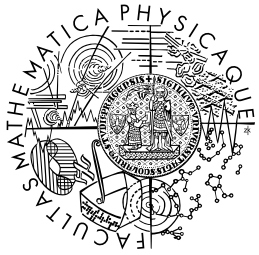


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Phuong Thao Hoang

Methods for enforcing non-negativity of solution in Krylov regularization

Department of Numerical Mathematics

Supervisor of the master thesis:	doc. RNDr. Iveta Hnětynková, Ph.D.
Study programme:	Numerical and Computational Mathematics
Study branch:	Numerical and Computational Mathematics

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to sincerely express my gratitude towards my supervisor doc. Iveta Hnětynková without who this thesis wouldn't be possible as she was constantly helping me, guiding me, and supporting me throughout the whole year. Special thanks also go to my family and friends who were always by my side, encouraging me during my studies.

Title: Methods for enforcing non-negativity of solution in Krylov regularization

Author: Phuong Thao Hoang

Department: Department of Numerical Mathematics

Supervisor: doc. RNDr. Iveta Hnětynková, Ph.D., Department of Numerical Mathematics

Abstract: The purpose of this thesis is to study how to overcome difficulties one typically encounters when solving non-negative inverse problems by standard Krylov subspace methods. We first give a theoretical background to the non-negative inverse problems. Then we concentrate on selected modifications of Krylov subspace methods known to improve the solution significantly. We describe their properties, provide their implementation and propose an improvement for one of them. After that, numerical experiments are presented giving a comparison of the methods and analyzing the influence of the present parameters on the behavior of the solvers. It is clearly demonstrated, that the methods imposing nonnegativity perform better than the unconstrained methods. Moreover, our improvement leads in some cases to a certain reduction of the number of iterations and consequently to savings of the computational time while preserving a good quality of the approximation.

Keywords: linear inverse problem, regularization, iterative methods, Krylov subspace, non-negativity

Contents

List of Symbols

The next list describes several symbols that will be later used in the document

$(.)_+$	a projection onto a set of non-negative vectors
$(.,.)$	an Euclidean inner product/dot product
α	a safety parameter
ϵ	an estimate of an Euclidean norm of the noise e
η	the standard deviation of a Gaussian white noise
$\mathbb{R}$	real numbers
$\mathcal{C}$	a condition space
$\mathcal{K}_k(A, b)$	the k-order Krylov subspace generated by a matrix A and a vector b
$\mathcal{K}_k^{shift}(A, b)$	a shifted Krylov subspace
$\mathcal{S}$	a search space
μ	the mean of a Gaussian white noise
Σ	a diagonal matrix containing singular values
σ_i	singular values
ζ	a noise level
A	a system matrix $\mathrm{R}^{m \times n}$
$A^\dagger$	the Moore–Penrose inverse
A^{-1}	an inverse of matrix A
A^T	a transpose of matrix A
b	a right-hand side
b^{exact}	an unperturbed right-hand side
$Cov(.)$	a covariance matrix
$dim(.)$	a dimension of a vector space
e	a perturbation/noise
$E(.)$	an expected value
I	an identity matrix
k_{opt}	an optimal iteration index

$r = rank(.)$	a rank of a matrix
r_k	a residuum in k-th iteration
$span$	a linear span of a vector set
U, V	unitary matrices holding left and right singular vectors, respectively
u_i	left singular vectors
v_i	right singular vectors
x	an approximate solution
x^{exact}	an exact solution
x^{naive}	a naive solution
x_0	an initial vector
x_k	an approximate solution in k-th iteration

Introduction

Inverse problems appear in many applications such as image processing, signal processing, computed tomography, geophysical prospecting, to name a few. Imagine a 3-piece scheme. The first piece is the input data which goes through the second part - some system that modifies it and ends up returning the output data - the last piece. Either the input or the system is unknown, while the rest is available. In real-world problems, we usually know a mathematical model of the system and the measured data contaminated with noise. This thesis will focus on a specific type of inverse problems that leads to a so-called Fredholm integral of the first kind [14]

$$\int_0^1 K(s,t)f(t)\,dt = g(s), \qquad s \in [0,1]. \tag{1}$$

In this example, kernel $K(.,.)$ represents the system, function $f(.)$ the unknown input data, and function $g(.)$ the measured data. However, we won't work directly with this equation. Instead, we focus on solving the approximation problem expressed by a system of linear equations

$$Ax \approx b, \tag{2}$$

where $A \in \mathbb{R}^{m\times n}$ is a matrix, $x \in \mathbb{R}^n$ is an unknown vector representing an approximation of the exact solution and $b \in \mathbb{R}^m$ is a vector with the imprecise measured data. Note that we'll only work with real numbers. We call the new problem (2) a discrete inverse problem.

Discrete inverse problems are complicated to solve as they are ill-conditioned. That means that even a small perturbation in the measured data can cause a large error in the solution. In Chapter 1 we address this issue by studying theory of inverse problems based on the books [14], [16] and the papers [5], [9], [10], [13], [22]. That includes mainly the Singular value decomposition analysis, the influence of Gaussian white noise, and the Discrete Picard condition. We explain how noise can drastically affect the solution. Hence, we can't use standard techniques for solving systems of linear equations or least squares problems here. Instead, we have to seek iterative regularization methods that stop the computations at the right moment - before noise starts to significantly affect the approximate solution. Besides this matter, we have to deal with the scalability of methods, as problems arising in image processing and other areas mentioned above usually produce large discrete inverse problems. Therefore, we don't solve them explicitly, but implicitly. A well-known class of methods that proved to work well here is the class of Krylov subspace methods studied in Chapter 2. Their great advantage is that there's only a need for matrix-vector multiplications during the computation process, which can lead to savings of computational time and memory and allows to apply the methods in a matrix-free form (if A is not available explicitly). We focused primarily on the CGLS algorithm [27] and on a phenomenon called semiconvergence [14]. This phenomenon is specific for iterative methods used on discrete inverse problems and must be carefully taken into account in order to be

able to determine high-quality approximations.

Many applications lead to inverse problems, where the exact solution has all of its entries non-negative (we speak about the non-negative solution). For example in image processing, the entries of the unknown true solution are simply some non-negative pixel values. Therefore, the second issue addressed in this paper is the requirement of non-negativity of the approximation. As we will see later in Chapter 3, this constraint proved to be hard to fulfill, however, necessary to obtain an accurate approximation and to interpret the results correctly. That is demonstrated on examples from the image reconstruction. It is well known [9] that imposing non-negativity in standard regularized Krylov subspace methods is not trivial. However, several successful approaches are known. These are for example the PRI and RSPRI algorithms proposed in [5], and the NN-FCGLS algorithm presented in the paper [9]. We summarize the main idea, properties, and correct implementation of these methods. Based on this knowledge, we then suggest a modification of the PRI algorithm leading to the reduction of the total number of iterations and the computational time.

In the last chapter, Chapter 5, we present a wide numerical study of the considered methods on 1D and 2D non-negative inverse problems from the RegTools [13] and IRTools [10] software packages for MATLAB. A part of the work was to study optimal parameter choice for each algorithm. Afterward, six methods, namely unconstrained CGLS, MCGLS, NN-FCGLS, PRI, MPRI, and RSPRI, were compared with each other. The results demonstrate the advantages and disadvantages of each particular method. At the end of the thesis, we summarize our main observations, give several final remarks and suggestions for future work.

1. Inverse problems

1.1 Ill-posed problems

Solving inverse problems is a difficult task because they belong to the ill-posed problems. At the beginning of the 20th century, french mathematician Jacques Hadamard stated three conditions for a problem to be considered well-posed (an opposite of ill-posed) [12]. These conditions are now referred to as a well-posedness in sense of Hadamard.

Definition 1 (Well-posedness in sense of Hadamard)**.** A problem is well-posed if the following conditions are fulfilled:

- **existence:** there exists a solution,
- **uniqueness:** there exists an unique solution,
- **stability:** the solution is continuously dependent on data and parameters

While it is possible to fix the first two conditions by reformulating the problem and adding extra requirements ([14], Chapter 1), to deal with the violation of the third condition we have to study the problem more deeply. It is not possible to completely fix instability, however, it is feasible to approximate the original problem by a stabler one, i.e. to regularize it. We'll discuss this topic in a later section.

Violation of stability condition means that small perturbations in either our measured data b or our system A or both can drastically change the solution. Such an example was shown in [14], Chapter 1. To simplify the issue, let us from now on only consider perturbations on the right-hand side, i.e.

$$Ax \approx b = b^{exact} + e, \tag{1.1}$$

where e is noise, which can be caused by various factors. One of the common examples is if we use a damaged tool/machine to measure our data, e.g. we take a photo with an old camera. There also exist different types of noise. In this thesis, we'll particularly talk about Gaussian white noise because of its nice properties. This will be explained shortly.

Definition 2 (Noise level)**.** A noise level of noise e is defined as follows

$$\zeta = \frac{||e||_2}{||b^{exact}||_2}. \tag{1.2}$$

Let us first showcase the complication we encounter by having noise on the right-hand side of an ill-posed problem. The exact equation would look like

$$Ax^{exact} = b^{exact}, \tag{1.3}$$

where $A \in \mathbb{R}^{m\times n}$ is a matrix, $x^{exact} \in \mathbb{R}^n$ is a vector representing our wanted solution and $b^{exact} \in \mathbb{R}^m$ is a vector with perfectly measured data. However, this flawless situation can only happen in an ideal world. More realistic would be to consider the problem (1.1). We can rewrite this equation and get

$$x = A^\dagger b = A^\dagger b^{exact} + A^\dagger e = x^{exact} + A^\dagger e, \tag{1.4}$$

where $A^\dagger$ is the Moore–Penrose inverse, i.e. the pseudoinverse, of the matrix A as we consider A a general rectangular matrix. Here we're looking for a solution x in a sense of the least squares. However, the expression $A^\dagger e$ in (1.4) can significantly spoil our solution. The smoothing effect of matrix A (a property of inverse problems' models [14], Chapter 3) and contrarily the accentuation of oscillations caused by multiplication with the matrix $A^\dagger$, and amount of the noise play a big role in this. Therefore, we will study the smoothing effect of the matrix A and properties of the noise e in the following two sections.

1.2 Singular value decomposition analysis

In the introduction, we mentioned that instead of solving the Fredholm integral of the first kind (1) we can solve a discrete approximation problem (2). This can be obtained by discretization methods such as Quadrature or Expansion Methods. Moreover, it can be shown that there exists a strong bond between the equations (1) and (2) ([14], Chapter 3). The matrix A is closely related to the kernel K and vectors x, b hold the information of the function f and the right-hand side g, respectively. Therefore a lot of properties of the continuous problem (1) are inherited by the matrix problem (2).

Before singular value decomposition (SVD) was found for matrices, there was a similar tool known for the integral formulation (1), the singular value expansion (SVE). SVE serves as a tool to obtain a better knowledge about a smoothing effect of the kernel K and an existence of the solution. If the kernel K is square integrable, i.e $\int_0^1 \int_0^1 K(s,t)\,ds\,dt < \infty$, then its SVE takes the form

$$K(s,t) = \sum_{i=1}^{\infty} \mu_i u_i(s) v_i(t), \tag{1.5}$$

where μ_i are singular values and u_i, v_i are left and right singular functions. Here $\mu_1 \geq \mu_2 \geq \cdots \geq 0$ decrease rapidly. Moreover, functions u_i and v_i create an orthonormal basis of $L_2([0,1])$ functions with an inner product defined as $\langle u_i, v_j \rangle = \int_0^1 u_i(t) v_j(t)\,dt$. Unfortunately, it is difficult to determine SVE analytically [14]. So besides a convenient manipulation on computers, this is another advantage of the matrix representation (1) as it's easier to obtain its SVD (or its part) in case that m, n are not too large.

Definition 3 (Singular value decomposition)**.** For any rectangular matrix $A \in \mathbb{R}^{m\times n}$ there exist unitary matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, i.e.

$$U^T U = I_m$$
$$V^T V = I_n,$$

and a rectangular diagonal matrix $\Sigma \in \mathbb{R}^{m\times n}$ with non-negative real numbers on the main diagonal, such that

$$A = U\Sigma V^T. \tag{1.6}$$

Another way to write the SVD decomposition is

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \tag{1.7}$$

where $r = rank(A)$. The middle expression in (1.7) is a matrix form of the SVD, whilst the latter expression is called a dyadic form of the SVD. Matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ hold vectors u_i and v_i in their columns. These vectors generate linearly independent sequences $\{u_1, u_2, ..., u_m\}$, $\{v_1, v_2, ..., v_n\}$, which define the finite-dimensional vector spaces $\mathbb{R}^m$, $\mathbb{R}^n$, respectively. On top of that, similarly as in the SVE, they are orthonormal, i.e.

$$u_i^T u_j = \delta_{ij}, \qquad v_i^T v_j = \delta_{ij},$$

where δ_{ij} is the Kronecker delta. The matrix $\Sigma \in \mathbb{R}^{m\times n}$ is a non-negative sparse matrix having its all nonzero elements on the main diagonal. These nonzero values are called singular values σ_i and they form a non-increasing sequence decaying to zero in the same way as in the SVE

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > \sigma_{r+1} = ... = \sigma_{min\{m,n\}} = 0. \tag{1.8}$$

Because the singular values with indices greater than the rank of A are all zeros, we can rewrite the matrix form of the SVD into an economic form

$$A = U_E \Sigma_E V_E^T, \qquad U_E \in \mathbb{R}^{m\times r}, V_E \in \mathbb{R}^{r\times n}, \Sigma_E \in \mathbb{R}^{r\times r}.$$

It can be shown that there is a strong connection between SVE and SVD. For inverse problems, SVE gives an insight into the smoothing effect of the kernel K, while SVD gives an insight into the smoothing effect of the matrix A. With this knowledge, we can further investigate the instability of the initial problem (1) and obtain similar conclusions and utilizations from SVE and SVD.

Replacing the matrix A in (1.4) by its SVD decomposition (1.7) in the dyadic form and using the property of unitary matrices, i.e. $U^{-1} = U^T$, $V^{-1} = V^T$, we get

$$x = A^\dagger b = V\Sigma^\dagger U^T b = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i. \tag{1.9}$$

The solution obtained in this way is called a naive solution. For inverse problems, this approximation usually notably differs from the true solution. The reason behind it is the instability of the problem (1.1) and the presence of noise and rounding errors that have a big impact on it. From (1.9) we see that x lies in a linear span of vectors v_i. These right singular vectors have more oscillations with an increasing index i as it is illustrated in Figure 1.1. Therefore, we have to be careful with multiplying these vectors by coefficients

$$u_i^T b = u_i^T b_{exact} + u_i^T e \tag{1.10}$$

in (1.9) as they determine how well our solution will behave. We'll study the behaviour of the approximate solution x for a test problem `shaw(32)` from the RegTools library [13] developed for Matlab in the next section.

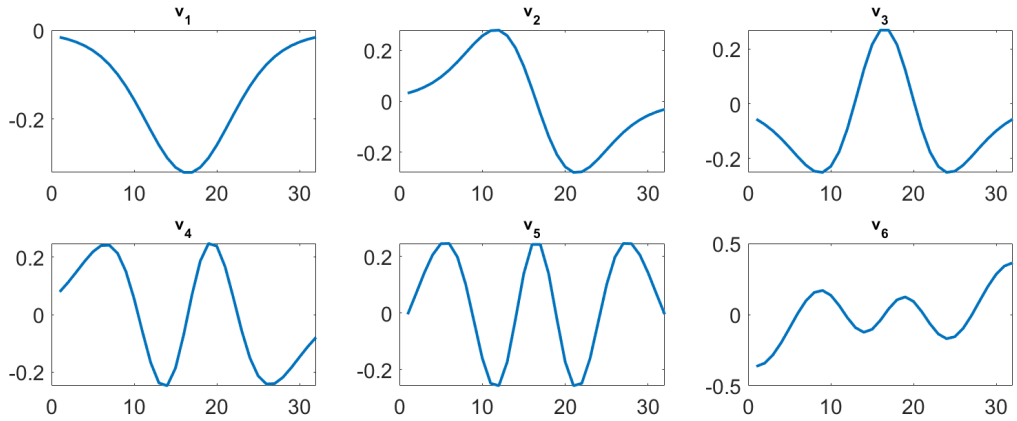


Figure 1.1: First six right singular vectors v_i of the `shaw(32)` test problem. Number of oscillations of these vectors is increasing with an increasing index i.

1.3 Gaussian white noise

As we said before, there exist many types of noise but the most common model is Gaussian white noise which models random errors. Using a technique called *prewhitening*, we can convert most of the noises, where the covariance matrix of the noise vector e is known, to Gaussian white noise (Section 5.1 in [6]). Therefore, we'll now focus only on Gaussian white noise (GWN) and in our further experiments we'll also do analytics only for the GWN added to the right-hand side of the equation (1.1).

Gaussian white noise e can be modeled as a vector $e = [e_1, e_2, ..., e_m]^T$ where each element e_i is withdrawn randomly from the same standard normal Gaussian distribution. This specific distribution has the mean $\mu = 0$ and the standard deviation η. Therefore, by computing a covariance matrix of e we get

$$Cov(e) = E[(e - \mu)(e - \mu)^T] = E(ee^T) = \eta^2 I,$$

where $E(.)$ denotes the expected value. The covariance matrix tells us how big the dispersion between elements e_i is. Moreover, covariance is a unitarily invariant operator, i.e.

$$Cov(U^T e) = Cov(e) = \eta^2 I. \tag{1.11}$$

That means that the coefficients $|u_i^T e|$ will more or less stay or slightly oscillate around one level.

First of all, let's assume that the well-known Discrete Picard Condition is satisfied for our inverse problem (1.1), else different, more complicated, methods would have to be used to approximate the solution.

The Discrete Picard Condition ([14], p.37) Let τ denote the level at which the computed singular values σ_i level off due to rounding errors. The discrete Picard condition is satisfied if, for all singular values larger than τ, the corresponding coefficients $|u_i^T b|$, on average, decay faster than σ_i.

This property can be nicely shown on a Picard plot which is already implemented, together with a `csvd` (Compact singular value decomposition) function, in the RegTools library. In Figure 1.2 we showcase the difference in the behavior of the perturbed and unperturbed problem. As we can see, for the unperturbed problem the coefficients $|u_i^T b|$ decay faster than σ_i till $i \approx 20$ where they reach the machine precision. Thus b^{exact} satisfies the Discrete Picard Condition. We also see that rounding errors play a role in adding errors to our problem. In comparison, the behavior of the coefficients on the Picard plot for perturbed problem with $\zeta = 10^{-3}$ gets bad much sooner as coefficients $|u_i^T b|$ already hit their plateau for $i \approx 10$.

