TECHNISCHE UNIVERSITÄT
ILMENAU

Fakultät für Elektrotechnik und Informationstechnik
Fachgebiet Drahtlose Verteilsysteme / Digitaler Rundfunk

Bachelorarbeit

Phase Transition Diagrams in Compressive Sensing - Computation and Analysis

vorgelegt von
Herrmann, Martin

Matrikelnummer:	48388
Geboren am:	07. März 1988 in Stuttgart
Studiengang:	Medientechnologie
Verantwortlicher Hochschullehrer:	Univ.-Prof. Dr.-Ing. Giovanni Del Galdo
Betreuender wiss. Mitarbeiter:	Dr.-Ing. Florian Römer

Ilmenau, May 8, 2015

urn:nbn:de:gbv:ilm1-2015200116

TECHNISCHE UNIVERSITÄT ILMENAU

Fakultät für Elektrotechnik und Informationstechnik

Thema - Bachelorarbeit

für:	Martin Herrmann
geboren am:	07.03.1988
Studiengang:	**Medientechnologie**
Matrikelnummer:	48388
Verantwortliche/r Hochschullehrer/in:	Univ.-Prof. Dr.-Ing. Giovanni Del Galdo
Betreuende/r wissenschaftliche/r Mitarbeiter/in:	Dr.-Ing. Florian Römer
Vorsitzende/r der Prüfungskommission:	Univ.-Prof. Dr.-Ing Giovanni Del Galdo
2. Gutachter/in:	Dr.-Ing. Florian Römer
Thema:	Phase Transition Diagrams in Compressive Sensing - Computation and Analysis
Ausgegeben am:	15.12.2014
Abgabetermin:	14.06.2015
Umfang:	360 Stunden

Ilmenau, den 28.11.2014

Univ.-Prof. Dr. phil. H. Krömker
Vorsitzender des Prüfungsausschusses
des verantwortlichen Studienganges

Verantwortlicher Hochschullehrer

Contents

Themenblatt i

Contents iii

1. Preface 1
1.1. Notation . . . 2

2. Compressive Sensing 4
2.1. Fundamentals of Compressive Sensing . . . 4
2.2. Recovery of Sparse Signals . . . 6
2.2.1. ℓ_0-minimization . . . 7
2.2.2. ℓ_1-minimization . . . 8
2.2.3. Greedy Methods . . . 9
2.3. Reconstruction Guarantees . . . 10
2.4. Summary . . . 12

3. Phase Transition Diagrams 13
3.1. Composition of Phase Transition Diagrams . . . 13
3.2. Computational Results in Literature . . . 15
3.3. Theoretical Results in Literature . . . 17
3.4. Summary . . . 19

4. Calculation of Phase Transition Diagrams 20
4.1. Description of the new Algorithm . . . 20
4.2. Description of PTD Builder, its Interfaces and Usage . . . 22
4.2.1. Structure and Installation . . . 23
4.2.2. Quick Start . . . 24
4.2.3. Parameter File *ParEx.m* . . . 24
4.2.4. Main Loop File *MakePhaseTrans.m* . . . 25
4.2.5. Evaluation File *PlotMultiPTD.m* . . . 27
4.2.6. Compressive Sensing File *CompressiveSensingClass.m* . . . 28
4.2.7. Troubleshooting . . . 33
4.2.8. Coherence Adjustment . . . 35
4.3. Computational Environment . . . 35
4.4. Summary . . . 36

5. Analysis of Phase Transition Diagrams **37**
5.1. Validation of the algorithm 37
5.2. Influence of the solving algorithm 41
5.3. Influence of the measurement matrix 46
5.4. Influence of the distribution type of the coefficient vector 51
5.5. Influence of the coherence of the measurement matrix 56
5.6. Other Influences 60
5.7. Summary 64

6. Conclusion **65**

A. Bibliography **67**

B. List of Figures **70**

C. List of Tables **72**

D. Acronyms **73**

E. Appendix **75**

F. Eigenständigkeitserklärung (Declaration of Authorship) **78**

Vereinbarung über die Verwertungsrechte **79**

1. Preface

A smart transformation of a sampled signal often leads to a representation by a small number of nonzero elements. Such signals are called sparse. Many transform coding algorithms use this as basic idea to reduce the amount of data. In image compression typically the JPEG algorithm is used. It derives a discrete cosine transformed representation of an image and quantizes the spectral components. By using some psycho optical effects, many of the frequency components can be quantized to zero, which regularly leads to a compressed version of the image, without deeper visible quality impacts. [24, cf. p. 56-64] There are plenty of applications in signal processing which utilize sparsity of signals. But all of these need a priori information about the signal.

The question is, can a sparsity approach be utilized before sampling the data? And fortunately the answer seems to be 'yes'. In 2004 Candès et al. [4] and independently Donoho [9] published articles about a mathematical method permitting exact reconstruction from incomplete measurements. While this was, prior to their researches, only possible in theory, but practically impossible, they showed a way to handle the complexity of the problem under certain conditions (see 2.2 *Recovery of Sparse Signals* for detailed information). [8, cf. p. 1-3][23, cf. p. 1]

The two mentioned articles [4, 9] constituted the scientific field of *Compressed* or *Compressive Sensing (CS)*. While the interest in this field was merely low, it monotonically increases since the publication of the mentioned articles in *IEEE Trans. Inform. Theory*. This technique challenges the *Shannon Nyquist Sampling Theorem*, which is known as yielding the lowest possible bound when sampling signals. Compressive sensing places high demands on the signal and the measurement, but allows sampling below the rate predicted by the Shannon Nyquist Sampling Theorem. It seems clear that, especially if sampling is time or cost expensive, reducing the sampling rate is a target improvement. Nowadays compressive sensing is, for example, used in seismology, radar technology or medical imaging. Roughly speaking there are many applications which would benefit of using compressed sensing techniques. But whereas the use of the Shannon Nyquist Sampling Theorem simply demands on knowledge of the maximum occurring frequency in a signal, the success of compressive sensing techniques depends on many more. The sparsity of the signal, the undersampling rate, the used algorithm or the measurement matrix, which will all be explained in detail later, play key roles. [18, cf. p. 1-3] This places high demands on engineers, implementing the technique into concrete applications. To help them finding well suited properties under given conditions, they could take a look at *Phase Transition Diagrams (PTDs)* (see 3 *Phase Transition Diagrams* for more information). These diagrams provide propositions about the probability of success of compressive sensing applications and allow the comparison of different algorithms and the estimation of the adjustable parameters. They are furthermore well suited to evaluate

the impact of different influencing factors and the verification of theoretical approaches.

But it turns out that these diagrams are often not available, due to the huge number of parameters and algorithms they depend on. Since even the length of the signal may have an influence on them, it seems yet impossible to calculate them for more than a few concrete parameters. This makes it all the more important to be able to quickly derive such diagrams and to understand their behavior when varying single parameters.

This thesis is mainly based on two papers of Donoho and Tanner [11, 12], where they proposed an algorithm to calculate phase transition diagrams. This algorithm will be improved and used to derive a new set of phase transition diagrams. Furthermore these results shall be examined and the influencing factors be structured and classified. The reader shall also gain intuitive access to the behavior of such diagrams, when varying different parameters.

1.1. Notation

All matrices and vectors in this thesis will be printed bold, in opposite to mathematical signs and scalars. Vectors $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \in \mathbb{C}^n$ are column vectors. Matrices are denoted

$$\boldsymbol{A} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \in \mathbb{C}^{n \times m}$$

Matrices with constant coefficients, as often used in programming languages, are denoted

$$\mathbf{1}_{N \times M} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \in \mathbb{C}^{n \times m}$$

whereby the number 1 is a place holder and can be changed with all other numbers.

For $1 \leq p < \infty$

$$\|\boldsymbol{x}\|_p = \left(\sum_{n=1}^{N} |x_n|^p \right)^{1/p} \tag{1.1}$$

denotes the ℓ_p-*norm*, where N is the length of the vector $\boldsymbol{x}$.[1]

[1] Equation (1.1) can also be applied to $0 < p < 1$. These are called *Quasinorms* because of them not satisfying the triangle inequality.

Often, the so called *ℓ_0-norm* is referred in context with compressive sensing. Although it is not a norm in a mathematical sense, it will be named as ℓ_0-norm in this thesis and is denoted

$$\|\boldsymbol{x}\|_0 \ = |\operatorname{supp} \boldsymbol{x}| \ . \tag{1.2}$$

Thereby supp is the support of the vector, which is the set of indices of the nonzero elements and its norm is the number of elements in this set. A vector $\boldsymbol{x}$ is called *K-sparse* if

$$\|\boldsymbol{x}\|_0 \ \leq K \ .$$

If noise is present, signals will not be exactly sparse in typical bases. They are then called *compressible*. [22, cf. p. 4-6]

The *support vector* of $\boldsymbol{x}$ is denoted $\boldsymbol{x}_\mathrm{S}$. That means the vector, resulting from the original vector, when cutting out all nonzero elements. If a matrix $\boldsymbol{A}$ is multiplied with a sparse vector $\boldsymbol{x}$ from the right side, this cuts out as much columns as nonzero elements are in the vector. The equation can then be written as $\boldsymbol{y} = \boldsymbol{A}_\mathrm{S} \cdot \boldsymbol{x}_\mathrm{S}$, where $\boldsymbol{A}_\mathrm{S}$ means $\boldsymbol{A}$ limited to the columns in the set S.

2. Compressive Sensing

Compressive sensing describes a special sampling technique. Its great advantage is the ability to massively decrease the number of samples, compared to methods commonly used today. The technique depends on the key fact, that signals can often be represented sparse, which more precisely reflects the amount of the actual information contained in these signals. Unfortunately successful use of the technique depends on several factors, which are in parts of probabilistic nature. This makes the concept of compressive sensing, at the first sight, somewhat hard to grasp and means that the truth of statements can often just be assured with overwhelming probability, instead of stated for sure.

Compressive sensing is nevertheless being used in several applications today. Especially if classic sampling of concrete signals is very expensive or due to technical restrictions not possible. Medical imaging is an application of this kind and the use of compressive sensing led to a decrease of time at the factor of about seven [8, cf. p. 4].

This chapter first introduces the main concepts of compressive sensing. The following parts describe the implementation of compressive sensing and answer the question of when its usage makes sense.

Note that this thesis considers noiseless and real valued discrete signals only. The existence of noise would go far beyond the scope of this work, though many of the concepts work there, too. In anticipation of further chapters, the restriction to noiseless data makes it easier to decide on success or failure of reconstruction. The limitation to real-valued signals, is motivated due to the fact that many natural signals are or can be assumed as real. For example audio, video and also communication data can be described with real values in the time domain. However some of the outcomes can be adopted to the complex case with little effort, some others may need more extensive research. The restriction to discrete signals is very common in literature, because the theory here is much further progressed. Nevertheless there is a working theory for continuous signals, but it shall also be excluded in this thesis.

2.1. Fundamentals of Compressive Sensing

Since Nyquist found the sampling bound $f_\mathrm{s} \geqslant 2 \cdot f_\mathrm{Ny}$, where f_s denotes the proposed sampling frequency and f_Ny the *Nyquist Frequency*, which is the highest frequency occurring in a signal, it is clear that sampling below double the Nyquist frequency leads to *Aliasing*[2]. This bound could not be challenged in most practical applications, until the

[2] distortion of the signal due to convolution of periodic spectral components

upcoming scientific field of compressive sensing. It allows massive undersampling of signals without any loss of information, but purchased with more complex signal processing steps.

As all sensing applications, CS contains two parts. The *Sampling* and *Reconstruction* or *Recovery* of the original *Signal*, which is, within the scope of this work, assumed as Nyquist sampled and quantized and is denoted with $\boldsymbol{s}$. It is written in its typical domain, while the vector $\boldsymbol{x}$, which is called the *Coefficient Vector* of the signal $\boldsymbol{s}$, describes the signal in a different domain. The specified domain shall ensure the coefficient vector $\boldsymbol{x}$ to be sparse, in opposite to the signal $\boldsymbol{s}$, which is in common not sparse. For instance audio signals are typically considered in the time domain, where of course they are not sparse. But mostly they are at least compressible if not actually sparse, in the frequency domain. And there are a lot more natural signals that are expressible sparse in at least one appropriate domain, being an orthonormal basis. It holds $\boldsymbol{s} = \boldsymbol{A} \cdot \boldsymbol{x}$, where $\boldsymbol{A}$ is called *Transformation Matrix*. The multiplication of the vector $\boldsymbol{x}$ with the transformation matrix $\boldsymbol{A}$ from the left side, is always carried along, instead of simplifying it, to bear in mind that there is a sparse representation of the signal. Thereby the length of the signal vector is N and the size of the transformation matrix then must be $N \times N$.

Sampling such a signal in a discrete environment can mathematically be described by a multiplication of a *Kernel Matrix* $\boldsymbol{\Phi}$, which is of the dimensions $M \times N$, with $\boldsymbol{s}$. This leads to a *Sampled Signal* $\boldsymbol{y}$, also called *Measurement Vector*, denoted

$$\boldsymbol{y} = \boldsymbol{\Phi} \cdot \boldsymbol{A} \cdot \boldsymbol{x} \ . \tag{2.1}$$

Sampling by default (or standard) technique means using the identity matrix as kernel matrix $\boldsymbol{\Phi}$ and will further be called *Nyquist* oder *Standard Sampling*. Taking $M \neq N$ leads to undersampling, which shall be subdivided into two groups. Easily ignoring all but every kth coefficient is called *Decimation Sampling* and means discarding all rows of the standard sampling measurement matrix, but every kth ones. On the other side there is compressive sensing, whose measurement matrices do also lead to undersampling but in a more complex way. They take the signal $\boldsymbol{s}$ and generate linear combinations of its samples, instead of just eliminating single values. This ensures, in opposite to decimation sampling, an influence of every signal sample in the measurement vector $\boldsymbol{y}$. Of course, these matrices are not limited to the identity matrix and fixed heights M and values. They can assume any other height, which stands for the number of measurements and all coefficients, also the ones outside the diagonal, can assume any value. Practical reasons can, in contrast to the pure mathematical point of view, limit the measurement matrix, when bringing the concept into real applications.

But, as mentioned, sampling is only half way there. The signal has to be recovered (also called reconstructed), which is the key step and leads to the following system of linear equations

$$\boldsymbol{y} = \boldsymbol{\Phi} \cdot \boldsymbol{A} \cdot \boldsymbol{x}^* \tag{2.2}$$

where $\boldsymbol{x}^*$ stands for the unknown vector to be recovered.

2.2. Recovery of Sparse Signals

When doing standard sampling, (2.2) has as many variables as independent equations and can efficiently be solved with standard math techniques[3]. Even more because of the structure of $\boldsymbol{A}$, which is the identity matrix, the samples can easily be read from the sampled vector. This means a reduction to an easy memory access. [5, cf. p. 1-3]

In opposite to this, compressive sensing, where $M \leq N$, leads to underdetermined systems of linear equations with either no or an infinite number of solutions. Thereby, the considered case without a solution is obsolete, because of the vector $\boldsymbol{y}$ to be known of being derived by sampling from an existing vector $\boldsymbol{x}$. Having to choose the one solution, coinciding with the original signal, out of an infinite group of solutions, makes recovery much more complex than before and is quite impossible, in common. [23, cf. p. 3-7]

When taking a more detailed look at these equations, it points out, that in some special cases recovery is indeed possible. Therefore a more detailed examination of the system of linear equations shall be done. The considered compressive sensing application is simplified by $\boldsymbol{B} = \boldsymbol{\Phi} \cdot \boldsymbol{A}$ and denoted $\boldsymbol{y} = \boldsymbol{B} \cdot \boldsymbol{x}^*$, where $\boldsymbol{y}$ is generated by multiplication with a sparse but unknown vector $\boldsymbol{x}$ (see eq. (2.1)), which shall be reconstructed. The matrix $\boldsymbol{B}$ is called *Measurement Matrix*.

As known from the Shannon Nyquist sampling theorem, the reconstruction is possible when sampling above or at the Nyquist frequency. So if the number of measurements M is higher or equal than the number of samples N, the success rate of reconstruction is at 100%. But this case is formerly known and out of the interest of this work. Using less measurements M than the signals length, is somewhat more complicated and leads to an underdetermined system of linear equations. Assume a strategy of trying all possible solutions. That means taking a support set $\boldsymbol{x}_{\mathrm{S}}^*$, which leads to a reduced system of linear equations $\boldsymbol{y} = \boldsymbol{B}_{\mathrm{S}} \cdot \boldsymbol{x}_{\mathrm{S}}^*$. The solution space now depends on the size $\tilde{K}$ of this support set and the rank of $\boldsymbol{B}$, which shall be assumed as full. Then three different cases have to be distinguished.

- If $\tilde{K}$ is greater than the number of measurements M[4], the reduced system has more unknowns than equations and is solvable with $M - \tilde{K}$ degrees of freedom, but hence it has an infinite number of solutions, the sought solution can not be identified.
- In a next step, a support set with exactly the same length $\tilde{K}$ as measurements M is taken, which leads to a well-determined, reduced system of equations. This can easily be solved with Gaussian elimination. But it points out, that every possible support set then leads to a single solution, which makes it also impossible to decide on one, even though the correct solution is in this group of possible solutions.
- The last possibility is considering a support set with length $\tilde{K}$ below the number of measurements M, where the resulting reduced system of linear equations is overdetermined and regardless of which support set is assumed, the equations generally conflict with each other, except the one case, when the support set with $\tilde{K}$ entries

[3] as the Gaussian elimination algorithm is
[4] This is also the number of equations

coincides with the support set of the original vector $\boldsymbol{x}$ with the length K. Then enough equations are redundant and the equation is solvable with just exact one unique solution, which then coincides with the original vector $\boldsymbol{x}$.

This leads to a very important fact, that compressive sensing is only possible if

$$K < M \ . \tag{2.3}$$

It outlines the field of applications where compressive sensing probably can be used. However it does not ensure success of reconstruction, but rather the opposite makes it impossible. [2, cf. p. 1-2]

There are three different approaches to recover the correct coefficient vector $\boldsymbol{x}$, which are introduced in the following section. Let $\boldsymbol{x}$ have K nonzero coefficients, thus called $\boldsymbol{x}$ being K-*sparse*, $\boldsymbol{y} = \boldsymbol{B} \cdot \boldsymbol{x}$ be the compressive sampled record of it and $\boldsymbol{B}$ a measurement matrix, which shall, at the moment, not be described in detail.

2.2.1. ℓ_0-minimization

ℓ_0-minimization is probably the most natural way to solve such a system in theory and its mathematical representation is denoted

$$\min \|\boldsymbol{x}^*\|_0 \quad \text{subject to} \quad \boldsymbol{y} = \boldsymbol{\Phi}^T \cdot \boldsymbol{A} \cdot \boldsymbol{x^*} \ . \tag{2.4}$$

As pointed out in *2.1 Fundamentals of Compressive Sensing*, the sought solution to the reconstruction problem, is the sparsest one of the solution space. The simple approach then is, assuming $\boldsymbol{x}^*$ to be of the lowest reasonable sparsity, which is one and try to solve the reduced equation $\boldsymbol{y} = \boldsymbol{A}_\mathrm{S} \cdot \boldsymbol{x}^*_\mathrm{S}$, by testing all possible support sets $\boldsymbol{x}^*_\mathrm{S}$ of this length. If a solution is found, the algorithm can be stopped, otherwise the assumed sparsity $\tilde{K}$ must be increased and the equation has to be tested with all new possible support sets until the solution is found or $\tilde{K}$ must be increased again.

Unfortunately ℓ_0-minimization leads to uneconomical high computing time, increasing exponential with the problem size, which already makes it more or less useless for applications with slightly longer signal vectors[5] or real time applications. It can be shown that the considered problem belongs to the class of *NP-hard*[6] problems.[26, cf. p. 1]

Although this technique points out as being more or less useless in practical situations, it is very helpful to understand the fundamental concept of compressive sensing. Anticipating the following sections, the area where compressive sensing can be used, could be enormously increased if in future the group of NP-hard problems could be solved efficiently, which is at the moment unthinkable, but it should not be ruled out.[8, cf. p. 4 - 16], [22, cf. p. 3-7]

[5] This maximum length depends on the computing machine, but most real applications suffer from too long signal vectors

[6] NP denotes a computational complexity class and defines the required computational cost, to solve such problems. This cost increases exponentially with the size of the problem, which makes it often impractical.

2.2.2. ℓ_1-minimization

The second technique is called *ℓ_1-minimization* and denoted

$$\min \|\boldsymbol{x}^*\|_1 \quad \text{subject to} \quad \boldsymbol{y} = \boldsymbol{\Phi}^T \cdot \boldsymbol{A} \cdot \boldsymbol{x}^* \ . \tag{2.5}$$

It is also called *Convex Relaxation* or *Basis Pursuit*. To explain it, a two-dimensional geometric interpretation is helpful[7].

Prior to this, the idea of *ℓ_p-balls* has to be introduced. Such ℓ_p-balls contain all vectors of the same ℓ_p-norm. Figure 2.1 shows balls of different norms (and quasinorms) in $\mathbb{R}^2$.

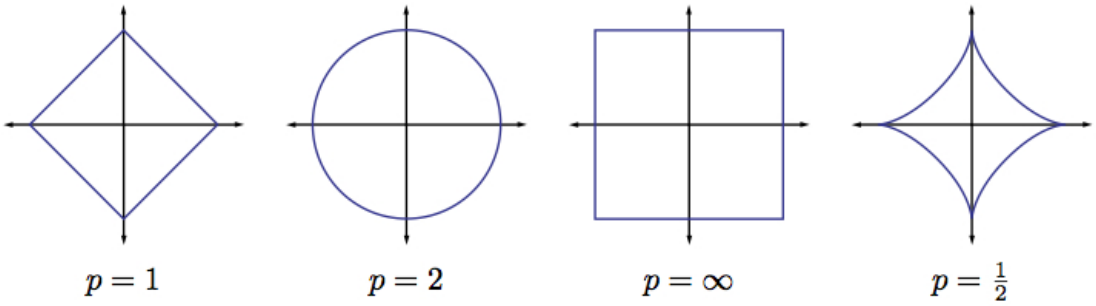


Figure 2.1.: Illustration of ℓ_p-balls with $p = 1, 2, \infty, \frac{1}{2}$ in $\mathbb{R}^2$.

Now consider a two-dimensional vector $\hat{\boldsymbol{x}} = [0, x_2]^T$, which is 1-sparse. That means it has one nonzero entry and lies on the nonvanishing axis and the line of solutions. Minimizing the ℓ_p-norm subject to a linear constraint, can in the two-dimensional case, be imagined by monotonously increasing the size of an ℓ_p-ball until touching the line of solutions A, as can be seen in Figure 2.2. Though the line of all solutions stays the same in all pictures, minimization is only successfully finding the sparse $\hat{\boldsymbol{x}}$ if $p \leq 1$. But note, the line of all solutions offers two 1-sparse solutions, which seems to conflict with earlier statements, but comes due to the violation of the $K < M$ limit.

It is very abstract when trying to transfer this imagination of the unit balls to the case where $p = 0$. But mathematically it works, and the intersection of the ℓ_0-ball delivers the sparsest solution. And although it is not any longer imaginable at more than a few dimensions, it points out, that the math behind this geometry can be transferred to all of them. While ℓ_0-minimization leads to NP-hard problems[8], the ℓ_1-minimization can be handled by efficient techniques, being not NP-hard. These techniques are *Linear Program* in the real-valued case and *Second Order Cone Program* in the complex-valued case. There exist efficient solvers in both cases. [23, cf. p. 7]

[7] Notice, compressed sensing in two-dimensional case is not possible, because (2.3) limits the sparsity K to be smaller than the height of the measurement matrix M, which would lead to a 0-sparse coefficient vector $\boldsymbol{x}$, then being the null vector. So the following example shall just illustrate the problem in an imaginable and presentable way.

[8] as the usage of all $p \in [0; 1)$ would

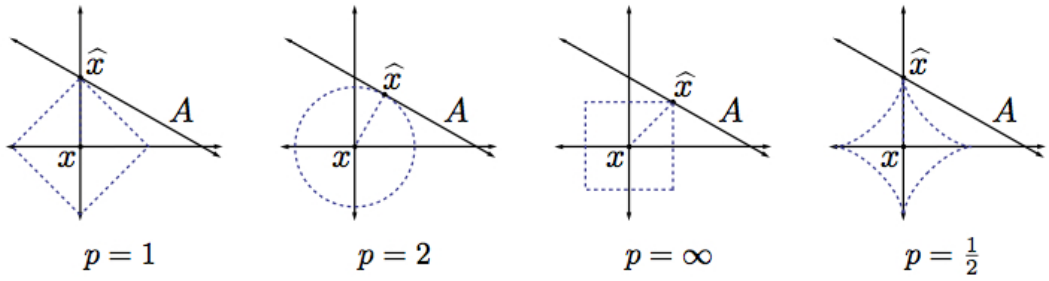


Figure 2.2.: Illustration of the intersection of ℓ_p-balls with the line of solutions, finding the solution with the smallest ℓ_p-norm.

Unfortunately using the ℓ_1-minimization leads to a problem. Consider a system of linear equations in three dimensions, where all possible solutions lie on a straight line[9] intersecting only one axis, with the intersection point at a large value. The intersection of the ℓ_1-ball and the line of solutions, with the smallest ℓ_1-norm, will then not be the sparsest solution, which would be the case when using the ℓ_0-ball. In other words, the delivered solution of the ℓ_1-minimization problem just not coincides with the original vector, it is the wrong solution. So if intending to use this method, it has to be ensured that its solution correspond to the original vector $\boldsymbol{x}$.

Therefore the *Null Space Property (NSP)* was invented. If a matrix $\boldsymbol{B}$ satisfies the null space property of the order K, exact recovery of all K-sparse signals $\boldsymbol{x}$ is guaranteed and the solution of the ℓ_1-minimization coincides with the solution of the ℓ_0-minimization. But proving the satisfaction of this property is in general impossible, which is why there are other measure functions which ensure $\boldsymbol{B}$ to satisfy the null space property of a special order. These are introduced in 2.3 *Reconstruction Guarantees*. For the moment please notice, that replacing the ℓ_0-minimization problem (2.4) by the ℓ_1-minimization problem (2.5) is often possible, but limits the area where reconstruction is successful. [8, cf. p. 28 - 31]

2.2.3. Greedy Methods

In contrast to that, a second type of practical algorithms exists, the so called greedy methods, which are algorithms based on a starting vector and then stepwise choosing the most promising vector to start with in a following iteration step. However these algorithms use other command variables, as the correlation is one, are easy to implement and mostly shorten the required computational time tremendously. But common literature implies them to suffer from an even worse reconstruction guarantee. It is differentiated between uniform recovery and nonuniform recovery, which means recovery of all signals

[9] The shape of the solution space is not limited to lines, but well suited to show the problem with this example.

or rather a given signal, by a fixed measurement matrix [1, cf. p. 1-2].

The development of greedy algorithms is far away from the advanced state of basis pursuit. Nevertheless there are hundreds of papers, concerning greedy methods (see a short list next paragraph or [22, p. 7] for more literature about greedy methods) or the comparison between greedy and basis pursuit (see [21, 27]). Not a few claim greedy algorithms to be mostly as good as the others in practical cases, and others who respond, arguing from a theoretical point of view, referring to the lack of uniform recovery guarantee of greedy methods. Chapter 5 *Analysis of Phase Transition Diagrams* tries to find an answer to this question by comparing greedy and basis pursuit solvers in the scope of several simulations.

