Regularizing Iterations for Image Restoration

Toke Koldborg Jensen

LYNGBY 2003
EKSAMENSPROJEKT
NR. 18

IMM

Trykt af IMM, DTU

Preface

This present report has been written at the department of Informatics and Mathematical Modelling (IMM) at the Technical University of Denmark (DTU). It constitutes my exam project of 35 ECTS credits for obtaining the Master of Science degree. The work has been carried out with Professor Per Christian Hansen as supervisor, whom I would like to thank for his many suggestions and always enthusiastic support.

Also, I would like to thank Ph.d. student M.Sc. Michael Jacobsen for his kind support and help.

Finally, I also want to thank my fellow students from the project room, Mark Wrobel and Harald Arnbak, for an always nice and productive atmosphere.

The Technical University of Denmark
Kgs. Lyngby, March 17th, 2003

Toke Koldborg Jensen

Abstract

Image restoration problems are set up as large-scale inverse problems and studied. Interesting properties and artifacts are observed and analyzed intensively – both by means of direct calculations, and by means of the iterative algorithms LSQR and GMRES. Properties of the Krylov subspaces of the iterative algorithms are studied, and different approaches for enhancing the regularized solutions are proposed and evaluated. Stopping criteria for the iterative algorithms based on the obtained insight are also investigated.

Keywords: Image restoration, iterative algorithms, Krylov subspaces, regularization, filter factors, stopping criteria.

Forord

Billedrestaureringsproblemer er opstillet som storskala inverse problemer og undersøgt. Interessante egenskaber og artifakter er observeret og analyseret – både gennem direkte beregninger og vha. de iterative algoritmer LSQR og GMRES. Egenskaber ved de Krylov underrum de iterative algoritmer arbejder i er studeret og forskellige måder at forbedre de regulariserede løsninger på er foreslået og evalueret. Til sidst er stopkriterier for de iterative algoritmer diskuteret på baggrund af den opnåede indsigt.

Nøgleord: Billedrestaurering, iterative algoritmer, Krylov underrum, regularisering, filterfaktorer, stopkriterier.

Notation and Used Symbols

The notation used in the present thesis is described in the following, and a list of important symbols are found below.

Throughout the report, matrices are given as bold, capital letters, e.g., the matrix $\mathbf{A}$, and vectors are denoted with bold lower-case letters, e.g., $\mathbf{b}$ and $\mathbf{x}$. Denoting a specific element of a vector is done with non-bold letters and a subscript, e.g., x_i. A column of a matrix is denoted as the i'th vector, e.g., $\mathbf{a}_i$ which is the i'th column of the matrix $\mathbf{A}$. A specific element of a matrix is denoted as a subscript in square brackets as $\mathbf{A}_{[i,j]}$ which is the element of $\mathbf{A}$ at row i, column j. This notation carries over to vectors, where the vector in itself is denoted with a subscript, e.g., $\boldsymbol{\phi}_{u[j]}$ which is the element j of the vector $\boldsymbol{\phi}_u$.

Vectors coming from iterative algorithms as the regularized solutions are denoted with a superscript (k), where k is the number of iterations performed, e.g., $\mathbf{x}_{\text{reg}}^{(k)}$.

Symbol	Description	Reference
$K(s,t)$, $K(u,v,s,t)$	One and two-dimensional kernels from Fredholm integral equations	(2.1), (3.1)
$f(t)$, $f(s,t)$	One and two-dimensional true functions	
$g(s)$, $g(u,v)$	One and two-dimensional blurred functions	
$\mathbf{x}$, $\mathbf{X}$	True unblurred data as vector and image matrix, respectively	
$\bar{\mathbf{b}}$, $\overline{\mathbf{B}}$	Blurred data as vector and image matrix, respectively	
$\mathbf{b}$, $\mathbf{B}$	Blurred and noisy data as vector and image matrix, respectively	
$\mathbf{e}$, $\mathbf{E}$	Noise vector vs. noise matrix	
η	Noise level in blurred right-hand side	(2.4)
ϵ_M	Machine precision	
$\mathbf{x}_{\text{reg}}$, $\mathbf{X}_{\text{reg}}$	Regularized solution as vector vs. image matrix	
$\text{vec}\,(\mathbf{X})$	Vectorization of image matrix	Def. 3.1
$\text{vec}^{-1}\,(\mathbf{x})$	Transforming vectorized image back to matrix form	Def. 3.2
$\mathbf{A}$	Blurring matrix	
$\mathbf{U\Sigma V}^T$	SVD of $\mathbf{A}$	(2.6)
$\mathbf{I}$	The identity matrix	
$\mathbf{L}$	General regularization matrix defining the semi-norm of the solution	(2.13)
λ	Tikhonov regularization parameter	(2.13)

$\otimes$	Kronecker product	Def. 3.3
$\mathbf{\Psi}$	Diagonal matrix containing the filter factors	(2.15)
$\hat{\mathbf{x}}$, $\widehat{\mathbf{X}}$	DCT of the vector $\mathbf{x}$, vs. the two-dimensional DCT of the image matrix $\mathbf{X}$	Def. 3.4, Def. 3.5
$\mathbf{G}$	One-dimensional DCT transformation matrix	(3.18)
$\mathbf{T}_\sigma$	Toeplitz matrix defining Gaussian blur with the parameter σ	(3.23)
$\mathbf{\Pi}$	Permutation matrix rearranging the Kronecker singular values, such that the singular values of $\mathbf{\Pi}(\mathbf{\Sigma}\otimes\mathbf{\Sigma})\mathbf{\Pi}$ appear in decaying order	(3.31)
$\mathbf{R}$	Matrix containing Euclidean distances	(3.32)
$\mathbf{P}$	Permutation matrix	p. 21
$\mathcal{K}_k(\mathbf{A}, \mathbf{b})$	Krylov subspace of dimension k, based on $\mathbf{A}$ and $\mathbf{b}$	Def. 4.1
$\mathbf{U}_{k+1}$	$m \times (k+1)$ left Lanczos vectors	(4.8)
$\mathbf{V}_k$	$m \times k$ right Lanczos vectors	(4.8)
$\mathbf{B}_k$	$(k+1) \times k$ bidiagonal matrix from Lanczos process	(4.8)
$\mathbf{P\Gamma Q}^T$	SVD of $\mathbf{B}_k$	(4.11)
$\mathbf{W}_k$	$m \times k$ basis vectors for the Arnoldi process	(4.16)
$\mathbf{H}_k$	$(k+1) \times k$ upper Hessenberg matrix from Arnoldi process	(4.16)
$\widehat{\mathbf{P}}\widehat{\mathbf{\Gamma}}\widehat{\mathbf{Q}}^T$	SVD of $\mathbf{H}_k$	(4.20)
$\boldsymbol{\phi}_u$, $\boldsymbol{\phi}_v$	Vectors of the coefficients $\mathbf{U}^T\mathbf{b}$ and $\mathbf{V}^T\mathbf{b}$, respectively	
$\mathrm{diag}(\mathbf{x})$	Diagonal matrix containing the elements of the given vector $\mathbf{x}$	
$\oslash$	Element-wise division of two vectors	
$\odot$	Element-wise product of two vectors	
$\theta^{(k)}$	Ritz values, defined as the eigenvalues of $\mathbf{B}_k^T\mathbf{B}_k$	(4.40)
$\mathbf{A}^\#$	Some regularized inverse of $\mathbf{A}$ giving rise to a certain regularized solution	
$\Omega(\mathbf{x})$	Some measure of the size of $\mathbf{x}$	(6.8)
$\mathbf{C}$	Preconditioner	(6.10)

Contents

Chapter 1

Introduction

The title of this project is composed of four words: regularizing, iterations, image, and restoration. In addition, terms as discrete ill-posed problems, and inverse problems, are central for this project. These terms need some explanation. The present chapter serves as a mostly non-mathematical introduction of the terms, and the practical relevance. Also a few implementation issues, as well as an overview of the rest of this thesis are provided.

1.1 Explaining the Terms

Starting with the last term – inverse problem – we have to imagine a system or process with an input and an output. Schematically, such a system can be sketched as in Fig. 1.1.

Going from input to output is known as the forward problem, and a lot of analysis of the system behaviour can be interesting in this case. But going the other way lead to two kinds of problems. Either estimating the system, or estimating the input – knowing the output. The latter is the most common case, and the problems studied here fall into this category. Both kind of problems are termed *inverse problems*, going against the natural direction of the arrow in Fig. 1.1. This, as we shall see, often gives a lot of troubles.

The real life example in connection with the present thesis is blurring of images. In this case, the system is some kind of blurring, and the input a real scenery. The output of the system is then a blurred realization of the scenery. Now for instance looking at astronomical images taken with a telescope, we cannot assume these images to be the true picture of the stars we are observing. More likely, what we see is the light that has been disturbed on its way through the atmosphere.

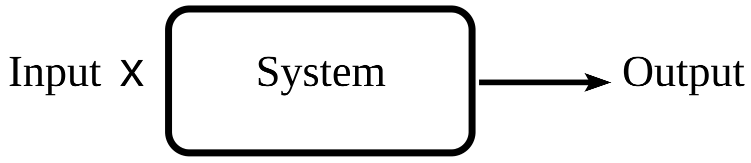


Figure 1.1: *Schematic illustration of a standard problem.*

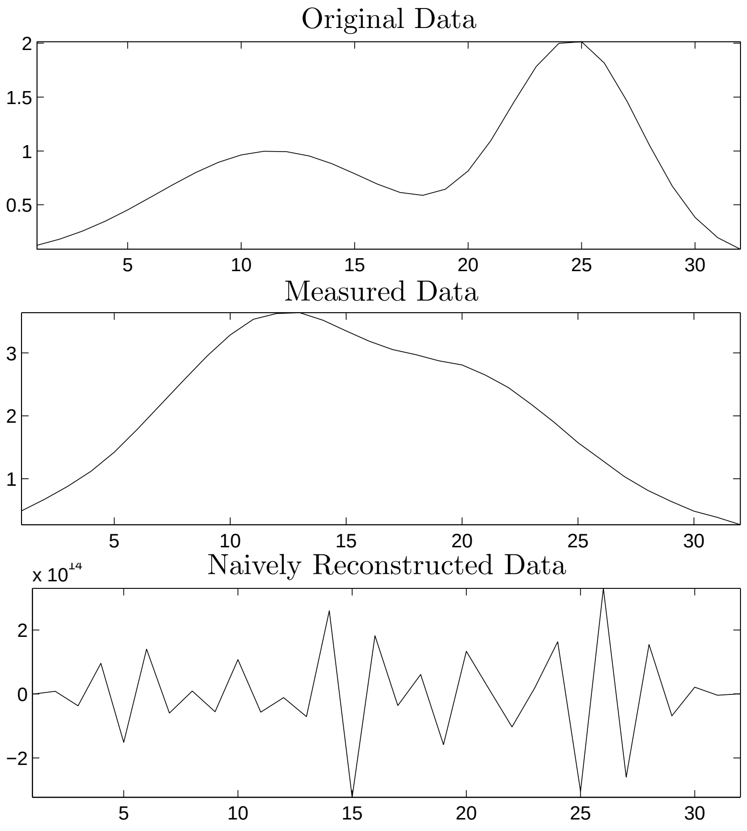


Figure 1.2: *Example showing a one-dimensional function, a blurred measurement of the function, and a naive reconstruction.*

Furthermore, the images observed are measured, thus probably containing some additional measurement errors. So what we actually have is a measure of the output from Fig. 1.1. And what we want is the input – the true image of the stars. We are left with an inverse problem, and dealing with *reconstruction* of the true image.

That the problems are *discrete* and *ill-posed* arises from a more mathematical description of the system. Discretizing the above problem can lead to a system of linear equations like:

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where here the matrix $\mathbf{A}$ is a representation of the system, the vector $\mathbf{x}$ is a representation of the input, and the vector $\mathbf{b}$ is a representation of the output. If the matrix $\mathbf{A}$ is well-conditioned, the system can be solved for $\mathbf{x}$ without problems, and we get the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. But if the matrix $\mathbf{A}$ is ill-conditioned, this means that solving the system for $\mathbf{x}$ will give great troubles. Sometimes, a reformulation of the problem leads to another matrix that is better conditioned, but for some problems, the ill-conditioning is unremovable – the discrete ill-posed problems. Solving this kind of problems naively as $\mathbf{x}_{\text{naive}} = \mathbf{A}^{-1}\mathbf{b}$ gives rise to very noisy solutions. Fig. 1.2 shows an example of a one-dimensional set of data, a simulated blurred realization of the same data – and an attempt to reconstruct the original data. As seen, the solution is unusable.

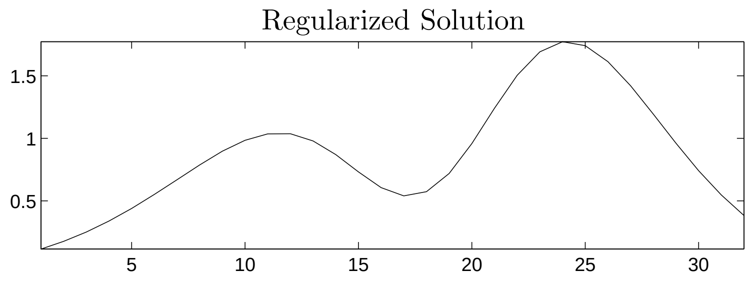


Figure 1.3: *A regularized solution of the problem in Fig. 1.2.*

To provide a more qualified guess on a solution to the discrete ill-posed problems, we try to pose some constraints on the solution, avoiding that it goes berserk as seen in Fig. 1.2. Any attempt to control the solution is called *regularization*. I.e., we find a regularized solution that is more pleasing than the naive solution. Decomposing **A** in clever ways, we can construct a solution consisting of only the wanted parts of the naive solution, and leave out the contributions from the noise. One attempt to solve the problem from Fig. 1.2 is seen in Fig. 1.3, showing a regularized solution. As seen, this solution has a nice correspondence with the original data.

However, for large problems, solving the problems directly by decompositions of **A** is cumbersome. And image problems indeed fall into this category, as we will see in the thesis. The way to go is to use *iterative* methods that iteratively find better and better solutions to the system, without need to do time and memory consuming decompositions of **A**. Problems where **A** is not known exactly, but only the effect of multiplying **A** to a vector also makes it impossible to decompose **A**. So in the case of image reconstruction, iterative algorithms are often needed. And this thesis deals with exactly the properties of using iterative algorithms for regularizing discrete ill-posed image restoration problems.

1.2 Implementation Issues

Working with this project, a lot of simulations and tests have been performed. For this purpose the mathematical software MATLAB has been widely used. And in this connection two toolboxes have been a great help. The by now classical *Regularization Tools* by P. C. Hansen [7], developed mainly for educational purposes, as well as the new toolbox, *Regularization Tools XP*, which is still work in progess and being developed by M. Jacobsen [12]. This new version is meant as a more specialized toolbox, building on Object Oriented MATLAB, which makes it possible to formulate any kind of large discrete ill-posed problems as standard linear algebra. In connection with this project, mainly the parts implementing Kronecker products and support for two-dimensional problems, such as two-dimensional vectors, as well as object oriented versions of the methods and algorithms, have been widely used. Being work in progress also means that this project to some extend has served as an α-testing of the toolbox, which has lead to changes and corrections of the toolbox.

A lot of code has been generated during the project. This covering a lot of scripts and functions for doing the various tests and simulations. In Appendix A, a few code examples are given. The GMRES algorithms has been changed during the project, and is here given with the implemented changes. Also, a bidiagonalization algorithm has been modified. To generalize the work with different test problems, a widely used function for problem generation has also been developed.

1.3 Thesis Overview

The starting point of the work done in this project was a desire to find good stopping criteria for iterative algorithms, used for image restoration. But as the project developed, it was clear that regularizing image problems yielded a lot of troubles. And before even looking at how to stop the iterative algorithms, it was necessary to gain insight into how images are reconstructed. The project therefore consists of a study of inverse image restoration problems in general, as well as iterative algorithms and the solutions they give. The problems involved in finding stopping criteria are covered in the last chapter, and the final report is organized as follows:

Chapter 1 has provided a brief overview of the background for looking at discrete ill-posed problems, as well as some implementation issues.

Chapter 2 introduces discrete ill-posed problems in a more mathematical setting, introducing some general concepts and tools, used throughout the report.

Chapter 3 looks at inverse problems in connection with images and image reconstruction. More tools and concepts are introduced and defined, and some studies using direct calculations are shown.

Chapter 4 is the first iterative chapter. The algorithms LSQR and GMRES are described and an analysis of the Krylov subspaces they work in is done. This chapter is strongly connected to the next.

Chapter 5 analyzes the solutions obtained from the iterative algorithms. Different artifacts are explained, and the nature of using iterative algorithms for reconstructing images are connected to the direct calculations, done in Chapter 3.

Chapter 6 discusses different ways to modify the two basic algorithms to avoid some of the artifacts analyzed in Chapter 5. Krylov subspaces and filter factors for the different approaches are compared and discussed.

Chapter 7 finally looks at how to stop the iterative algorithms after an appropriate number of iterations. The chapter shows some of the difficulties that arise for the image reconstruction problems.

Chapter 8 concludes the work done by summarizing the results and giving suggestions for future work.

Last, but not least, the original images used throughout the report, as well as some of the code generated working with the project, are found in the appendices.

CHAPTER 2

Discrete Ill-posed Problems

This project concerns solving linear systems of equations, which in principle should be a simple task. But here the focus is directed towards linear systems that are hard to solve – namely systems having a large condition number. Furthermore, the main focus will be on two-dimensional images. In many cases problems with large condition numbers arise if the formulation of the model is bad or wrong, and the solution is therefore to reformulate the problem. But in some cases, the underlying problem is also ill-posed, and the linear algebra formulation inherits this ill-posedness without any chance to get rid of it. This is for instance the case in situations where the underlying problem can be formulated as a first order Fredholm integral equation:

$$\int_0^1 K(s,t)f(t)dt = g(s), \quad 0 \le s \le 1, \tag{2.1}$$

where t and s in this case are normalized to lie in the interval $[0,1]$. The integration with the kernel $K(s,t)$ is in functional analysis known to be a compact operator $\mathcal{K}$, and the forward calculation of knowing $K(s,t)$ and $f(t)$ and getting to $g(s)$ is a bounded operation. On the other hand, wanting to solve the inverse problem by calculating $f(t)$ from $K(s,t)$ and $g(s)$, we need to invert the operator, which gives the unbounded operator $\mathcal{K}^{-1}$. That this operator is unbounded means that a tiny perturbation of the data in $g(s)$ might give a huge difference in the solution $f(t)$. This is the ill-posed property that carries over to the discretized linear algebra setting:

$$\mathbf{A}\mathbf{x} = \bar{\mathbf{b}}, \tag{2.2}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\bar{\mathbf{b}} \in \mathbb{R}^m$. The ill-posed property gives rise to $\mathbf{A}$ having a huge condition number.

In (2.2), $\bar{\mathbf{b}}$ is the exact right-hand side, which is very seldom known. In a true setting, the right-hand side is often measured, and therefore it contains measurement noise. In this case, the right-hand side is not $\bar{\mathbf{b}}$, but $\bar{\mathbf{b}} + \mathbf{e}$, yielding the expression:

$$\mathbf{A}\mathbf{x} = \bar{\mathbf{b}} + \mathbf{e} = \mathbf{b}, \tag{2.3}$$

where $\mathbf{e}$ is a noise vector, often considered to be normal distributed white noise with zero mean, $\mathcal{N}(0,\sigma)$. The variance of the noise σ is assumed to be not too big, which in this case means $\|\mathbf{e}\|_2 \leq \|\bar{\mathbf{b}}\|_2$. If the noise gets larger, the measured signal will be completely covered in noise, and no approximate solution can be found. Throughout the rest of this work, the signal to noise ratio, or noise level, will refer to the quantity

$$\eta = \frac{\|\mathbf{e}\|_2}{\|\bar{\mathbf{b}}\|_2}. \tag{2.4}$$

In the case of (2.3), $\mathbf{b}$ cannot even be expected to belong to the range of $\mathbf{A}$, $\mathcal{R}(\mathbf{A})$, and no exact solution to the system exists. Instead it might be useful to look at the *least squares* formulation:

$$\mathbf{x}_{\text{LS}} = \min_x \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2, \tag{2.5}$$

that minimizes the two-norm of the residual.

2.1 Singular Value Decomposition

The condition number of $\mathbf{A}$ from (2.2) and (2.3) is often large, which yields a lot of problems in solving the systems. Valuable tools for analyzing the matrix $\mathbf{A}$ are different kinds of decompositions. What we want are rank revealing decompositions that provide information about the subspaces carrying information about the wanted solutions. A widely used decomposition is the *Singular Value Decomposition* (SVD). The SVD of a matrix is the following:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=0}^{n} \mathbf{u}_i \sigma_i \mathbf{v}_i^T, \tag{2.6}$$

where $\mathbf{U} \in \mathbb{R}^{m\times n}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$, and $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$. Furthermore, the matrices $\mathbf{U}$ and $\mathbf{V}$ consist of orthonormal vectors and are called the left and right singular matrices, containing the left and right singular vectors, $\mathbf{u}_i$ and $\mathbf{v}_i$. The matrix $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values, which are positive and non-increasing, down the main diagonal, so that:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & & 0 \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \sigma_n \end{bmatrix} \quad , \quad \sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0 \tag{2.7}$$

The SVD of a matrix gives a great deal of insight about the number of linearly independent vectors. Effectively, the number of nonzero σ_i's gives the rank of the matrix. So if:

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_n = 0, \tag{2.8}$$

then $r = \text{rank}(\mathbf{A})$. Working on computers, the singular values will properly never be exact zero, but roll off at $\sigma_1 \epsilon_M$, where ϵ_M is the machine precision.

Apart from the rank revealing properties of the SVD, it is also justified in the literature that for discrete ill-posed problems, it has a frequency decomposing property. This means that looking at the singular vectors, these have more and more

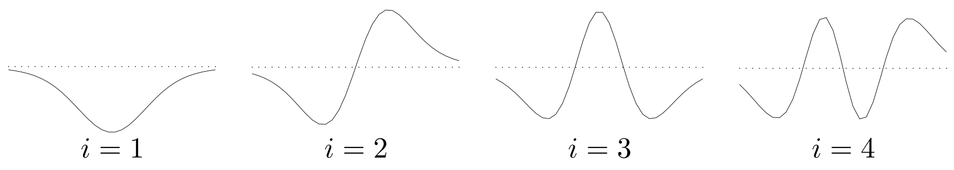


Figure 2.1: *First four left singular vectors* $\mathbf{u}_i$*,* $i = 1, 2, 3, 4$ *showing the increasing number of zero crossings, i.e. the increase in frequency.*

zero crossings as the index i increases. The first vectors are generally low-frequent, slowly varying, and the later are higher-frequent, faster varying. An example on this property is showed in Fig. 2.1, which is heavily inspired by the analysis in [8, Section 2.1.3]. The right singular vectors shown arise from the standard test problem shaw taken from M. Jacobsen's MATLAB Toolbox, Regularization Tools XP [12] (first implemented by P. C. Hansen in [7]). The matrix $\mathbf{A}$ is of size 32×32.

This interesting property shows that the subspaces carrying the most information are low-frequent, whereas the high-frequent vectors correspond mainly to small singular values.

2.1.1 Inversion of Matrices

For matrices, $\mathbf{A} \in \mathbb{R}^{n \times n}$ that are square and non-singular, an inverse $\mathbf{A}^{-1}$ exists. This is easily described in terms of the SVD, and can be written as:

$$\mathbf{A}^{-1} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^{-1} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \sum_{i=1}^{n} \mathbf{v}_i \sigma_i^{-1} \mathbf{u}_i^T. \tag{2.9}$$

Here the relationship between the well-conditioning, and the fact that none of the singular values are equal to zero, is obvious, as $\mathbf{A}^{-1}$ is simply the sum of outer products of orthonormal vectors, weighted by $1/\sigma_i$ for $i = 1, 2, \ldots, n$.

If, on the other hand, $\mathbf{A}$ is rank-deficient and/or rectangular, both of which can apply to inverse problems, the inverse is not defined. One way to generalize the concept of matrix inverse is to use a pseudo-inverse by truncating the sum in (2.9), only including elements until reaching the rank of $\mathbf{A}$, $r = \operatorname{rank}(\mathbf{A})$. This kind of pseudo-inverse is known as the Moore Penrose pseudo-inverse, and is given by:

$$\mathbf{A}^{\dagger} = \sum_{i=1}^{r} \mathbf{v}_i \sigma_i^{-1} \mathbf{u}_i^T, \tag{2.10}$$

Solving (2.3) with this inverse gives the least squares solution to the problem, defined in (2.5). For more information, see for instance [4, Section 5.5.3–5.5.4].

2.1.2 Picard Plot

Now we turn to look at the solutions obtained, to get a better handle for describing the ill-posedness of the system. Assuming now that $\mathbf{A} \in \mathbb{R}^{n \times n}$ and using the inverse of $\mathbf{A}$, defined by means of the SVD of $\mathbf{A}$ in (2.9), we get:

$$\begin{aligned}\mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ &= \sum_{i=1}^{n} \mathbf{v}_i \sigma_i^{-1} \mathbf{u}_i^T \mathbf{b} \\ &= \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i. \end{aligned} \tag{2.11}$$

For (2.11) to be well defined, intuitively the solution coefficients $\mathbf{u}_i^T\mathbf{b}$ must decay faster than the singular values σ_i, so that the solution does not blow up. This is indeed often the case for the unperturbed right-hand side, $\bar{\mathbf{b}}$. For the measured right-hand side the situation is different, as the noise is not restricted to be low-frequent, but covers the whole spectrum. Remembering that the singular vectors increase in frequency with the index i, it is clear that the first vectors will be dominated by the signal contents, whereas the last vectors will be dominated by the noise components. The coefficients $|\mathbf{u}_i^T\mathbf{b}|$ are therefore expected to level off at some value connected to the noise level. The plot obtained by plotting those coefficients along with the singular values σ_i as well as the solution coefficients $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$, is known as a Picard plot.

For an illustration of a *Picard plot*, see Fig. 2.2. Here the singular values, the coefficients, and the solution coefficients, are shown for the small test problem `shaw`. White noise of size $\eta = 10^{-5}$ has been added to the true right hand side $\bar{\mathbf{b}}$. As seen, the solution coefficients level off from one point onwards. Therefore we cannot expect to get an informative solution including components beyond the point where these coefficients level off.

2.2 Regularized Solutions

The term *regularized solution* is in general used for a solution to the equation (2.3), where the ill-posedness of the problem is somehow controlled. As rank($\mathbf{A}$) is not full, $\mathbf{A}$ will have a *null space* of some dimension. The null space of a matrix is defined by:

$$\mathrm{null}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{0}\}. \tag{2.12}$$

This implies that a solution $\mathbf{x}$, partly belonging to this subspace of $\mathbf{A}$, will be able to take on very large values, not changing the residual norm $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$. So even though the residual is small, the solution might be useless, and in the general setting, we want to regularize the solution by imposing some constraints on the solution.

A standard method of regularizing the solution is the well-known *Tikhonov regularization.* This method suggests exactly to balance the residual norm $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ and some (semi)norm of the solution $\|\mathbf{L}\mathbf{x}\|_2$:

$$\mathbf{x}_{\mathrm{reg}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda^2 \|\mathbf{L}\mathbf{x}\|_2^2\right\}. \tag{2.13}$$

Often $\mathbf{L}$ is chosen to be the identity matrix $\mathbf{I}$, but letting $\mathbf{L}$ be an approximation to the first or second order derivative operator, it is possible to constrain the derivatives of the regularized solution.

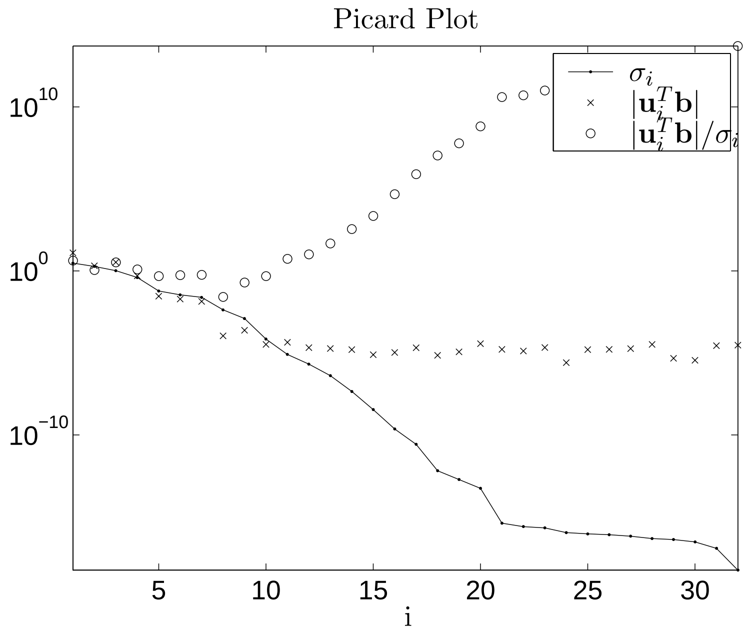


Figure 2.2: *Picard plot for simple* shaw *test problem, showing singular values σ_i, solution coefficients $|\mathbf{u}_i^T\mathbf{b}|$, and the ratio between them. The noise level is set to $\eta = 10^{-1}$.*

As seen from the Picard plot, there are two things giving rise to troubles. The often dominating is that the solution coefficients for the measures noisy right-hand side level off. But for very small noise levels, the problem is the numerical null space of $\mathbf{A}$, where $\sigma_i \approx \epsilon_M$, which spoils the solution. What we obviously want is to choose wanted parts of the spectrum, and filter out the noisy parts. To do this, the concept of filter factors is now introduced.

2.2.1 Filter Factors

Using the Picard plot, it is clear that we want to select the components corresponding to the left part of the plot, and filter away the right part that is dominated by noise. In a general setting, this can be achieved by introducing some general filter factors, ψ. Then the regularized solution can be defined as:

$$\begin{aligned} \mathbf{x}_{\text{reg}} &= \sum_{i=1}^{n} \psi_i \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i & (2.14)\\ &= \mathbf{V\Psi\Sigma}^{-1}\mathbf{U}^T\mathbf{b}, & (2.15) \end{aligned}$$

where $\mathbf{\Psi}$ is a diagonal matrix having the filter factors ψ_i down the diagonal. The least squares solution is seen to have the same formulation with the filter factors:

$$\psi_i = \begin{cases} 1 & \text{for} \quad i < r \\ 0 & \text{for} \quad i \geq r \end{cases} \tag{2.16}$$

where r is the rank of $\mathbf{A}$.

2.2.2 Truncated SVD

The filter factors can be chosen in several ways. The most intuitive and direct approach might be just to cut off the unwanted SVD components, which is known as the *truncated SVD* (TSVD) approach:

$$x_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i. \tag{2.17}$$

This is the general regularized solution (2.15) with the filter factors chosen to be:

$$\psi_i = \begin{cases} 1 & \text{for} \quad i < k \\ 0 & \text{for} \quad i \geq k \end{cases}, \tag{2.18}$$

where k is chosen to be not the r, but a value corresponding to the place where the noise enters the solution coefficients. For the test example in Fig. 2.2, $k \approx 9$ seems a wise choice as this is the point where the solution coefficients level off at the noise level.

2.2.3 Tikhonov

While the TSVD approach might seem a bit rough, another possibility is to return to the Tikhonov formulation in (2.13). Two alternative formulations exist for the Tikhonov problem:

$$(\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{L}^T \mathbf{L})\mathbf{x} = \mathbf{A}^T \mathbf{b} \quad \text{and} \quad \min \left\| \begin{pmatrix} \mathbf{A} \\ \lambda \mathbf{L} \end{pmatrix} \mathbf{x} - \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \right\|_2. \tag{2.19}$$

From the first expression in (2.19) above, it is seen that the Tikhonov inverse of $\mathbf{A}$ for a given value of λ can be written as:

$$\mathbf{A}_\lambda^\# = (\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{L}^T \mathbf{L})^{-1} \mathbf{A}^T. \tag{2.20}$$

In the case where $\mathbf{L} = \mathbf{I}$, the filter factors can be calculated by using the SVD of $\mathbf{A}$. The regularized solution is written:

$$\begin{aligned} \mathbf{x}_{\text{reg}} &= \mathbf{A}_\lambda^\# \mathbf{b} \\ &= (\mathbf{A}^T \mathbf{A} + \lambda^2 \mathbf{I})^{-1} \mathbf{A}^T \mathbf{b} \\ &= (\mathbf{V\Sigma U}^T \mathbf{U\Sigma V}^T + \lambda^2 \mathbf{I})^{-1} (\mathbf{U\Sigma V}^T)\mathbf{b} \\ &= (\mathbf{V\Sigma}^2 \mathbf{V}^T + \lambda^2 \mathbf{I})^{-1} (\mathbf{U\Sigma V}^T)\mathbf{b} \\ &= \sum_{i=1}^{n} \frac{1}{\sigma_i^2 + \lambda^2} \mathbf{u}_i^T \mathbf{b} \mathbf{v}_i \\ &= \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \lambda_i} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i^2} \mathbf{v}_i. \end{aligned} \tag{2.21}$$

From (2.21) it is seen that the filter factors are given by:

$$\psi_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}. \tag{2.22}$$

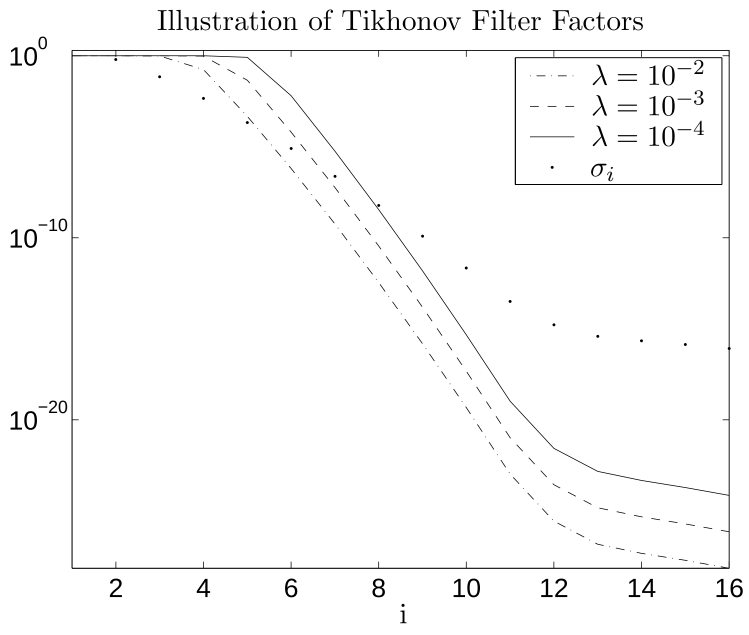


Figure 2.3: *Illustration of the Tikhonov filter factors $\phi_i = \sigma_i^2/(\sigma_i^2 + \lambda^2)$ for a small test problem. The value of λ is seen to change the cut off in the filter factors, so that lower λ gives a higher cut off point.*

The overall behavior of those filter factors is the same as the filter factors for the TSVD, i.e., for the large σ_i's, the filter factors are approximately equal to one, whereas for the small σ_i's, the filter factors are close to zero. The transition between large and small filter factors though is different. Here the value λ defines a point in the spectrum from where the filter factors fall off with approximately σ_i^{-2}. An illustration of those filter factors for a small test problem, is seen in Fig. 2.3.

Chapter 3

Digital Images and Inverse Problems

As briefly noted in the last chapter, the main focus of this thesis is on images. So now the basic signals are two-dimensional gray-scale images. In the real world, those can be described by two-dimensional functions of intensities, $f(s,t)$, describing a scenery. Looking at image de-blurring, we need a mathematical model for the blurring. Often, the blurring of images can be modeled as Fredholm integral equations, similar to the one-dimensional case in (2.1), as:

$$\int_0^1 \int_0^1 K(u,v,s,t)\, f(s,t)\, dsdt = g(u,v) \quad 0 \le u,v \le 1, \tag{3.1}$$

where now $K(u,v,s,t)$ is a two-dimensional kernel. The ill-posed properties described for the one-dimensional case are the same for this two-dimensional version. Discretizing the problem, the images become two-dimensional arrays of intensities. For simplicity, the images are assumed to be square, and thus a digital image can be represented by the matrix $\mathbf{X} \in \mathbb{R}^{N\times N}$. To be able to formulate the problem on the basic form (2.3), we need the following notation to denote the vector containing the stacked columns of an image matrix $\mathbf{X}$ where $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_n\end{bmatrix}$:

Definition 3.1 *Definition of the* vec *operator transforming a two-dimensional image into a vector by stacking the columns:*

$$\mathrm{vec}\,(\mathbf{X}) \equiv \begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n\end{bmatrix}.$$

Note that information about the dimensions of the image is lost in the vectorized form. Only in the case of square images, the dimensions of the original image are known to be $N \times N$, where N^2 is the length of the vectorized image.

Talking about notation, at a later point, we need a similar notation to get back from the vectorized image to the two-dimensional form, and we define:

Definition 3.2 *Definition of the* vec^{-1} *operator for transforming a column-wise stacked image back to a two-dimensional representation:*

$$\mathrm{vec}^{-1}\left(\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\\ \vdots\\ \mathbf{x}_n\end{bmatrix}\right) = \mathbf{X}$$

As noted in Definition 3.1, the dimensions of the original image $\mathbf{X}$ *are lost. In practice additional information about those dimensions is needed in case of rectangular images.*

Using Definition 3.1, the general two-dimensional inverse problem can be formulated similar to (2.3) as:

$$\mathbf{A}\mathrm{vec}\,(\mathbf{X}) = \mathrm{vec}\,(\mathbf{B})\,, \tag{3.2}$$

where again $\mathbf{A}$ is some discretization of the blurring kernel $K(u,v,s,t)$, $\mathrm{vec}\,(\mathbf{X})$ is the vectorized true image, and $\mathrm{vec}\,(\mathbf{B})$ is the vectorization of the blurred, measured image. Note that the two vectorized images are vectors of length N^2 and that the blurring matrix $\mathbf{A} \in \mathbb{R}^{N^2\times N^2}$. Similar to the one-dimensional case, the measured image $\mathbf{B}$ is assumed to contain additive white noise, so that:

$$\mathbf{B} = \overline{\mathbf{B}} + \mathbf{E}, \tag{3.3}$$

where $\overline{\mathbf{B}}$ is the true blurred image $\mathbf{A}\mathrm{vec}\,(\mathbf{X})$, and the elements of $\mathbf{E} \in \mathcal{N}(0,\sigma)$ are additive white noise. Where nothing is given, the norm of an image is the two-norm of the vectorized image or equivalently the Frobenius-norm of the two-dimensional array. The noise level is then given as:

$$\eta = \frac{\|\mathrm{vec}\,(\mathbf{E})\,\|_2}{\|\mathrm{vec}\,(\overline{\mathbf{B}})\,\|_2} = \frac{\|\mathbf{E}\|_F}{\|\overline{\mathbf{B}}\|_F}. \tag{3.4}$$

To keep notation short, the vectorized images $\mathrm{vec}\,(\mathbf{X})$ is also in the following denoted simply by $\mathbf{x}$, which makes the formulation of the problems equal to the one-dimensional case. Looking at these vectorized images, it is clearly seen why these problems are large scale problems. Within many application areas, images of size 512×512 or 1024×1024 or even bigger are very common. In the last case this leads to blurring matrices larger than $10^6 \times 10^6$ which are extremely un-handy to work with and even to store. Fortunately, it is often assumed that the blurring is spatially invariant in which case the blurring matrix $\mathbf{A}$ is block Toeplitz with Toeplitz blocks (BTTB) [15, Section 1]. Furthermore, a special case of the spatially invariant blur is when the blurring seperates in the two directions, vertical and horizontal. In this case, the blurring in each direction can be written separately and combined by means of the Kronecker product formulation.