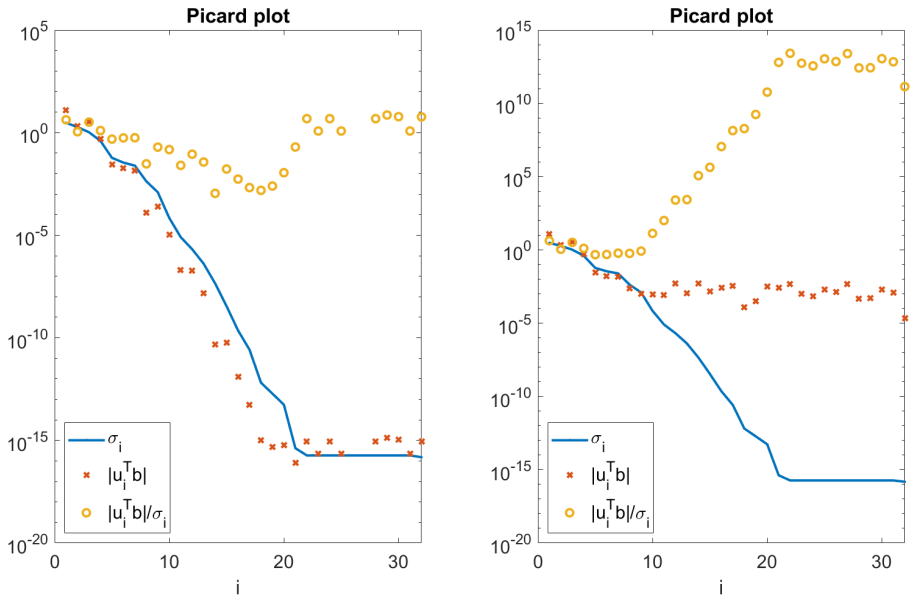


Figure 1.2: Picard plot for `shaw(32)` test problem. Left graph displays a Picard plot for unperturbed problem while right graph shows a Picard plot for a perturbed right-hand side with the noise level $\zeta = 10^{-3}$.

Second important observation is that if the noise level $\zeta < 1$, then at least the first few coefficients $|u_i^T b^{exact}|$ will be greater than $|u_i^T e|$. However, because we assume that the Discrete Picard Condition is satisfied, we know that the coefficients $|u_i^T b^{exact}|$ decay to zero with an increasing index i, even faster than the singular values σ_i. That said, what might happen is that from a certain index i the coefficients $|u_i^T b^{exact}|$ will on the contrary be smaller than $|u_i^T e|$ because of the property (1.11), see Figure 1.3.

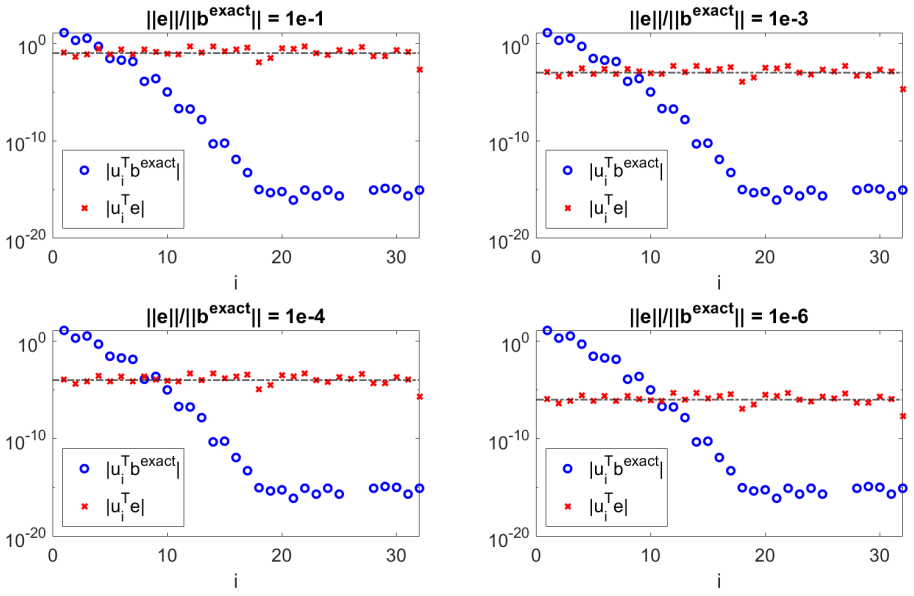


Figure 1.3: Gaussian white noise added to the right hand side of the test problem `shaw(32)`. In these four plots we show the absolute values of coefficients $|u_i^T b^{exact}|$, $|u_i^T e|$ in (1.10) with noise levels equal to $10^{-1}, 10^{-3}, 10^{-4}, 10^{-6}$ (illustrated by the dot dashed line). This manifests which component in the equation (1.10) outbalances the other and around which iteration that happens.

This scenario would lead to the components $|u_i^T b|$ from (1.10) losing the information about b^{exact} which is undesired. Unfortunately, this is a typical behavior of discrete inverse problems. The consequence it has on the naive solution computed directly from the SVD decomposition (1.9) is shown in Figure 1.4. As we can see, this solution doesn't even remotely resemble the exact solution x^{exact}. Therefore, a different approach will have to be studied.

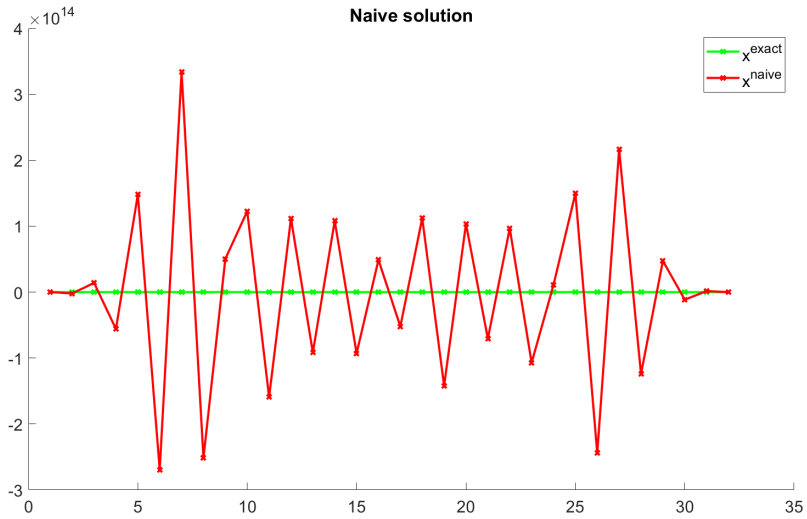


Figure 1.4: Comparison of the exact and naive solution for the perturbed test problem `shaw(32)` with $\zeta = 10^{-3}$. Clearly, x^{naive} does not represent a meaningful approximation of x^{exact}.

1.4 Regularization

To solve the problem stated above, we try to use fewer SVD components while computing the approximate solution x in (1.9) as the error caused by noise e creeps in primarily in posterior iterations. We'll call this solution a truncated SVD solution x^k, see [14], Chapter 4.

Definition 4 (Truncated SVD)**.** Truncated SVD or TSVD solution x^k is defined as a sum of the first k components of the naive solution, i.e.

$$x^k = \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i, \qquad 1 \leq k \leq r. \tag{1.12}$$

The reason behind doing this is simple. As we've explained in the previous section, the coefficients $|u_i^T b|$ linked to the larger singular values are more trustworthy, whilst the other coefficients spoil our solution. This can be observed from Figure 1.3 where coefficients $|u_i^T b|$ hold the useful information for case $\zeta = 10^{-3}$ only until the index $i = 7$. Therefore, it makes sense to choose the parameter k equal to 7, and as we can see in the Figure 1.5, this gives us the best solution so far.

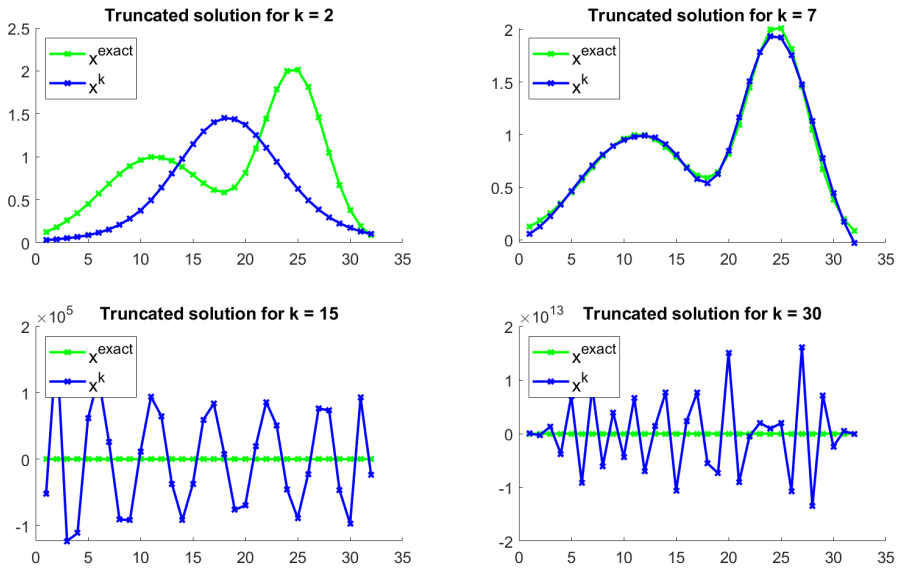


Figure 1.5: Truncated SVD solutions for a test problem `shaw(32)` with $\zeta = 10^{-3}$ obtained by formula (1.12) with various parameters k.

If we compare a condition number of the matrix $A_7 = \sum_{i=1}^{7} \sigma_i u_i v_i^T$ with the original matrix A, we see that the condition number of the new matrix is much smaller. Concretely, $cond(A) = \frac{\sigma_1}{\sigma_r} = 1.9678 * 10^{16}$ while $cond(A_7) = \frac{\sigma_1}{\sigma_7} = 122.0332$. Hence, the new regularized problem

$$x^k = A_k^{\dagger} b$$

is less ill-conditioned and stabler.

However, it's not easy to choose the regularization parameter k in practice. In our example, we were able to track the best solution belonging to $k = 7$ because we had the exact measurement data b^{exact}. Unfortunately, in reality it's not available. There's a whole big theory around various approaches for the choice of the parameter k. In this thesis, we won't go into details about these methods. The algorithms we'll study in the chapter *Considered methods* will mostly use the discrepancy principle in a combination with other criteria.

The discrepancy principle ([14], p.90) We say that a vector x_k satisfies the discrepancy principle, if

$$||b - Ax_k|| \leq \alpha\epsilon, \tag{1.13}$$

where $\alpha \geq 1$ is a prescribed safety parameter and ϵ an estimate of the norm of noise e in the right-hand side of (1.1). The discrepancy principle used as a regularization parameter selection method chooses k as the largest integer satisfying (1.13).

A curious reader can find more explanation of this and other parameter selection methods and references to original papers in [14], Chapter 4.

2. Krylov subspace methods

So far we've shown the properties of discrete inverse problems only on a small 1D test problem `shaw(32)`. Here it was feasible to explicitly compute the SVD decomposition of the matrix A and trim its troublesome components afterward. However, for a larger problem (1.1) this would be computationally and memory-wise too expensive. In these cases, it's wiser to solve the problem (1.1) by iterative regularization methods.

2.1 Projection methods

The most known and employed group of iterative methods for real problems, which lead to large system matrices, are Krylov subspace methods [17]. These methods belong to a group of projection methods that project the original problem to a small dimensional subspace where the approximate solution is computed, see Figure 2.1. In other words, if the original system matrix $A \in \mathbb{R}^{m \times n}$ then the solution x lives in a space $\mathcal{S} = \mathbb{R}^n$. By applying a project method on the original problem (1.1) we reduce the n-dimensional search space $\mathcal{S}$, i.e. a space where we search for an approximate solution x, to some low-dimensional subspace $\mathcal{S}_k \subset \mathcal{S}$ where $k < n$. We have to choose a suitable basis for this low-dimensional subspace and then define a new problem.

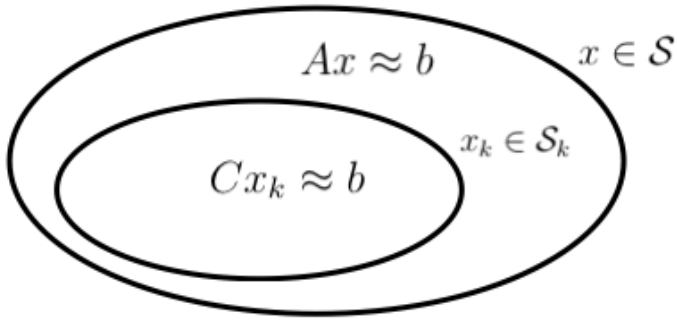


Figure 2.1: Illustration of the idea of projection methods.

The problem can be defined in various ways. We'll introduce only two of them in this chapter. First, we can look for an approximate solution that minimizes an error, i.e. *error projection methods* ([21], p. 18)

$$x_k = \underset{y \in x_0 + \mathcal{S}_k}{\operatorname{argmin}} ||y - x^{exact}||_2. \tag{2.1}$$

The second possibility, which we'll focus on, is to minimize a residuum, i.e. *residual projection methods* ([21], p. 18), where the approximate solution is chosen such that

$$x_k = \underset{y \in x_0 + \mathcal{S}_k}{\operatorname{argmin}} ||Ay - b||_2. \tag{2.2}$$

In both cases x_k lies in the space $x_0 + \mathcal{S}_k$ where x_0 denotes the initial approximation. For simplification of further derivations let's consider $x_0 = 0$ throughout the thesis.

One might find a similarity between the projection approach and the truncated SVD we've seen in the previous chapter. There the dimension of the search space of the initial problem defined by column vectors of the matrix V from the SVD decomposition was reduced to the vector space defined by only its k vectors $v_1, ..., v_k$ corresponding to the k largest singular values $\sigma_1, ..., \sigma_k$. In other words, here formally

$$\mathcal{S}_k = span\{v_1, \ldots, v_k\}$$
$$C = A_k.$$

We know that these vectors are very significant for reconstructing a good approximate solution that lives in their span. Unfortunately, as we stated above, for large problems it's computationally inefficient to obtain the right singular vectors by the SVD decomposition. The question now is if there is a way to find a set of basis vectors that have the same overall features as the first singular vectors, namely, being dominated by low-frequency components.

It's good to note that the singular values $\sigma_1, ..., \sigma_r$ of the matrix A from (1.8) are simultaneously the square roots of eigenvalues of the matrix $A^T A$ [18], Theorem 2.6.3, p. 150. This holds, since for every singular value σ and its pair of singular vectors u, v of A

$$Av = \sigma u,$$
$$A^T u = \sigma v.$$

Therefore, by using this knowledge we get

$$A^T Av = \sigma A^T u = \sigma^2 v, \tag{2.3}$$

which shows that σ^2 is an eigenvalue of the matrix $A^T A$. Moreover, the vector v is the corresponding eigenvector. This can be used to our advantage because there exist several methods on how to approximately acquire the eigenvectors of a matrix.

2.2 The partial eigenvalue problem

Probably the most famous methods for the solution of partial eigenvalue problem are the Power iteration ([11], Chapter 7, 8), the Arnoldi method [2] and the Lanczos method for symmetric matrices [20]. While the basic Power iteration is typically used to approximate the extremal eigenvalues and their associated eigenvectors, the latter methods enable us to approximate more eigenvalues simultaneously.

Definition 5 (Krylov subspace)**.** Let $B \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$. The subspace

$$\mathcal{K}_k(B, q) = \text{span}\{q, Bq, \ldots, (B)^{k-1} q\}$$

is called the k-th Krylov subspace generated by the matrix B and the vector q.

Recall that we are interested in the eigenvalues of A^TA, which is symmetric. The Lanczos method can be used to sequentially compute bases of Krylov subspaces

$$\mathcal{K}_k(A^TA, q) = \text{span}\{q, A^TAq, \ldots, (A^TA)^{k-1}q\}.$$

The principle of this method is the following:

- take a non-zero initial vector $q \in \mathbb{R}^n$ and put $w_1 = \frac{q}{||q||_2}$,
- get the next vector w_2 by orthogonalizing the vector A^TAw_1 against w_1 followed by normalisation, i.e.

$$\tilde{w}_2 = A^TAw_1 - (A^TAw_1, w_1)w_1, \quad w_2 = \frac{\tilde{w}_2}{||\tilde{w}_2||_2},$$

  where $(.,.)$ denotes the Euclidean inner product,
- obtain every following vector w_j by orthogonalizing the vector $(A^TA)w_{j-1}$ against previous two vectors w_{j-1}, w_{j-2} followed by normalisation.

The question now is how to choose the initial vector q. If we choose it randomly, the Krylov subspace $\mathcal{K}_k(A^TA, q)$ won't carry any information about our measured data b. Instead, we can look at the normal equation

$$A^TAx \approx A^Tb, \tag{2.4}$$

which is just the equation (1.1) multiplied by A^T from the left. This equation still carries all the information of our original problem and therefore motivates us to choose $q = A^Tb$. The Krylov subspace of our interest will hence be

$$\mathcal{K}_k(A^TA, A^Tb) = \text{span}\{A^Tb, (A^TA)A^Tb, \ldots, (A^TA)^{k-1}A^Tb\}. \tag{2.5}$$

Figure 2.2 displays three bases of the same Krylov subspace (2.5) obtained by three different methods.