Though talking about greedy, it should also be mentioned that there are many different greedy algorithms, such as *Matching Pursuit (MP)*, *Orthogonal Matching Pursuit (OMP)* or *Iteratively Reweighted Least Squares (IRWLS)*, to name a few. All greedy algorithms have their pros and cons and differ, for instance, in the manner of speed, efficiency, accuracy, success rate or the command variable. More information can be found in literature [7, 14, 19] and chapter (4.2.6) *Compressive Sensing File CompressiveSensingClass.m*, where a concrete implementation of all of them is introduced. [17, cf. p. 1-2 and p. 15]

2.3. Reconstruction Guarantees

Handling the reconstruction step of compressive sensing is far from the simplicity of Nyquist sampling, mainly due to its probabilistic character. While section (2.2) *Recovery of Sparse Signals* explains different reconstruction methods, this section introduces some basic ideas, which guarantee success of reconstruction.

First, it is important to clarify the needs of different applications. While reconstruction guarantee is mandatory in some, others are repeatable or insensitive to mistakes. Hence it can be sufficient if the probability of error is unequal to zero, but very low. The before introduced null space property seems pretty much cut out to the former mentioned applications. But as also noted, it is NP-hard to prove the satisfaction of the NSP to a given measurement matrix, which makes it rather useless in practical cases. An alternative is the *Restricted Isometry Property (RIP)*. If $\boldsymbol{B}$ satisfies the restricted isometry property of the order K, the satisfaction of the null space property of the order K is guaranteed. But to prove a concrete matrix $\boldsymbol{B}$ on satisfaction of the RIP is still even hard. It belongs to the class of NP-hard problems, too [26, cf. p. 6-7]. And what might be also very important, the other way round, suggesting $\boldsymbol{B}$ satisfying the restricted isometry property, due to given satisfaction of the null space property is not possible. By the way the satisfaction of the restricted isometry property is a problem also belonging to the class of NP-hard problems.

This makes it impractical to use these two properties to measure the quality of a measurement matrix, which is why there is a third one, making a suggestion on the possibility of exact reconstruction. It is the *Coherence* of a matrix, which is sometimes also called *Self Coherence* to emphasize not speaking about the coherence be-

tween two different matrices. Assume a measurement matrix $\boldsymbol{B}$ with normalized columns $\|\boldsymbol{b}_j\|_2 = 1,\ j \in \{1, 2, \dots N\}$. The coherence μ is calculated by

$$\mu \ = \max_{j \neq k} |\langle \boldsymbol{a}_j, \boldsymbol{a}_k \rangle| \tag{2.6}$$

and quantifies the pairwise similarity of its columns.

The coherence and the restricted isometry property can be linked together, which yields into

$$(2K - 1) \cdot \mu < 1 \tag{2.7}$$

and guarantees the satisfaction of the restricted isometry property and thereby also exact reconstruction for K-sparse signals. But it must be emphasized, that reconstruction can also be guaranteed for larger coherence values, due to the fact that exact reconstruction can be guaranteed by the much weaker null space property, which can be satisfied by a matrix not satisfying the restricted isometry property. On the other side, the coherence of a matrix cannot be infinitesimal small. The *Welch Bound* provides the following lower bound of the coherence of any $M \times N$ matrix

$$\mu \geq \sqrt{\frac{N - M}{M \cdot (N - 1)}} \tag{2.8}$$

[22, cf. p. 13-15].

These properties emphasize the importance of the structure of the matrix $\boldsymbol{B}$, whose design can be denoted as a key problem in the scientific field of compressive sensing. But until this point it was merely introduced as black boxes and certainly its design has to be elaborated more detailed. Often taking a deterministic matrix is the first assumption. It shows however, that the use of such matrices does not lead to good recovery results. The use of *Random Matrices* stays in contrast to this. It can be shown, that these matrices satisfy the restricted isometry property with high probability, which makes the RIP this important in the field of compressive sensing. The use of random matrices makes it possible to state clearly better recovery guarantees, than in the former mentioned case with deterministic matrices. As far as this does not change, using random matrices is the better choice, as *Bernoulli* or *Gaussian* matrices are of this type. The coefficients b_{ij} of the Bernoulli matrix are of the value $\pm 1/\sqrt{M}$. In the Gaussian case they are independent and identical distributed from a continuous normal distribution with the expected value zero and the variance $1/M$. [23, cf. p. 13-16]

However real cases mostly suffer from the fact, that the sampling step cannot be modeled with completely random matrices. Therefore a third sort of matrices must be established, the *Partial Random Matrices*. One the best known matrices of this category are partial random Fourier matrices, which shall be introduced here as representatives of this group. However this group is much larger and is still growing. The partial random Fourier matrices are based on the *Discrete Fourier Matrix*, being $\boldsymbol{A} \in \mathbb{C}^{N \times N}$ and a *Random Selection Matrix* being $\boldsymbol{\Phi} \in \mathbb{R}^{M \times N}$. While $\boldsymbol{A}$ is fixed by a given N, $\boldsymbol{\Phi}$ has M coefficients of the value one, randomly distributed over its columns, without any column

containing more than one nonzero value. The rest of them are zero. It has the property of cutting out random rows of the transformation matrix $\boldsymbol{A}$, which is the fact it is named after. As long as considering a real-valued case the Fourier transformation matrix can be substituted with the *Discrete Cosine Transformation Matrix*, having similar properties. As the former mentioned random matrices, the partial random ones also satisfy the restricted isometry property of high orders with high probability. [22, cf. p. 16-25]

When talking about reconstruction it has to be defined what successful reconstruction means. It is successful if $\boldsymbol{x} = \boldsymbol{x}^*$. But using digital processors can lead to rounding errors due to their limited computational accuracy. Due to this, the reconstruction is also successful if an allowed deviation $\epsilon = |\boldsymbol{x} - \boldsymbol{x}^*|$ is not exceeded. This deviation is later called *Specified Accuracy*. In the noiseless case another approach can be used. As known from 2.1 *Fundamentals of Compressive Sensing*, the exact solution can be easily derived with Gaussian elimination if the correct support set is assumed, hence it is sufficient to compare the support sets of the solution and the original vector. If these coincide, the vectors will also coincide. Again the computational accuracy has to be considered, which leads into a permitted deviation ϵ where values are treated as zeros. The noisy case is a bit more complicated. Special measure functions, measuring the similarity of two vectors, are required.

2.4. Summary

Compressive sensing contains two parts, sampling of a signal and its recovery. The sampling step can mathematically be described by $\boldsymbol{y} = \boldsymbol{\Phi} \cdot \boldsymbol{A} \cdot \boldsymbol{x}$, while recovery is successful if, on the basis of $\boldsymbol{y}$, a vector $\boldsymbol{x}^*$ can be found, which is equal to $\boldsymbol{x}$. The use of this technique is limited to the area where the sparsity K is smaller than the undersampling rate M.

There exist several solution approaches. ℓ_0-minimization is of importance in theoretical considerations, but with the two methods greedy and ℓ_1-minimization, there are practical ones available. Unfortunately these methods further limit the area of successful reconstruction. They are able to recover K-sparse signals if the NSP of the order K is satisfied by the measurement matrix. But these are theoretical considerations, too, due to the NP-hardness of its examination.

To guarantee success in concrete applications further methods have to be found, which are introduced in the following chapter. Until this point it could be shown that the use of random and partial random measurement matrices leads to good results, because of them satisfying the RIP. This shows the importance of the choice of suitable measurement matrices $\boldsymbol{B}$, however, their shape can be limited due to practical reasons.

3. Phase Transition Diagrams

The use of phase transition diagrams is widely spread in science. In the field of compressive sensing these diagrams provide information about the success of compressive sensing applications under certain conditions. They are further utilized to find suitable settings for concrete applications or to make performance statements. The diagrams were, in the field of compressive sensing, introduced by Donoho and Tanner in 2009, who describe their framework and results in [11] and [12].

The following chapter sums up their most important ideas, beginning with an explanation of the parameters, variables and the statements, such diagrams can make. The further sections show some computational and theoretical results from the mentioned papers, regarding phase transition diagrams.

3.1. Composition of Phase Transition Diagrams

Phase transition diagrams are two-dimensional figures. Their y-axis is denoted $\rho = {}^{K}/_{M}$ and represents the *Sparsity* of $\boldsymbol{x}$. It is sometimes a bit more precisely called *Density* of $\boldsymbol{x}$, because it describes the nonzero elements to sampled elements ratio. In general linguistic usage some would speak of a high sparsity if the number of nonzero elements is low, which would lead to low values of ρ and is a bit confusing since in context with PTDs generally high values of ρ are meant. The same is true for the x-axis on the other side. It is denoted $\delta = {}^{M}/_{N}$ and called *Undersampling Rate*, which is the rate of sampled elements in contrast to the length of the vector $\boldsymbol{x}$. As before, speaking of high undersampling rates often means high values of δ, which stays in contrast to the general linguistic use of undersampling rate. Hence in this thesis always concrete values or explanatory notes are given. Both axes are normalized and as known from 2.3 *Reconstruction Guarantees*, the area of interest lies in-between $0 < \delta, \rho < 1$, which is the area were $K < M$ and the sampling rate is below the rate, predicted by the Shannon Nyquist Sampling Theorem. Hence such diagrams are mostly limited to that area.

Every point in this diagram represents concrete settings and presents the probability of success of a compressive sensing reconstruction step, depending on the parameters K, M and N. The probability of success is also called *Success Rate (CS)* and the points, defined by K, M and N are the *Estimation Points*. There is no universal analytic function to derive these probabilities until now, which is why these have to be estimated. This can be achieved by doing a *Monte Carlo* experiment, which means generating independent and random *Scenarios*, under given conditions. Counting the relative frequency of successes, estimates its probability, whereby the accuracy rises with an increasing number of scenarios. Doing such a Monte Carlo experiment for every undersampling rate M and

sparsity K, leads to a continuous diagram, where every possible point is estimated. The experiment can indeed often be reduced to a few curves, representing points of constant success rates.

Anticipating the following chapters, PTDs in compressive sensing have a very common structure. They show a relatively sharp transition zone between regions where recovery is crowned with success and such where it is not. The position of this transition zone from one to the other region can be described by a curve, representing 50% probability of success. The width of this zone can be outlined by the 10% and the 90% curves and is usually very thin. This region between, is called *Phase Transition Zone* and represents the area where the success rate has its maximum slope. It divides the diagram into a subjacent sector where the success rate is quite high, and an overlying sector marking the area with low success rates. Figure 3.1 shows a continuous example, where the three regions can easily be identified.

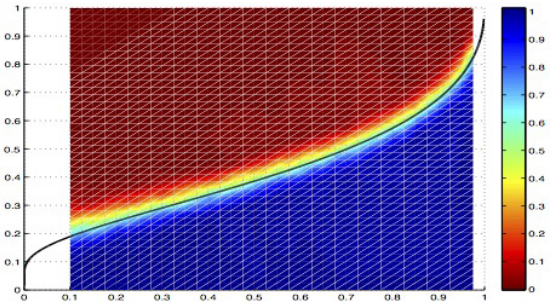


Figure 3.1.: Example of a continuous phase transition diagram, showing the success rate colored, the x-axis denotes $\rho = {}^{K}/_{M}$ and the y-axis $\delta = {}^{M}/_{N}$. The three mentioned areas are shown. The one of low success rates is red, the one of high success rates blue and the thin phase transition zone lies in-between them.

Some parameters have to be defined and to stay constant during the whole experiment, these are the length of the signal N, the measurement matrix $\boldsymbol{B}$, or if not deterministic, the type of $\boldsymbol{\Phi}$ and $\boldsymbol{A}$, the specified accuracy and a concrete implementation of an algorithm, which is used to reconstruct the original vector.

Furthermore some technical settings have to be set. The *Horizontal Step Size* dictates the points on the x-axis to be evaluated[10]. Setting it to one, which means measuring at all integer values between $0 < N$ and leads to continuous diagrams, is very time-consuming and often not practicable. Larger values can be used if a reduction of the diagram to a few curves is sufficient. The intermediate values have to be interpolated. This can be done since the curves with constant success rate are assumed as piecewise continuous and monotonous. The calculation of the boundary point at $\rho = 0$ can be skipped and directly set to zero. It is sometimes connected with the rest of the curves. But bear in mind that the deviation between the real and the interpolated curve is at its maximum in this region, due to the high gradient values of the success rate. If an exact estimation of the success rate in this region is required, the horizontal step size has to be adjusted and the points with low values of δ have separately to be simulated. At the boundary point $\rho = 1$, at the other side, which represents the standard sampling point,

[10]These points do not have to be equidistant. Sometimes it may be more clever to use unequally distributed points.

property	**value**
N	1600
h./v. step size	160/1
problem sets	200
solving algorithm	MOSEK
specified accuracy	$1e-6$
$\boldsymbol{A}$	DCT
$\boldsymbol{\Phi}$	selection matrix
distribution type of $\boldsymbol{x}$	standard Gaussian
Abbr.: h.: horizontal, v.: vertical	

Table 3.1.: Listing of the parameters being used in Figure 3.2.

often the same is done, because it is known to be manageable with success rate of one, by using standard sampling technique. But connecting this point with the rest of the lines is critical, because compressive sensing solvers may have problems there. The behavior in the vertical direction is indeed continuous and monotonous too, but the gradient is very high at the thin transition zone. Hence this is the most interesting area, the *Vertical Step Size*, which determines the increase of the sparsity K, should be one, to gain the largest possible preciseness. Finally the number of scenarios per point, called *Problem Sets* or *Problem Instances*, must be set. The higher its value is chosen, the preciser can the success rate be estimated. These three parameters, horizontal and vertical step size and problem sets, determine the estimation points and the number of scenarios to be done. This number can be very high and nevertheless leads to high computational costs.

All scenarios must be done at constant conditions and settings, but with independently and randomly drawn values of the sensing matrices and signal vectors. The original vector $\boldsymbol{x}$ must not presented to the solving algorithm, but is used to check on success of the recovery in the following step. In the end, the success rate at all estimation points is calculated by dividing the sum of successes by the number of simulated problem sets. The curves, lying on the points with selected success rates, then are plotted in the suggested diagram. [12, cf. p. 915-917] Figure 4.2 shows the principle of constructing such a phase transition diagram from a grid of single success rate estimations.

3.2. Computational Results in Literature

Donoho and Tanner did many experiments with the aim of creating phase transition diagrams. Figure 3.2 shows a typical result, which was created with the settings, listed in Table 3.1. They used partial random cosine measurement matrices and the amplitudes of the nonzero elements of the coefficient vectors were drawn from a standard Gaussian distribution. The signal has $N = 1600$ coefficients, they measured in nine equal steps with horizontal step size of 160 and with 200 problem instances per estimation point. The used solving algorithm was *MOSEK*. It is a software package, capable of solving mathematical optimization problems, hence also capable of solving ℓ_1-minimization problems and is

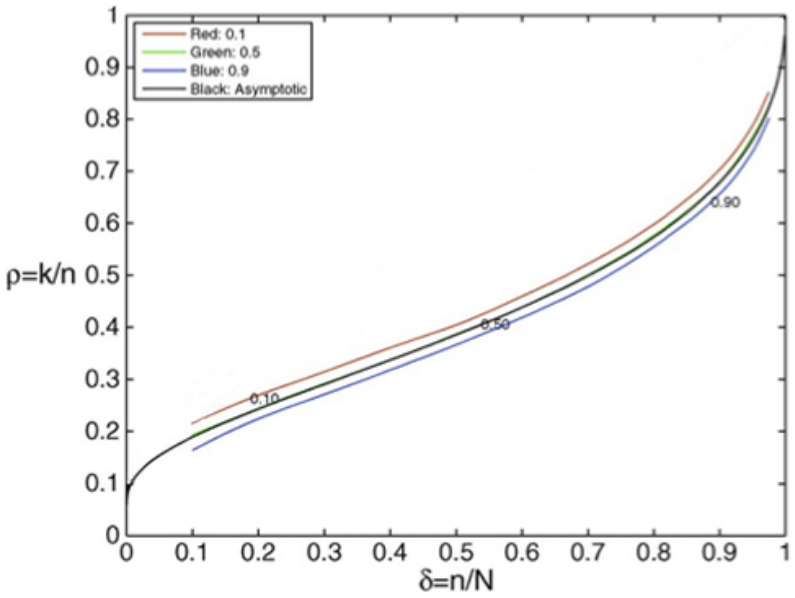


Figure 3.2.: Phase Transition Diagram of Donoho and Tanner.

available for *MATLAB*, but subject to a professional license. But the authors claim their results to be reproducible with free alternatives, for example SDPT3.

The colored lines frame the transition zone, where the success rate shrinks from 90% (blue line) to 10% (red line). Inside the transition zone, the curve representing 50% (green line) is plotted. The overlaid asymptotic line (black) is a calculational result, which will be discussed in 3.3 *Theoretical Results in Literature*, below this section.

This experiment points out a few important characteristics. The area where exact recovery is possible with high probability, is limited and as predicted in 2.2 *Recovery of Sparse Signals*, widely smaller than the area where $K < M$, which would be the whole diagram. This does not surprise, because the used solvers belong to the group of basis pursuit. It shows furthermore, that the transition zone is very small and shrinking with the increasing number of measurements M. The authors claim the width of the zone to be proportional to $1/\sqrt{M}$.

A second interesting phase transition diagram is shown in Figure 3.3. The diagram shows the 50% curves of multiple experiments, all done with the same settings as in Figure 3.2, but different measurement matrices $\boldsymbol{B}$, all from the group of (partial) random matrices but drawn from different distribution types.

As can easily be seen, all curves merge to one, which shows, that the choice of the measurement matrix, seems to have no effect on the success of reconstruction if choosing from the group of (partial) random matrices. This is interesting due to their different structure and coherence. But consider, this measurement was done without the existence of noise and solved by a basis pursuit algorithm. Chapter 5 *Analysis of Phase Transition Diagrams* reproduces this experiment and extends it to the case of greedy solvers.

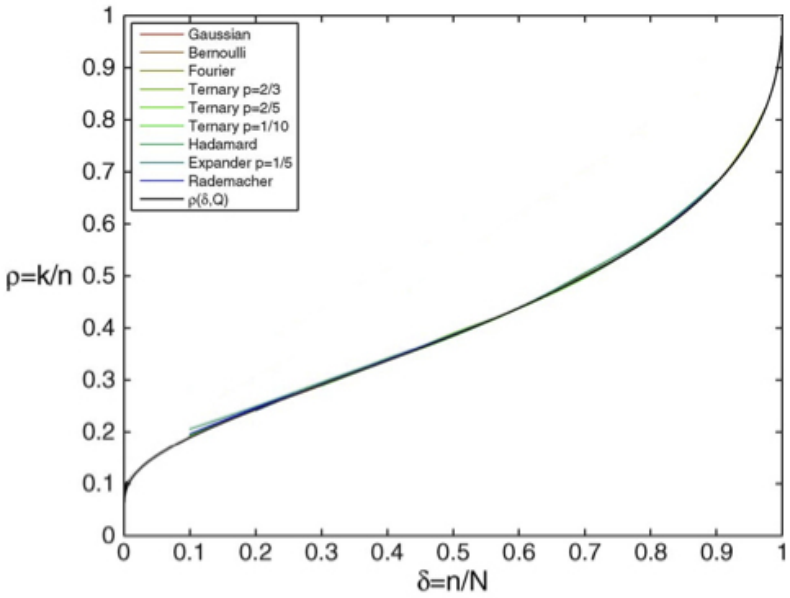


Figure 3.3.: Phase transition diagram showing the dependency on $\boldsymbol{B}$.

A last observation appears when studying Figure 3.4, which shows phase transition curves with the success rate 99%. The interesting part is, all shown curves were measured with identical settings, as those in Figure 3.2, but with different signal lengths N. A tendency can easily be observed. The greater N, the better the reconstruction results. This is also part of a simulation in Chapter 5 *Analysis of Phase Transition Diagrams* and detailed studied in Figure 5.3. [11, 12]

3.3. Theoretical Results in Literature

From a theoretical point of view, it is interesting to find rules, predicting success of compressive sensing. This bounds, of course, should be as precise as possible. But the probabilistic nature of compressive sensing poses this as a problem. The main approach to derive such bounds is the restricted isometry property. These bounds typically are of easy mathematical form, but very weak. There can many of them be found in the literature, depending on the type of recovery guarantee (nonuniform or uniform), the type of solving strategy and the type of the used measurement matrices. These bounds typically are of a simple mathematical form as (3.1) is an example.

$$M \geq CK \ln\left(\frac{N}{K}\right) \tag{3.1}$$

Thereby C is a universal constant, with $C > 0$.

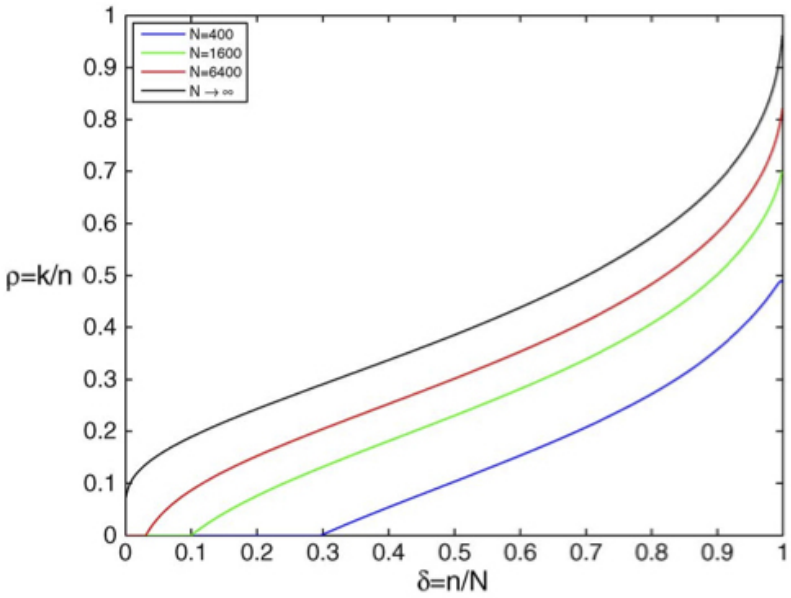


Figure 3.4.: Comparison of phase transition diagrams where the signal length N is varied, showing 99% curves.

The bound (3.1) can be found in literature and guarantees uniform recovery under the use of random measurement matrices. But as mentioned before, these bounds are very weak and primarily not necessary for successful reconstruction, due to the restricted isometry property to sufficient, but not necessary. Hence this bound is useless for performance statements, but it most notably shows, that the number of required measurements mainly depends on the sparsity K and scales proportional to the logarithm of the ratio N by K. [22, cf. p. 15-16] Another simple approach is the coherence of the measurement matrix. It can be shown, that exact recovery of all K-sparse signals $\boldsymbol{x}$ is ensured if

$$(2K - 1)\mu < 1 \tag{3.2}$$

This result is even valid to some greedy methods, but in the end, it is very weak, too. [27, cf. p. 8-9]

Both bounds have to be seen asymptotical, which means, that they are well suited to lead off dependencies on the different parameters. It is possible to infer so called *Scaling Laws* from them. But they are neither strong nor precise bounds, predicting whether a compressive sensing application is successful or not.

A third and in contrast to the former ones, very sharp bound, is proposed by Donoho and Tanner. It is drawn as black curve in the figures 3.2, 3.3 and 3.4 and coincides, as can be seen, with the 50% curves. But their calculation is also NP-hard, which makes them impractical in daily use. The derivation requires some more math, but the idea is to link the success rate with the face counts of more dimensional geometric objects with the face counts of their projected ones. [12, cf. p. 917-919]

Finally, concrete performance predictions can solely be done with the help of phase transition diagrams. They enable tight performance statements and benefit from their relatively simplicity.

3.4. Summary

It could be pointed out, that using phase transition diagrams is the sole, relatively easy to use method to gain precise performance statements to concrete compressive sensing applications. This emphasizes their importance in the scientific field of compressive sensing. Thus the almost absence of such diagrams in literature is remarkable, but understandable if considering the big number of influencing factors. On the other side the scaling laws are also very important and enable rough estimations. But their weakness and the complexity of the bounds predicted by Donoho and Tanner make these bounds useless for concrete performance statements.

4. Calculation of Phase Transition Diagrams

Calculating phase transition diagrams requires

$$P = T \cdot \frac{N \cdot (N-1)}{2} \tag{4.1}$$

single reconstruction steps, whereby T denotes the number of problem sets. The complexity of a reconstruction step of linear program is $O(N^3)$. This leads to a combined complexity of $O(T \cdot N^5)$, which is very high and time-consuming if N is large. [2, cf. p. 3]. This complexity was the main motivation to speed up the algorithm. As long as the phase transition is reduced to curves with specific success rates, the calculation can, without a lack of precision and quality, enormously be accelerated, by an algorithm, developed in the context of this thesis. It is implemented in the program *PTD Builder*, which is part of the work and in the current version, supports noiseless simulations with real-valued signal vectors. The use of complex-valued signals is potentially possible, but not supported by all solving algorithms (see 4.2.6 *Compressive Sensing File CompressiveSensingClass.m*). Future versions of the program shall support more solvers to handle complex-valued cases and offer the possibility of adding noise.

The following chapter first describes the newly developed algorithm in detail, while further sections treat the program PTD Builder and contain an overview, a quick start guide, detailed information about its interfaces, variables and parameters, troubleshooting and an explanation of a further used algorithm.

4.1. Description of the new Algorithm

As illustrated in Figure 4.2 creating a phase transition diagram starts with a two-dimensional array, containing the success rate of each estimation point, defined by K and M. The prior introduced algorithm of Donoho and Tanner, simply runs through the array and stores the number of successes per cell. As mentioned in the introduction, this requires a computational cost of $O(T \cdot N^5)$ and is very time-consuming. The new algorithm verifies its position in the grid after each scenario, instead of simulating every single estimation point. The main parts of the algorithm are printed as pseudo code in Algorithm 1 *Calculation of PTDs*. A verbal description of its ideas follows here.

The algorithm requires at least two arrays. The first contains the number of simulated scenarios per estimation point and the second the number of successes per estimation

point during these simulations. It begins in the lower left corner at the first value of M, defined by the horizontal step size and K is set to 1. After every simulated scenario, the algorithm updates its arrays and calculates the success rate at the current estimation point. The following evaluation begins with a check on the number of scenarios at this point. If it is greater or equal than the desired number, the success rate is checked. If its value is greater or equal the desired probability, too, the sought curve can be printed at $\delta = {}^{M}/_{N}$ and has the value $\rho = {}^{K}/_{M}$. If its value is below the desired probability and the current sparsity $K == 1$ the sought curve can be printed at $\delta = {}^{M}/_{N}$, but has the value $\rho = 0$. Otherwise, if the number of simulated scenarios is below the desired number, the program moves in vertical direction, by adjusting the sparsity K. If the Success Rate (SR) is above the desired probability of success and below the number of measurements M, it is increased by one. Else if the SR is below the desired probability of success and the sparsity $K > 1$, K is decreased by one.

This permanent evaluation leads to a rough convergence of the current estimation point, to the point of the sought probability of success. The estimation point swings between two neighbor points, which frame the real sought point. Finally the point with the highest sparsity K, guaranteeing the sought probability of success will be found by the algorithm. If this point is found, these steps are repeated with the following undersampling rates M, which were selected to be estimated by the horizontal step size parameter. The sparsity does not need to be reset to one, because of the assumption of the curves being monotonically increasing and the permanent evaluation of the position in the grid.