3.1 The Kronecker Product

The *Kronecker Product* of two matrices, $\mathbf{A}_1 \in \mathbb{R}^{m\times n}$ and $\mathbf{A}_2 \in \mathbb{R}^{p\times q}$ is defined as:

Definition 3.3 *Definition of the Kronecker product of two matrices* $\mathbf{A}_1 \in \mathbb{R}^{m \times n}$ *and* $\mathbf{A}_2 \in \mathbb{R}^{p \times q}$*:*

$$\mathbf{A}_1 \otimes \mathbf{A}_2 = \begin{bmatrix} \mathbf{A}_{1[1,1]}\mathbf{A}_2 & \mathbf{A}_{1[1,2]}\mathbf{A}_2 & \dots & \mathbf{A}_{1[1,n]}\mathbf{A}_2 \\ \mathbf{A}_{1[2,1]}\mathbf{A}_2 & \mathbf{A}_{1[2,2]}\mathbf{A}_2 & \dots & \mathbf{A}_{1[2,n]}\mathbf{A}_2 \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_{1[m,1]}\mathbf{A}_2 & \mathbf{A}_{1[m,2]}\mathbf{A}_2 & \dots & \mathbf{A}_{1[m,n]}\mathbf{A}_2, \end{bmatrix},$$

where $\mathbf{A}_{1[i,j]}$ *denotes element* (i, j) *of the matrix* $\mathbf{A}_1$*. The size of the resulting matrix is* $mp \times nq$*.*

A few important relations hold true for the Kronecker product, namely:

$$\begin{aligned} (\mathbf{A}_1 \otimes \mathbf{A}_2)^T &= \mathbf{A}_1^T \otimes \mathbf{A}_2^T & (3.6) \\ (\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{B}_1 \otimes \mathbf{B}_2) &= (\mathbf{A}_1\mathbf{B}_1) \otimes (\mathbf{A}_2\mathbf{B}_2) & (3.7) \end{aligned}$$

But at this point, the most important relation is the following that combines the vectorized images and the Kronecker product notation:

$$\text{vec}\left(\mathbf{A}_2\mathbf{X}\mathbf{A}_1^T\right) = (\mathbf{A}_1 \otimes \mathbf{A}_2)\,\text{vec}\,(\mathbf{X})\,. \tag{3.8}$$

This relation shows that in the cases where the blurring separates in one blurring of the columns of the image, and one blurring of the rows, the large blurring matrix $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$ can be described by a Kronecker product of two much smaller matrices $\mathbf{A}_1$ and $\mathbf{A}_2 \in \mathbb{R}^{N \times N}$.

Using this last relation, it is obvious that the ill-posed problem from (3.2), in the case of separable blurring, can be written in one of the three following ways:

$$\begin{aligned} (\mathbf{A}_1 \otimes \mathbf{A}_2)\text{vec}\,(\mathbf{X}) &= \text{vec}\,(\mathbf{B}) & (3.9) \\ \text{vec}\left(\mathbf{A}_2\mathbf{X}\mathbf{A}_1^T\right) &= \text{vec}\,(\mathbf{B}) & (3.10) \\ \mathbf{A}_2\mathbf{X}\mathbf{A}_1^T &= \mathbf{B}, & (3.11) \end{aligned}$$

where (3.9) makes the problem look at in the one-dimensional case, whereas (3.10) and (3.11) serves for faster computation. The Kronecker product is implemented in the Regularization Tools XP [12], and the fast calculations are performed in practise.

3.1.1 Kronecker SVD

The Kronecker product has another nice property in connection with the SVD, namely that the SVD of a Kronecker product can be calculated by means of the SVD of the smaller matrices constructing the Kronecker product. If

$$\mathbf{A}_1 = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T \quad \text{and} \quad \mathbf{A}_2 = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T \tag{3.12}$$

are the SVD's of $\mathbf{A}_1$ and $\mathbf{A}_2$, then it follows from (3.7) that:

$$(\mathbf{A}_1 \otimes \mathbf{A}_2) = (\mathbf{U}_1 \otimes \mathbf{U}_2)(\mathbf{\Sigma}_1 \otimes \mathbf{\Sigma}_2)(\mathbf{V}_1 \otimes \mathbf{V}_2)\,. \tag{3.13}$$

This is exactly the SVD of $(\mathbf{A}_1 \otimes \mathbf{A}_2)$, except for the ordering of the singular vectors and values that do not appear in a strictly decaying order. But rearranging the matrices by simple permutations, the true SVD is obtained.

3.2 Discrete Cosine Transform

The last tool to introduce is a tool for analyzing the frequency content in the signals. A standard transform to use is the well known *Fourier Transform*, but in connection with images, another widely used transform is the *Cosine Transform*, and in the discrete case, the *Discrete Cosine Transform* (DCT). The following definition introduces the DCT:

Definition 3.4 *Definition of the Discrete Cosine Transform, transforming a vector* $\mathbf{x} \in \mathbb{R}^N$ *from a spatial representation to a frequency representation:*

$$\hat{\mathbf{x}}_j = c_j \sum_{i=1}^{N} \mathbf{x}_i \cos\left(\frac{\pi(2i-1)(j-1)}{2N}\right), \quad \text{for} \quad j = 1, 2, \ldots, N$$

$$c_j = \begin{cases} \sqrt{\frac{1}{N}} & \text{for} \quad j = 1 \\ \sqrt{\frac{2}{N}} & \text{for} \quad j = 2, 3, \ldots, N \end{cases}$$

Here the hat ($\hat{}$) *denotes the DCT-transformed vector.*

For the images, we need the two-dimensional DCT, which is just a generalization of the one-dimensional DCT. The transform is separable and defined as:

Definition 3.5 *Definition of the two-dimensional Discrete Cosine Transform, transforming an image* $\mathbf{X} \in \mathbb{R}^{N \times N}$ *from its spatial representation to a frequency representation:*

$$\widehat{\mathbf{X}}_{[p,q]} = c_p c_q \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{X}_{[i,j]} \cos\left(\frac{\pi(2i-1)(p-1)}{2N}\right) \cos\left(\frac{\pi(2j-1)(q-1)}{2N}\right)$$

$$c_i = \begin{cases} \sqrt{\frac{1}{N}} & \text{for}\, i = 1 \\ \sqrt{\frac{2}{N}} & \text{for}\, i = 2, 3, \ldots, N \end{cases}$$

In the notation, $\mathbf{X}_{[i,j]}$ *denotes the element* (i, j) *of the matrix* $\mathbf{X}$*. The variables* i *and* j *are discrete and run from* 1 *to* N*, so the transformed image,* $\widehat{\mathbf{X}}$*, is of the same size as the original image* $\mathbf{X}$*.*

As mentioned, the two-dimensional DCT is separable, and deriving the orthogonal transformation matrix, $\mathbf{G}$, from Definition 3.4, we get:

$$\mathbf{G}_{[i,j]} = c_j \cos\left(\frac{\pi(2i-1)(j-1)}{2N}\right), \quad \text{for} \quad j = 1, 2, \ldots, N. \tag{3.18}$$

Using (3.18), we can now write the DCT of the vector $\mathbf{x}$ as a matrix-vector product:

$$\hat{\mathbf{x}} = \mathbf{G}^T \mathbf{x}, \tag{3.19}$$

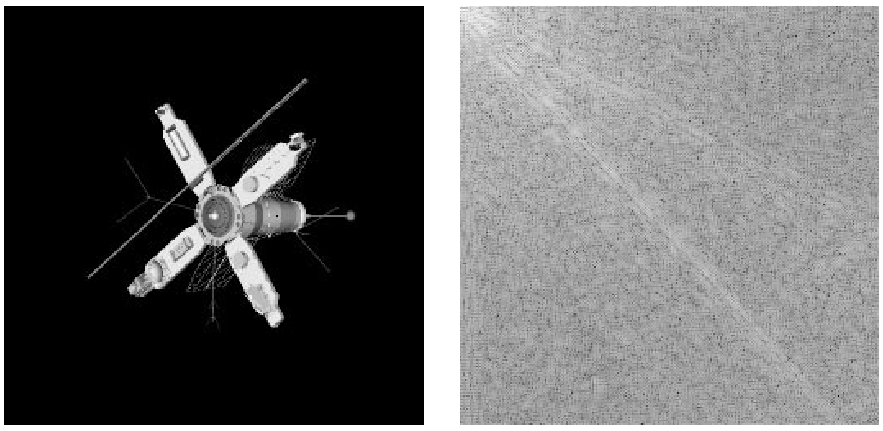

(a) (b)

Figure 3.1: *Illustration of two-dimensional DCT. (a) Is the original test image* $\mathbf{X}$*, and (b) is a visualization of the 2D-DCT of* $\mathbf{X}$*,* $\log|\widehat{\mathbf{X}}|$*. The logarithm is used while the low frequencies in the upper left corner are much more intense than the higher frequencies, and most of the image would have been black if the logarithms was not used.*

which in turn can be used to calculate the two-dimensional DCT of an image $\mathbf{X}$ as:

$$\begin{aligned} \widehat{\mathbf{X}} &= \mathbf{G}^T \mathbf{X} \mathbf{G} \Leftrightarrow && (3.20) \\ \mathrm{vec}\left(\widehat{\mathbf{X}}\right) &= \left(\mathbf{G}^T \otimes \mathbf{G}^T\right) \mathrm{vec}\left(\mathbf{X}\right). && (3.21) \end{aligned}$$

An illustration of the results of using the 2D-DCT is shown in Fig. 3.1. Here the spatial representation of the image, $\mathbf{X}$, is shown to the left, and its 2D-DCT, $\widehat{\mathbf{X}}$ is shown to the right. To get more information, the logarithm is applied to the $\widehat{\mathbf{X}}$ as many of the high frequency components are very small compared to the low frequency components. So what is actually shown is $\log|\widehat{\mathbf{X}}|$. In the figure, the lowest frequencies lie in the upper left corner, whereas the highest frequencies lie in the lower right. In between, the intensity in each point denotes the strength of each frequency component in a given direction. For instance, the almost diagonal lines in the frequency spectrum come from the many sharp edges on the satellite having this orientation. To represent sharp edges where the intensity change almost instantaneously from bright white to black, all frequencies are needed, as seen.

3.3 Image Blurring

A number of different types of blurring can apply to images. The most studied kind is the *atmospheric turbulence blur* [9, Section 6.4], which e.g. applies to a great number of astronomical image deblurring problems. The atmospheric turbulence blur is simply described by a two-dimensional Gaussian function, defined by the kernel:

$$K(u,v,s,t) = \frac{1}{\sqrt{2\pi\sigma_c\sigma_r}} \exp\left(-\frac{1}{2}\left(\frac{u-s}{\sigma_c}\right)^2 - \frac{1}{2}\left(\frac{v-t}{\sigma_r}\right)^2\right), \qquad (3.22)$$

where σ_c and σ_r are constants describing the amount of blurring of the columns and the rows, respectively.

3.3.1 Spatially Invariant Blur

This kind of blurring is assumed to be the same for all the points in an image. In case of atmospheric turbulence blur, the blurring is separable and can be formulated as the Kronecker product of two Toeplitz matrices, defined by:

$$\mathbf{T}_{\sigma[i,j]} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{i-j}{\sigma}\right)^2\right). \tag{3.23}$$

This is simple to implement, as only the first column of the two Toeplitz matrices are needed to completely specify the blurring. Furthermore, this vector can be truncated at a specified bandwidth as the coefficients fall off towards zero, often making the matrix sparse. The blurring is then given by the Kronecker product:

$$\mathbf{A} = \left(\mathbf{T}_{\sigma_c} \otimes \mathbf{T}_{\sigma_r}\right), \tag{3.24}$$

where $\mathbf{T}_{\sigma_c}$ defines the blurring in the columns, and $\mathbf{T}_{\sigma_r}$ defines the blurring in the rows. If $\mathbf{T}_{\sigma_c} = \mathbf{T}_{\sigma_r}$, the blurring is circular, and if $\mathbf{T}_{\sigma_c} \neq \mathbf{T}_{\sigma_r}$, the blurring is elliptical. But in both cases, the resulting Kronecker product $\mathbf{A}$ is a symmetric matrix.

An illustration of spatially invariant atmospheric turbulence blur is given in Fig. 3.2 (a) for $\mathbf{T}_{\sigma_c} = \mathbf{T}_{\sigma_r}$ and in (b) for $\mathbf{T}_{\sigma_c} \neq \mathbf{T}_{\sigma_r}$. The true image is found in Appendix B, Fig. B.3.

3.3.2 Spatially Variant Blur

The spatially variant blur is a blurring that varies from one part of the image to another, which in fact often is the case. Assuming that the blurring is invariant is often done to get to the simple formulation in (3.24), but being able to solve systems with variant blur is anyway an important issue. In this project, one kind of spatially variant blur is adopted from a paper by D. Calvetti et al. [2]. If $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$ then the blurring is defined as:

$$\mathbf{A} = \mathbf{I_1}\left(\mathbf{T}_1 \otimes \mathbf{T}_1\right) + \mathbf{I}_2\left(\mathbf{T}_2 \otimes \mathbf{T}_2\right), \tag{3.25}$$

where $\mathbf{T}_1 \in \mathbb{R}^{N \times N}$ and $\mathbf{T}_2 \in \mathbb{R}^{N \times N}$ are two different Toeplitz matrices describing Gaussian blur following (3.24), and $\mathbf{I}_1$ and $\mathbf{I}_2$ are described by:

$$\mathbf{I}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{I}_2 = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{3.26}$$

where $\mathbf{I}$ is the identity matix of size $N^2/2 \times N^2/2$.

If $\mathbf{T}_1$ and $\mathbf{T}_2$ are different, this blurring results in one circular blurring in the left part of the image and another in the right part. Now $\mathbf{A}$ is not a simple Kronecker product, but a combination of two Kronecker product. Calculating the Kronecker SVD and solving a large inverse problem with this matrix directly is no longer possible.

An illustration of this kind of spatially variant blur is seen in Fig. 3.2 (c).

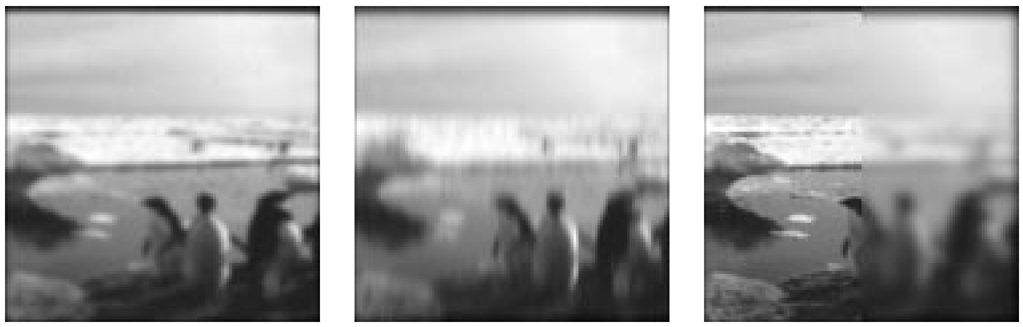


Figure 3.2: *Illustration of (a) spatially invariant blur with $\sigma_c = \sigma_r = 4.5$, (b) spatially invariant and elliptical blur with $\sigma_c = 2$ and $\sigma_r = 10$ (more blurring across rows than across columns), and (c) spatially variant blur as defined in (3.25) with $\sigma_c = \sigma_r = 2$ in the left side of the image, and $\sigma_c = \sigma_r = 10$ in the right side of the image.*

3.4 Regularized Solution in 2D

Similar to the one-dimensional case, a solution can be defined by using the filter factors as in (2.15):

$$\mathrm{vec}\left(\mathbf{X}_{\mathrm{reg}}\right) = \mathbf{V}\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathrm{vec}\left(\mathbf{B}\right). \tag{3.27}$$

Using this as well as one of the properties of the Kronecker product (3.8), the solution can be written in terms of the right singular vectors $\mathbf{V}_1$ and $\mathbf{V}_2$ as well as the filtered solution coefficients $\mathbf{C}$:

$$\begin{aligned}\mathbf{X}_{\mathrm{reg}} &= \mathbf{V}_2\boldsymbol{\Psi}_2\boldsymbol{\Sigma}_2^{-1}\mathbf{U}_2^T\mathbf{B}\mathbf{U}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Psi}_1\mathbf{V}_1^T \\ &= \mathbf{V}_2\mathbf{C}\mathbf{V}_1^T,\end{aligned} \tag{3.28}$$

where $\mathbf{C} = \boldsymbol{\Psi}_2\boldsymbol{\Sigma}_2^{-1}\mathbf{U}_2^T\mathbf{B}\mathbf{U}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Psi}_1$ contains the filtered solution coefficients in the SVD basis.

Now looking at the regularized solution, and combining this with the 2D-DCT, we can study the frequency content:

$$\mathbf{X}_{\mathrm{reg}}^G = \mathbf{G}^T\mathbf{X}_{\mathrm{reg}}\mathbf{G} = \mathbf{G}^T\mathbf{V}_2\mathbf{C}\mathbf{V}_1^T\mathbf{G} \tag{3.29}$$

$$\mathrm{vec}\left(\mathbf{X}_{\mathrm{reg}}^G\right) = \mathrm{vec}\left(\mathbf{G}^T\mathbf{X}_{\mathrm{reg}}\mathbf{G}\right) = \left[(\mathbf{G}^T\mathbf{V}_1)\otimes(\mathbf{G}^T\mathbf{V}_2)\right]\mathrm{vec}\left(\mathbf{C}\right) \tag{3.30}$$

It is seen that the solution coefficients $\mathbf{C}$ are transformed by the Kronecker product of the DCT transformation matrix and the right singular vectors of $\mathbf{A}$, contained in the right singular matrices $\mathbf{V}_1$ and $\mathbf{V}_2$. This transformation of the singular vectors allows for a combination of the frequency domain analysis and the SVD analysis. In the following section a study of this special transformation is carried out.

3.4.1 DCT and SVD

In the one-dimensional case, each singular vector can be plotted in a 2D-graph as seen in the literature, e.g. [8, Fig. 2.2]. The frequency contents in the singular vectors can like-wise be shown in a 2D-graph. Looking at all the singular vectors at the same time, i.e. looking at the full matrix $\mathbf{V}$ or some frequency representation of this matrix, such as $\mathbf{G}^T\mathbf{V}$ when using the DCT, a 3D-plot is obtained. This method for studying the singular vectors is used e.g. in [10], where studies of the left singular vectors, $\mathbf{U}$, and Fourier transformations of those are carried out. See for instance [10, Fig. 4.2]. Those studies lead to the conclusion that the SVD bases have frequency decomposing properties similar to the Fourier basis. The first SVD vectors contain the low frequencies, and the last SVD vectors are the most high-frequent. Here a study of the two-dimensional vectors is carried out.

For two-dimensional problems, the singular vectors of $\mathbf{A}$ represent two-dimensional. Plotting one singular vector therefore results in a 3D-plot – or an image if seen from the top with the value of each point denoting the intensity. Trying again to look at all the singular vectors at the same time, a series of three-dimensional structures is obtained. I.e. a four-dimensional plot would be needed for visualization.

A small example problem is used to illustrate and analyze the problems with the visualization as well as studying the singular vectors. No particular image is used, as only the blurring matrix $\mathbf{A}$ is studied. The blurring matrix $\mathbf{A}$ is a Kronecker product of two identical Toeplitz matrices, describing symmetric, spatially invariant blur, as seen in (3.24). The matrix is of size 64×64.

In the 2D-DCT domain, the lowest frequencies lie in one corner of the domain, and the highest in the opposite. The location of the frequency component in the image denotes the direction of the spatial variation corresponding to this specific frequency component. Images of the first four singular vectors, transformed to the DCT domain, are seen in Fig. 3.3. Mathematically speaking, we show the show is:

$$\begin{aligned} \widehat{\mathbf{V}}_{\text{unord}} &= \text{vec}^{-1}\left((\mathbf{G}^T\mathbf{V}_1) \otimes (\mathbf{G}^T\mathbf{V}_2)\right) \\ &= \text{vec}^{-1}\left((\mathbf{G} \otimes \mathbf{G})^T(\mathbf{V}_1 \otimes \mathbf{V}_2)\right) \end{aligned}$$

Introducing the permutation matrix $\mathbf{\Pi}$ such that $\mathbf{\Pi}(\mathbf{\Sigma}\otimes\mathbf{\Sigma})\mathbf{\Pi}^T$ contains the ordered singular values, we get:

$$\widehat{\mathbf{V}}_i = \text{vec}^{-1}\left((\mathbf{G} \otimes \mathbf{G})^T\mathbf{\Pi}(\mathbf{V}_1 \otimes \mathbf{V}_2)\mathbf{\Pi}^T\right)_{[:,i]} \quad i = 1, 2, 3, 4 \tag{3.31}$$

From these plots it is clearly seen that the first vector, plotted in 2D, has the maximum frequency response in the upper left corner corresponding to the lowest frequency. The two next vectors have their maximum contribution for the next frequency represented, the first horizontal and the next vertical. The fourth vector has its main contribution on the diagonal at an even higher frequency.

To obtain an overview of all the singular vectors, 64 plots of this type is needed, and for a real image restoration problem, the number of plots would be huge. So plotting all vectors at the same time is necessary to obtain an overview. A plot of the full matrix $\left[(\mathbf{G}^T\mathbf{V}_1) \otimes (\mathbf{G}^T\mathbf{V}_2)\right]$, corresponding to the plot mentioned earlier in [10, Fig. 4.2], is seen in Fig. 3.4. As all the singular vectors are two-dimensional as seen in Fig. 3.3, but here plottet as one-dimensional vectors, no obvious increase in frequency is observed.

Turning again to look at the individual singular vectors, we know that the 2D frequencies increase radially from lowest in one corner to highest in the opposite.

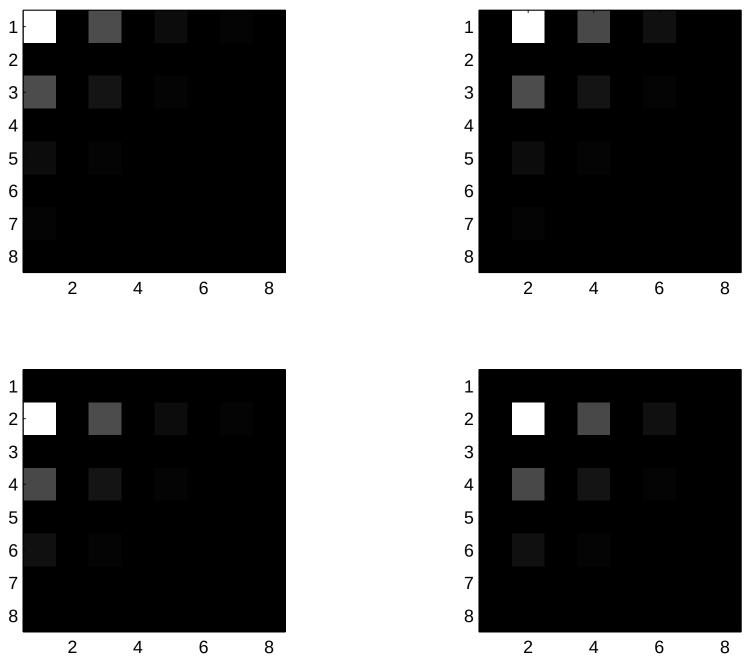


Figure 3.3: *2D-Plot of first four DCT-transformed right singular vectors. First singular vector is at top right, the next top left, the third at bottom right and the fourth at bottom left.*

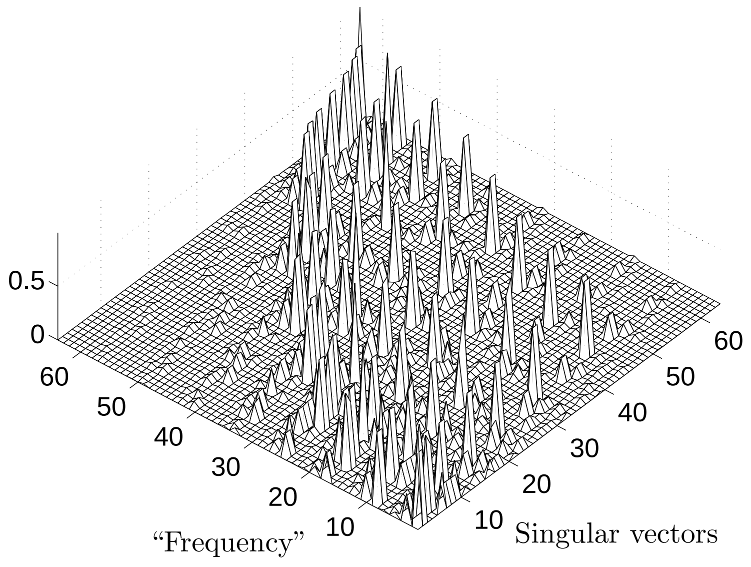


Figure 3.4: *Plot of Kronecker Product as it is.*

Creating a matrix $\mathbf{R}$ containing the Euclidean distance to the upper left corner:

$$\mathbf{R}_{[i,j]} = \sqrt{i^2 + j^2}, \quad \text{where } i, j = 1, 2, \dots, N \tag{3.32}$$

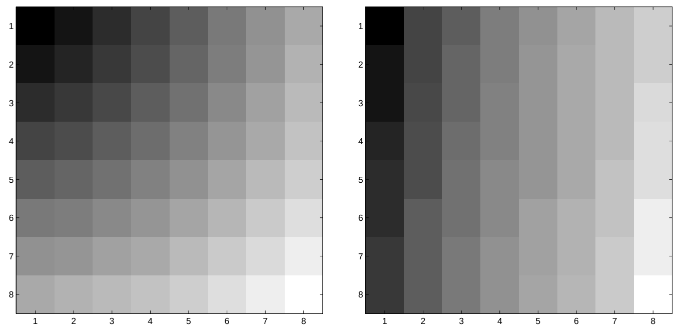
will look as in Fig. 3.5 (a). This gray level plot shows how the frequencies increase radially. Stacking this matrix column-wise and sorting the elements, it is possible to make a permutation matrix $\mathbf{P}$ of size $N^2 \times N^2$ that rearranges the frequencies so the elements of the vector $\mathbf{P}\text{vec}\,(\mathbf{R})$ will lie in non-increasing order. For the elements of $\mathbf{R}$ that are identical, the ordering is not well-defined. This occurs for elements (i_1, j_1) and (i_2, j_2) where

$$\sqrt{i_1^2 + j_1^2} = \sqrt{i_2^2 + j_2^2}, \tag{3.33}$$

The ordering of those elements can be done randomly, column-wise or row-wise. Using permutation matrix like this on the Euclidean distance map and transforming back to an image:

$$\mathbf{R}_{\text{ord}} = \text{vec}^{-1}\left(\mathbf{P}\text{vec}\,(\mathbf{R})\right), \tag{3.34}$$

results in the image showed in Fig. 3.5 (b). As seen, the frequencies are now ordered column-wise.



(a) (b)

Figure 3.5: *(a) Image of matrix* $\mathbf{R}$*, i.e. an euclidean map of the distances to the upper left corner. (b)* $\mathbf{R}_{\text{ord}}$ *as defined in* (3.34)*, i.e. euclidean map ordered by the generated permutation matrix.*

Now using the permutation matrix, $\mathbf{P}$, on the Kronecker product:

$$\widehat{\mathbf{V}}_{\text{ord}} = \mathbf{P}\left((\mathbf{G} \otimes \mathbf{G})^T \mathbf{\Pi}(\mathbf{V}_1 \otimes \mathbf{V}_2)\mathbf{\Pi}^T\right), \tag{3.35}$$

the effect of rearranging the frequencies of every singular vector, studied in one dimension, is seen in Fig. 3.6. This figure should be compared to Fig. 3.4. It is clearly seen that rearranging the frequencies results in a much more diagonal structure, indicating that the SVD basis also in the two-dimensional case has a frequency decomposing property. Apart from the small contributions off the diagonal, the diagonal structure itself is disturbed a few places. This is due to the identical frequencies, which results in ambigious ordering in the permutation matrix $\mathbf{P}$. Here this ordering is chosen so most elements fall on the diagonal.

Non-symmetric Blurring

In the case $\mathbf{T}_{\sigma_c} \neq \mathbf{T}_{\sigma_r}$ in (3.24), the situation is different. Now the ordering of the singular vectors does not follow the radial form described by $\mathbf{R}$ in (3.32) as the contribution is "stronger" in one direction than in the other. To illustrate the difference, two blurring matrices are constructed. One, $\mathbf{A}_s$, consisting of identical blurring in the columns and the rows, and the other, $\mathbf{A}_{ns}$, consisting of different blurring. Again it is important to distinguish between non-symmetric blurring and the matrix

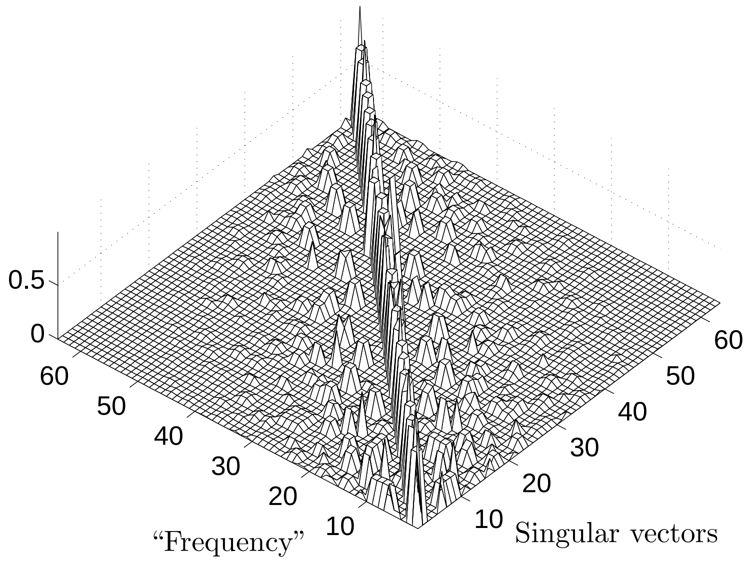


Figure 3.6: *3D plot of Kronecker product with frequencies permuted so the lowest frequencies are in the top and the highest in the bottom.*

A being non-symmetric. As both $\mathbf{A}_s$ and $\mathbf{A}_{ns}$ are described by a Kronecker product of two symmetric Toeplitz matrices, both matrices are mathematically symmetric. The blurring matrices $\mathbf{A}_s$ and $\mathbf{A}_{ns}$ are chosen to be of size 30625×30625. Looking at the sum of the 2D-DCT of the first 3000 singular vectors:

$$\widehat{\mathbf{V}}_\Sigma = \sum_{i=1}^{3000} \text{vec}^{-1}\left((\mathbf{G} \otimes \mathbf{G})^T \mathbf{\Pi} (\mathbf{V}_1 \otimes \mathbf{V}_2) \mathbf{\Pi}^T\right)_{[:,i]}, \tag{3.36}$$

the images in Fig. 3.7 are obtained. Here it is clearly seen which frequencies are picked out in the two cases, and that in the case of symmetric blur, all picked out frequencies lie within a circle in the upper left corner of the image. In the non-symmetric case, it is seen that higher frequencies in one direction are picked out before lower frequencies in the other direction, as described.

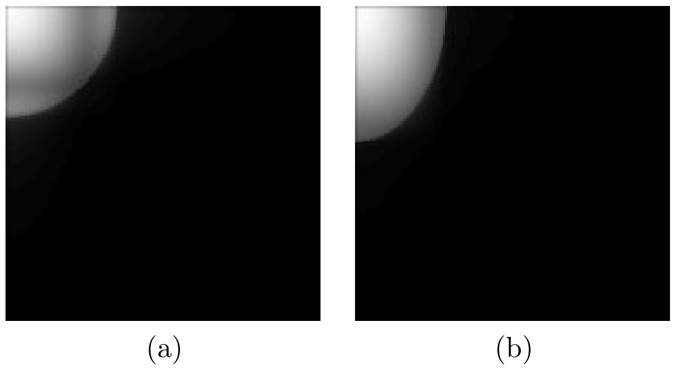


Figure 3.7: *Illustration of frequency content in the first 3000 right singular vectors in (a) the symmetric case, and (b) the non-symmetric case.*

3.5 Singular Values and Frequencies

It is often observed in image restoration that some singular values occur more than once. In addition it is observed that the singular values often decay much slower for this kind of problems than for other inverse problems. For example, the singular values of the small, symmetric blurring matrix from last section are seen in Fig. 3.8. To verify that the singular values that look equal also are equal, the function:

$$f_\sigma(i) = \begin{cases} 0 & \text{for } \sigma_i - \sigma_{i-1} \leq 10\epsilon_M \\ 1 & \text{for } \sigma_i - \sigma_{i-1} > 10\epsilon_M \end{cases} \qquad i = 2, 3, \ldots, N, \tag{3.37}$$

where ϵ_M denotes the machine accuracy, is plotted in Fig. 3.9 as circles.

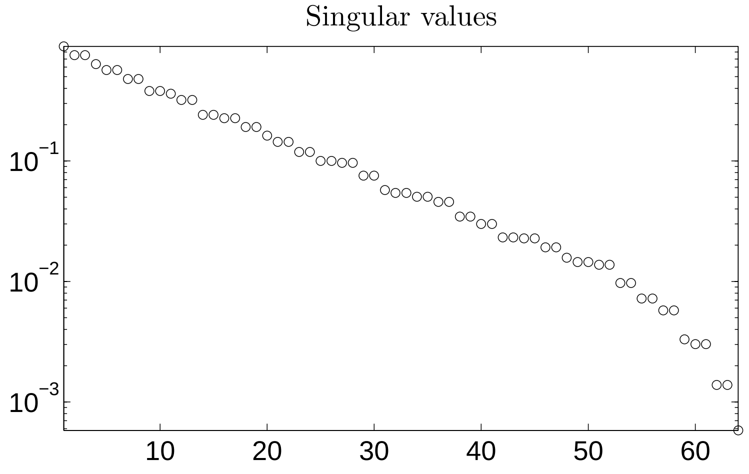


Figure 3.8: *Singular values for the test problem.*

As discussed earlier, the 2D-restoration gives rise to frequencies in many directions. It is also seen from Fig. 3.5 and in (3.33) that the same frequencies sometimes appear in more than one direction. When the blurring is equal in the two directions, the frequency component in the two directions must also be the same. Therefore having the same singular value for the same frequency in different directions seems plausible. To check whether this is true, a plot of the function:

$$f_{\text{freqs}}(i) = \begin{cases} 0 & \text{for } (\mathbf{P}\text{vec}\,(\mathbf{R}))_i - (\mathbf{P}\text{vec}\,(\mathbf{R}))_{i-1} \leq 10\epsilon_M \\ 1 & \text{for } (\mathbf{P}\text{vec}\,(\mathbf{R}))_i - (\mathbf{P}\text{vec}\,(\mathbf{R}))_{i-1} > 10\epsilon_M \end{cases} \qquad i = 2, 3, \ldots, N, \tag{3.38}$$

where again ϵ_M denotes the machine accuracy, is shown in Fig. 3.9 with '$\times$' on top of the plot of the differences of the singular values. Comparing the two plots in this figure now shows that there is a great correspondence between singular values being identical and reconstructed frequencies in different directions being identical.

That the singular values decay slower for this kind of problems than for other inverse problems is also clearly caused by the many identical frequencies in different directions. As seen in e.g. Fig. 3.6 each singular vector mainly picks out one frequency component in one direction. So to reach the higher frequencies, more singular vectors are needed.

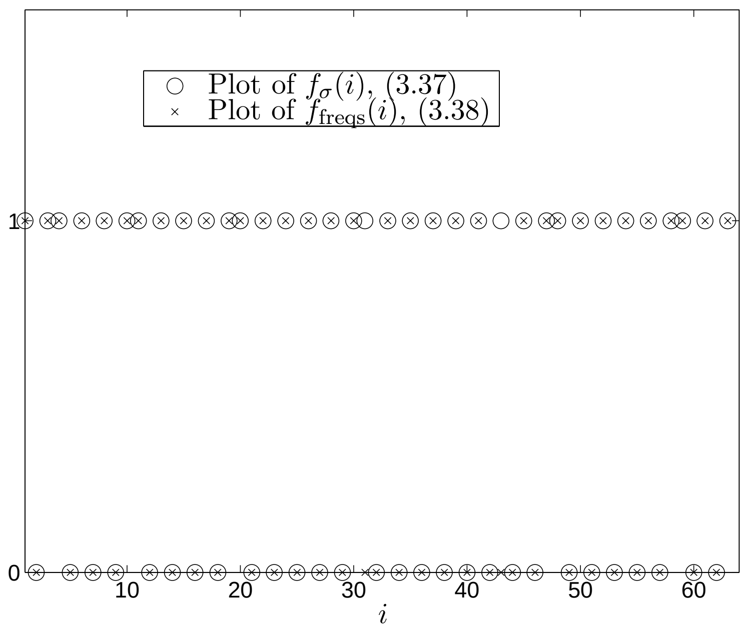


Figure 3.9: *The '○' denotes the plot of* (3.37) *and the '×' denotes the plot of* (3.38)

In the case where the blurring is non-symmetric, the frequencies reconstructed are not the same in both directions. This results in much fewer multiple singular values - if at all any exist. The decay of the singular values is still somewhat slow, while frequencies in many directions are still reconstructed. A plot of the singular values for a small test blurring with different blurring of the columns and the rows, is shown in Fig. 3.10. None of the singular values are exactly identical, but the slow decay is still observed while many singular values are still almost identical.

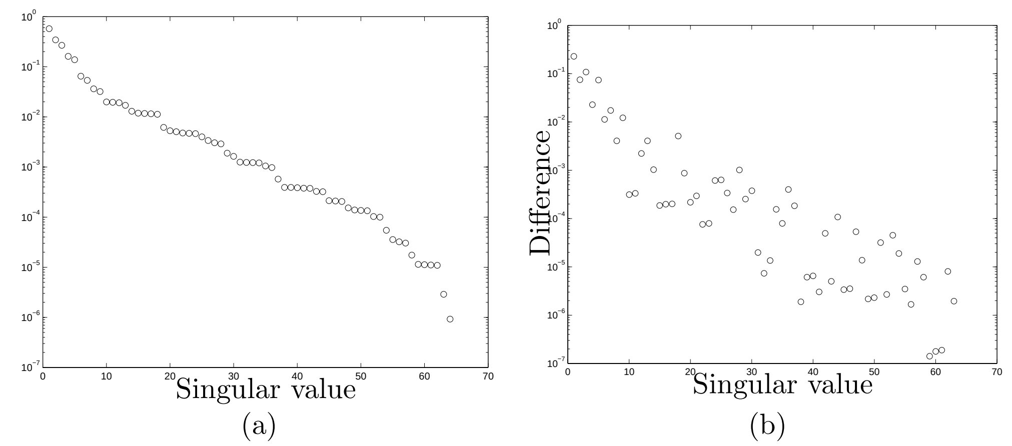


Figure 3.10: *(a) Singular values for an asymmetric blurring operator. (b) Differences between the singular values - none is zero.*

3.6 Numerical Examples

In the 1D case, the results of the regularization is well studied in the literature. It is well known that the singular vectors have an increasing number of zero crossings – that they increase in frequency. This means that the restorations with a low number k of included singular vectors in the TSVD approach or similarly a high value of λ for the Tikhonov solution, mainly consist of low frequency components. At a certain point, when k increases or λ decreases, the noise components get bigger than the coefficients to the true solution, and the SVD components corresponding to these frequencies blow up.

In the 2D case the singular vectors also correspond more or less to specific frequencies as described above. But in this case the same frequencies exist in many directions, and these directions are picked up one at a time. This means that even though a noise component makes one frequency component in one direction blow up, the same or similar frequencies are not yet reconstructed in other directions. The overall contribution from that specific noise component is therefore smaller and the solution does not blow up as fast as in the 1D case. It blows up a little in one direction but is still more "smooth" in other directions. If the blurring is identical in all directions, then the noise is seen in the solution as small circles, while the noise components get larger than the coefficients to the true solution at a certain frequency. This corresponds to the 1D case where the solution starts to oscillate with the singular vectors and thereby the frequencies affected by the noise components. If the blurring is elliptical, then the noise will enter the solution at different frequencies in the different directions resulting in "ovals" in stead of circles. The right singular vectors also "pick out" the frequencies in a different order - so higher frequencies in some directions are restored before lower frequencies in other directions, as showed in Fig. 3.7.

Now a small test image is introduced. The image is blurred with spatially invariant, separable blur with $\sigma_c = \sigma_r = 3$, described by a Kronecker product as in (3.24). Furthermore, to the blurred image has been added white noise with the noise level $\eta = 0.05$. It is now possible to do direct calculations using the TSVD and Tikhonov filter factors. The true image is seen in Appendix B, Fig. B.2.

3.6.1 TSVD and Tikhonov Solutions

First, we show the problem to solve in Fig. 3.11. We want to de-blur the image, and to get an idea of the properties we look at the Picard plot for the problem in Fig. 3.12 (a). This illustrates some of the difficulties in calculating a regularized solution, showing the slow decay of the singular values, as well as the size of the problem. Also it is seen that only a very small percentage of the singular components are wanted. To get an idea of what truncation parameter k to choose for the TSVD solution, a closer look at the first part of the plot is shown in Fig. 3.12 (b). Here it is seen that the noise starts to dominate the solution coefficients from around the 1000'th singular value:

$$|\mathbf{u}_i^T \mathbf{b}| \approx \begin{cases} |\mathbf{u}_i^T \bar{\mathbf{b}}| & \text{for} \quad i < 1000 \\ |\mathbf{u}_i^T \mathbf{e}| & \text{for} \quad i \geq 1000 \end{cases}, \tag{3.39}$$

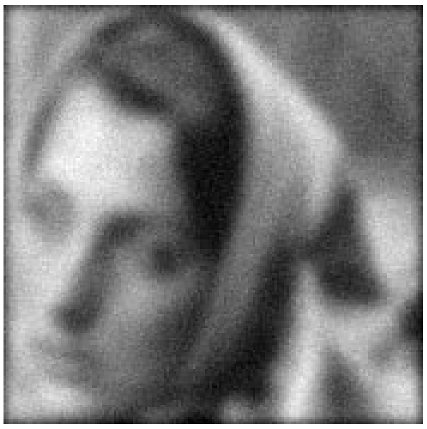


Figure 3.11: *Blurred and noisy realization of the Barbara image from Fig B.2 from Appendix B. The blurring used is spatially invariant with $\sigma_c = \sigma_r = 3$, and the noise level is set to $\eta = 0.05$.*

suggesting that the truncation parameter should be chosen to be $k = 1000$. Thus even though only a small part of the components are wanted in the solution, the actual number, k, of components is quite large, showing again the size of the problem.