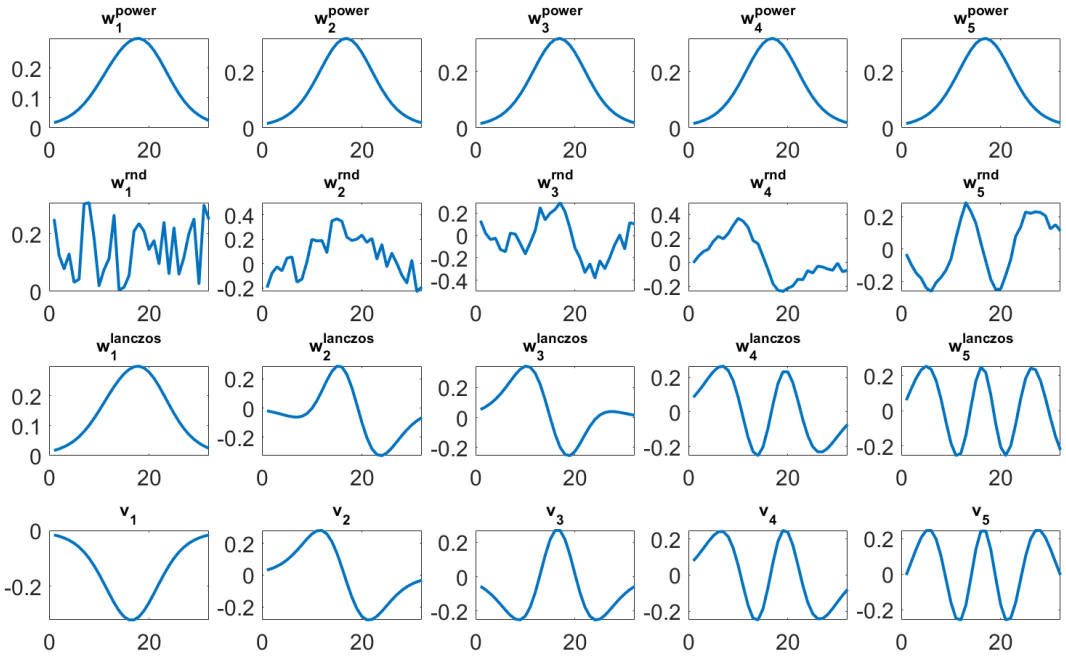


Figure 2.2: Various bases of the Krylov subspace $\mathcal{K}_5(A^TA, A^Tb)$ for the problem `shaw(32)`. The first row from the top shows a naive basis computed by the Power method as $w_i^{power} = (A^TA)^{i-1}A^Tb/||(A^TA)^{i-1}A^Tb||_2$. The second row shows a basis computed by the Lanczos method with a random initial vector q. The third row shows a basis computed by the Lanczos method with an initial vector $q = A^Tb$. The last row contains the first five right singular vectors v_i of the SVD decomposition. We can see that the basis vectors from the third row are very similar to these vectors.

The last row is acquired by the SVD decomposition. As explained before, we want to approximate this basis. In the Power basis, the linear independence of basis vectors is lost due to the dominance of the first vector. Using the Lanczos method with a random initial vector leads to basis vectors with oscillations. However, using the Lanczos method with an initial vector $q = A^Tb$ leads to a fairly good approximation of the right singular vectors v_i of the SVD decomposition. A relation between these bases and the SVD basis can be also studied from an expansion of the basis vectors of Krylov subspace $\mathcal{K}_5(A^TA, A^Tb)$ in the SVD basis. This is shown in Figure 2.3, where a dominant component of every vector can be observed.

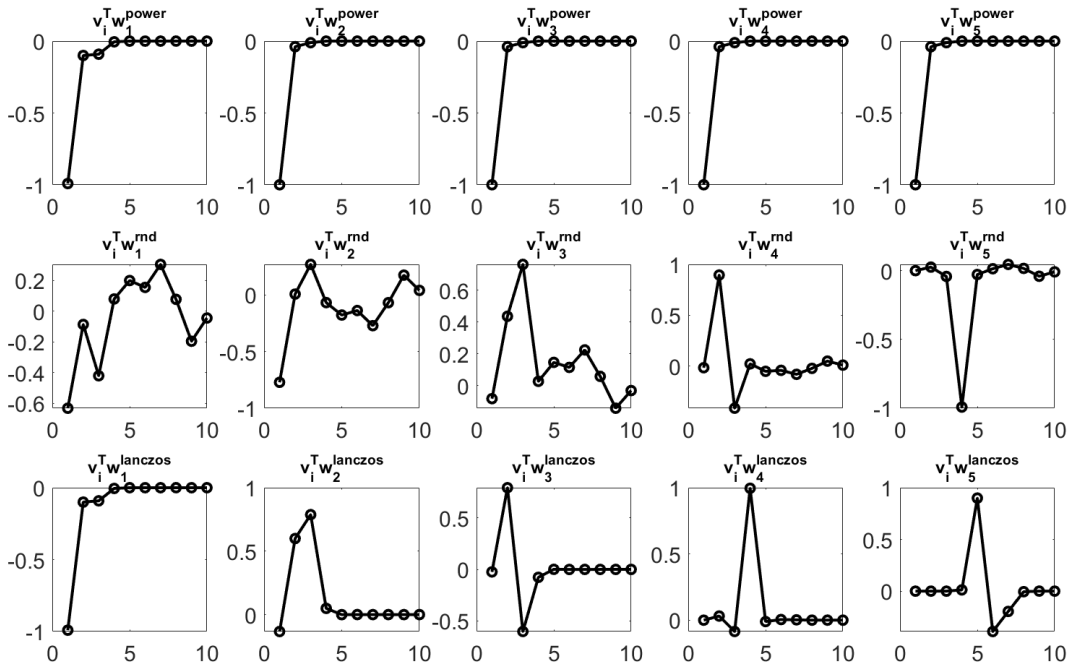


Figure 2.3: Expansion of the basis vectors of the Krylov subspace $\mathcal{K}_5(A^TA, A^Tb)$ in the SVD basis. Here we investigate the further similarity of the basis vectors from Figure 2.2 and the right singular vectors v_i of the SVD decomposition. See that the basis vectors computed naively converge to one vector. The basis vectors from the second row don't have a dominant component in most cases. The basis vectors $w_i^{lanczos}$ computed by the Lanczos method with an initial vector $q = A^Tb$ seem to do much better as they are generally dominated by a vector v_j (starting from $i = 2$) with an index $j \approx i$.

Therefore, let's set the search space $\mathcal{S}_k$ to $\mathcal{K}_k(A^TA, A^Tb)$. Since we're now working with the normal equation (2.4) instead of the original equation (1.1), and considering that we want to minimize the residuum of the normal equation, (2.2) is replaced with

$$x_k = \underset{y \in \mathcal{S}_k}{\operatorname{argmin}} ||A^TAy - A^Tb||_2. \tag{2.6}$$

This already fully defines the approximate solution x_k. Projection methods are generally defined by two conditions: the choice of a search space $\mathcal{S}_k$, where $x_k \in \mathcal{S}_k$, and the choice of a condition space $\hat{\mathcal{C}}_k$, where $z_k \perp \hat{\mathcal{C}}_k$, $z_k \equiv A^Tb - A^TAx_k$. The search space is already set and the equation (2.6) determines how the condition space would look, concretely

$$\hat{\mathcal{C}}_k = \mathcal{S}_k = \mathcal{K}_k(A^TA, A^Tb). \tag{2.7}$$

Now, note that

$$z_k = A^Tb - A^TAx_k = A^T(b - Ax_k) = A^Tr_k, \tag{2.8}$$

where r_k is a residuum to the original problem (1.1). From (2.7) and (2.5), we

obtain after some manipulation

$$r_k \perp \mathcal{C}_k, \quad \mathcal{C}_k = \mathcal{K}_k(AA^T, b). \tag{2.9}$$

Definition 6 (Projection method; [27], Chapter 5)**.** Let $B \in \mathbb{R}^{n\times n}$ and $\mathcal{S}_k$, $\mathcal{C}_k$ (for $k \in \mathbb{N}$, $k \leq n$) are two k-dimensional subspaces of $\mathbb{R}^n$. A projection technique onto the subspace $\mathcal{S}_k$ and orthogonal to $\mathcal{C}_k$ is a process which finds an approximate solution x_k to (1.1) by imposing the conditions that x_k belongs to $\mathcal{S}_k$ and that the corresponding residual vector is orthogonal to $\mathcal{C}_k$, i.e.

$$Find\ x_k \in \mathcal{S}_k, \qquad such\ that \quad b - Ax_k \perp \mathcal{C}_k. \tag{2.10}$$

2.3 CGLS

By choosing $\mathcal{C}_k = \mathcal{K}_k(AA^T, b)$ we'll get the Lanczos method for solving symmetric linear systems (LSLS) ([27], Section 6.6). This method is mathematically equivalent [1] to the Conjugate gradient method (CG) [17] applied on the equation (2.4), otherwise known as the Conjugate gradient for least squares (shortly CGLS, also known as CGNR) algorithm ([4], p. 288-293). As the name of CGLS hints, it finds the least squares approximate solution to the original system (1.1). However, note that this method minimizes the norm of error in consideration to the normal equation (2.4), which leads to the minimization of Euclidean norm of the residuum, i.e.

$$\begin{aligned} ||x_k - x^{exact}||^2_{A^TA} &= (A^TA(x^{exact} - x_k), (x^{exact} - x_k)) \\ &= ||A(x^{exact} - x_k)||_2^2 \\ &= ||r_k||_2^2. \end{aligned}$$

As seen previously, both TSVD and LSLS methods have to explicitly compute and store the basis. CGLS method, however, computes the approximate solutions x_k implicitly, see the Algorithm 1. Computationally and storage-wise it's very advantageous because CGLS uses only two matrix-vector multiplications per iteration, one by the matrix A and the other by the matrix A^T, and it doesn't have to store all the previous vectors. Therefore, notice in Algorithm 1 that the matrix A^TA doesn't need to be explicitly computed to obtain the normalized basis vectors of the Krylov subspace $\mathcal{K}_k(A^TA, A^Tb)$. Only matrix-vector multiplications must be available.

[1] By mathematical equivalency we mean that the search and the condition space are for both methods defined in the same way, i.e. $\mathcal{S}_k = \mathcal{K}_k(A^TA, A^Tb)$, $\mathcal{C}_k = \mathcal{K}_k(AA^T, b)$, and thus the approximate solution is identical to the one obtained by LSLS in the exact arithmetics.

Algorithm 1 CGLS/CGNR [27]

Compute $r_0 = b - Ax_0, z_0 = A^T r_0, p_0 = z_0$
for $k = 0, 1, \ldots$ until convergence **do**
    $w_k = Ap_k$
    $\alpha_k = ||z_k||_2^2 / ||w_k||_2^2$
    $x_{k+1} = x_k + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k w_k$
    $z_{k+1} = A^T r_{k+1}$
    $\beta_k = ||z_{k+1}||_2^2 / ||z_k||_2^2$
    $p_{k+1} = z_{k+1} + \beta_k p_k$
end

2.4 Semiconvergence

It can be shown that using certain iterative methods on discrete inverse problems often leads to a phenomenon called semiconvergence ([14], Chapter 6). That is when the relative error

$$\frac{||x_k - x^{exact}||_2}{||x^{exact}||_2}$$

first decreases with an increasing number of iterations but at some iteration index k the approximate solution x_k starts to deviate from the exact solution x^{exact}, leading to an increase in the relative error, see Figure 2.4. Note that in order to compute the relative error, we must know the true noise free solution. In reality that is unknown, however, it is available in these artificial experiments. If we can stop the iterative method soon enough, near some optimal iteration k_{opt} where $k_{opt} = \underset{k \in \mathbb{N}}{\operatorname{argmin}} ||x^{exact} - x_k||_2$, then we can often obtain a good-quality regularized approximate solution of (1.1).

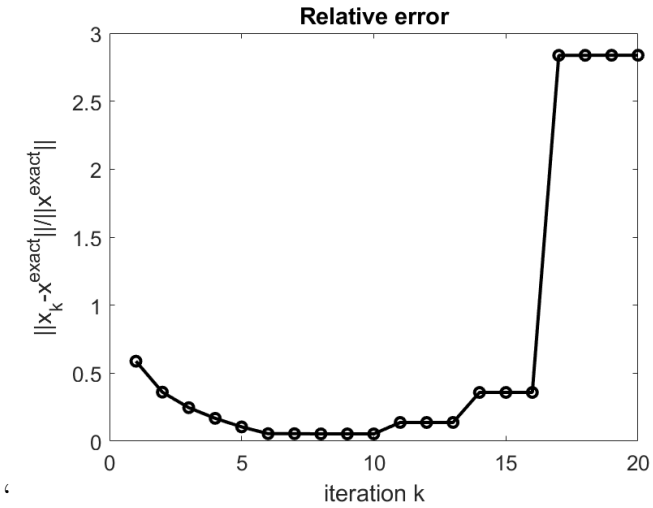


Figure 2.4: This plot shows the relative error of the approximate solutions x_k for a test problem `shaw(32)`, $\zeta = 10^{-3}$ computed by CGLS. It illustrates a phenomenon of semiconvergence commonly present in iterative solution of discrete inverse problems.

The phenomenon of semiconvergence is inspected in Figure 2.5. The plot displays the first 20 approximations obtained by the CGLS algorithm. The approximate solution is enhanced every iteration up to $k = 10$, where the high-frequency components start to dominate the basis vectors. Methods for choosing regularization parameter k such as the discrepancy principle mentioned in Chapter 1 proved to be quite good in many cases [14]. This is the reason why iterative regularization methods, especially regularized projection methods as they have generally faster convergence, are such powerful tools for solving discrete inverse problems.

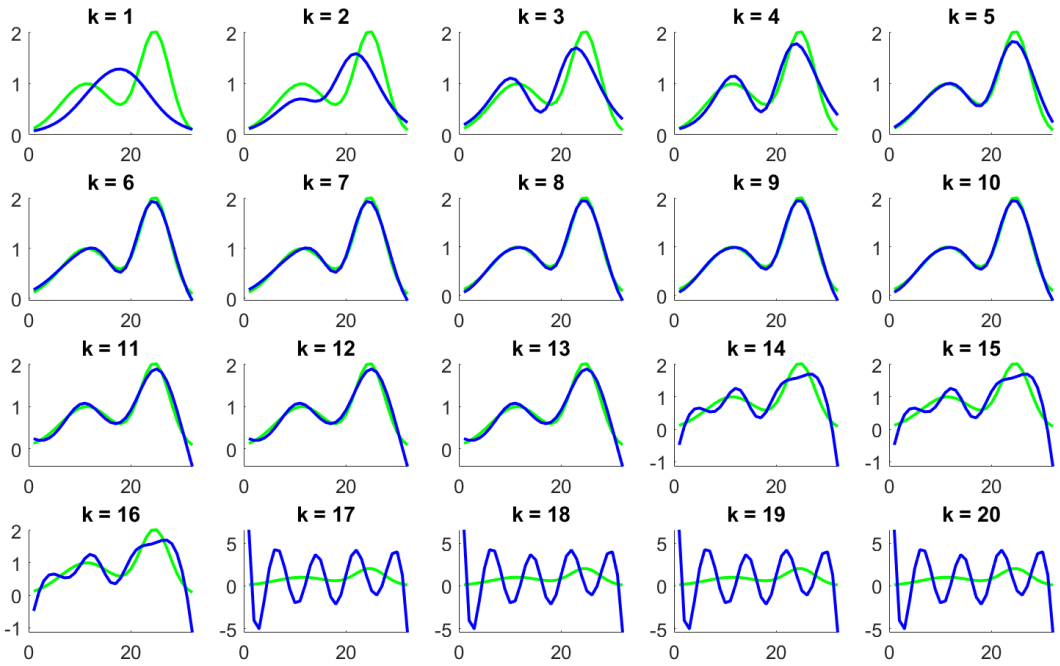


Figure 2.5: The first 20 iterations of CGLS algorithm applied on the test problem `shaw(32)` with the noise level $\zeta = 10^{-3}$. The green line is the exact solution x^{exact} and the blue line an approximate solution x_k. The best approximate solution is obtained for $k = 10$ and notice how it starts to deviate from the exact solution and only gets worse within next iterations.

2.5 Other Krylov subspace methods

Note that if we chose to solve a different problem from the beginning, e.g. (2.1), we would obtain a different condition space, and therefore a different method. The one we derived in section 2.2 corresponds to Krylov subspace methods called CGLS and LSQR, which are mathematically equivalent.

Apart from that, there exist also other Krylov subspace methods related to normal equations, e.g. LSMR [8], CGNE [27]. For example, CGNE is associated

with the system

$$\begin{aligned} AA^T y &= b, \\ x &= A^T y, \end{aligned} \tag{2.11}$$

which is obtained by substituting x by $A^T y$ in the original equation (1.1).

However, sometimes we don't even have to consider normal equations. At the beginning, we worked with a general rectangular matrix $A \in \mathbb{R}^{m \times n}$. If our model matrix A has special properties such as $A \in \mathbb{R}^{n \times n}$ is nonsingular and possibly moreover symmetric or even positive definite (SPD), i.e. $x^T Ax > 0$ for all $x > 0$, other Krylov subspace methods are applicable for regularization, see the following table.

Matrix	**Algorithm**	**Krylov subspace**	**Solution**
Symmetric	MINRES	$\mathcal{K}_k(A, b)$	$x^{(k)}$
	MR-II	$\mathcal{K}_k(A, Ab)$	$\bar{x}^{(k)}$
Nonsymmetric and square	GMRES	$\mathcal{K}_k(A, b)$	$x^{(k)}$
	RRGMRES	$\mathcal{K}_k(A, Ab)$	$\bar{x}^{(k)}$
Any	CGLS, LSQR	$\mathcal{K}_k(A^T A, A^T b)$	$\hat{x}^{(k)}$

Figure 2.6: Various residual minimizing methods and the corresponding Krylov search subspaces. Table taken from [19].

Let's start with the most specific category mentioned above. For solving systems of linear equations with SPD matrices the most used algorithm is CG. If the matrix A is only symmetric and nonsingular, we can use a projection method called MINRES [25] which is based on the Lanczos tridiagonalization and it minimizes a residual norm of the approximate solution x_k. For a nonsymmetric nonsingular square matrix, a well-known method is GMRES [26]. However, in the case of discrete inverse problems we have to be careful with using these methods directly because they are based on the Krylov subspaces generated by the matrix A and the vector b

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, \ldots, (A)^{k-1} b\}.$$

The noise present in the vector b can cause major oscillations which would spoil the smoothness of the solution. From Chapter 1 we know that for discrete inverse problems the matrix A has a smoothing effect. Therefore, in these cases, where a smooth solution is desired, modification of these methods is preferred: RRGMRES and MINRES-II (MR-II). Instead of the Krylov subspace above, they are based on a shifted subspace.

Definition 7 (Shifted Krylov subspace)**.** The k-th shifted Krylov subspace generated by the matrix A and the vector b is defined as follows

$$\mathcal{K}_k^{shift}(A, b) = \text{span}\{Ab, A^2 b, \ldots, (A)^k b\} = \mathcal{K}_k(A, Ab).$$

This span contains less noise as we omitted the vector b and shifted the span one place to the right. See [19] for detailed explanation.

3. Non-negative inverse problems

Definition 8 (Non-negative vector)**.** The vector $x \in \mathbb{R}^n$ is non-negative if and only if all of its components are non-negative, i.e.

$$x_i \geq 0 \qquad \forall i = 1, \ldots, n.$$

We denote $x \geq 0$.