The main improvement of the algorithm is its speed. This comes on the one hand from the skipping of the measurements above, and on the other hand from the incompleteness of the measurements below the desired Bounds. The advantage in speed can not be measured exactly, because it depends on randomness and the concrete structure of the curve, because the computational time of a reconstruction step depends on the position in the diagram. But a rough count of the number of required reconstruction steps shows the improvement. The minimal number of steps would be $T \cdot N$ if only the points on the curve are simulated. Simulating all points would require P steps. The new algorithm requires much less reconstruction steps, which shall be shown in a typical example. The creation of a phase transition diagram with $N = 400$, 200 problem sets and nine equally spaced estimation points, shows the following. While the primal algorithm requires 360000 reconstruction steps, PTD Builder was in need of just 5098 reconstruction steps, which corresponds to about $1,4\%$ and $3 \cdot T \cdot N$ reconstruction steps. More precise measurements could not be done in time, but the actually saved computational time, may be noticeably smaller, due to a more complex evaluation and much more memory accesses.

Unfortunately the measurement does not provide curves, tracing other success rates. However, previous results can easily be integrated into new experiments, speeding up the additional required computational time. This compensates the incomplete measurement problem.

The following shows the pseudo code of the algorithm. It contains the whole loop, controlling the complete simulation. *Bounds* hereby denotes the desired probability of success, which shall be reflected by the curve. The array $TransLine$ contains the estimated values of $\rho = K/M$ and can then be printed.

Algorithm 1 Calculation of PTDs

Input: *N, ProblemSets, HorizontalDistance, Bounds*

Set $K = 1$ and $M = HorizontalDistance$.
Allocate *CountMap* and *SuccessMap* and set cells to zero.
while $M < N$ **do**
    SolveNewCSScenario(*K, M*).
    $CountMap[K, M] = CountMap[K, M] + 1$
    if CheckOnSuccess == *true* **then**
        $SuccessMap[K, M] = SuccessMap[K, M] + 1$
    if $CountMap[K, M] >= ProblemSets$ &&
      ($SuccessMap[K, M]$ / $CountMap[K, M] >= Bounds$ || $K == 1$) **then**
        if $K == 1$ &&
          $SuccessMap[K, M]$ / $CountMap[K, M] >= Bounds$ **then**
            $TransLine[M] = 0$
        else
            $TransLine[M] = K$ / M
        Increase M as specified by $HorizontalDistance$.
    else if $SuccessMap[K, M]$ / $CountMap[K, M] >= Bounds$ **then**
        if $K < M$ **then**
            $K = K + 1$
    else if $SuccessMap[K, M]$ / $CountMap[K, M] < Bounds$ && $K > 1$ **then**
        $K = K - 1$

Output: $SuccessMap$, $CountMap$, $TransLine$

4.2. Description of PTD Builder, its Interfaces and Usage

The program PTD Builder enclosed to this thesis is written in MATLAB. It was designed for usage within unix systems and was tested with the current MATLAB versions (R2011 and above) on Mac OS X 10.8.5 and CentOS 6.6.

It is able to create phase transition diagrams, measures additional information as the required computational time and stores its results in a file. Furthermore all generated data can easily be evaluated and plotted in two- or three-dimensional diagrams. It thereby supports a few solving algorithms already, representing the typical solution approaches to compressive sensing applications. These solvers are integrated in the included compressive sensing class *CompressiveSensingClass.m*, which provides a lot of functions

and could also be used outside PTD Builder. The class allows creation of vectors and matrices, supports different distribution types, deterministic measurement matrices or matrices with concrete coherence values and controls the *MATLAB Random Number Generator (RNG)*. This generator enables the repetition of experiments and facilitates debugging. In the end simulation of many different compressive sensing applications is possible.

4.2.1. Structure and Installation

The program comes in the following structure:

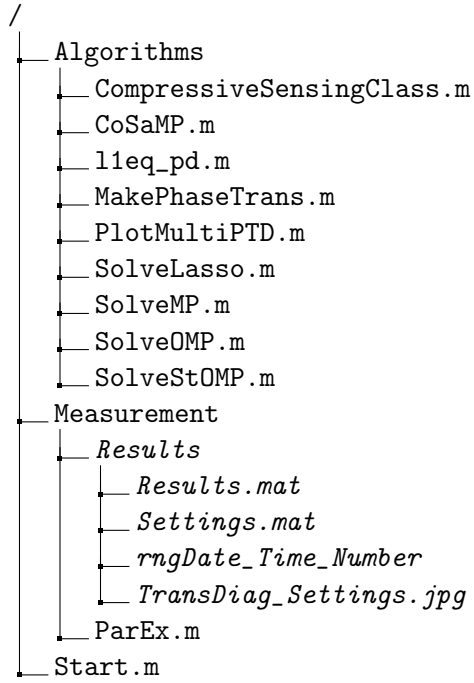


Figure 4.1.: Directory tree of PTD Builder.

The italic written files and directories may be missing, but will be created by the program when running it.

Installing PTD Builder is easy and contains of two steps. First simply copy the directory *PTDBuilder* to your MATLAB current folder. Second the *cvx* program package has to be installed, which could not be integrated, due to complexity and licensing reasons. See [16] or `http://web.cvxr.com/cvx/doc/install.html` for more details and detailed installation instructions.

4.2.2. Quick Start

Using the program is easy. Adjust the parameter file *ParEx.m* and the evaluation file *PlotMultiPTD.m* to set your desired settings, run the *Start.m* file and wait for the PTD to be created[11]. But a few more suggestions shall be given here, while the usage of the particular files itself is described in detail in the corresponding sections.

It is advisable to keep an unchanged original of the program to restore it if some editing went wrong. So create a copy of the folder containing the program before using it. It is further subtle to rename your parameter file, to enable easy identification later. Of course the *Start.m* file has then to be adjusted, too. If you are planning to do several simulations it is clever to create an own directory for each of it. This prevents you from overwriting previous results.

The main file *Start.m* contains two function calls in the second part. The first command calls the *MakePhaseTrans.m* function, which is responsible for the calculation of the data. It requires two input parameters, first the name and location of the parameter file and second the name and location of the directory the data shall be stored. The second one calls the *PlotMultiPTD.m* file, capable of doing a graphical evaluation of your measurement. It requires a directory the output shall be stored in and the destination of the *Results.mat* file(s). Both functions require the parameters to be strings, which can be given as relative paths.

The parameters in the *ParEx.m* file are shortly described in the file itself and in more detail in section 4.2.3 *Parameter File ParEx.m*. The *PlotMultiPTD.m* file has to be adjusted, too. And as before, all parameters are shortly explained in the file and in more detail, in section 4.2.5 *Evaluation File PlotMultiPTD.m*. All changeable parameters are marked and can be found at the beginning of the file. Consider that the required time of a simulation can require a long time and depends on different factors, whereby the chosen parameters and the processor speed are in main focus.

The use of the program is free, as the adjustment and extension is if itself again is published under the same conditions. Please accord your extensions to me and contact me for all questions at `mh.herrmann@gmail.com`.

4.2.3. Parameter File *ParEx.m*

All settings, regarding the calculation of a phase transition diagram, can be done in the parameter file *ParEx.m*. If unchanged, it contains a full example with all possible parameters set to standard values and is separated in three parts. First, a short description of all parameters is given. Second, the changeable part comes and third, a small algorithmic part you should not change is specified. This third part is used for the import of the parameters and creates an *Import.mat*-file, when executed. Unset parameters will be set to their default values during the measurement. Table 4.1 contains all parameters, their defaults and a short description. The particular and extensive explanations can be found in *4.2.6 Compressive Sensing File CompressiveSensingClass.m*.

[11] The CVX package has to be installed. See section 4.2.1 for more information.

Some parameters, just marginally touching compressive sensing itself, shall be discussed here. First, the use of *RandomizerSettings*. Every time the program is used it saves a file *rngActualDate_ Time_ Number.mat*, containing the settings of the random number generator. This enables the user to remeasure a concrete setup. Or if, for instance, results appear strange, the measurement can be redone and be debugged. Unfortunately this function was deficient in earlier versions, which were used to measure some of the presented results. It is nevertheless possible to import these *rng* files, but the mistake results in a different use of the data, when imported. Therefore, exact recalculation of these setups fails in all likelihood, whereat the PTD will look very similar, but the concrete values will slightly differ.

The *Feedback* parameter should be set to zero. If not, it enables all messages from the algorithms. This will cause unnecessary CPU time and displays millions of text lines on your screen. If disabled, the necessary messages are displayed anyway.

The program can reuse earlier measurements by setting the *PreResults* parameter to one. The *Results.mat* file from the earlier calculation must therefore be placed in the new output folder. It will be renamed through the algorithm, by adding the prefix *Pre* and a continuous number, beginning with one. The main loop then uses the earlier measurements as basis and updates it with the new settings, so it should only be used to calculate further transition curves with identical parameter setting but different target success rates. The transition lines will be reordered by increasing success rate and the *Results.mat* file be updated with the new measurement data. If strange results occur or something gone wrong, it can be helpful to slightly increase the problem sets variable.

4.2.4. Main Loop File *MakePhaseTrans.m*

This file controls the whole measurement, organizes the data storage and contains the main loop, whose algorithm is explained detailed in 4.1 *Description of the new Algorithm*. The function expects two parameters, first the destination of the parameter file and second the directory where the output shall be saved after execution. When called, the indicated files and folders are tested on existence and subsequent the parameter file is evaluated and comes under scrutiny. Not or wrong set parameters are tried to be set to standard and are stored in the *Settings.mat* file in the desired output folder after the import.

When all tests are completed, some information is displayed and an instance of the *CompressiveSensingClass* is created. Subsequent the main loop is started and the algorithm controls the simulation and ensures data storage. While it is in progress, four arrays of the size $N \times N$ are created. Every cell of these arrays is preallocated with zero and represents an estimation point[12]. The iterative loop parameters K and M select the active cells, the appropriate data is saved in. The arrays are

- *CountMap*, whose cells contain the number of simulated scenarios per estimation point, and is increased by one at every loop cycle.

[12]Since $K < M$ these arrays are triangular matrices. A double use would reduce the required storage volume in further versions.

name	**type**	**def**	**short description**
N	num	400	Length of the signal vector. Can freely be chosen.
SupportThreshold	num	1e-5	Allowed deviation between the coefficients of $\boldsymbol{x}$ and $\boldsymbol{x}^*$. Can freely be chosen.
RandomizerSettings	str	”	Specifies the name of the file containing the *rng* settings. Set as string or leave a blank string.
AmpDist	str	Gauss	Distributional type of the amplitudes of the coefficient vector. Possible values are: *Gauss*, *RandomPhase*, *UniformPositive*, *Uniform*, *Rademacher*, *Benchmark*.
PhiType	str	Select	Type of the measurement matrix $\boldsymbol{\Phi}$. Possible values are: *Select*, *StrictSelect*, *Gauss*, *Bernoulli*.
AType	str	DCT	Type of the transformation matrix $\boldsymbol{A}$. Possible values are: *DCT*, *DFT*, *eye*.
Solver	str	SDPT3	Specifies the used solver for the reconstruction step. Possible values are: *SDPT3*, *SeDuMi*, *l1_magic*, *STOMP*, *OMP*, *MP*, *LASSO*, *SL0*.
ProblemSets	num	200	Defines the number of scenarios per estimation point. Can freely be chosen.
DHor	num	[40 80 ... 400]	Defines the points of M to be estimated. Can freely be chosen, but must be an array if multiple points shall be evaluated.
Bounds	num	0.9	Defines the target success rate of the printed curves in the PTD. Can freely be chosen between 0 and 1. Designating an array calculates the PTD with all bounds in one step. (ex. [0.1 0.5 0.9]).
Feedback	num	0	Either 0 (off) or 1 (on). Activates all additional notification messages. Recommend: 0.
PreResults	num	0	Either 0 (new measurement) or 1 (use existing Results.mat as basis).
DesCoh	num	0	Adjusts the coherence of $\boldsymbol{B} = \boldsymbol{\Phi} \cdot \boldsymbol{A}$. Can either be 0 (no coherence adjustment) or between (0 , 1], denoting the desired coherence.
CohTresh	num	0.01	Defines the allowed deviation of the coherence and the desired coherence of $\boldsymbol{B}$. Can freely be chosen (but is only active when DesCoh is unequal zero).
Abbr.: num: numeric, str: string, def: default			

Table 4.1.: List of all possible parameters, a short description and the standard values for PTD Builder.

- *SuccessMap*, whose cells contain the number of successful reconstruction steps.
- *CohMap*, whose cells contain the sum of the coherence of all used matrices $\boldsymbol{B}$, per estimation point[13].
- *CPUTimeMap*, whose cells contain the sum of the required CPU time at each estimation point[13].

Entry-wise division of the *SuccessMap* by the *CountMap* results in the success rates at each estimation point. As shown in Algorithm 1 *Calculation of PTDs*, all points representing the desired success rates are stored in a further array *TransLine*, which is one-dimensional, with size N. The sought SR, represented by the transition curve, is set by the *Bounds* parameter. The program allows to derive several transition curves in one calculational step. Therefore simply put the desired bounds in an array. For example [0.1 0.5 0.9].

The parameter *DHor* defines the estimation points and is the equivalent to the horizontal step size parameter. In contrast to earlier versions the parameter must be an array, containing the values of M of all measurement points in an ascending order, which enables non equidistant measurements. It was not renamed to keep consistency between previous and the current version of the program. Equally spaced measurements can easily be set up, by using the `DHor=round(N/10):round(N/10):N-1;` command in the parameter file. All used variables are stored in a file called *Results.mat*, at the end of the simulation.

4.2.5. Evaluation File *PlotMultiPTD.m*

The file *PlotMultiPTD.m* is created for evaluation purposes. It uses the calculation results to produce a phase transition diagram as described in 3 *Phase Transition Diagrams* and is optimized for the needs of this thesis. Two or more parameters are required, first the desired output directory, followed by a set of strings, pointing to result files. Additionally, a few settings have to be specified at the beginning of the file.

- If *ZP* is set to one the success rate at the estimation point at $M = 0$ is set to zero and is used in the interpolation step. This continues the line between $M = 0$ and the first estimation point.
- *OP* does the same as *ZP* for $M = 1$. If the point was simulated, the setting will be ignored and the measured one is used.
- *FileFormat* defines the output format of the phase transition diagram, where all formats, supported by MATLAB, are allowed.
- *FileName* defines the filename of the PTD.

[13] Consider earlier versions of PTD Builder stored the value's average instead of its sum, which is a bit unhandy when reusing the results.

- Specify the transition curves you want to plot via the *DesiredBounds* variable. You can choose single bounds or several ones by specifying an array.
- The parameters *LegParam1* to *LegParam3* control the legend of the plot, where all changeable parameters of the parameter file can be set and displayed, ordered by ascending number.
- The x-axis of the plot is denoted by $\delta = {}^{M}/_{N}$ and the y-axis by $\rho = {}^{K}/_{M}$, which leads to the standard two-dimensional plot if the *dim* variable is set to two. You can create a three-dimensional plot, too, by setting it to three. The third axis can show the average computational time or the average coherence of the used matrices $\boldsymbol{B}$. Therefore *ZParam* has to be set to *CPUTimeMap* or *CohMap*.
  LabelY and *LabelZ* define the caption of the axes.
- The displayed axis intercept can be adjusted with *MinX*, *MaxX*, *MinY*, *MaxY*, *MinZ* and *MaxZ*.

After the adjustment of the parameters, the function imports the measurement data, creates the phase transition diagram and its labels and saves it as *TransDiag_ suffix.FileFormat*, whereby the *suffix* can be replaced at the end of the function.

Bear in mind, that the simulations are already done in the main loop and saved in the *TransLine* variable. The evaluation file just creates the phase transition diagram from this data. The *TransLine* variable contains an entry for all estimated points, which are identified by the evaluation file. Following this, the intermediate points are calculated by *Piecewise Cubic Hermite Interpolation* technique, all real estimated points are marked with different red markers.

Figure 4.2 visualizes the steps of the creation of a phase transition diagram from a cluster of simulations, done in the main loop and the evaluation file. The grid thereby visualizes the estimation points and the cells of the arrays.

The evaluation can, of course, be done in many different ways, easily adjust the file or create your own evaluation file if alternative plots are needed.

4.2.6. Compressive Sensing File *CompressiveSensingClass.m*

Besides the main loop, the *CompressiveSensingClass.m* plays a key role. After a setup, where the parameters are set, it provides lots of functions for the user and can also be applied to many applications, besides the calculation of phase transition diagrams.

When creating an instance of the class with `CSClass=CompressiveSensingClass(SilentMode, PathStr);`, only two parameters are required.

- The *SilentMode* parameter activates additional information messages in the console if set to zero. It is recommended to set it to one, to avoid the software of printing millions of text lines. All necessary messages are printed anyway.
- *PathStr* has to be specified as string and points to the directory where the output shall be saved.

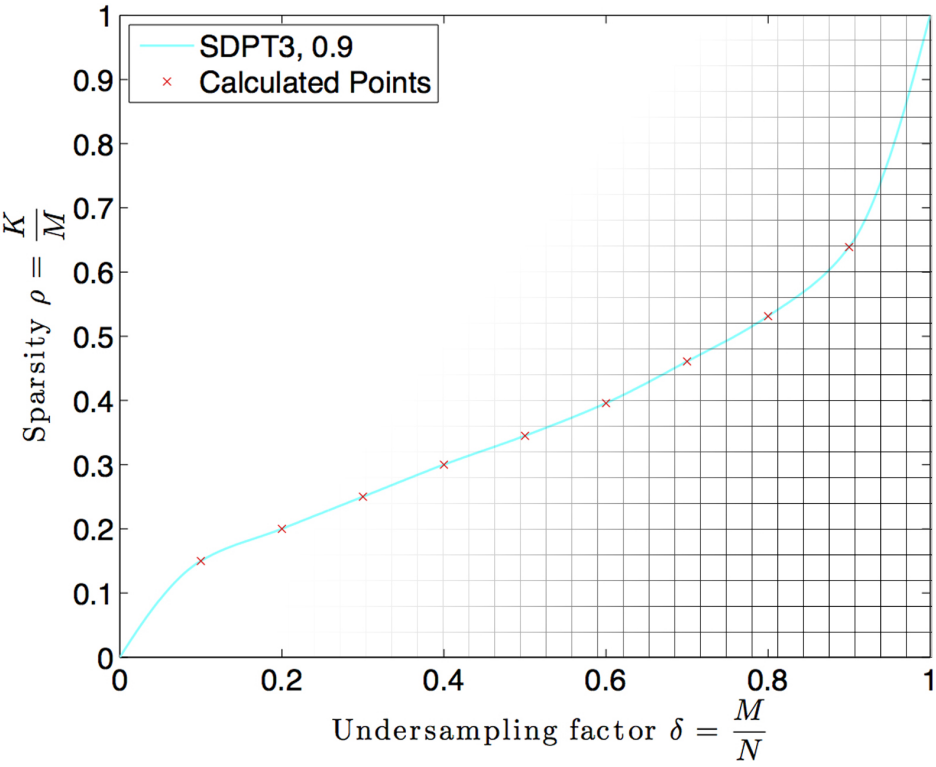


Figure 4.2.: Illustrates the construction of a phase transition diagram from a cluster of simulations, as it is done by the main loop and the evaluation file.

All the other parameters have to be set later. After its initialization, first the settings of the random number generator are stored in the output directory, which was a priori specified by the *PathStr* parameter. If you intend to use an earlier RNG setting, the *SetRandomizer()* function can be used by `CSClass.SetRandomizer(Filename)`. *Filename* thereby specifies the name of the desired rng file as string and consider it to be placed in the output directory. The, at the initialization of the *CompressiveSensingClass* created, rng file will be deleted and the specified settings will be inherited. To guarantee the successful use of this file, ensure the *SetRandomizer()* function to be called directly after the initialization of the class, otherwise the earlier results can not be recalculated, because of the use of random numbers in the other functions.

After this optional step, the setup can be done by calling the *Setup()* function with `CSClass.Setup(N);`. It is mandatory to set

- the length N of the vector $\boldsymbol{x}$.

In addition to that, the following can be set as strings. If not, the default values will be taken.

- The type of $\boldsymbol{\Phi}$.

- The type of $\boldsymbol{A}$.
- The algorithm, used in the reconstruction step.
- The distribution type of the coefficients of the vector $\boldsymbol{x}$.

All possible values and the defaults are listed in Table 4.1 and are extensively described later in this section. The function itself inherits the parameters and controls the internal variables.

Setting the number of samples M can be done in two ways. There is a single function, *SetM()* to update its value. It requires the specification of N and is called by `CSClass.SetM(M);`. A second way is to call the *NewSet()* function, which also requires the current parameter M, by `CSClass.NewSet(M);`. Instead of just inheriting the parameter, the function calls two further ones to generate an instance of the measurement matrix $\boldsymbol{\Phi}$ and the transformation matrix $\boldsymbol{A}$. The functions, used for this purpose, can be called manually, too. They are named *SetPhi()* and *SetA()*, accepting either the particular matrix type as string parameter or when left blank, taking the internal value. The respective calls are `CSClass.SetPhi();` and `CSClass.SetA();`

There are many different types of matrices being supported currently, because of the possibility of choosing $\boldsymbol{A}$ and $\boldsymbol{\Phi}$ separately. The transformation matrix $\boldsymbol{A}$ can be:

- The discrete Fourier transform matrix, which indispensably leads to complex values. It is specified by the string *DFT*.
- The discrete cosine transform matrix, which keeps all coefficients real-valued. It is specified by the string *DCT*.
- The identity matrix. It is specified by the string *eye* and affects the measurement matrix to be equal to the kernel matrix, denoted $\boldsymbol{B} = \boldsymbol{\Phi}$. This, for instance, allows to use completely random matrices.

The kernel matrix $\boldsymbol{\Phi}$ can assume the following forms, extensively described in chapter 2.3 *Reconstruction Guarantees*:

- It is set as random selection matrix by the string *Select*.
- It is set as strict random selection matrix by the string *StrictSelect*. This leads to a random selection matrix, but prohibits the matrix of scrambling the rows of $\boldsymbol{B}$.
- It is set as random Gaussian matrix by the string *Gauss*.
- It is set as random Bernoulli matrix by the string *Bernoulli*.

It is possible to add own matrices or matrix types. Easily add a `case` block to the appropriate functions and orientate on the given blocks.

If intended, the coherence of the matrix $\boldsymbol{B}$ can be increased. Therefore the built in function *EditSelfCoherence()* can be used. It has two different modes. By giving no parameter (`[NewMat,Coh,msg,Count,IterStepSize]=EditSelfCoherence();`),

mode one is called. The function easily calculates the coherence of the normalized columns of the matrix $\boldsymbol{B} = \boldsymbol{\Phi}^{\mathrm{T}} \cdot \boldsymbol{A}$. If the parameter *DesCoh*, denoting the desired coherence value and *eps*, denoting the allowed deviation of the coherence are given (`[NewMat,Coh,msg,Count,IterStepSize]=EditSelfCoherence(DesCoh,eps);`), mode two is called. The function tries to adjust the coherence of $\boldsymbol{B}$ to the specified value. This is, at first, compared to the welch bound and if below it, the calculation will be aborted, because no matrix of this shape, satisfying the coherence constraint, can be created. Otherwise the algorithm tries to adjust the coherence within 50 trials. It will also abort the calculation, if no matrix of this coherence can be created. Consider, if this mode is used, a new instance of $\boldsymbol{B}$ is created in every trial. If the desired actions could be successfully done, whatever mode was chosen, it returns the quintet:

- *NewMat* contains the derived matrix $\boldsymbol{B}$, which can also be the unedited if the coherence already fulfills the constraints or if mode one was used.
- The value *Coh* contains the coherence of the new matrix.
- *msg* contains a status number, which can be (i) 0: Indicating the occurrence of an error, (ii) 1: Indicating success of the adjustment, (iii) 2: Indicating the usage of mode one, without a required adjustment action.
- The number of iteration steps, which were needed by the algorithm, is stored in *Count*.
- *IterStepSize* shows the smallest used iteration step size.

A detailed description of the algorithm and its pseudo code can be found below this section in 4.2.8 *Coherence Adjustment*.

To complete the compressive sensing problem, a sparse coefficient vector $\boldsymbol{x}$ and the sampled signal $\boldsymbol{y}$ is required. These can be created by the *MakeSparseSignal()* function. It requires the current sparsity parameter K, given as number and the additional parameter *AmpDist* and is called by `[y,X]=CSClass.MakeSparseSignal(K,AmpDist);`. The function generates a vector of the length N, selects uniformly random chosen cells and fills them with values of the given distributional type. Different distribution types are supported already and must be specified as strings. These are:

- *Gauss* leads to the generation of standard Gaussian distributed values, $\mathcal{N}(0, 1)$.
- *RandomPhase* leads to the generation of values, with amplitudes $1\mathrm{e}^{\mathrm{j}2\pi z}$, where z is of a standard uniform random distribution, $\mathcal{U}(0, 1)$.
- *UniformPositive* leads to the generation of values, with amplitudes drawn randomly from a standard uniform distribution, $\mathcal{U}(0, 1)$. If chosen, the information of the coefficients being positive, is given to the solving algorithm a priori. But only SDPT3 and SeDuMi can handle this information until now.
- *Uniform* leads to the generation of uniformly random distributed values between minus one and one, $\mathcal{U}(-1, 1)$.

- *Rademacher* leads to the generation of values, either one or minus one by equal chance.

- *Benchmark* leads to the generation of values, with the value one. It is a one point distribution.

As before, new types can easily be added inside the appropriate `switch`-statement.
In the end, the function generates the sampling vector $\boldsymbol{y}$ by multiplication, as defined in (2.1). It returns the sampled signal and the sparse original signal [y, X[14]].

The function *ReconstructIt()* does the reconstruction step as in formula (2.2). No parameter is required, the previously chosen solver is used and the call is `CPUTime= CSClass.ReconstructIt();`. If the solver found a solution to the problem, the internal variable *X_hat* (which represents $\boldsymbol{x}^*$) is set and the required computational time in seconds is returned (*CPUTime*). The reconstruction is done by the chosen solver, which can be done by setting the *solver* parameter. There are quite a few supported solvers already:

- *SDPT3* is a solver from the group of basis pursuit and the default solver of the program. It is an implementation of the *Primal Dual Infeasible Interior Point* algorithm [29], is provided by the CVX package[15], which is a huge framework for specifying and solving convex programs [15, 16], and there used as default solver, too. The solvers of the CVX package are denoted, as delivering as good results as MOSEK[16], which was used to create the PTDs, presented in [12, 11]. Thus the CVX solvers are used as reference solvers.

- *SeDuMi* is a solver from the group of basis pursuit, too and also provided by the CVX package [25]. It implements a variant of the *Centering Predictor Corrector Method*
  The CVX documentation on the one hand states the SDPT3 algorithm to be slightly better as the SeDuMi algorithm, which on the other hand works a bit faster. Both can be seen as reference algorithms for the group of basis pursuit. [16]

- *l_1_magic* is another solver from the group of basis pursuit. l_1_magic implements a *Primal Dual Interior Point* algorithm, too and is denoted as being much faster than the other basis pursuit solvers SDPT3 and SeDuMi, but working less reliable. [3]

- *MP* is a solver from the group of the greedy methods and provided by the *SparseLab* package, which is included in PTD Builder [10]. It implements the *Matching Pursuit* algorithm.

[14] Note, the algorithm uses the capital X to emphasize its domain, which in the programs standard application is the frequency domain.
[15] Consider the CVX package must be installed, see section 4.2.1 for more details.
[16] MOSEK today is also part of the CVX package, but subject to a professional license. CVX delivers two more solvers, which are subject to the professional license, and therefore not included in the PTD Builder software by default. These are *Gurobi* and *GLPK*.