To illustrate how different choices of the truncation parameter affect the solutions, and how the noise is included, a number of different truncation parameters are used in Fig. 3.13. Here it is used that the noise vector $\mathbf{e}$ is explicitly known, and we are able to write the TSVD solution from (2.17) as:

$$x_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T(\bar{\mathbf{b}}+\mathbf{e})}{\sigma_i}\mathbf{v}_i = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\bar{\mathbf{b}}}{\sigma_i}\mathbf{v}_i + \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\mathbf{e}}{\sigma_i}\mathbf{v}_i. \tag{3.40}$$

This allows for looking explicitly at the contribution to the solution from the true, blurred image, and from the noise. The three truncation parameters $k =$

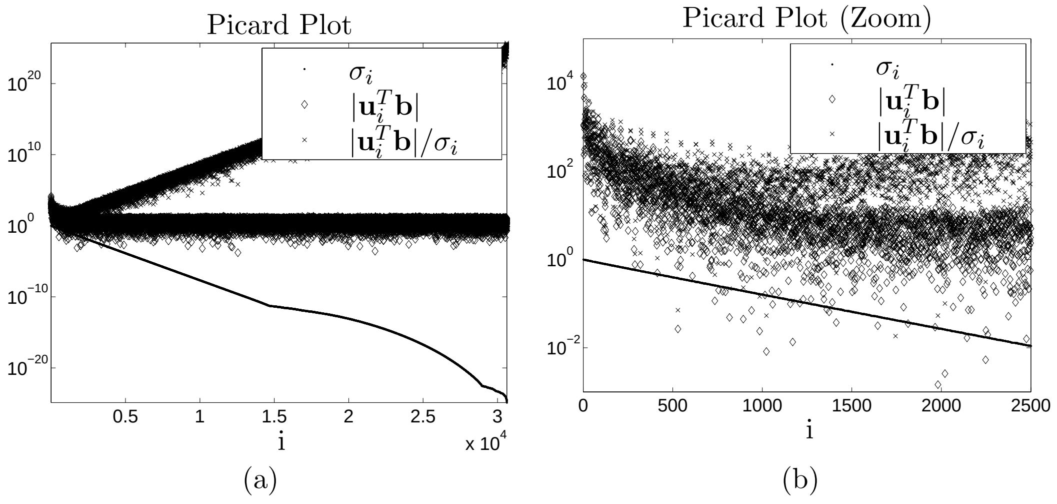


Figure 3.12: *Illustration of Picard plot for an image problem. In (a), the full plot is shown, and in (b), the interesting part is zoomed up.*

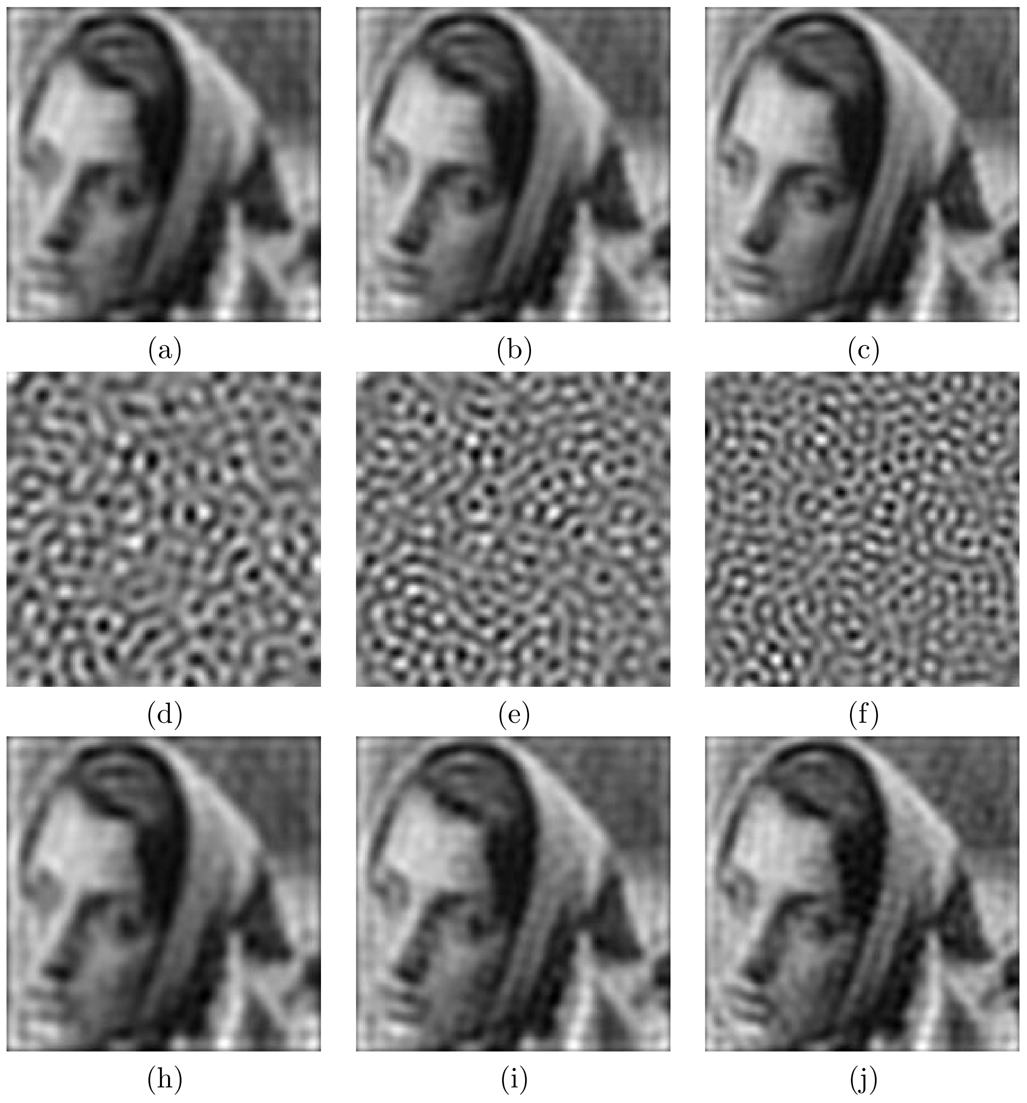


Figure 3.13: *Images (a), (b), and (c) show the TSVD solutions to the true blurred right-hand side* $\widetilde{\mathbf{b}}$ *for 800, 1000, and 1200 included SVD components; (d), (e), and (f) show the TSVD solutions to the noise vector* $\mathbf{e}$*; and (h), (i), and (j) show the TSVD solutions to the blurred and noisy right-hand side* $\mathbf{b}$.

800, $k = 1000$ and $k = 1200$ are chosen, and as seen from Fig. 3.13, the noise is getting visible in the solutions for 1000 included components and is clearly starting to dominate the solution when including 1200 singular vectors, whereas including only 800, the solutions to $\bar{\mathbf{b}}$ and $\mathbf{b}$ are almost identical. Of course, including more SVD components give more details in the solution, but it also makes the noise component stronger. The visually best of the three solutions in the figure is seen to be the one including 1000 SVD components, being a compromise between detail level and damping of the noise – which also corresponds to with the Picard plot in Fig. 3.12.

Choosing a similar set of regularization parameters λ for the Tikhonov solution, the same results as for the TSVD solution are seen in Fig. 3.15. The regularization

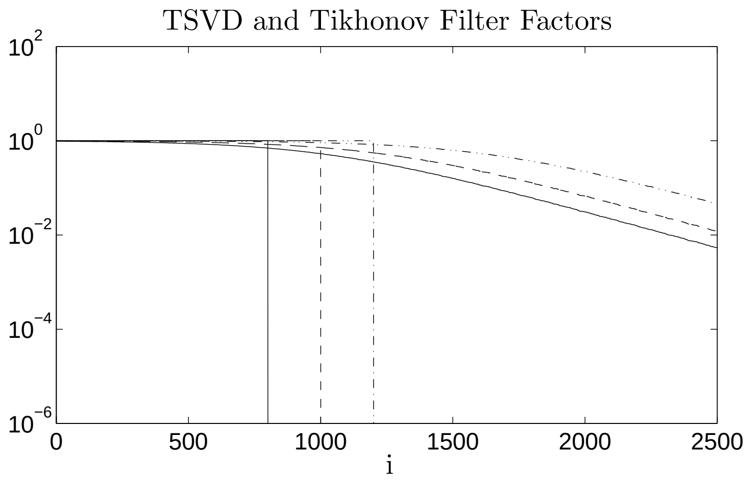


Figure 3.14: *TSVD and Tikhonov filter factors used to obtain the results in Figs. 3.13 and 3.15.*

parameters chosen are $\lambda = 0.15$, $\lambda = 0.10$ and $\lambda = 0.05$, and a comparison of those Tikhonov filters, and the TSVD filters from above, is seen in Fig. 3.14. As observed, the Tikhonov filter factors fall off very slowly caused by the equally slow decay of the singular values σ_i.

It is observed that even for the largest regularization parameter $\lambda = 0.15$, the noise contribution to the full solution is notable, making small "freckles" appear in the regularized solution in Fig. 3.15 (h). This is again especially seen comparing the solutions of the noise-free right-hand side $\sum \psi_i(|\mathbf{u}_i\bar{\mathbf{b}}|/\sigma_i)\mathbf{v}_i$ to the solutions to the noisy right-hand side $\sum \psi_i(|\mathbf{u}_i\mathbf{b}|/\sigma_i)\mathbf{v}_i$. On the other hand, as seen from the images (a), (b), and (c) in the two figures, the underlying true solutions are all more detailed in case of Tikhonov regularization. This is somehow expected as many more components are partly included in the solution through the slowly decaying filter factors.

A more stringent comparison of the solutions is found in Table 3.1. Knowing the true image, it is possible to calculate the two-norm of the difference between the regularized solution and the true solution $\|\mathbf{x}_{\text{reg}} - \mathbf{x}\|_2$. This comparison clearly shows that the visual effects of the noise entering the solutions does not necessarily have a great impact on the two-norm of the difference to the true solution. In fact, the best solution is here seen to be the second Tikhonov solution. This is discussed further in the following.

	$k = 800$, $\lambda = 0.15$	$k = 1000$ $\lambda = 0.10$	$k = 1200$, $\lambda = 0.05$
TSVD	2.8161e+03	2.6087e+03	2.5443e+03
Tikhonov	2.5554e+03	2.4829e+03	2.8437e+03

Table 3.1: *Comparison of TSVD and Tikhonov solutions in terms of the 2-norm difference between the regularized solutions and the true solution.*

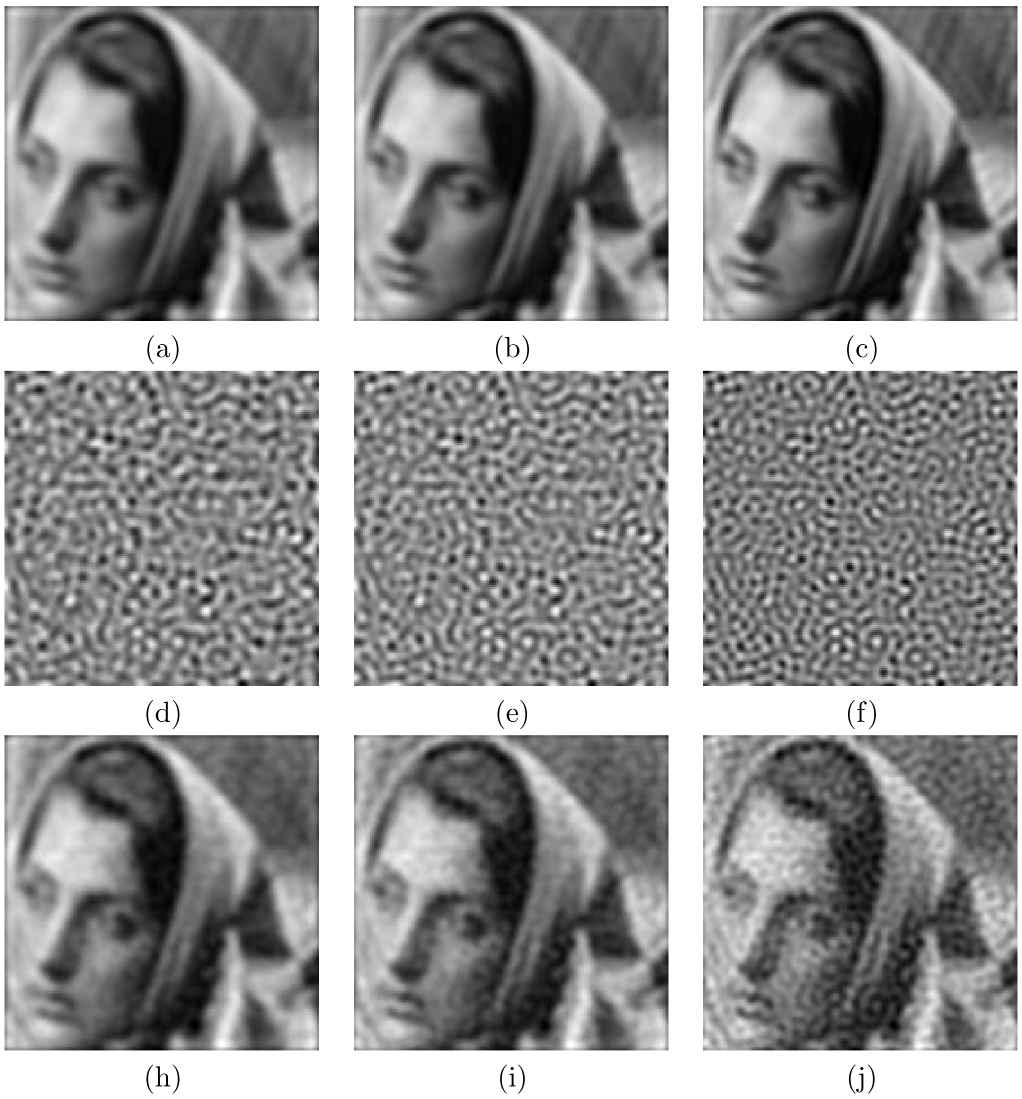


Figure 3.15: *Images (a), (b), and (c) show the Tikhonov solutions to the true blurred right-hand side* $\widetilde{\mathbf{b}}$ *for* $\lambda = 0.15$, $\lambda = 0.10$, *and* $\lambda = 0.05$; *(d), (e), and (f) show the same Tikhonov solutions to the noise vector* $\mathbf{e}$; *and (h), (i), and (j) show the Tikhonov solutions to the blurred and noisy right-hand side* $\mathbf{b}$.

3.6.2 Row-wise analysis

To illustrate the problems of the regularized solutions in another way, one single line of the image is plotted with the intensity as a function of the column. In Fig. 3.16, row number 50 of 175 rows of the second Tikhonov regularized solution is shown. As seen, we e.g. have high intensity for Barbara's forehead and low intensity for her hair.

As seen, the true image contains much more high frequent information than the regularized solution. Also the "freckles" seen in the regularized image are obvious to the human observer, but are hardly noticeable when looking at the single line and comparing this with the true image. Actually the 1D reconstruction of the line is

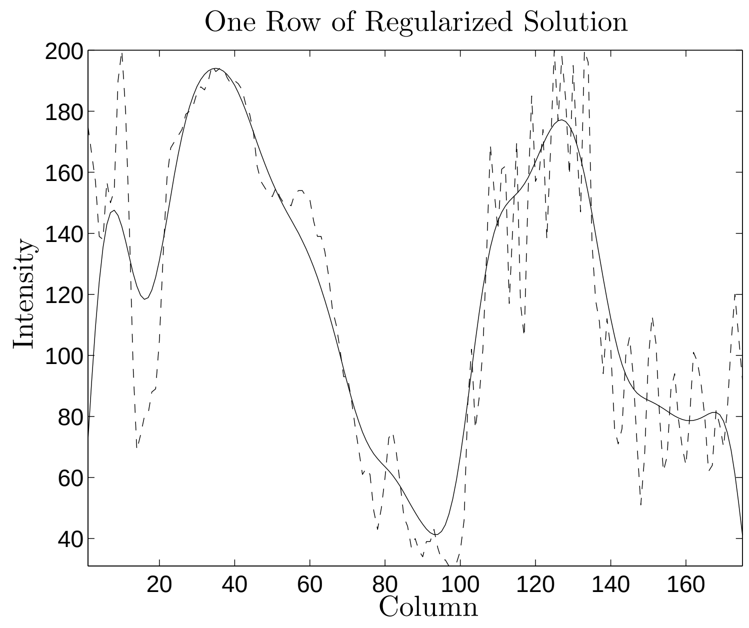


Figure 3.16: *One row picked out of the second Tikhonov regularized solution in Fig. 3.15 (h). The dashed line is the true image and the solid line is the regularized solution.*

good and low-frequent. Looking again at the norms, the two-norm of the true image is $2.0454 \cdot 10^4$. Comparing this with Table 3.1, it is seen that this is much higher than for the regularized solutions, due to many high-frequency components in the image. So even though the noise visually disturbs the solutions in two dimensions, this is not the case for neither the norms nor looking at the solutions in one dimension.

3.7 A Good Solution

An important part of solving inverse problems is to choose the best possible restoration – choosing a regularization parameter that in some sense gives the best regularized solution. But how to evaluate whether a solution is good or bad. In the previous section a visual judgment of the regularized solutions was used, as was the two-norm of the difference between the true image and the solution in Table 3.1. In general we can look at two different kinds of errors.

The forward error is some measure of how close the regularized solution is to the true solution. One example is the two-norm already used, but other norms and semi-norms can also be used. And in connection with images with a human observer as the final target, we might want to include information about the human perception into the measure of the forward error.

The backward error is a measure telling how much information we have extracted from the system, or how well the found regularized solution fits into the linear system of equations. In effect some measure like the residual $\|\mathbf{A}\mathbf{x}_{\mathrm{reg}} - \mathbf{b}\|_2$. Of course using the perturbed right-hand side $\mathbf{b}$ will give a possibility for $\mathbf{x}_{\mathrm{reg}}$ to be fitted to the noise and give a small residual. So what we really want to keep small is the residual

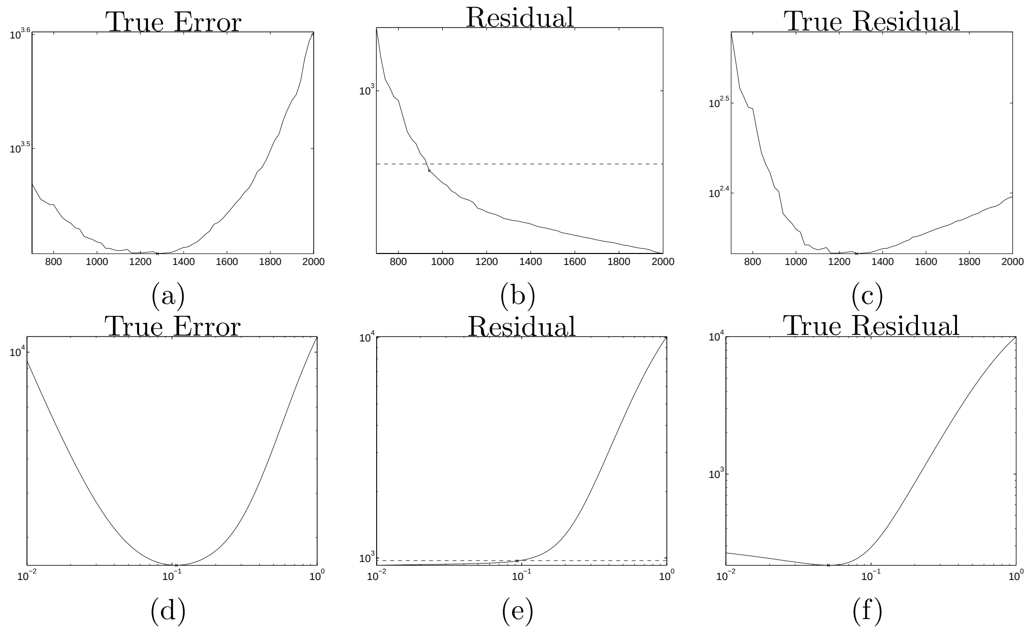


Figure 3.17: *(a) Shows the true error $\|\mathbf{x}-\mathbf{x}_{\text{reg}}\|_2$, (b) shows the residual $\|\mathbf{A}\mathbf{x}_{\text{reg}}-\mathbf{b}\|_2$ and with dashed line the size of the noise δ_e, and (c) shows the true residual $\|\mathbf{A}\mathbf{x}_{\text{reg}}-\bar{\mathbf{b}}\|_2$. All for the TSVD solutions, including a varying number of singular components. (d), (e), and (f) show the same information for the Tikhonov solutions with varying λ.*

$\|\mathbf{A}\mathbf{x}_{\text{reg}}-\bar{\mathbf{b}}\|_2$. If we know the noise level and by using the first expression, we can also choose as criterion for a good solution that the norm should be reduced until the residual is of same size as the noise level. This leads to the parameter choice method known as *the discrepancy principle*:

$$\|\mathbf{A}\mathbf{x}_{\text{reg}}-\mathbf{b}\|_2 = \delta_e\,, \quad \text{where} \quad \|\mathbf{e}\|_2 \leq \delta_e. \tag{3.41}$$

To see how these measures work in connection with regularization of images, we show the plots of $\|\mathbf{x}-\mathbf{x}_{\text{reg}}\|_2$, $\|\mathbf{A}\mathbf{x}_{\text{reg}}-\mathbf{b}\|_2$, and $\|\mathbf{A}\mathbf{x}_{\text{reg}}-\bar{\mathbf{b}}\|_2$, respectively in Fig. 3.17 for the Barbara example for both TSVD and Tikhonov regularization. And in connection with these plots, the parameters leading to the best solutions for the different measures are seen in Table 3.2.

Comparing the correspondence between these best parameters, and the visual inspection of the solutions from above, we see that in most cases, the different criteria choose regularization parameters that give rise to "freckles". For instance, we can take a closer look at the Tikhonov solutions. The visually "optimal" solution from

	$\min \|\mathbf{x}-\mathbf{x}_{\text{reg}}\|_2$	$\|\mathbf{A}\mathbf{x}_{\text{reg}}-\mathbf{b}\|_2 < \eta$	$\min \|\mathbf{A}\mathbf{x}_{\text{reg}}-\bar{\mathbf{b}}\|_2$
TSVD, k	1280	940	1280
Tikhonov, λ	0.107	0.0933	0.0509

Table 3.2: *For TSVD the "optimal" truncation parameter k in the three cases, and for Tikhonov the "optimal" λ for the corresponding criteria.*

above yields a regularization parameter around $\lambda = 0.15$. The lowest forward error is reached with a regularization parameter approximately $\lambda = 0.11$, but to extract all information and get the best backward error, $\lambda = 0.05$ should be chosen. And as seen from Fig. 3.15 (j), this last option is not desirable. This means that the most desirable solution from a visual perspective, might not always be the solution that extracts the most information from the system, as the visually disturbing "freckles" start to appear.

3.7.1 Human Perception

As a final remark in this section, we stress that more problems arise when the information comes in shape of an image, and the final observer is a human being. The *Human Visual System* (HVS) is very complex and consists of physical as well as psychological parts. Investigations, especially within the area of Image Compression, have shown that the human judgment of visual quality does not correlate well with the two-norm or Mean Squared Error (MSE) of a degraded image compared to the original. The psychological parts, such as deriving information based on intuition and knowledge about the observed scene are difficult, if not impossible, to formalize. But much effort has been done to describe the physical part of the HVS, and one of the resulting characteristics is mentioned and used in a paper by N. B. Nill [17]. Here the spatial frequency sensitivity of the human eye is estimated by a function $H(r)$, described in the 2D-DCT frequency domain, with $r = \sqrt{u^2 + v^2}$ being the euclidian distance to the low-frequency corner of the domain. Basically the empirical function $H(r)$ is given by:

$$H(r) = \begin{cases} 0.05e^{r^{0.554}} & \text{for } r < 7 \\ e^{-9(|log_{10}r - log_{10}9|)^{2.3}} & \text{for } r \geq 7 \end{cases} , \tag{3.42}$$

where r is given as *cycles pr. degree.* The concept of cycles pr. degree means that the observed frequency changes with the distance to the observed object. When the distance to an observed object is doubled, the angle is halved, and so the observed frequencies have changed. For instance, the raster images from the news papers are so high-frequent that the eyes, from a normal viewing distance, do not see the artifacts, but only the more low-frequent image. Moving closer to the image, the high-frequent dots are clearly seen.

The eye's frequency sensitivity is not isotropic either, i.e., the same sensitivity vertically and horizontally, but the assumption is usually made. This results in the two-dimensional weight function shown in Fig. 3.18. The plot illustrates how the eyes sensitivity is highest at a certain frequency, given as cycles pr. degree. When comparing two images, one could apply this weight function to base the concept of norm on the frequencies best perceived by the human observer.

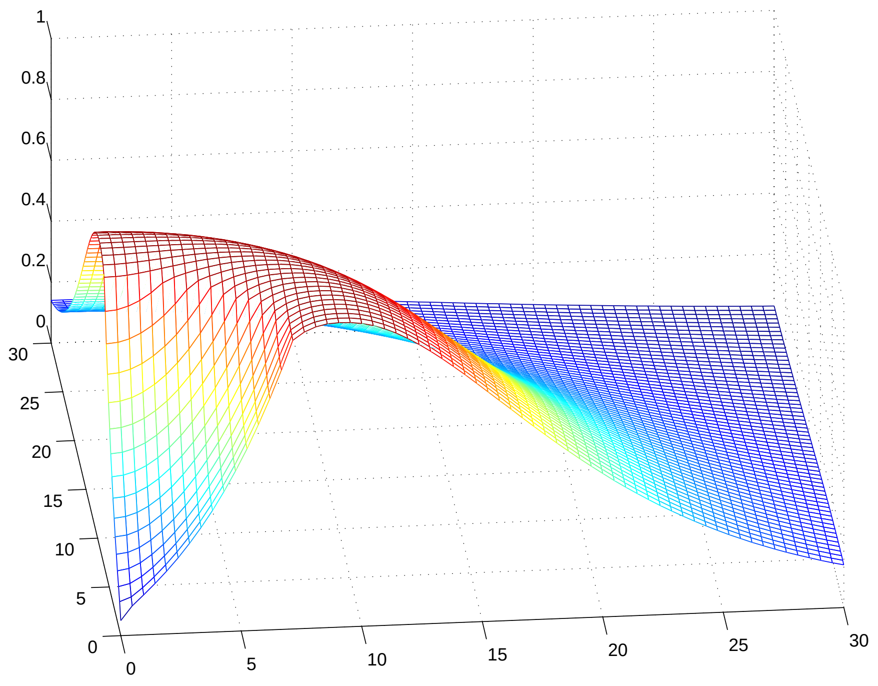


Figure 3.18: *Frequency response of the Human Visual System (HSV)*

CHAPTER 4

Regularizing Iterations

Solving large scale ill-posed problems by direct methods is in general difficult, if not impossible. For instance calculating some orthogonal decomposition of an enormous matrix **A**, i.e. the SVD used earlier, would not be a nice job. Instead the focus is now directed towards iterative algorithms. Second best, after having a directly calculated decomposition or factorization of **A** for analyzing the problem, must be to have an estimate of a decomposition. Or at least some decomposition with some spectral properties, so a regularized solution can be constructed.

The class of iterative methods discussed here, are the so-called *Krylov subspace methods*. Among them, the oldest and most popular is the *Conjugate Gradient (CG)* or, with the same mathematical properties, the *LSQR*. Another Krylov subspace method discussed in this chapter is the *GMRES-algorithm*.

4.1 Krylov Subspaces

The iterative algorithms discussed here, do all belong to the class of Krylov subspace methods, so we start by defining the Krylov subspace:

Definition 4.1 *The Krylov Subspace:*

$$\mathcal{K}_k(\mathbf{A}, \mathbf{b}) \equiv \operatorname{span}\left\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\right\}.$$

All Krylov subspace methods work by finding a regularized solution belonging to the Krylov subspace. To justify that this subspace is a good choice for finding a regularized solution, we look at the case where **A** is non-singular. Investigation of the properties of the Krylov subspace is easiest done by looking at the *minimal polynomial*, defined as the unique monic polynomial, q, of minimal degree fulfilling $q(\mathbf{A}) = 0$. Using the Jordan decomposition, the minimal polynomial can be written as:

$$q(t) = \prod_{j=1}^{d} (t - \lambda_j)^{m_j}, \tag{4.2}$$

where d is the number of distinct eigenvalues of **A**, λ_j are the distinct eigenvalues, and m_j are their corresponding Jordan index. Now m is defined as the sum of the

Jordan indices $m = \sum_j m_j$. In the case where $\mathbf{A}$ is diagonizable, there are d Jordan indices, all equal to one, and then m is just the number of distinct eigenvalues of $\mathbf{A}$. Now the above expression can be formulated as:

$$q(t) = \sum_{j=0}^{m} \alpha_j t^j, \quad \alpha_0 = \prod_{j=1}^{d} (-\lambda_j)^{m_j} \tag{4.3}$$

This shows that $\alpha_0 \neq 0$ under the assumption that $\mathbf{A}$ is non-singular. Through a few calculations, the inverse of $\mathbf{A}$ can be expressed by means of the above minimal polynomial, in terms of $\mathbf{A}$, in the following way:

$$\begin{aligned} q(\mathbf{A}) &= \alpha_0 \mathbf{I} + \alpha_1 \mathbf{A} + \cdots + \alpha_m \mathbf{A}^m = 0 & (4.4) \\ -\alpha_0 \mathbf{A}^{-1}\mathbf{I} &= \mathbf{A}^{-1} \sum_{j=1}^{m} \alpha_j \mathbf{A}^j & (4.5) \\ \mathbf{A}^{-1} &= -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} \mathbf{A}^j & (4.6) \end{aligned}$$

Writing out the solution to the system, it is now clearly seen that this solution must lie in the Krylov subspace $\mathcal{K}_m(\mathbf{A}, b)$:

$$x = \mathbf{A}^{-1} b = -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} \mathbf{A}^j b \tag{4.7}$$

In case that $\mathbf{A}$ is singular, a splitting of the Jordan decomposition into two parts can be done, as described e.g. by J. M. Rasmussen in [18, Section 7.2.3] or by I. C. F. Ipsen and C. D. Meyer in [11]. Further information about existence of solutions are also found here. At this point only a justification of the correspondence between the Krylov subspaces and the solution to the linear system of equations is made.

4.2 LSQR

This method is a method for solving least squares problems and is mathematically equivalent to the *Conjugate Gradient* (CG) algorithm applied to the normal equations $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$. Numerically it has proven to be more stable than the basic CG algorithm, why the LSQR is used here. The basic building block for this algorithm is the *Lanczos bidiagonalization*, introduced in the following.

4.2.1 Lanczos Bidiagonalization

The Lanczos bidiagonalization algorithm is related to the Lanczos tridiagonalization, which is a special case of the Arnoldi process introduced later, for symmetric matrices. After k steps of the bidiagonalization algorithm, the following decomposition is obtained.

$$\mathbf{A}\mathbf{V}_k = \mathbf{U}_{k+1}\mathbf{B}_k, \tag{4.8}$$

where $\mathbf{V}_k \in \mathbb{R}^{n\times k}$ and $\mathbf{U}_{k+1} \in \mathbb{R}^{n\times(k+1)}$ have orthonormal columns, and $\mathbf{B}_k \in \mathbb{R}^{(k+1)\times k}$ is lower bidiagonal of the following form:

$$\mathbf{B}_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix} \tag{4.9}$$

The Algorithm takes the form of Algorithm 4.1 where $\mathbf{V}_k = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_k\end{bmatrix}$, and $\mathbf{U}_{k+1} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_{k+1}\end{bmatrix}$. As seen, multiplication with both $\mathbf{A}$ and $\mathbf{A}^T$ is needed.

Algorithm 4.1 *Lanczos Bidiagonalization*
$[\mathbf{U}_{k+1}, \mathbf{B}_k, \mathbf{V}_k]$ = LANCZOS $(\mathbf{A}, \mathbf{b}, k)$
Given the starting vector $\mathbf{x}$*, a matrix* $\mathbf{A}$*, and a parameter* k*, the decomposition in Eqn.* (4.8) *is obtained. The matrix* $\mathbf{V}_k$ *is an orthonormal basis for the Krylov subspace* $\mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$.

```
$\beta_1 = \|\mathbf{x}\|_2$
$\mathbf{u}_1 = \beta_1^{-1}\mathbf{x}$
$\mathbf{v}_0 = 0$
FOR $i = 1, 2, \dots, k$
    $\mathbf{p}_i = \mathbf{A}^T\mathbf{u}_i - \beta_i\mathbf{v}_{i-1}$
    $\alpha_i = \|\mathbf{p}_i\|_2$
    $\mathbf{v}_i = \alpha_i^{-1}\mathbf{p}_i$
    $\mathbf{q}_i = \mathbf{A}\mathbf{v}_i - \alpha_i\mathbf{u}_i$
    $\beta_{i+1} = \|\mathbf{q}_i\|_2$
    $\mathbf{u}_{i+1} = \beta_{i+1}^{-1}\mathbf{q}_i$
```

A remarkable property is that in exact arithmetic, the basis vectors $\mathbf{v_k}$ and $\mathbf{u}_k$ are by construction orthogonal to all previously generated vectors. In practise, though, this property only holds true for the first few vectors, whereas the following loose the orthogonality for numerical reasons. If orthogonality or simulation of infinite precision is needed, explicit reorthogonalization of the new vectors generated is necessary. This can e.g. be done by means of modified Gram-Schmidt.

4.2.2 LSQR Algorithm

The LSQR algorithm constructs new solution vectors by always ensuring that the new residual vector is $\mathbf{A}$-orthogonal[1] to all the previous residuals. This corresponds to finding the solution with minimum two-norm residual in the Krylov subspace $\mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$, which in case of $\mathbf{A}$ being symmetric and positive definite is similar to the following definition of the solution in the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$:

$$\mathbf{x}^{(k)} = \min_{\mathbf{x}\in\mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_{\mathbf{A}^{-1}}. \tag{4.10}$$

[1] $\mathbf{x}_1$ and $\mathbf{x}_2$ are $\mathbf{A}$-orthogonal if $\mathbf{x}_1^T\mathbf{A}\mathbf{x}_2 = 0$

The norm $\|\mathbf{z}\|_{\mathbf{A}^{-1}} = \sqrt{(\mathbf{A}^{-1}\mathbf{z},\mathbf{z})} = \sqrt{\mathbf{z}^T\mathbf{A}^{-1}\mathbf{z}}$ can only be considered a norm, if $\mathbf{A}$ is symmetric and positive definite.

The Lanczos bidiagonalization from Algorithm 4.1 is run with starting vector $\mathbf{b}$, which results in the bidiagonal matrix $\mathbf{B}_k$. From [8, Section 6.3.2], this matrix is known to have singular values that approximate some of the singular values of $\mathbf{A}$. Those singular values of $\mathbf{B}_k$ are the square roots of the so-called *Ritz values* that are defined as the eigenvalues of $\mathbf{B}^T\mathbf{B}$. Here the singular values of $\mathbf{B}_k$ are named *singular Ritz values*. Especially the larger singular Ritz values will quickly converge to the larger true singular values of $\mathbf{A}$. Computing an SVD of the small matrix $\mathbf{B}_k$:

$$\mathbf{B}_k = \mathbf{P}\boldsymbol{\Gamma}\mathbf{Q}^T, \tag{4.11}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are the right and left singular matrices and $\boldsymbol{\Gamma}$ is a diagonal matrix containing the singular values:

$$\boldsymbol{\Gamma} = \begin{bmatrix} \gamma_1 & & & \\ & \gamma_2 & & \\ & & \ddots & \\ & & & \gamma_k \end{bmatrix}.$$

From the connection with the Ritz values, (4.11) is now seen to reveal information about the SVD of the original matrix $\mathbf{A}$. An approximation to the SVD of $\mathbf{A}$ is now given as:

$$\mathbf{A}\left(\mathbf{V}_k\mathbf{Q}\right) = \left(\mathbf{U}_{k+1}\mathbf{P}\right)\boldsymbol{\Gamma} \tag{4.13}$$

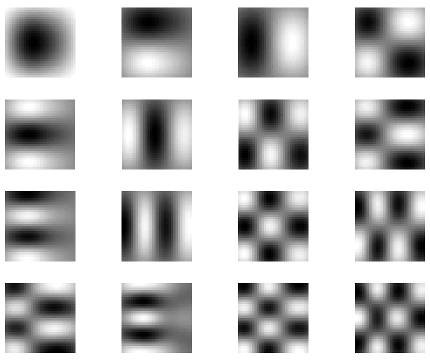
By comparing expression (4.13) with the SVD of $\mathbf{A}$ in (2.6), it is clear that as $\boldsymbol{\Gamma}$ approximates the singular values of $\mathbf{A}$, then $(\mathbf{U}_{k+1}\mathbf{P})$ and $(\mathbf{V}_k\mathbf{Q})$ must approximate the right and left singular matrices of $\mathbf{A}$. Those vectors are here called *singular Ritz vectors* from their relationship with the singular Ritz values.

Now a solution can be written by means of the Lanczos bidiagonalization after k iterations by:

$$\begin{aligned} \mathbf{x}^{(k)} &= \left(\mathbf{V}_k\mathbf{Q}\right)\boldsymbol{\Gamma}^{-1}\left(\mathbf{U}_{k+1}\mathbf{P}\right)^T\mathbf{b} & (4.14)\\ &= \sum_{i=1}^{k} \frac{\left(\mathbf{U}_{k+1}\mathbf{P}\right)_{:,i}^T\mathbf{b}}{\gamma_i}\left(\mathbf{V}_k\mathbf{Q}\right)_{:,i}. & (4.15)\end{aligned}$$

The solution can now theoretically be investigated in the same way as usual, by looking at the vectors constructing the solution $(\mathbf{V}_k\mathbf{Q})_{:,i}$ as well as the coefficients $(\mathbf{U}_{k+1}\mathbf{P})_{:,i}^T\mathbf{b}$ and the singular values γ_i. The singular Ritz vectors $(\mathbf{V}_k\mathbf{Q})_{:,i}$ after 16 iterations of the bidiagonalization algorithm for a small test problem, are shown in Fig. 4.1 (b). The true right singular vectors $\mathbf{V}$ of $\mathbf{A}$ are shown in Fig. 4.1 (a). As seen there is a nice correspondance between true singular vectors and the approximations for the first 11 vectors, corresponding to the 11 largest singular values. The difference in e.g. the first vector is simply caused by a sign change.

In practice, the LSQR solution is not calculated by means of the SVD of $\mathbf{B}_k$, but by means of a QR-factorization of $\mathbf{B}_k$. In this way, the solution $\mathbf{x}_k$ can be updated efficiently from the last iteration $\mathbf{x}_{k-1}$, without storing all earlier iterates. This also


(a)

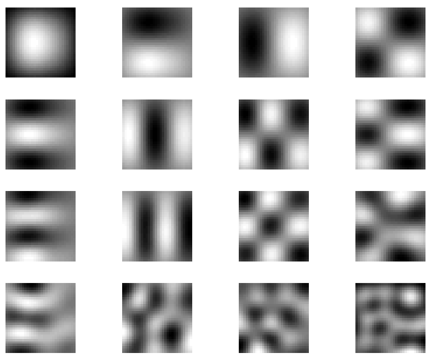

(b)

Figure 4.1: *(a) First 16 2D right singular vectors of* **A** *(from top left to bottom right). (b) 2D Ritz vectors for LSQR solution after 16 iterations (from top left to bottom right).*

means that when no reorthogonalization is used in the Lanczos Bidiagonalization, the vectors in $\mathbf{V}_k$ and $\mathbf{U}_{k+1}$ do not need to be stored either. The reason for this nice update property, is the specific choice of the starting vector $\mathbf{b}$ in the Lanczos bidiagonalization algorithm.

Implementation

Different algorithms have been used. For studying the matrix $\mathbf{B}_k$ as well as the matrices $\mathbf{U}_{k+1}$ and $\mathbf{V}$ coming directly from the Lanczos bidiagonalization algorithm, an implementation from Regularization Tools [7] was changed for working with the object oriented structures in Regularization Tools XP [12]. The LSQR algorithm using embedded Lanczos bidiagonalization, is from Regularization Tools XP.

4.3 GMRES

The *Generalized Minimum Residual* algorithm, GMRES, is a method developed by Saad and Schultz [21] for working with general square matrices $\mathbf{A}$, where no assumptions on symmetry is made. In the case of symmetric $\mathbf{A}$, the more efficient, but mathematically equivalent *MINRES*-algorithm, have been implemented. To discuss GMRES, we first introduce the *Arnoldi Process* which is the basic building block for the algorithm.

