Regularization Using a Parameterized Trust Region Subproblem

by

Oleg Grodzevich

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We present a new method for regularization of ill-conditioned problems that extends the traditional trust-region approach. Ill-conditioned problems arise, for example, in image restoration or mathematical processing of medical data, and involve matrices that are very ill-conditioned. The method makes use of the L-curve and L-curve maximum curvature criterion as a strategy recently proposed to find a good regularization parameter. We describe the method and show its application to an image restoration problem. We also provide a MATLAB code for the algorithm. Finally, a comparison to the CGLS approach is given and analyzed, and future research directions are proposed.

Acknowledgements

I would like to thank my supervisor, Henry Wolkowicz for his support, assistance and advices during my studies. I would like to thank Arkadii Nemirovskii for his suggestions and many discussions I have greatly benefited from. I gratefully acknowledge the time Etienne De Klerk and Edward Vrscay spent on reviewing this work. I am also grateful to Urs von Matt who provided me with GCV MATLAB code which was helpful while developing the algorithm.

Finally, I have to thank my parents and my wife for their understanding, patience and encouragement.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 What is regularization?

Regularization centers on finding approximate solutions for least-squares problems such as

$$\min_x \|Gx - d\|_2 \,, \tag{1.1}$$

where G is a singular or ill-conditioned *forward operator* and d is a vector of *observed data*. This problem arises from mathematical models $Gx = d$, where the data contains noise η,

$$Gx = Gx_{\text{true}} + \eta = d = d_{\text{true}} + \eta.$$

It is remarkable that, for many applications, a small amount of noise η results in a solution x that has no relation to x_{true}, i.e. we can make the size of the error $\|\eta\|_2$ arbitrarily small, while the size of the error in the solution $\|x - x_{\text{true}}\|_2$ is arbitrarily large. Moreover, in the G singular case, there can be no solution or an infinite number of solutions x_{true}. (See e.g. the survey article [25] or the book [1].) Here we restrict G to being a square $n \times n$ matrix. The least-squares problem (1.1) typically arises from discretizations of linear equations in infinite dimensional spaces, e.g. $Tx = d$, where T is typically a compact operator and so has an unbounded inverse. This means that x is not a continuous function of the data d. Such problems are called *ill-posed* [15, 16].

To obtain meaningful solutions to the mathematical model one often uses various methods of *regularization*. The aim is to find algorithms for constructing *generalized solutions*

that are stable under small changes in the data d. One method uses the solution of the constrained least-squares problem:

$$\begin{aligned} \min \quad & \|Gx - d\|_2 \\ \text{subject to} \quad & \|x\|_2 \leq \varepsilon. \end{aligned} \tag{1.2}$$

The restriction on $\|x\|_2$ results in a larger residual error $\|Gx - d\|_2$ but reduces the propagated data error. As ε increases we reduce $\|Gx(\varepsilon) - d\|_2$ and expect $x(\varepsilon)$ to approximate the best least-squares solution $x_{\text{true}} = G^\dagger d_{\text{true}}$, where $G^\dagger$ denotes the Moore-Penrose generalized inverse of G. However, in practice the error propagation stays small for small ε but then eventually causes divergence of the iterates from x_{true}. (See *semiconvergence* in [23].) Regularization depends on controlling/choosing the parameter ε.

By squaring the objective and the constraint, (1.2) can be reformulated as the so-called *trust region subproblem*, TRS , e.g. [8]:

$$(\text{TRS }) \qquad \mu(A, a, \varepsilon) := \begin{aligned} \min \quad & q(x) := x^T A x - 2a^T x \\ \text{subject to} \quad & \|x\|_2^2 \leq \varepsilon^2, \end{aligned}$$

where $A := G^T G$ is $n \times n$ (we assume $n \geq 2$) symmetric, $a := G^T d$ is an n-vector, ε is a *positive* scalar, and x is the n-vector of unknowns. All matrix and vector entries are real.

In this thesis, we apply known results for TRS to efficiently control the parameter ε and find regularized solutions of (1.1). We also compare our approach to the Conjugate Gradients method, which is often used for regularization.