- *OMP* is a solver from the group of the greedy methods, too and also provided by the SparseLab package. It implements the *Orthogonal Matching Pursuit* algorithm.
- *STOMP* is a third greedy solver and also provided by the SparseLab package. It implements the *Stagewise Orthogonal Matching Pursuit* algorithm.
- *LASSO* is a solver, provided by the SparseLab package and can not directly be classified. It implements the *Least Angle Regression* algorithm. It uses a slightly different solution approach as the basis pursuit solvers, but works a bit similar to greedy methods in its iteration steps [13].
- *SL0* is an autonomous solver and again from a different type. SL0 stands for *Smoothed L0* and tries to directly minimize the ℓ_0-norm, by minimizing a smooth approximation function. [20]

Adding new solver algorithms is possible, too. As mentioned, when explaining the other functions, just add a new `case` statement and orientate on the given implementations.

In the end the success of reconstruction must be checked. This can be done by the *CheckSuccess()* function. It is called by `Success=CSClass.CheckSuccess(Threshold)` and compares the support sets of the original and the reconstructed vector. The *Threshold* parameter needs to be given, which specifies the allowed deviation of values, to be treated as zero. In the end the function signals success by returning one and failure by returning zero.

Reading the values of the internal variables is possible due to the *GetAccess* attribute being set to `public`. Setting a value can only be done by a specific function, to ensure functionality of the class. If needed, you can change the given functions of the class or add your own functions to the class to improve the functionality or stability.

4.2.7. Troubleshooting

Table 4.2 contains the most common warning messages and Table 4.3 the most common error messages. Both tables contain descriptions of how to avoid the problems.

Warnings	
message	**description**
Output folder created	A desired folder had to be created by MATLAB. No further action is required.
There exists a Results.mat file already. It will be renamed!	There is a previous results file in the desired output directory. It will be renamed by adding the prefix 'Pre' and a continuous number as suffix. No data gets lost.

Table 4.2.: Troubleshooting

Errors	
message	**description**
Wrong number of input arguments.	The MakePhaseTrans.m function expects exactly two arguments. Check the number of arguments in your function call.
Input arguments must be strings.	The parameters passed to MakePhaseTrans must be strings. Check the type of the parameters in your function call.
The parameter file you specified does not exist.	The function can not find a parameter file with the specified name. Check the spelling of your parameter and its location.
Unknown file format of the parameter file. You can use .m or .mat-files.	Only .m and .mat files can be handled by the program. Have you specified another? If not, probably the code part of the parameter file is damaged. Try to repair it, by adding the line `save Import.mat`.
Too many input arguments!	Initialization of the compressive sensing class failed, because of too many arguments in the function call. Try to call it with '`CompressiveSensingClass(Feedback,PathstrDir);`' string.
Insufficient number of input arguments!	Initialization of the compressive sensing class failed because of too few arguments in the function call. See above for troubleshooting.
Please run the Setup function before!	The compressive sensing class has to be fully set up, before it is available for use. Call the setup with '`CSCoder.Setup(N,PhiType,AType,Solver,AmpDist);`' command.
Can not set the Original Data vector X! (*AmpDist*) is unknown. Did you check the spelling?	Check the spelling of the *AmpDist* parameter.
Can not set the transformation matrix A! (*AType*) is unknown. Did you check the spelling?	Check the spelling of the *AType* parameter.
Can not set the transformation matrix A! (*PhiType*) is unknown. Did you check the spelling?	Check the spelling of the *PhiType* parameter.
Can not solve the problem! (*Solver*) is unknown. Did you check the spelling?	Check the spelling of the *Solver* parameter.
Attempted to access CohMap(1,0); index must be a positive integer or logical.	Check your *DHor* parameter. Calculation at $M = 0$ is not allowed.
Your desired bounds could not be found in the Results.mat file!	You specified bounds, which were not being calculated, in at least one evaluated measurement. Recalculate with this bounds or adjust the *DesiredBounds* parameter.

Table 4.3.: Troubleshooting

property	**Cluster A**	**Cluster B**
Compute Nodes	100	48
Main Memory	24 GByte	64 GByte
CPU	Intel X5550 2,67 GHz (8-core)	AMD OPTERON 6134 2.3GHz (32-core)

Table 4.4.: Technical details of the massive parallel computer clusters of the Ilmenau University of Technology.

4.2.8. Coherence Adjustment

In the context of creating phase transition diagrams, a second algorithm shall be described in detail. The function *EditSelfCoherence()* is brought along, by the compressive sensing class. It is able to measure the coherence of a matrix and adjust it with the aim of increasing the coherence to a given value. While the use of the function is described in section 4.2.6 *Compressive Sensing File CompressiveSensingClass.m*, this section describes how the coherence adjustment is done. The pseudo code is shown in Algorithm 2.

The function takes a given matrix and creates a second matrix $\boldsymbol{\Delta}$ of the size $M \times M$. All elements of this matrix are set to a starting value ρ_s between zero and one, but the diagonal elements, which are set to one. This can mathematically be written as

$$\boldsymbol{\Delta} = (\rho_{\mathrm{s}} \cdot \boldsymbol{I}_{M\times M} + (\mathbf{1}_{M\times M} - \rho_{\mathrm{s}} \cdot \boldsymbol{I}_{M\times M})) \tag{4.2}$$

whereby $\boldsymbol{I}_{M\times M}$ denotes the identity matrix of the size $M \times M$.

The matrix multiplication

$$\tilde{\boldsymbol{B}} = \boldsymbol{\Delta} \cdot \boldsymbol{B} \tag{4.3}$$

preserves the size of $\boldsymbol{B}$, but adds a deterministic part and for this reason, increases the coherence of the original matrix, dependent on the value ρ_s. The closer ρ_s comes to one, the higher will the coherence be.

In following iteration steps, ρ_{s} is refined, until the coherence of $\tilde{\boldsymbol{B}}$ has the desired value, less an adjusted threshold ϵ.

4.3. Computational Environment

Many simulations were done with PTD Builder. Therefore, the computer clusters of the *Ilmenau University of Technology* were used, which were kindly made available, by the computing center of the university. They are providing two massive parallel computer clusters, managed by an administrational software *Load Sharing Facility*. All tasks are arranged in render queues and executed on compute nodes if free resources are available. The operation system, the clusters are running with, is *CentOS 6.6*. All technical details can be found in Table 4.4. [28]

Algorithm 2 Adjustment of the matrix coherence

Input: $\epsilon, \boldsymbol{B}, DesiredCoherence, M$

Set $Coh = DesiredCoherence$, $\rho_\mathrm{s} = 0.5$, $cmp = 0.5$ and Stepsize = 0.1.
repeat
    $\tilde{\boldsymbol{B}} = (\rho_\mathrm{s} \cdot \boldsymbol{I}_{M\times M} + (\mathbf{1}_{M\times M} - \rho_\mathrm{s} \cdot \boldsymbol{I}_{M\times M})) \cdot \boldsymbol{B}$
    $Coherence =$ CalculateCoherence($\tilde{\boldsymbol{B}}$)
    if $Coherence > DesCoh$ **then**
        if $cmp == \rho_\mathrm{s} - Stepsize$ **then**
            $Stepsize = Stepsize \cdot 0.1$
        $cmp = \rho_\mathrm{s}$
        $\rho_\mathrm{s} = \rho_\mathrm{s} - Stepsize$
    else
        if $cmp == \rho_\mathrm{s} + Stepsize$ **then**
            $Stepsize = Stepsize \cdot 0.1$
        $cmp = \rho_\mathrm{s}$
        $\rho_\mathrm{s} = \rho_\mathrm{s} - Stepsize$
until $Coh - DesCoh \leq \epsilon$

Output: $\tilde{\boldsymbol{B}}, Coherence, Stepsize$

4.4. Summary

PTD Builder is a program, developed in the scope of this work. It provides a large range of functions, regarding compressive sensing, but is optimized for the needs of the creation of phase transition diagrams. It is on the one hand very easy to use, but can on the other hand be used for complex simulations as well.

It uses a newly developed algorithm, which is much faster than the classical approach, because of the permanent evaluation of the simulation results. Its sole disadvantage is the impossibility of producing continuous phase transition diagrams. But diagrams from literature and the ones presented in the following chapter, point out that these are often not required. Yet, two to three curves, denoting the 10% and the 90% or the 10%, 50% and 90% points, are entirely sufficient. These can very fast and efficiently be simulated by PTD Builder.

5. Analysis of Phase Transition Diagrams

As stated in chapter 3.3 *Theoretical Results in Literature*, having no sharp bounds, is a problem, when dealing with compressive sensing. Many different bounds can be given, which guarantee success and are sufficient, but not necessary conditions. As long as this lack of tight bounds exists, experimental bounds, as phase transition diagrams are, have to be used.

This chapter provides a lot of such diagrams, describes the settings, they were simulated with and analyzes the results. The first PTDs are compared to the ones of Donoho and Tanner and used to validate the algorithms of PTD Builder. The further ones measure the impact of different influencing factors, as the solving algorithm, the type of $\boldsymbol{B}$, the distributional type of the coefficients of the vector $\boldsymbol{x}$, the coherence of $\boldsymbol{B}$ and others. All outcomes are compared to the theoretical and computational results, presented in the previous chapters. Besides the success rate, the computational costs are considered, which are very important in real applications, and stated as major advantage of greedy solvers.

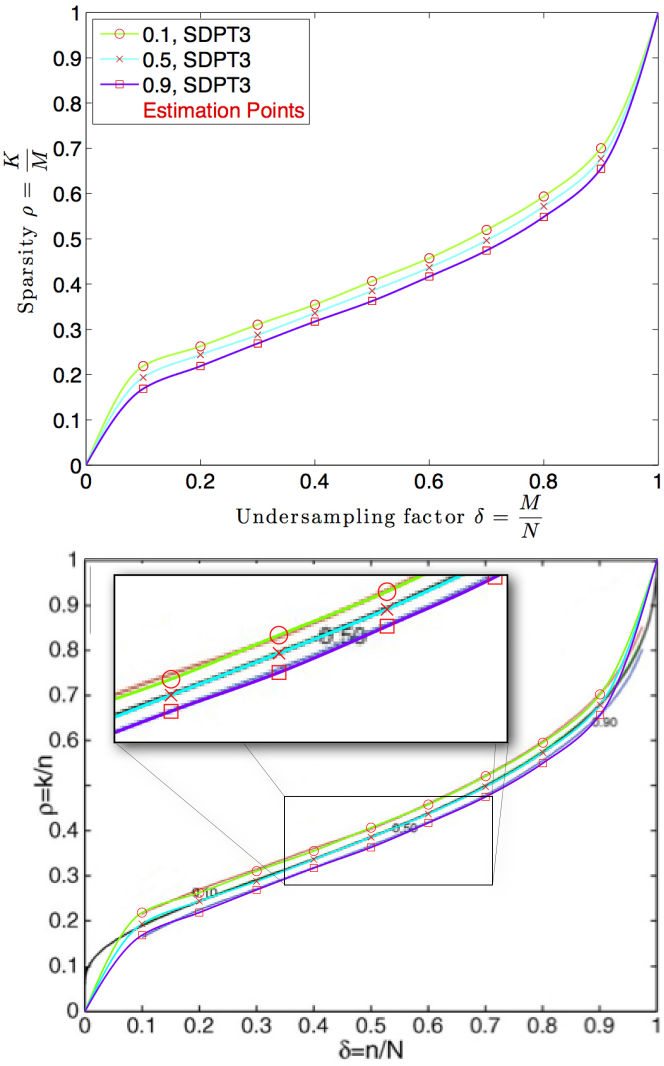
All results presented here, are included in the digital version of this work and can, at least in parts[17], be retraced. However, some of the experiments were done with a predecessor version, where handling may differ.

5.1. Validation of the algorithm

Before the examination of the influencing factors, the question of validity of the algorithms, being used in PTD Builder, has to be clarified. Except for two differences, the program was tested with settings very similar to the ones, used by Donoho and Tanner in [12], referred in Table 3.1. First, the solver was changed from MOSEK to SDPT3, to enable recalculation without a professional cvx license, which is neglectable, as stated by Donoho and Tanner in [11, cf. p. 13-14]. Second the distribution of the signal has slightly changed from Gaussian distributed entries to Rademacher distributed ones, admitting only the values one or minus one. This was a mistake, but since basis pursuit algorithms are stated to ensure uniform guarantee, it should not be of any consequence. Due to long calculation time, when using SDPT3, recalculation could not be covered in this thesis. However, to make a validation statement, PTDs of shorter signal lengths (reducing N from 1600 to 400) and both considered distributional types were done. It

[17] As mentioned in chapter 4.2.3 *Parameter File ParEx.m*, there was a deficient function in the compressive sensing class, which is why exact recreation fails. However the resulting PTD will be similar.

depicts that there is no structural difference between them with the signal amplitudes, being distributed Gaussian and the Rademacher ones, when recovered by SDPT3.[18]



(a) First simulation results of PTD Builder, showing the success rates 10%, 50% and 90%, recovered by SDPT3. Settings: $N = 1600$, $\boldsymbol{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher

(b) Overlap of phase transition diagrams, created in the scope of this work (see 5.1a) and the one by [12] (see 3.2). For a more detailed view, a piece from the middle is sliced out. The coincidence of both diagrams seems obvious.

Figure 5.1.: Validation of PTDBuilder on the basis of the PTDs of Donoho and Tanner.

The results of the validation are shown in Figure 5.1a, whose calculation took about 26 days. The lower figure is an overlay of the Figures 5.1a and 3.2. To show the coincidence of both phase transition diagrams, a middle part is sliced out and scaled up. The graphs reveal that the estimated points almost fit with the associated curves of Donoho and Tanner. Their exact values are unknown, but the average deviation is approximately zero. Though, two apparent differences exist. First the curves of Donoho and Tanner are

[18] See Appendix E.1 for details.

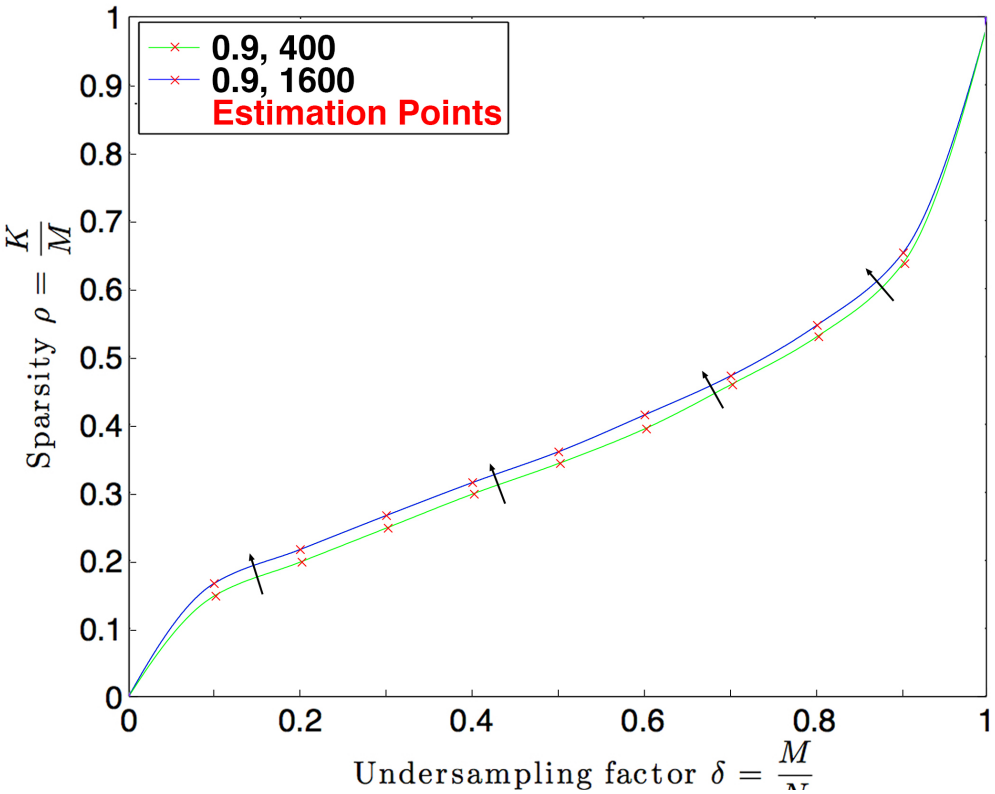


Figure 5.2.: Illustrating the 90% curves of identical experiments, except different signal lengths N, recovered by SDPT3. Settings: $\mathbf{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher

smoother than the others. This may come due to a more exact reconstruction algorithm, a better interpolation algorithm or a subsequent smoothing of the lines. While PTD Builder uses piecewise cubic hermite interpolation technique, there is no information given about the smoothing technique from Donoho and Tanner. The second difference regards the boundary points. As described in 3.2 *Computational Results in Literature*, the deviation of the interpolated curves in these regions was proposed as high. A direct comparison of these lines is therefore not very meaningful, because the black line represents an analytically derived bound, while the others are interpolated. To achieve more information in these regions, simulations have to be done at these extreme estimation points (see section 5.6 *Other Influences*). Unfortunately these points were not simulated by Donoho and Tanner.

A further diagram shall address the property of the shifted curves, due to increased signal length N. Therefore the results of Figure 5.1a were reused and compared to a phase transition with similar settings, except a reduced signal length of $N = 400$. Again,

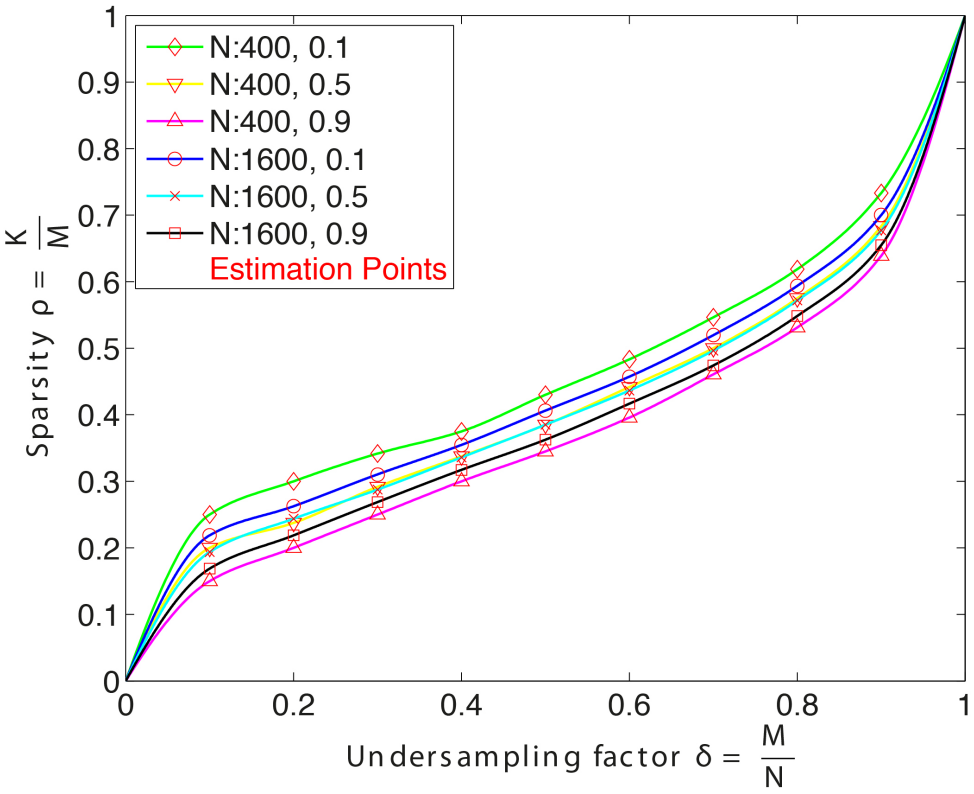


Figure 5.3.: Overlay of two phase transition diagrams with different vector lengths N, showing the shrinking phase transition zone with increasing N. Settings: $\boldsymbol{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher, but different N, Solver: SDPT3, 10%, 50% and 90% curves.

the result (see Figure 5.2) shows the expected behavior. In difference to Figure 3.4, the distance between the two lines is much smaller. But do not forget, that Figure 5.2 just shows the 90% curves instead of the 99% ones, which explains the difference. This and the results of another diagram offer more information. Figure 5.3 shows the 10%, 50% and 90% curves, from the previous simulation. A scaled up sector from the middle can be seen in Figure 5.4. The fact that both 50% curves coincide, shows that the transition region by itself is constant. But the other curves show, that the width of the transition region shrinks, with increasing signal length N. Comparing these outcomes with the derived curve, depicting $N \to \infty$ of FigurePTD01, shows clearly, that the width of the transition region tends to zero for $N \to \infty$. This means the larger N the sharper gets the transition region.

All of these observations lead to the conclusion, that the results of PTD Builder are

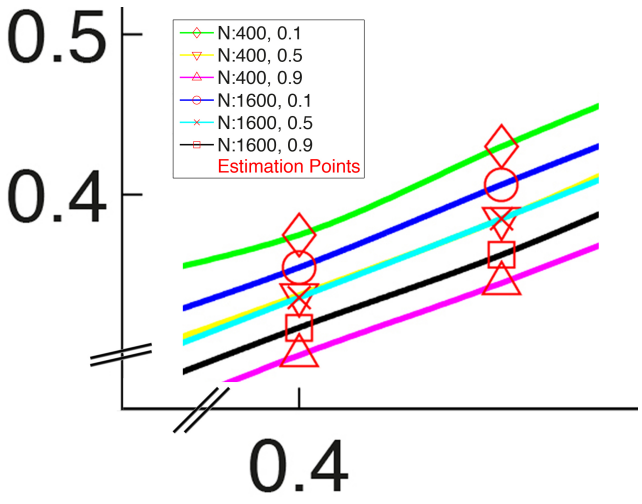


Figure 5.4.: Scaling of a middle part of Figure 5.3. Settings: $\boldsymbol{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher, but different N, Solver: SDPT3, 10%, 50% and 90% curves.

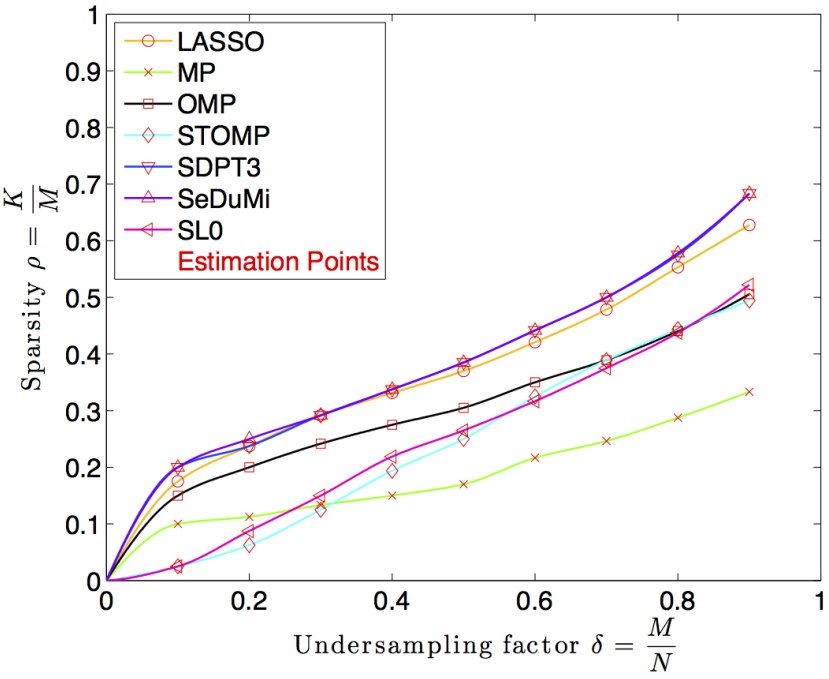
consistent with the ones, shown by Donoho and Tanner. The program PTD Builder works as desired.

5.2. Influence of the solving algorithm

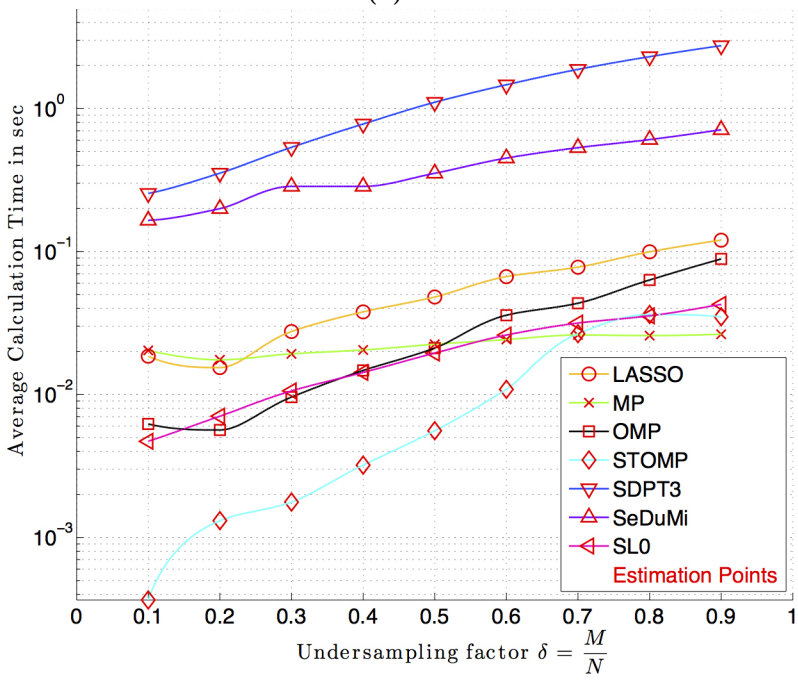
The solver is a very important part in compressive sensing applications. Since the interest in compressive sensing grew and suitable hardware is available, there are many different algorithms and hundreds of different implementations, all with their pros and cons. To find out more about the behavior of the different solvers, a further simulation was done, using similar settings, as in the ones before. The signals had the length of $N = 400$, the amplitudes were Rademacher distributed and partial random cosine matrices were used. The 50% curves are plotted in Figure 5.5a, Figure 5.5b below it shows the average time required per estimation point, in a logarithmic scale.

It depicts the expected results, first the basis pursuit solvers, represented by SDPT3 and SeDuMi, delivered the best ones. Their phase transition curves lie clearly above the others, and the curve, representing the greedy solver MP, mostly lies at the lowest sparsity points. But some more details are to be discovered at a second glance. The reconstruction with LASSO brings up nearly as good results as the basis pursuit solvers, whose success rate continuously lies slightly beneath the associated basis pursuit rates. Considering the average time taken per measurement (see Figure 5.5b), the LASSO algorithm shows its strength. It works, at least, about six to ten times faster than SeDuMi. Thereby SeDuMi itself is about two to five times faster than SDPT3, without being worse in the recovery results.

Further evaluation shall treat the first mentioned point, that the simulation indicates similar reconstruction results of SeDuMi and SDPT3, while SDPT3 requires clearly more computational time. This is interesting because SDPT3 is predicted as being more reliable as SeDuMi [16]. To further examine the solvers, a simulation was done, to compare



(a)



(b)

Figure 5.5.: Comparison of (a) the success rates (50% curves) and (b) the average computational time, needed per estimation point, using different solvers. Settings: $N = 400$, $\mathbf{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher.