4.3.1 Arnoldi Process

The *Arnoldi process* is a method for forming an orthogonal projection onto the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$. The method was invented by Arnoldi in 1951 and thought to give the possibility of estimating the eigenvalues of the matrix $\mathbf{A}$, which it indeed does. The algorithm is given in Algorithm 4.2.

Algorithm 4.2 *Arnoldi Process*

$[\mathbf{W}_{k+1}, \mathbf{H}_{k+1}] = \text{ARNOLDI}(\mathbf{A}, \mathbf{x}, k)$

Given a normed starting vector $\mathbf{x}$*, a matrix* $\mathbf{A}$*, and a parameter* k*, it returns an orthonormal basis* $\mathbf{W}_{k+1}$ *for the Krylov subspace* $\mathcal{K}_k(\mathbf{A}, \mathbf{x})$*, and the Hessenberg matrix* $\mathbf{H}_k$*:*

```
$\mathbf{w_1} = \mathbf{x}$
FOR $k = 1, 2, 3, \ldots$
    $\mathbf{v} = \mathbf{A}\mathbf{w_k}$
    FOR $j = 1$ TO $k$
        $h_{j,k} = \mathbf{w_j}^H \mathbf{v}$
        $\mathbf{v} = \mathbf{v} - h_{j,k}\mathbf{w_j}$
    $h_{k+1,k} = \|\mathbf{v}\|_2$
    $\mathbf{w_{k+1}} = h_{n+1,k}^{-1}\mathbf{v}$
```

The normed starting vector $\mathbf{x}$ is chosen to be the first vector in the orthonormal basis $\mathbf{W}_k$. As seen, in each of the j iterations, the previous vector in $\mathbf{W}_k$ is multiplied from the left by $\mathbf{A}$, which exactly generates the wanted Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{x})$. To get an orthonormal basis, an orthogonalization of the newly generated vector to all the previous vectors is needed. This is simply done by modified Gram-Schmidt orthogonalization in the inner loop. The starting vector $\mathbf{x}$ can in general be any vector, but choosing a vector with large components in the directions of the largest eigenvectors of $\mathbf{A}$ will of course enrich the Krylov subspace with components in those directions. This is obtained by choosing the right-hand side $\mathbf{b}$ as starting vector.

After running the iteration k times, the following relation holds:

$$\mathbf{A}\mathbf{W}_k = \mathbf{W}_{k+1}\mathbf{H}_k, \tag{4.16}$$

where the columns of $\mathbf{W}_k \in \mathbb{R}^{n\times k}$ is an orthonormal basis spanning the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$, and $\mathbf{H}_k \in \mathbb{R}^{(k+1)\times k}$ is an upper Hessenberg matrix, i.e. a matrix of the following form:

$$\mathbf{H}_k = \begin{bmatrix} h_{1,1} & \dots & \dots & h_{1,k} \\ h_{2,1} & h_{2,2} & \dots & h_{2,k} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & & h_{k,k-1} & h_{k,k} \\ 0 & \dots & 0 & h_{k+1,k} \end{bmatrix}, \tag{4.17}$$

containing the appropriate scaling and rotation parameters for (4.16) to hold.

In the special case when $\mathbf{A}$ is a Hermitian matrix, i.e. $\mathbf{A}^T = \mathbf{A}$, then the upper $(k \times k)$ part of $\mathbf{H}$ is symmetric, which is seen by multiplying both sides of (4.16) by $\mathbf{W}_k^T$:

$$\mathbf{W}_k^T\mathbf{A}\mathbf{W}_k = \mathbf{W}_k^T\mathbf{W}_{k+1}\mathbf{H}_k = \begin{bmatrix}\mathbf{I_k} & 0\end{bmatrix}\mathbf{H}_k, \tag{4.18}$$

where $\mathbf{I}_k$ is the identity matrix of size $(k \times k)$. As $\mathbf{H}_k$ still has Hessenberg form, the only possibility is tridiagonal. In this case the iterations simplifies a great deal, leading to the Lanczos tridiagonalization, which is connected to the bidiagonalization discussed earlier. This is the fact exploited by MINRES to make a more efficient implementation.

4.3.2 GMRES Algorithm

Now turning to the GMRES algorithm, we want to find the solution $\mathbf{x}^{(k)}$ that minimizes the residual in the Krylov subspace generated by the Arnoldi Process:

$$\mathbf{x}^{(k)} = \min_{\mathbf{x}\in\mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2. \tag{4.19}$$

An advantage with this algorithm, compared to the Lanczos bidiagonalization, is that no multiplications with $\mathbf{A}^T$ are required. This of course saves work, but most important, the matrix $\mathbf{A}^T$ or operation is not always available. E.g. in applications where $\mathbf{A}$ is given as a black box function, returning $\mathbf{A}$ times a given input vector. In such a case it might be difficult, if not impossible, to get access to $\mathbf{A}^T$ or a similar black box for evaluating the matrix vector product with $\mathbf{A}^T$.

As for the Lanczos iterations, so-called *Ritz pairs* consisting of *Ritz values* and *Ritz vectors* also exists for the Arnoldi process. In this connection, the Ritz values are defined as the eigenvalues of upper $(k \times k)$ square part of $\mathbf{H}_k$. These Ritz values approximate the true eigenvalues of $\mathbf{A}$.

So to describe and analyze the solution in this case, we would like to proceed using eigenvalues and eigenvectors. But for calculating those Ritz pairs, only the top square part of $\mathbf{H}_k$ is used. In the GMRES algorithm, the full Hessenberg matrix $\mathbf{H}_k$, which is rectangular, is used. In this case the eigenvalue decomposition is of course not possible and instead we use the SVD to analyze the algorithm. The SVD of $\mathbf{H}_k$ is given as:

$$\mathbf{H}_k = \widehat{\mathbf{P}}\widehat{\mathbf{\Gamma}}\widehat{\mathbf{Q}}^T, \tag{4.20}$$

where $\widehat{\mathbf{P}}$ and $\widehat{\mathbf{Q}}$ are the left and right singular matrices and $\widehat{\mathbf{\Gamma}}$ is diagonal and contains the singular values. By combining (4.20) and (4.16), we get:

$$\mathbf{A}\left(\mathbf{W}_k\widehat{\mathbf{Q}}\right) = \left(\mathbf{W}_{k+1}\widehat{\mathbf{P}}\right)\widehat{\mathbf{\Gamma}} \tag{4.21}$$

Unfortunately, the convergence of the basis vectors for this decomposition is not connected to the Ritz vectors and does therefore not necessarily converge to neither the eigenvalue decomposition or the SVD of $\mathbf{A}$. Still it is possible to construct a regularized solution from the decomposition. Choosing the right-hand side $\mathbf{b}$ to be the starting vector for the Arnoldi process, and defining $\beta = \|\mathbf{b}\|_2$, the following is obtained:

$$\begin{aligned}
\mathbf{x}^{(k)} &= \left(\mathbf{W}_k\widehat{\mathbf{Q}}\right)\widehat{\mathbf{\Gamma}}^{-1}\left(\mathbf{W}_{k+1}\widehat{\mathbf{P}}\right)^T\mathbf{b} && (4.22)\\
&= \sum_{i=1}^{k} \frac{\left(\mathbf{W}_{k+1}\widehat{\mathbf{P}}\right)^T_{:,i}\mathbf{b}}{\hat{\gamma}_i}\left(\mathbf{W}_k\widehat{\mathbf{Q}}\right)_{:,i} && (4.23)\\
&= \sum_{i=1}^{k} \frac{\widehat{\mathbf{P}}_{[1,i]}\beta}{\hat{\gamma}_i}\left(\mathbf{W}_k\widehat{\mathbf{Q}}\right)_{:,i}. && (4.24)
\end{aligned}$$

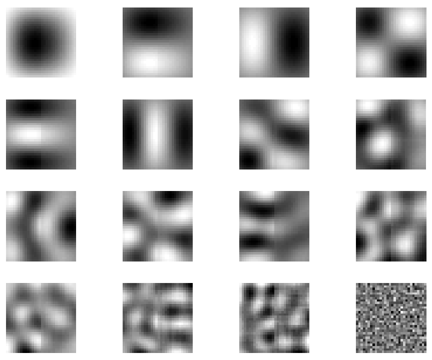


Figure 4.2: *2D Basis vectors for GMRES solution after 16 iterations (from top left to bottom right).*

To derive (4.24) from (4.23), it is used that the first basis vector in $\mathbf{W}_{k+1}$ is chosen to be $\beta^{-1}\mathbf{b}$ and that all the following vectors are orthogonal to this one. $\widehat{\mathbf{P}}_{[1,i]}$ denotes the element $(1, i)$ of $\widehat{\mathbf{P}}$. As mentioned, the space spanned by the vectors in $\mathbf{W}_k\widehat{\mathbf{Q}}$, cannot be assumed to approximate the singular vectors of $\mathbf{A}$. The behavior of these solutions must be studied further, and the basis vectors $\mathbf{W}_k\hat{\mathbf{Q}}$ for a small test example are shown in Fig. 4.2. These basis vectors should be compared to the basis vectors in Fig. 4.1 for the LSQR-solution. Even though some correspondence is seen, it is also observed that fewer vectors have converged to something corresponding to the true SVD of $\mathbf{A}$, and especially that the last vector (bottom right) is high-frequent and very different.

In practice, the regularized solution is calculated via a QR factorization. What we want to minimize is the two-norm residual in (4.19). The regularized solution lies in the space spanned by $\mathbf{W}_k$, and therefore $\mathbf{x}_{GMRES}^{(k)} = \mathbf{W}_k\mathbf{y}$ for some $\mathbf{y} \in \mathbb{R}^k$. Here it is assumed that the initial guess $\mathbf{x}_0$ is the zero vector. With $\mathbf{e}_1$ denoting the first vector of the identity matrix of size $k \times k$, i.e. $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$, this leads to the following least squares problem:

$$\mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{b} - \mathbf{A}\mathbf{W}_k\mathbf{y} = \beta\mathbf{W}_1 - \mathbf{W}_{k+1}\mathbf{H}_k\mathbf{y} = \mathbf{W}_{k+1}\left(\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\right), \tag{4.25}$$

as $\mathbf{W}_{k+1}$ has orthonormal columns and therefore does not change the 2-norm, we have the following identity:

$$\min_{\mathbf{x}\in\mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 = \min_{\mathbf{y}\in\mathbb{R}^k} \|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2 \tag{4.26}$$

According to the calculations done by J. M. Rasmussen in [18], an expression for the regularized solution, by means of the QR factorization $\mathbf{H}_k = \mathbf{Q}_k\mathbf{R}_k$ is given by:

$$\mathbf{x}^{(k)} = \beta\mathbf{W}_k\widehat{\mathbf{R}}_k^{-1}\widehat{\mathbf{Q}}_k^H\mathbf{e}_1, \tag{4.27}$$

where $\widehat{\mathbf{R}}$ and $\widehat{\mathbf{Q}}$ are equal to $\mathbf{R}$ with the last row and $\mathbf{Q}$ with the last column removed.

Implementation

Different implementations of the algorithm have been used. First an implementation inspired by J. M. Rasmussen's code from [18] was implemented for testing purposes to work with the object oriented base classes from Regularization Tools XP. And later on, as the Toolbox was extended with a GMRES algorithm, this implementation was adopted and tested. The code for the GMRES algorithm is found in Appendix A. This code also includes some changes discussed later in connection with modification of the basic algorithm.

4.4 Different Subspaces

To gain more theoretical insight, the Krylov subspaces used in the GMRES and LSQR algorithms are studied directly. We remind that the solutions constructed after k iterations belong to the following subspaces:

$$\begin{aligned}
\mathbf{x}_{LSQR}^{(k)} &\in \mathcal{K}_k\left(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b}\right) \\
&= \operatorname{span}\left\{\mathbf{A}^T\mathbf{b}, (\mathbf{A}^T\mathbf{A})\mathbf{A}^T\mathbf{b}, \dots, (\mathbf{A}^T\mathbf{A})^{k-1}\mathbf{A}^T\mathbf{b}\right\} \qquad (4.28) \\
\mathbf{x}_{GMRES}^{(k)} &\in \mathcal{K}_k\left(\mathbf{A}, \mathbf{b}\right) \\
&= \operatorname{span}\left\{\mathbf{b}, \mathbf{A}\mathbf{b}, \dots, \mathbf{A}^{k-1}\mathbf{b}\right\} \qquad (4.29)
\end{aligned}$$

Using the SVD of $\mathbf{A}$ (2.6), and that $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{I}$, we see that the subspace from which LSQR constructs the solution is the following:

$$\begin{aligned}
\mathbf{x}_{LSQR}^{(k)} &\in \operatorname{span}\left\{\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}, \mathbf{V}\mathbf{\Sigma}^3\mathbf{U}^T\mathbf{b}, \dots, \mathbf{V}\mathbf{\Sigma}^{2k-1}\mathbf{U}^T\mathbf{b}\right\} \Rightarrow \\
\mathbf{V}^T\mathbf{x}_{LSQR}^{(k)} &\in \operatorname{span}\left\{\mathbf{\Sigma}\boldsymbol{\varphi}_u, \mathbf{\Sigma}^3\boldsymbol{\varphi}_u, \dots, \mathbf{\Sigma}^{2k-1}\boldsymbol{\varphi}_u\right\},
\end{aligned}$$

where $\boldsymbol{\varphi}_u = \mathbf{U}^T\mathbf{b}$. As $\mathbf{\Sigma}$ is diagonal, the latter subspace can be described in matrix form by two diagonal matrices and a vandermonde matrix given by the vector of squared singular values:

$$\begin{bmatrix} \varphi_{u[1]} & & \\ & \ddots & \\ & & \varphi_{u[n]} \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \begin{bmatrix} 1 & \sigma_1^2 & \sigma_1^4 & \dots \sigma_1^{2k-2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \sigma_n^2 & \sigma_n^4 & \dots \sigma_n^{2k-2} \end{bmatrix} \qquad (4.30)$$

Likewise, the subspace for the GMRES solution is given as:

$$\begin{aligned}
\mathbf{x}_{GMRES}^{(k)} &\in \operatorname{span}\left\{\mathbf{b}, \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T b, \dots, \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^{k-1}\mathbf{b}\right\} \Rightarrow \\
\mathbf{V}^T\mathbf{x}_{GMRES}^{(k)} &\in \operatorname{span}\left\{\boldsymbol{\varphi}_v, \mathbf{V}^T\mathbf{U}\mathbf{\Sigma}\boldsymbol{\varphi}_v, \dots, \left(\mathbf{V}^T\mathbf{U}\mathbf{\Sigma}\right)^{k-1}\boldsymbol{\varphi}_v\right\}, \qquad (4.31)
\end{aligned}$$

where $\boldsymbol{\varphi}_v = \mathbf{V}^T\mathbf{b}$. This subspace involves, in general, non-diagonal matrix products, hence it cannot easily be written in a simple way. This also means that the singular value σ_j not only apply to the coefficient $\boldsymbol{\varphi}_{v[j]}$, but contribute to more coefficients through the multiplications by $\mathbf{V}^T\mathbf{U}$.

In the case where $\mathbf{A}$ is symmetric, then the left and right singular vectors $\mathbf{U}$ and $\mathbf{V}$ are identical and of course still orthonormal. This leads to $\mathbf{V}^T\mathbf{U} = \mathbf{I}$, $\boldsymbol{\varphi}_u = \boldsymbol{\varphi}_v$, and thus reducing the expression for the Krylov subspace for GMRES to:

$$\mathbf{V}^T\mathbf{x}_{GMRES}^{(k)} \in \operatorname{span}\left\{\boldsymbol{\varphi}_v, \boldsymbol{\Sigma}\boldsymbol{\varphi}_v, \dots, \boldsymbol{\Sigma}^{k-1}\boldsymbol{\varphi}_v\right\} \tag{4.32}$$

In this case, a matrix expression similar to (4.30) for the LSQR solution can be written as:

$$\begin{bmatrix} \boldsymbol{\varphi}_{u[1]} & & \\ & \ddots & \\ & & \boldsymbol{\varphi}_{u[n]} \end{bmatrix} \begin{bmatrix} 1 & \sigma_1 & \sigma_1^2 & \dots\sigma_1^{k-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \sigma_n & \sigma_n^2 & \dots\sigma_n^{k-1} \end{bmatrix} \tag{4.33}$$

It is observed that even in this case, the subspaces are not equal, thus giving different solutions.

Multiplying the singular values to the vectors of the subspace must dampen the higher frequencies and thereby the noise in these vectors. This is the opposite to amplifying the noise components in the solution where a division with the singular values are performed (2.14). Comparing the two subspaces, it is obvious that this damping effect of the higher frequencies must be much more pronounced for LSQR than for GMRES. In fact the first vector of the GMRES subspace is not damped at all, only consisting of an orthonormal transformation of the normed, blurred right-hand side $\mathbf{b}$. It is also seen that in the symmetric case, the vectors in the LSQR subspace are equal to the 2nd, 4th, 6th etc. vectors in the GMRES subspace. The remaining vectors must come in between and be less damped.

In the non-symmetric case where $\mathbf{U}$ and $\mathbf{V}$ are not identical, the matrix in the expression for the GMRES subspace is not diagonal. This leads to vectors different from those used in the LSQR subspace. As the singular values, through the multiplication with the non-diagonal matrix, do not only apply to specific coefficients, the damping effect must also be smaller.

All in all, we must expect the GMRES solutions to contain more noise than the LSQR solutions in both the symmetric and the non-symmetric case.

4.5 Filter Factors

Instead of looking at the Krylov subspaces one could also look at the SVD filter factors. Writing the solution in terms of the SVD of $\mathbf{A}$ is for instance seen in (3.27), where $\boldsymbol{\Psi}$ denotes the general filter factors. Looking at the solution in the domain of the left singular vectors of $\mathbf{A}$, and assuming that the exact solution $\mathbf{x}$ as well as the noise $\mathbf{e}$ are known, then we can calculate the SVD filter factors directly by:

$$\begin{aligned} \mathbf{V}^T\mathbf{x}_{\text{reg}} &= \boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} \qquad (4.34) \\ &= \boldsymbol{\Psi}\left(\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\bar{\mathbf{b}} + \boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{e}\right) \\ &= \boldsymbol{\Psi}\left(\mathbf{V}^T\mathbf{x} + \boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{e}\right) \Leftrightarrow \\ \boldsymbol{\Psi} &= \operatorname{diag}\left(\mathbf{V}^T\mathbf{x}_{\text{reg}} \oslash \left(\mathbf{V}^T\mathbf{x} + \boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{e}\right)\right), \qquad (4.35) \end{aligned}$$

where $\oslash$ denotes the element-wise division of the two vectors and $\operatorname{diag}(\mathbf{z})$ denotes the diagonal matrix with the elements of the vector $\mathbf{z}$ along this diagonal. These calculations are only possible when the problem is fairly small, or when $\mathbf{A}$ is a simple Kronecker product. In this case the SVD can be calculated for the matrices constructing the Kronecker product and then combined to the full SVD of $\mathbf{A}$ as seen in (3.13).

In connection with especially images, the 2D-DCT has been introduced as a frequency decomposing transform. And other filter factors arise by projecting the solution to this domain. A. K. Louis [13] suggests that a filtered solution in general the solution can be given as:

$$\mathbf{x}_{reg} = \mathbf{F}_{\text{post}}\mathbf{A}^{\#}\mathbf{b} = \mathbf{A}^{\#}\mathbf{F}_{\text{pre}}\mathbf{b}, \tag{4.36}$$

where $\mathbf{A}^{\#}$ is some regularized inverse of $\mathbf{A}$ and the filter factors are given as either a post filter or a pre filter instead of like in (4.35) as a filtering of the solution coefficients. Now performing not an SVD of $\mathbf{A}$, but a DCT decomposition of $\mathbf{A}$, we get:

$$\mathbf{A} = \mathbf{G}\boldsymbol{\Xi}\mathbf{G}^T \tag{4.37}$$

In this expression, $\boldsymbol{\Xi}$ is not necessarily diagonal. In fact this would require that $\mathbf{G}$ is the matrix containing the eigenvectors of $\mathbf{A}$.

In the square case where $m = n$ then combining (4.36) with (4.37) the regularized solution can be written as:

$$\begin{aligned}
\mathbf{x}_{\text{reg}} &= \mathbf{F}_{\text{post}}\mathbf{G}\boldsymbol{\Xi}^{-1}\mathbf{G}^T\mathbf{b} \\
\mathbf{x}_{\text{reg}} &= \mathbf{G}\mathbf{G}^T\mathbf{F}_{\text{post}}\mathbf{G}\boldsymbol{\Xi}^{-1}\mathbf{G}^T\mathbf{b} && (4.38)\\
\mathbf{G}^T\mathbf{x}_{\text{reg}} &= \mathbf{F}_{\text{DCT}}\boldsymbol{\Xi}^{-1}\mathbf{G}^T\mathbf{b}, \quad \text{where} \quad \mathbf{F}_{\text{DCT}} = \mathbf{G}^T\mathbf{F}_{\text{post}}\mathbf{G} && (4.39)
\end{aligned}$$

Here the filter factors are given by the DCT transformation of some post filter $\mathbf{F}_{\text{post}}$, and the expression could be compared to (4.34). Unfortunately the matrix $\mathbf{F}_{\text{DCT}}$ is still not diagonal, and calculating these filter factors explicitly is not possible. As no information can be obtained from this, we have to turn to cases, where direct calculations are possible, and in the following at the classical filter factors, and we only use the standard SVD filter factors described in (4.35).

4.5.1 Ritz Polynomials

Whereas the filter factors in (4.35) are directly calculated, using any regularized solution obtained, it is possible to theoretically describe the solution and the filter factors in case of LSQR. This algorithm is controlled by a polynomial, and the regularized solution is in this case described by the so-called *Ritz polynomial*:

$$\mathcal{R}_k(\sigma) = \prod_{j=1}^{k} \frac{\theta_j^{(k)} - \sigma^2}{\theta_j^{(k)}}, \tag{4.40}$$

where $\theta_j^{(k)}$ are the k Ritz values, defined as the eigenvalues of $\mathbf{B}_k^T\mathbf{B}_k$, and $\mathbf{B}_k$ is the bi-diagonal matrix (4.9) from the Lanczos bidiagonalization in Algorithm 4.1. The LSQR solutions are therefore controlled by the squared singular values of $\mathbf{B}_k$ and

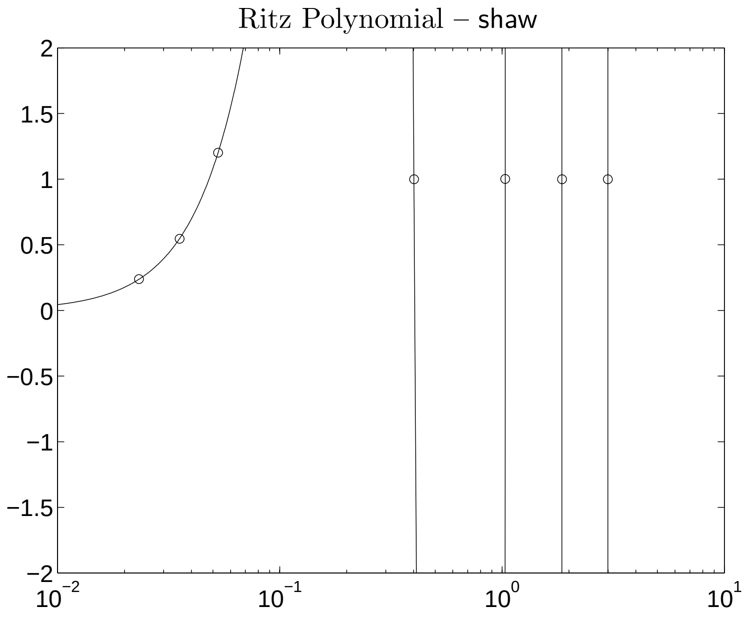


Figure 4.3: $1 - \mathcal{R}_5(\sigma)$ *for the standard* shaw test problem after running 5 LSQR iterations. The filter factors are shown with ◦'s.

their convergence to the true singular values of $\mathbf{A}$. The filter factors for the LSQR regularized solutions are then given by [8, Eqn. 6.22]:

$$f_i^{(k)} = 1 - \prod_{j=1}^{k} \frac{\theta_j^{(k)} - \sigma_i^2}{\theta_j^{(k)}}, \quad i = 1, \ldots, n. \tag{4.41}$$

Here σ_i is the i'th singular value of the blurring matrix $\mathbf{A}$. This function is plotted for the continuous variable σ and is a fast oscillating polynomial, being equal to one whenever $\theta_j^{(k)}$ is equal to one of the singular values of $\mathbf{A}$. So the filter factors obtained by running LSQR, depend on how the singular values of $\mathbf{B}_k$ converge to the true singular values of $\mathbf{A}$. An illustration of the filter factors and the Ritz polynomial is given in [8, Fig. 6.2]. The example is the shaw test problem from Regularization Tools [7], and the plot of the Ritz polynomial and the filter factors after 5 LSQR iterations is repeated here in Fig. 4.3.

As described by P. C. Hansen [8, Section 6.3.2], the polynomial oscillates and takes on very large positive and negative values between the singular Ritz values that have converged to a true singular value, σ_i, of $\mathbf{A}$. Also it is observed that after k iterations, approximately k filter factors are 1, whereas the remaining fall off towards zero, so that:

$$\begin{aligned} f_i^{(k)} &\approx 1 \qquad \text{for} \qquad \sigma_i^2 \geq \theta_k^{(k)} \\ 0 \leq f_i^{(k)} &\leq 1 \qquad \text{for} \qquad \sigma_i^2 \leq \theta_k^{(k)}, \end{aligned}$$

where $\theta_k^{(k)}$ is the smallest Ritz value. As the filter factors are tied to the Ritz polynomial which in turn is tied to the convergence of the singular Ritz values to the real singular values of $\mathbf{A}$, this is actually the basic reason why the iteration number k can be considered a regularization parameter.

We now turn to look at some images to see how all this works in practise.

CHAPTER 5

Analysis and Insight

In Section 4.4 we say that the two iterative algorithms, LSQR and GMRES, must generate different solutions as LSQR minimizes the two-norm of the residual in $\mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$, and GMRES minimizes the two-norm of the residual in $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$. In this chapter different test problems are introduced to illustrate and evaluate the differences between LSQR and GMRES. We gain insight into the visual properties of the regularized solutions, the subspaces used, as well as the resulting filter factors.

5.1 An Image Problem – and its Solution

An image deblurring problem is introduced to illustrate the differences between the basic LSQR and GMRES algorithms. The original image is the penguin image found in Appendix B Fig. B.3, and the blurring used is the simple spatially invariant blur with $\sigma_c = \sigma_r = 4$, described in Section 3.3, Eqn. (3.24). We apply white noise $\mathbf{E}$ with the noise level, η, set to:

$$\eta = \frac{\|\mathbf{E}\|_2}{\|\overline{\mathbf{B}}\|_2} = 0.05.$$

The blurred and noisy realization of the image are seen in Fig. 5.1.

We denote by $\mathbf{X}^{(k)}_{\text{LSQR}}$ and $\mathbf{X}^{(k)}_{\text{GMRES}}$ the LSQR solution and the GMRES solution after k iterations, and following the notation from Section 3.2, the 2D-DCT transformed solutions are denoted:

$$\widehat{\mathbf{X}}^{(k)}_{\text{LSQR}} = \mathbf{G}^T\mathbf{X}^{(k)}_{\text{LSQR}}\mathbf{G} \tag{5.2}$$

$$\widehat{\mathbf{X}}^{(k)}_{\text{GMRES}} = \mathbf{G}^T\mathbf{X}^{(k)}_{\text{GMRES}}\mathbf{G}. \tag{5.3}$$

The solutions after a varying number of iterations are shown in Fig. 5.2 for the two algorithms. The obvious difference is that GMRES introduces a lot of noise, whereas the LSQR solutions are observed to behave somewhat similar to the Tikhonov solutions from Fig. 3.15 by introducing “freckles”. It is basically those visual artifacts that are discussed in the following.

First, to see where the noise in the GMRES solutions comes from, we turn to look at the LSQR and GMRES subspaces from which the solutions are found.

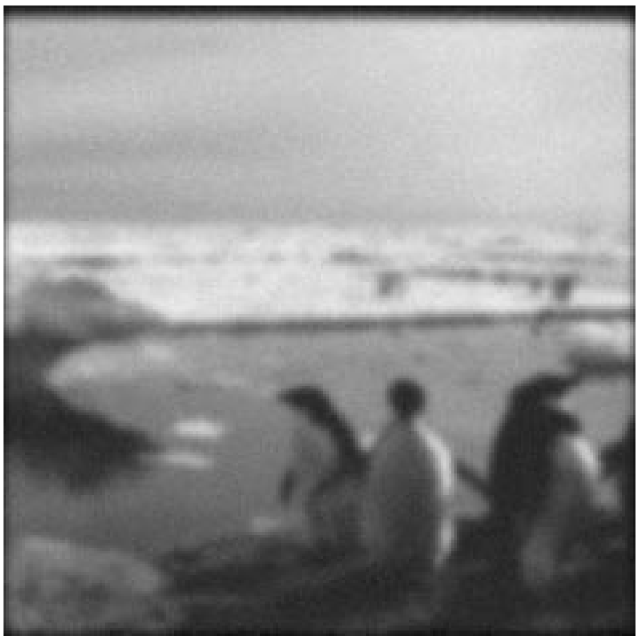


Figure 5.1: *Blurred and noisy realization of penguin image.*

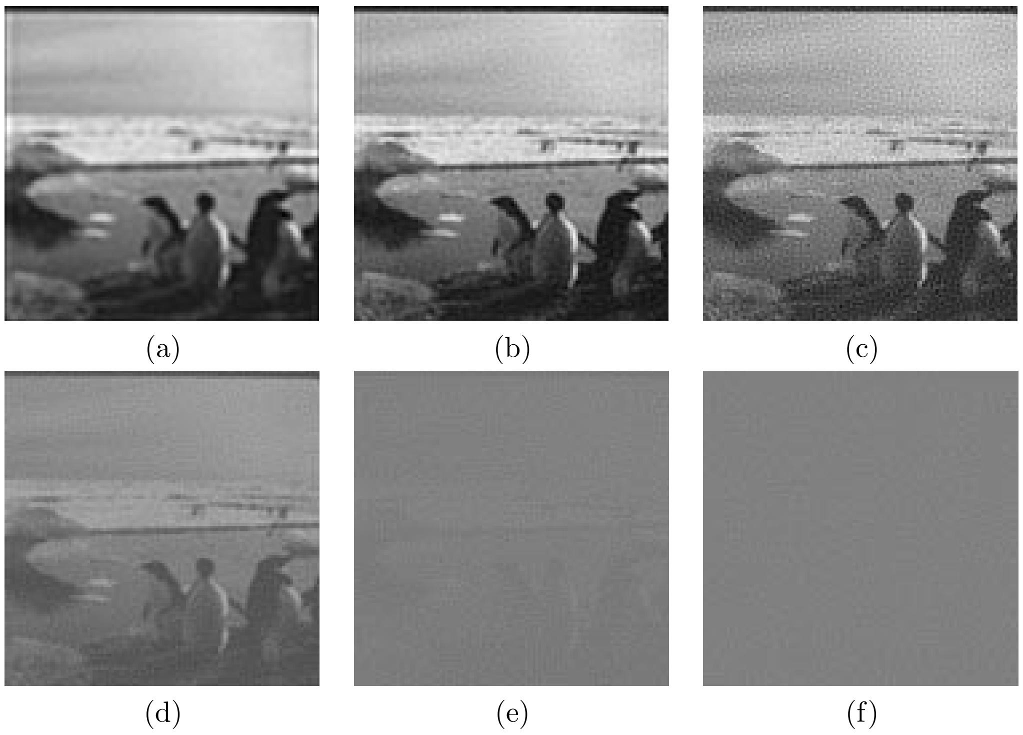


Figure 5.2: *Regularized solutions. (a), (b), and (c) using LSQR after 4, 10 and 25 iterations, and (d), (e), and (f) using GMRES after 4, 10 and 25 iterations.*

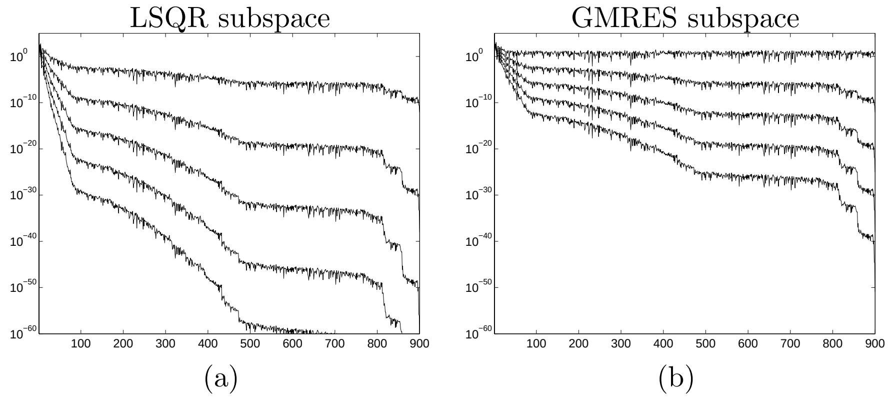


Figure 5.3: *First five Krylov vectors in the symmetric case for (a) LSQR,* $\mathbf{\Sigma}^{2k-1}\mathbf{U}^T\mathbf{b}$ *and (b) GMRES,* $\mathbf{\Sigma}^{k-1}\mathbf{V}^T\mathbf{b}$ *for* $k = 1, \ldots, 5$. *The ones in the top are the first vectors,* $\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}$ *and* $\mathbf{V}^T\mathbf{b}$*, respectively.*

5.2 Krylov Subspaces

Four different situations are tried out here. The two algorithms combined with a symmetric vs. a non-symmetric matrix $\mathbf{A}$. To be able to do direct calculations of the SVD in both cases, only a small part of the penguin image from above of size 30×30 is used.

The matrix $\mathbf{A} \in \mathbb{R}^{900\times900}$ is the Kronecker product of two identical Toeplitz matrices doing Gaussian blurring, as defined in (3.24). Calculating the SVD of $\mathbf{A}$ by means of (2.6), or in this case the Kronecker SVD (3.13), it is possible to explicitly construct the Krylov subspaces from which the solutions are found. The Krylov subspaces are defined in (4.30) and (4.32) for the LSQR solutions and the GMRES solutions respectively.

In Fig. 5.3, the first five vectors in those Krylov spaces are illustrated. In the plots, the vectors on top are the first Krylov vector for the two subspaces, $\mathbf{\Sigma}\mathbf{U}^T\mathbf{b}$ and $\mathbf{V}^T\mathbf{b}$. The second from the top are the second Krylov vectors, and so on and so forth. Here the damping of the Krylov vectors, described in Section 4.4, is clearly observed.

In the case of LSQR, the first vector is clearly damped by the multiplication of the coefficients $\mathbf{U}^T\mathbf{b}$ with $\mathbf{\Sigma}$ as it falls off slowly for the higher coefficients. Comparing with the first vector in the case of GMRES, this falls off only for around the first 50 coefficients, and stays constant at a significant level connected with the noise level. It is also seen that the 2nd GMRES vector is similar to the 1st LSQR vector, the 4th GMRES-vector is similar to the 2nd LSQR vector etc. Actually those are theoretically identical as $\mathbf{U}$ and $\mathbf{V}$ in this case are identical, thus leading to the solution coefficients $\mathbf{U}^T\mathbf{b}$ and $\mathbf{V}^T\mathbf{b}$ being identical as well.

In case of the matrix $\mathbf{A}$ being non-symmetric, or in this case the blurring being spatially variant, the matrix $\mathbf{A}$ is generated as (3.25). The same plots as for the symmetric case are seen in Fig. 5.4 for the case with spatially variant blur. It is observed that the Krylov vectors in case of LSQR behave as in the symmetric case whereas the Krylov vectors for the GMRES subspace become even less damped. In fact the vectors 2 to 5 are difficult to distinguish from one another in this plot. This

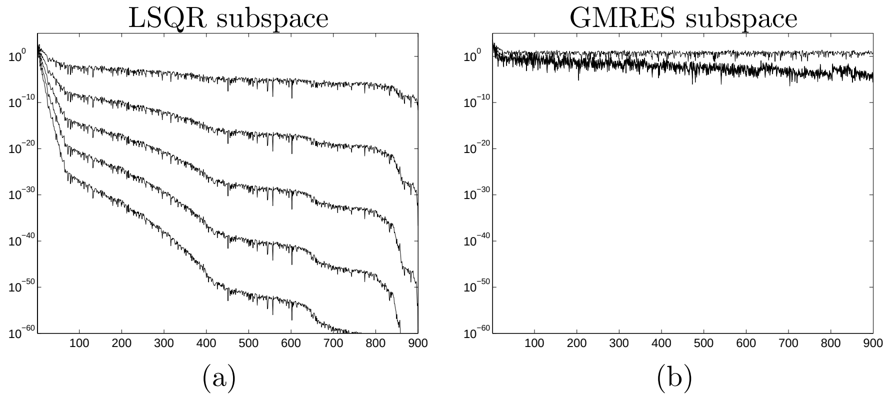


Figure 5.4: *First five Krylov-vectors in the non-symmetric case for (a) LSQR and (b) GMRES.*

is due to the mixing of the singular values through the multiplication with the, in this case, non-diagonal matrix $\mathbf{V}^T\mathbf{U}$. From this plot it seems that no new information is introduced to the subspace from those vectors. But another way of defining the new information contents for one vector, compared to the previous vectors, is to look at the angle between the subspace spanned by the k previous vectors and the new vector $k+1$. This is done for LSQR and GMRES respectively in Table 5.1. Here it is seen that new information in fact is introduced to the GMRES solution, similar to the information introduced to the LSQR solution even though no difference is seen from the plots. Actually, in these examples there is a larger angle between the GMRES subspaces than the LSQR subspaces.

From this, we see that looking directly at the Krylov subspaces clearly explains why the noise in the regularized GMRES solutions blows up as shown in Fig. 5.2. It also indicates that the GMRES solutions are very sensitive to noise, so if the blurred right-hand side is covered with too much noise, even the first GMRES iterates will be unusable. We try to change the noise level to:

$$\eta = \frac{\|\mathbf{E}\|_2}{\|\overline{\mathbf{B}}\|_2} = 0.5,$$

and after only two and four iterations with GMRES, we get the solutions in Fig. 5.5. As clearly seen, these solutions are unusable.

Angle	Sym. LSQR	Sym. GMRES	Asym. LSQR	Asym. GMRES
$\{1\}\angle\{2\}$	1.0135e-01	1.1907e-01	9.8835e-02	1.2230e-01
$\{1,2\}\angle\{3\}$	1.1956e-02	2.9412e-02	1.2694e-02	3.4057e-02
$\{1,2,3\}\angle\{4\}$	2.4806e-03	5.4537e-03	1.9956e-03	2.0978e-02
$\{1,2,3,4\}\angle\{5\}$	1.7561e-04	1.4625e-03	1.0639e-04	2.4321e-02

Table 5.1: *Angle between subspaces*

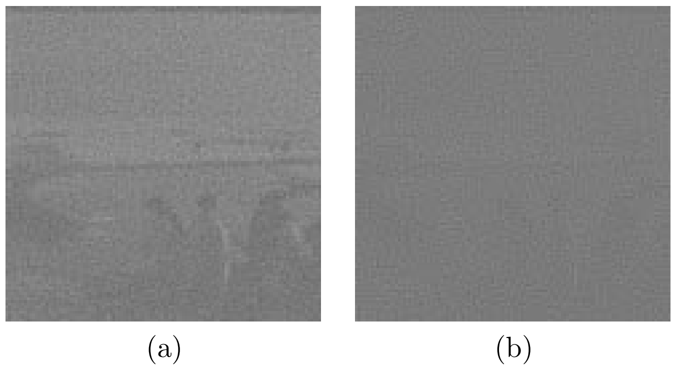


Figure 5.5: *GMRES solution with noise level set to $\eta = 0.5$. (a) After 2 iterations, and (b) after 4 iterations.*

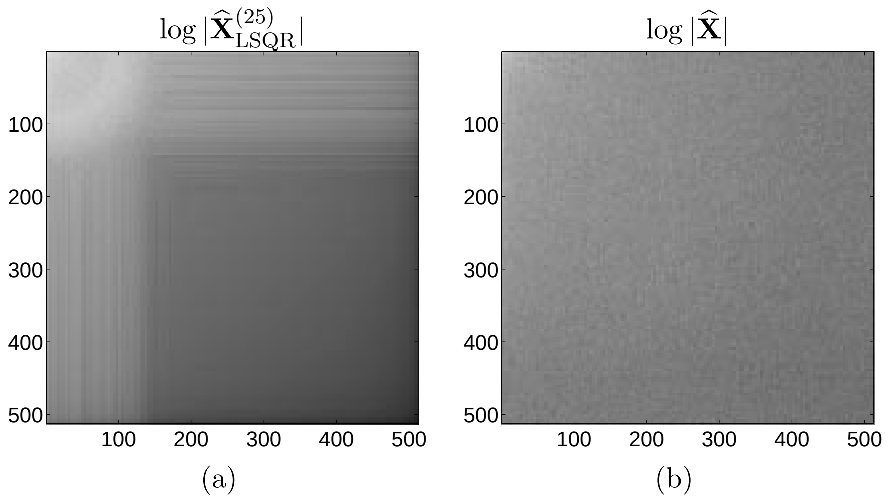


Figure 5.6: *(a) Image showing the LSQR solution in Fig. 5.2 (c) transformed to the DCT frequency domain. (b) Image showing the frequency domain of the true penguin image from Fig. B.3.*

5.3 The "Freckles"

When reconstructing with especially LSQR, it is seen that the solutions obtained get covered with small "freckles" – somewhat similar to the effects seen for the TSVD and especially the Tikhonov solutions in Fig. 3.13 and Fig. 3.15. To investigate the "freckles", we need to get a handle to these effects, and understand more specifically how they look. We now use the 2D-DCT to study the frequency content of the "freckles".

