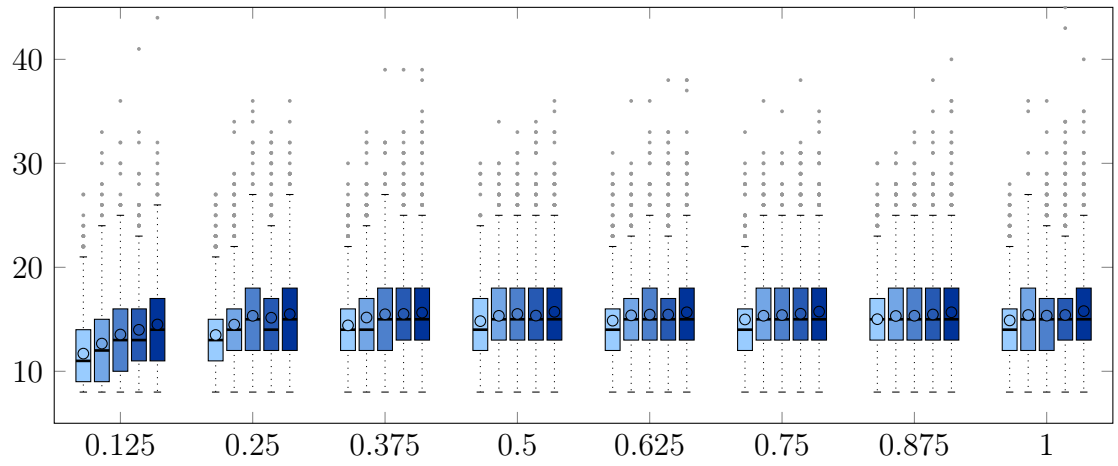


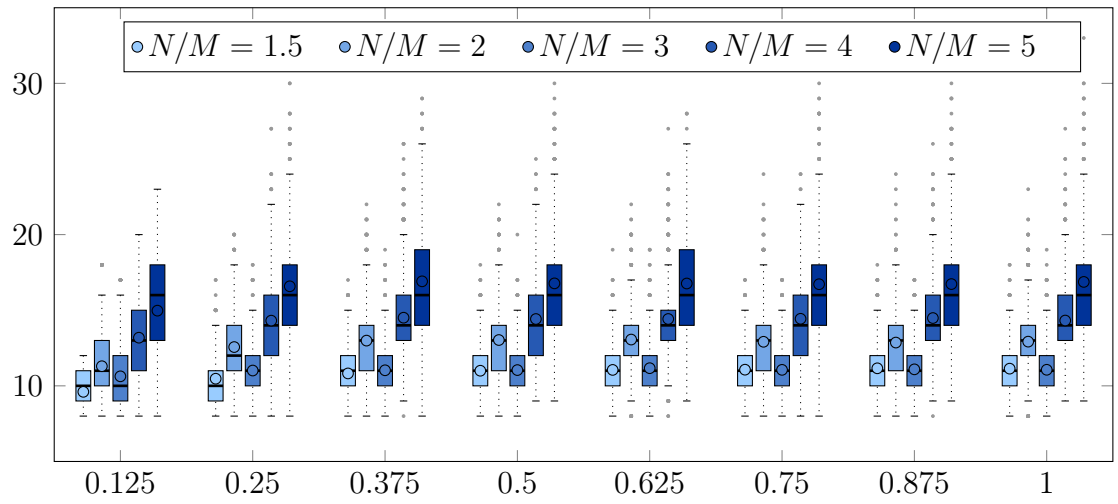
Figure A.9.: Influence of BASC matrices on the variants of the analysis from Section 4.2.2 for different dimensions $N = 80$, $M = 16$, $N/M = 5$. The reconstruction is successful if $|\hat{\boldsymbol{x}} - \boldsymbol{x}| < 10^{-14}$ componentwise.