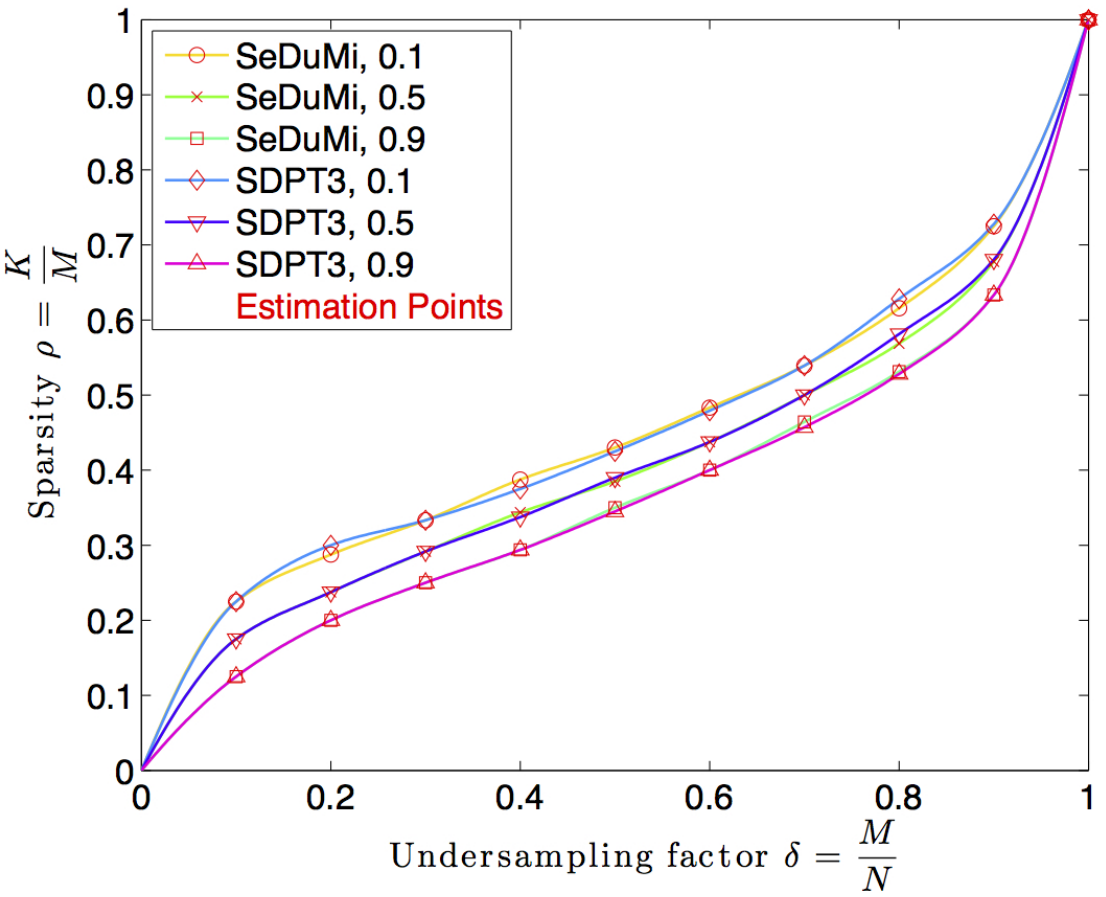


Figure 5.6.: Phase transition diagrams of SeDuMi and SDPT3. Settings: $N = 400$, $\boldsymbol{\Phi}$: Gauss, $\boldsymbol{A}$: eye, AmpDist: Gauss, 10%, 50% and 90% curves.

the position and width of the transition zones of both solvers. The settings were changed to Gaussian distributed amplitudes of $\boldsymbol{x}$ and a random Gaussian measurement matrix $\boldsymbol{B}$, at the same signal length $N = 400$. The result can be seen in Figure 5.6 and again shows equality of both outcomes. Both solvers deliver similarly good recovery results, and equal widths of their phase transition zones, under this conditions. SeDuMi is, considering these two experiments, faster and as good as SDPT3, which emphasizes its power. By now it will act as reference.

The second mentioned point regards the power of the greedy solvers. The curve of LASSO was surprisingly good, why another experiment was done, addressing the often worse assessed greedy methods. While their curves lie, in the former plot, clearly below the ones of basis pursuit, Figure 5.7, showing the 90% curves, depicts, that this must not be assumed for every setting. The settings of the experiment were $N = 400$, Gaussian distributed amplitudes of $\boldsymbol{x}$ and partial random cosine measurement matrices $\boldsymbol{B}$.

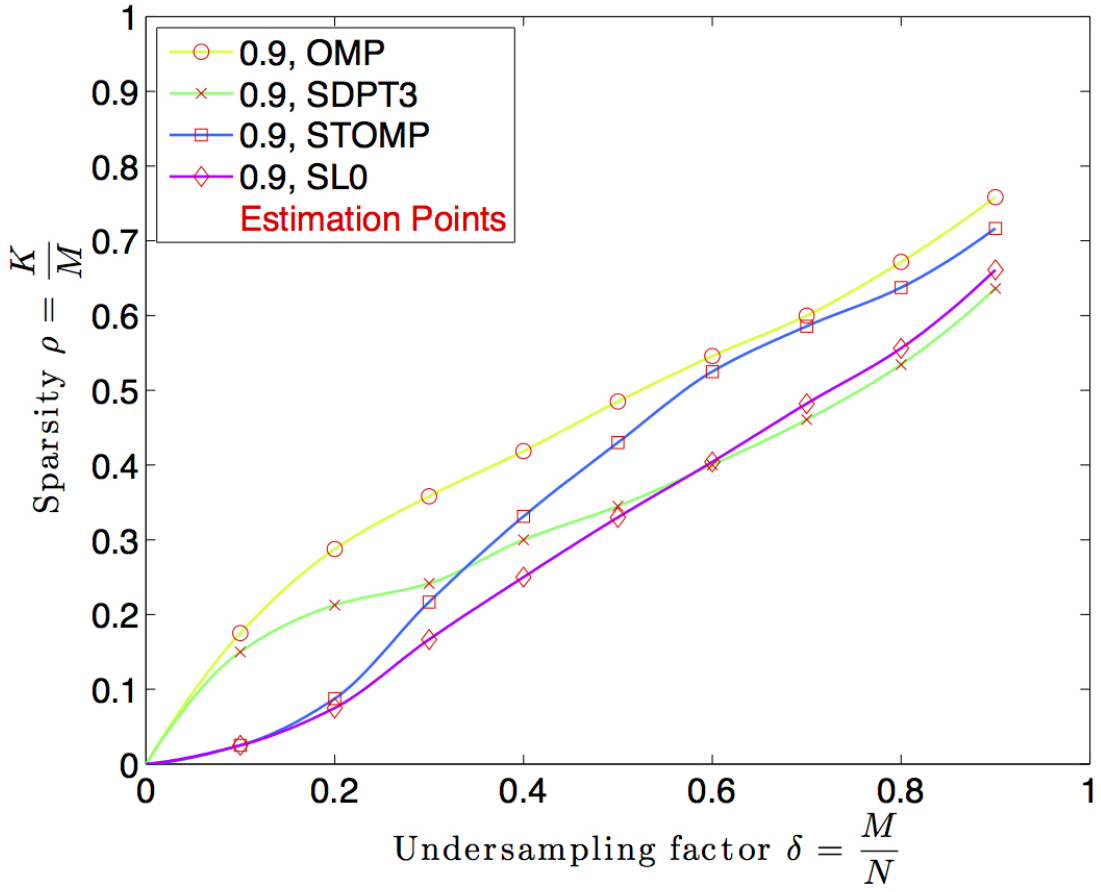


Figure 5.7.: Illustrating the 90% curves of SDPT3, OMP, STOMP and SL0. All three non basis pursuit solvers gain, in at least a small region, better recovery results as the basis pursuit reference SDPT3. Settings: $N = 400$, $\mathbf{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Gauss

Dependent on the settings, greedy solvers can, in manners of success, be superior to basis pursuit solvers. This is clearly shown by the former experiments, whose plots also depict, that greedy solvers seem to have a form of special area of expertise and areas where their results are not a quarter as good. The mentioned example shows a setting where OMP and STOMP mostly beat SDPT3.

While both basis pursuit solvers SDPT3 and SeDuMi reach the point $\delta = 1$, $\rho = 1$ with success rate of one, many greedy methods seem to not achieve this point and touch the right diagrams corner at middle sparsity values. This behavior was, due to the simple reconstruction technique at the undersampling rate of one, not expected and the greedy methods were at first implied to be as successful as the basis pursuit ones, at this point. Unfortunately, this was noticed very late and extensive additional simulations, evaluating the appropriate behavior of the greedy solvers, for all phase transition diagrams, were

not possible. This is the main reason why the affected diagrams are cut off at the highest simulated undersampling rate. The second reason is, that these areas are less important. A minimal reduction of the undersampling rate, already leads to heavily increased computational costs in the reconstruction step and is very ineffective. This can be seen in Figure 5.5b. The computational time, required by the algorithms, is at its maximum at the highest undersampling rates, while the saving in the sample rate is minimal. Nevertheless, this theoretic result should be mentioned and the performed experiments should in future be extended, to gain reliable data at this point. Considering adaptive applications could lead to situations, where signals make it necessary to sample at high rates near $\delta = 1$. Using greedy methods, probably requires the solver to be changed, then.

The SL0 solver shows an interesting behavior. Its curve keeps at very low sparsity rates, at low undersampling rates. But as the undersampling rate passes a certain point, the curve raises and shows a relatively constant slope. This stays in contrast to the typical shape of most of the other curves, which remind of a mirrored 'S'. This makes SL0 useless, at these low undersampling rates, but as the rate is at medium values, SL0 gets interesting, especially because of its high speed.

In general, the choice of a suitable solver always depends on the position, an application works, in the diagram. While it is well suited for the use in one area, it may be unsuitable for another, which can clearly be seen at the curves of SL0. It is well suited at middle undersampling rates, but useless at low rates. Following observations shall take this into account and differentiate three areas, which are the extreme low, the middle and the extreme high undersampling rate areas.

In contrast to the previous experiment, another one depicts some disadvantages of the greedy algorithms. Figure 5.8 shows the 10%, 50% and the 90% curves of the phase transition diagrams of SDPT3 and LASSO. The signal length again was $N = 400$, the amplitudes of the coefficient vectors were Rademacher distributed and the measurement matrices were partial random cosine matrices. The experiment is equal to the one presented in Figure 5.5, but showing the whole phase transition zone.

It can be seen, that the width of the phase transition zone of SDPT3, stays relatively constant, with increasing undersampling rate. In contrast to that, the width of the phase transition zone of LASSO grows, within the same range. While the upper curves (10% and 50%) keep relatively constant, the lower curve (90%) has a clear lower raise. Bringing this to real applications again, LASSO proves as worse if high success rates are mandatory.[19]

Finally choosing a suitable solver for concrete compressive sensing applications, leads to a weighting of the three points (i) success rate, (ii) speed and (iii) width of the transition zone, which shall be used as measure of the reliability.Although, the results of greedy methods are not as good and consistent, against the change of settings, as the ones of the basis pursuit solvers, the group of greedy algorithms should be considered, when seeking a suitable solver for a concrete application, even if high success rates are required.

[19]This behavior could not be observed, in this clarity, in the other experiments.

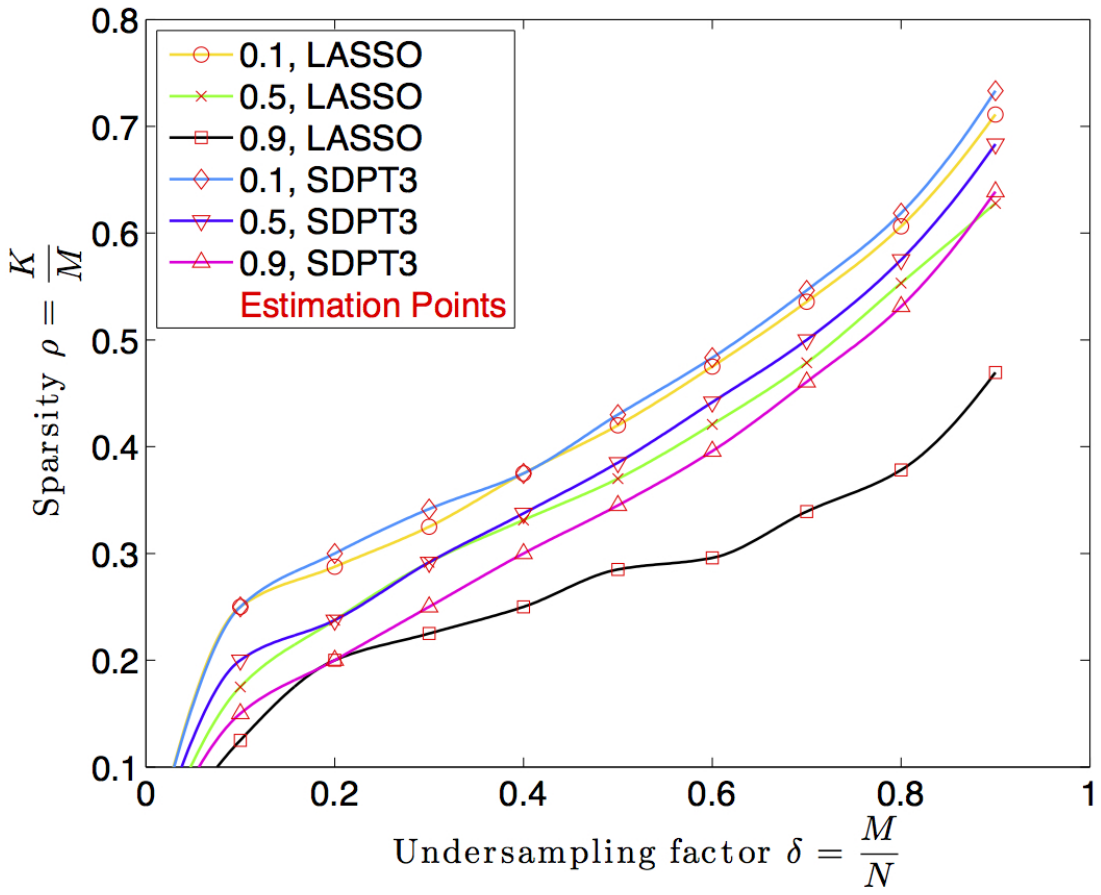


Figure 5.8.: The width of the phase transition zone of LASSO grows with increasing undersampling rate, while the phase transition zone of SDPT3 stays relatively constant, within the same range. Settings: $N = 400$, $\boldsymbol{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, AmpDist: Rademacher, 10%, 50% and 90% curves.

5.3. Influence of the measurement matrix

The measurement matrix $\boldsymbol{B}$ plays a key role in compressive sensing and is, as former chapters showed (see 3.3 *Theoretical Results in Literature*), a key to derive analytic limits and bounds. But it primarily has to model a real sensing scheme, which can lead to some restrictions. Although, the program is able to create lots of combined measurement matrices, it is subtle to evaluate the influence of different kernel and transformation matrices, to phase transition diagrams, in one step. Therefore, this thesis focuses onto the mentioned ones.

Therefore, one solver of every group was tested with constant settings $N = 400$ and coefficient vectors $\boldsymbol{x}$ with Gaussian distributed amplitudes, since no solvers had problems with such signals in the former tests. The group of basis pursuit is represented by

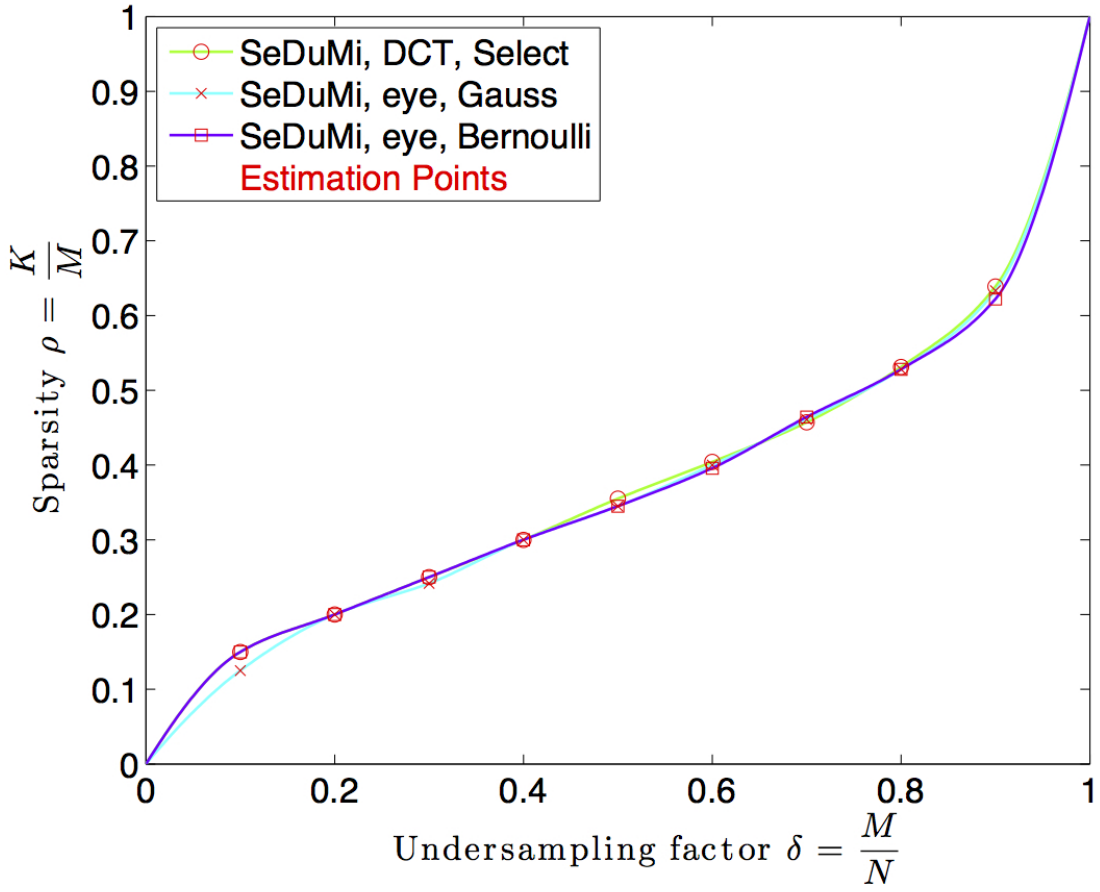


Figure 5.9.: There is no structural difference in the 90% curves of SeDuMi, when varying the measurement matrices. Settings: $N = 400$, AmpDist: Gaussian. $\boldsymbol{B}$: Partial random cosine matrix, Gaussian random matrix and Bernoulli random matrix.

SeDuMi, greedy by OMP and LASSO and SL0 as not directly classified types. All solvers had to pass the same experiment three times, while the measurement matrix changed from a partial random cosine over Gaussian random to Bernoulli random type. To meet the claim of a benchmarking and stable algorithm, first SeDuMi was studied and its results are shown in Figure 5.9.

As expected, the basis pursuit solver produces three more or less identical curves. This tightens the statement of stability and independence of basis pursuit.

As representative of the greedy solvers, OMP was tested next. Its results can be viewed in Figure 5.10. The settings seem to fit very well with OMP, all three results are, in every simulated part, better than with basis pursuit. Only the points at very low and high undersampling rates and Gaussian or Bernoulli random matrices, are the sole exception. They are equal with the points of the SeDuMi curves. The most interesting curve, is the one, measured with the partial random cosine matrices. It clearly lies above the others

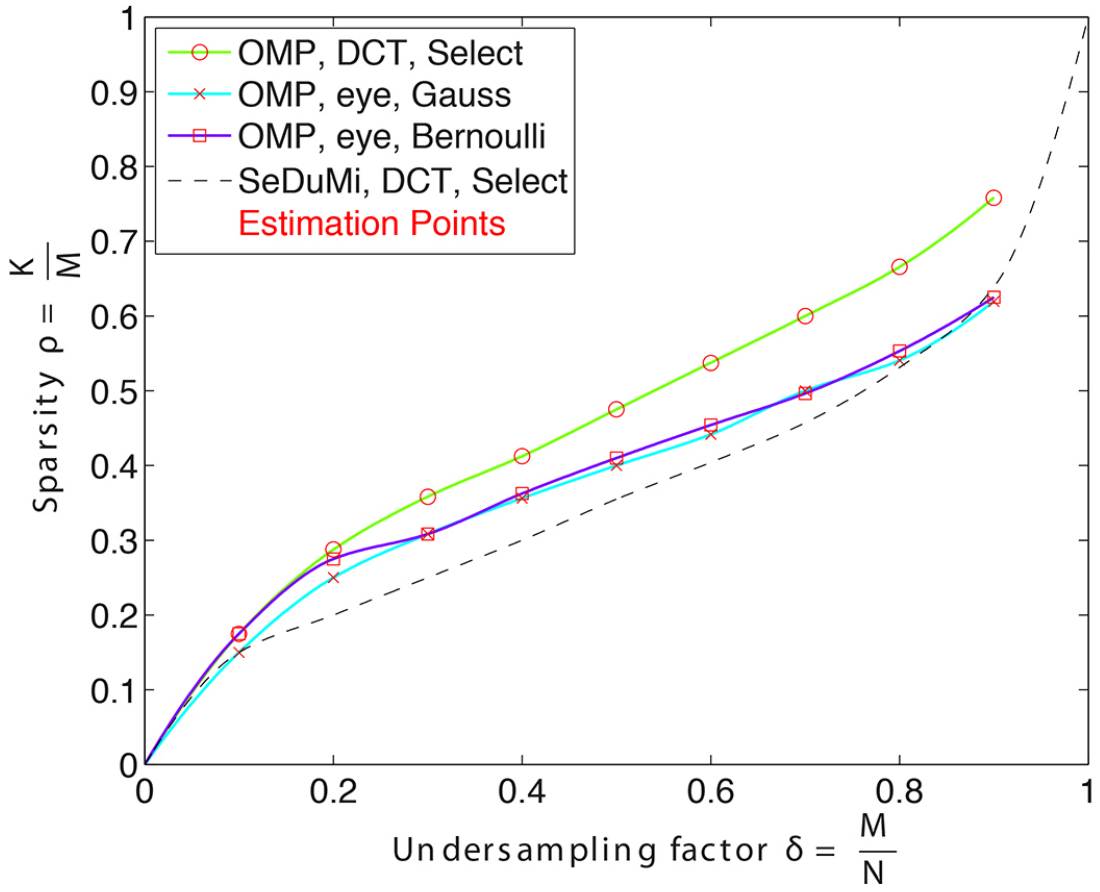


Figure 5.10.: The 90% curves of OMP, representing the greedy algorithms, only show small influence of the varied measurement matrix. Settings: $N = 400$, AmpDist: Gaussian. $\boldsymbol{B}$: Partial random cosine matrix, Gaussian random matrix and Bernoulli random matrix.

and is considerably better than the curve of SeDuMi.

While the influence of the measurement matrix on OMP is small but visible, the LASSO algorithm is clearly influenced by the type of the measurement matrix, which can be observed in Figure 5.11 and shows surprising results.

The first point to note, is that all curves are clearly below the ones of SeDuMi. Second, it depicts that the success rate collapses if used with random matrices. These curves show an interesting peak at low undersampling rates. Its slope to increasing undersampling rates is relatively flat and can, at the moment, not be explained. Another peak can be observed at the curves of the partial random cosine matrices. It lies at the high undersampling rate of 0.8 and is until now not detected in any other simulation. The surprise is that, in contrast to that, the 90% LASSO curve, shown in Figure 5.8, does not have this fall, despite similar settings. Certainly, there are two differences in the parameter

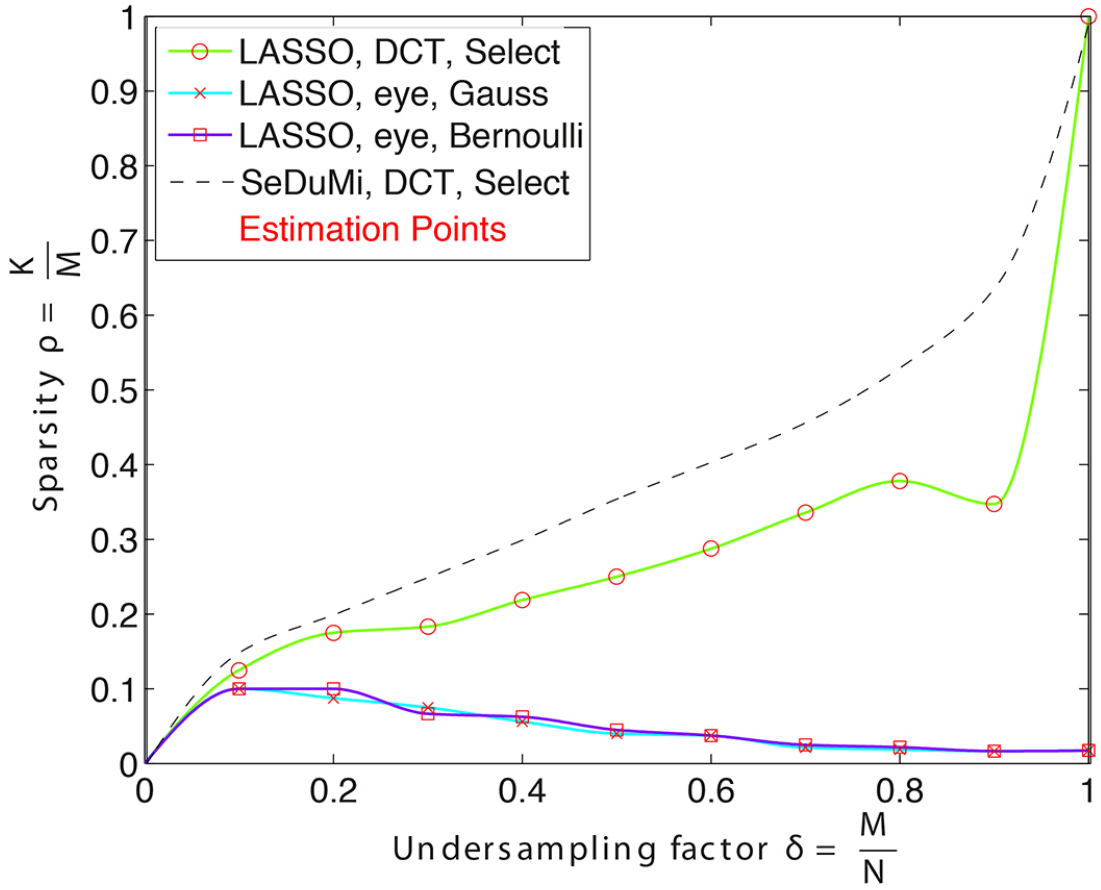


Figure 5.11.: The 90% curves of LASSO, when varying the measurement matrix, are shown. It is compared to the 90% curve of SeDuMi. Settings: $N = 400$, AmpDist: Gaussian, $\boldsymbol{B}$: Partial random cosine matrix, Gaussian random matrix and Bernoulli random matrix.

files. The former measurement was done with Rademacher distributed amplitudes and a kernel matrix being of the *StrictSelect* type, while the latter one was done with Gaussian distributed amplitudes and a kernel matrix being of the *Select* type. Section 5.4 *Influence of the distribution type of the coefficient vector* concerns with the observed anomaly and presents a further simulation (see Figure 5.15) with the aim of clarifying this behavior.

A third point shall be mentioned. The simulation contains the estimation point at $\delta = 1$. It depicts that the upper curve, which was all in all successful, reaches this point at $\rho = 1$. Both other, less successful, curves do not. They converge to a very low sparsity point slightly above $\rho = 0$. It follows that the knowledge of the behavior at this point, predicts if the solver suits to a concrete application.

Last, a simulation was done with SL0 as solver. Its outcomes are shown in Figure 5.12. All three curves are yet more or less identical, as if using a basis pursuit solver.