1.2 Contributions

This thesis extends the traditional trust-region approach for regularization of ill-conditioned problems. We show that this is an effective tool that can be used in conjuction with the L-curve maximum curvature criterion.

Unlike the traditional TRS with a fixed trust region radius, here ε changes at each iteration to get a new point on the L-curve, thus acting as a regularization parameter. We reveal the relations between various TRS parameters and employ them to efficiently guide the

algorithm along the L-curve. As a result, we require very few iterations to get to the *elbow* (to be defined below). Furthermore each iteration is accelerated by using the data from the previous step.

In comparison to [9] we use a more robust way of choosing/controlling the regularization parameters. We explicitly compute the curvature and provide a more reliable method to determine the location of the *elbow*, and we do not require an *apriori* knowledge of the norm of the noise to estimate the initial (starting) point.

1.3   Applications

Many problems in the mathematical sciences have solutions that are unstable with respect to the initial data. Classical examples include differentiation of functions known only approximately, solutions of integral equations of the first kind, and solution of singular or ill-conditioned linear equations. These examples arise in mathematical processing and interpretation of data in various fields, e.g.

- geophysical: determine an earthquake hypocenter in space and time, vertical seismic profiling and wave propagation;

- medical: computer-assisted tomography (CAT), magnetic resonance imaging and magnetoencephalography (MRI, MEG);

- imaging: deconvolution of telescope images and image restoration.

In Section 5.2 we consider an image restoration example: deblurring of an image. This is a typical problem in astrophotography. Pictures taken by the ground telescopes are subject to atmospheric blur and require the restoration procedure (see e.g. the forthcoming book [35]). We observe the difference between the least-squares and regularized solutions, and illustrate the performance of the algorithm for this type of problem.

Chapter 2

Basic Regularization Theory

2.1 Tikhonov Regularization

Regularization dates back to work by Tikhonov [33]. (See also [34].) For the equation $Tx = d$, one solves the damped normal equation

$$(T^*T + \alpha^2 I)x_\alpha = T^*d, \tag{2.1}$$

where $d = d_{\text{true}} + \eta$. In this thesis we restrict our analysis to a finite-dimensional discretization of an operator T, represented by a matrix G. We replace (2.1) by

$$(G^T G + \alpha^2 I)x_\alpha = G^T d. \tag{2.2}$$

Regularization involves choosing the correct value for the parameter $\alpha > 0$, when given some information on the size of the error η. If $\alpha = 0$, (2.2) degenerates to the normal equations for the linear least-squares problem.

The regularization is equivalent to choosing the correct value for ε in (1.2). See Remark 3.1.1.

Moreover, a solution to (2.2) is also a solution to

$$\min_x \|Gx - d\|_2^2 + \alpha^2 \|x\|_2^2. \tag{2.3}$$

To see this denote

$$G_{\text{ext}} = \begin{bmatrix} G \\ \alpha I \end{bmatrix}, \ d_{\text{ext}} = \begin{bmatrix} d \\ 0 \end{bmatrix}.$$

We can now rewrite (2.2) as

$$G_{\text{ext}}^T G_{\text{ext}} x_\alpha = G_{\text{ext}}^T d_{\text{ext}}. \tag{2.4}$$

Since $\alpha > 0$, G_{ext} is full-rank then (2.4) are the normal equations for the problem

$$\min_x \left\| \begin{bmatrix} G \\ \alpha I \end{bmatrix} x - \begin{bmatrix} d \\ 0 \end{bmatrix} \right\|_2^2, \ \alpha > 0,$$

which is equivalent to (2.3).