A.4. Runtime Analysis in Simplex Steps

A.4.1. Simplex Phase One



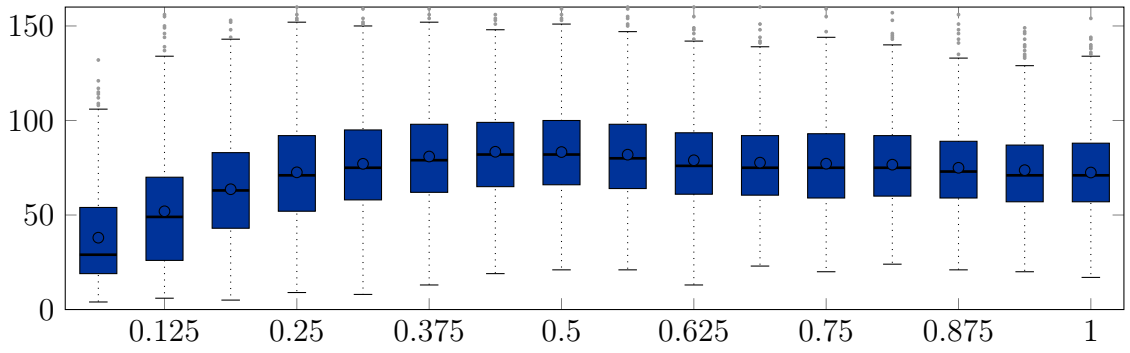
(a) For Gaussian sensing matrices.



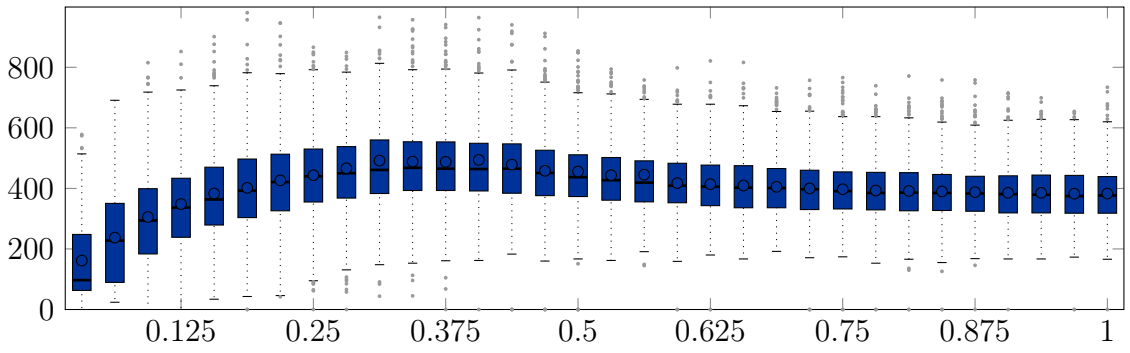
(b) For BASC sensing matrices.

Figure A.10.: Iterations required over k/M in the first phase by the different variants from Table 4.1 of the algorithms for $M = 8$.

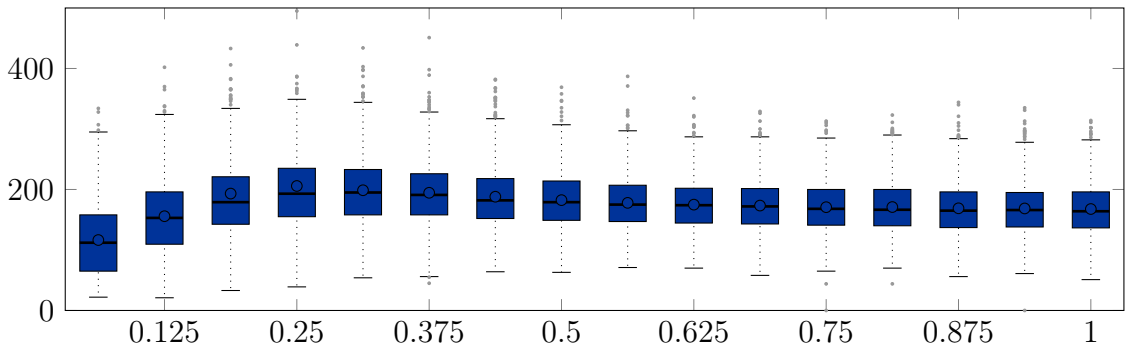
A.4.2. Simplex Phase Two using Rule of Bland