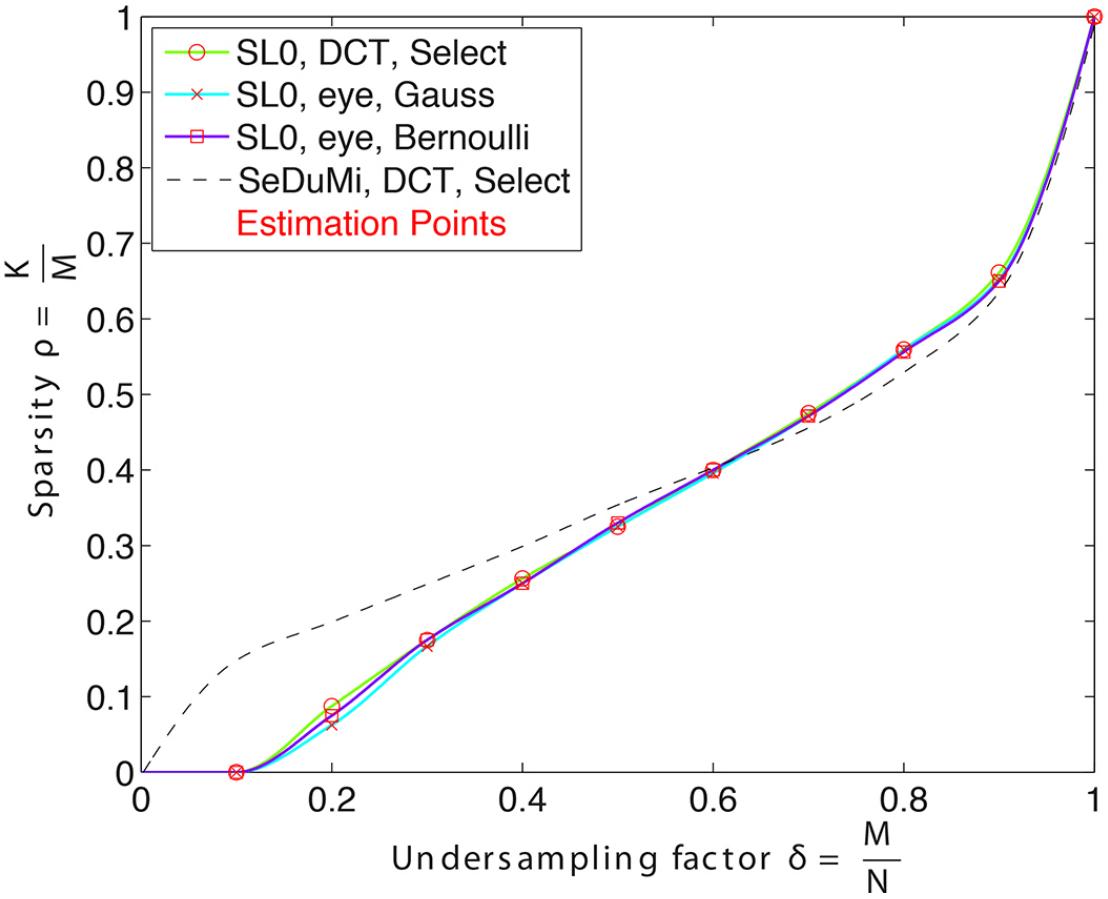


Figure 5.12.: Comparison of the 90% curves of SL0, while varying the type of the measurement matrix. The curves are identical. Settings: $N = 400$, AmpDist: Gauss, $\boldsymbol{B}$: Partial random cosine matrix, Gaussian random matrix and Bernoulli random matrix.

SL0 shows its typical shape and seems insensitive against the type of the measurement matrix.

All experiments clearly show, the greedy methods, working iterational towards the direction of the greatest gain, are highly dependent on the type of the measurement matrices. The tested representatives seem adapted to the real case, which is represented by the partial random cosine matrix, and suffer from the theoretical case with the fully random constellations. But the experiment with OMP shows, that the intensity of this behavior differs. In opposite to them, the basis pursuit solver and the SL0 solver seem to be insensitive against the change of the measurement matrices. This was not expected, because of the theoretical results of chapter 3.3 *Theoretical Results in Literature*, where it is stated that the use of different measurement matrices $\boldsymbol{B}$, lead to different bounds.

5.4. Influence of the distribution type of the coefficient vector

In a fourth step, the distribution type of the coefficient vector $\boldsymbol{x}$ was varied. At the moment, just a few different types are implemented in the program. Future versions shall provide more types, to enable modeling of more real applications. Records of human speech, for example, differ from ones of earthquakes or heart rates.

While Nyquist sampling does not make a difference between different signals, compressive sensing could. It was stated, that if the null space property of order K, is held by a measurement matrix, the recovery of all K-sparse vectors is possible. This leads to the expectation, that at least the basis pursuit solver should be insensitive to the change of the signal type. This shall be evaluated in the following section, by testing the different algorithms with a set of standard signal models. These signal models are defined by the distribution type of the values of the coefficient vector $\boldsymbol{x}$. It is varied by (i) Benchmark, (ii) Rademacher, (iii) UniformPositive and (iv) Gauss. The position of the nonzero elements is uniformly distributed, in all cases. This should be changed in future versions, to simulate signals, which are more typical to real ones.

Again all four solver types were tested with identical settings, where $N = 400$ and $\boldsymbol{B}$: partial random cosine matrix, which should minimize other influences, since all chosen solvers seemed to work quite good with this settings.

Figure 5.13 shows the 90% phase transition curves of SDPT3, representing the group of basis pursuit solvers.[20] There is no difference between the curves, as expected, due to the uniform recovery guarantee of basis pursuit solvers.

More interesting is the question, how the other solvers behave. OMP shall represent the group of greedy solvers and LASSO shall also be tested again.

Figure 5.14 depicts, that the recovery of coefficient vectors with Gaussian distributed amplitudes, delivers the best results. The curve, representing the recovery of coefficient vectors with uniformly positive distributed amplitudes, lies a bit below. Both curves are better than the referenced curve of SDPT3. But the signal models with amplitudes, being one (Benchmark) and plus, minus one (Rademacher), seem to challenge the solver any more. The recovery results are clearly below the others (also the reference), which is interesting, due to the easy structure of the signals.

The following simulation was motivated in section 5.3 *Influence of the measurement matrix*, where the results of LASSO already indicated an influence of the distribution of $\boldsymbol{x}$ and showed a strange peak at $\delta = 0.8$. The settings were as before, with $N = 400$ and $\boldsymbol{B}$, being of the type of partial random cosine matrices.

The simulation results are plotted in Figure 5.15, which shows interesting curves. There was a clear dependence expected, but the figure shows four curves, lying in the same region and crossing each other several times. All curves are below the SDPT3 reference curve and three of them show a peak in the region around the undersampling rate of 0.8, but reach the standard sampling point. The comparison of the green curve, representing the Gaussian distributed coefficient vector and a partial random cosine measurement

[20] Actually SDPT3 and SeDuMi were tested here, and delivered similar results.

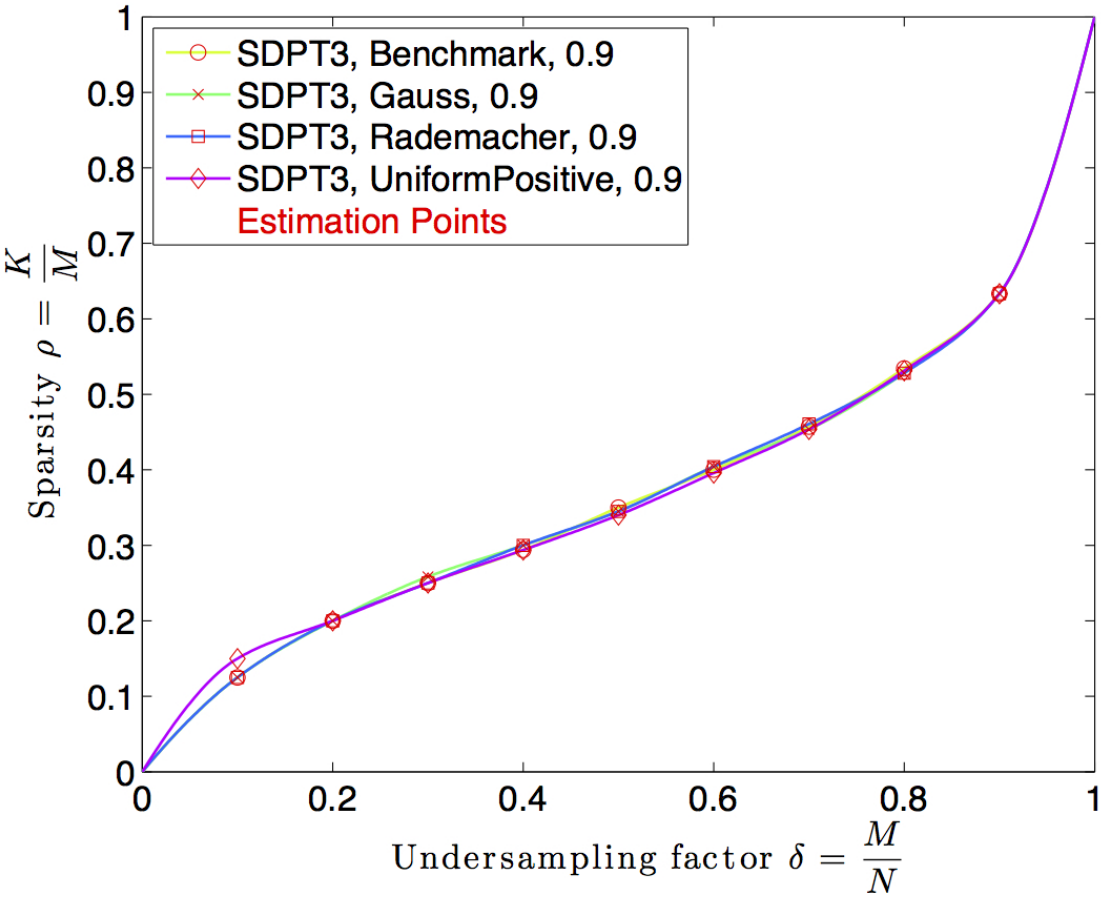


Figure 5.13.: SDPT3 shows no difference between the recovery of different signal models. Settings: $N = 400$, $\mathbf{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves, AmpDist: Benchmark, Gauss, Rademacher, UniformPositive.

matrix, with the green curve in Figure 5.11, created with the same settings, but days before, offers an important information. Despite, both curves were created with the same settings, their simulation results show a large deviation at the estimation point at $\delta = 0.8$, while the residual estimation points merge. Figure 5.16 focuses the mentioned area. This indicates a worse numerical stability of LASSO and the existence of an unknown influence at this point. It enforces the future redo of all LASSO simulations, with the parameter *ProblemSets* $\gg 200$, to validate the results, concerning the LASSO solver. The dependence of LASSO on the change of the signal model, keeps an open question, too.

This outcome motivated to simulate some experiments again, with an increased value of the *ProblemSets* parameter. But the increase of the parameter *ProblemSets*, massively increases the required computational time. This is why no complete simulations could

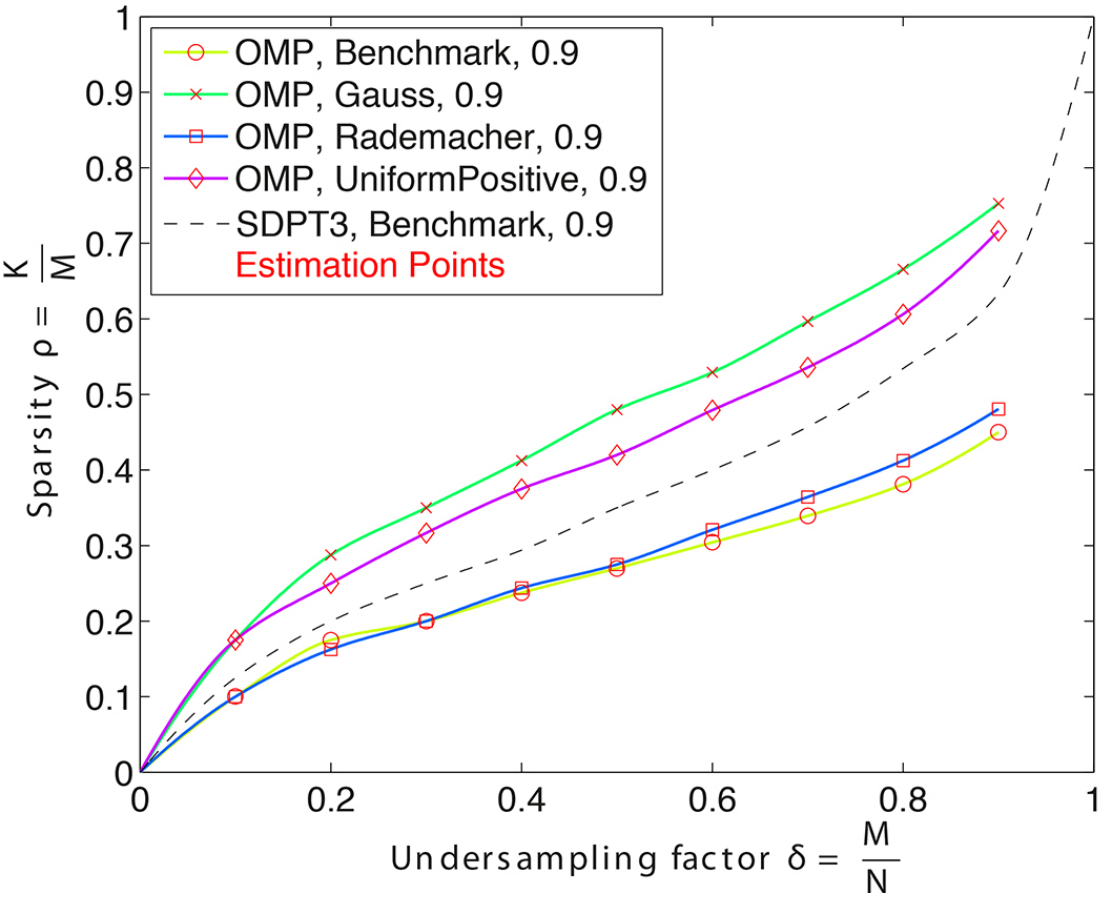


Figure 5.14.: Comparison of the 90% curves of OMP, while varying the distributional types of the coefficient vector. Settings: $N = 400$, $\mathbf{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves, AmpDist: Benchmark, Gauss, Rademacher, UniformPositive.

be redone, until now. But quick tests depict that the increase smoothes the curves. No surprising or structural deviations could be found. Finally, these tests indicate that the variance of experiments with *ProblemSets*= 200 is moderately high and can be decreased by simulating with an increased number of scenarios. While the results of LASSO should therefore be handled with care, the other results are expected to be confirmed. These solvers gain from a much less numerical instability as LASSO.

The last test should clear if SL0 again tends to the behavior, shown by basis pursuit or the one of the greedy methods, meaning be dependent on the variation of the signal model. Figure 5.17 shows the clear result. While SL0 seems independent to the type of the measurement matrix, it shows the same dependence to the signal type as the greedy solver OMP. The Gaussian curve is the best one, directly followed by the UniformPositive curve and far behind, the Rademacher and benchmark curves.

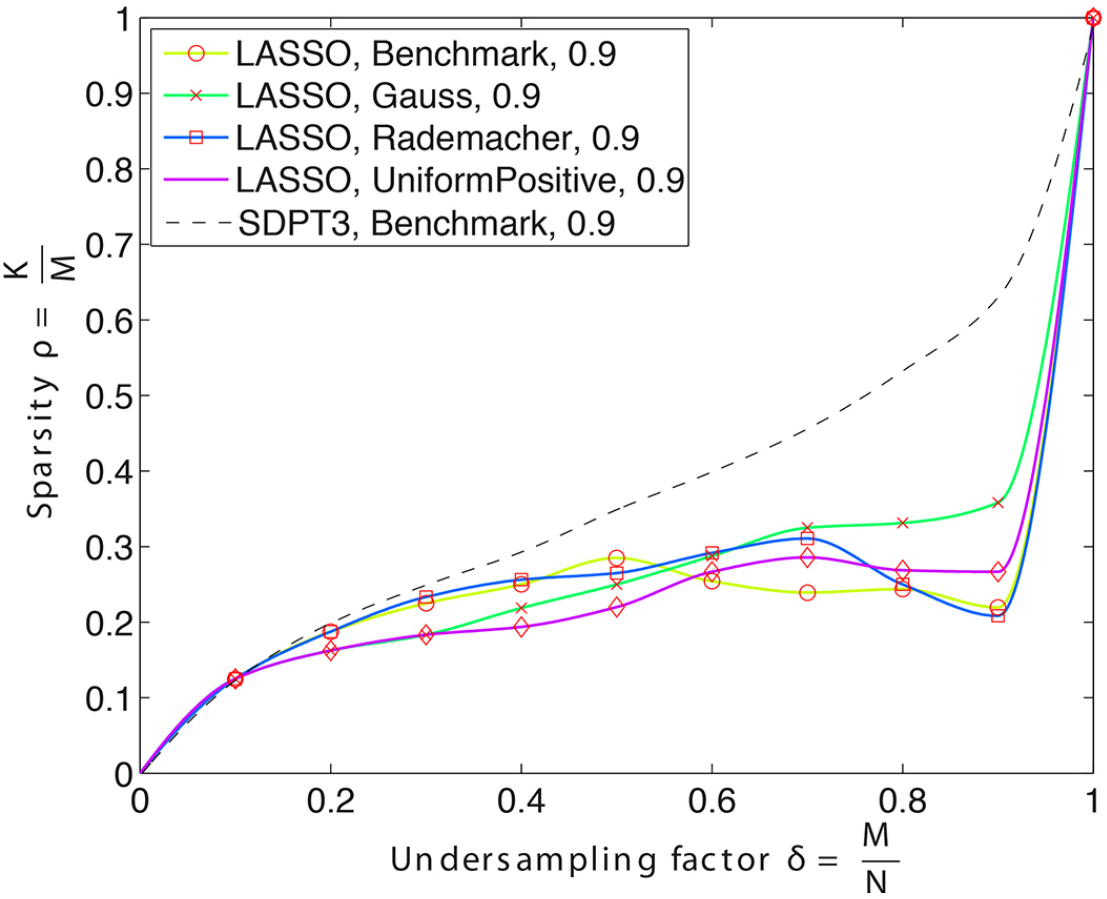


Figure 5.15.: Comparison of the 90% curves of LASSO, while varying the distributional types of the coefficient vector. Settings: $N = 400$, $\mathbf{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves, AmpDist: Benchmark, Gauss, Rademacher, UniformPositive.

It seems that recovery of coefficient vectors, with constant amplitudes, is problematic to many solvers. Future tests should isolate the problem and find out if this behavior is typical to the whole group of greedy solvers and if the problem occurs at complex valued coefficient vectors, with fixed amplitudes, either. It should be proven if signals, with Gaussian distributed magnitudes, fit best to greedy, while those, with fixed magnitudes, fit worst.

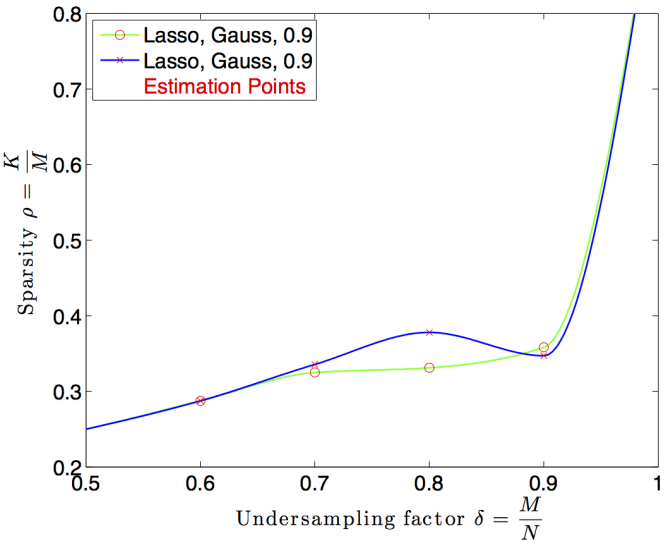


Figure 5.16.: Comparison of two LASSO curves, simulated with identical settings but distinguishing results. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, AmpDist: Gauss, 90% curves.

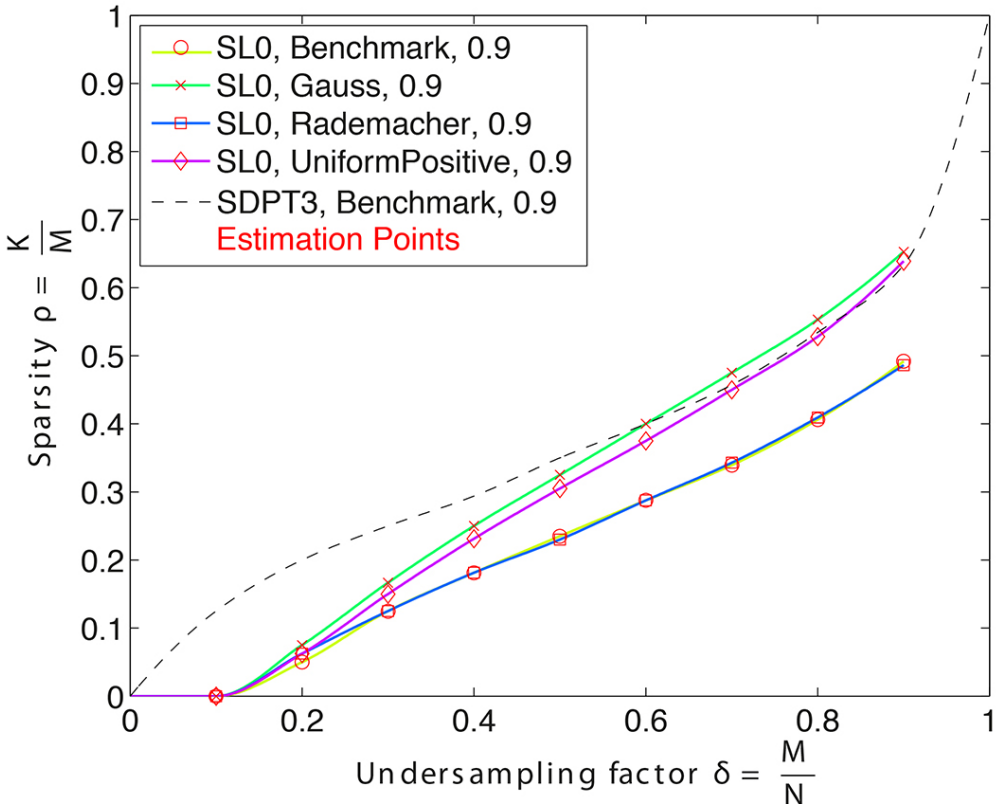


Figure 5.17.: SL0 follows the behavior of OMP and shows a dependence on different signal models. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves, AmpDist: Benchmark, Gauss, Rademacher, UniformPositive.

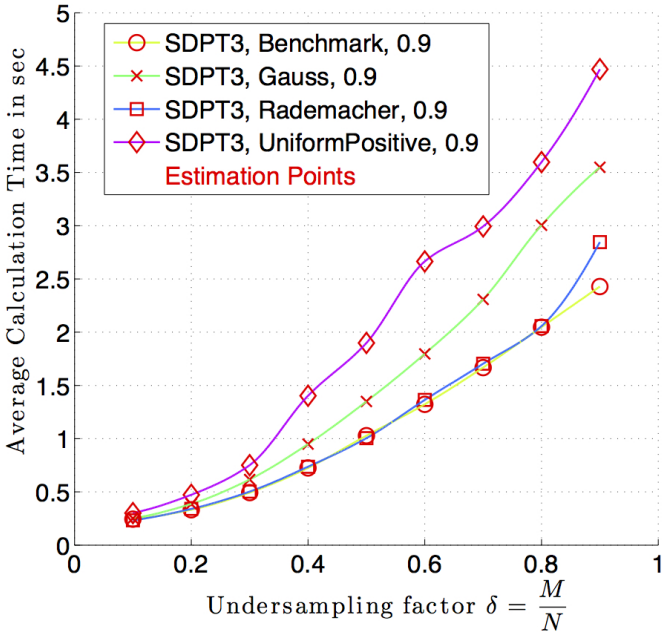


Figure 5.18.: Comparison of the computational time, required for the reconstruction step, is dependent on the signal model, while the success rates are not. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves, AmpDist: Benchmark, Gauss, Rademacher, UniformPositive.

In the end of the section, another interesting effect occurred in the experiment of SDPT3, shall be mentioned. While the different signal models did not affect the success rate of the basis pursuit solvers, it showed that nevertheless, the required computational time was affected. Figure 5.18 shows the average required time per reconstruction step, corresponding to the simulation of Figure 5.13. Although, all phase transition curves are approximately identical, the recovery of the coefficient vectors, with uniform positive distributed amplitudes, required about double the time of the recovery of the ones, with Rademacher and benchmark distributed amplitudes. The time, required for the recovery of the Gaussian signal models, lies between the extremes.

5.5. Influence of the coherence of the measurement matrix

Taking the coherence of the measurement matrix, is another approach to derive bounds, guaranteeing exact recovery. They are easy to derive and it would be interesting to know how far they can be used as an indication of how good measurement matrices are. The idea was then, to see if the adjustment of the measurement matrix $\boldsymbol{B}$ would change the behavior of the phase transition curve. Experiments with $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT and AmpDist: Gauss, were done. The use of partial random cosine matrices and the Gaussian signal model should again minimize side effects. The coherence of the matrices $\boldsymbol{B}$ were set to $\mu = min$, 0.65; 0.99; 1.0, whereby min. means doing no adjustment. An allowed deviation of $\epsilon = 0.001$ was set, which is much greater than the machine precision. This means, $\mu = 1.0$ will not exactly be reached. It is, therefore, replaced with $1 - \epsilon$ in the plots and shall indicate that $1 - \epsilon \leq \mu \leq 1$. To reach a coherence of exactly one, would require an allowed deviation ϵ, smaller than the machine precision. But this case

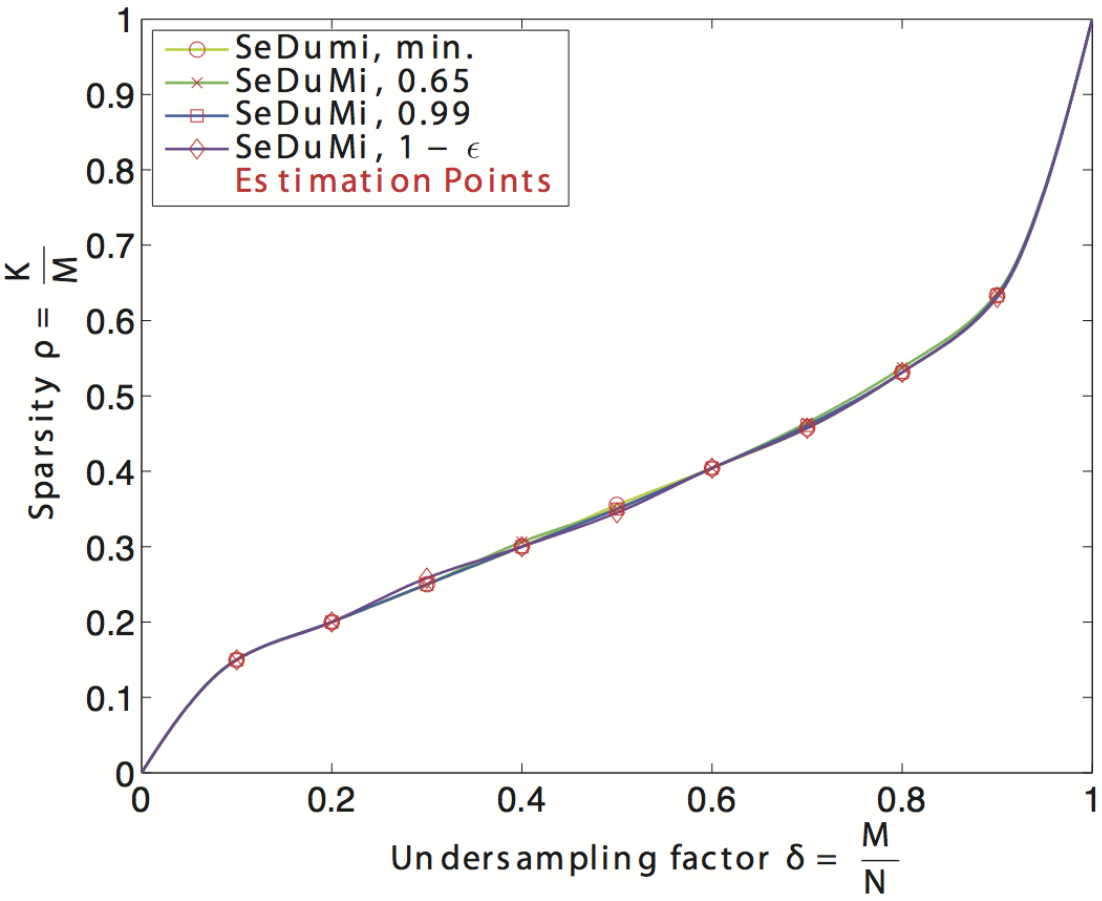


Figure 5.19.: Comparison of the success rates dependent on the coherence of the measurement matrix $\boldsymbol{B}$ and SeDuMi as solver. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, AmpDist: Gauss, Solver: SeDuMi, 90% curves.

is irrelevant, because such coherence adjustments lead to measurement matrices, with equal rows and therefore to sampled vectors $\boldsymbol{y}$, with all coefficients having exactly the same value. Recovery of a signal would be impossible.