2.2   Using Singular Values

The singular value decomposition (SVD) of the matrix G is a tool that helps in understanding the L-curve analysis (see Section 2.3). We will write the SVD as

$$G = USV^T,$$

where matrix S is a diagonal $n \times n$ matrix consisting of singular values σ_i of G, $\sigma_1 \leq \ldots \leq \sigma_n$, and U, V are orthogonal matrices, i.e.

$$U^T U = I, \ V^T V = I.$$

We can characterize the Tikhonov regularized solution x_α using the SVD in the following way. Substitute the SVD of the matrix G into (2.2):

$$\begin{aligned} (G^T G + \alpha^2 I) x_\alpha &= G^T d \\ (V S U^T U S V^T + \alpha^2 I) x_\alpha &= V S U^T d \\ V(S^2 + \alpha^2 I) V^T x_\alpha &= V S U^T d \\ V^T x_\alpha &= (S^2 + \alpha^2 I)^{-1} S U^T d. \end{aligned}$$

Finally using the orthogonality of the matrices U and V we get

$$x_\alpha = V(S^2 + \alpha^2 I)^{-1} S U^T d = \sum_{i=1}^n f_i \frac{U_{:i}^T d}{\sigma_i} V_{:i} \tag{2.5}$$

5

and

$$\|x_\alpha\|_2^2 = d^T U(S(S^2 + \alpha^2 I))^2 U^T d = \sum_{i=1}^n \left(f_i \frac{U_{:i}^T d}{\sigma_i} \right)^2, \tag{2.6}$$

where $U_{:i}$ denotes the ith column of U.

Similarly,

$$d - Gx_\alpha = d - USV^T x_\alpha = U(I - S(S^2 + \alpha^2 I)^{-1} S) U^T d,$$

$$\|Gx_\alpha - d\|_2^2 = \sum_{i=1}^n \left((1 - f_i) U_{:i}^T d \right)^2, \tag{2.7}$$

where the f_i are the so called Tikhonov filter factors, defined as

$$f_i = \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2}. \tag{2.8}$$

Note that if G is invertible, then setting $\alpha = 0$ gives the true least-squares solution x_0 with the norm of the residual $\|Gx_0 - d\|_2 = 0$, as all filter factors are equal to one. This means that the solution x_α for $\alpha > 0$ should always have a norm smaller than $\|x_0\|$, since the SVD components corresponding to the small singular values are filtered by f_i. It also follows that choosing the value of α^2 larger than σ_n^2 is unreasonable, since the factors f_i are small and the corresponding solution x_α would have an almost zero norm.

Expressions (2.5), (2.6) and (2.7) can also be used to illustrate what happens to the solution in the presence of noise. Consider first of all, the least-squares solution x_0 for the "true" right-hand side d_{true}, i.e. without any noise. Then, since $d_{\text{true}} = Gx_{\text{true}} = USV^T x_{\text{true}}$, we have $x_0 = V(S^2)^{-1} S^2 V^T x_{\text{true}}$. If we further assume that the matrix G, and hence S, is invertible then $x_0 = x_{\text{true}}$. However adding uncorrelated noise η would result in extra contribution to the solution caused by the noise components. Specifically, $f_i = 1, \forall i$, and

$$\|x_0\|_2^2 = \sum_{i=1}^n \left(\frac{U_{:i}^T d_{\text{true}}}{\sigma_i} + \frac{U_{:i}^T \eta}{\sigma_i} \right)^2.$$

It is easy to see that these contributions can be very large in the case of small singular values whenever the noise vector is not orthogonal to the corresponding singular vectors, $U_{:i}$'s. This explains why the naive least-squares solution is not meaningful and a regularized solution should be sought instead.

6

The situation continues to be problematic if the matrix G has very small singular values in a sense that they are numerically close to 0, even if the noise component is absent. This can be observed by looking at the ratio $\frac{U_{:i}^T d_{\text{true}}}{\sigma_i}$. We require that

$$\boxed{\text{the Fourier coefficients } |U_{:i}^T d_{\text{true}}| \text{ decay faster than the } \sigma_i.}$$