So far general inverse problems were introduced. This thesis will however focus on one specific kind, non-negative inverse problems. These arise in applications where a non-negative solution x is expected. We can look at this as solving constrained least squares

$$\min_{x \in \mathbb{R}^n} ||Ax - b||_2, \quad x \geq 0. \tag{3.1}$$

These problems are very common, mainly in image deblurring, but we can find a lot of one-dimensional cases as well. Some of them can be found in the Reg-Tools package for Matlab [13]. One such example was already illustrated in the previous chapters, the `shaw` test problem, see e.g. Figure 2.5. As we can see in this figure, where CGLS is used to find an approximate solution, or in Figure 1.5, where a truncated SVD solution is computed, the constraint of non-negativity is not always fulfilled and the approximate solution in such cases is very inaccurate. Moreover, it's sometimes very complicated to interpret an approximate solution if the non-negativity is not satisfied. For example, such uncertainty can happen when considering the task of deblurring an image. Images are represented on computers as arrays with non-negative values, which correspond to the intensity of the light that hits each pixel. But how to use this analogy if we get a solution with negative elements?

In this chapter, we'll talk about the difficulties one might come across when trying to solve non-negative inverse problems by standard methods. Later we'll look at methods specially designed to produce approximations satisfying the non-negativity constraint. Most of these methods will be tested on 1D test problems from RegTools package [13] and 2D test problems from IR Tools package for Matlab [10], which contains mainly realistic large-scale problems arising in image reconstruction.

3.1 Image representation

First, it's useful to summarize the basics of how the 2D images are represented, processed, and manipulated on computers, and how point-spread functions define the blurring. An image is a 2D object consisting of pixels, which are small single (colored) points. Imagine for example stitching an embroidery, the principle is the same. On computers, a gray-scale image can be represented as an $M \times N$ matrix where each element holds information about a color of a particular pixel. For colored images, every color can be expressed as a combination of three color channels, where each channel marks an intensity of either red (R), green (G), or

blue (B). The intensity is denoted by an integer varying from 0 to 255. This is often referred to as the RGB color model, see Figure 3.1.

There exist other formats of storing images but the RGB format suffices for the explanation. For example, the red color is represented by a value (255,0,0), as red is one of our main three colors and therefore it should have the highest intensity on the first color channel and zeros on the two left channels. However, that doesn't mean that this is the only option how to obtain red. Every color comes in various shades and it can be changed with how much intensity we put on that color. By combining colors we can get a new color, e.g. yellow is represented by a value (255, 255, 0). Hence, the RGB model can cover the whole color spectrum.

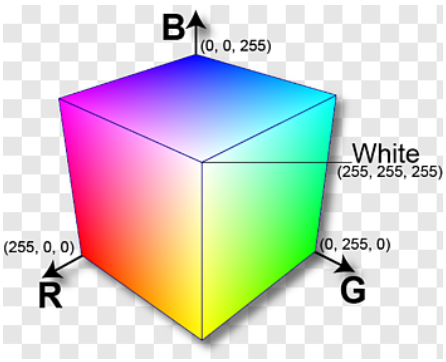


Figure 3.1: RGB colour space [1]

In this thesis we will make things easier by focusing only on gray-scale images, i.e. black-and-white pictures. There's no need of three colour channels for gray-scale images anymore. We only need to say how much intensity of the light falls on each pixel. That can be represented by one number varying from 0 to 255, where 0 is black and 255 is white. This value can be normalized to fit in an interval $[0, 1]$, where 0 is black and 1 is white. This is how `imagesc` works in Matlab. With the commands `imagesc(X), axis image, colormap(gray)` in this order, we display a grayscale image X ([16], Chapter 1). From all the knowledge we have gathered until now, we know that the variable X can be seen as a 2D array with values from 0 to 1. Consider a simple example,

$$X = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

to see what each of the commands above does in Figure 3.2.

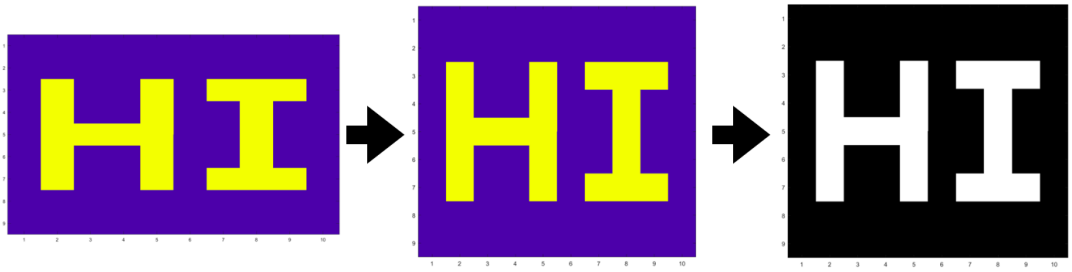


Figure 3.2: Displaying an array X with Matlab commands `imagesc(X), axis image, colormap(gray)` in this order. The first image is stretched to fit the axes position. Command `axis image` shrinks it to a square image and the third command converts the image into a gray scale.

3.2 Point spread function (PSF)

The blurring of an image can be caused by many factors. For example, the camera can be out of focus, our hands are shaking while taking a picture, the blurring is caused by atmospheric turbulence (Figure 3.3), and more (see Section 3.1 in [10]). Many of such blurring processes can be modeled using the so-called point spread function (PSF). In a continuous setting described by the Fredholm integral of the first kind

$$\int K(s,t)f(t)\,dt = g(s), \qquad s,t \in \mathbb{R}, \tag{3.2}$$

PSF is represented by a function $K(s,t)$ called blurring kernel. This function specifies how the points in the image are distorted [10]. Moreover, PSF can be either spatially invariant or spatially variant, where a spatially invariant PSF is obtained if $K(s,t) = K(s-t)$.

A task of image deblurring on computers is a discrete, finite-dimensional problem, and hence we directly work with a discrete model $Ax \approx b$, see (1.1). The matrix A is determined by the PSF, x is approximate of a true image and b is a blurred noisy image. The PSF is applied on every pixel of a sharp image, i.e. on each element of the vector x. Firstly, let's explain why a true image and a blurred image are represented in our discrete model by vectors and not matrices when we clearly showed above that grayscale images are 2D arrays.

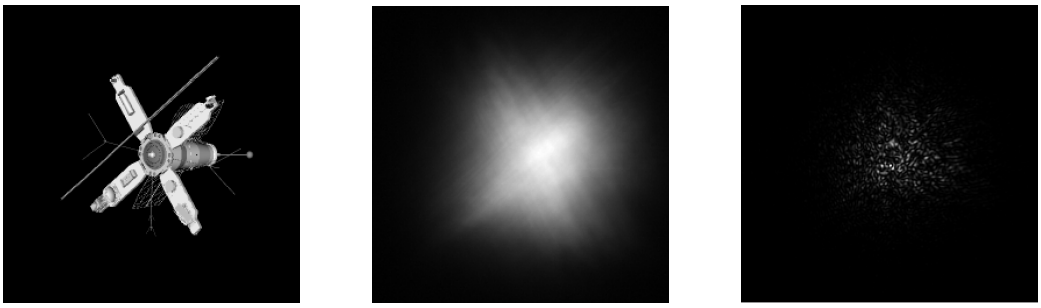


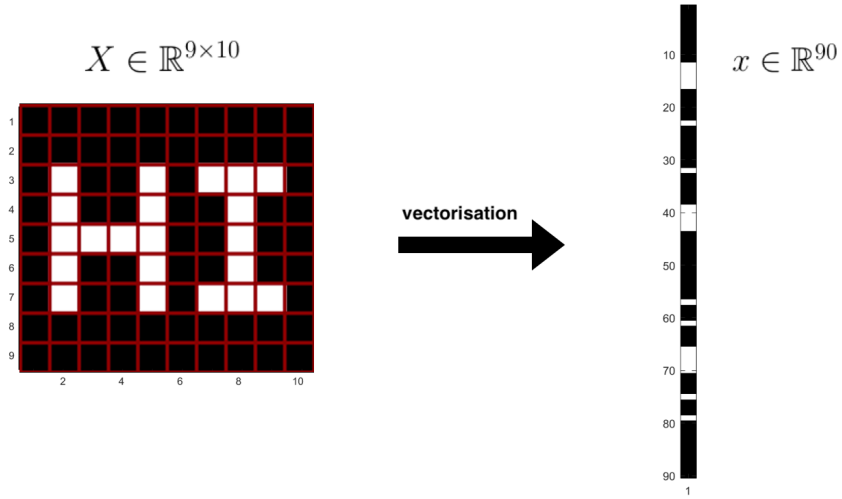
Figure 3.3: An example of a blurred `satellite` image, where the blurring is caused by atmospheric turbulence. On the left we see the exact image, in the middle a blurred noisy image and on the right the PSF obtained by `PRblurspeckle` function implemented in IR Tools.

3.3 Storing images

Let's assume that $X \in \mathbb{R}^{M\times N}$, $B \in \mathbb{R}^{M\times N}$ are a sharp and a blurred image, respectively. Notice that they must have the same sizes. If the blurrings of the rows and columns are independent, then we can express the blurring of an image by the equation

$$A_c X A_r^T = B = B_{exact} + E, \tag{3.3}$$

where $E \in \mathbb{R}^{M\times N}$ is a noise matrix, $A_c \in \mathbb{R}^{M\times M}$ blurs the columns of the image X and $A_r^T \in \mathbb{R}^{N\times N}$ blurs the rows of X [16]. Assuming that the blurring operation is linear, which in many applications holds or the blur can be approximated by a linear model [16], we can convert the equation (3.3) to the equation (1.1). The vector x is a result of the vectorization of X, i.e. the columns of X are stacked into a long vector. The vector b is created the same way from the matrix B. Therefore, the dimensions of x, b, A are the following: $x, b \in \mathbb{R}^{MN}$ and $A \in \mathbb{R}^{MN\times MN}$, see an example below. However, bear in mind that the matrix A is never explicitly constructed. Instead, we operate with a Matlab object, a small sparse matrix (e.g. blurring of images), or a function handle (e.g. inverse diffusion) corresponding to the PSF. Such examples are implemented in the IR Tools [10].



Now we have a problem of the form (1.1) and we can try to solve it by the standard Krylov subspace regularization methods recalled in Chapter 2. These methods are implemented in the IR Tools package and we will freely use them to showcase the performance of the methods on non-negative inverse problems.

3.4 Using standard methods

Let's take a test problem `dotk` from IR Tools, which is an image of stars, and a PSF generated by the function `PRblurdefocus`. This PSF simulates a spatially invariant, out-of-focus blur [10]. Moreover, a Gaussian noise e of relative noise level $\zeta = 0.01$ is added to the right-hand side, see the top right picture of Figure 3.4. We use the CGLS algorithm stopped by the discrepancy principle (DP CGLS) with $\alpha = 1.01$, as described in Chapter 2. In this case, the algorithm stops at the iteration $k = 100$ due to reaching the maximum number of iterations. In Figure 3.4 we see, that this approximate solution is still very noisy and inaccurate. The optimal stopping iteration refers to the iteration, where the relative error is minimal. The solution obtained in this iteration is called the best-regularized solution.

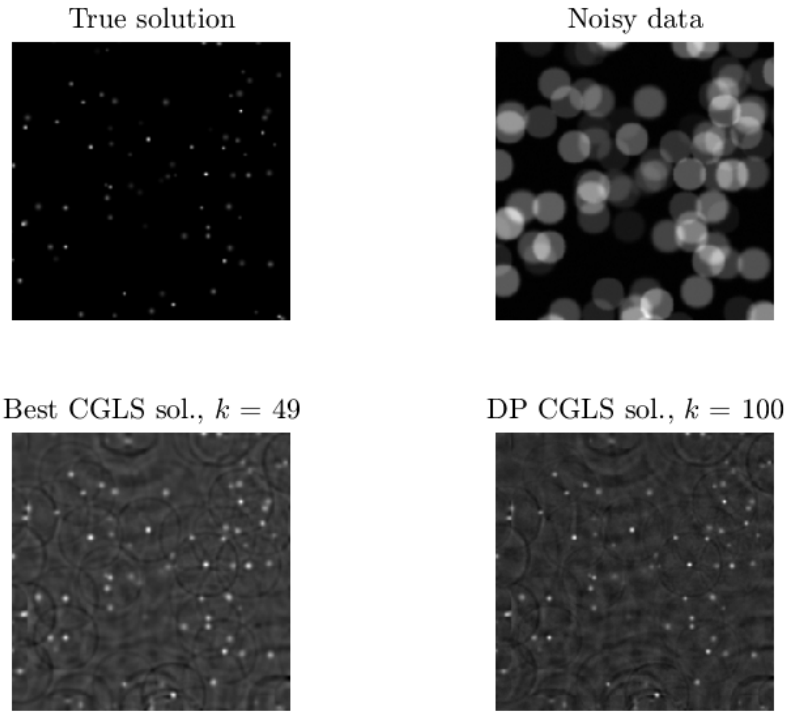


Figure 3.4: Reconstructed `dotk` noisy image blurred with `PRblurdefocus` using regularized CGLS. These pictures illustrate how inaccurate the approximate solution obtained by regularized CGLS is, even at the optimal stopping iteration (Best CGLS). The tracks of circles caused by out-of-focus blur are still visible and the black background is never fully recovered.

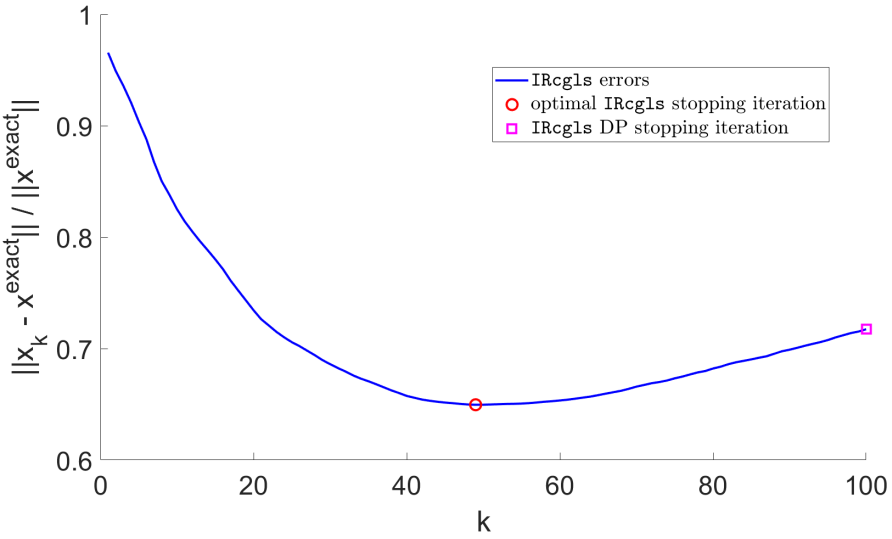


Figure 3.5: The error history for the test problem `dotk` with `PRblurdefocus` blurring. Two points are highlighted: the red circle denotes the optimal iteration for stopping the CGLS method to obtain the best-regularized solution; the pink square denotes an actual iteration obtained by regularized CGLS stopped by the discrepancy principle. The reconstructed images acquired at these two points are displayed in Figure 3.4.

Another example, where the inaccuracy of reconstructed solutions can be shown, is the standard regularized RRGMRES algorithm employed on the inverse PDE problem (2D diffusion problem of the size $N = 64$) implemented in the IR Tools package, see Figure 3.6.

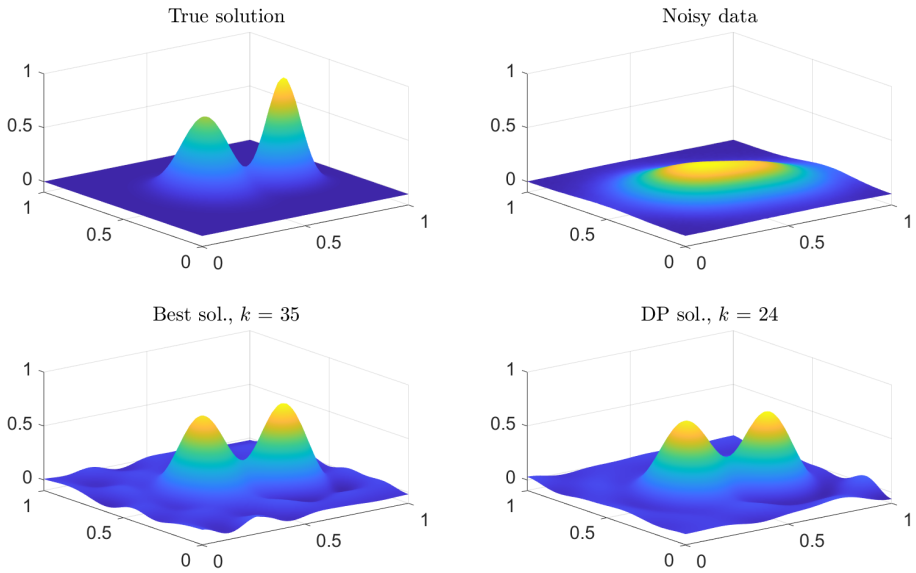


Figure 3.6: Inverse diffusion problem produced by `PRdiffusion` with Gaussian noise, $\zeta = 0.005$, added to the right-hand side. Again we see, that the approximate solution obtained by regularized RRGMRES is not able to reconstruct the second peak well and the surface at level 0 is warped.

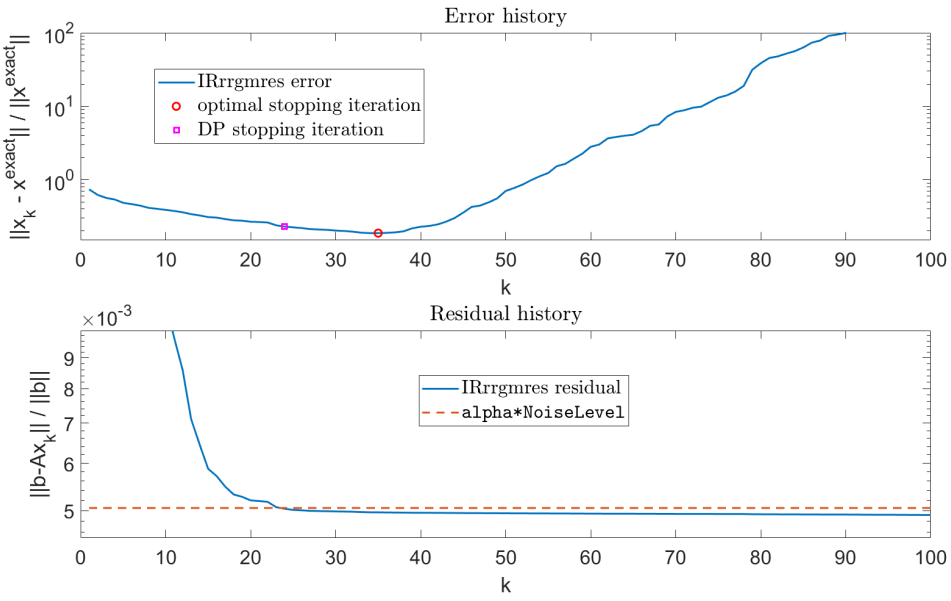


Figure 3.7: Error history and residual history of RRGMRES used on the inverse diffusion problem constructed by `PRdiffusion`. The Gaussian white noise of the noise level $\zeta = 0.005$ was added to the right-hand side of this problem. On the error plot, we see that the optimal stopping iteration is $k = 35$ but the discrepancy principle (with $\alpha = 1.01$) stopped the iterations a bit earlier, at $k = 24$. This is due to the intersection of relative residual norm curve, defined as $||b - Ax_k||_2/||b||_2$, and $\alpha\zeta$, shown in residual history.