(a) $N = 32,\ M = 16 \leftrightarrow N/M = 2$



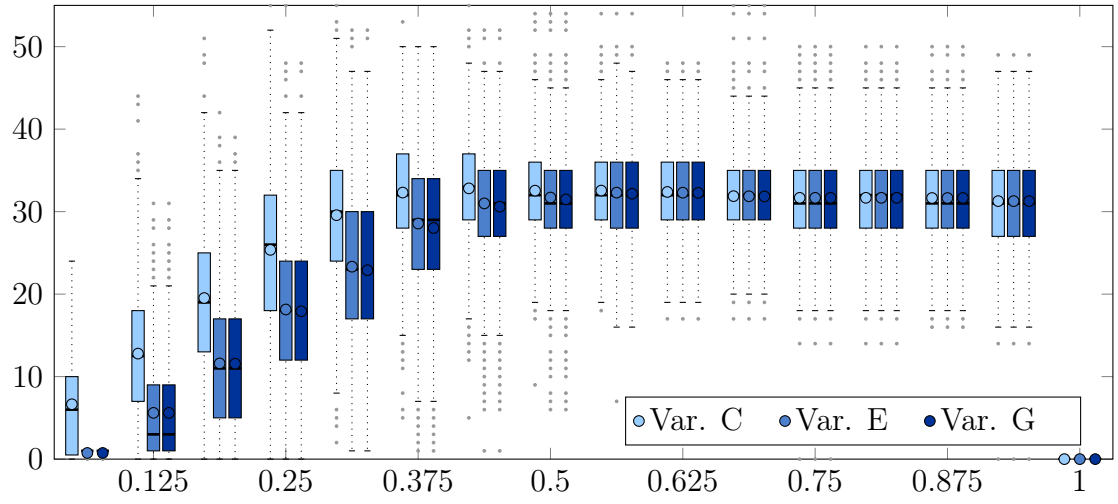
(b) $N = 64,\ M = 32 \leftrightarrow N/M = 2$



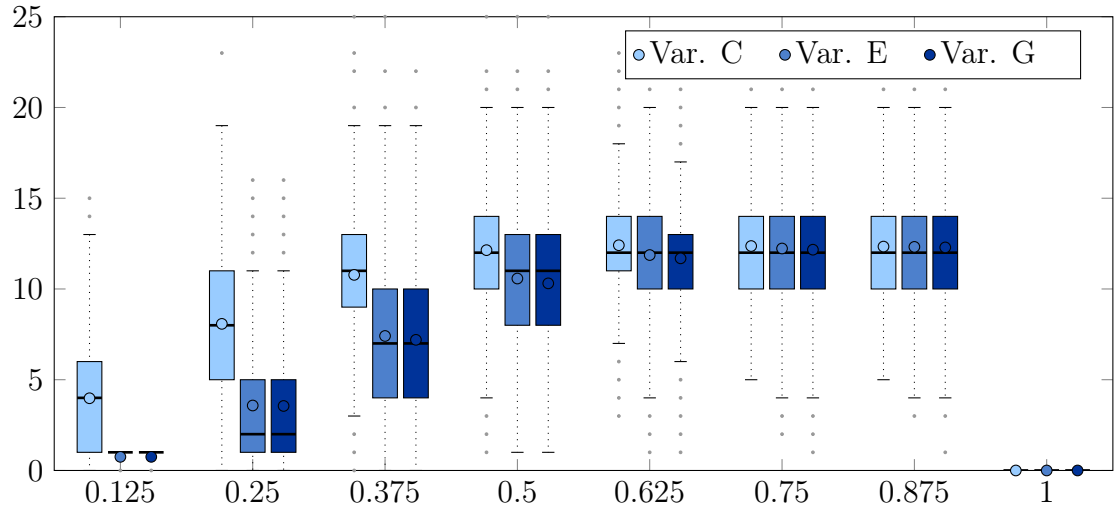
(c) $N = 64,\ M = 16 \leftrightarrow N/M = 4$

Figure A.11.: Iterations required by Var. A (default Simplex algorithm) using rule of Bland for the second phase using Gaussian matrices over k/M.