This condition, also known as *the Discrete Picard Condition*, e.g. see [21], guarantees that the least-squares solution has a reasonable norm and thus is physically meaningful. However, if the σ_i's become smaller than machine epsilon, i.e. the smallest number we can numerically operate with, the Picard condition fails. The next example illustrates this situation.
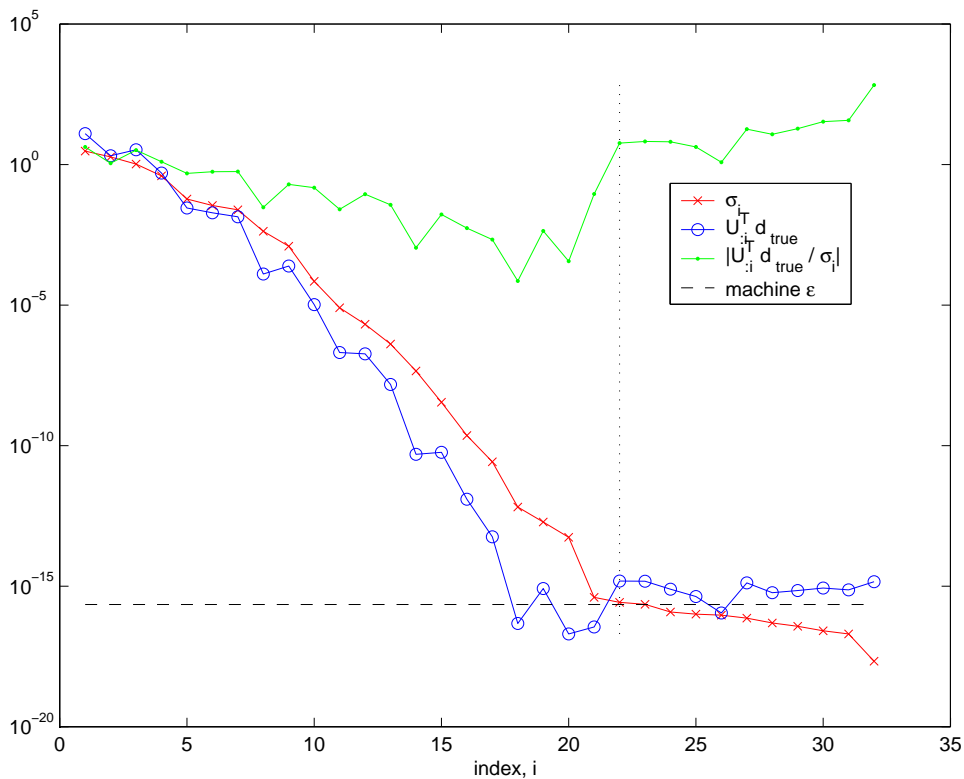
Figure 2.1: Picard plot for a Shaw problem

7

Example 2.2.1. *We consider a Shaw problem from the Hansen MATLAB package (see [19]) with $n = 32$. This is a one-dimensional image restoration problem which is constructed via discretization of a Fredholm integral equation of the first kind (see [29]). The MATLAB* **shaw** *command produces the matrix G and the right-hand side vector d_{true}, as well as the true solution vector x_{true}.*

We then compute the SVD of the matrix G and plot the Fourier coefficients $|U_{:i}^T d_{\text{true}}|$, the singular values σ_i and the ratio $\frac{|U_{:i}^T d_{\text{true}}|}{\sigma_i}$. The resulting plot is Figure 2.1. It can be seen that the Picard condition holds until the singular values (x marked line) hit the machine epsilon level (horizontal dashed line). But for the larger indices, round-off error steps in and the Picard condition fails. The norm of the least-squares solution computed via SVD, i.e. by using (2.6), is $\sim 10^5$ while the true solution has norm of ~ 10. A good approximation of the true solution is still recoverable via a truncated SVD, i.e. by setting to 0 all the singular values less than machine epsilon.

2.3 The L-curve analysis

As we have seen in Sections 2.1 and 2.2, finding the regularized solution involves finding the regularization parameter. In this thesis, we use the approach of studying the correspondence between the norm of the solution and the norm of the residual to obtain the value of the regularization parameter α. Such a relation can be naturally expressed as a plot of one of these quantities versus another, i.e. as a log-log curve based on

$$\Big(\log(\|Gx_\alpha - d\|_2), \log(\|x_\alpha\|_2) \Big).$$

In the literature (see [21] for example), this plot is often referred to as the *L-curve*. The curve usually features a strong L-shaped form with almost linear vertical and horizontal parts and a well distinguishable *elbow* or *corner*. Although for most practical problems its form is L-shaped, it may vary depending on the structure of the problem. Basing on the analysis presented in Section 2.2, we give an overview of the L-curve characteristics.