5.3.1 Applying the DCT

The "freckles" are quite dominant in the solutions and have to appear in some way in the frequency domain. In Fig. 5.6 (a), an image showing the DCT transform of the last LSQR solution from Fig. 5.2, $\widehat{\mathbf{X}}^{(25)}_{\mathrm{LSQR}}$, is shown. Very large as well as very small values appear, so to be able to visualize the frequency domain, what is actually shown is the logarithm of the absolute value of the spectrum, $\log|\widehat{\mathbf{X}}^{(25)}_{\mathrm{LSQR}}|$.

In this figure it is clearly seen that the spectrum primarily consists of low-frequent information in the upper left corner, as well as some border effects. But more interestingly, it is also seen that a small band of frequencies lie as a ring around 100 pixels from the upper left corner. This means that some frequencies, living in this area of the spectrum, are in some way dominating. Now to be sure that this ring is not something coming from the true image, the same DCT plot of the true image from Fig. B.3 is seen in the right part of the figure. This is much more high-frequent, as expected, and does not have the same band of amplified frequencies as does the LSQR solution.

Another way to verify that this enhanced band of frequencies is actually the reason for the "freckles" is to artificially construct a frequency domain, only consisting of a band of random values, corresponding to the ring seen in Fig. 5.6 (a). This artificially constructed frequency domain $\widehat{\mathbf{F}} \in \mathbb{R}^{N\times N}$ is defined as:

$$
\begin{aligned}
I &= \left\{ (i,j) \mid f_1 \le \frac{\sqrt{(i-1)^2+(j-1)^2}}{N} \le f_2 \right\} \quad f_1, f_2 \in [0;1[, \quad f_1 < f_2 \\
\widehat{\mathbf{F}}_{[i,j]} &= \begin{cases} \mathcal{N}(0,1) & \text{for } (i,j) \in I \\ 0 & \text{for } (i,j) \notin I \end{cases} \quad i,j = 1,2,\ldots,N, \qquad (5.5)
\end{aligned}
$$

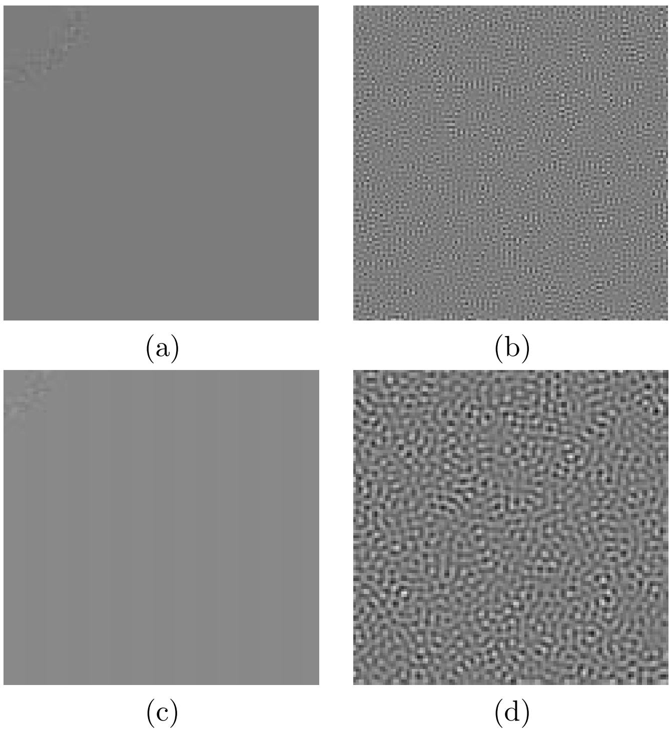


Figure 5.7: *(a) Artificially generated band of normal distributed random values to emulate band in frequency domain. $f_1 = 0.20$ and $f_2 = 0.25$ (b) The spatial representation of the band of random intensities in the frequency domain. (c) Frequency domain with $f_1 = 0.10$ and $f_2 = 0.15$. (d) Spatial representation of (c).*

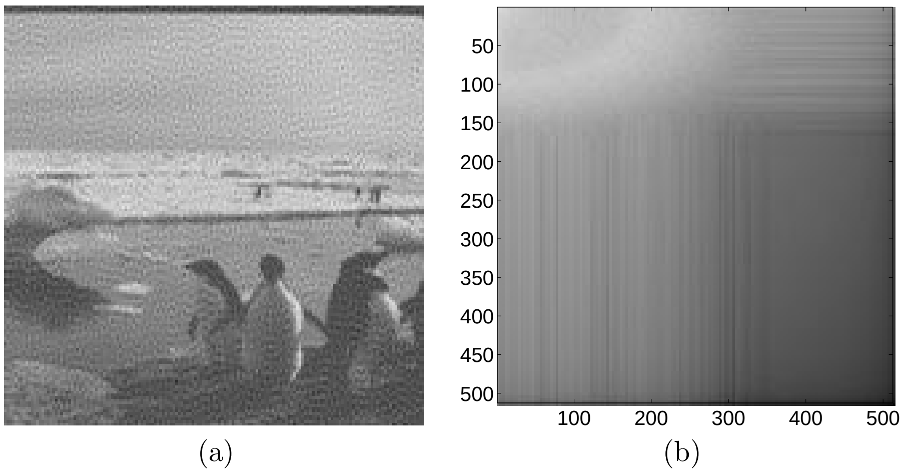


Figure 5.8: *(a) LSQR solution of penguin image non-symmetrically blurred after 25 iterations, and (b) the frequency representation of the regularized solution.*

where each element in the band is chosen from a normal distribution. Transforming this frequency image to the spatial domain, we obtain the image $\mathbf{F}$. This is shown in Fig. 5.7, where $\widehat{\mathbf{F}}$ and $\mathbf{F}$ are shown for two different choices of f_1 and f_2. Here it is clear that a band of amplified frequency components creates “freckles” in the spatial domain. The spatial images could be compared to the solutions to the noise in case of the TSVD and Tikhonov solutions from Figs. 3.13 and 3.15 (d), (e), and (f). The size of the “freckles” is seen to be connected with the position of the ring of frequencies in the frequency domain.

In Fig. 5.8, we show another example of a LSQR solution after 25 iterations. Now the blurring has been changed to be non-symmetric with $\sigma_r = 2$ and $\sigma_c = 4$, and the “freckles” are seen to be more elliptic than circular. Also in the frequency domain the ring of enhanced frequencies has become an ellipse. Section 3.5, and Fig. 3.10, show an example of how the singular vectors increase in frequency in different directions when the blurring is non-symmetric. And as seen here, the noise contribution follows the reconstruction of different frequencies, and affects one direction before the other. This indicates that more details have been reconstructed correctly in one direction, whereas less details can be reconstructed in the other direction before the noise causes “freckles”.

To further illustrate the impact of different noise levels on the “freckles”, we show one last example in Fig. 5.9. Here the noise level in the given right-hand side image, $\mathbf{B}$, has again been changed to $\eta = 0.5$. The solution shown is the one obtained after only 10 LSQR iterations. As observed, the change of noise level changes the location of the ring of enhanced frequencies, and thereby the size of the “freckles”. The “freckles” are in this case larger as the ring starts a bit closer to the upper left corner. At no time using the LSQR algorithm, the white noise in $\mathbf{B}$ enters the solutions $\mathbf{X}_{\text{LSQR}}^{(k)}$ as white noise. The noise always seems to trigger different frequencies, resulting in some of the described “freckles”. Those being small, large, symmetrical, or elliptical. It is seen that the noise level η, as well as the amount of blurring σ_r and σ_c change the “freckles”.

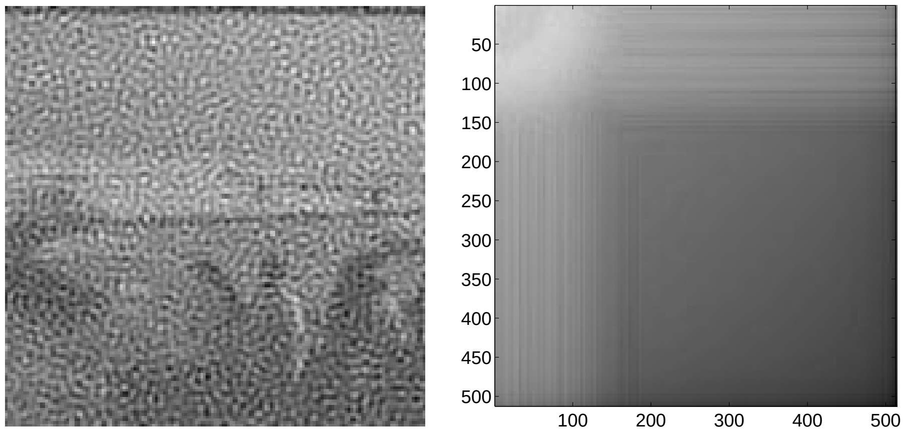


Figure 5.9: *(a) LSQR solution of penguin image after 10 iterations with the noise level set to* $\eta = 0.5$*. (b) Frequency domain of the solution.*

To out knowledge, these "freckles" are not commented in the literature, and the origin is seeked in the following. But first it could be interesting to see whether this phenomenon does also apply to one-dimensional problems.

5.3.2 One Dimensional Problem

To get a one-dimensional problem similar to the two-dimensional problems studied above, we use one row of the penguin image to generate the one-dimensional problem. Row number $i = 256$ is picked out of the full penguin image, and the blurring applied to this row is the first matrix contained in the Kronecker product describing spatially invariant and symmetric blur in the 2D case. This blurring is simply one-dimensional Gaussian blur. Then the equation for the i'th row in the 2D problem in (3.11) is described by:

$$\mathbf{A}_1 \mathbf{X}^T_{[i,:]} = \mathbf{b}, \tag{5.6}$$

where $\mathbf{b} = \mathbf{A}_1 \mathbf{X}^T_{\text{true},[i,:]} + \mathbf{e}$ and $\mathbf{e}$ is normal distributed white noise of size 0.05 compared to the true blurred image. Plots of $\mathbf{X}^T_{\text{true},[i,:]}$ and $\mathbf{b}$ are seen in Fig. 5.10.

Now solving this problem by means of LSQR leads to the solutions in Fig. 5.11 (a), (b), and (c), where the solutions after 10, 20, and 30 iterations are shown. The corresponding DCT transformed solutions are shown in Fig. 5.11 (d), (e), and (f). Here the simple one-dimensional DCT from Definition 3.4 is used.

As seen from the frequency plots, the noise band from the 2D case appears as well here as an amplification of a small frequency band. This example problem is fairly small, and a direct study of the singular values, filter coefficients and Ritz polynomials is possible. To illustrate the differences between the normally studied inverse problems and the ones we are working with here, we use the standard test problem, shaw [8, Section 1.4.3] [7, p. 94], which is a severely ill-posed one-dimensional problem. The Picard plot for shaw is shown in Fig. 5.12 (a), and in Fig. 5.12 (b), the corresponding Picard plot for the one dimensional penguin problem is seen. The differences are obvious. For the shaw test problem, the singular values decay very

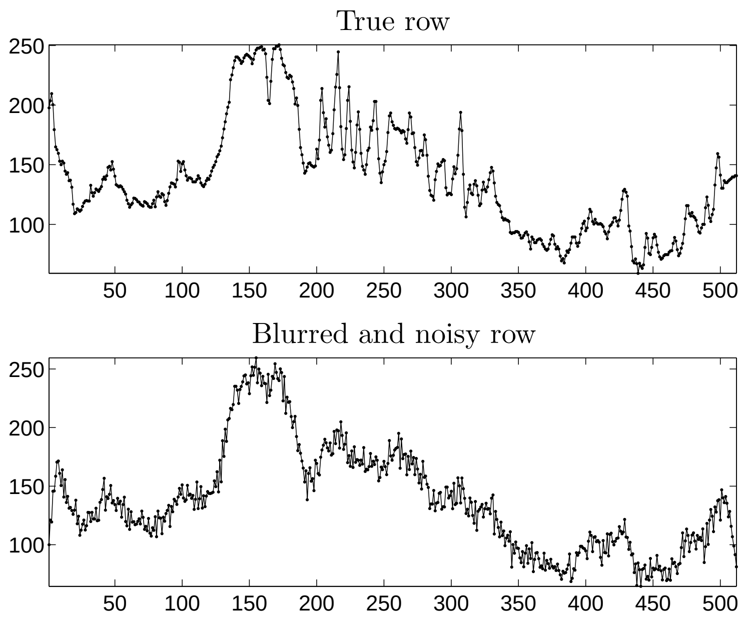


Figure 5.10: *One-dimensional example problem.*

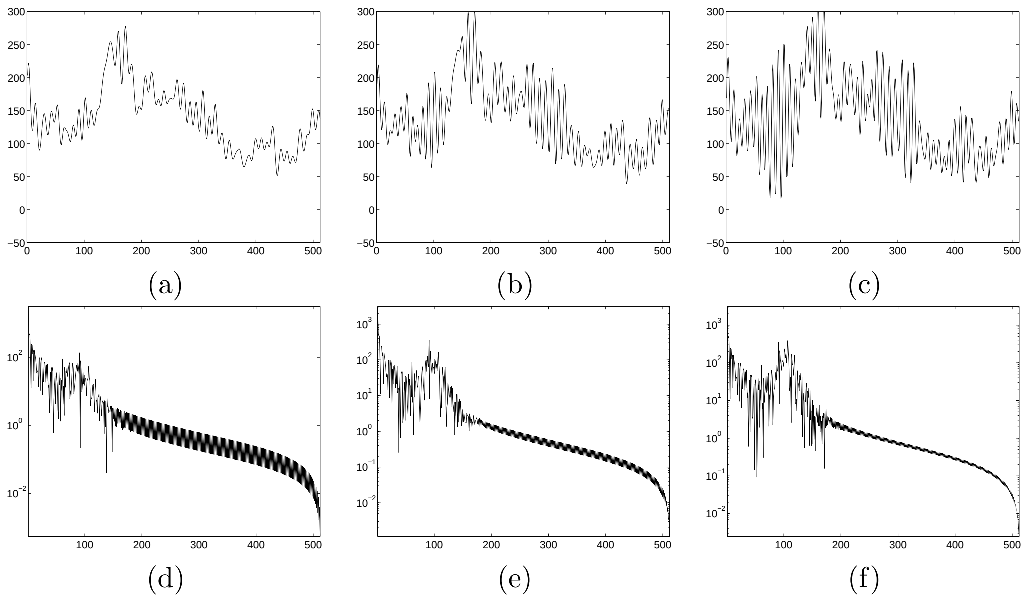
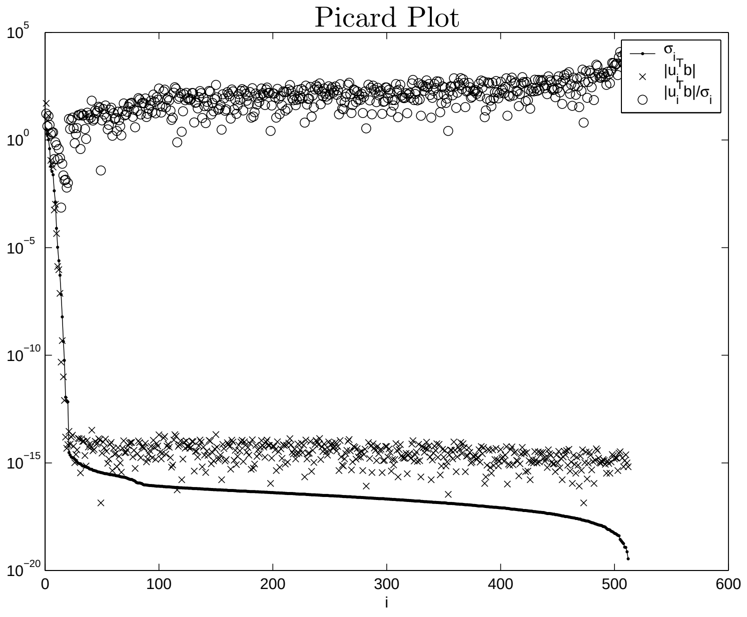
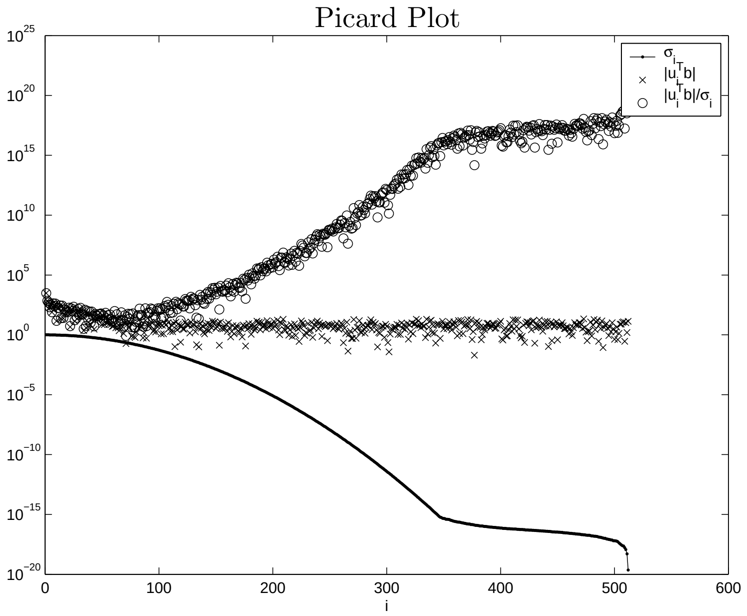


Figure 5.11: *(a), (b), and (c) show LSQR regularized solutions to the one-dimensional problem after 10, 20 and 30 iterations. (d), (e), and (f) show the DCT transformed frequency domain of the solutions.*



(a)



(b)

Figure 5.12: *(a) Picard plots of standard one dimensional test problem* shaw*. (b) Picard plot of one dimensional penguin problem.*

rapidly to zero, whereas the decay in the other case is much slower. As a result, the transition between good and bad solution coefficients $|\mathbf{u}_i^T\mathbf{b}|$ is less pronounced in case of the one-dimensional penguin problem than in shaw.

Studying the LSQR solution, it is possible to theoretically describe what happens as the regularized solutions can be described by the Ritz polynomial, $\mathcal{R}_k(\theta)$,

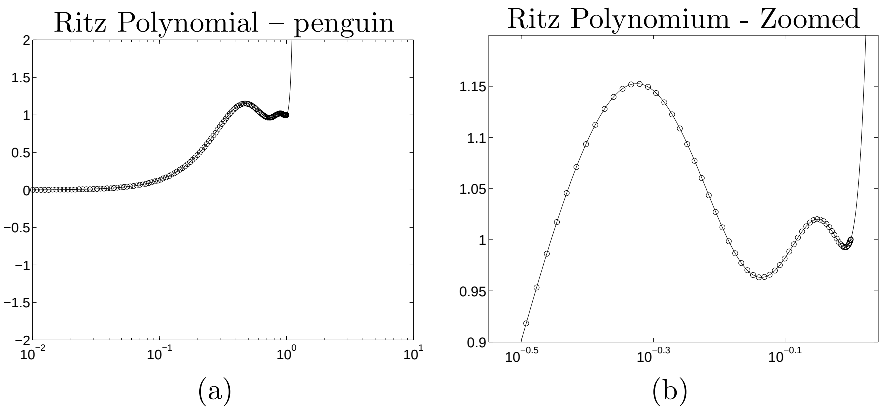


Figure 5.13: *The polynomial (4.41) plotted for a continuous variable σ (solid line) after 5 LSQR iterations with the filter factors corresponding to the singular values of* **A** *shown as circles. (a) Shows the whole polynomium, and (b) shows a zoom of the interesting part.*

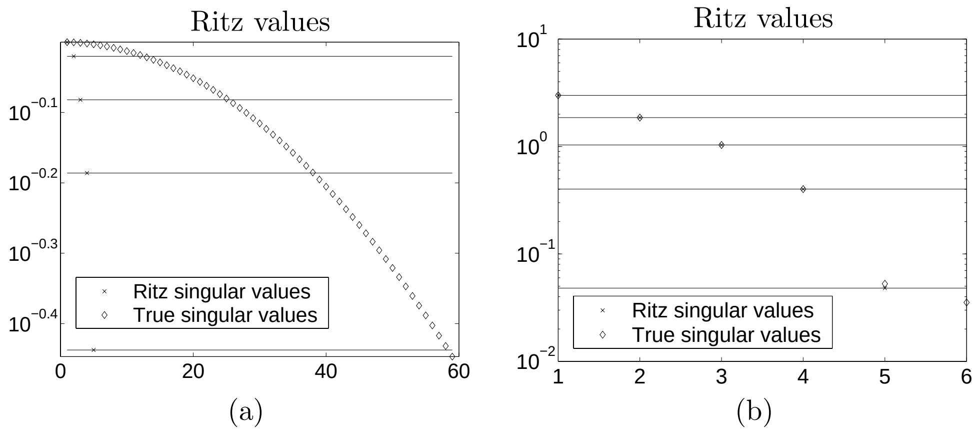


Figure 5.14: *(a) Approximated and true singular values for the one dimensional penguin problem. (b) Approximated and true singular values for the* shaw *test problem.*

discussed in Section 4.5.1.

Now looking at a plot of the Ritz polynomial and the filter factors for the one dimensional penguin problem, a plot corresponding to Fig. 4.3 is seen in Fig. 5.13 (a). It is observed that the Ritz polynomial in itself oscillates slowly, and that a lot of filter factors follow the oscillations. According to [8, Section 6.3.2], for a continuous parameter θ lying between the converged singular Ritz values, the polynomial will oscillate. From the plot in Fig. 5.13 (a), the conclusion is that a lot of the filter factors that oscillate around one correspond to Ritz values that have not yet converged, as no oscillations of the Ritz polynomial are observed between those filter factors.

The explanation of this behaviour of the filter factors is found by looking closer at the the Ritz polynomial in Fig. 5.13 (b) and a plot of the singular values as seen in Fig. 5.14 (a). From (4.41) it is seen that the sign of the product term must change as σ^2 changes from lying on one side of an approximate singular value to lie on the other. The number of terms in the product is equal to the number k of iterations performed, which of course is also the number of approximate singular values. This sign change of the product term is illustrated in Table 5.2 for $k = 5$. Now looking at Fig. 5.14 (a), it is seen that a lot of true singular values lie between the approximate singular values, and that only one singular value has converged. This shows that only one filter factor is reliably equal to one, whereas the remaining filter factors that oscillate around one are caused by the very slowly decaying true singular values. Comparing this with a similar plot for the shaw test problem in Fig. 5.14 (b), it is seen that here more Ritz singular values have converged. Furthermore, only one true singular value lies between $\theta_4^{(5)}$ and $\theta_5^{(5)}$, giving rise to $f_5^{(5)} > 1$ in Fig. 4.3. The very large positive and negative values seen in that figure are caused by the fact that:

$$1 - \prod_{j=1}^{k} \frac{\theta_j^{(k)} - \sigma_i}{\theta_j^{(k)}} \to \infty \quad \text{for} \quad \theta_i^{(k)} \to 0 \tag{5.7}$$

And in Fig. 5.14 it is indeed seen that the approximated singular values for the shaw test problem are much smaller than for the one dimensional penguin problem.

From this analysis, it is seen that the "freckles" observed in the image reconstruction also appear for one-dimensional problems that have slowly decaying singular values. It is also seen that some of the explanation of the "freckles" might be connected with the behaviour of the Ritz polynomial that introduces oscillations in the filter factors for the regularized solution. But as "freckles" were also observed in connection with Tikhonov regularization in Fig. 3.15, we now turn to look at the decay rate of the filter factors.

5.4 Decay of Filter Factors

Returning to two-dimensional problems, in case of simple blur, and knowing the true image **X**, we are able to calculate the filter factors for any regularized solution directly by means of (4.35). Doing so, we can evaluate the filter factors for the solutions from Fig. 5.2. To study the decay, we look again at the Ritz polynomial. Looking at (4.41) and Table 5.2, we see that the filter factors must decay monotonically for $\sigma_i \to 0$. Furthermore, as stated in [8], the filter factors for $\sigma_i^2 \ll \theta_k^{(k)}$ are given by:

$$f_i^{(k)} = \sigma_i^2 \sum_{j=1}^{k} \frac{1}{\theta_j^{(k)}} + \mathcal{O}\left(\frac{\sigma_i^4}{\theta_k^{(k)}\theta_{k-1}^{(k)}}\right) \quad \text{for} \quad \sigma_i^2 \ll \theta_k^{(k)}. \tag{5.8}$$

	$\theta_1^{(5)}$	$\theta_2^{(5)}$	$\theta_3^{(5)}$	$\theta_4^{(5)}$	$\theta_5^{(5)}$	sign $(\prod)$
$\sigma^2 > \theta_1^{(5)}$	$-$	$-$	$-$	$-$	$-$	$-$
$\theta_1^{(5)} > \sigma^2 > \theta_2^{(5)}$	$+$	$-$	$-$	$-$	$-$	$+$
$\theta_2^{(5)} > \sigma^2 > \theta_3^{(5)}$	$+$	$+$	$-$	$-$	$-$	$-$
$\theta_3^{(5)} > \sigma^2 > \theta_4^{(5)}$	$+$	$+$	$+$	$-$	$-$	$+$
$\theta_4^{(5)} > \sigma^2 > \theta_5^{(5)}$	$+$	$+$	$+$	$+$	$-$	$-$
$\theta_5^{(5)} > \sigma^2$	$+$	$+$	$+$	$+$	$+$	$+$

Table 5.2: *Illustration of sign changes of the Ritz polynomial*

The proportionality with σ_i^2 for σ_i small is seen to be similar to the Tikhonov filter factors which establishes a connection with the "freckles" seen in Fig. 3.15. We repeat here the general expression for a regularized solution, using the filter factors, ψ_i:

$$\mathbf{x}_{\text{reg}} = \sum_{i=1}^{N^2} \psi_i \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i. \tag{5.9}$$

From the discussion in Section 2.1.2, we know that the coefficients $|\mathbf{u}_i^T \mathbf{b}|$ must decay faster than the singular values σ_i. As this is not the case from the point where the noise has entered the solution, the filter factors are used to filter out these unwanted components instead. But as the filter factors in the case of Tikhonov as well as in the case of LSQR are strongly connected to the singular values of the problem, then the decay of the filter factors is slow when the decay of the singular values is slow. In Fig. 5.15, the filtered solution coefficients $(\phi_i \mathbf{u}_i^T \mathbf{b})/\sigma_i$ are shown for the Barbara test problem from Chapter 3 applying the LSQR filter factors after 10 iterations. And as seen, the filtered solution coefficients contain a notable level also a while after the filter coefficients have topped. This phenomenon causes a lot of higher frequencies to be restored with a filter factor close to one. As those higher frequencies are dominated by noise, the noise is thus included in the regularized solution.

For comparison, we also show the corresponding GMRES filter factors in Fig. 5.16. Worth to mention is that only 5 iterations have been performed to get filter factors that start to decay for approximately the same value of σ as does the LSQR filter factors after 10 iterations. It is also seen that the filter factors seem to fall

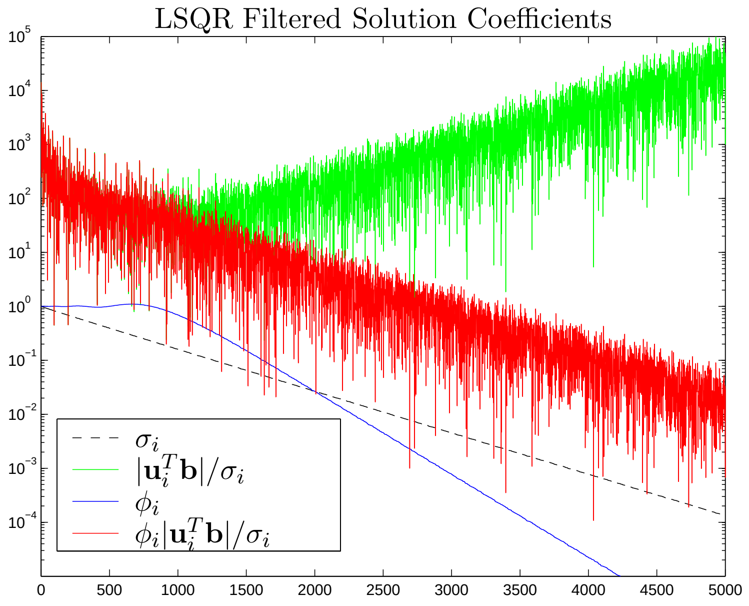


Figure 5.15: *Zoom of part of Picard plot for Barbara test problem showing the singular values σ_i as dashed black line, the filter factors as a simple solid blue line, the solution coefficients $|\mathbf{u}_i^T \mathbf{b}|/\sigma_i$, that rise in the end caused by the noise entering, in green, and the filtered solution coefficients $\phi_i |\mathbf{u}_i^T \mathbf{b}|/\sigma_i$ in red. The LSQR filter factors after 10 iterations are used.*

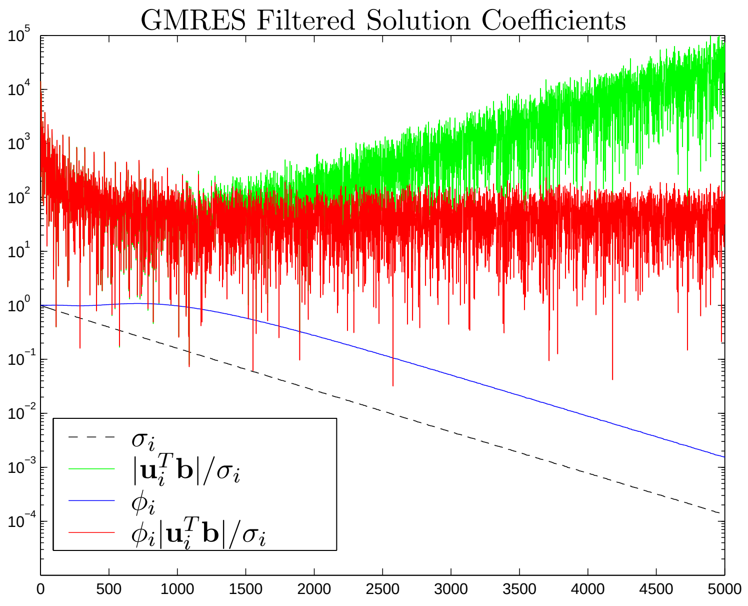


Figure 5.16: *The plot is similar to the one in Fig. 5.15, but here the GMRES filter factors after 5 iterations are used.*

off proportional to σ_i, and not σ_i^2. This in turn means, that the amplification of the solution coefficients, caused by the decay of the singular values, is only just balanced by the GMRES filter factors. And as seen from the plot, the filtered solution coefficients stay constant after the point where the filter factors start to fall off. This is of course connected with the analysis of the Krylov subspaces from Section 4.4 and 5.2, where the Krylov vectors are damped differently for the two algorithms by $\mathbf{\Sigma}^2$ and $\mathbf{\Sigma}$, respectively.

5.5 Gained Insight

From the studies in this chapter, we have gained a lot of insight into how regularization of images work in practise. First of all, the problems are large, and the transition between wanted, informative solution coefficients, and bad solution coefficients contaminated by noise, is slow and covers a large amount of singular vectors. We have seen that the choice of Krylov subspace, through the choice of the method, have a great impact on the properties of the solutions obtained. Those properties can be studied by looking directly at the SVD filter factors for the solutions. Determining what solution is actually the best, is a hard choice – especially when considering the work involved with each method.

The GMRES algorithm needs only around half the number of the iterations, compared to the LSQR algorithm, to reach the same number of SVD components. Furthermore, only $\mathbf{A}$ and not the transpose is needed. But on the other hand, the damping in the filter factors is so weak that high-frequent noise enters the solutions. The LSQR algorithm is slower, but the damping in the filter factors is better. The problem with the LSQR filter factors is that the “freckles” appear, as the most low-frequent part of the noise, just above the wanted solution coefficients, is included in the regularized solution. Comparing with the brute force TSVD solutions from Section 3.6, it seems that we have the “freckles” better controlled when truncating the SVD components, but on the other hand the wanted information that after all exists also in the first components contaminated by noise, is also cut off. This leads to a less detailed solution, but without “freckles”. Summarizing all in short, we have:

GMRES: A lot of SVD components are included in few iterations, but a lot of high-frequent noise appears due to the very slow decay of the filter factors and the noise included in the Krylov subspace.

LSQR: Fewer SVD components are included, but still many more than the number of iterations. The noise enters the solution as “freckles” as the filter factors fall off quite slowly and the first part of the noisy SVD components are partly included.

TSVD: Less details are included, but the “freckles” are better controlled as the truncation of the SVD components is complete from one point onwards.

Chapter 6

Modifications of Iterative Algorithms

We have seen that none of the discussed algorithms are optimal. The LSQR algorithm quite fast introduces unwanted "freckles", and the GMRES algorithm introduces quite a lot of noise – especially if the noise level in the blurred right-hand side is not very small. But there are a number of ways we might improve the algorithms, of which the following will be discussed in this chapter.

Inner Regularization has been proposed in many different shapes in the literature. Basically this approach covers the methods known as *Hybrid methods* e.g. mentioned in [8, Section 6.6].

Preconditioning is a means for changing the condition number, or the clustering of the singular values of the blurring matrix, making the system easier to solve.

Working in other Krylov subspaces, which is a very general formulation. Different algorithms, giving rise to other filter factors than the LSQR and GMRES filter factors could be proposed.

Each of the three might change the basic properties of the solutions, and hopefully provide an insight into how better solutions can be obtained.

6.1 Inner Regularization

As seen from Fig. 4.2, the SVD analysis of the vectors constructing the GMRES solution shows that the noise contribution is primarily contained in the last vector. As the Arnoldi process in Algorithm 4.2 has to store all the vectors $\mathbf{W}_k$ as well as the Hessenberg matrix $\mathbf{H}_k$, doing direct calculations with those is possible. Furthermore, the size of $\mathbf{H}_k$ is small, only being $(k+1) \times k$ after k iterations, thus doing decompositions of this matrix is quite inexpensive compared to the rest of the algorithm. Having computed the SVD of $\mathbf{H}_k$, we can now apply regularization to the inner problem by doing either:

TSVD on the inner problem, cutting off the last noisy vector, or maybe better some of the last vectors.

Tikhonov on the inner problem, choosing an appropriate λ to filter the SVD components.

The basic algorithm for doing GMRES iterations with inner regularization is given in Algorithm 6.1.

Algorithm 6.1 *GMRES with Inner Regularization*
$\mathbf{x}_{\text{reg}} = \textsc{Gmres}_{\text{INT}}(\mathbf{A}, \mathbf{b}, k)$
Given $\mathbf{A}$*,* $\mathbf{b}$*, and the parameter* k*, the algorithm returns the regularized solution from the Krylov subspace* $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$*:*

```
$\beta = \|\mathbf{b}\|_2$
$\mathbf{w} = \beta^{-1}\mathbf{b}$
$[\mathbf{W}_{k+1}, \mathbf{H}_k] = \textsc{Arnoldi}(\mathbf{A}, \mathbf{w}, k)$
    $\mathbf{x}_{\text{inner}} = \textsc{Inner}(\mathbf{H}_k, \mathbf{W}_k^T\mathbf{b})$
    $\mathbf{x}_{\text{reg}} = \mathbf{W}_k\mathbf{x}_{\text{inner}}$
```

In the algorithm Inner, we can apply either TSVD or Tikhonov regularization, or in general any other filtering of the SVD components provided. Having the SVD of $\mathbf{H}_k$, as given in (4.20):

$$\mathbf{H}_k = \widehat{\mathbf{P}}\widehat{\mathbf{\Gamma}}\widehat{\mathbf{Q}}^T, \tag{6.1}$$

the inner regularized solution can be defined like in (2.15) as:

$$\mathbf{x}_{\text{inner}} = \widehat{\mathbf{Q}}\widehat{\mathbf{\Psi}}\widehat{\mathbf{\Gamma}}^{-1}\widehat{\mathbf{P}}^T(\mathbf{W}_k^T\mathbf{b}), \tag{6.2}$$

where $\mathbf{W}_k^T\mathbf{b}$ is the given right-hand side, transformed to the inner problem, and $\widehat{\mathbf{\Psi}}$ is a diagonal matrix with the filter factors corresponding to the regularization method of choice. The solution to the outer problem is then obtained by transforming the regularized inner solution back by using $\mathbf{W}_k$, which yields:

$$\begin{aligned}
\mathbf{x}_{\text{reg}} &= \mathbf{W}_k\mathbf{x}_{\text{inner}} \\
&= \mathbf{W}_k\widehat{\mathbf{Q}}\widehat{\mathbf{\Psi}}\widehat{\mathbf{\Gamma}}^{-1}\widehat{\mathbf{P}}^T(\mathbf{W}_k^T\mathbf{b}) \\
&= \sum_{i=1}^{k} \hat{\psi}_i \frac{(\mathbf{W}_k\widehat{\mathbf{P}})_{[:,i]}^T\mathbf{b}}{\hat{\gamma}_i}(\mathbf{W}_k\widehat{\mathbf{Q}})_{[:,i]} \qquad (6.3) \\
&= \sum_{i=1}^{k} \hat{\psi}_i \frac{\widehat{\mathbf{P}}_{[1,i]}\beta}{\hat{\gamma}_i}(\mathbf{W}_k\widehat{\mathbf{Q}})_{[:,i]}, \qquad (6.4)
\end{aligned}$$

where the last derivation is similar to (4.24), with β being the two-norm of $\mathbf{b}$ (recall that $\beta^{-1}\mathbf{b}$ is chosen to be the first vector in the orthonormal basis $\mathbf{W}_k$. Thus regularizing internally is easy – first we regularize the small problem, and then we transform this back by using $\mathbf{W}_k$. We just need the inner regularization parameter.

6.1.1 Regularization Parameters

Both TSVD and Tikhonov need a parameter to work. The TSVD needs the truncation parameter, k, and Tikhonov needs the regularization parameter, λ. Providing global regularization parameters, we would need to analyze the problem beforehand, which for large problems will be just as cumbersome as solving the system directly. Also, the small problem grows dynamically with the iterations k, which means that the optimal internal regularization parameter changes. Thus we need a method for the algorithm to choose the regularization parameters dynamically. As we have the full SVD of the small problem, a number of well-studied methods are available:

- Discrepancy Principle [8, Section 7.2]
- Generalized Cross Validation (GCV) [8, Section 7.4]
- L-curve Criterion [8, Section 7.5], [6]

The Discrepancy Principle requires knowledge, or an estimate, of the size of the perturbation noise $\|e\|_2$ in the noisy right-hand side. Then the method suggests to use a regularization parameter λ for which:

$$\|\mathbf{A}\mathbf{x}_\lambda - \mathbf{b}\|_2 = \delta_e\,, \quad \text{where} \quad \|\mathbf{e}\|_2 \le \delta_e. \tag{6.5}$$

In the case where the regularization parameter is discrete, the smallest k fulfilling $\|\mathbf{A}\mathbf{x}_k - \mathbf{b}\|_2 \le \delta_e$ should be chosen. This seems fair, as we cannot expect to extract more information when the norm of the residual gets smaller than the norm of the noise in the measured data. But it has the obvious disadvantage that it requires knowledge about the size of the error, which is seldom available. Unfortunately, examples show that estimating this noise level is difficult, and furthermore that a wrong estimate can lead to very bad regularized solutions.