In a similar fashion as in the first example solved by the CGLS algorithm above, the semi-convergence behavior can be spotted in an error history of the RRGMRES algorithm, see Figure 3.7. The RRGMRES method was stopped before it reached the optimal stopping iteration. The argument is provided in the residual history. The discrepancy principle tells us to stop the iteration as soon as $||Ax_k - b||_2 \leq \alpha\epsilon$ is satisfied, where α is a safety factor (usually slightly larger than 1) and ϵ is an estimate of the norm of the noise $||e||_2$. Therefore, as soon as the left-hand side function of the above equation intersects with the right-hand side function, we stop the iterations. From the residual history, we see that that happens for $k = 24$.

From this, we can conclude that standard Krylov subspace methods might sometimes not work well on discrete inverse problems with a non-negativity constraint. Obtained approximate solutions are not actually non-negative, i.e. x may contain negative elements. That can be nicely seen on the bottom two plots in Figure 3.6. Hence, these approximations are hard to interpret. One way how to deal with this issue is to find a minimal (negative) element of the solution and shift the solution by the absolute value of this number to the right (positive) side. This is how function `imagesc` deals with images containing negative values. The images of reconstructed solutions for `dotk` test problem in Figure 3.4 were converted and displayed this way. However, as we summarized in the results above, both standard CGLS and RRGMRES don't perform well. Maybe if we strive for keeping the non-negativity constraint (3.1) satisfied for an approximate

solution at each iteration of the Krylov subspace method, we might significantly improve its performance. Next chapter will introduce methods that deal with this constraint.

4. Considered methods

In this chapter we would like to present some algorithms for solving non-negative inverse problems that were proposed in papers [5], [9], and were further tested on problems in [10] and [13]. We will also propose several modifications of these methods and discuss their performance in comparison with the original versions.

All of the methods we'll talk about here belong to the projection methods that preserve the non-negativity of the approximate solution or they do so to some extend. For an explanation of these approaches, we will consider only CGLS and RRGMRES algorithms, however, an arbitrary Krylov subspace method can be used.

By methods preserving non-negativity of approximations we basically mean that $x_k \geq 0$ component-wise for some or all $k \in \mathbb{N} \cup \{0\}$. One of the first ideas on how to achieve that was introduced in [5]. Here we simply use the unconstrained Krylov subspace method and project negative elements of the solution x_k to zero, i.e. $((x_k)_+)_i = max\{0, (x_k)_i\}$. The symbol $(.)_+$ denotes a projection onto a set of non-negative vectors

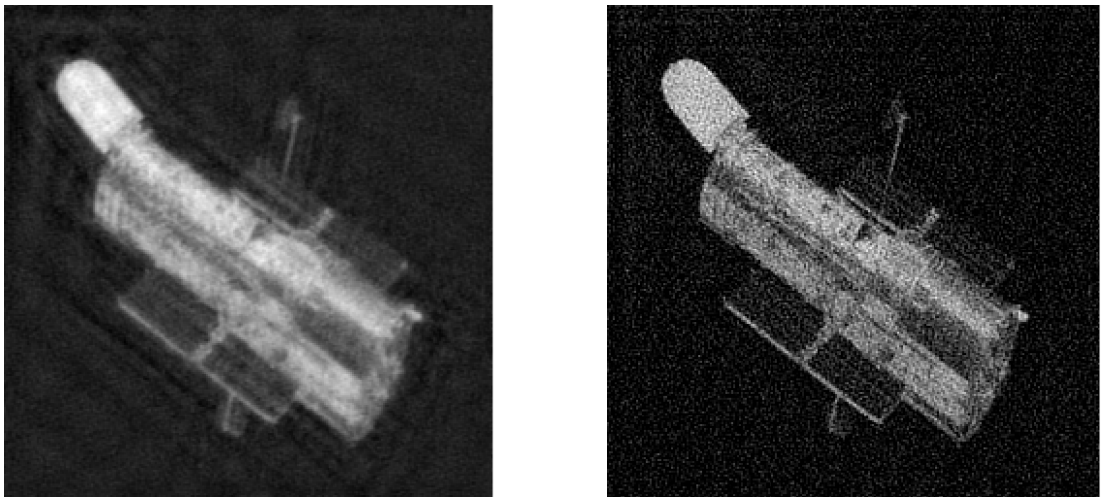
$$\mathbb{S} = \{x \in \mathbb{R}^n : x \geq 0\}. \tag{4.1}$$

4.1 Basic non-negative projections

There are several ways how to impose non-negativity. Let us start with a modification of the CGLS algorithm. As we can see in the Algorithm 1 in Chapter 2, there's no guarantee that the approximate solution x_k is non-negative. A quick and easy fix we suggest in this thesis (similar idea was studied in [9]) is to preserve the non-negativity right after computing a new approximation x_{k+1}. We use the projection to the set $\mathbb{S}$ and call this modification the Modified CGLS, i.e. MCGLS - see Algorithm 2. This method brings some better results than CGLS, see Figure 4.1.

Algorithm 2 MCGLS

Compute $r_0 = b - Ax_0, z_0 = A^T r_0, p_0 = z_0$
for $k = 0, 1, \ldots$ until convergence **do**
    $w_k = Ap_k$
    $\alpha_k = ||z_k||_2^2 / ||w_k||_2^2$
    $x_{k+1} = (x_k + \alpha_k p_k)_+$
    $r_{k+1} = r_k - \alpha_k w_k$
    $z_{k+1} = A^T r_{k+1}$
    $\beta_k = ||z_{k+1}||_2^2 / ||z_k||_2^2$
    $p_{k+1} = z_{k+1} + \beta_k p_k$
end



(a) (b)

Figure 4.1: On the left picture we see a reconstruction of `hst` deblurring problem with a speckle point spread function and noise with $\zeta = 10^{-2}$ by standard CGLS. The picture on the right is a reconstruction of the same problem by MCGLS. Notice how sharper and refined the approximate solution by MCGLS is in comparison with the unconstrained CGLS.

Similarly as in Algorithm 2, we can modify other Krylov regularization methods such as RRGMRES, etc. In Section 5, we use CGLS and RRGMRES algorithms preimplemented in the IRTools that we modified to impose the non-negativity. Unfortunately, that led to certain restrictions on what we were able to do. For example, the modification inside the CGLS algorithm was quite straightforward. However, that was not the case for the RRGMRES which uses Householder reflections to construct Hessenberg matrices which are then used to obtain the approximate solution x_k.

The issue with MCGLS is that we have no idea how the modification on line 5 in Algorithm 2 influences the convergence, therefore the success of this strategy is not guaranteed. Another proposal is to have a method with inner and outer iterations where in the inner iteration the approximation to the system $Ax \approx b$ is solved using regularized CGLS or RRGMRES (or any other Krylov subspace method). Once the inner cycle is quitted by satisfying the stopping criterion or reaching the maximum number of inner iterations $maxIn$, we project the obtained approximation denoted as x_{k+1} to the non-negative orthant $\mathbb{S}$. After that the algorithm is restarted with x_{k+1} as a new initial vector, unless the stopping criterion of outer cycle is satisfied or the maximum number of outer iterations $maxOut$ is reached. This idea is used in Algorithm 3. Let's assume throughout this chapter that we have an estimate of the norm of noise e on the right-hand side of the equation (1.1). Therefore, the discrepancy principle can be used as a stopping criterion.

Algorithm 3

1: **Input:** $A, b, x_0 \geq 0, maxIn, maxOut, \epsilon, \alpha$
2: **Output:** Approximate non-negative solution to (3.1)
3: $r_0 = b - Ax_0, k = 0$
4: **while** the stopping criterion is not satisfied and $k \neq maxOut$ **do**
5: **Inner iteration:** Compute an approximate solution x_{k+1} to a system $Ax \approx b$ by regularized CGLS or RRGMRES with the initial vector x_k. Terminate the iterations when the discrepancy principle is satisfied or $maxIn$ number of iterations is reached.
6: $x_{k+1} = (x_{k+1})_+$
7: $k = k + 1$
8: **end**

4.2 Projected restarted iteration

This approach was studied in [5], where authors named it a Projected restarted iteration (PRI). Their algorithm corresponds to the idea described in Algorithm 3, however, it looks slightly different. Instead of solving the system $Ax \approx b$ inside the while loop, they perform an iterative refinement, which should improve the accuracy of numerical solutions.

Denote the exact solution of $Ax = b$ (in sense of the least squares) as follows $x^* = \tilde{x} + w$, where $\tilde{x}$ is the computed approximation and w is a deflection. This leads to a derivation

$$\begin{aligned} Ax^* &= b \\ A\tilde{x} + Aw &= b \\ Aw &= b - A\tilde{x} \\ Aw &= \tilde{r}. \end{aligned}$$

Here $\tilde{r}$ denotes the residuum. The last equation is called the correction equation (in general case the equation is again in sense of the least squares). PRI runs inner and outer iterations. In the inner iterations it solves the correction equation $Aw = r_k$ using one of the standard Krylov subspace methods mentioned above. In this equation, the matrix A and the vector r_k are known, whilst w is unknown. The iterations are terminated once the maximum number of inner iterations $maxIn$ (arbitrarily chosen by the user) is reached or the discrepancy principle is satisfied. The final approximate solution is denoted w_k. Once the inner cycle is finished, a new iterate is defined as a former iterate corrected by the vector w_k, projected onto the set $\mathbb{S}$, i.e.

$$x_{k+1} = (x_k + w_k)_+.$$

This vector is accepted as a final approximate solution only if it satisfies the outer iteration's stopping criterion. This can be again defined as a fulfillment of the discrepancy principle or reaching a maximum number of outer iterations $maxOut$. If none of the outer stopping criteria is satisfied, define $k = k + 1$, compute a new residuum r_k, and restart the iterative method, see Algorithm 4.

Algorithm 4 Projected restarted iteration

Input: $A, b, x_0 \geq 0, maxIn, maxOut, \epsilon, \alpha$
Output: Approximate non-negative solution to (3.1)
$k = 0, \tilde{w} = 0$
while $||r_k||_2 = ||b - Ax_k||_2 > \alpha\epsilon$ and $k \neq maxOut$ **do**
    Inner iteration: Compute an approximate solution w_k to a correction equation $Aw = r_k$ by regularized CGLS or RRGMRES with the initial vector $\tilde{w}$. Terminate the iterations when the discrepancy principle is satisfied or $maxIn$number of iterations is reached.
    $x_{k+1} = (x_k + w_k)_+$
    $k = k + 1$
end
Let *out* be the stopping iteration, i.e. x_{out} is the final output.

This algorithm still leads in many cases to large errors or a long computational time, and hence we consider several modifications. One is our own (MPRI) and the second one (RSPRI) is suggested by the authors of PRI in the same paper [5].

4.3 Modified projected restarted iteration

Instead of projecting the whole vector x_k onto the set $\mathbb{S}$ like in the Algorithm 4, we select a threshold $tol > 0$ which controls whether the element $(x_k)_i$ should be projected to 0 or not. The purpose of this method is to preserve the information about the real solution as much as possible, taking into account that we have to make a choice between accuracy and the non-negativity constraint. This leads to savings of total computational time in many cases, where the algorithm stops a bit earlier than the original PRI, however with a negligible difference in the relative error. Therefore, there is no visible deterioration in the reconstructed images, see the example below.

Algorithm 5 Modified projected restarted iteration (MPRI)

Input: $A, b, x_0 \geq 0, maxIn, maxOut, \epsilon, \alpha, tol$
Output: Approximate non-negative solution to (3.1)
$k = 0, \tilde{w} = 0$
while $||r_k||_2 = ||b - Ax_k||_2 > \alpha\epsilon$ and $k \neq maxOut$ **do**
    Inner iteration: Compute an approximate solution w_k to a correction equation $Aw = r_k$, with an initial vector $\tilde{w}$, by regularized CGLS or RRGMRES. Terminate the iterations when the discrepancy principle is satisfied or we reached $maxIn$.

    for i = 1, . . . , n **do**
        if $(x_{k+1})_i < -tol$ **then**
            $(x_{k+1})_i = 0$
    end
    $k = k + 1$
end
Let *out* be a stopping iteration, i.e. x_{out} is the final output.

Inspired by the paper [22], we drew a plot displaying the largest negative values in the approximate solution at each iteration of CGLS compared with its relative errors, to get a closer look at how the CGLS behaves and how the non-negativity is enforced, see Figure 4.2.

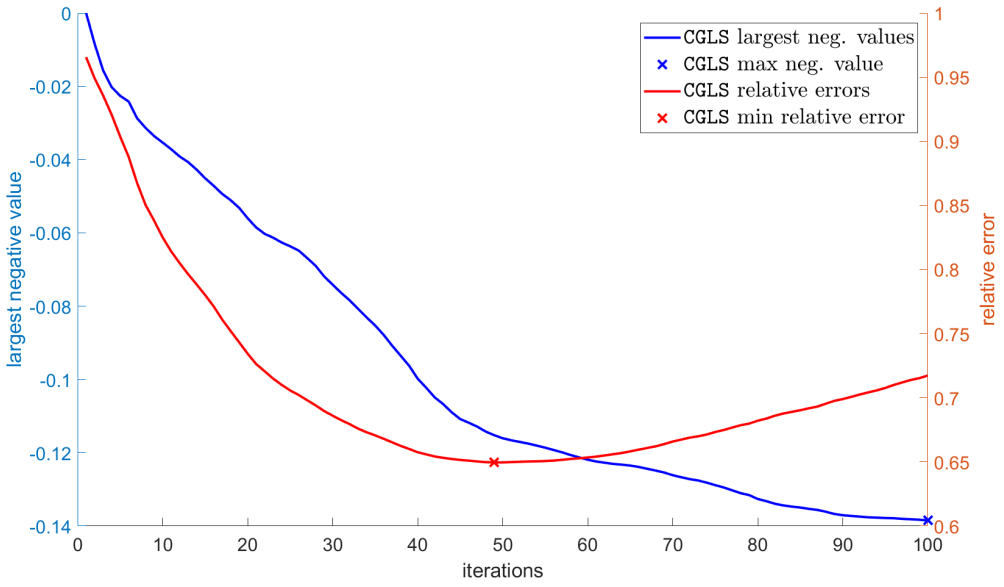


Figure 4.2: Largest negative values (in absolute value) in the approximate solution at each iteration of CGLS compared with its relative errors for the `dotk` deblurring problem with a defocus spread function, $\zeta = 0.01$, $\alpha = 1.01$. Note that as the iteration proceeds, some recovery is made, but negative values remain.

This helps us to choose a suitable threshold *tol* - we try to choose it so that most of the largest negative values are below this value, therefore, in the algorithm

they are projected to 0. As shown in Figure 4.3, MPRI with $tol = 0.001$ was stopped a bit earlier than PRI, but the reconstructed image looks the same as the one from PRI.

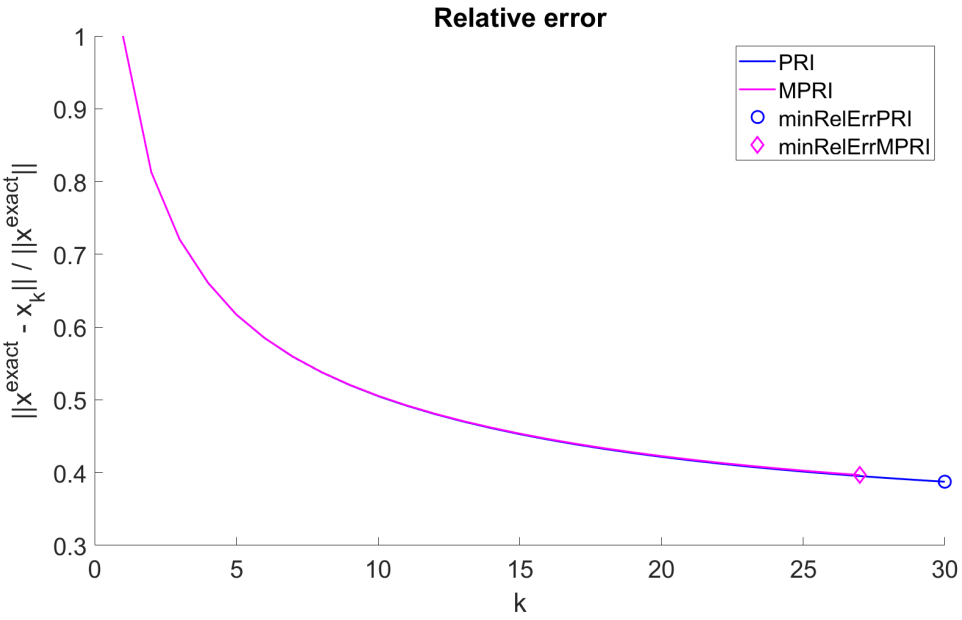
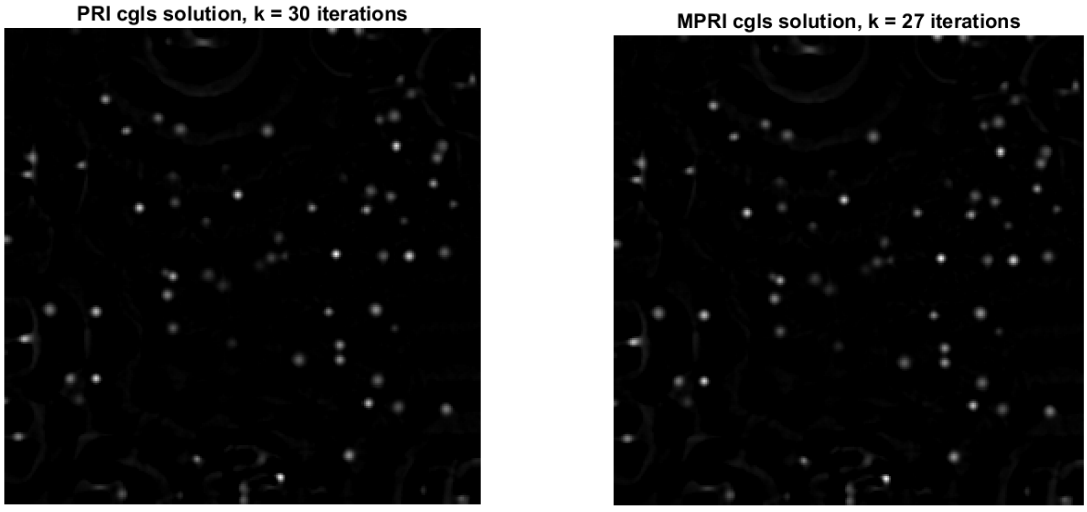


Figure 4.3: Comparison of the relative error history of PRI and MPRI with $tol = 0.001$ for the problem defined above. MPRI stops a few iterations before PRI with a minimal relative error difference.