The results presented in this thesis use a nonstandard way of plotting the L-curve, i.e. the abscissa represents $\log(\|x_\alpha\|_2)$, rather than the traditional orientation which uses the residual instead. We choose a different view because our analysis centers on changing

the trust region radius ε, and it is more convinient to have a parameter of interest as an abscissa of the plot.

Recall the expressions (2.7) and (2.6) for the norms of the residual and the solution. It is worthwhile to rewrite the latter as

$$\|x_\alpha\|_2^2 = \sum_{i=1}^{n} \left[f_i \left(\frac{U_{:i}^T d_{\text{true}}}{\sigma_i} + \frac{U_{:i}^T \eta}{\sigma_i} \right) \right]^2.$$

If we assume uncorrelated noise, then the expected value of the Fourier coefficients of η should satisfy

$$\mathcal{E}\left(|U_{:i}^T \eta| \right) \approx \|\eta\|_2 , \forall i.$$

This means that the noise does not satisfy the Picard condition when there are small singular values. Furthermore, the Fourier coefficients of perturbed data should eventually become larger than the corresponding singular values, even if the original data satisfies the Picard condition. This happens, roughly, once the Fourier coefficients, corresponding to the true unperturbed data, become dominated by the Fourier coefficients from the noise.

We illustrate our considerations by means of an example - a deblurring of a 20×20 image. The details of how the problem data is constructed are outlined in Section 5.2. Figure 2.2 shows the Picard plot for the unperturbed right-hand side. It is evident that on average the Fourier coefficients corresponding to the unperturbed data vector decay faster than the singular values. Hence, the Picard condition holds and the least-squares solution recovers the true solution in the absence of the noise.

Then we build a random vector η to represent the noise. The Fourier coefficients for η are plotted in Figure 2.3. We can see that on average they stay on the same level and hence fail to satisfy the Picard condition. As expected, the Picard plot for the perturbed (noisy) right-hand side levels off at approximately $\|\eta\|_2$ as shown in Figure 2.4.

Now we no longer restrict $f_i = 1$ and start looking at solutions x_α corresponding to the different values of the regularization parameter α. Since

$$f_i \simeq \begin{cases} 1, & \sigma_i \gg \alpha \\ \frac{\sigma_i^2}{\alpha^2}, & \sigma_i \ll \alpha, \end{cases}$$
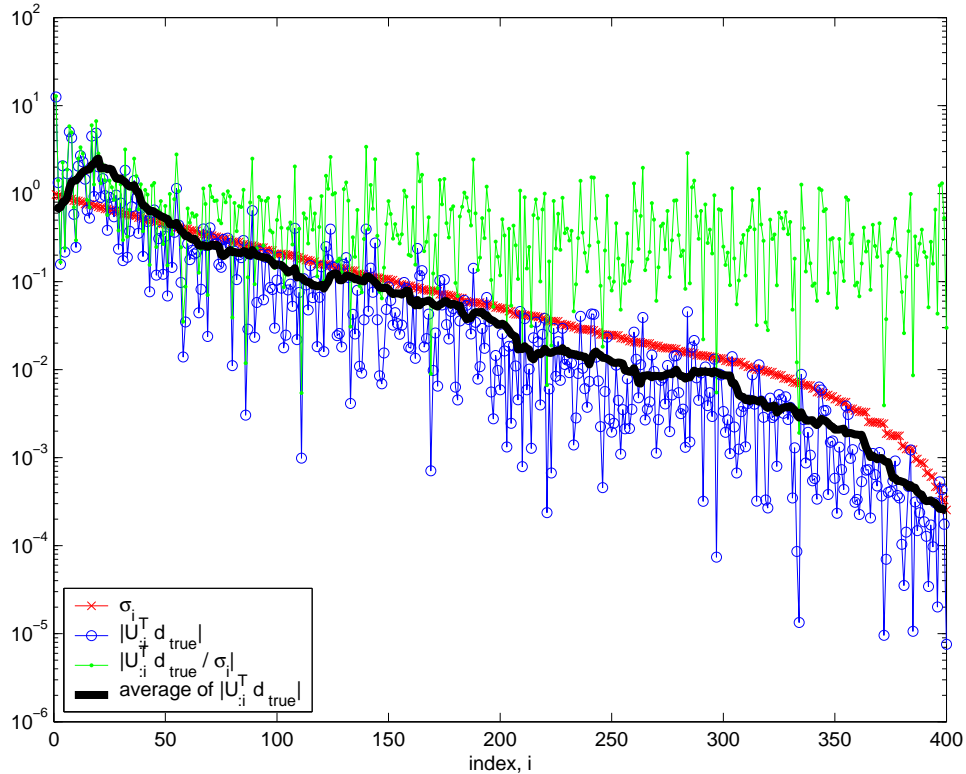
Figure 2.2: Picard plot for the unperturbed right-hand side