The Generalized Cross Validation requires no knowledge about the size of the noise, and builds upon statistical considerations. The idea is to minimize the error $\|\mathbf{A}\mathbf{x}_\lambda - \bar{\mathbf{b}}\|_2$. But as $\bar{\mathbf{b}}$ is unknown, the following function is used:

$$\mathcal{G}(\lambda) = \frac{\|\mathbf{A}\mathbf{x}_\lambda - \mathbf{b}\|_2^2}{\operatorname{trace}\left(\mathbf{I}_m - \mathbf{A}\mathbf{A}^{\#}\right)^2}, \tag{6.6}$$

where $\mathbf{A}^{\#}$ is the regularized inverse of $\mathbf{A}$, giving rise to the regularized solution $\mathbf{x}_{\text{reg}}$, so $\mathbf{x}_{\text{reg}} = \mathbf{A}^{\#}\mathbf{b}$. The regularization parameter is then chosen to be the minimizer of the GCV function:

$$\lambda_{\text{opt}} = \underset{\lambda}{\operatorname{argmin}}\ \mathcal{G}(\lambda). \tag{6.7}$$

The L-curve is the last generally used method for finding the regularization parameter, and here the idea is to balance some norm of the residual and some norm of the the regularized solution. As more SVD components are included in the solution, the norm of the residual reduces, but the norm of the solution grows. At some point, the noise enters the solution, and the size of the solution increases a lot, which is clearly seen to be the case for the GMRES basis vectors in the small test problem in Fig. 4.2. The transition between a rapid decrease of the residual and a

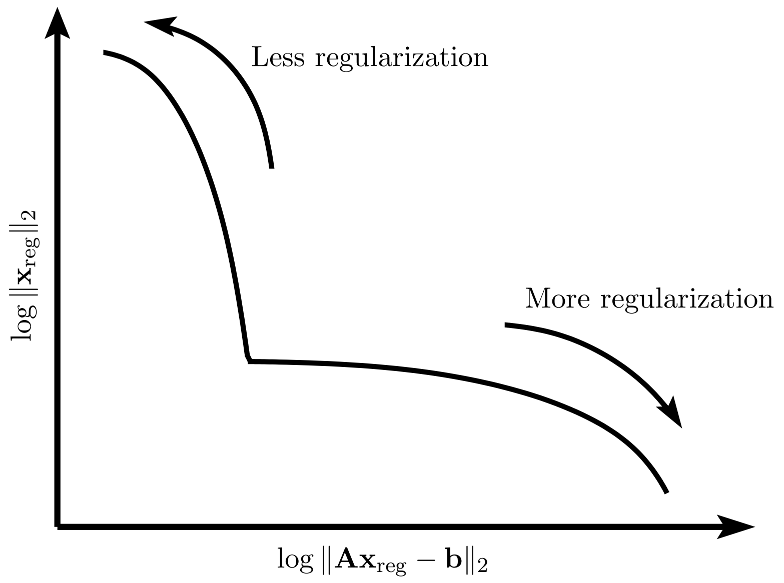

Less regularization
More regularization
$\log \|\mathbf{x}_{\text{reg}}\|_2$
$\log \|\mathbf{A}\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2$

Figure 6.1: *Generic L-Curve showing the theoretical corner arising in the transition from a fast decrease of the residual to a fast increase of the solution norm.*

rapid increase of the solution defines, in theory, the corner on the L-curve, where the L-curve is defined as:

$$(\zeta(\lambda), \eta(\lambda)) = (\log \|\mathbf{A}\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2, \log \Omega(\mathbf{x}_{\text{reg}})) , \tag{6.8}$$

with $\Omega(\mathbf{x}_{\text{reg}})$ being some measure of the size of the regularized solution. Here is simply used that $\Omega(\mathbf{x}_{\text{reg}}) = \|\mathbf{x}_{\text{reg}}\|_2$. A generic L-Curve is seen in Fig. 6.1, showing how the theoretical corner of the L-Curve is located as a compromise between a low residual and a low solution norm.

6.1.2 Results Using Inner Regularization

We try to apply different combinations of inner regularization methods and parameter choice algorithms to the GMRES solution after 10 iteration on the penguin image as seen in Fig. 5.2 (e). The combinations are shown in Table 6.1 together with the two-norm of the true error, and the results are seen in Fig. 6.2.

It is observed that all the attempts to regularize the inner problem give results as expected as the noise from the pure GMRES solution is more or less removed. Interesting enough is that the "freckles" are seen to appear in the solutions – especially the solution regularized internally by Tikhonov. This suggests that underlying

	TSVD	Tikhonov
GCV	(a)8.5426e+03	(b)7.7804e+03
L-Curve	(c)7.4892e+03	(d)1.0683e+04

Table 6.1: *Tests of inner regularization showing the two-norm of the true error for the solutions visualized in Fig. 6.2.*

the noisy GMRES solution is all the information from the LSQR or Tikhonov style solutions. It is also seen that the additional problem of introducing an inner regularization parameter, that must be chosen dynamically, gives well-known problems of under and over estimation of the regularization parameter for the GCV function and the L-Curve, which in practise adds another level of complexity to the study of the obtained solutions.

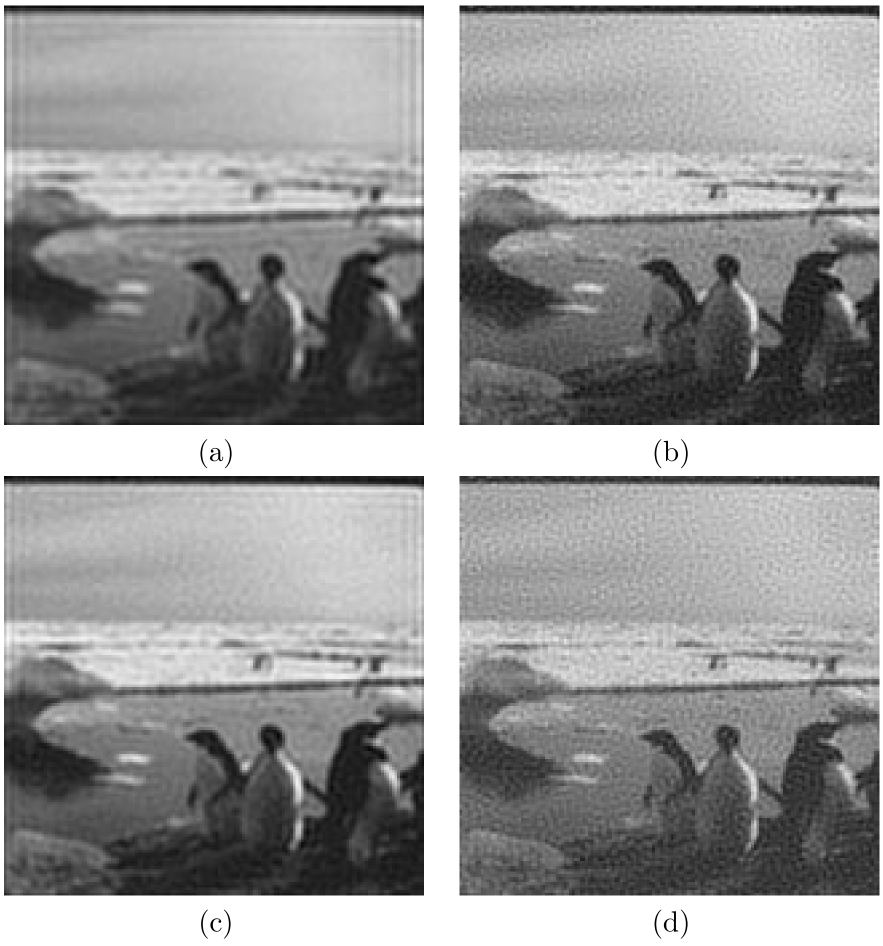


Figure 6.2: *GMRES solutions after 10 iterations with inner regularization applied. The regularization applied follows Table 6.1.*

This, unfortunately, also means that one cannot expect the norm of the solution to increase monotonically, and not either the norm of the residual to decrease monotonically. For each new step of the GMRES algorithm, the new internally regularized solution changes. On the other hand, we must expect the regularized solution always to be fairly good, even when the GMRES is run too far. Fig. 6.3 shows the norm of the error to the true solution $\|\mathbf{x} - \mathbf{x}_{\text{reg}}^{(k)}\|_2$ when running GMRES with inner Tikhonov regularization, using the GCV function to choose the regularization parameter. As seen, the norm stabilizes, and the solutions do not change dramatically after a certain point where all wanted information has been extracted by the Krylov subspace. After a certain number of iterations, the additionally included vectors are all too noisy, and they are filtered away again by the inner regularization.

Implementation in Matlab

The inner regularization has been implemented in the basic GMRES algorithm in various forms. The final implementation, inspired by the work done in this project, has now been included in the Regularization Tools XP [12]. The code is seen in Appendix A.

6.2 Preconditioning

Another way of enhancing the quality of the solutions is by means of preconditioning. The idea of preconditioning is to speed up the convergence, and force the convergence towards a wanted solution.

If the matrix $\mathbf{A}$ had been well-conditioned, the solution of the system would have been simple. Especially in case of $\mathbf{A}$ being the identity matrix $\mathbf{I}$, we would not need to solve the system at all, as the right-hand side would be the solution. So if somehow the system could be changed in such a way, that $\mathbf{A}$ approximates the identity, or at least a matrix better conditioned, solving the system would be easier.

In principle what is done to modify the given problem so that the blurring matrix has some nicer properties is to introduce a matrix $\mathbf{C}$ in the following way:

$$\mathbf{A}\mathbf{C}^{-1}\mathbf{y} = \mathbf{b} \tag{6.9}$$

$$\mathbf{C}\mathbf{x} = \mathbf{y} \tag{6.10}$$

where $\mathbf{C}$ is constructed such that it resembles $\mathbf{A}$. Another way of defining the preconditioner is to transform the system to:

$$\widetilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \tag{6.11}$$

where $\widetilde{\mathbf{A}} = \widetilde{\mathbf{C}}^{-1}\mathbf{A}\widetilde{\mathbf{C}}^{-1}$, $\tilde{\mathbf{x}} = \widetilde{\mathbf{C}}\mathbf{x}$, and $\tilde{\mathbf{b}} = \widetilde{\mathbf{C}}^{-1}\mathbf{b}$. This approach is especially used in connection with preconditioned Conjugate Gradients [4, Section 10.3], that is

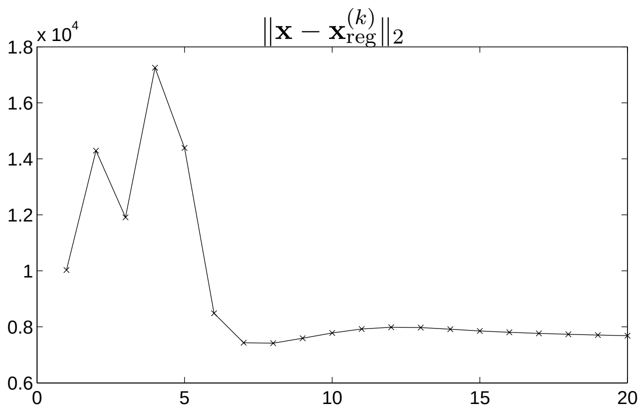


Figure 6.3: *Plot showing $\|\mathbf{x} - \mathbf{x}_{\mathrm{reg}}^{(k)}\|_2$ for the GMRES solution with inner Tikhonov regularization, using the GCV function to choose the regularization parameter.*

defined for symmetric positive definite systems. Defining the preconditioned system as in (6.11) assures that the transformed system is also positive definite, if the preconditioner $\widetilde{\mathbf{C}}$ is positive definite. In our case, we do not assume that the system is positive definite, and we use the preconditioner $\mathbf{C}$ from (6.9) and (6.10).

If again $\mathbf{A}$ had been well-conditioned, then choosing $\mathbf{C} = \mathbf{A}$ would be a clever choice, resulting in the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$, where $\mathbf{y} = \mathbf{b}$ as $\mathbf{AC}^{-1} = \mathbf{I}$. But when $\mathbf{A}$ is not well conditioned, some problems arise. If $\mathbf{C}$ is too close to $\mathbf{A}$, this matrix will be ill-conditioned as well – and solving the system (6.10) yields additional problems. If on the other hand $\mathbf{C}$ is too far from $\mathbf{A}$, the preconditioning will not work either, as solving (6.10) in this case has no relation to solving the real problem.

Even though $\mathbf{C}$ does not make the condition number of $\mathbf{AC}^{-1}$ any better, preconditioning might work anyway if the clustering of the singular values of the resulting matrix $\mathbf{AC}^{-1}$ is better than for the original matrix $\mathbf{A}$. In this connection better will mean that the singular values corresponding to the wanted parts of the solution will be largest and clustered and kept away from the singular values corresponding to the noise. When running any of the iterative algorithms, the largest singular values are approximated first. And if we make sure, with the matrix $\mathbf{C}$, to change the system in such a way that all wanted information components have large singular values and the unwanted are either left unchanged or even damped, then we would expect fast convergence – and even convergence to the wanted solution.

Several ways to choose a proper preconditioner for discrete ill-posed problems have been discussed in the literature. E.g. in [16] by J. G. Nagy and R. J. Plemmons. Here $\mathbf{C}$ is constructed as a circulant matrix and in such a way that the singular values of $\mathbf{AC}^{-1}$ hopefully select out the signal part of the solution and leave out the noisy components.

This circulant preconditioner has been implemented by J. G. Nagy in the Matlab package RestoreTools [14], and the implementation is used as well in the Regularization Tools XP [12] by M. Jacobsen. This is the preconditioner studied here.

6.2.1 Constructing the Preconditioner

Following [16], we want to find and use a circulant matrix $\mathbf{C}$ as preconditioner to the system. The basis is the optimal circulant approximation to $\mathbf{A}$, defined by T. Chan in [3]:

$$\mathbf{C} = \underset{\mathbf{X}}{\operatorname{argmin}} \; \|\mathbf{A} - \mathbf{X}\|_F, \tag{6.12}$$

where $\mathbf{X}$ is any circulant matrix. The matrices $\mathbf{A}$ used here in the spatially invariant case are of the type Block Toeplitz with Toeplitz Blocks (BTTB), and the optimal preconditioner in this case is Block Circulant with Circulant Blocks (BCCB):

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_0 & \mathbf{C}_{-1} & \dots & \mathbf{C}_{-n+1} \\ \mathbf{C}_1 & \mathbf{C}_0 & \dots & \mathbf{C}_{-n+2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{n-1} & \mathbf{C}_{n-2} & \dots & \mathbf{C}_0 \end{bmatrix}, \tag{6.13}$$

where the $\mathbf{C}_i$'s are circulant approximations to the matrices from the Kronecker product in Definition 3.3. Circulant preconditioners simplify the calculations needed for solving the preconditioned system, as a circulant matrix has the eigenvalue decomposition:

$$\mathbf{C} = \mathcal{F}^H \mathbf{\Lambda} \mathcal{F}, \tag{6.14}$$

where $\mathcal{F}$ denotes the Fourier basis. This means that any circulant matrix is completely defined by its eigenvalues, and that computations can be done using the Fast Fourier Transform. Furthermore, the matrix being circulant means that only the first vector needs to be stored.

As noted, the matrix $\mathbf{C}$ being too close to $\mathbf{A}$ leads to $\mathbf{C}$ being ill-conditioned as well. This problem is dealt with by modifying the eigenvalues of $\mathbf{C}$, leaving only the ones corresponding to the signal subspace unchanged, and setting the remaining to one. This modification depends on some parameter τ, that must be chosen at beforehand, selecting the wanted parts of the spectrum. The choice of τ is crucial to the performance of the preconditioner.

6.2.2 Implementation in GMRES and Results

Preconditioning can be implemented in many iterative algorithms, and is often needed to make e.g. Conjugent Gradient work properly. Here it is implemented in the GMRES algorithm, which is an easy task. The preconditioned GMRES algorithm is seen in Algorithm 6.2.

Algorithm 6.2 *Preconditioned GMRES:*
$\mathbf{x}_{\text{reg}} = \text{GMRES}_{\text{INT}}(\mathbf{A}, \mathbf{b}, k, \mathbf{C})$
Given $\mathbf{A}$ *and* $\mathbf{b}$*, the number of iterations* k *as well as the preconditioner* $\mathbf{C}$*, the algorithm returns the preconditioned regularized solution from the Krylov subspace* $\mathcal{K}_k(\mathbf{A}\mathbf{C}^{-1}, \mathbf{b})$*:*

```
$\beta = \|\mathbf{b}\|_2$
$\mathbf{w} = \beta^{-1}\mathbf{b}$
FOR $i = 1, 2, 3, \dots, k$
    $\mathbf{z}_i = \mathbf{C}^{-1}\mathbf{w}_i$
    $\mathbf{v}_i = \mathbf{A}\mathbf{z}_i$
    FOR $j = 1$ TO $i$
        $h_{j,i} = \mathbf{w_j}^H \mathbf{v}$
        $\mathbf{v} = \mathbf{v} - h_{j,i}\mathbf{w_j}$
    $h_{i+1,i} = \|\mathbf{v}\|_2$
    $\mathbf{w_{i+1}} = h_{n+1,i}^{-1}\mathbf{v}$
$\mathbf{x}_{\text{reg}} = \mathbf{C}^{-1}\mathbf{V}_k\mathbf{y}_k$, where $\mathbf{y}_k$ is found as in Eqn. (4.26)
```

The algorithm is seen to be a slight modification of the Arnoldi process from Algorithm 4.2, where the preconditioning affects one line of code in the Arnoldi part and one when constructing the solution. In the Arnoldi part, the preconditioning is applied to all new Krylov vectors, and when constructing the solution from the found $\mathbf{y}_k$ that minimizes $\|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2$ as seen in (4.26), the preconditioner is again used to get the regularized solution $\mathbf{x}_{\text{reg}}$. As mentioned in [20], it is even possible to

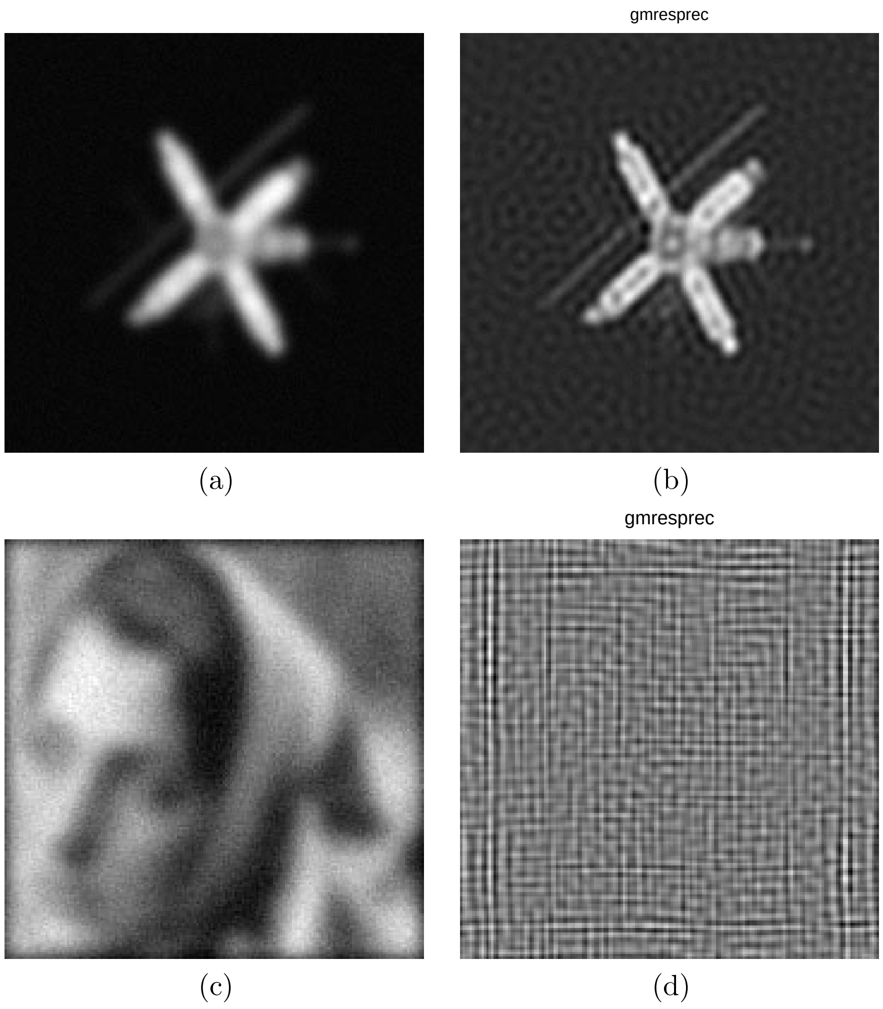


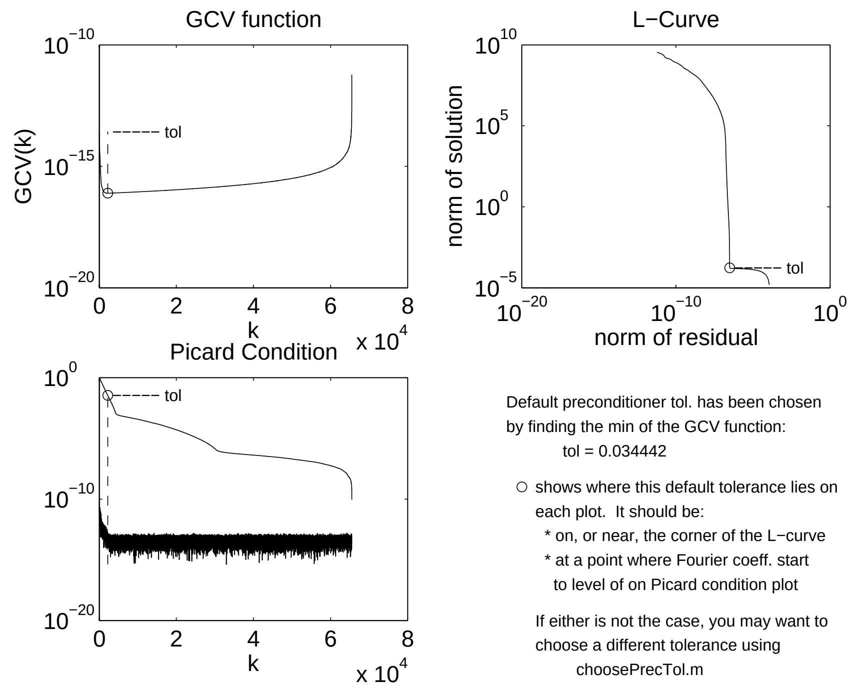
Figure 6.4: *(a) Shows a blurred artificial satellite and (b) is the preconditioned GMRES solution after only one iteration. (c) and (d) show the similar test for the Barbara image, used earlier.*

implement GMRES with variable preconditioning, in which case it is only needed to store all the $\mathbf{z}_i$'s generated in Algorithm 6.2. The last line in the algorithm could then be rewritten as:

$$\begin{aligned} \mathbf{x}_{\text{reg}} &= \mathbf{C}^{-1}\mathbf{V}_k\mathbf{y}_k \\ &= \mathbf{Z}_k\mathbf{y}_k. \end{aligned} \tag{6.15}$$

One could then apply different preconditioners $\mathbf{C}_i$ when generating the Krylov subspace, affecting different vectors in the matrix $\mathbf{Z}_k$.

In theory, preconditioning is beautiful – and certainly an idea to follow up. But in practise to find a good preconditioner is difficult. For instance finding the best truncation parameter τ for the truncation of the eigenvalues of the circulant preconditioner described. To illustrate the difficulties, a blurred and noisy artificial image of a satellite is seen in Fig. 6.4 (a). The true image is shown in Appendix B in Fig. B.5. J. G. Nagy's standard truncation parameter, found by means of the



Default preconditioner tol. has been chosen
by finding the min of the GCV function:
tol = 0.034442

O shows where this default tolerance lies on
each plot. It should be:
* on, or near, the corner of the L−curve
* at a point where Fourier coeff. start
to level of on Picard condition plot

If either is not the case, you may want to
choose a different tolerance using
choosePrecTol.m

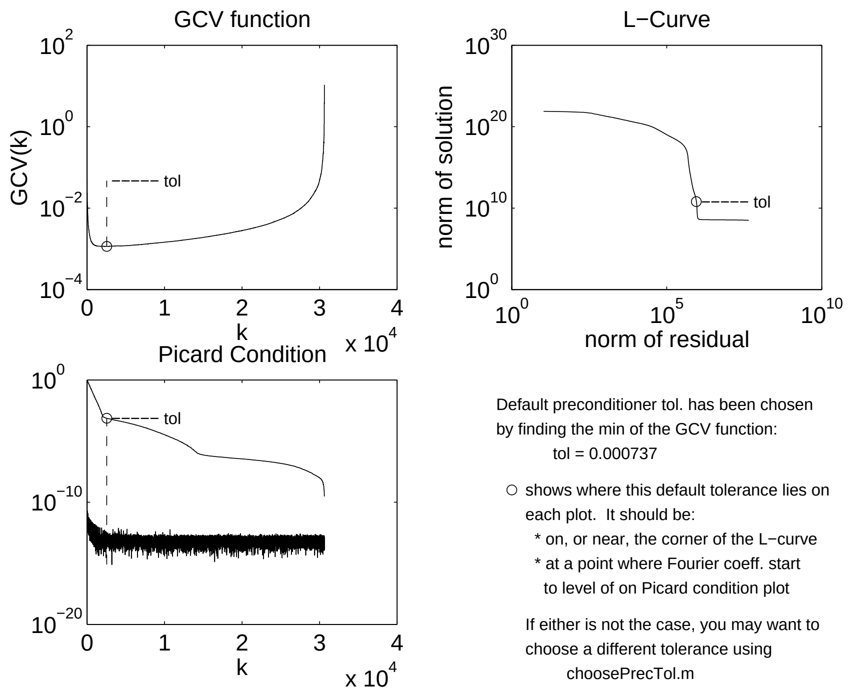
Figure 6.5: *Illustration of the truncation parameter τ found for preconditioner to the Satellite example.*

GCV function, is used, and the result of running one iteration of Algorithm 6.2 is seen in Fig. 6.4 (b). This amounts to just solving the inner preconditioned system $\mathbf{z}_i = \mathbf{C}^{-1}\mathbf{w}_i = \beta^{-1}\mathbf{C}^{-1}\mathbf{b}$ once, and the result shows that $\mathbf{C}$ in this case resembles $\mathbf{A}$ quite well as the solution seems good – only including the now well-known "freckles".

If we on the other hand choose the Barbara image used earlier, and find the truncation parameter in the same way, the results are different, and are seen in Fig. 6.4 (c) and (d) showing the blurred image and the solution after one iteration, respectively. As observed, the preconditioned GMRES solution is nonsense. This is due to a fatally chosen truncation parameter. In general it seems that choosing this truncation parameter is difficult for natural scenes, and easier for computer graphics images, like the satellite.

Looking at the chosen truncation parameters τ from the plots generated by J. G. Nagy's code, shown in the Figs. 6.5 and 6.6, we see that in case of the Satellite, the minimum for the GCV function corresponds well with the corner of the L-curve. Moreover, the Picard condition plot shows that only very few SVD components are included. For the Barbara test problem, the situation is different. The minimum for the GCV function does not correspond to the corner of the L-curve, and more components are included as seen in the Picard condition plot. This makes the preconditioner include unwanted parts of $\mathbf{A}$, thus leading to the problems observed in Fig. 6.4 (d).

Now one could choose another parameter τ for the Barbara example, which is seen in Fig. 6.7 (a). Here a value of τ is chosen experimentally to include around 1000 SVD components, which results in a parameter lying close to the corner of the



Default preconditioner tol. has been chosen
by finding the min of the GCV function:
tol = 0.000737

○ shows where this default tolerance lies on
each plot. It should be:
* on, or near, the corner of the L−curve
* at a point where Fourier coeff. start
to level of on Picard condition plot

If either is not the case, you may want to
choose a different tolerance using
choosePrecTol.m

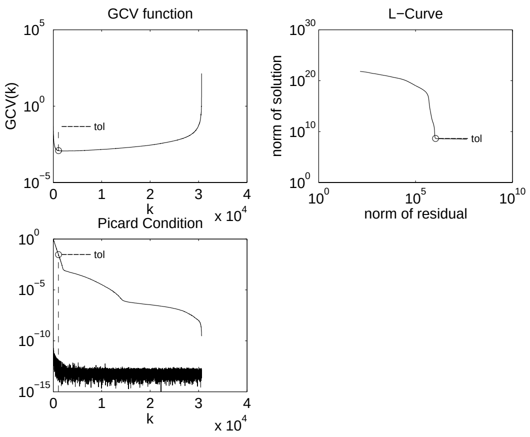
Figure 6.6: *Illustration of the truncation parameter τ found for preconditioner to the Barbara example.*

L-curve, and also corresponds well to the observations done in Section 3.6. But even this does not seem to cure the problems in choosing a good preconditioner. As seen in Fig. 6.7 (b), the solution of the preconditioner step in the first iteration of the GMRES algorithm still spoils the solution.

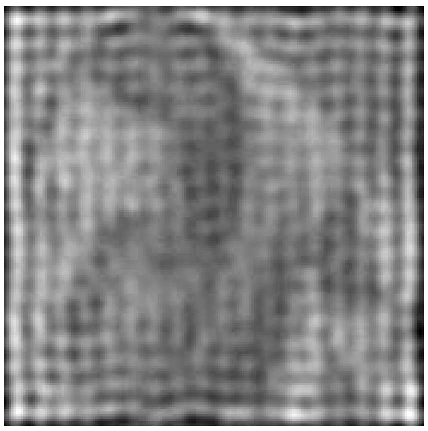
As a final issue, it could also be interesting to look at the filter factors for the preconditioned solutions. These are shown for the Satellite solution and for the best Barbara solution in Fig. 6.8. As seen, we mainly get filter factors close to one for the first part of the spectrum, and decaying filter factors for the rest, as expected. But it is also seen that the filter factors are very noisy, which shows that the behaviour of the system is changed dramatically. Indeed, it is possible for some problems to get fast convergence and a good regularized solution, but the choice of the preconditioner must be studied further in connection with two-dimensional problems and image problems in specific to obtain further conclusions.

Implementation in Matlab

The preconditioning of GMRES has been implemented in the GMRES algorithm from Regularization Tools XP, and the code is found in Appendix A. The implementation supports use of different preconditioners for each iteration. Also an attempt was made to dynamically choose the preconditioner from a set of given preconditioners to get the best reduction of the residual. But as seen above, even choosing a good preconditioner for e.g. the Barbara example, gives rise to bad solutions. Having an algorithm that supports varying preconditioners might though be

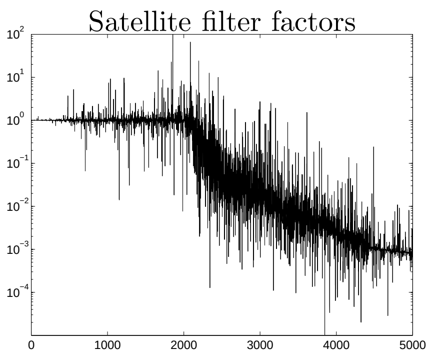


(a)

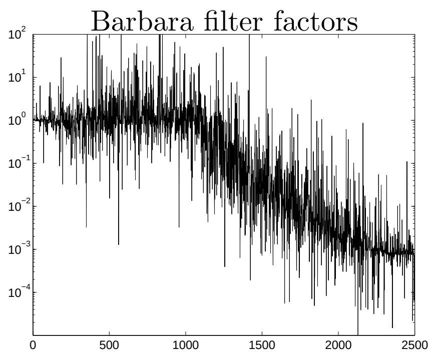


(b)

Figure 6.7: *New preconditioner used for the Barbara example.*



(a)



(b)

Figure 6.8: *Filter factors resulting from using circulant preconditioner for solving (a) the Satellite example, and (b) the Barbara example.*

a good idea as discussed by Y. Saad in [20]. As a further extension, the dynamically chosen preconditioner also gives to possibility to run an iterative algorithm at the preconditioning step in stead of defining the preconditioner explicitly. This aspect has not been further investigated.

6.3 Changing Krylov Subspace

The last modification discussed is changing the subspace the algorithms are working in. From the discussion of the subspaces used for the two algorithms in Section 4.4 and the connection with the decay of the filter factors, we might be able to better control the "freckles" if the filter factors are damped more. We study two ways of changing the Krylov subspace. First we start the LSQR algorithm differently, and second, we investigate the effect of restarting.

6.3.1 Starting in Different Subspace

The damping of the filter factors arising when using LSQR and GMRES come from the multiplication with the singular values $\mathbf{\Sigma}$ in e.g. (4.30) and (4.31). Instead of solving the given system, as does GMRES, or the normal equations, as does LSQR, we could solve the system:

$$\left(\mathbf{A}^T\mathbf{A}\right)\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)\mathbf{A}^T\mathbf{b}. \tag{6.16}$$

This would mean that we are now looking for a solution in the Krylov subspace:

$$\mathbf{x}_{\text{reg}} \in \mathcal{K}_k\left((\mathbf{A}^T\mathbf{A})(\mathbf{A}^T\mathbf{A})), \left(\mathbf{A}^T\mathbf{A}\right)\mathbf{A}^T\mathbf{b}\right), \tag{6.17}$$

which in practise can be achieved by using the standard LSQR algorithm, starting with $\left(\mathbf{A}^T\mathbf{A}\right)$ and $\mathbf{A}^T\mathbf{b}$ instead of just $\mathbf{A}$ and $\mathbf{b}$. Below, we call this approach, using the LSQR algorithm for working in the new subspace, *LSQR-new.*

Similar to the descriptions of the subspaces for the standard LSQR and GMRES algorithms in Section 4.4, we can again use the SVD of $\mathbf{A}$ to write out the subspace for the differently started LSQR-new algorithm. We get:

$$\begin{aligned} \mathbf{x}^{(k)}_{\text{LSQR-new}} &\in \operatorname{span}\left\{\mathbf{V\Sigma}^3\mathbf{U}^T\mathbf{b}, \mathbf{V\Sigma}^7\mathbf{U}^T\mathbf{b}, \ldots, \mathbf{V\Sigma}^{4k-1}\mathbf{U}^T\mathbf{b}\right\} \Rightarrow \\ \mathbf{V}^T\mathbf{x}^{(k)}_{\text{LSQR-new}} &\in \operatorname{span}\left\{\mathbf{\Sigma}^3\boldsymbol{\varphi}_u, \mathbf{\Sigma}^7\boldsymbol{\varphi}_u, \ldots, \mathbf{\Sigma}^{4k-1}\boldsymbol{\varphi}_u\right\}, \end{aligned} \tag{6.18}$$

where $\boldsymbol{\varphi}_u$ are the coefficients $\mathbf{U}^T\mathbf{b}$. Following the notation from earlier, the subspace can be evaluated easily as:

$$\begin{bmatrix} \varphi_{u[1]} & & \\ & \ddots & \\ & & \varphi_{u[n]} \end{bmatrix} \begin{bmatrix} \sigma_1^3 & & \\ & \ddots & \\ & & \sigma_n^3 \end{bmatrix} \begin{bmatrix} 1 & \sigma_1^4 & \sigma_1^8 & \ldots \sigma_1^{4k-4} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \sigma_n^4 & \sigma_n^8 & \ldots \sigma_n^{4k-4} \end{bmatrix}, \tag{6.19}$$

where the last matrix is again constructed as a Vandermonde matrix now given by the vector of σ_i^4. We see that the damping of this subspace must be twice as high as for the standard LSQR subspace, and four times higher than for the GMRES

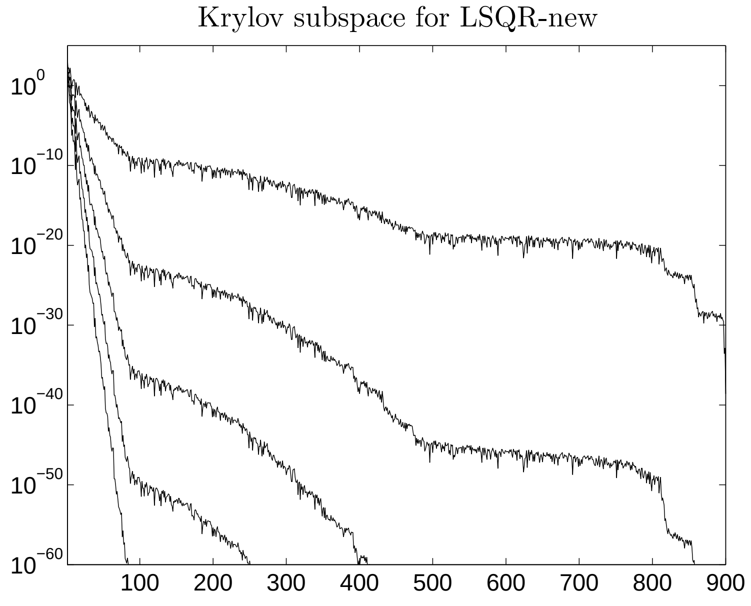


Figure 6.9: *First five Krylov vectors for the modified LSQR subspace.*

subspace, looking at the diagonal matrix $\mathbf{\Sigma}$, where the exponent goes to $4k-1$ in stead of $2k-1$ for the LSQR and only $k-1$ for the GMRES. An illustration of the first five Krylov vectors in case of symmetric blurring is seen in Fig. 6.9. This plot should of course be compared to the plots in Fig. 5.3. The idea is that with this damping, the filter factors will then lie closer to the sharp TSVD filter factors than the normal LSQR filter factors, thus being an iterative method for better controlling the "freckles".

An example showing solutions regularized by LSQR-new is seen in Fig. 6.10. The solutions are comparable to the solutions from Fig. 5.2, and show the solutions after 4, 10, and 25 iterations. As seen, the solutions do not include as many details as did the solutions from the previous chapter which indicates that the convergence is slower. This is somehow expected, making parallels to the difference between the standard LSQR and GMRES which showed that the GMRES converged approximately twice as fast as the LSQR. In this case the LSQR-new must be expected to converge twice as slow as the ordinary LSQR algorithm. But as also seen, no obvious "freckles" appear in the solution after 25 iterations.

6.3.2 Restarting

The concept of restarting is generally used in algorithms for estimating eigenvalues and singular values, like e.g. the Arnoldi algorithm or the Lanczos bidiagonalization algorithm. If the memory is not a limited resource, the algorithms could run with reorthogonalization until a wanted number of eigenvalues were found. But as all vectors in the Krylov space are needed for orthogonalizing the next vectors, the memory could be a problem. Restarting in this connection will mean to start the algorithms over again, now using the last Krylov vector as starting vector. This will,

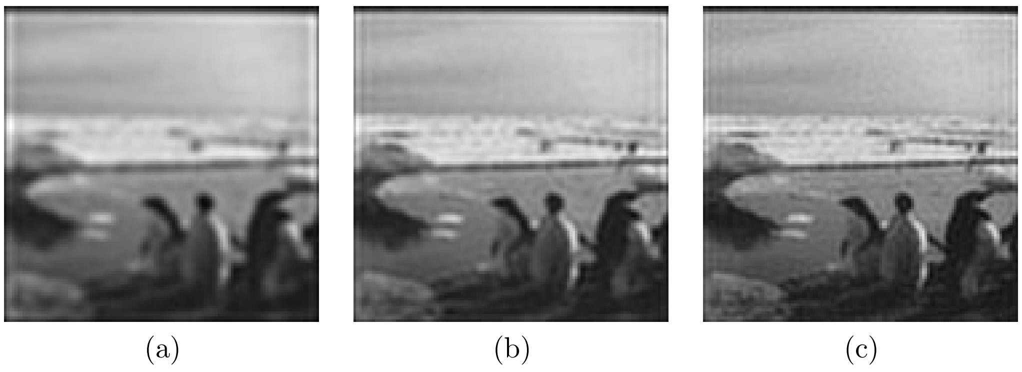


Figure 6.10: *Solutions obtained by using the LSQR-new subspace after (a) 4 iterations, (b) 10 iterations, and (c) 25 iterations.*

in case of the Arnoldi algorithm, generate a Krylov subspace as:

$$\operatorname{span}\left\{\mathbf{A}^m\mathbf{b}, \mathbf{A}^{m+1}\mathbf{b}, \ldots, \mathbf{A}^{m+k-1}\mathbf{b}\right\}, \tag{6.20}$$

when generating a subspace of dimension k, using the last Krylov vector after $m+1$ iterations as new starting vector. This directs the convergence of the eigenvalues of the Hessenberg matrix $\mathbf{H}_k$ towards the first k true eigenvalues of $\mathbf{A}$.

In this project some interest has been paid to running the Lanzcos Bidiagonalization with implicit restart, after an idea by Åke Björck [1]. Implicit restart is a method for changing the subspace implicitly, i.e. without restarting the whole algorithm from scratch with a new starting vector. This approach proves more stable in practice than explicit restart. As described in [1, pp. 523-524], to do the restart, we have to sacrifice one Krylov vector to obtain the restarted Krylov subspace. This means that if we have the Krylov subspace after k iterations:

$$\mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b}) = \operatorname{span}\left\{\mathbf{A}^T\mathbf{b}, (\mathbf{A}^T\mathbf{A})\mathbf{A}^T\mathbf{b}, \ldots, (\mathbf{A}^T\mathbf{A})^{k-1}\mathbf{b}\right\} \tag{6.21}$$

and do a single restart, then we sacrifice one vector to get the subspace:

$$\operatorname{span}\left\{(\mathbf{A}^T\mathbf{A})\mathbf{A}^T\mathbf{b}, (\mathbf{A}^T\mathbf{A})^2\mathbf{A}^T\mathbf{b}, \ldots, (\mathbf{A}^T\mathbf{A})^{k-1}\mathbf{b}\right\}, \tag{6.22}$$

which mathematically is the same as starting the bidiagonalization algorithm with the right-hand side $(\mathbf{A}^T\mathbf{A})\mathbf{A}^T\mathbf{b}$ and doing $k-1$ iterations.