A.4.3. Simplex Phase Two for Gaussian Matrices



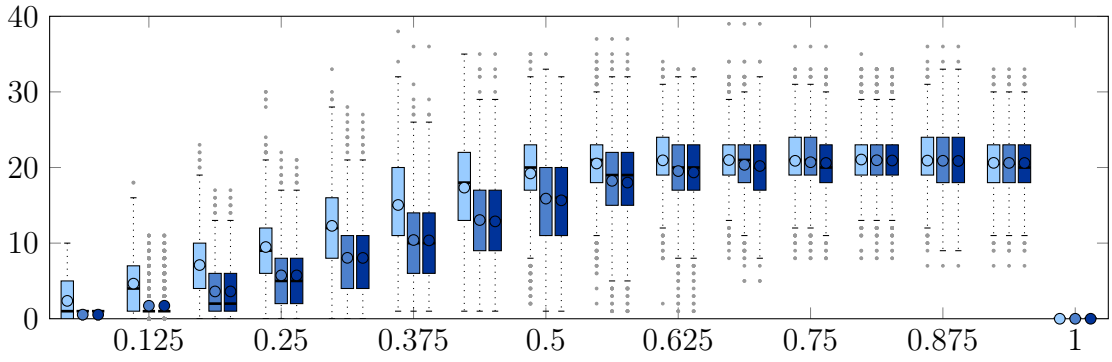
(a) $N = 48$, $M = 16 \leftrightarrow N/M = 3$



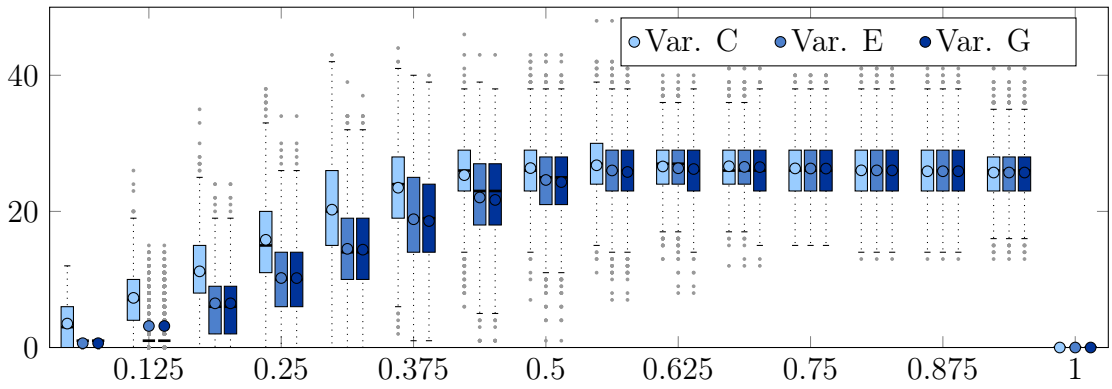
(b) $N = 32$, $M = 8 \leftrightarrow N/M = 4$

Figure A.12.: Comparison of required Simplex steps for the second phase for several variants from Table 4.1 over k/M using Gaussian matrices.