the filter factors control which terms in the summation contribute to the norm of the residual and the solution. Figure 2.4 demonstrates that when the regularization parameter corresponds to the larger singular values, the norm of the residual varies greatly with α, but the norm of the solution is almost unaffected, since all the terms corresponding to the smaller singular values are filtered (this is also known as *oversmoothing* a solution). This situation gives rise to the vertical part of the L-curve. On the other hand, when α is small the norm of the residual does not change much, but small changes in the regularization parameter cause a dramatic change in the norm of the solution because the noise does not satisfy the Picard condition. This corresponds to the horizontal part. Depending on the particular Picard plot, the smoothness of the transition between the vertical and horizontal parts can vary in a broad range. For example, the L-curve for the deblurring problem is

10
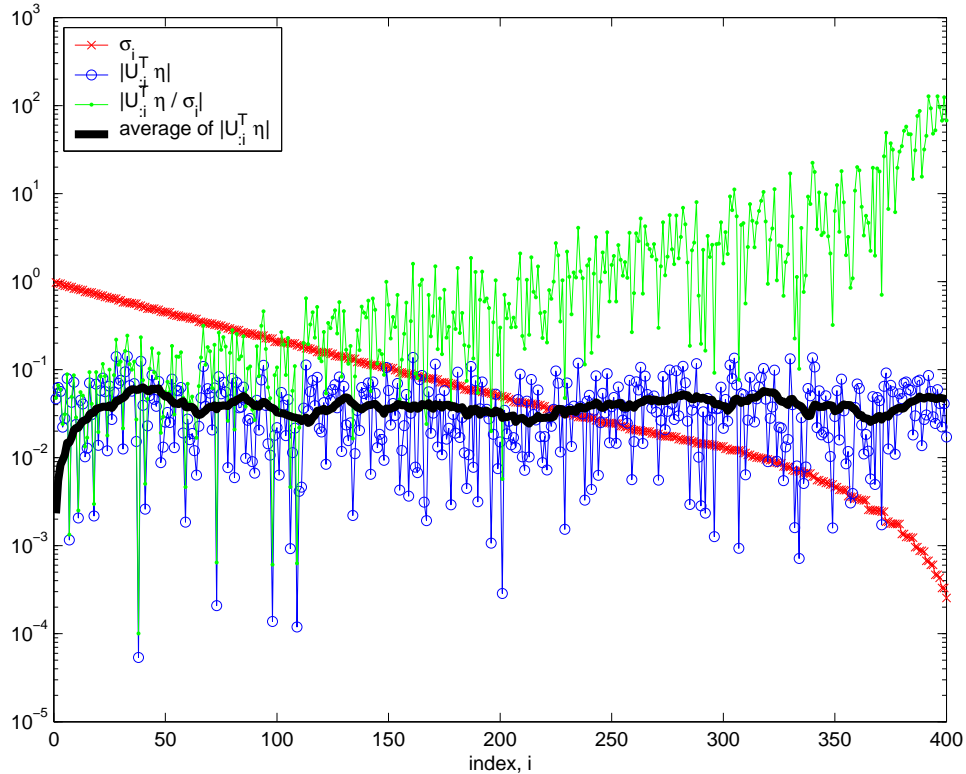
Figure 2.3: Picard plot for the noise vector

presented in Figure 2.5. It is not strongly L-shaped, but it is still possible to locate a distinguishable *elbow*. However, this discussion is only relevant when the log-log scale is used. In linear scale the plot is always convex, e.g. see [20].