Doing restarts, it is possible to capture the first k singular values very accurately with a Krylov subspace of dimension k. This gives filter factors that approximate the filter factors of the TSVD. One obvious disadvantage in connection with the image problems is the size, that from the TSVD solutions from Section 3.6 show that some 1000 singular vectors are needed to generate a satisfactory solution. In case of restarted Lanczos, we then need to generate and store a Krylov subspace of dimension 1000, which in practise is both memory and time consuming.

In Appendix A, the code for doing the Lanczos bidiagonalization with restarts is given. The code is a stripped down version of the original code by Eric Grimme for doing a specific number of iterations, followed by a specific number of restarts,

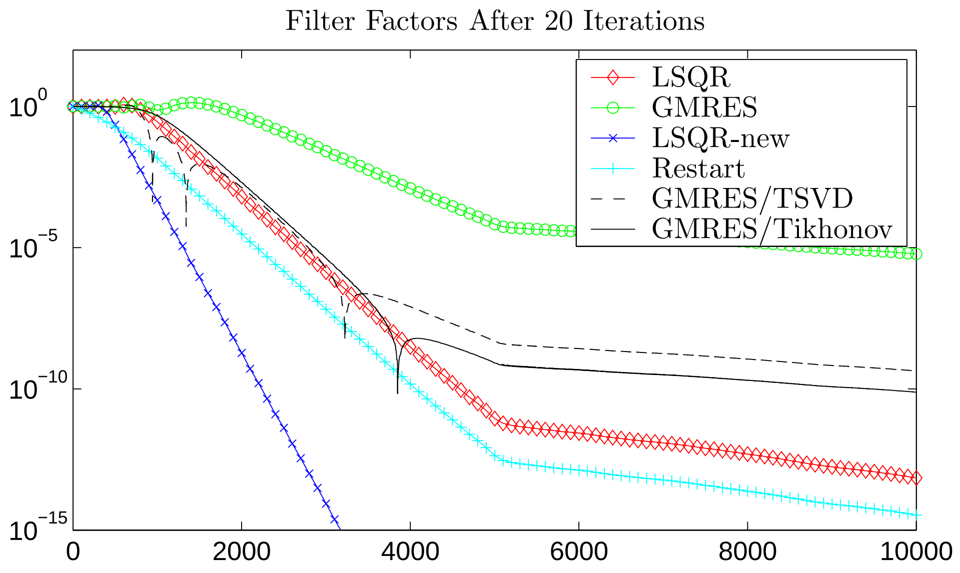


Figure 6.11: *Resulting filter factors when applying the different modified algorithms to the Barbara test image. See also Fig. 6.12.*

which makes it possible to build up a desired Krylov subspace. In the original implementation, a GCV function is used, and restarts are performed to try to make an approximate GCV function approximate the true unknown GCV function of the problem, and then stop the iterations exactly when the wanted singular components are included, leading to the TSVD style solution. This code was run in connection with a small image problem, but terminated when it, because of the huge amount of singular vectors to include, did not converge within reasonable time. The approach might still be of theoretical interest as a method for obtaining TSVD style solutions.

6.4 Resulting Filter Factors

When using one of the above modifications to the basic iterative algorithms, one obtains different filter factors as seen in Fig. 6.11.

In this figure, we see all the aspects discussed, apart from the preconditioned GMRES. These filter factors are very noisy and have been left out in this combined plot.

The filter factors for the GMRES algorithm are seen to fall off slowest, but capturing almost 2000 components in the 20 iterations. With internal regularization, the filter factors behave a lot like the filter factors for the LSQR algorithm in the fast decaying region, but fall off as standard GMRES for $i > 5000$. Here it must be noted that for the inner Tikhonov regularization, a special version of the L-Curve has been used. To make the filter factors comparable, a regularization parameter λ has been chosen, corresponding to the regularization parameter k found in case of TSVD, so the two set of filter factors correspond to choosing a regularization parameter at the same place of the L-Curve. This means that the λ for the Tikhonov filter factors does not necessarily correspond exactly to the corner of the L-Curve. It is

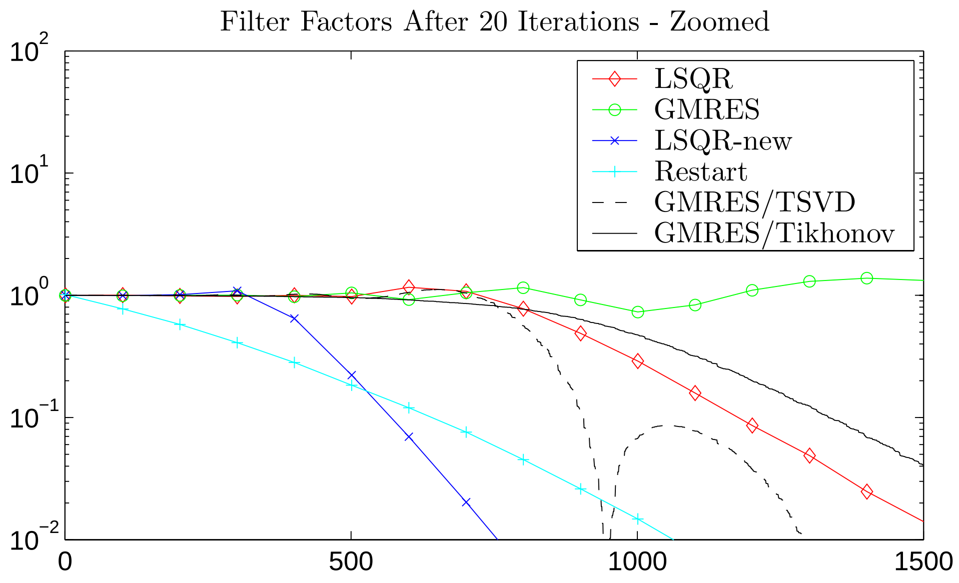


Figure 6.12: *Zoom of first part of the filter factors. Showing that GMRES with inner Tikhonov regularization has almost no oscillations.*

interesting to observe that some singular vectors seem to be removed completely by the inner regularization, and also that looking closer at the flat part of the filter factors in Fig. 6.12, it is seen that the inner Tikhonov regularization removed most of the oscillations of the filter factors.

The LSQR-new filter factors, for LSQR started in another Krylov subspace, fall off proportional to σ_i^4 – i.e. twice as fast as for the normal LSQR algorithm. On the other hand, the faster the decay of the filter factors, the more iterations are needed to capture the same number of singular values. A bit beside this picture falls the restarted bidiagonalization, for which the filter factors basically fall off like the normal LSQR algorithm, but only captures very few singular values, and therefore needs many iterations to include the same amount of information in the regularized solution. If run long enough, though, the restarted bidiagonalization would be able to select an exact amount of singular values quite accurately. The method has been run for 21 iterations, followed by one restart to create a Krylov subspace of the same dimensions as the other methods.

Finally, to conclude this chapter, the regularized solutions corresponding to the filter factors shown in the figures above is visualized in Fig. 6.13. Again, all aspects are seen. Noise, "freckles", and in case of LSQR-new and restarted LSQR, under-regularized solutions. The problem now is to choose the best method and the number of iterations to perform. This is discussed in the next chapter about stopping criteria.

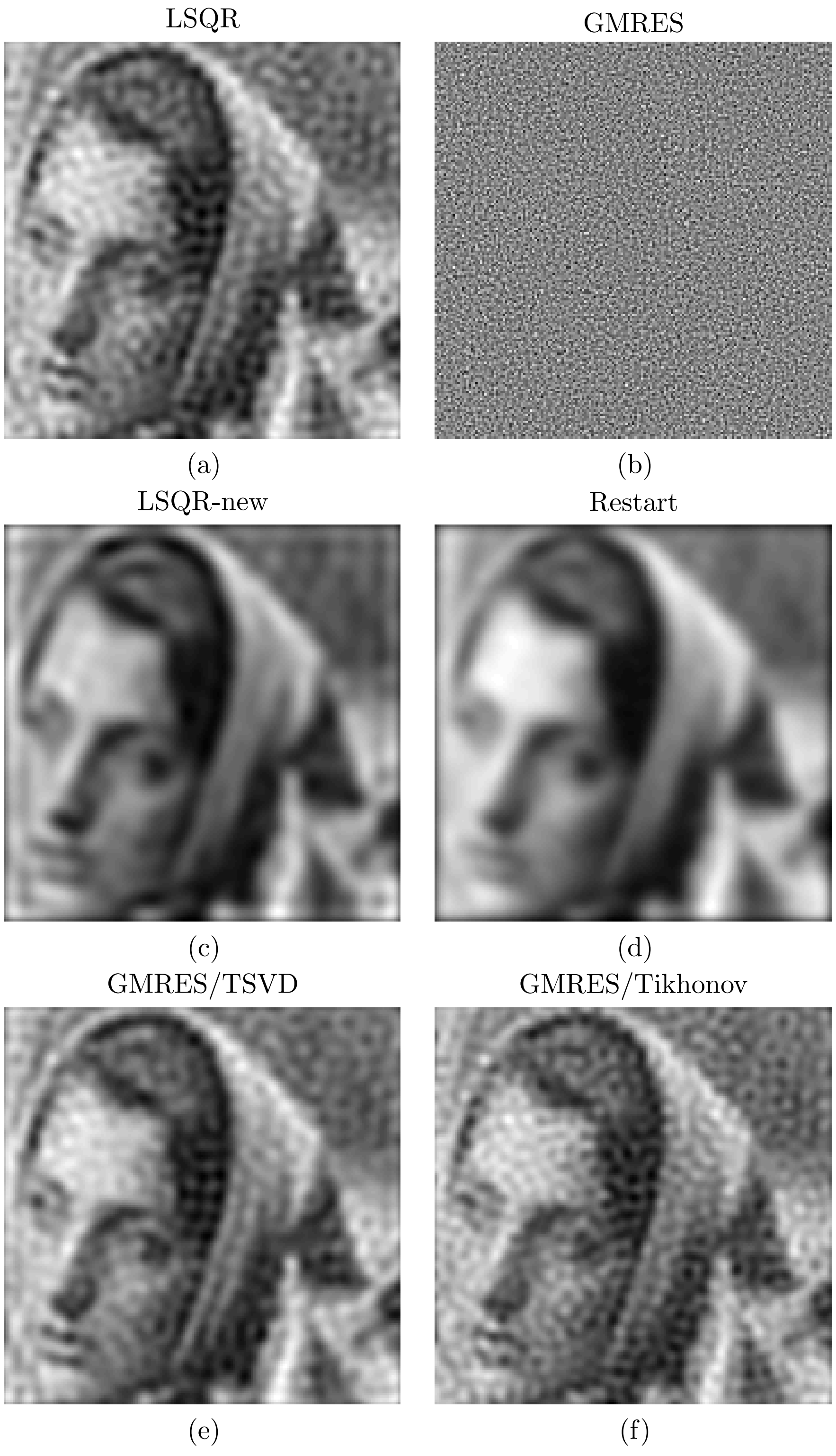


Figure 6.13: *All solutions corresponding to the filter factors shown in Figs. 6.11 and 6.12.*

Chapter 7

Stopping Criteria

Now we turn the attention towards a very important topic, namely stopping the iterative algorithms after an appropriate number of iterations. The Krylov methods are known to have semi-convergence, which means that the algorithms first include wanted information in the regularized solution, but at some point start to include noise. First the solution semi-converges to a wanted solution and later on the convergence goes towards the true, noisy, or naive solution. Stopping at the right time can be crucial for at least two reasons.

- Algorithms like LSQR can use an update strategy so all earlier iterates do not need to be stored. Running the algorithm too far, the solution will have passed the best solution and head on towards the true noisy solution – without any possibility for going back through the iterates to find the best one.

- Each iteration might be very expensive for larger problems – especially GMRES that needs to do reorthogonalization in each step, but also when running LSQR with forced reorthogonalization. We want to find the best solution as soon as possible.

First we provide a short description of the classical stopping criteria in connection with iterative algorithms.

7.1 Classical Stopping Criteria

The three stopping criteria mentioned in connection with the inner regularized GMRES in Section 6.1 are the well-studied classical stopping criteria. As stopping criteria for the iterative algorithms, the following can be said.

7.1.1 The Discrepancy Principle

The Discrepancy Principle is the simplest kind of stopping criteria to handle if the noise level is known or possible to estimate to a good accuracy. As mentioned earlier, we want to choose the first regularized solution for which:

$$\|\mathbf{A}\mathbf{x}_{\text{reg}}^{(k)} - \mathbf{b}\|_2 < \delta_e, \tag{7.1}$$

where δ_e is the size of the noise $\|\mathbf{e}\|_2$. The discrepancy principle is practical in connection with iterative algorithms, which we want to stop as soon as possible. But as seen from the example using Tikhonov in Section 3.6, Table 3.2 in connection with Fig. 3.15 show that the "freckles" can appear for this kind of solutions before the residual reaches the noise level. For the TSVD style solutions from the same section, we see the opposite – that we might actually include more components to get a more detailed solution, without including too many "freckles".

So even though the discrepancy principle surely provides a fair guess of when to stop the algorithms, one might want a more sophisticated stopping criteria, somehow taking into account the visual properties of the solution.

7.1.2 General Cross Validation

The General Cross Validation is based on the function in (6.6). This function includes some information about the regularized inverse of $\mathbf{A}$, $\mathbf{A}^{\#}$, giving rise to the calculated regularized solution. This regularized inverse is not directly obtainable in connection with iterative algorithms, which means that the GCV approach is not in general suited for iterative algorithms. It is not studied here.

7.1.3 The L-Curve

The L-Curve is the last of the three classical choices. The norms of the solutions $\|\mathbf{x}_{\text{reg}}^{(k)}\|_2$ and the residuals $\|\mathbf{A}\mathbf{x}_{\text{reg}}^{(k)} - \mathbf{b}\|_2$ after k iterations are easy to obtain, so constructing the L-Curve, defined in (6.8), is possible. For the standard algorithms, as well as for the LSQR-new, running in the modified Krylov subspace, the residual norm decrease monotonically and the norm of the solution increase monotonically with k if the initial guess is $\mathbf{x}_0 = \mathbf{0}$. In these cases the L-Curve is well-defined. For the inner regularized GMRES, and for the preconditioned solutions, the convergence of the solution and the residual cannot be assumed to be monotonic. This was for instance seen in Fig. 6.3. In these cases the L-Curve is not well-defined.

A problem using the L-Curve in connection with the iterative algorithms is though that the iterates appears one at a time, and to detect the corner of the L-Curve we possibly need some iterates after the corner. As we want to stop the algorithms as soon as possible, we want to avoid doing too many extra iterations.

Locating the Corner

Different strategies exist for locating the corner. P. C. Hansen's standard approach [8, Section 7.5] is based on the curvature of the L-Curve, defined as:

$$\kappa(k) = \frac{\zeta'\eta'' - \zeta''\eta'}{((\zeta')^2 + (\eta')^2)^{3/2}}, \tag{7.2}$$

where ζ and η are defined in (6.8), and the first and second derivatives are used, respectively. In the implementation in Regularization Tools [7], the curve itself is approximated by a B-Spline and the iteration lying closest to the point with maximum curvature is chosen as the corner of the discrete L-Curve. This strategy needs a minimum of iterates to fit the B-Spline as well as it needs some points after the corner for fitting the spline.

Another way of choosing the corner of the L-curve follows an idea by G. Rodriguez and D. Theis [19]. This is still work in progress, but the idea might become useful in connection with iterative algorithms. Generally the algorithm suggests looking at the vectors:

$$\mathbf{v}_i = \begin{bmatrix} \zeta_{i+1} \\ \eta_{i+1} \end{bmatrix} - \begin{bmatrix} \zeta_i \\ \eta_i \end{bmatrix}, \quad \text{for}, i = 1, 2, \ldots, k-1, \tag{7.3}$$

between two successive points on the L-Curve. Now one can look at the angle between two successive vectors, which actually describes the curvature of the L-Curve in a straight forward manner. It is suggested to define the corner as being the point characterized by an angle $\alpha \approx -\pi/2$ between two vectors. Following [19], we can find the corner by finding the minimum z-coordinate of the wedge product between two vectors as:

$$w_i = (\mathbf{v}_i \wedge \mathbf{v}_{i+1})_z = \|\mathbf{v}_i\| \|\mathbf{v}_{i+1}\| \sin(\alpha), \quad i = 1, 2, \ldots, k-2, \tag{7.4}$$

which can be easily computed as:

$$(\mathbf{v}_i \wedge \mathbf{v}_{i+1})_z = \det\left([\mathbf{v}_i \ \mathbf{v}_{i+1}]\right). \tag{7.5}$$

One advantage using this method is that as k grows, the new vectors can just be added to the vectors above, whereas the spline from the other approach, or at least a part of it, needs to be refitted. In this way we can update the wedge products as well element after element while tracking the minimum. Of course, we still might need a few extra iterations to make sure that the corner has been reached, but the updating idea seems to suit the iterative approach well.

7.2 Solution Properties

Working with images it is because of size difficult to do many fast calculations. Also it is difficult to manage the results as the information content is large. To handle those data in a more suitable manner, a *Graphical User Interface (GUI)* for MATLAB has been developed. The purpose has not been to develop a nice looking interface, but simply to be to be able to do fast changes between different iterations to get a visual understanding of the solutions. As well, different filtering in the cosine transformed frequency domain can be applied to try to distinguish good and bad solutions. An example plot of the generated GUI is seen in Fig. 7.1, using yet another test example. The original image used in this case is seen in Appendix B as Fig. B.7. The interface shows two solutions simultaneously – one GMRES or inner regularized GMRES solution, and one LSQR solution. With the scrollbars to the left, it is then possible to show the solutions after a different number of iterations for a visual comparison of those. In the right part of the interface, different plots of the norms of the solutions can be provided, selecting different filtering of the 2D-DCT frequency domain.

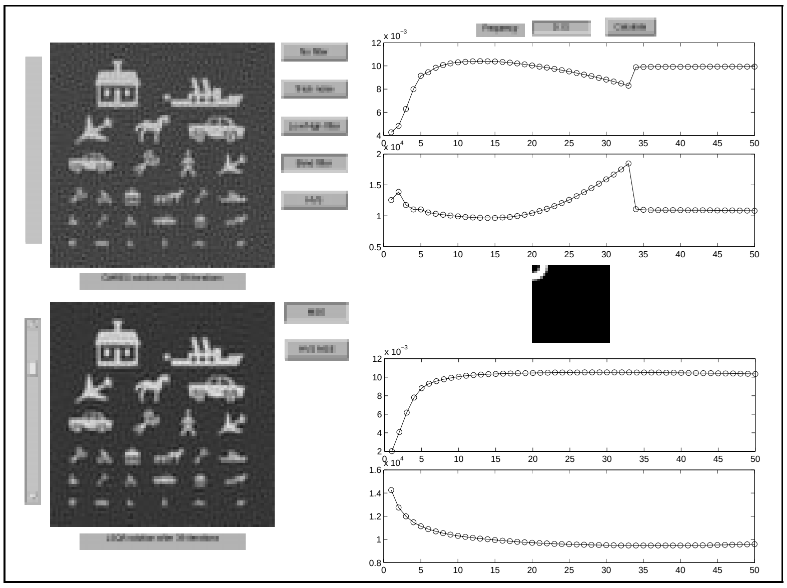


Figure 7.1: *Example of GUI interface used to study the regularized solutions and the solution norms* $\|\mathbf{X}_{\text{reg}}\|_2$.

To be able to investigate more objectively the "freckles" introduced in the regularized solutions, and find out when to stop, an artificial image consisting of different frequencies and different amplitudes has been constructed. The borders of the image have been smoothed to reduce the border effects. The true image is seen in Appendix A Fig. B.6, and the blurred, noisy realization is seen in Fig. 7.2.

In connection with Tikhonov regularization (2.13) as well as for working with the L-Curve, we try to balance some norm of the solution and some norm of the residual. Looking at 50 iterations of LSQR and GMRES, and the norms of the

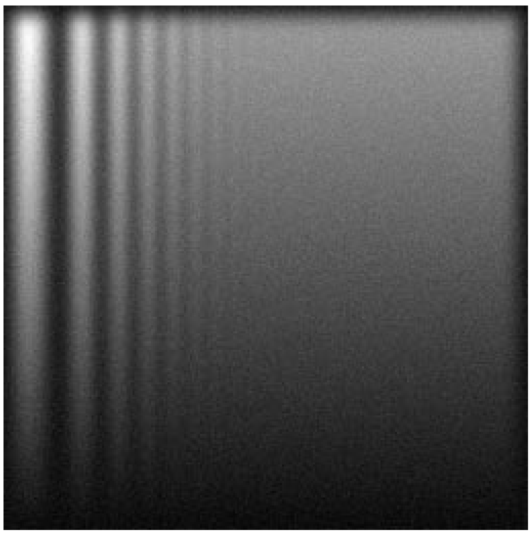
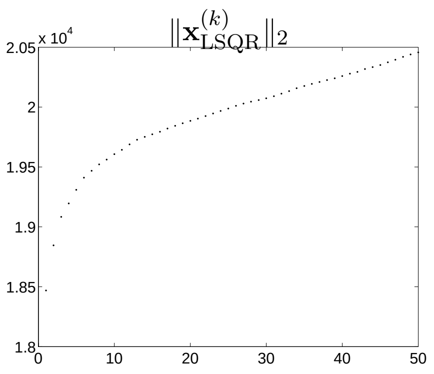
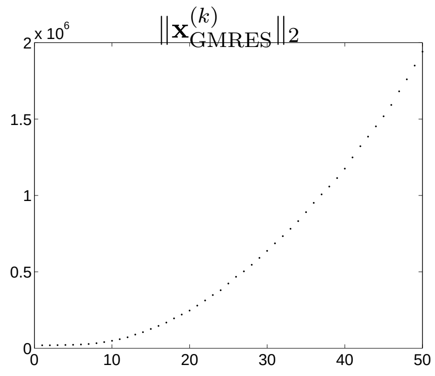


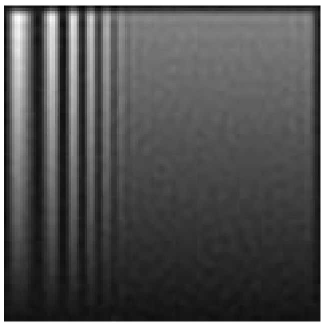
Figure 7.2: *Blurred and noisy realization of frequency and amplitude sweep.*



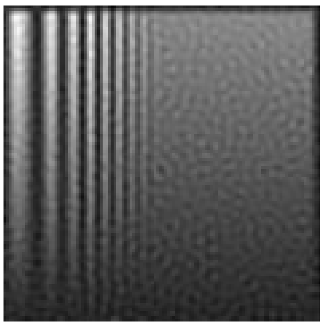
(a)



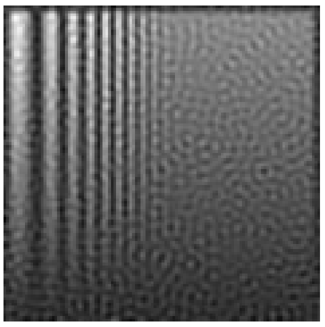
(b)



(c)



(d)



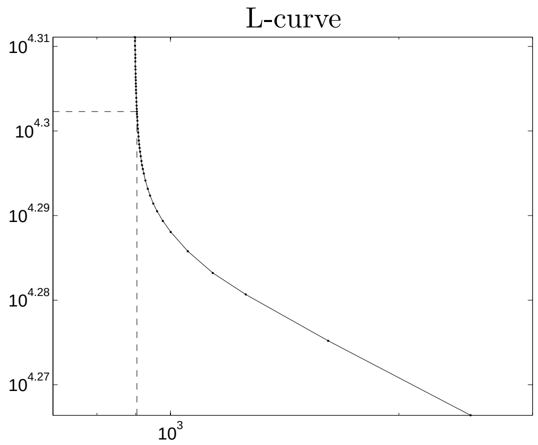
(e)

Figure 7.3: *Two-norm of the solutions for (a) LSQR and (b) GMRES over 50 iterations. (d), (e), and (f) show the LSQR solution after 10, 30 and 50 iterations.*

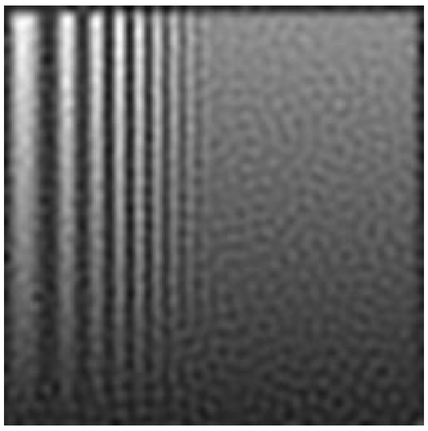
solutions obtained, we get the plots in Fig. 7.3. Here we observe the big difference once again. The high frequent noise in the GMRES solution causes the solution norm $\|\mathbf{x}^{(k)}_{\text{GMRES}}\|_2$ to increase drastically with k, whereas the solution norm for the LSQR solutions $\|\mathbf{x}^{(k)}_{\text{LSQR}}\|_2$ only increases modestly. The important part of looking at these plots is that the “freckles” in the LSQR solutions, and therefore also the “freckles” from inner regularized GMRES etc., do not affect the solution norm. In the bottom part of the figure, three LSQR solutions taken from the steadily increasing part of the solution norm curve, are seen. Obviously, we do not see much increase in the solution norm, indicating the increasing amount of “freckles”. Following the analysis from Section 3.5 and the studies of the different filter factors, it is clear that the “freckles” sneak into the solutions and do not make the solution norm “explode” at any point. Lets see how the L-curve handle the iterative algorithms and the nature of the image reconstructions.

7.3 L-curve in Practise

The norm of the regularized solution in case of image reconstruction increases slowly as the noise enters the solution as low-frequent “freckles”. This means that a lot of iterations will lie very close to the corner of the L-curve, making the curvature change very slowly. This makes it difficult to choose the most appropriate iteration.



(a)



(b)

Figure 7.4: *(a) Example showing L-Curve for the frequency test problem after 50 standard LSQR iterations. The corner found by using B-splines is shown with dashed lines and correspond to iterate $k = 30$. (b) Solution corresponding to the corner found on the L-Curve, $\mathbf{x}_{\text{reg}}^{(k)}$.*

Furthermore, as mentioned in Section 3.7.1, the human perception of the regularized solution is difficult to work with and include in a stringent measure. An example showing an L-curve after 50 iterations using LSQR and the frequency test image is seen in Fig. 7.4 (a). The corner of the L-curve is found by using the B-Spline approach, and is marked with dotted lines. The corner corresponds to the solution after $k = 30$ iterations. This solution is shown in Fig. 7.4 (b). As observed earlier, the "freckles" do not affect the two-norm of the solution very much, leading to a solution covered with "freckles" using this corner.

Thus we need a method to make the size of the solution $\Omega(\mathbf{x}_{\text{reg}})$ increase earlier, and in this way move the corner. To do this we would need something triggered by the "freckles", and we turn to look in the frequency domain.

7.3.1 Solution Norm

A simple idea is to use the noise ring in the frequency domain as studied in Section 5.3. We do a filtering of the frequency domain by generating $\widehat{\mathbf{F}}$ similar to (5.5), setting the elements within a certain distance of the upper left corner of the matrix to one and the remaining to zero:

$$\begin{aligned} I &= \left\{ (i,j) \mid f_1 \leq \frac{\sqrt{(i-1)^2 + (j-1)^2}}{N} \leq f_2 \right\} \quad f_1, f_2 \in [0;1[, \quad f_1 < f_2 \\ \widehat{\mathbf{F}}_{[i,j]} &= \left\{ \begin{array}{ll} 1 & \text{for}\,(i,j) \in I \\ 0 & \text{for}\,(i,j) \notin I \end{array} \right. \quad i,j = 1,2,\ldots,N. \end{aligned} \tag{7.6}$$

A filter like this is seen in the middle of the right part of Fig. 7.1. Now defining the matrix $\widehat{\mathbf{F}}_d$ as the diagonal matrix:

$$\widehat{\mathbf{F}}_d = \operatorname{diag}\left(\operatorname{vec}\left(\widehat{\mathbf{F}}\right)\right), \tag{7.7}$$

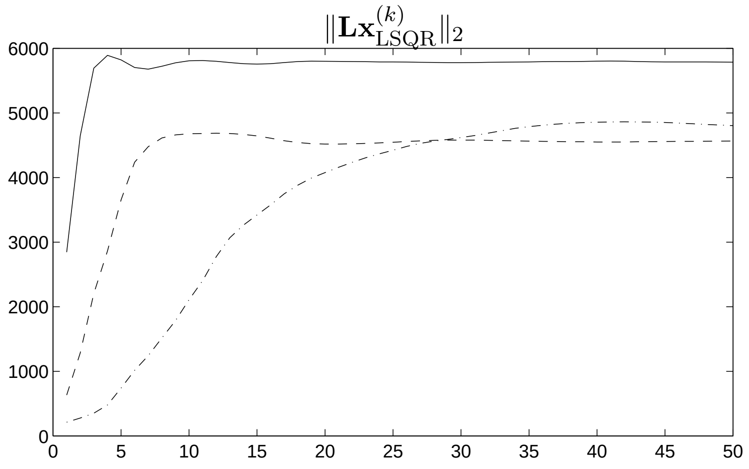


Figure 7.5: *Example showing different semi-norms of the solution for (solid line)* $f_1 = 0.05$, $f_2 = 0.10$, *(dashed line)* $f_1 = 0.10$, $f_2 = 0.15$, *and (dash-dotted line)* $f_1 = 0.15$, $f_2 = 0.20$.

containing the column-wise stacked elements of $\widehat{\mathbf{F}}$ down the diagonal, we can filter the 2D-DCT domain of a solution by:

$$\begin{aligned} \mathbf{x}_{\mathrm{filt}}^{(k)} &= \widehat{\mathbf{F}}_d(\mathbf{G}^T \otimes \mathbf{G}^T)\mathrm{vec}\left(\mathbf{x}_{\mathrm{reg}}^{(k)}\right) && (7.8) \\ &= \mathbf{L}\mathrm{vec}\left(\mathbf{x}_{\mathrm{reg}}^{(k)}\right). && (7.9) \end{aligned}$$

This defines the **L**-matrix that can be used in evaluating the semi-norm of the regularized solution, defining the Ω-function in the definition of the L-Curve in (6.8) as:

$$\Omega(\mathbf{x}_{\mathrm{reg}}) = \|\mathbf{L}\mathbf{x}_{\mathrm{reg}}\|_2. \quad (7.10)$$

The problem with this approach is that the two frequencies f_1 and f_2 from (7.6) need to be specified. And from the observations done using the interface, as well as the discussions in Chapter 5, the ring of amplified frequencies giving rise to the "freckles" is connected with both the noise level and the blurring. So depending on this, one would like to choose the frequencies f_1 and f_2 differently. Also **L** contains a large null-space, covering in effect all the frequencies that fall outside the specified band. I.e., **x** could have large frequency components outside the band specified in **L** still leading to $\|\mathbf{L}\mathbf{x}\|_2 = 0$ if all frequency components in the band are zero. An example showing different choices of f_1 and f_2 is seen in Fig. 7.5. As observed, we can somewhat control the behaviour of the size of the solution $\Omega(\mathbf{x}_{\mathrm{LSQR}}^{(k)})$ by changing the frequencies.

A further complication arises if the blurring is not symmetric. As shown in e.g. Figs. 3.7 and 5.8, we get reconstruction of frequencies in different directions in a different order. This means that the noise ring in this case is not a ring, but an ellipse. Making a general filtering of the 2D-DCT domain to evaluate the amount of "freckles" in the regularized solution thus depends highly on the noise as well as the blurring. Therefore, we might need some kind of semi-interactive approach, depending on the image to be restored, and the solution wanted. In some cases we might want a solution with higher detail level – and be able to live with the "freckles", whereas in other cases, we might want a visually more pleasing solution. A semi-interactive approach could also include the use of the HVS-function (3.42) that needs a parameter denoting the distance to the observed image. This is not investigated further here.

7.4 Cross Spectral Density

A new parameter choice method using the *Cross Spectral Density* (CSD) has been proposed in [10, Section 7] as a means to combine residual information and information about the regularized solution. The CSD function is in general given as:

$$\operatorname{csd}(\mathbf{x},\mathbf{r}) = \operatorname{conj}\left(\mathbf{F}^H\mathbf{x}\right) \odot \left(\mathbf{F}^H\mathbf{r}\right), \tag{7.11}$$

where $\mathbf{F}^H$ is a matrix representing the Discrete Fourier Transform and $\odot$ denotes the element wise product of two vectors. In [10] an analysis has been carried out leading to an optimal regularization parameter, in case of Tikhonov regularization, given as:

$$\lambda_{\text{CSD}} = \underset{\lambda}{\operatorname{argmin}} \ \|\operatorname{csd}\left(\mathbf{x}_\lambda, \mathbf{r}_\lambda\right)\|_\infty. \tag{7.12}$$

Arriving at this conclusion, a function similar to the CSD, but described in terms of the SVD, is used. In fact the Fourier base has just been exchanged by the SVD bases, leading to:

$$\mathbf{s}_\lambda = \left(\mathbf{V}^T\mathbf{x}_\lambda\right) \odot \left(\mathbf{U}^T\mathbf{r}_\lambda\right). \tag{7.13}$$

Using the Tikhonov filter factors and that the regularized solution in general is given as $\mathbf{x}_\lambda = \mathbf{V}\mathbf{\Psi}_\lambda\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$ similar to the two dimensional formulation in (3.27) as well as the residual is given as $\mathbf{r}_\lambda = \mathbf{U}\left(\mathbf{I} - \mathbf{\Psi}_\lambda\right)\mathbf{U}^T\mathbf{b}$, lead to:

$$\left(\mathbf{s}_\lambda\right)_i = \sigma_i\psi_i(1-\psi_i)\left(\frac{\beta_i}{\sigma_i}\right)^2 = \theta_i\left(\frac{\beta_i}{\sigma_i}\right)^2, \tag{7.14}$$

where $\beta_i = \mathbf{U}^T\mathbf{b}$, and $\theta_i = \sigma_i\psi_i(1-\psi_i)$.

Now we consider θ a function of the continuous variable σ. Studying $\theta(\sigma)$, this is seen to act as a filter, filtering out specific components of the solution coefficients. The analysis in [10] shows that looking at $\|\mathbf{s}_\lambda\|_\infty$ gives insight into the solution coefficients and further more that the minimum of this ∞-norm of $\mathbf{s}_\lambda$ over λ, will give a good estimate of λ_{optimal}. The correspondance between the frequency properties of the SVD bases and the Fourier base, also shown in [10], gives the link to the CSD.

Now for the class of 2D problems studied here, a function similar to the CSD can be set up. Using Definition 3.5 and (3.21) defining the 2D-DCT, we get:

$$\operatorname{csd}_2(\mathbf{X},\mathbf{R}) = \left(\mathbf{G}^T\mathbf{X}\mathbf{G}\right) \odot \left(\mathbf{G}^T\mathbf{R}\mathbf{G}\right), \tag{7.15}$$

or using the vec-notation:

$$\operatorname{csd}_2\left(\operatorname{vec}(\mathbf{X}), \operatorname{vec}(\mathbf{R})\right) = \left(\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\operatorname{vec}(\mathbf{X})\right) \odot \left(\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\operatorname{vec}(\mathbf{R})\right), \tag{7.16}$$

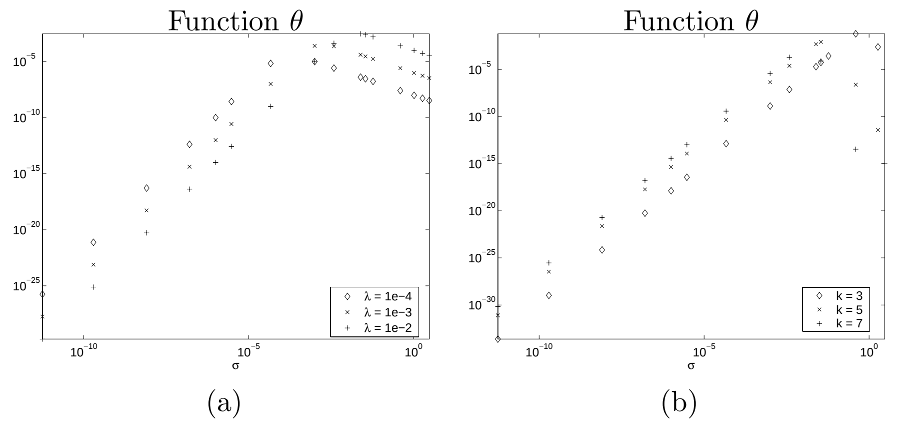


Figure 7.6: *(a) Function θ plotted using Tikhonov filter factors with different λ. (b) Function θ plotted using the filter factors comming from different iterations of the LSQR algorithm. The filter factors are calculated by means of* (4.35).

The expression for $\mathbf{s}_\lambda$ given in (7.13) is still valid for two dimensional problems with $\mathbf{x}_\lambda = \operatorname{vec}(\mathbf{X}_\lambda)$ and $\mathbf{r}_\lambda = \operatorname{vec}(\mathbf{R}_\lambda)$. Thus also the the optimal regularization parameter can in this case be found by means of $\mathbf{s}_\lambda$. According to the study from Chapter 3, it is possible to define a permutation matrix $\mathbf{P}$ so that $\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{V}$ is close to diagonal. In case of a square system with identical blurring of the columns and the rows, we have $\mathbf{U} = \mathbf{V}$. Then the following relations:

$$\begin{aligned}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right) &= \mathbf{P}^T\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{V}\mathbf{V}^T \\ &= \mathbf{P}^T\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{U}\mathbf{U}^T\end{aligned}$$

where $\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{V} \approx \mathbf{I}$ and $\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{U} \approx \mathbf{I}$ lead to:

$$\begin{aligned}\operatorname{csd}_2(\mathbf{x},\mathbf{r}) &= \mathbf{P}^T\left(\left(\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{V}\mathbf{V}^T\mathbf{x}\right) \odot \left(\mathbf{P}\left(\mathbf{G}^T \otimes \mathbf{G}^T\right)\mathbf{U}\mathbf{U}^T\mathbf{r}\right)\right) \quad (7.17)\\ &\approx \mathbf{P}^T\mathbf{s}_\lambda. \quad (7.18)\end{aligned}$$

Since a permutation of the elements in $\mathbf{s}_\lambda$ does not affect the ∞-norm, we have:

$$\|\operatorname{csd}_2(\mathbf{x},\mathbf{r})\|_\infty \approx \|\mathbf{s}_\lambda\|_\infty, \quad (7.19)$$

and similar to (7.12), the optimal regularization parameter is given by:

$$\lambda_{\mathrm{CSD2}} = \underset{\lambda}{\operatorname{argmin}} \;\|\operatorname{csd}_2(\mathbf{x}_\lambda,\mathbf{r}_\lambda)\|_\infty. \quad (7.20)$$

To see how this works in practice, the small one dimensional example shaw is again used. In Fig. 7.6, the function θ from (7.14) is shown to the left for the Tikhonov filter factors different values of λ, and to the right for different iterations of the LSQR algorithm. It is seen how different singular values – and then different solution coefficients – are picked out.