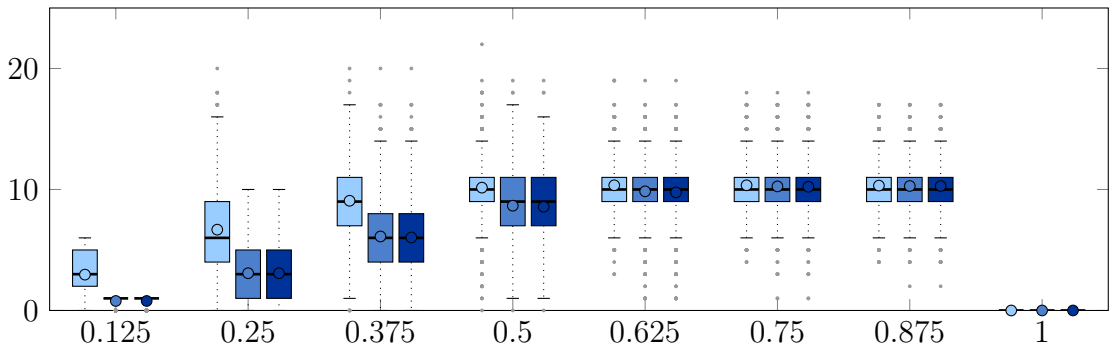
A.4.4. Simplex Phase Two for BASC Matrices



(a) $N = 32$, $M = 16 \leftrightarrow N/M = 2$



(b) $N = 48$, $M = 16 \leftrightarrow N/M = 3$



(c) $N = 32$, $M = 8 \leftrightarrow N/M = 4$

Figure A.13.: Comparison of required Simplex steps for the second phase for several variants from Table 4.1 over k/M using BASC matrices.

List of Figures

Bibliography

[1] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[2] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[3] G. B. Dantzig, A. Orden, and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality restraints." *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[4] D. Needell and J. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, no. 3, pp. 301 – 321, 2009.

[5] E. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[6] A. Cohen, W. Dahmen, and R. A. DeVore, "Compressed sensing and best k-term approximation," *Journal of the American Mathematical Society*, vol. 22, no. 1, pp. 211–231, 2009.

[7] S. Foucart and H. Rauhut, *A mathematical introduction to compressive sensing*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser, 2013.

[8] T. Arens, F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger, and H. Stachel, *Mathematik*, 1st ed. Spektrum Akademischer Verlag, 2 2008.

[9] R. Baraniuk, M. Davenport, R. Devore, and M. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, vol. 28, no. 3, pp. 253–263, 2008.

[10] A. Amini and F. Marvasti, "Deterministic construction of binary, bipolar, and ternary compressed sensing matrices." *IEEE Transactions on Information Theory*, vol. 57, no. 4, pp. 2360–2370, 2011.

[11] D. E. Lazich, H. Zörlein, and M. Bossert, "Low coherence sensing matrices based on best spherical codes," in *Proceedings of 9th International ITG Conference on Systems, Communication and Coding (SCC)*, Munich, Germany, Jan. 2013.

[12] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 227–234, Apr. 1995.

[13] U. Schöning, *Algorithmik*. Spektrum, Akademischer Verlag, 2001.

[14] S. Chen and D. Donoho, "Basis pursuit," in *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, vol. 1, Oct 1994, pp. 41–44 vol.1.

[15] D. Ge, X. Jiang, and Y. Ye, "A note on the complexity of lp minimization." *Mathematical Programming*, vol. 129, no. 2, pp. 285–299, 2011.

[16] E. J. Candès, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9-10, pp. 589 – 592, 2008.

[17] S. Foucart, "A note on guaranteed sparse recovery via -minimization," *Applied and Computational Harmonic Analysis*, vol. 29, no. 1, pp. 97 – 103, 2010.

[18] D. E. Lazich, "Compressed sensing," University Lecture, Ulm, 2012/13.

[19] D. M. Sommerville, *Introduction to the Geometry of N Dimensions, with Sixty Diagrams*. E.P. Dutton & Co., 1929.

[20] H. Zörlein, D. E. Lazich, and M. Bossert, "On the Noise-Resilience of OMP with BASC-Based low coherence sensing matrices," in *Proceedings of 10th international conference on Sampling Theory and Applications (SampTA)*, Bremen, Germany, Jul. 2013, pp. 468–471.

[21] MATLAB, *version 8.2.0.701 (R2013b)*. Natick, Massachusetts: The MathWorks Inc., 2013.

[22] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

[23] P. Wolfe, "The simplex method for quadratic programming," *Econometrica*, vol. 27, pp. 382–398, 1959.