Figure 5.19 shows the phase transition diagrams of the recovery with SeDuMi, where all three curves coincide. Basis pursuit seems immune against the adjustment of the coherence, as long as the samples in $\boldsymbol{y}$ minimally differ. But if repeating the experiment with the presence of noise, this should change and the impact of the coherence becomes clearly visible. Future simulations should evaluate the noisy case.

The average calculated coherence values of $\boldsymbol{B}$, at each estimation point, are shown in Figure 5.20. It shows the expected behavior, namely the coherence having the desired values, and in the *min.* case, being mainly defined by the size of the matrix (see eq. 2.8).

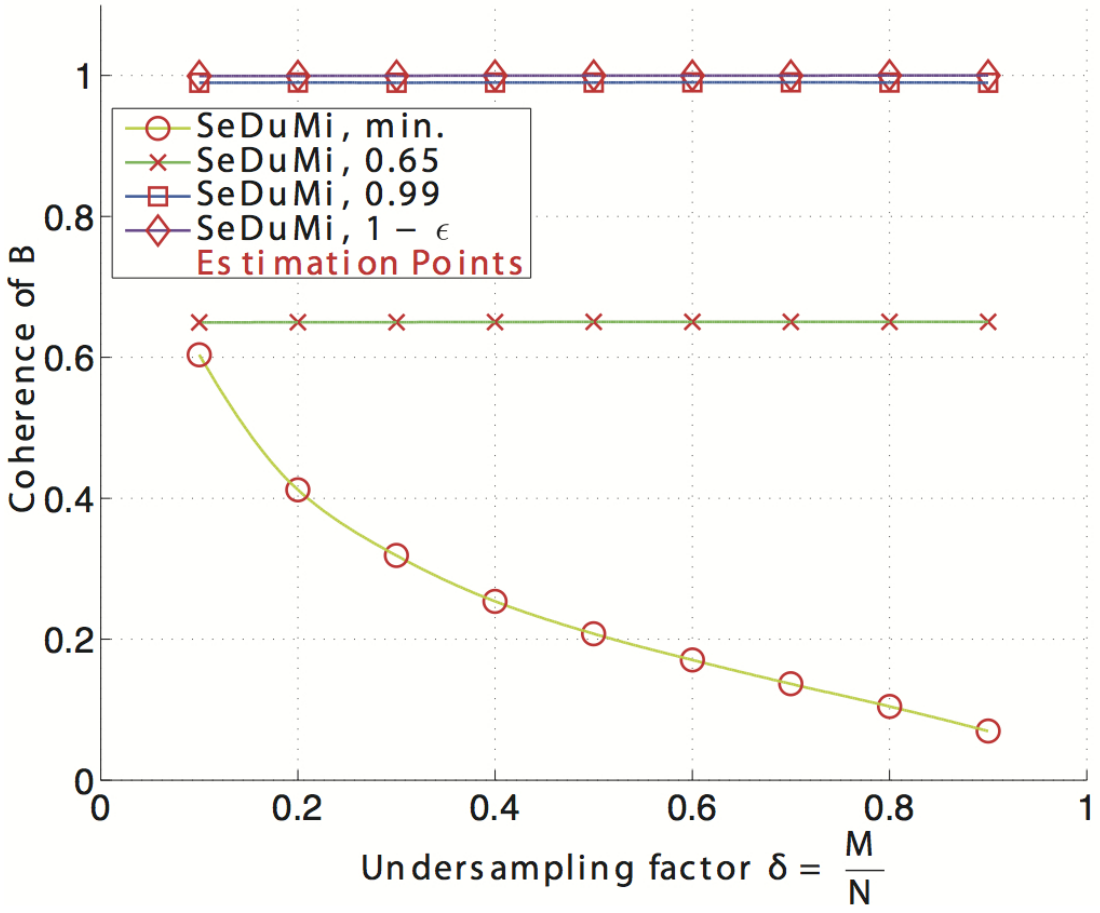


Figure 5.20.: The diagram shows the average coherence of the measurement matrix $\boldsymbol{B}$, at the estimation points. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, AmpDist: Gauss, Solver: SeDuMi.

Repeating the experiment with greedy methods roughly changes everything. The phase transition diagrams in 5.21 show a completely different view. While all settings stayed the same[21], the matching pursuit algorithm reacted highly sensible to the coherence adjustment. As long as the measurement matrix is untouched, the phase transition curve shows the expected behavior, as in Figure 5.5a. But as the matrix is adjusted by the EditSelfCoherence function of the program, the curve drops instantly. The one, representing $\mu = 0.65$, shows that recovery at the lowest estimation point is quite good. But keeping the coherence constant, while increasing the undersampling rate and therefore the size of the measurement matrix, brings the curve back to the bottom of the diagram. This constancy is against the natural behavior, where the coherence of $\boldsymbol{B}$ is decreas-

[21]The coherence values of $\boldsymbol{B}$ correspond to the ones of SeDuMi in Figure 5.20.

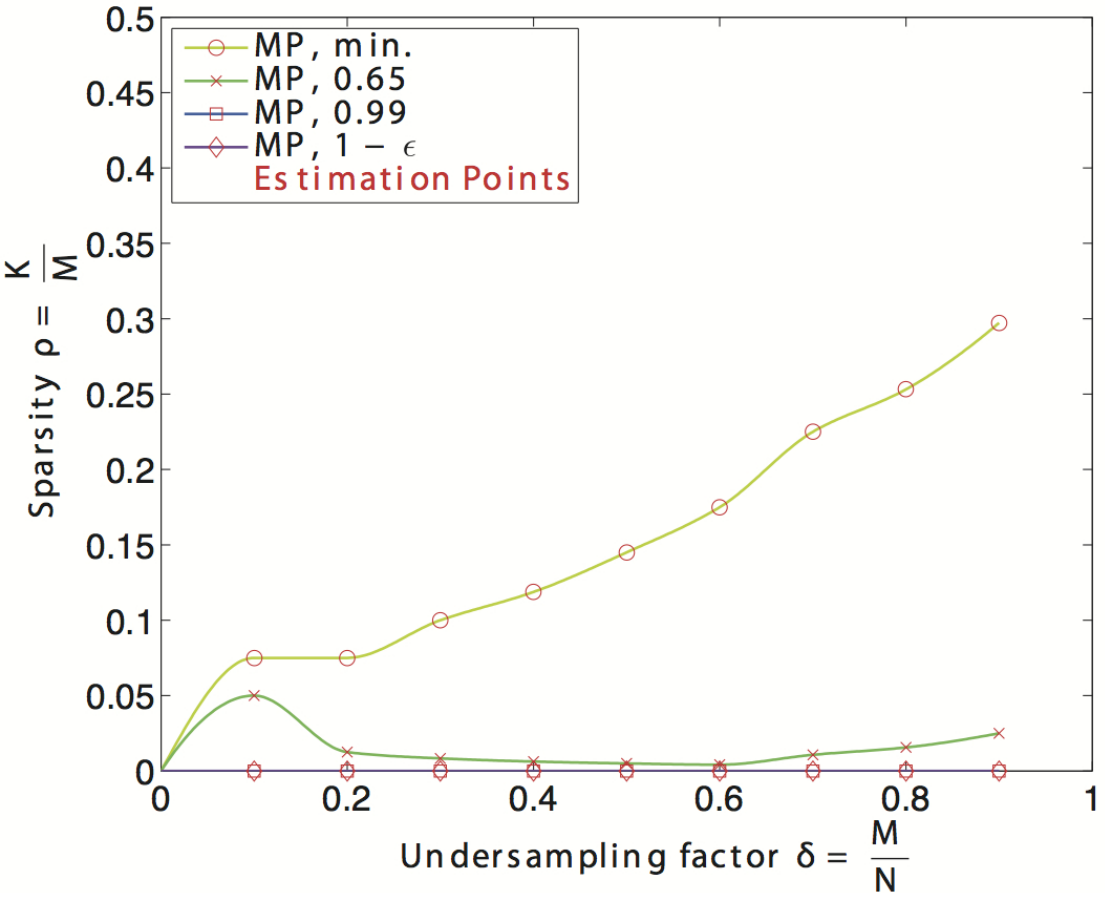


Figure 5.21.: Comparison of the success rates, with adjusted coherence values of the measurement matrix $\boldsymbol{B}$ and MP as solver. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves.

ing. Setting the coherence to a constant value, for all values of δ, increases the level of adjustment with increasing values of δ. Future simulations should test if keeping the deviation of the coherence, between two simulations, constant, leads to approximately equal shaped transition curves, but shifted. It should further be improved if this behavior comes due to the adjusted coherence or due to side effects of the used adjustment method, by implementing other algorithms, adjusting the coherence of a matrix.

Further measurements with OMP and STOMP show similar behaviors. OMP is relatively insensitive against the coherence adjustment. The success rate only falls at higher undersampling rates, where the deviation of the coherences is at its maximum. Both solver, OMP and STOMP are insensitive against a small adjustment, to the value of $\mu = 0.65$. Extreme values of the coherence lead to a drop of the OMP curve, in the transition area from the middle to the higher undersampling rates and make the recov-

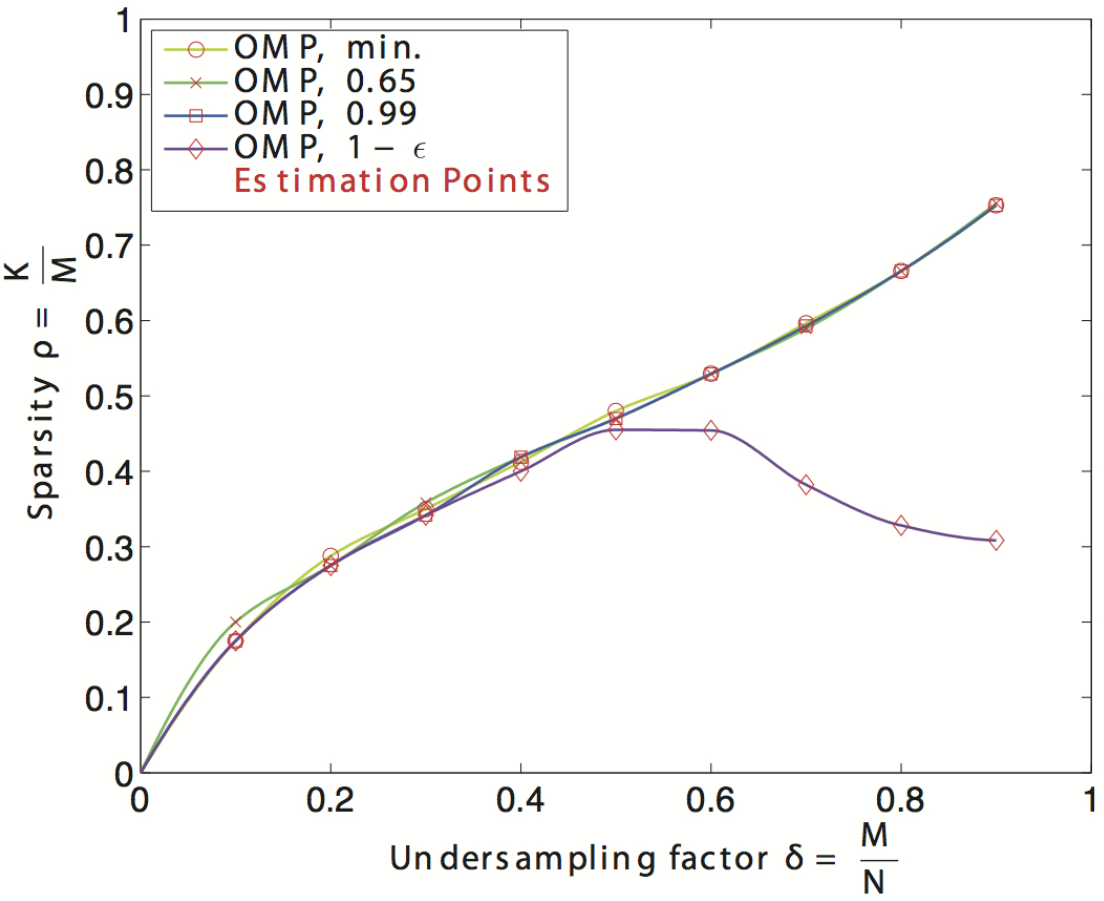


Figure 5.22.: Comparison of the success rates, with adjusted coherence values of the measurement matrix and OMP as solver. $\boldsymbol{B}$. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves.

ery with STOMP impossible. Figure 5.22 and 5.23 show the associated phase transition diagrams of OMP and STOMP.

All simulations show that greedy solvers, in opposite to the basis pursuit solvers, are highly dependent on the coherence of the measurement matrix. But the intensity of the impact varies.

5.6. Other Influences

Until now, the main focus was set on mid-values of M and treated the behavior of the middle area. While the area of high undersampling rates was proven as less important, a further simulation was done to find out more about the behavior in the area of the extreme low values of δ. It is relatively important, because there are applications, handling with

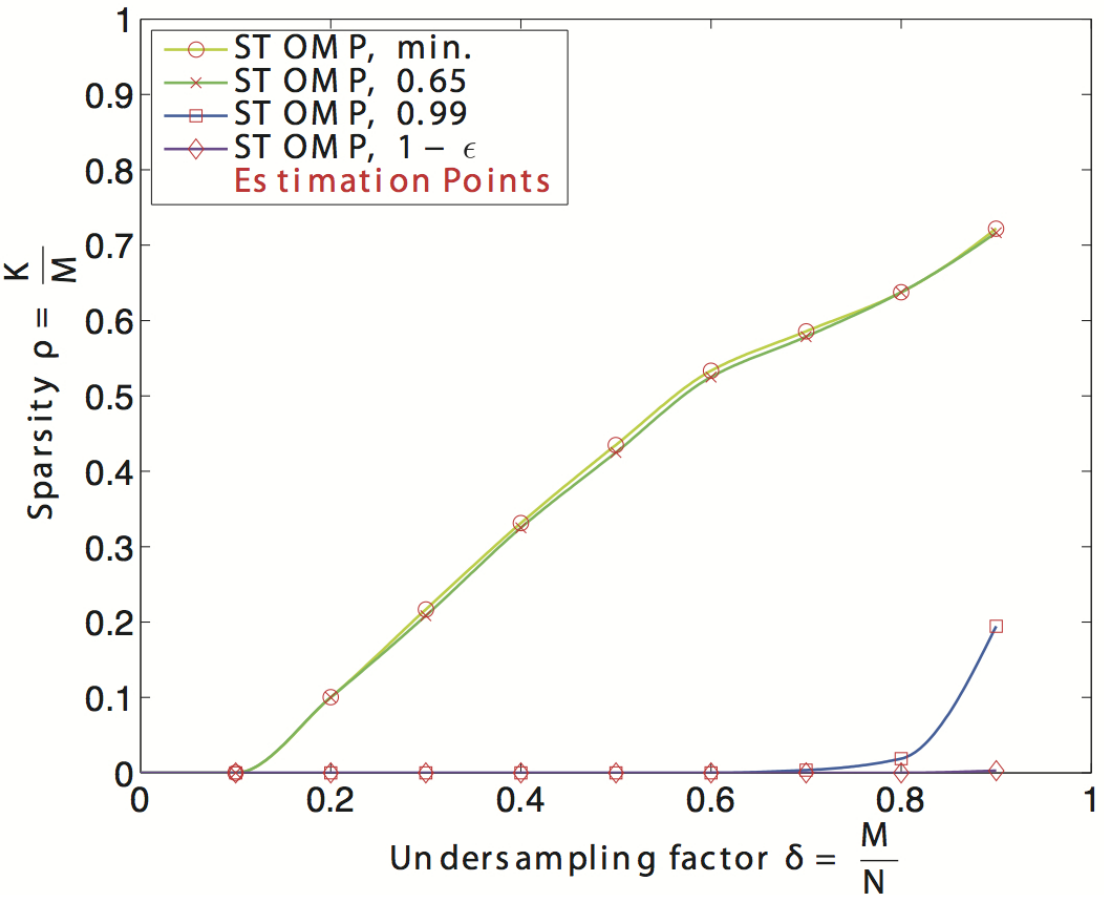


Figure 5.23.: Comparison of the success rates, with adjusted coherence values of the measurement matrix and STOMP as solver. $\boldsymbol{B}$. Settings: $N = 400$, $\boldsymbol{\Phi}$: Select, $\boldsymbol{A}$: DCT, 90% curves.

very sparse signals.

Figure 5.24 shows the phase transition diagrams of different solvers, with $N = 400$, Gaussian distributed coefficient vectors and partial random cosine measurement matrices. These settings were used to guarantee optimal behavior of all solvers. The estimation points are distributed unequal, at the points $M \in [1, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30, 35, 40, 50, 100, 200, 300, 400]$. The 90% curves are plotted.

The plot offers several information. It is obvious first, that all solvers stay at the sparsity value of zero, at the first few estimation points. The curve of the basis pursuit solver SeDuMi is the first one rising up. It is able to recover 0.1-sparse signals at an undersampling rate of $\delta = 0.025$. The greedy solvers MP and OMP follow at the next estimation point at $\delta = 0.0375$, but with a little less good sparsity. SeDuMi, at this point, shows at first a strange behavior. The curve falls and then rises again and coincides

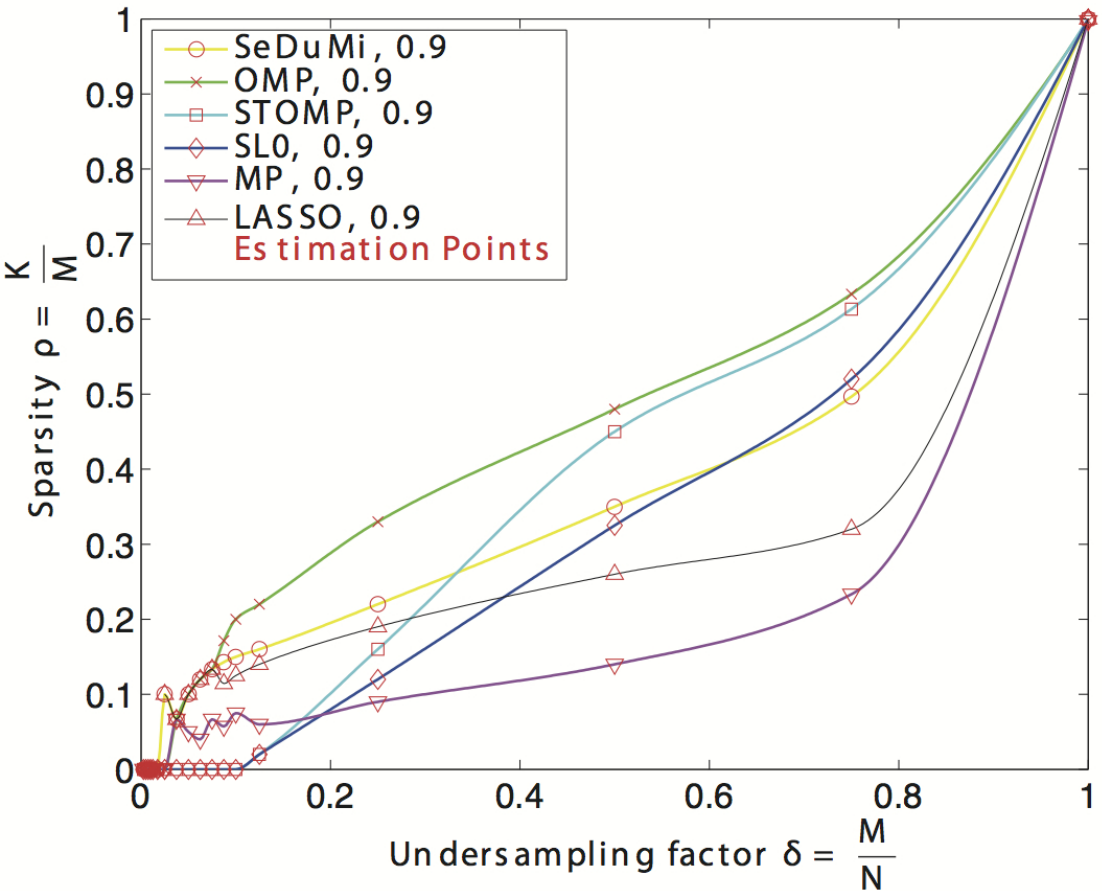


Figure 5.24.: Phase transition diagrams at very low undersampling rates, showing the 90% curves. Settings: $N = 400$, AType = DCT, BType = Select and AmpDist = Gauss.

with the one of OMP, until $\delta = 0.075$. The following behavior is known from former simulations and at the moment, out of interest, because it treats the middle area. This fall is interesting, since MP and LASSO show the same behavior. Especially the curves of MP show several falls and rises, before it gets constantly rising at about $\delta = 0.15$. The point is that, in this area, a small increase of the number of measurements M, often not leads to an increase of the largest possible sparsity K, but a decreased sparsity factor ρ and the observed *zig-zag* form. It only occurs in the lower undersampling rate areas, where an increase of M has the largest impact on ρ.

The curve of OMP does not show such a fall, which comes due to its high rise, up to moderately higher values of δ. That OMP is not totally immune to this, can be seen when evaluating its rise, which has no constant shape.

The curves of SL0 and STOMP do not have this zig-zag form, which is easy to explain. They stay at a sparsity of zero, until moderately higher undersampling rates of about

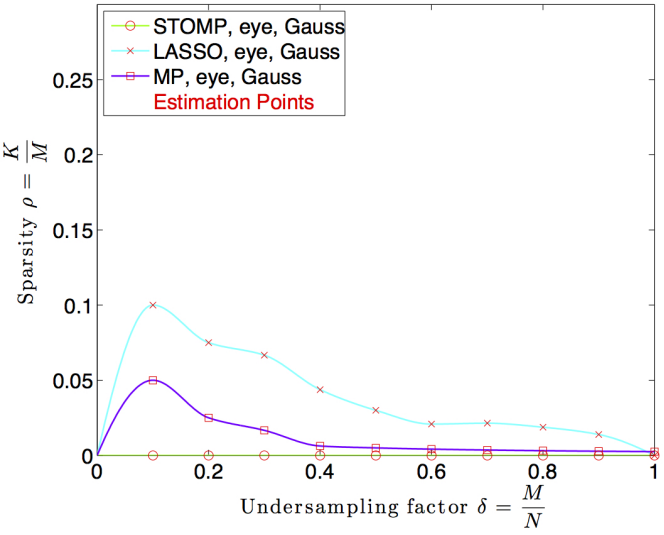


Figure 5.25.: Comparison of the success rates of STOMP, MP and LASSO, with random Gaussian measurement matrices $\boldsymbol{B}$. Settings: $N = 400$, $\boldsymbol{\Phi}$: Gaussian, $\boldsymbol{A}$: eye, 90% curves and different signal models.

$\delta = 0.1$, which makes them, in comparison to the others, useless for applications, working with very sparse signals.

At last the curve of LASSO shall be mentioned, although its outcomes are proven as relatively unreliable. It starts at $\delta = 0.025$, because the used implementation of SparseLab constantly produced errors, when sampling at lower rates. It coincides, in this area, with the curve of SeDuMi and shows the observed zig-zag form, too. But, consider these results as being an indication on a special behavior, instead of a reliable statement.

The simulations turned out that there is a set of combinations of settings, which lead to tremendously worse results, were the signals can more or less not be recovered. The combination of a selection matrix and the identity matrix of course leads to an unusable measurement matrix $\boldsymbol{B}$. This also makes the basis pursuit solvers fail, because it leads to a selection of single random values. While this is easy to explain, a second observation is somewhat more interesting. The greedy solvers STOMP and MP and the LASSO solver, absolutely fail when used with random Gaussian measurement matrices, which can not be explained by now, but indicates an optimization to regarding the type of the measurement matrix. These observations are depicted by Figure 5.25.

Although l1_magic is implemented in the program, it was not yet named in the analysis. The solver was taken into account, but delivered strange results, which could not be reproduced and seemed to happen due to wrong parametrization. To ensure not showing incorrect phase transitions and statements, all measurements with l1_magic were left out. New measurements are in process, but not completed yet. The high adjustability of the algorithms is a big problem. Although, all algorithms were extensively tested and parameters were sought[22], which fit the most universal demands, it can not be guaranteed that these fit best, in all presented simulations. Redoing the simulations, with the

[22]These are often the standard parameters.

same greedy methods, but different implementations or configurations, could tighten, correct or even call some of the outcomes into question. But it helps to find out if the observed influences are triggered by the chosen greedy method or the particular used implementation. Even though, this thesis often refers the common names of the solving methods, instead of the names of their implementation, it can not generally be suggested on the behavior of all implementations of the particular method.

5.7. Summary

The aims of this chapter was the validation of the program PTD Builder and the evaluation of the impact on the phase transition diagram of different influencing factors. The diagrams of the first simulations equaled the ones of Donoho and Tanner and led to the conclusion that PTD Builder works as intended and its outcomes are reliable.