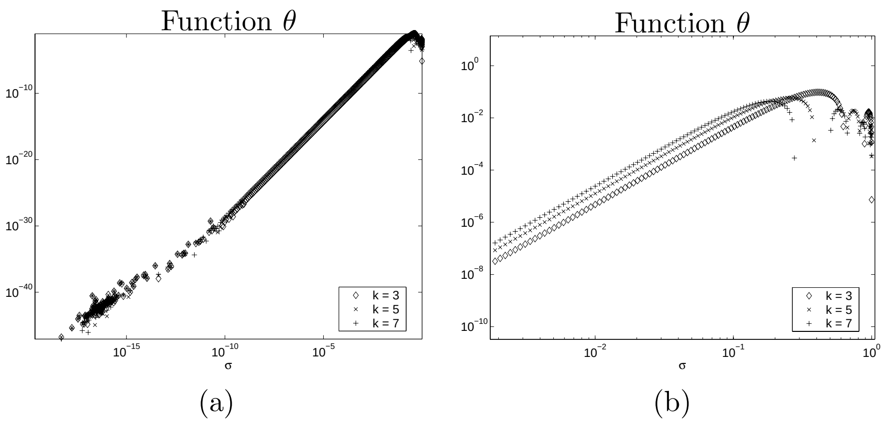


Figure 7.7: *Function θ plotted using the filter factors coming from different iterations of the LSQR algorithm for the one dimensional penguin problem. The filter factors are calculated by means of* (4.35). *(a) Shows the full plot, and (b) shows a zoom of the top part.*

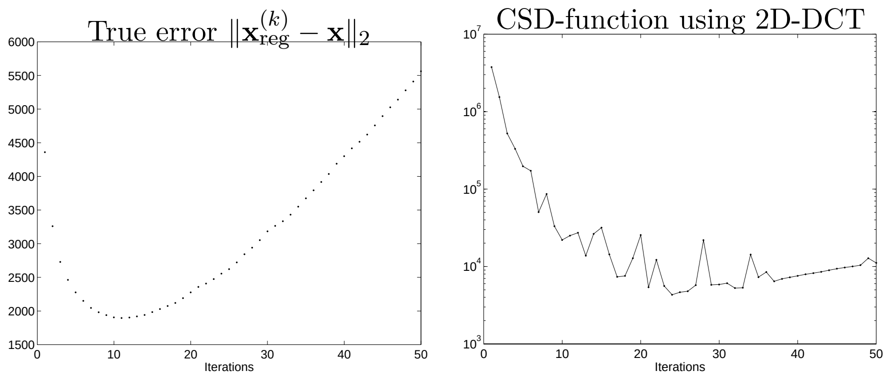
Now looking at the one row of the penguin image from Section 5.3.2, we know from the analysis of the Ritz polynomial that the filter factors for the regularized solution do not behave exactly like the Tikhonov filter factors. In fact the study showed that many filter factors are close to one, even though the singular values have not yet converged, which leads to filter factors following the oscillating Ritz polynomial. The function θ is shown for different iterations of the LSQR algorithm for this problem in Fig. 7.7. The right plot is a zoom of the left. It is clearly seen that the specific pick of frequencies does not apply to this $\theta(k)$ as the filters have several local maxima.

Using this stopping criteria for a two dimensional penguin test problem of size $N = 100$, the results are seen in Fig. 7.8, where a comparison of the two-norm $\|\mathbf{x}_{\text{reg}}^{(k)} - \mathbf{x}\|_2$ and the CSD-function is shown. It is clearly seen that there is no obvious relation between the minimum two-norm and the minimum of the CSD-function. Based on the analysis of the filter factors, there is no indication that this stopping criterion will be succesfull either. The CSD function has a large fluctuating region close to the minimum, and therefore no well-defined minumum.

7.5 Final Remarks

The stopping criteria discussed here assume that the iterative algorithms have semi-convergence. For the solutions found by preconditioning and inner regularized GMRES, the convergence is different. As the preconditioned algorithm discussed in Section 6.2 did not give positive results, it is not discussed further here.

For an example problem, the norm of the solutions found by inner regularized GMRES was shown in Fig. 6.3. In this case it is seen that the need for a stopping criteria is not as crucial as for the standard algorithms. As the noise is included in the subspace, this is removed again by the inner regularization. This changes the difficulty in choosing the right stopping criteria for the iterative algorithm to a difficulty in choosing the right regularization parameter for the inner regularization.



(a) (b)

Figure 7.8: *(a) True two-norm error between regularized solution and true solution. (b) The ∞-norm of the Cross Spectral Density, defined by the 2D-DCT transform* (7.16).

Of course it is still desirable to stop the iterations when no more information is found to save possibly expensive calculations. In this case an obvious criteria could be when the solution norm is in some way stabilized. This stabilized solution thus still has the same problem with the “freckles” as do e.g. the LSQR and Tikhonov solutions.

A quality measure of the solution that can be used in connection with e.g. the L-curve to find the best solution is therefore still needed, and leaves the possibility for future work in this direction. In this connection also different statistical considerations and analysis of the found solutions might be of interest. Defining different criteria for different classes of images and developing semi-interactive approaches might as well be of future interest.

CHAPTER 8

Conclusion

This last chapter concludes the work done in this thesis. Also, it provides some ideas and suggestions for possible future work.

As briefly mentioned in the introduction, the main motivation for starting this project was a desire to look for new and more powerful stopping criteria for iterative algorithms in connection with image restoration. The stopping criteria at hand did not seem to give nice solutions. But during the work it became clear that talking about stopping criteria did not make sense without having a better understanding of images and what regularization does to them. This has lead to the various studies carried out in this thesis.

First a study of directly calculated solutions has been carried out. A study of how the singular values in theory, as well as in practise, behave for image restoration problems. It was seen that the frequency decomposing properties of the SVD carries over to the two-dimensional domain, even though the concept of frequency is more complicated as also the direction of the frequency is important. This in turn lead to an explanation of why the singular values fall off so slowly for this kind of problems.

The two iterative algorithms LSQR and GMRES have been studied, and the solutions they provide have been throughly analyzed. The results have been connected to the studies done for the directly analyzed properties of the image problems. This has lead to an understanding of mainly two important issues:

- The noisy GMRES solutions have been explained through a direct study of the Krylov subspaces from which the algorithm construct the solutions. Looking at the filter factors also provided part of the insight.

- The "freckles" seen in connection with especially Tikhonov regularization and the solutions coming from the basic LSQR algorithm have been described, and their origin has been analyzed. Also, the "freckles" are found in a constructed one-dimensional problem showing that this is a general property for problems with slowly decaying singular values. Furthermore, the two-dimensional "freckles" are seen to have a visually disturbing effect.

The insight obtained has then been used to suggest changes of the iterative algorithms to cure the artifacts. For GMRES, inner regularization seems like a good idea as this quite dramatically removes the noisy components of the solutions. This, on the other hand, leaves us again with solutions that possibly get covered with "freckles", depending on the regularization parameter chosen for the inner problem. Preconditioning has also been suggested, and for some problems it is seen to work, whereas for others the impact of preconditioning has a catastrophic effect.

Trying to control the "freckles", attempts to make the iterative algorithms create sharper TSVD-like filter factors have been made. This was absolutely possible, e.g., by starting the LSQR algorithm differently. The results showed a connection between the filter factors and the convergence speed of the algorithm. The sharper filter factors, the slower convergence.

When it comes to stopping criteria it is seen that the properties of image restoration give severe difficulties in choosing the correct iteration for stopping the algorithms. First, the noise enters very slowly making only little change to the solution norm. And second, finding a solution that is pleasing to the eye needs a more stringent measure connected with the properties of the human visual system.

Concluding this work, we feel that the above studies provide a great deal of insight into regularization of images – and will be useful as a basis for future work.

Future Work

Based on the analysis of the effect of regularizing images, we here give a few suggestions for how to proceed in the future.

- The stopping criteria for the different iterative algorithms still need deeper investigation.

- Other types of preconditioners than the one studied here might lead to different and maybe better solutions. Current work in preconditioners is e.g. going on in the development of Restoration Tools for MATLAB by J. G. Nagy [14].

- Other measures for the quality of a regularized image. E.g., measures based on statistical considerations including knowledge about the actual scene observed. Using techniques from image compression or statistical image analysis might also prove useful.

- Investigation of other algorithms and their properties. Either working in other Krylov subspaces or finding other optimal solutions from the Krylov subspaces. An interesting algorithm might, e.g., be the *minimal error* method by M. Hanke [5].

Appendix A

Matlab Code

A.1 GMRES with Preconditioning and Inner Regularization

This code shows the GMRES algorithm from the Regularization Tools XP [12]. The inner regularization is based on work done in this project, and the parts providing the possibility for using preconditioners is still not included in the toolbox. These parts are implemented during the project for testing purposes only, based on a paper by Y. Saad [20].

```
function [X,extra,restart] = gmres(A,b,options,restart)
% gmres Generalized Minimum RESidual
%
% <Synopsis>
%  [X,extra,restart] = gmres(A,b,options,restart)
%
% <Description>
%  Solve the linear system
%  (1)        Ax = b
%  where A is square (but not necessarily symmetric).
%
% <Input Arguments>
%   * A,b      Square LinearOperator
%   * options  A structure with options (optional)
%    - Iter  Maximum number of iterations (size(K,1)). If MaxIter is an
%              vector max(MaxIter) iterations are performed and the result
%              of iterations in the vector are return in X.
%    - TolRes   Residual reduction tolerance (1e-6)
%    - x_true A Vector containing the (true) solution. The extra output
%             structure will then have the relative error w.r.t. x_true for
%             each iteration.
%    - InSolv   Name of inner regularization solver ([] / gauss eliminiation)
%    - InArgs  Arguments for inner regulariztion
%   * restart  Information for a restart of algorithm (NOT IMPLEMENTED)
%
%  <Output Arguments>
%    * X       The result after last iteration. If MaxIter is a vector a
%              VectorCollection with results after each iteration listed.
%    * extra   A structure with extra information (if asked for)
%     - V      Basis of Krylow subspace used
%     - rho    Estimated residual norm after each iteration
%     - eta    Norm of solution after each iteration
%     - iter   The iteration nm for the entry/entries in X
%     - error  Error w.r.t. x_true after each iteration
%    * restart  A structure with information needed to restart GMRES (as
%                if it was never stopped, i.e., not a restart in the
%                usual GMRES vocabulary). (NOT IMPLEMENTED).
%
%  The default option structure is returned by
%  >>  opt = gmres('defaults');
%
% <See also>
%   lsqr_c, cgls
%
% <References>
%   1. Youcef Saad, Martin H. Schultz, "GMRES: A Generalized Minimal
%   Residual Algorithm for Solving Nonsymmetric Linear Systems", SIAM
%   Sci. Stat. Comput., vol 7, pp 856--869, 1986.
%

% Jan M. Rasmussen, IMM, DTU, 2000

% Last revised $Date: 2003/02/13 14:19:44 $ by $Author: mj $
% $Revision: 1.8 $

% Based on gmres by Jan Marthedal Rasmussen, October 2000

defaultopt = struct(...
                'Iter', inf, ...
                'TolRes', 1e-6, ...
                'x_true', [], ...
                'Precond', [], ...
                'InSolv', [],...
                'InArgs', []);
if nargin == 1 & nargout <= 1 & isequal(A, 'defaults')
        X = defaultopt; return;
end
if nargin == 2, options = defaultopt; end

[m,n] = size(A);
tol       = regget(options, 'TolRes', 1e-6);
k         = regget(options, 'Iter', n);
x_true    = regget(options, 'x_true', []);
inner     = regget(options, 'InSolv', []);
innerargs = regget(options, 'InArgs', []);
P         = regget(options, 'Precond', []);

% Find which iterations to return.
k = sort(k); maxiter = k(end);

% Reserve memory
h = zeros(maxiter,1);
res = zeros(maxiter,1);
xnorm = zeros(maxiter,1);
iter = zeros(length(k),1);
V = VectorCollection(maxiter);
Z = VectorCollection(maxiter);
```

```
W = VectorCollection(maxiter);
X = VectorCollection(length(k));
r = b;

eta = norm(r);
Q(1,1) = 1;
res(1) = eta;
if ~isempty(x_true), error = zeros(maxiter,1); end

x = zeros(b);
xnorm(1) = norm(x);
count = 1;
nns = 0;
for i=1:maxiter,
  progress;
  % Check tolerance, if we stop store last result
  if res(i) <= tol*res(1),
                if isempty(inner)
                        X(:,count) = x;
                else
                        H = Matrix(Q(:,1:i-1)*T);
                        [UU,SS,VV] = svd(H);
                        SVD = OperatorSVD(UU(1:i-1,1:i-1),SS,VV);
                        VTb = V(:,1:i-1)'*b;
                        X(:,count) = Z(:,1:i-1)*feval(inner,SVD, VTb, innerargs);
                end
                iter(count) = i;count = count + 1;
    break;
  end

  V(:,i) = r / eta;

  if(~isempty(P)),
    if(~iscell(P)),
      Z(:,i) = P\V(:,i);
      r = A*Z(:,i);
    else
      rtmp = realmax;
      disp('Selecting Preconditioner...');
      for l=1:length(P),
        %disp(sprintf('Calculating preconditioner %d',l));
        Ztmp = P{l}\V(:,i);
        AZ = A*Ztmp;
        % Perform Gram-Schmidt!!
        for j=1:i
                htmp(j) = V(:,j)'*AZ;
                AZ = AZ - htmp(j)*V(:,j);
        end
        nz = norm(AZ);
        nns(i,l) = nz;

        if(nz<rtmp),
          rtmp = nz;
          Z(:,i) = Ztmp;
          r = A*Ztmp;
          disp(sprintf('Better preconditioner: %d', l));
        end;
        %plot(l,nz,'.');hold on;
      end;
    end
    %opts = regset('Iter', 2);
    %Z(:,i) = jacobi(A, V(:,i), V(:,i), 4);
    %Z(:,i) = lsqr(A, V(:,i), opts);
    %Z(:,i) = gmres(A, V(:,i), opts);
  else
    Z(:,i) = V(:,i);
    r = A*Z(:,i);
  end

  %r = A*Z(:,i);

  % Modified Gram-Schmidt on new vector
        for j=1:i
                h(j) = V(:,j)'*r;
                r = r - h(j)*V(:,j);
        end
        eta = norm(r);

        % Apply previous rotations to h
        T(1:i,i) = Q(1:i,1:i)'*h(1:i);

        % Compute Givens rotation parameters
        rc = T(i,i);
        if eta == 0
                c = 1; s = 0;
        elseif abs(eta) > abs(rc)
                tau = -rc/eta;
                s = 1 / sqrt(1 + abs(tau)^2);
                c = s*tau;
        else
                tau = -eta/rc;
                c = 1 / sqrt(1 + abs(tau)^2);
                s = c*tau;
        end

        % Apply Givens rotation
        T(i,i) = c*rc - s*eta;
        Q(1:i, [i i+1]) = Q(1:i,i)*[c s];
        Q(i+1, [i i+1]) = [-s c];

        if abs(T(i,i)) <= eps
                disp('Operator Numerially Singular - aborting');
                break;
        end

        % Update W
        if i == 1
                W(:,1) = V(:,1) / T(1,1);
        else
                W(:,i) = (V(:,i) - W(:,1:i-1)*T(1:i-1,i))/T(i,i);
        end

        % Update solution (without inner regularization
        %x = x + Z(:,i)*(V(:,i)'*(res(1)*Q(1,i)*W(:,i)));
        x = Z(:,1:i)*res(1)*(T(1:i,1:i)\Q(1,1:i)');
        %x = x + res(1)*Q(1,i)*W(:,i);

        % Update output variables
        if ~isempty(x_true), error(i) = norm(x-x_true)/norm(x_true); end
        if i == k(count)
                if (isempty(inner) || i == 1),
                        X(:,count) = x;
                else
                        H = Matrix(Q(:,1:i)*T);
                        size(H)
                        [UU,SS,VV] = svd(H);
                        SVD = OperatorSVD(UU(1:i,1:i),SS,VV);
                        VTb = V(:,1:i)'*b;
                        X(:,count) = Z(:,1:i)*feval(inner,SVD, VTb, innerargs);
                end
                iter(count) = i;count = count + 1;
        end
        res(i+1) = res(1)*abs(Q(1,i+1));
        xnorm(i+1) = norm(x);
end

% Trim the result VectorCollection
if count == 2
        X = X(:,1);
else
        X = X(:,1:count-1);
end

extra.iter = iter(1:count-1);
extra.rho = res(1:i);
extra.eta = xnorm(1:i);

if ~isempty(x_true), extra.error = error(1:i); end
restart = 0;
extra.nns = nns;
```

A.2 Restarted Bidiagonalization

The code given here is a striped down version of the Restarted Bidiagonalization algorithm with inner GCV function as provided by Å. Björck and E. Grimme and P. van Dooren [1]. This version gives the possibility for creating a bidiagonalization after a given number of iterations, followed by a given number of restarts.

```
function [x, info] = lbdr(A,b,c,nRestarts)
%
% A: The coefficient matrix
% b: The right-hand side
% c: The number of iterations to perform before restarting
% nRestarts: The number of restarts to perform
%
% Initialization
if nargin < 2, error('The rhs vector was not specified'); return; end
if nargin < 3, c = 3; end
```

```
[m,n] = size(A);

% Compute a bidiagonal factorization of size c
[U,V,B] = lanc_b(0,c+1,A,b);

k = c+1;
restarts = zeros(c+1,1);

while k<nRestarts+c+1,
     k = k + 1;
     restarts = [restarts;0];
     [U,V,B] = lanc_b(k-1,k,A,U,V,B);
     [B,U,V] = bsvdstep(B,U,V);
     U = U(:,1:k); V = [V(:,1:k-1),B(k,2)*V(:,k)+V(:,k+1)]; B=B(1:k-1,:);
     restarts(k) = restarts(k) + 1;
     [U,V,B] = lanc_b(k-1,k,A,U,V,B);
end

% Compute a solution
[P,omega,Q] = bsvd(B);

OM = DiagonalOperator(Vector(omega))
OP = OperatorProduct({U, Matrix(P)})
OQ = Matrix(Q)
AA = OperatorSVD(OP, OM, Matrix(Q));
opts = regset('RegPar', length(omega));
x = tsvd(AA,b,opts);
x = V(:,1:k)*x;

% Compute number of restarts needed
rst = 0;
for i=1:k,
  rst = rst + restarts(i);
end

info.restarts = restarts;
info.V = V;
info.U = U;
info.B = B;
```

Lanc_b

Subroutine calculating the Lanczos bidiagonalization. The implementation is changed to work with the objects defined in Regularization Tools XP [12].

```
function [U,V,B] = lanc_b(k1,k2,A,U,V,B)
%
%BIDIAG [U,V,B] = lanc_b(k1,k2,A,U,V,B)
%
% Performs k2-k1 steps of the Lanczos bidiagonalization process with
% starting vector u, producing a lower bidiagonal matrix
%             [b_11                   ]      [b_21      b_11]
%             [b_21 b_22              ]           [b_32      b_22]
%         B = [     b_32  .           ] stored as [ .          .  ]
%             [           .  b_kk     ]           [b_(k+1)k  b_kk]
%             [              b_(k+1)k]
% U and V consist of the left and right Lanczos vectors.
```

```
%
% If k1 is non-zero, it is assumed that a k1 step bidiag factorization
% already exists and is contained in (U,V,B).  Otherwise, it is assumed
% that U contains the starting vector while V and B are ignored.
%
% Note that V is always constructed with the residual vector
% tacked on as the last column of V.

% Eric Grimme, Univ. of Illinois at UC, 10/14/93
% Offtake of version by P. C. Hansen, UNI-C

% Reference: G. H. Golub and C. F. Van Loan, "Matrix Computations", John
% Hopkins, Baltimore, 1989; Section 9.3.4.

% Initialization
if (nargin<4), error('Insufficient number of input arguments'), end
if (nargin<6 & k1>0), error('Bidiag factorization of size k1 missing'), end
if (k1<0), error('Starting Index must be non-negative'), end
if ((k2-k1)<0), error('Number of steps must be non-negative'), end

if k1==0, B = [ ];
 b = U/norm(U);
  U = VectorCollection(k2);
  V = VectorCollection(k2);
 U(:,1) = b;
 V(:,1) = A'*U(:,1);
 B = sparse(0,0);
end
B = [B;zeros(k2-k1,2)];

% Compute the additional steps
for i = k1+1:k2,
  progress;
  alpha = norm(V(:,i));
  V(:,i) = V(:,i)/alpha;
  U(:,i+1) = A*V(:,i)-alpha*U(:,i);
  for j=1:i,
    U(:,i+1) = U(:,i+1) - (U(:,i+1)'*U(:,j))*U(:,j);
  end
  beta = norm(U(:,i+1));
  U(:,i+1) = U(:,i+1)/beta;
  V(:,i+1) = A'*U(:,i+1)-beta*V(:,i);
  for j=1:i,
    V(:,i+1) = V(:,i+1) - (V(:,i+1)'*V(:,j))*V(:,j);
  end
  B(i,2) = alpha; B(i,1) = beta;
end
```

Bsvdstep

Subroutine used by the Restarted Bidiagonalization algorithm.

```
function [B_k,U,V] = bsvdstep(B_k,U,V);
%BSVDSTEP apply an SVD step to a bidiagonal matrix stored in ``compact form''.
%
% [B_k] = bsvdstep(B_k)
% [B_k,U,V] = bsvdstep(B_k,U,V)
%
```

```
% Applies a Golub-Kahan SVD step to the lower bidiagonal matrix B stored in
% compact form in B_k.
%
% B is overwritten with the lower bidiagonal matrix Ql'*B*Qr where Ql and Qr
% are orthogonal and Ql is essentially the orthogonal matrix obtained from the
% QR decomposition of B*B'.  If U and V are supplied as inputs, they are
% overwritten with U*Ql and V*Qr respectively.  If U and V have more than
% k+1 and k columns respectively, these extra columns are modified by no more
% than a scalar factor.

% Eric Grimme, University of Illinois at UC, 7/9/94.

% Reference: J. Demmel and W. Kahan, "Accurate Singular Values of Bidiagonal
% Matrices", SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873--912.

% Initialization.
if (nargout > 1) & (nargin < 3), error('Too few input arguments'); end

[k,t] = size(B_k);
oldc = 1;
f = B_k(1,2); g = B_k(1,1);

% Chase a bulge down the lower bidiagonal with Givens rotations.
for i=1:k-1,
  G = givens(f,g); c = G(1,1); s = G(1,2);
  r = c*f+s*g;
  if (i ~= 1), B_k(i-1,1) = olds*r; end
  if (nargin == 3), U(:,i:i+1) = U(:,i:i+1)*G'; end
  f = oldc*r;
  g = B_k(i+1,2)*s; h = B_k(i+1,2)*c;
  G = givens(f,g); c = G(1,1); s = G(1,2);
  r = c*f+s*g;
  B_k(i,2) = r;
  if (nargin == 3), V(:,i:i+1) = V(:,i:i+1)*G'; end
  f = h; g = B_k(i+1,1);
  oldc = c; olds = s;
end
G = givens(f,g); c = G(1,1); s = G(1,2);
r = c*f + s*g;
B_k(k-1,1) = olds*r; f = oldc*r;
B_k(k,1) = 0; B_k(k,2) = f;
if (nargin == 3), U(:,k:k+1) = U(:,k:k+1)*G'; end

% If V has a (k+1)st column, treat it as an implicit restart residual vector.
if (nargin==3)
  [m,n] = size(V);
  if n > k, V(:,k+1) = s*V(:,k+1); end
end
```

Bsvd

Wrapper function calculating the SVD of a matrix **B** where the bidiagonal is stored as two columns.

```
function [U,s,V] = bsvd(B);
  [U,s,V] = csvd(full(spdiags(B, [-1 0],size(B,1)+1,size(B,1))));
```

A.3 Generation of Problems

The code given here is used to generate problems in a unified form. Given a problem number, a type of blurring, and a noise level, the matrix **A**, as well as the images **b**, $\bar{\mathbf{b}}$, **x**, and the noise **e** are returned. Also, the size of the problem as well as the name of the problem are returned. In case of the Frequency/Amplitude image, the size of the image can be specified. All other images are fixed in size and loaded from the disk.

```
function [A, b, bex, e, xtrue, problemsize, name] = ...
           genProblem(problem, blur, noisefactor, tamano);
% [A, b, bex, e, xtrue, problemsize] = genProblem(size, blurtype, noisefactor);
% size denotes: 0: large, 1: small, 2: mini, 3: tiny
%
% blur: matrix defining blur
%
%   blur = [4, 13; 4.5, 13] e.g. spatially variant (half and half)
%   blur = [4, 13]          e.g. symmetric
%   blur = [4, 13, 4, 13]   e.g. asymmetric but spatially invariant
%
% Implemented problems:
%   1: penguins - full size (512x512)
%   2: penguins - small     (100x100)
%   3: penguins - mini      (50x50)
%   4: penguind - tiny      (30x30)
%   5: barbara  - full size (512x512)
%   6: barbara  - small     (175x175)
%   7: goldhill - full size (512x512)
%   8: lena     - full size (512x512)
%   9: FreqAmp  - variable  Use 4th input to specify size
%  10: Map image- full size (400x400)
%  11: Satellite- full size (256x256)
%  12: Eyetest  - full size (290x290)
%

  switch(problem)
   case 1, load penguins;          name = 'Penguin 512x512';
   case 2, load penguins_small;    name = 'Penguin 100x100';
   case 3, load penguins_mini;     name = 'Penguin 50x50';
   case 4, load penguins_tiny;     name = 'Penguin 30x30';
   case 5, load barbara;           name = 'Barbara 512x512';
   case 6, load barbara_small;     name = 'Barbara 175x175';
   case 7, load goldhill;          name = 'Goldhill 512x512';
   case 8, load lena;              name = 'Lena 512x512';
   case 9, xtrue = freqamp(tamano); name = sprintf('Freq/Amp-sweep %dx%d',tamano,tamano);
   case 10, load map;              name = 'Computer Map 400x400';
   case 11, load satellite;        name = 'Satellite 256x256';
   case 12, load eyetest;          name = 'Eye test 290x290';
  end;

  n = size(xtrue); n = n(1);
  e = Vector2D(randn(n));

  if(min(size(blur)) == 1),
    if(max(size(blur)) == 4),
        A = genBlur(n, blur(1), blur(2), blur(3), blur(4));
    else
```

```
        A = genBlur(n, blur(1), blur(2));
    end
  else
    [A1, T1] = genBlur(n, blur(1,1), blur(1,2));
    [A2, T2] = genBlur(n, blur(2,1), blur(2,2));
    nsq = n^2; nsqh = nsq/2;
    I1 = [speye(nsqh, nsqh) sparse(nsqh, nsqh); sparse(nsqh, nsq)];
    I2 = speye(n^2) - I1;
    I1 = Matrix(I1);
    I2 = Matrix(I2);
    A = I1*A1 + I2*A2;
  end

  bex = A*xtrue;
  e = e*(norm(bex)/norm(e))*noisefactor;
  b = bex + e;
  problemsize = size(b);
```

Generation of Blur

This subroutine generates the desired blurring matrix **A** from given blurring parameters σ_1 and σ_2 as well as bandwidths for truncating the Toeplitz matrices. The first Toeplitz matrix can be returned as well. The function uses the KroneckerProduct2D and Matrix objects from Regularization Tools XP [12].

```
function [A, T] = genBlur(n, sigma, band, sigma2, band2);

if nargin < 3,
    band = 5;
    if nargin < 2,
        sigma = 0.7;
    end
end

col = zeros(n,1);
col(1:band) = 1/(sqrt(2*pi)*sigma) * exp(-0.5*(([0:band-1]')/sigma).^2);
T = Matrix(toeplitz(col));

if(nargin>3)
    col = zeros(n,1);
    col(1:band2) = 1/(sqrt(2*pi)*sigma2) * exp(-0.5*(([0:band2-1]')/sigma2).^2);
    T2 = Matrix(toeplitz(col));
else
    T2 = T;
end

A = KroneckerProduct2D(T, T2);
```

Appendix B

True Images Used

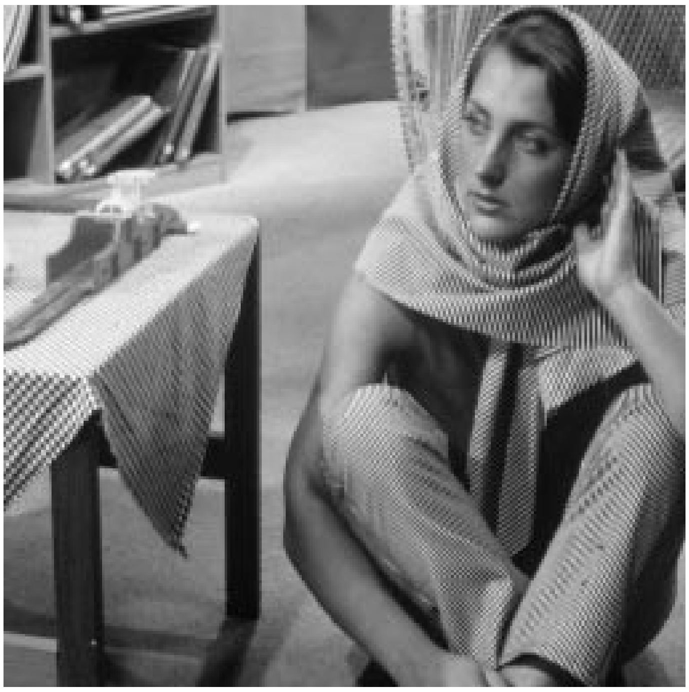


Figure B.1: *Barbara – Size* 512×512. *Original image can be found several places on the internet. E.g. here: http://herbert.the-little-red-haired-girl.org/en/research/papers/filter_evaluation/*

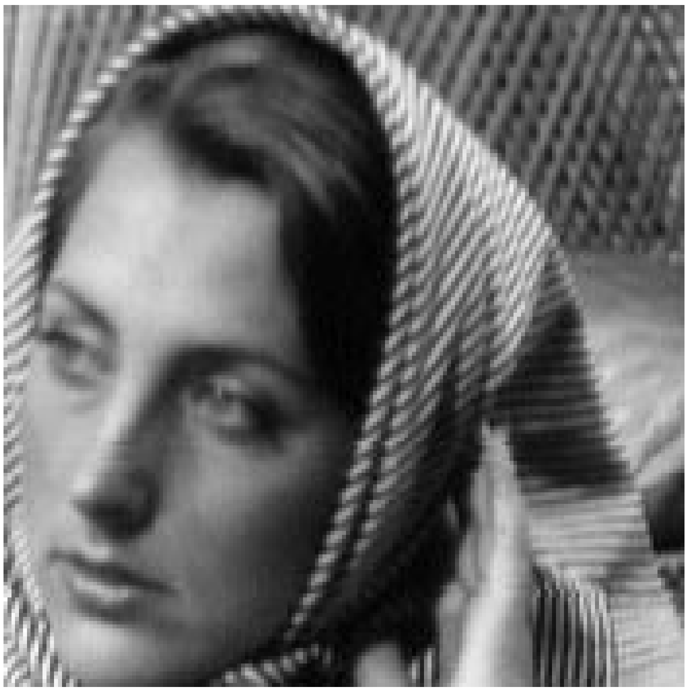


Figure B.2: *Small part of Barbara from above – Size* 175×175.

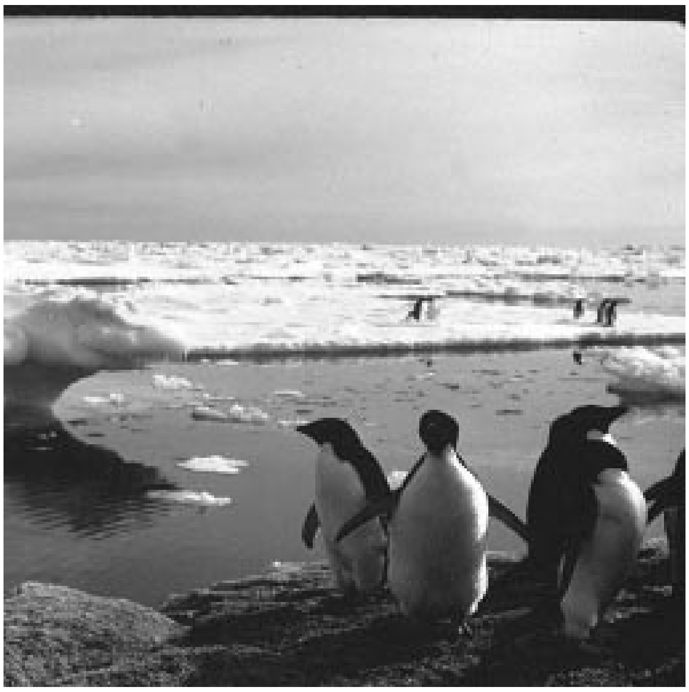


Figure B.3: *Penguins – Size* 512×512 *pixels. The original image is the Antarctic Penguins from NASA, image number AC86-0614-22.*

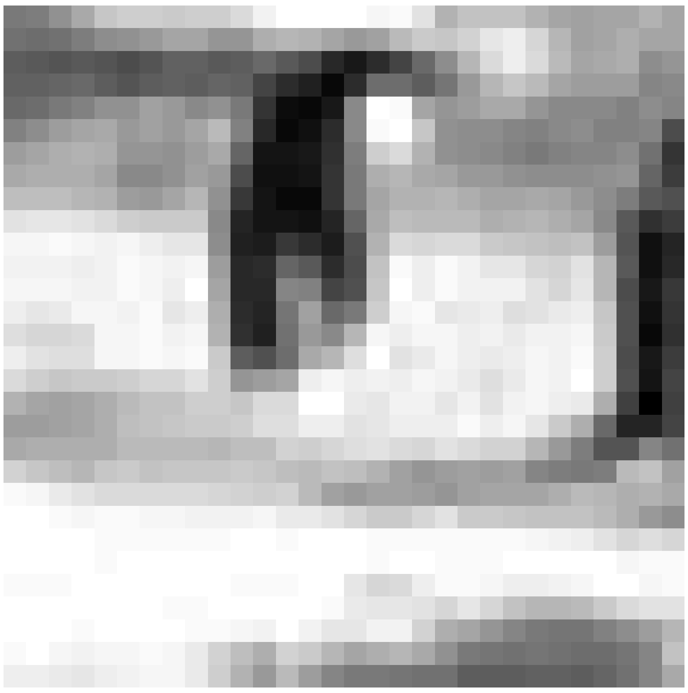


Figure B.4: *Small part of Penguins from above – Size* 30 × 30 *pixels.*

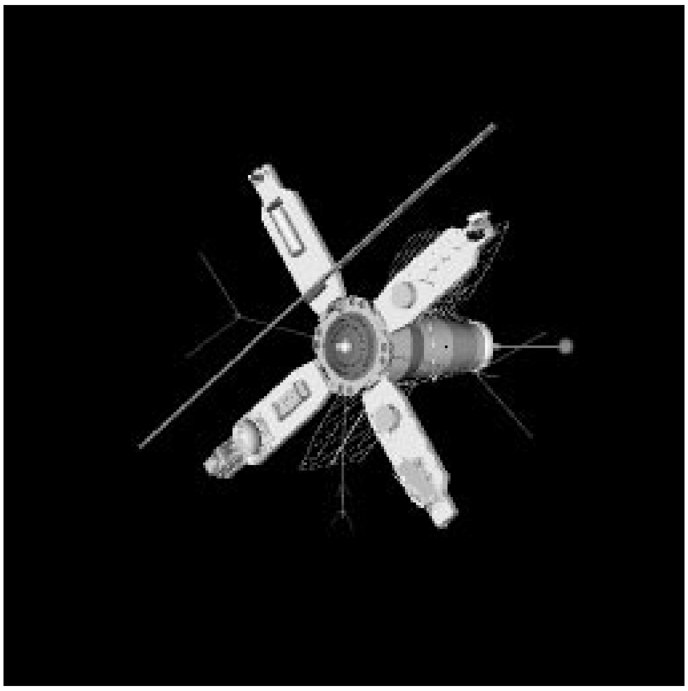


Figure B.5: *Satellite – Size* 256 × 256 *pixels. The original image can be found in the* MATLAB package Restoration Tools by J. G. Nagy [14].

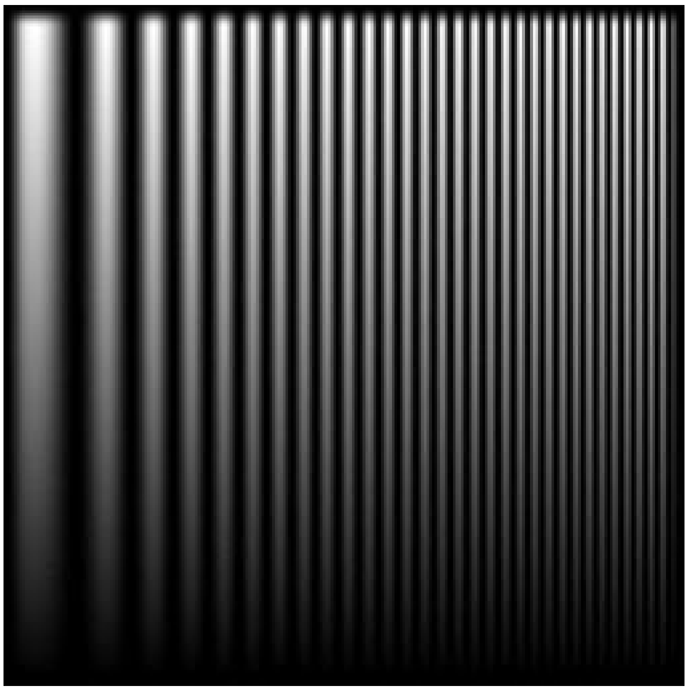


Figure B.6: *Frequency and Amplitude Sweep – Size* 256 × 256 *pixels. This image is generated by a* MATLAB function found at my homepage http://www.imm.dtu.dk/~tkj. The image is always square, and the size is variable from 32 × 32 and larger.

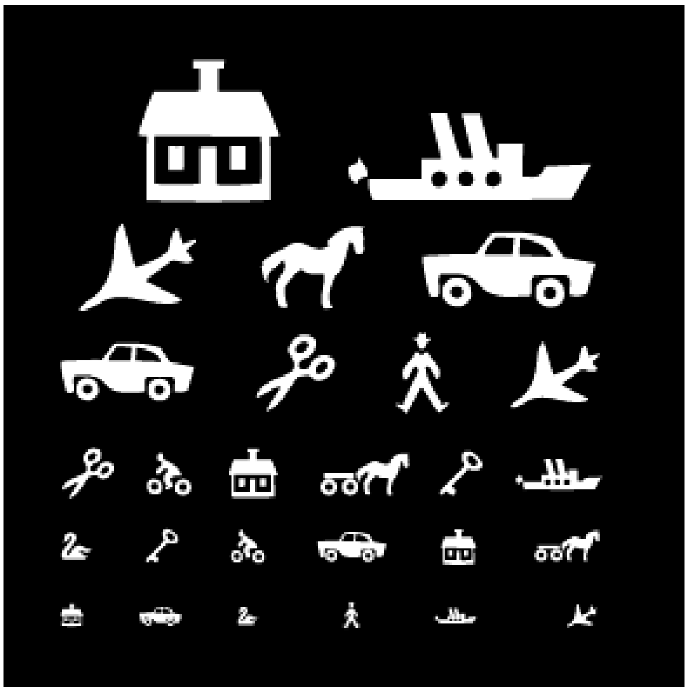


Figure B.7: *Eye test – Size* 290 × 290 *pixels. The original image was found at http://www.netsyn.dk/Netsyn/for+born.htm.*

Bibliography

[1] Å. Björck, E. Grimme, and P. van Dooren. An implicit shift bidiagonalization algorithm for ill-posed problems. *BIT*, 34:510–534, 1994.

[2] D. Calvetti, B. Lewis, and L. Reichel. GMRES, L-curves, and discrete ill-posed problems. *BIT*, 42:44–65, 2002.

[3] T. Chan. An optimal circulant preconditioner for toeplitz systems. *SIAM Journal on Scientific Computing*, 9:766–771, 1998.

[4] G. H. Golub and C. F. van Loan. *Matrix Computations - third edition*. The Johns Hopkins University Press, 1996.

[5] M. Hanke. The minimal error conjugate gradient method is a regularization method. Proceedings of the American Mathematical Society, 1995.

[6] P. C. Hansen. The L-curve and its use in the numerical treatment of inverse problems. Invited paper for P. Johnston (Ed.), *Computational Inverse Problems in Electrocardiology*, WIT Press, Southampton, 2001; pp. 119-142.

[7] P. C. Hansen. Regularization tools – a MATLAB package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, 6:1–35, 1994.

[8] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. SIAM, 1998.

[9] P. C. Hansen. Deconvolution and regularization with toeplitz matrices. *Numerical Algorithms*, 29:323–378, 2002.

[10] P. C. Hansen, M. E. Kilmer, and R. H. Keldsen. Exploring residual information in the regularization of discrete ill-posed problems. *Submitted*, 2002.

[11] I. C. F. Ipsen and C. D. Meyer. The idea behind krylov methods. *American Mathematical Monthly*, 105:889–899, 1998.

[12] M. Jacobsen. Regularization tools xp. http://www.imm.dtu.dk/˜mj/RegToolsXP/.

[13] A. K. Louis. A unified approach to regularization methods for linear ill-posed problems. *Inverse Problems*, 15:489–498, 1999.

[14] J. G. Nagy. Restoration tools – an object oriented MATLAB package for image restoration. http://www.mathcs.emory.edu/˜nagy/RestoreTools/index.html.

[15] J. G. Nagy and D. P. O'Leary. Restoring images degraded by spatially-variant blur. *SIAM Journal on Scientific and Statistical Computing*, 19:1063–1082, 1998.

[16] J. G. Nagy and Robert J. Plemmons. Iterative image restoration using FFT-based preconditioners. Proceedings to the 30^{th} Annual Allerton Conf. on Communication, Control and Computing, University of Illinois at Urbana-Champaign, 1992.

[17] N. B. Nill. Visual model weighted cosine transform for image compression and quality assessment. *IEEE Transactions of Communications*, COM-33:551–557, 1985.

[18] J. M. Rasmussen. Compact linear operators and krylov subspace methods. Master's thesis, The Technical University of Denmark, 2001.

[19] G. Rodriguez and D. Theis. An algorithm for estimating the optimal regularization parameter by the L-curve. *Submitted*, 2003.

[20] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14:461–469, 1993.

[21] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.