(a) (b)

Figure 4.4: Reconstructed images using PRI and MPRI algorithm with the threshold $tol = 0.001$. The images look the same to the naked eye. However, MPRI reduced the number of iterations and the computational time significantly. PRI ended with 30 outer iterations, 261 inner iterations and the running time of 79.49 seconds. Meanwhile, MPRI with the same input parameters and $tol = 0.001$ stopped at 27 outer iterations, 243 inner iterations and time of 61.99 seconds.

The question which should be asked at this point is if the sequence $\{r_0, \ldots, r_{out}\}$ is nonincreasing, i.e.

$$||r_k||_2 \leq ||r_{k-1}||_2, \qquad k = 1, 2, \ldots, out, \tag{4.2}$$

because that's normally the case for standard CGLS and RRGMRES. The reason we want to preserve this property is because both CGLS and RRGMRES, which we use in the inner iterations, minimize the residual norm and have this behavior (4.2). Unfortunately, the projection onto a set of non-negative vectors spoils this property, and the monotonicity of the norm of residua is not guaranteed. A solution to this problem is suggested in [5], the new method is called Restricted step projected restarted iteration (RSPRI).

4.4 RSPRI

We mentioned the motivation for this method, which is to ensure (as much as possible) that the norm of the next residuum in a sequence is reduced or equal to the previous one. That can be achieved by adding a line search into the Algorithm 4 between the lines 5 and 6. Remember that we're trying to minimize the norm of our new residuum. Hence, the function we're minimizing is

$$f(\tilde{x}) = ||\tilde{r}||_2 = ||b - A\tilde{x}||_2. \tag{4.3}$$

There are two common strategies in the optimization theory to decide in which direction to shift the approximate solution to find a local minimum. One is a line search approach and the other a trust-region approach. Authors of RSPRI used a line search method with an Armijo condition (or Armijo rule), see [28]. Line search methods move a solution in a descent direction p_k with a step size γ, i.e. a new iterate is defined as $x_{k+1} = x_k + \gamma p_k$. The choice of the length of the step size is crucial here. Choosing it too small would lead to a very slow convergence and choosing it too large won't give us convergence at all and could lead to a well-known zig-zag effect. This is what the Armijo condition deals with. It checks if a function f decreases sufficiently in the direction p_k with a chosen step size. The step size γ is accepted if the Armijo condition is satisfied. Let's introduce a new function

$$\Phi(\gamma) = f(x_k + \gamma p_k). \tag{4.4}$$

Generally, the Armijo condition looks like this

$$\Phi(\gamma) < \Phi(0) + C\gamma\Phi'(\gamma), \tag{4.5}$$

where C is a small positive constant between 0 and 1, usually chosen as 10^{-4} [5]. By definition of a function Φ and formulation (4.5), we'll get

$$f(x_{k+1}) < f(x_k) + C\gamma\nabla f(x_{k+1})^T p_k, \tag{4.6}$$

where we can clearly see, that this line search moves the next approximate in the direction of gradient descent.

Looking at the PRI Algorithm 4, we can identify the vector w_k with a descent direction p_k, since a new iterate is corrected in this way: $x_{k+1} = (x_k + w_k)_+$.

Putting (4.3), (4.4), (4.5) together, defining a step size in the form of $\gamma = 2^{-m}$, where $m \in \mathbb{N} \cup \{0\}$, and substituting w_k for p_k including the following projection onto a set $\mathbb{S}$, we get the Armijo condition for our specific problem

$$||b - A(x_k + 2^{-m}w_k)_+||_2 < (1 - 2^{-m}C)||b - Ax_k||_2. \tag{4.7}$$

The Armijo condition is first checked for $m = 0$. If it's satisfied then we stop and move to the line 6 in Algorithm 4, else we increase m by one and repeat until (4.7) is satisfied. More on the Armijo condition can be found in [7], [23]. The final pseudocode of the RSPRI algorithm is shown below.

Algorithm 6 RSPRI

1: **Input:** $A, b, x_0 \geq 0, maxIn, maxOut, \epsilon, \alpha$
2: **Output:** Approximate non-negative solution to (3.1)
3: $k = 0, \tilde{w} = 0$
4: **while** $||r_k||_2 = ||b - Ax_k||_2 > \alpha\epsilon$ and $k \neq maxOut$ **do**
5: **Inner iteration:** Compute an approximate solution w_k to a correction equation $Aw = r_k$ by regularized CGLS or RRGMRES with the initial vector $\tilde{w}$. Terminate the iterations when the discrepancy principle is satisfied or $maxIn$ number of iterations is reached.
6: **Line search:** Find the smallest integer $m \geq 0$ such that it satisfies the Armijo condition (4.7).
7: $x_{k+1} = (x_k + 2^{-m}w_k)_+$
8: $k = k + 1$
9: **end**
10: Let *out* be a stopping iteration, i.e. x_{out} is the final output.

4.5 NN-FCGLS

Projection to the non-negative orthant in the earlier methods led to the approximations x_k not belonging to the Krylov subspace $\mathcal{K}_k(A^TA, A^Tb)$ (if the CGLS was used). Therefore, these methods would be often prone to premature stagnation. The new approach Non-negative Flexible CGLS, shortly NN-FCGLS, is the first systematic attempt to enforce non-negative approximations within the framework of Krylov subspace methods introduced in [9]. This method uses Karush-Kuhn-Tucker (KKT) conditions to ensure the optimality of the solution of the problem (3.1).

Definition 9 (Karush-Kuhn-Tucker (KKT) conditions [9, 22, 4])**.** KKT conditions can be compactly expressed as

$$XA^T(Ax - b) = 0, \quad where \quad X = diag(x), \quad x \geq 0, \quad A^T(Ax - b) \geq 0. \tag{4.8}$$

Consequently, in each iteration the linear system

$$X^{(k)}A^T(Ax - b) = 0 \tag{4.9}$$

should be solved, where $X^{(k)} = diag(x_{k-1})$. Note that the matrix $X^{(k)}$ acts like a left preconditioner (more about preconditioned iterations in [27], Chapter 9)

$$X^{(k)}A^TAx = X^{(k)}A^Tb,$$

however, its main purpose is to regularize the original problem by including new information about the solution as soon as a new iteration is computed [9]. Denote the preconditioner $L^{(k)} = X^{(k)}$ and we see that it updates from step to step, hence the name *flexible* CGLS.

Krylov subspace methods generally compute the approximate solution at the k-th iteration by the following formula ([3], Chapter 12)

$$x_k = x_{k-1} + \sum_{j=0}^{k-1} \alpha_j^{(k-1)} p_j, \tag{4.10}$$

where $\alpha_j^{(k-1)}$ are the step sizes and p_k the search directions defined as

$$p_k = z_k + \sum_{j=0}^{k-1} \beta_j^{(k-1)} p_j, \tag{4.11}$$

where z_k is as defined in (2.8) and $\beta_j^{(k-1)}$ are coefficients. The standard CGLS algorithm simplifies the equations (4.10), (4.11) to

$$x_k = x_{k-1} + \alpha_{k-1} p_{k-1}, \tag{4.12}$$

$$p_k = z_k + \beta_{k-1} p_{k-1} \qquad \forall k \in \mathbb{N}, \tag{4.13}$$

due to the optimality condition (3.1) and the orthogonality of Ap_j [9]. Coefficients α_{k-1}, β_{k-1} are defined as in the Algorithm 1. However, NN-CGLS works with $\bar{z}_k = L^{(k)}z_k = L^{(k)}A^Tr_k$ instead of z_k in (4.13), considering the fulfillment of (4.9). That leads back to the full recurrence (4.11) and unfortunately increases the storage cost with an increasing number of iterations.

To address this issue, the recursion (4.11) is truncated at a chosen parameter $\hat{m} > 0$. According to [9], the truncation causes loosing the orthogonality of Ap_j, and therefore the optimality property (3.1) is not guaranteed anymore. However, that shouldn't have a big impact on the convergence unless the extremal eigenvalues are well separated [24].

Now, what remains is to ensure the non-negativity of the approximate solution at every iteration, i.e. $x_k \geq 0$, $\forall k \in \mathbb{N}$. Taking into account the equation (4.12), we have to bound the step size α_{k-1} along the search direction p_{k-1}. This new bounded step size $\bar{\alpha}_{k-1}$ is computed as

$$\hat{\alpha}_{k-1} = min(\alpha_{k-1}, min(-x_{k-1}(p_{k-1} < 0)/p_{k-1}(p_{k-1} < 0))). \qquad [9] \tag{4.14}$$

Remark: The above notation is considered in the MATLAB convention.

Furthermore, it can be proven that the scalar $\bar{\alpha}_{k-1}$ is non-negative (see [9], Proposition 3.1.). Unfortunately, with the new step size a new issue arises - the method is now prone to stagnation. In order to overcome this, a restart will be needed as soon as the maximum number of inner iterations $maxIn$ is performed, or $\bar{\alpha}_m = 0$ [9]. Therefore, we will use double indices: an upper index denoting outer iterations and a lower index counting inner iterations. Altogether, we obtain the Algorithm 7.

Algorithm 7 NonNegative FCGLS method [9]

Input: $A, b, x_0^0 \geq 0, \hat{m}, maxIn, maxOut$
for $k = 1, \ldots,$ till a stopping criterion is satisfied OR $k = maxOut$ **do**
    $L^{(0)} = X^{(0)} = diag(x_0^{k-1})$
    $r_0^{k-1} = b - Ax_0^{k-1}, \bar{z}_0^{k-1} = L^{(0)}A^T r_0^{k-1}, p_0^{k-1} = \bar{z}_0^{k-1}$, and $w_0^{k-1} = A\bar{z}_0^{k-1}$
    for $m = 1, \ldots$ till $\bar{\alpha}_{m-1} = 0$, OR $m = maxIn$, OR a stopping criterion is satisfied **do**
        Set $\alpha_{m-1} = (r_{m-1}^{k-1}, w_{m-1}^{k-1})/(w_{m-1}^{k-1}, w_{m-1}^{k-1})$.
        Set $\bar{\alpha}_{m-1} = min(\alpha_{m-1}, min(-x_{m-1}^{k-1}(p_{m-1}^{k-1} < 0)/p_{m-1}^{k-1}(p_{m-1}^{k-1} < 0)))$.
        Update $x_m^{k-1} = x_{m-1}^{k-1} + \bar{\alpha}_{m-1}p_{m-1}^{k-1}$.
        Update $L^{(m)} = X^{(m)} = diag(x_m^{k-1})$.
        Update $r_m^{k-1} = r_{m-1}^{k-1} - \alpha_{m-1}w_{m-1}^{k-1}$.
        Compute $z_m^{k-1} = A^T r_m^{k-1}$ and $\bar{z}_m^{k-1} = L^{(m)}z_m^{k-1}$.
        Compute $A\bar{z}_m^{k-1}$.
        For $j = max\{0, m - \hat{m}\}, \ldots, m - 1$,
        set $\beta_j^{(m-1)} = -(A\bar{z}_m^{k-1}, w_j^{k-1})/(w_j^{k-1}, w_j^{k-1})$.
        Compute $p_m^{k-1} = \bar{z}_m^{k-1} + \sum_{j=max\{0,m-\hat{m}\}}^{m-1} \beta_j^{(m-1)} p_j^{k-1}$.
        Update $w_m^{k-1} = Ap_m^{k-1} = A\bar{z}_m^{k-1} + \sum_{j=max\{0,m-\hat{m}\}}^{m-1} \beta_j^{(m-1)} p_j^{k-1}$.
    end
    Let n_k be the stopping iteration. Take $x_0^k = x_{n_k}^{k-1}$.
end

5. Experiments

In order to study the behavior of the methods we saw in the previous chapter, it's time to present some experiments on real discrete inverse problems. This section contains experiments on 1D and 2D problems from the RegTools and the IRTools package, respectively. All the tests were performed in MATLAB R2019b on a single processor 1.6 GHz Intel Core i5.

It's important to say that we used the standard CGLS, RRGMRES, and NN-FCGLS algorithms from the IRTools toolbox, and our modifications of CGLS, PRI, and RSPRI algorithms. Although the PRI and RSPRI algorithms were available in the IRTools, we decided to implement them by ourselves. The reason was to have a better understanding of how things work inside the algorithms. As we later compare the results of our PRI and the one incorporated in the toolbox, the outcome was mostly the same. However, it's important to note that if the prebuilt algorithms were used in the later experiments, we might have had slightly different results. Therefore, any conclusion presented in this section is entirely based on the performance of our implementations.

5.1 1D problems

First, we started with testing PRI and RSPRI algorithms on 1D discrete inverse problems derived by discretization of a Fredholm integral equation of the first kind [13], i.e. the equation (1). The RegTools package contains 14 such build-in test problems, from which we've chosen only few (`phillips`, `baart`, `blur`, `foxgood`) to showcase the performance of the considered methods. Inside the PRI, RSPRI algorithms, we use the standard CGLS, LSQR, and RRGMRES to solve the correction equation derived in Section 4.2. All of these Krylov subspace methods are included in the RegTools.

Our experiments consist of $N \times N$ problems with $N = 32, 100, 300, 1000$. The Algorithm 4 and Algorithm 6 take as inputs following parameters: the system matrix A, the right-hand side vector b, $maxIn$, $maxOut$ number of iterations, the safety factor α and the estimate of the norm of noise ϵ. We tried different combinations of these parameters in order to find the optimal one that can be later used on larger problems, or to come to a general conclusion.

Parameter values

- problem size N: 32, 100, 300, 1000,
- Krylov subspace method: CGLS, RRGMRES, LSQR,
- the estimate of the norm of noise ϵ: 0.1, 0.01, 0.0001,
- the safety factor α: 1.01, 2,

- maximum number of inner iterations maxIn: 5, 15, 30,
- maximum number of outer iterations maxOut: 100.

Test problem: `foxgood` + `baart`
Both `foxgood` and `baart` test problems had a trivial solution (`foxgood`: $f(t) = t$; `baart`: $f(t) = sin(t)$). Hence, it is not surprising that both PRI and RSPRI did really well and their results were identical. They found an accurate solution after 2 outer iterations and in total less than 5 overall iterations.

Test problem: `blur`
This was already a bit more difficult problem in comparison with the previous ones. The sequence of relative errors of PRI was always nonincreasing, while the norms of residuals weren't for a small noise level 10^{-4}. Because this example was more complex than the previous ones, we only tested PRI and RSPRI for $N = 32$ and $N = 100$.

Test problem: `phillips`
Surprisingly, PRI performed better than or equally well as RSPRI in almost every aspect: computational time, number of iterations, the norm of error, and residual. When we looked more deeply into the choices the RSPRI algorithm made, we concluded that the Armijo's rule was merely ever fulfilled (even when the parameter m from (4.7) was increased up to 30, more would be computationally ineffective). That leads to our next observation: the sequence $||r_k||_k$ (for $k = 1, \ldots, 100$) was merely ever decreasing. The same holds for relative errors, but there it makes sense as it's caused by the semiconvergence (see our explanation in Chapter 2). There were cases where we reached the maximum number of outer iterations, which was set to 100. These problems arose when choosing $\epsilon = 10^{-4}$.

In general, algorithms CGLS and LSQR gave very similar results, which were mostly better than the ones obtained using RRGMRES. The difference was most visible in their computational time, where CGLS was the fastest out of these three algorithms. As a stopping criterion, we used the discrepancy principle since it's assumed that we received the estimate of the norm of noise e from the user. In the Figure 5.1 we see a table of input and output values from experiments executed on the `phillips` test problem. Besides, they manifest a general behavior of PRI and RSPRI algorithms using these specific combinations. It is obvious that the best choice for the parameters α and ϵ was 1.01 and 10^{-4}, respectively.

		Input data					Output data			
N	method	alpha	epsilon	maxIn		relative residuum	relative error	innit	outit	time (s)
					PRI					
300	rrgmres	1.01	0.1	30		1.516232161	0.05979385681	4	3	0.0282662
300	cgls	1.01	0.1	30		1.524624084	0.06521281345	4	3	0.014385
300	lsqr	1.01	0.1	30		1.524624084	0.06521281345	4	3	0.0150971
300	rrgmres	1.01	0.01	30		0.1536882761	0.01102139722	8	5	0.0555837
300	cgls	1.01	0.01	30		0.1540383636	0.01188233356	9	5	0.0306961
300	lsqr	1.01	0.01	30		0.1540383636	0.01188233356	9	5	0.0357532
300	rrgmres	1.01	0.0001	30		0.00154411068	0.002972838666	74	35	0.5320444
300	cgls	1.01	0.0001	30		0.001544347323	0.002567775532	70	30	0.2888979
300	lsqr	1.01	0.0001	30		0.001542376584	0.00278947216	59	24	0.2316278
300	rrgmres	2	0.0001	30		0.002975624958	0.004975493339	16	9	0.0788545
300	cgls	2	0.0001	30		0.002900965469	0.004995234875	20	10	0.0483258
300	lsqr	2	0.0001	30		0.002900965464	0.004995234875	20	10	0.0406885
					RSPRI					
300	rrgmres	1.01	0.0001	30		0.00154411068	0.002972838666	74	35	0.3555676
300	cgls	1.01	0.0001	30		0.001544347323	0.002567775532	70	30	0.1186349
300	lsqr	1.01	0.0001	30		0.001542376584	0.00278947216	59	24	0.1046724

Figure 5.1: Experimenting with different combinations of input parameters for PRI and RSPRI algorithms applied on the `phillips` test problem of the size 300×300. In the table we see the input data on the left side and the output data on the right side. The highlighted rows show the best results obtained by specific parameter choices.