Further experiments led to a large range of outcomes. The most important and interesting ones are:

- The growth of N leads to a shrinking width of the transition zone, which tends to zero for $N \to \infty$.
- The success of the reconstruction step is highly dependent on the choose of the solver. While the Basis pursuit solvers are relatively easy to handle and their results often stick to the ones predicted, the results of the others have to be differentiated. But it is, from the first experiments on, obvious, that greedy methods can be very efficient and successful.
- While the results of the different basis pursuit solvers were consistent, in manners of success, they differed, in manners of speed, anyway. SeDuMi pointed out as much faster than SDPT3.
- The greedy methods turned out as even more faster than the basis pursuit solvers. Most of them have a special area of expertise, which means that they deliver best results under certain conditions. These are even better than with the basis pursuit solvers and emphasizes their power.
- But greedy methods also turned out as very sensitive against the change of parameters, which often leads to the collapse of the success rate. This makes the use of greedy methods critical if high success rates are mandatory.
- The used value of 200 scenarios per estimation point, turned out as very low. It led to unreliable outcomes with LASSO, which turned out as numerically unstable.
- As long as noise is absent, the coherence has a lot smaller influence as predicted, when using basis pursuit, but shows at least a moderate impact, when using the other solvers.

It is not possible to make universal and precise predictions about the impact of a certain influencing factor, on a phase transition diagram. This emphasizes the importance of concrete simulations.

6. Conclusion

While the ideas of compressive sensing sound amazing and implausible to someone, knowing the Nyquist Sampling Theorem, at the first time, it points out as a very strong and probably useful technique for many applications. It is being used in many of them already and shows its power in the presented simulations. While Nyquist Sampling is absolutely unable to ensure exact recovery of undersampled signals, compressive sensing shows that recovery is often possible. But as pointed out in the first chapters, the technique suffers from a lack of success guarantee. While there are quite a lot sufficient requirements, no one is able to theoretically derive tight bounds, guaranteeing success of compressive sensing under certain conditions. The alternatively used phase transition enable such performance statements, but their creation is very time-consuming. These points turn out as a large problems and make the use of compressive sensing complicated.

The creation of PTD Builder took about a third of the time, deployed for this thesis. It was stamped by a relatively fast programming part and a long search for suitable parameter sets and solvers. These had to represent the different solution approaches and should follow the typical behavior of the particular group. Igor Carron, for example, lists over fifty different implementations on his homepage [6]. In the end, the choice was affected by different factors, as for example the publicity, generality and the number of references in other publications. The range of possible parameters was not even smaller. A convenient set of signal models and measurement matrices had to be found.

The first simulations, proved the functionality of the program and made its outcomes reliable. These depict, that the choice of a solver is the most relevant and adjustable parameter at the moment. On the one hand, there are the basis pursuit solvers. These acted very insensitive to parameter changes and showed consistent results over the different simulations. They often followed the predictions, made in the theoretical part and were relatively easy to handle. This makes them useful for theoretical considerations, while the high computational cost strongly limits their usage in practical applications. The large group of greedy and other fast methods (as LASSO and SL0) on the other hand, was very hard to grasp. The aggravating factor was, that these methods often reacted differently to a certain change of parameters. It was just not possible to infer on the behavior of one solver from previous results, although the experiments were limited to very few parameter sets. It came apparent, that all greedy methods are highly sensitive against the change of a single parameter. But many of them showed a special area of expertise, where their reconstruction results were extremely good. This is, besides their high speed, the main factor, why they are preferentially used in practical applications.

After finalizing all simulations, summarized in this thesis, some topics still remain open and should be answered in the future.

- Examine the behavior of further solvers, recovering signals under the mentioned conditions.
- Find answers to open questions. Especially, theoretical explanations of the behavior of the greedy solvers are interesting.
- Develop the program by adding new features, as for example the implementation of noisy measurements, the ability to use further distributional types for the position of the nonzero elements in the coefficient vectors, the creation of an easy to use graphical interface or the addition of new signal models, solvers and other parameters.
- Consider noise in further simulations and compare these results to the actual ones. Does the existence of noise rise the range of open questions? Examine the influence of the coherence of $\boldsymbol{B}$ in the noisy case.
- Complete open simulations, as for example all LASSO simulations should be redone with an increased number of scenarios.
- Find out if the outcomes are dependent on the particular algorithm or the actual implementation. This could in a first step be tested by redoing experiments with identical settings, but different implementations of the solvers.

All in all compressive sensing is a very fast growing field of science. To become a widespread used technique, key elements are the deployment of tight bounds, to predict on the success of the reconstruction step and the further development of fast and solid algorithms. The outcomes of the experiments emphasize, that especially the prediction of their behavior is, until now, nearly impossible. This makes it essential for developers to have simulation programs, to examine the potential of compressive sensing under concrete conditions and find suitable algorithms and settings. It is further very important for scientists to verify theoretical results. The program PTD Builder is perfectly cut out for these tasks and benefits from its newly developed fast algorithm.

A. Bibliography

[1] Ayaz, U. and Rauhut, H. (July, 2011). Nonuniform Sparse Recovery with Subgaussian Matrices. *Electronic Transactions on Numerical Analysis*, 41:167–178.

[2] Baraniuk, R. G. (2007). Compressive Sensing. *Lecture Notes in IEEE Signal Processing Magazine*, 24(4):118–120.

[3] Candès, E. and Romberg, J. (October, 2005). l_1_MAGIC : Recovery of Sparse Signals via Convex Programming (User's Guide). Retrieved: January 21, 2015 from `http://users.ece.gatech.edu/justin/l1magic/downloads/l1magic.pdf`.

[4] Candès, E., Romberg, J., and Tao, T. (2006). Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information. *IEEE Trans. Inform. Theory*, 52(2):489–509.

[5] Candès, E. and Wakin, M. B. (March, 2008). An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):21–30.

[6] Carron, I. (2009). Compressive Sensing: The Big Picture. Retrieved: March 09, 2015 from `http://sites.google.com/site/igorcarron2/cs`.

[7] Daubechies, I., DeVore, R., Fornasier, M., and Güntürk, C. S. (2010). Iteratively Re-weighted Least Squares Minimization for Sparse Recovery. *Commun. Pure Appl. Math.*, 63(1):1–38.

[8] Davenport, M. A., Duarte, M. F., Eldar, Y. C., and Kutyniok, G. (2012). Introduction to Compressed Sensing. In *Compressed Sensing: Theory and Applications*. Cambridge University Press, Retrieved: January 21, 2015 from `http://users.ece.gatech.edu/~mdavenport/publications/ddek-chapter1-2011.pdf`.

[9] Donoho, D. L. (2006). Compressed Sensing. *IEEE Trans. Inform. Theory*, 52(4):1289 – 1306.

[10] Donoho, D. L., Stodden, V., and Tsaig, Y. (March, 2007). About SparseLab (User's Guide). Retrieved: January 21, 2015 from `http://sparselab.stanford.edu/SparseLab_files/Documentation_files/AboutSparseLab.pdf`.

[11] Donoho, D. L. and Tanner, J. (2009). Observed Universality Of Phase Transitions In High-dimensional Geometry, With Implications For Modern Data Analysis and Signal Processing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367:4273–4293.

[12] Donoho, D. L. and Tanner, J. (May, 2010). Precise Undersampling Theorems. *IEEE Proceedings*, 98(6):913–924.

[13] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (July, 2004). Least Angle Regression. *Annals of Statistics (with discussion)*, 32(2):407–499.

[14] Gilbert, A. C. and Tropp, J. A. (December, 2007). Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit. *IEEE Trans. Inform. Theory*, 53(12):4655–4666.

[15] Grant, M. and Boyd, S. (2008). Graph implementations for nonsmooth convex programs. In Blondel, V., Boyd, S., and Kimura, H., editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited. Retrieved: January 21, 2015 from `http://stanford.edu/~boyd/graph_dcp.html`.

[16] Inc., C. R. (August, 2012). CVX: Matlab Software for Disciplined Convex Programming, version 2.0. Retrieved: January 21, 2015 from `http://cvxr.com/cvx`.

[17] Kunis, S. and Rauhut, H. (2008). Random Sampling of Sparse Trigonometric Polynomials II - Orthogonal Matching Pursuit versus Basis Pursuit. *Found. Comp. Math. 8(6) p. 737-763*, 8(6):737–763.

[18] Kutyniok, G. (October 2, 2014). Compressed Sensing. Retrieved: November 15, 2014 from `http://www.math.tu-berlin.de/fileadmin/i26_fg-kutyniok/Kutyniok/Papers/CompressedSensingDMV.pdf`.

[19] Mallat, S. and Zhang, Z. (December, 1993). Matching Pursuit with Time-Frequency Dictionaries. *IEE Trans. Signal Processing*, 41(12):3397–3415.

[20] Mohimani, G. H., Babaie-Zadeh, M., and Jutten, C. (July, 2007). Fast Sparse Representation Based on Smoothed ℓ^0 Norm. In Davies, M., James, C., Abdallah, S., and Plumbley, M., editors, *Independent Component Analysis and Signal Separation*, volume 4666 of *Lecture Notes in Computer Science*, pages 389–396. Springer Berlin Heidelberg, Retrieved: March 25, 2015 from `http://ee.sharif.edu/~SLzero/SL0_ICA07.pdf`.

[21] Rauhut, H. (2008). On the Impossibility of Uniform Sparse Reconstruction using Greedy Methods. *Sampl. Theory Signal Image Process*, 7(2):197–215.

[22] Rauhut, H. (2010). Compressive Sensing and Structured Random Matrices. In *Theoretical Foundations and Numerical Methods for Sparse Recovery*, volume 9 of *Radon Series Comp. Appl. Math.*, pages 1–92. deGruyter, Retrieved: January 14, 2015 `http://rauhut.ins.uni-bonn.de/LinzRauhut.pdf`.

[23] Rauhut, H. (November, 2007). *Sparse Recovery*. Habilitationsschrift, Universität Wien, Retrieved: December 27, 2014 from `http://rauhut.ins.uni-bonn.de/HabilStart.pdf`.

[24] Reimers, U. (1997). *Digitale Fernsehtechnik*. Springer-Verlag.

[25] Sturm, J. F. (1999). Using SeDuMi 1.02, a MATLAB*toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653.

[26] Tillmann, A. M. and Pfetsch, M. E. (2014). The Computational Complexity of the Restricted Isometry Property, the Nullspace Property, and Related Concepts in Compressed Sensing. *IEEE Trans. Inform. Theory*, 60(2):1248–1259.

[27] Tropp, J. A. (October, 2004). Greed is Good: Algorithmic Results for Sparse Approximation. *IEEE Trans. Inform. Theory*, 50(10):2231–2242.

[28] TU Ilmenau (January 31, 2015). Universitäts Rechenzentrum: Advanced Computing. Retrieved: January 31, 2015 from `https://www.tu-ilmenau.de/it-service/struktureinheiten/advanced-computing/`.

[29] Tütüncü, R., Toh, K., and Todd, M. (2002). Solving semidefinite-quadratic-linear programs using SDPT3. In *Mathematical Programming*, Ser. B, 95 (2003), pages 189–217. Springer Verlag.

B. List of Figures

2.1. Illustration of ℓ_p-balls with $p = 1, 2, \infty, \frac{1}{2}$ in $\mathbb{R}^2$. In: Davenport, M.A., Duarte, M.F., Eldar, Y.C. and Kutyniok, G. (2012) *An Introduction To Compressive Sampling.* IEEE Signal Processing Magazine [21]. p. 5. . . . 8
2.2. Illustration of the intersection of ℓ_p-balls with the line of solutions. In: Davenport, M.A., Duarte, M.F., Eldar, Y.C. and Kutyniok, G. (2012) *An Introduction To Compressive Sampling.* IEEE Signal Processing Magazine [21]. p. 6. . . . 9

3.1. Example of a continuous phase transition diagram. In: Donoho, D. L., Tanner, J. (2009). Observed Universality Of Phase Transitions In High-dimensional Geometry, With Implications For Modern Data Analysis and Signal Processing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367:4280. . . . 14
3.2. Phase Transition Diagram of Donoho and Tanner. In: Donoho, D.L. and Tanner, J. (2010). Precise Undersampling Theorems. *IEEE Proceedings*, 98(6):916, edited by Martin Herrmann. . . . 16
3.3. Phase transition diagram showing the dependency on $\boldsymbol{B}$. In: Donoho, D.L. and Tanner, J. (2010). Precise Undersampling Theorems. *IEEE Proceedings*, 98(6):917, edited by Martin Herrmann. . . . 17
3.4. Comparison of PTDs, where the signal length N is varied. In: Donoho, D.L. and Tanner, J. (2010). Precise Undersampling Theorems. *IEEE Proceedings*, 98(6):919 . . . 18

4.1. Directory tree of PTD Builder. . . . 23
4.2. Illustrates the construction of a phase transition diagram. . . . 29

5.1. PTD: Validation of PTD Builder. . . . 38
5.2. PTD: Comparison of identical PTDs, except different signal lengths N. . . 39
5.3. PTD: Comparison of phase transition zones of PTDs with different signal lengths N. . . . 40
5.4. PTD: Scaling of Figure 5.3 41
5.5. PTD: Comparison of different solving algorithms. . . . 42
5.6. PTD: Comparison of SDPT3 and SeDuMi 43
5.7. PTD: Comparison of the curves of basis pursuit and greedy. . . . 44
5.8. PTD: Comparison of the phase transition zones of SDPT3 and LASSO. . . 46
5.9. PTD: Curves of SeDuMi, with different measurement matrices. . . . 47
5.10. PTD: Curves of OMP, with different measurement matrices. . . . 48

5.11. PTD: Curves of LASSO, with different measurement matrices. 49
5.12. PTD: Curves of SL0, with different measurement matrices. 50
5.13. PTD: Curves of SDPT3, with different signal models. 52
5.14. PTD: Curves of OMP, with different signals. 53
5.15. PTD: Curves of LASSO, with different signal models. 54
5.16. PTD: Comparison of LASSO curves with identical settings. 55
5.17. PTD: Curves of SL0, with different signal models. 55
5.18. PTD: Computational time required by SDPT3, when the signal model is varied. 56
5.19. PTD: Curves of SeDuMi, with adjusted coherence of $\boldsymbol{B}$. 57
5.20. PTD: Average coherence curves of $\boldsymbol{B}$. 58
5.21. PTD: Curves of MP, with adjusted coherence of $\boldsymbol{B}$. 59
5.22. PTD: Curves of OMP, with adjusted coherence of $\boldsymbol{B}$. 60
5.23. PTD: Curves of STOMP, with adjusted coherence of $\boldsymbol{B}$. 61
5.24. PTD: Evaluation of the behavior at very low undersampling rates. 62
5.25. PTD: Curves of STOMP, MP and LASSO with random Gaussian measurement matrices $\boldsymbol{B}$. 63

E.1. PTD: Comparison of Rademacher and Gaussian signal models and SDPT3 as solver. 77

C. List of Tables

3.1. Listing of the parameters being used in Figure 3.2 15

4.1. List of all possible parameters, a short description and the standard values for PTD Builder. 26
4.2. Troubleshooting 33
4.3. Troubleshooting 34
4.4. Technical details of the massive parallel computer clusters of the Ilmenau University of Technology. 35

E.1. List of all measured points, used in Figure E.1 76

D. Acronyms

CentOS	Community ENTerprise Operating System. Operating System for Computers, based on Linux
CS	Compressive Sensing
CVX	Matlab Software for Disciplined Convex Programming
IRWLS	Iteratively Reweighted Least Squares
LARS	Least Angle Regression
LASSO	Least Absolute Shrinkage and Selection Operator
Mac OS X	Operating System for Computers from Apple Inc.
MATLAB	MATrix LABoratory, Numeric Programming Software
MP	Matching Pursuit
MOSEK	MATLAB Software for Optimization Tasks
NSP	Null Space Property
OMP	Orthogonal Matching Pursuit
PTD	Phase Transition Diagram
PTD Builder	MATLAB Software for the Creation of Phase Transition Diagrams
RIP	Restricted Isometry Property
RNG	MATLAB Random Number Generator
SeDuMi	Self Dual Minimization Algorithm
SDPT3	MATLAB Software for Semidefinite-Quadratic-Linear Programming
SL0	Smoothed L0
SparseLab	MATLAB Software Package, Designed to Find Sparse Solutions to Systems of Linear Equations
SR	Success Rate
STOMP	Stagewise Orthogonal Matching Pursuit

Symbols

$\boldsymbol{A}$	Transformation Matrix
$\boldsymbol{B}$	Measurement Matrix
δ	Undersampling Factor
K	Number of Nonzero Elements of $\boldsymbol{x}$
M	Number of Samples
N	Length of the Coefficient Vector $\boldsymbol{x}$
$\boldsymbol{\Phi}$	Kernel Matrix
ρ	Sparsity
$\boldsymbol{s}$	Signal in its Typical Domain
$\boldsymbol{x}$	Coefficient Vector
$\boldsymbol{x}^*$	Recovered Coefficient Vector
$\boldsymbol{y}$	Sampled Signal or Measurement Vector

E. Appendix

Comparison of coefficient vectors with Rademacher and Gaussian distributed values

To validate the algorithms of the newly developed program PTD Builder, the presented simulations of Donoho and Tanner were redone. Unfortunately, a small mistake happened and the distribution type of the coefficient vectors differ, between the simulation in [12] and the one in chapter 5.1 *Validation of the algorithm*. The following simulation was done to show that this mistake does not lead to a mistake and the validation can nevertheless be done on the basis of Figure 5.1.

Therefore, two simulations with $N = 400$, partial random cosine measurement matrices and SDPT3 as solver were done. Both differ in the distribution type of their coefficient vectors, which are Rademacher and Gaussian. The exact sparsity values ρ, of both simulations, are listed in Table E.1, the plot is shown in Figure E.1. The differences between the simulations are summed up and have a total value of 0.0221, which is an average deviation of $\overline{d} = 8.179E-4$ per estimation point. The plot also depicts that the particular curves of both simulations coincide. This confirms the former statement, that there is no structural difference between both PTDs and justifies the ability to verify the algorithms of PTD Builder on the basis of the done simulations.

M / sr	ρ **(Rademacher)**	ρ **(Gaussian)**
40/0.1	0.25	0.25
40/0.5	0.2	0.2
40/0.9	0.15	0.15
80/0.1	0.3	0.3
80/0.5	0.2375	0.25
80/0.9	0.2	0.2125
120/0.1	0.3417	0.3333
120/0.5	0.2917	0.3
120/0.9	0.25	0.2417
160/0.1	0.375	0.3813
160/0.5	0.3375	0.3438
160/0.9	0.3	0.3
200/0.1	0.43	0.425
200/0.5	0.385	0.385
200/0.9	0.345	0.345
240/0.1	0.4833	0.4833
240/0.5	0.4417	0.4375
240/0.9	0.3958	0.4
280/0.1	0.5464	0.5393
280/0.5	0.5	0.5071
280/0.9	0.4607	0.4607
320/0.1	0.6188	0.6219
320/0.5	0.575	0.575
320/0.9	0.5313	0.5344
400/0.1	0.6389	0.7278
400/0.5	0.6833	0.6833
400/0.9	0.6389	0.6361
Abbr.: sr.: success rate		

Table E.1.: List of all measured points, used in Figure E.1

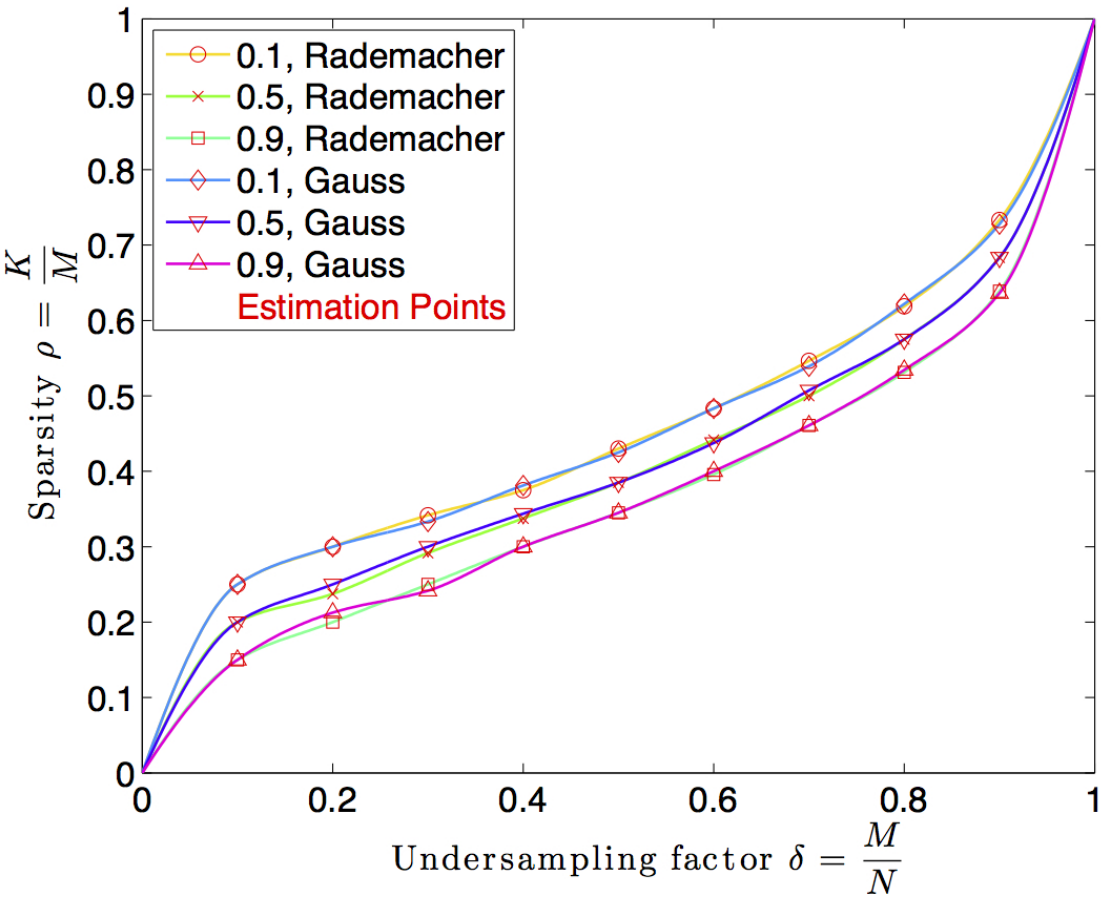


Figure E.1.: Comparison of Rademacher and Gaussian distributed coefficient vectors. Settings: $N = 400$, $\mathbf{\Phi}$: StrictSelect, $\boldsymbol{A}$: DCT, Solver: SDPT3.

F. Eigenständigkeitserklärung (Declaration of Authorship)

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat.
Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

(The author of this thesis, declares that it was created autonomously and without using other than the stated references.
All parts which are cited (including electronic sources), directly or indirectly, are marked as such. This thesis has not been used in the same or similar forms in parts or total in other examinations.)

Bernstadt, May 8, 2015

Vereinbarung über die Verwertungsrechte an einer studentischen Arbeit

Wird eine Vereinbarung im Zusammenhang mit einer Diplomarbeit, Bachelor- oder Masterarbeit abgeschlossen, ist sie zur Studentenakte zu nehmen.

Name, Vorname der / des Studierenden (Angaben bitte in Druckbuchstaben):	Matrikel – Nr.:
Herrmann, Martin	**48388**

Studiengang:	Fakultät:
Medientechnologie	**EI**

Nach den Vorschriften des Gesetzes über das Urheberrecht (Urheberrechtsgesetz – UrhG) und des Arbeitnehmererfindergesetzes (ArbnErfG) liegen die Verwertungsrechte an einer Studien- oder Prüfungsarbeit ohne eine Vereinbarung bei der/dem Studierenden als Urheber oder Erfinder.
Andererseits kann die TU Ilmenau (zum Beispiel bei gemeinsamen Projekten mit Forschungspartnern) ein Interesse an diesen Rechten haben. Zur Klärung der damit verbundene Fragen wird folgendes vereinbart:

☐ Studienarbeit / Medienprojekt ☐ Diplomarbeit ☒ Bachelorarbeit ☐ Masterarbeit

Sonstiges: .....

Thema der Arbeit:

Phase Transition Diagrams in Compressive Sensing - Computation and Analysis

Weitere an der Arbeit Beteiligte:

Verantwortlicher Hochschullehrer:

Univ.-Prof. Dr.-Ing. Giovanni Del Galdo

Alle prüfungsrechtlichen Bestimmungen besitzen Gültigkeit.

Ansprechpartner: Transferstelle (Telefon *2512, Email: transfer@tu-ilmenau.de)

A ☐ **Alle Rechte verbleiben bei dem/der Studierenden.**
- zum Beispiel bei externen studentischen Arbeiten -
(Punkte B bis D entfallen.)

Die TU Ilmenau (in Person des verantwortlichen Hochschullehrers)
- ☐ erhält die während der Arbeit entstandenen Unterlagen (Programme, Messprotokolle etc.) zur Verwendung für eigene wissenschaftliche (nicht kommerzielle) Zwecke in Forschung und Lehre,
- ☐ darf Kopien von der Arbeit anfertigen,
- ☐ erhält auf Verlangen weitere Exemplare.

Die sonstigen Verwertungsrechte des/der Studierenden bleiben unberührt.

B ☒ Es wird ein **nicht ausschließliches Verwertungsrecht durch den/die Studierende(n) und die TU Ilmenau** vereinbart, d.h. der/die Studierende und die TU Ilmenau sind gleichermaßen berechtigt, die Ergebnisse der Arbeit zu nutzen. Im Falle einer Erfindung wird eine gesonderte Vereinbarung zwischen dem/der Studierenden und der TU Ilmenau abgeschlossen.
(Punkte C und D entfallen.)

C ☐ Aufgrund einer vertraglichen Vereinbarung hat die TU Ilmenau dafür Sorge zu tragen, dass die Rechte an der Arbeit auf einen Projektpartner übergehen. Der/die Studierende **überträgt die Rechte an den Ergebnissen seiner Arbeit auf die TU Ilmenau.** Er/sie wird im Falle einer Erfindung Arbeitnehmern der TU Ilmenau gleich gestellt.
(Punkte D entfällt.)
- ☐ Dem/der Studierenden verbleibt das Recht zur Nutzung der Ergebnisse der Arbeit für eigene wissenschaftliche (nicht kommerzielle) Zwecke in Forschung und Lehre.
- ☐ Der/die Studierende wurde über Geheimhaltungsverpflichtungen, die sich aus der vertraglichen Vereinbarung mit dem Projektpartner ergeben belehrt.

Projektpartner:

Vertragsnummer der TU Ilmenau:

D ☐ Die TU Ilmenau hat zur Weiterführung ihrer Forschungsarbeiten ein gesteigertes Interesse an den Ergebnissen dieser Arbeit. Aus diesem Grund **überträgt** der/die Studierende **die Rechte an den Ergebnissen seiner Arbeit auf die TU Ilmenau**. Er/sie wird im Falle einer Erfindung Arbeitnehmern der TU Ilmenau gleich gestellt.
- ☐ Dem/der Studierenden verbleibt das Recht zur Nutzung der Ergebnisse der Arbeit für eigene wissenschaftliche (nicht kommerzielle) Zwecke in Forschung und Lehre.

E Eine **Veröffentlichung/Weitergabe** der Arbeit darf ganz oder teilweise sowohl seitens des Studierenden als auch der TU Ilmenau
- ☐ uneingeschränkt
- ☒ nur im gegenseitigen Einvernehmen
- ☐ nur bis auf die in der **beigefügten weiteren Vereinbarung** ausdrücklich ausgenommenen Abschnitte
- ☐ erst nach dem ..... erfolgen.

Die Veröffentlichung der Zusammenfassung der Arbeit (Abstract) ist uneingeschränkt möglich.

Ilmenau, 1.12.14	Ilmenau, 3.12.14	Ilmenau, 1.12.14
Studierende(r)	Prüfungsamt (TU Ilmenau)	Verantwortlicher Hochschullehrer (TU Ilmenau)