5.2 2D problems

While the 1D problems from the RegTools were quite easy and straightforward to solve, they were not good examples of the inverse problems from real-world applications. The IRTools package, however, is more complex and provides a set of large-scale test problems in the form of discretizations of 2D linear inverse problems [10]. Such examples are inverse diffusion, inverse interpolation problem, or reconstruction of blurred and noisy 2D images. Similarly, as in RegTools, this new package contains a few preimplemented methods, concretely some of the considered methods for solving non-negative inverse problems discussed in Chapter 4 (PRI, RSPRI, NN-FCGLS). All of these methods are designed so that they can work with objects, function handles (when the matrix corresponding to the point-spread function is not explicitly given, there's a function `A_times_vec` that deals with it) and non-vectorized pictures, i.e. raw pictures. Moreover, we can find there different types of point-spread functions, for more details see the section devoted to the reconstruction of images.

Tested methods:

- unconstrained methods: CGLS, RRGMRES
- partially constrained method: MCGLS
- constrained methods: PRI, MPRI, and RSPRI with CGLS
- constrained and preconditioned method: NN-FCGLS

The decision to choose the CGLS algorithm for solving the correction equation inside the inner cycle of the methods PRI, MPRI, and RSPRI was made consid-

ering the results obtained on the RegTools test problems. CGLS outperformed LSQR and RRGMRES in most aspects. Besides, we can compare the results with the corresponding standard CGLS method.

Stopping criteria

We also experimented with the stopping criteria. We considered:

1. the discrepancy principle explained in Chapter 1 (implemented in the toolbox as well),
2. stopping when the norm of the residuum starts to increase, i.e. if for any $k \in \mathbb{N}$

$$||r_{k+1}||_2 > ||r_k||_2, \tag{5.1}$$

3. stopping when the difference between two consecutive residuals is smaller than some tolerance τ, i.e.

$$\frac{|||r_{k-1}||_2 - ||r_k||_2|}{||r_{k-1}||_2} < \tau, \tag{5.2}$$

4. stopping when we reach the maximum number of iterations.

In most cases, criterion 2 stopped the algorithm prematurely, therefore we didn't use it in further experiments.

Our study focused particularly on the following points:

- How large is the relative residual norm of the approximate solution?
- How large is the relative error norm of the approximate solution?
- How many inner and outer iterations did we run in total?
- How long was the computation?

5.2.1 EXdiffusion

This example showcases the failure of PRI and RSPRI algorithms on a 2D diffusion problem with $\zeta = 0.01$, $\alpha = 1.01$. Both of the methods were used with the CGLS method (solving the correction equation inside the inner cycle) and were compared with the regularized CGLS. While the PRI and RSPRI had a smaller norm of the relative error, they used 62 times more iterations than CGLS, see Figure 5.2, which led to the computational time equal to 3.2 hours (3 times longer than the CGLS). However, even then the approximate has a quite large error and the shapes of the peaks shown in the graph of the true solution are not reconstructed well. The reason the computational time was so large in both cases is because the stopping criterion (the discrepancy principle) was never satisfied for both inner and outer cycle. For comparison, we solved the problem also by

versions of PRI and RSPRI algorithms available in IRTools and we obtained similarly poor results.

Messages received by running the IRTools script `IRconstr_ls`:
" The stopping criterion of the inner iterations is never satisfied."
"The outer stopping criterion is never satisfied."

On the other hand, the NN-FCGLS algorithm was able to produce better results after only 93 iterations.

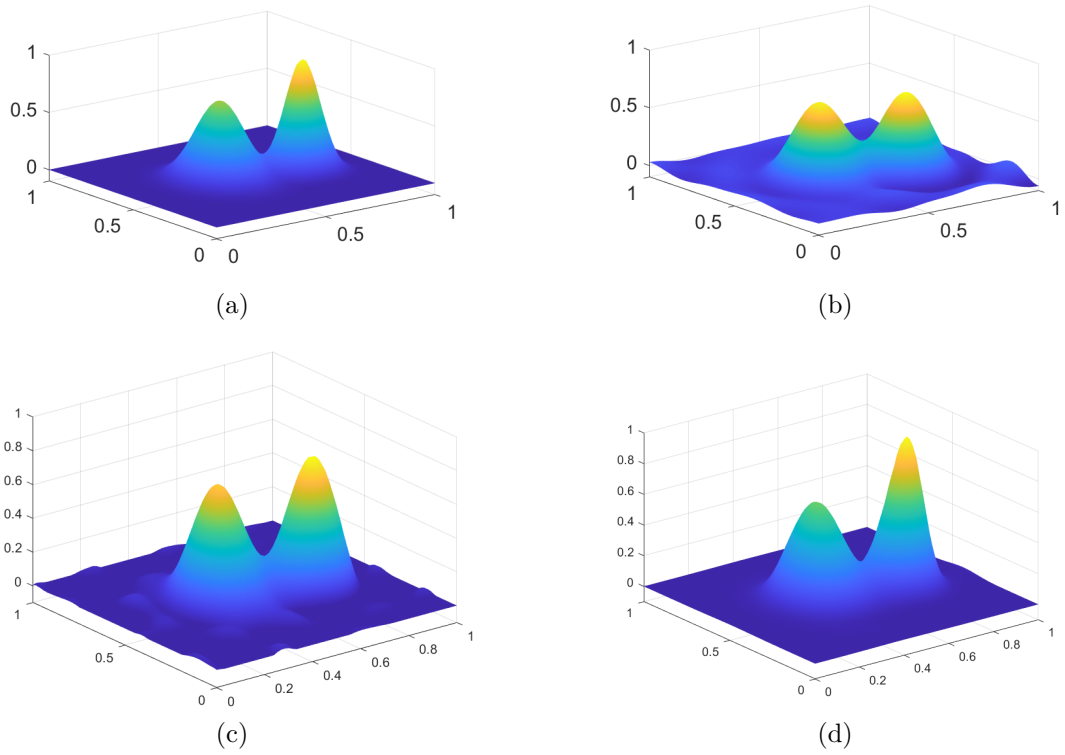


Figure 5.2: In this figure we see (a) the true solution of `diffusion test problem` with $\zeta = 0.01$ together with the approximate solutions obtained by (b) unconstrained CGLS (24 it.), (c) PRI with $\alpha = 1.01$ (1485 it.), and (d) NN-FCGLS (93 it.).

5.2.2 EXblur

This section contains the description of the largest group of our experiments. It's dedicated to the restoration of the blurred and noisy images collected by the Space Telescope Science Institute, see below.

Employed images:

- `dotk`: n/2 small Gaussian shaped dots, e.g., stars (placement is random, reset using rng(0))
- `satellite`: satellite test image
- `hst`: image of the Hubble space telescope

All of these images are included in the IRTools together with various PSFs, which we used freely. We did analyses of CGLS, MCGLS, NN-FCGLS, PRI, MPRI and RSPRI on five different image deblurring problems:

1. `PRblurdefocus` - image deblurring problem with a defocus spread function,
2. `PRblurspeckle` - image deblurring problem with a speckle point spread function caused by atmospheric turbulence,
3. `PRblurrotation` - image deblurring problem with spatially variant rotational motion blur around the center of the image,
4. `PRblurshake` - image deblurring problem with random camera motion,
5. `PRblurmotion` - image deblurring problem with linear motion blur.

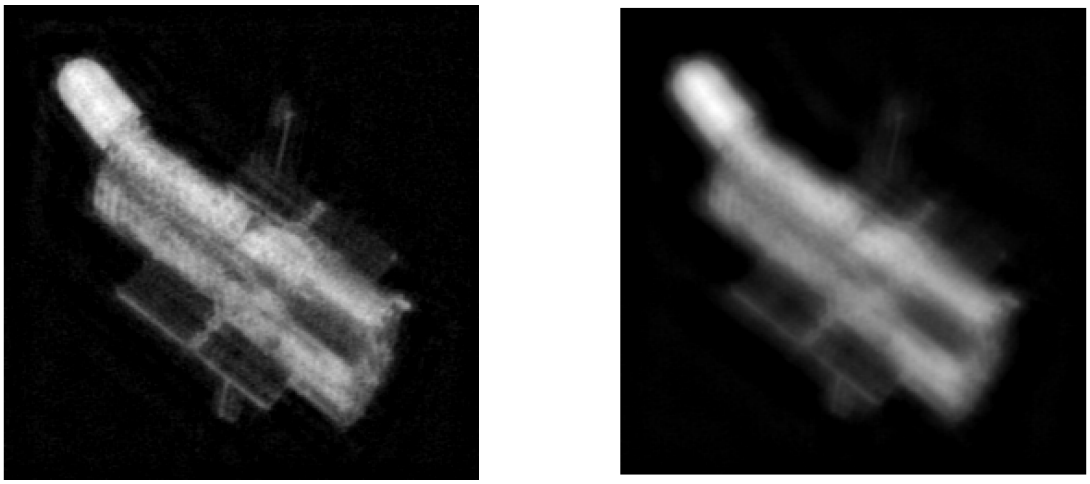
5.2.3 Choice of parameters

All experiments were done on images of the size 256×256 pixels. As for other optional variables we experimented with:

- true image: `satellite`, `hst` (image of the Hubble space telescope), `dotk` ($n/2$ small Gaussian shaped dots, e.g. stars),
- noise level: 10^{-2}, 10^{-6},
- safety factor α: 1.01, 1.1, 2,
- parameter C from the Armijo condition (4.7): 10^{-4} (as recommended in [5]),
- maximum number of inner iterations: 10, 30,
- maximum number of outer iterations: 30,
- Krylov subspace method: CGLS, RRGMRES,
- parameter τ from the equation (5.2): 10^{-2}, 10^{-3}, 10^{-6},
- threshold *tol* for MPRI: 10^{-2}, 10^{-3}, 10^{-6}.

The maximum numbers of inner and outer iterations were chosen based on the results on 1D inverse problems from the Regtools toolbox, and to prevent the computational time from getting too large like it happened for the inverse diffusion test problem.

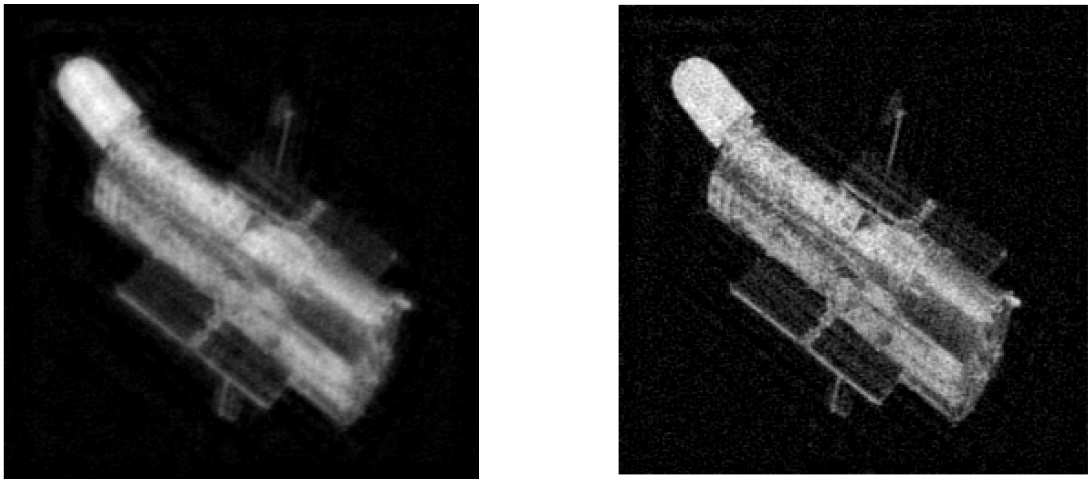
Safety factor α
In literature [5] it's recommended to set $\alpha \geq 1$. We did not observe a significant difference between the approximations obtained for values $1.01, 1.1, 2$. However, when enlargening the safety factor, the deterioration in the image could be seen, see Figure 5.3 for a basic illustration.



(a) (b)

Figure 5.3: Comparison of restored images for `PRblurspeckle` on `hst` test problem ($\zeta = 0.01$) by PRI (with CGLS) with (a) $\alpha = 1.01$, (b) $\alpha = 100$.

Maximum number of inner iterations $maxIn$
We tested different values of $maxIn$ varying from 5 to 30. It can be seen that with a larger $maxIn$ we get a more accurate approximate solution, however, in cost of the computational time. In Figure 5.4 we see on the left the solution obtained by PRI with $maxIn = 10$, and on the right PRI with $maxIn = 30$. The first one stopped after 10 outer iterations, 51 inner iterations with the total computational time of 17.5 seconds, while the second one stopped after 3 outer iterations, 435 inner iterations with the time = 58.84 seconds.

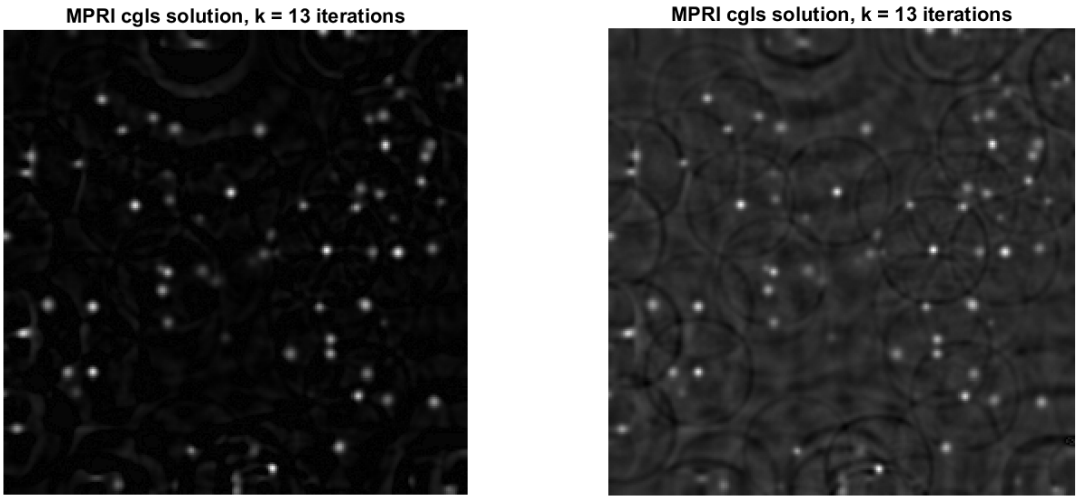


(a) (b)

Figure 5.4: Comparison of restored images for `PRblurspeckle` on `hst` test problem ($\zeta = 0.01$, $\alpha = 1.01$) by PRI (with CGLS) with (a) $maxIn = 10$, (b) $maxIn = 30$.

Tolerance for MPRI
In Chapter 4, section Modified PRI we saw the plot of the largest negative values (in absolute value) in the approximate solution at each iteration of CGLS,

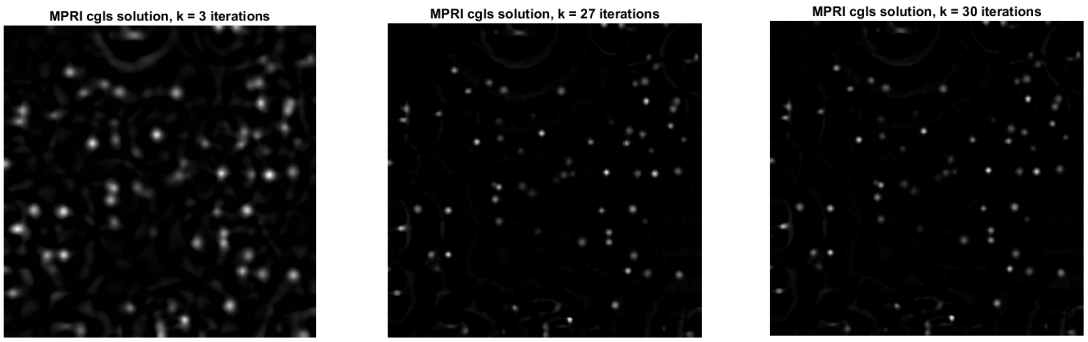
which can help us with choosing a good threshold *tol*. On Figure 4.4 we see a reconstruction of the image `dotk`, PSF `PRblurdefocus` by PRI and MPRI with $tol = 0.001$. This choice leads to a reduction of the number of iterations and consequently to saving of the computational time. With the choice $tol = 0.01$ the savings were even more significant (stopped after 13 outer iterations, 117 inner iterations, and time 22.17 seconds which is almost three times less than with $= 0.001$) without any drastic degradation in the solution, see example (a) in Figure 5.5. However, enlargening the threshold, e.g. $tol = 0.08$, can already worsen the result, see example b) in Figure 5.5.



(a) (b)

Figure 5.5: Degradation of the quality of the restored image `dotk` with `PRblurdefocus` ($\zeta = 0.01$, $\alpha = 1.01$) by MPRI (with CGLS) when enlargening the threshold from (a) $tol = 0.01$ to (b) $tol = 0.08$.

Parameter τ for the stopping criterion defined by equation (5.2)
Equation (5.2) ensures that the difference between two consecutive residuals is smaller than some parameter $\tau > 0$. The larger the difference we allow, the earlier the algorithm stops. That can lead to premature stopping, and therefore blurrier images, as shown in Figure 5.6. On the other hand, if we choose τ too small, it might be hard to fulfill the condition in a reasonable computational time. The optimal τ was found between 10^{-3} and 10^{-6}.



(a) (b) (c)

Figure 5.6: Gradually improved results for `dotk` image with `PRblurdefocus` ($\zeta = 0.01, \alpha = 1.01$, $tol = 10^{-3}$) obtained by MPRI when decreasing the value of τ: (a) $\tau = 0.1$, (b) $\tau = 10^{-3}$, (c) $\tau = 10^{-6}$.

5.2.4 Example 1

For this example, we consider the test image `satellite` of size 256×256 pixels blurred with the rotational motion blur `PRblurrotation` from [10] and noise $\zeta = 10^{-2}$. We solve this problem with six approaches using Krylov subspace methods - unconstrained CGLS, MCGLS, NN-FCGLS, PRI, MPRI, and RSPRI. The experiments were run with various parameters. However, the results presented here are the ones with the best outcomes, chosen according to the relative error norm of the approximate solution and the computational time, see the table in Figure 5.7.

method	noise level	alpha	tau	tol	Rel. error	Rel. residuum	# outit	# innit	time (s)
CGLS	0.01				0.2467235418	0.01001597566	33		4.9697142
MCGLS	0.01				0.2383583087	0.01001597566	33		2.1976198
NN-FCGLS	0.01				0.1441561599	0.01713335887	34	300	16.4509835
PRI	0.01	1.01	1.00E-06		0.1138047468	0.02749403961	30	261	13.4204405
MPRI	0.01	1.01	1.00E-06	1.00E-06	0.113804747	0.02749403964	30	261	12.7398456
RSPRI	0.01	1.01	1.00E-06		0.1541054164	0.02978805756	30	261	22.0209345

Figure 5.7: The best parameter combination for each method used to solve the image `satellite` deblurring problem with rotational motion blur and noise with $\zeta = 10^{-2}$, corresponding to the images displayed in Figure 5.8.

In the table above, we see ten columns corresponding to the input data (first five columns) explained in Section 5.2.3., and the output data (the next five columns) referring to the observations of the relative error norm of the approximate solution, the relative residual norm of the approximate solution, total number of outer and inner iterations, and the computational time, respectively. The PRI, MPRI, and RSPRI algorithms allowed a maximum of 10 iterations in each inner cycle and a maximum of 30 outer iterations. The same $maxIn$ and $maxOut$ values were set for the NN-FCGLS algorithm from [10], however, we don't have an explanation why the number of outer iterations went above 30. Meanwhile, CGLS and MCGLS allowed up to 100 iterations in total, as they don't contain two cycles.

Notice that the PRI and MPRI algorithms outperformed the others in terms of minimizing the relative error norm of the approximate solution. Also, notice that the MPRI method was one second faster than PRI. Comparing all the methods by the relative residual norms of the approximate solution, we see that PRI, MPRI, and RSPRI, on the contrary, obtained the worst results. That might be caused by the fact that CGLS is used to compute the correction equation, while the other approaches are fundamentally based on the CGLS algorithm, which is known to minimize the residual norm of the approximation of the equation $Ax \approx b$. Computationally-wise, standard CGLS, and MCGLS are the fastest, however, as seen in the next Figure 5.8, at the price of losing the accuracy of reconstructions.

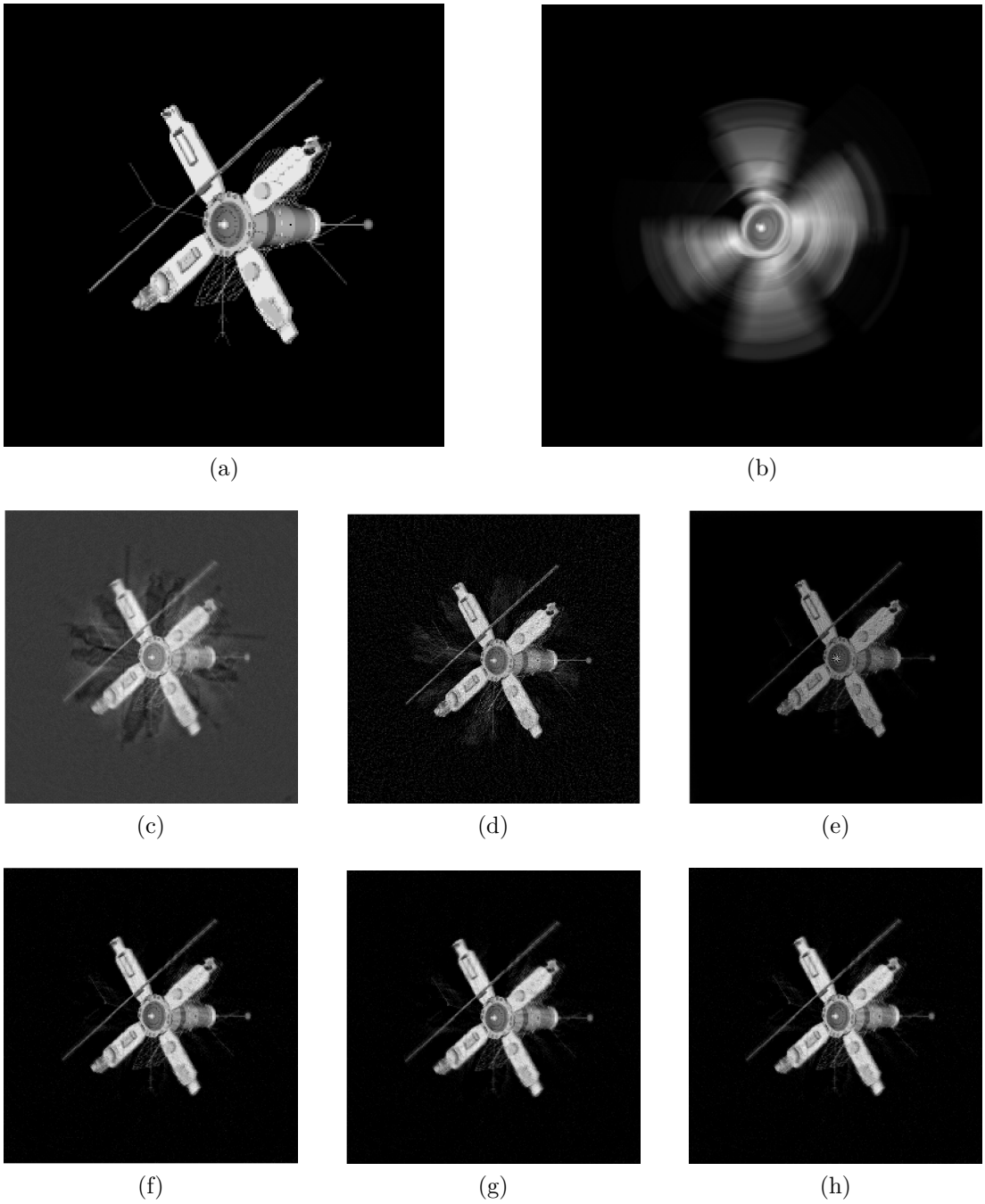


Figure 5.8: Image `satellite` deblurring problem with the rotational motion blur and noise with $\zeta = 10^{-2}$. **(a)** Exact image. **(b)** Blurred and noisy image. The next images are restorations obtained by: **(c)** CGLS method, **(d)** MCGLS method, **(e)** NN-FCGLS method, **(f)** PRI, **(g)** MPRI, **(h)** RSPRI with paramater values as set in Figure 5.7.

We can see that standard CGLS (c) gives the worst results. Unlike other reconstructions, the restored image by CGLS has a gray background and it shows defined lines from the rotation motion, which are not desired. These lines are also visible in MCGLS (d). However, the background here is already black, even though it still contains some noise visualized by tiny gray dots, and the computational time is halved. In picture (e) we finally see a clear black background

as in the exact image (a) and the lines from the rotation motion are fully suppressed. Even though this restoration by NN-FCGLS is already decent, it can be improved. Images (f), (g), (h) show restorations by PRI, MPRI, and RSPRI, respectively. These images don't differ too much from each other visually but in comparison with other methods we mentioned before, these restorations are more accurate.

5.2.5 Example 2

The second image reconstruction experiment was done on the test image `hst` of size 256×256 pixels with a speckle point spread function `PRblurspeckle` from [10] and noise $\zeta = 10^{-2}$. Like in the previous example, we compared the best reconstruction images obtained by the unconstrained CGLS, MCGLS, NN-FCGLS, PRI, MPRI, and RSPRI algorithms. The table below displays the best results achieved by a concrete choice of parameters α, τ, and *tol* (explained in Section 5.2.3.). The limits for the maximum number of inner and outer iterations for each method were set to the same values as in Example 1.

method	noise level	alpha	tau	tol	Rel. error	Rel. residuum	# outit	# innit	time (s)
CGLS	0.01				0.1908937301	0.01004958101	33		6.8871101
MCGLS	0.01				0.1972353578	0.01004958101	33		5.4831623
NN-FCGLS	0.01				0.1476602938	0.01091762001	53	300	43.6465093
PRI	0.01	1.01	0.01		0.1692470262	0.02190201441	7	40	9.9054572
MPRI	0.01	1.1	0.01	1.00E-06	0.1692470262	0.02190201443	7	40	8.8538465
RSPRI	0.01	1.01	1.00E-06		0.1874532832	0.02165478732	30	261	62.3289345

Figure 5.9: The best parameter combination for each method used to solve the image `hst` deblurring problem with a speckle point spread function and noise with $\zeta = 10^{-2}$, corresponding to the images displayed in Figure 5.10.

See that the NN-FCGLS algorithm gives the smallest relative error norm of the approximate solution. However, the gap between the computational time of NN-FCGLS and RSPRI compared to the other methods is quite large. Again notice that both PRI and MPRI return the same value of the relative error norm of the approximate solution, but MPRI is a bit faster. That seemed to be a common behavior in our experiments. Besides, the observation that PRI, MPRI, and RSPRI give the worst results for the relative residual norm and that the unconstrained CGLS and MCGLS are the fastest methods, was also recurring. Next, we will look at the reconstruction images corresponding to the data presented in Figure 5.9.

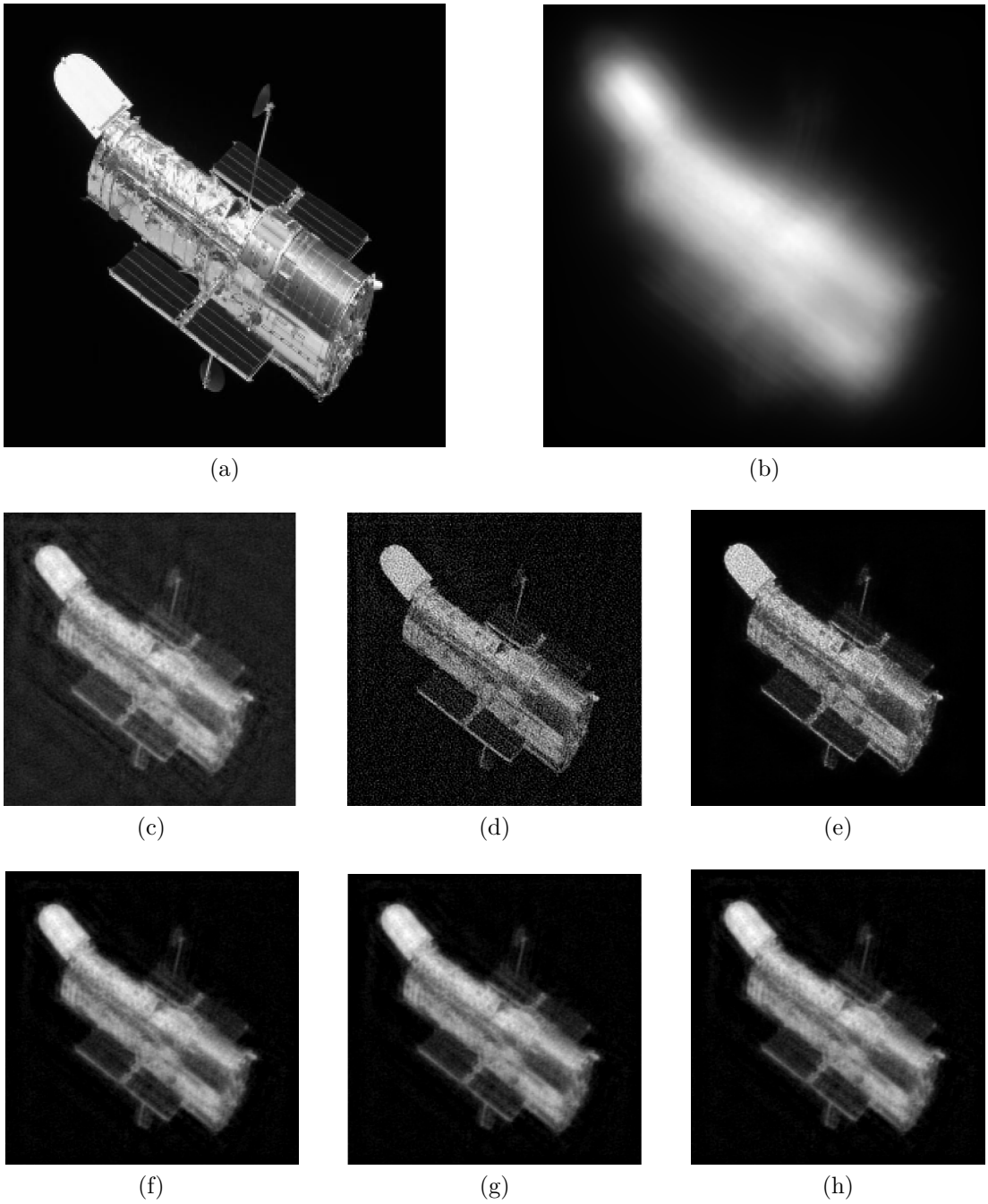


Figure 5.10: Image `hst` deblurring problem with a speckle point spread function and noise with $\zeta = 10^{-2}$. **(a)** Exact image. **(b)** Blurred and noisy image. The next images are restorations obtained by: **(c)** CGLS method, **(d)** MCGLS method, **(e)** NN-FCGLS method, **(f)** PRI, **(g)** MPRI, **(h)** RSPRI with paramater values as set in Figure 5.9.

In this case, we see that the best restoration is obtained by the NN-FCGLS method. Again, images (f), (g), (h) computed by PRI, MPRI, and RSPRI give visually the same results. In comparison with MCGLS or NN-FCGLS, it's very blurry and noisy. By projecting the negative elements to the positive sector we obtain a more detailed image, however still quite noisy.

Conclusion

In this paper, we described several iterative regularization methods based on Krylov subspace projections to solve non-negative inverse problems. Apart from studying the well-known approaches, we proposed some modifications of one of them, yielding a method we called MPRI. Since the performance of the algorithms depends strongly on careful selection of various parameters, we presented a numerical study on a variety of test problems. Based on the results, we proposed how to select parameters optimally. We concentrated particularly on four factors: the relative error norm of the approximate solution, the relative residual norm of the approximate solution, the total number of outer and inner iterations, and the computational time.

In comparison with other considered methods, MCGLS and CGLS were a lot faster, however, not very accurate. Even though the MCGLS was able to significantly improve the approximate solution from the one obtained by the unconstrained CGLS, its performance was unpredictable. Example 1 in Chapter 5 showcased successful approximations by PRI, MPRI, and RSPRI, while Example 2 illustrated the better reconstruction by the NN-FCGLS algorithm. These methods are undoubtedly competitive with each other. In terms of minimizing the relative residual norm of the approximate solution, PRI, MPRI, and RSPRI lose in this aspect. That is caused by the fact that CGLS is used to solve the correction equation inside the inner cycle instead of the original system $Ax \approx b$. Generally, choosing the right set of parameters for each method is an important aspect in enhancing the performance. We did several tests with different values of input data and we came to a conclusion that it's safe to choose the safety parameter α in the range $[1, 2]$ and the parameter τ for stopping criterion in the range $[10^{-3}, 10^{-6}]$. As for the tolerance for MPRI, we should first plot the largest negative values (in absolute value) in the approximate solution at each iteration of CGLS and then decide on the optimal *tol* according to its curve. It was shown that the choice of the parameter *tol* in MPRI can influence the results of the computation drastically. Concretely, we observed a reduction in the number of iterations and the running time.

There are several directions in which our results can be extended. For example, we focused only on the reconstruction of images of size 256×256 pixels. To come to a more complex conclusion, we should scale up the test problems and observe the behavior of the considered methods in these cases. Moreover, there exist various stopping criteria for inverse problems suggested for example in [14], [9], which weren't applied in this thesis. For future work, we advise looking into these techniques. So far, we've only considered modifications of the CGLS, RRGMRES, and LSQR algorithms on 1D test problems, and CGLS on 2D problems. However, other Krylov subspace methods can be investigated. And lastly, we can also study the performance of the methods on other test problems such as the ones from AIR Tools II software package [15] and compare them with other state-of-the-art methods known for solving non-negative inverse problems, see for example experiments in [9].

Bibliography

[1] Rgb colour space. `https://www.pngwing.com/en/free-png-sndev`.

[2] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, 9(1): 17–29, 1951.

[3] Owe Axelsson. *Iterative solution methods*. Cambridge university press, 1996.

[4] Åke Björck. *Numerical methods for least squares problems*. SIAM, 1996.

[5] D Calvetti, G Landi, L Reichel, and F Sgallari. Non-negativity and iterative methods for ill-posed problems. *Inverse Problems*, 20(6):1747, 2004.

[6] Daniela Calvetti and Erkki Somersalo. *An introduction to Bayesian scientific computing: ten lectures on subjective computing*, volume 2. Springer Science & Business Media, 2007.

[7] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[8] David Chin-Lung Fong and Michael Saunders. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971, 2011.

[9] Silvia Gazzola and Yves Wiaux. Fast nonnegative least squares through flexible krylov subspaces. *SIAM Journal on Scientific Computing*, 39(2): A655–A679, 2017.

[10] Silvia Gazzola, Per Christian Hansen, and James G Nagy. Ir tools: a matlab package of iterative regularization methods and large-scale test problems. *Numerical Algorithms*, 81(3):773–811, 2019.

[11] Gene H Golub and Charles F Van Loan. Matrix computations, 4th. *Johns Hopkins*, 2013.

[12] Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[13] Per Christian Hansen. Regularization tools: a matlab package for analysis and solution of discrete ill-posed problems. *Numerical algorithms*, 6(1):1–35, 1994.

[14] Per Christian Hansen. *Discrete inverse problems: insight and algorithms*. SIAM, 2010.

[15] Per Christian Hansen and Maria Saxild-Hansen. Air tools—a matlab package of algebraic iterative reconstruction methods. *Journal of Computational and Applied Mathematics*, 236(8):2167–2178, 2012.

[16] Per Christian Hansen, James G Nagy, and Dianne P O'leary. *Deblurring images: matrices, spectra, and filtering*. SIAM, 2006.

[17] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[18] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2013.

[19] Toke Koldborg Jensen and Per Christian Hansen. Iterative regularization with minimum-residual methods. *BIT Numerical Mathematics*, 47(1):103–120, 2007.

[20] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[21] Jörg Liesen and Zdenek Strakos. *Krylov subspace methods: principles and analysis*. Oxford University Press, 2013.

[22] James G Nagy and Zdenek Strakos. Enforcing nonnegativity in image reconstruction algorithms. In *Mathematical Modeling, Estimation, and Imaging*, volume 4121, pages 182–190. International Society for Optics and Photonics, 2000.

[23] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[24] Yvan Notay. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22(4):1444–1460, 2000.

[25] Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4): 617–629, 1975.

[26] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[27] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2000.

[28] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35:33–36, 1999.