This kind of behaviour and the existence of a distinct *elbow* leads to a strategy for choosing the regularization parameter known as the *L-curve criterion* (proposed in [18, 22]). The idea is to choose the value of the parameter that corresponds to a point on the L-curve with maximum curvature (details on curvature calculation are given in Sections 3.6 and 3.7). Due to the L-shaped form, a point of maximum curvature coincides with an *elbow* that, by above discussion, separates the regions where the solution is dominated by regularization errors (oversmoothing) and perturbation errors respectively.

We continue the analysis by looking at how the noise affects the problem. We illustrate that

11
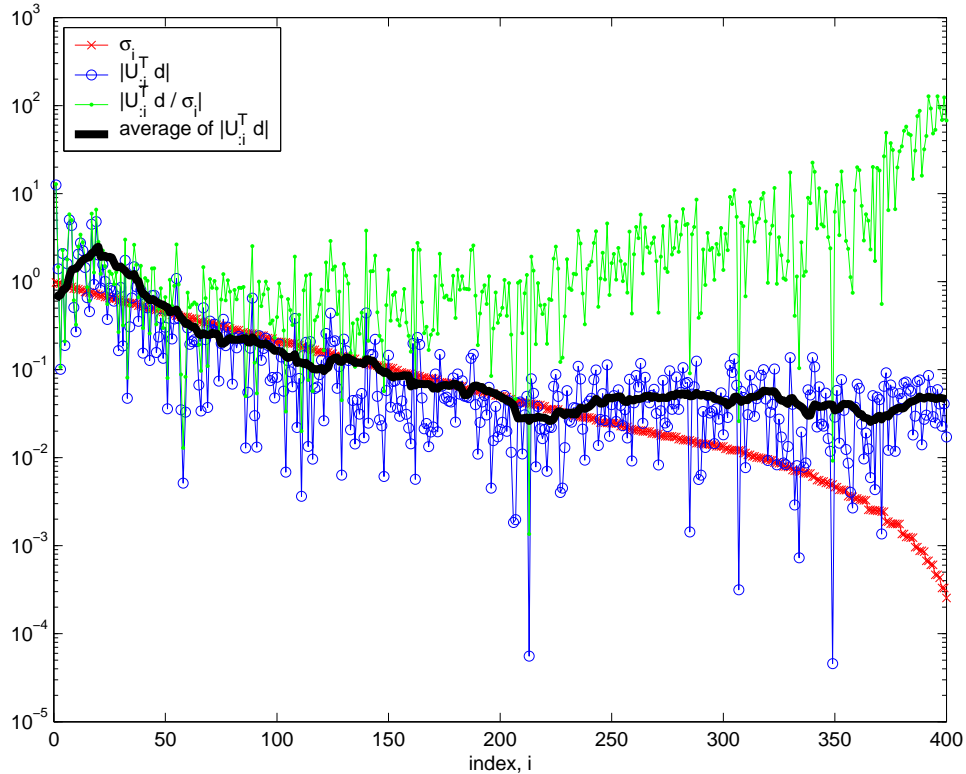
Figure 2.4: Picard plot for the perturbed right-hand side

even in the presence of noise, the least-squares solution can still be a good approximation to the true solution, providing that the noise vector is specifically chosen. We run a series of problems each with a different noise vector but the same matrix G and d_{true}. For every instance k we construct η such that it lies in the span of the first k singular vectors $U_{:i}$, i.e.

$$\eta_k = \sum_{i=1}^{k} r_i U_{:i},$$

where r is a random vector of the norm ~ 1. We then solve each instance of the problem $\min_x \left\| Gx^k - d_{\text{true}} - \eta_k \right\|_2$ for the least-squares solution x_0^k. Finally, we compare the results towards the true solution by computing the relative accuracy

$$\text{rel.acc.}_k = \frac{\left\| x_{\text{true}} - x_0^k \right\|_2}{\left\| x_{\text{true}} \right\|_2}.$$
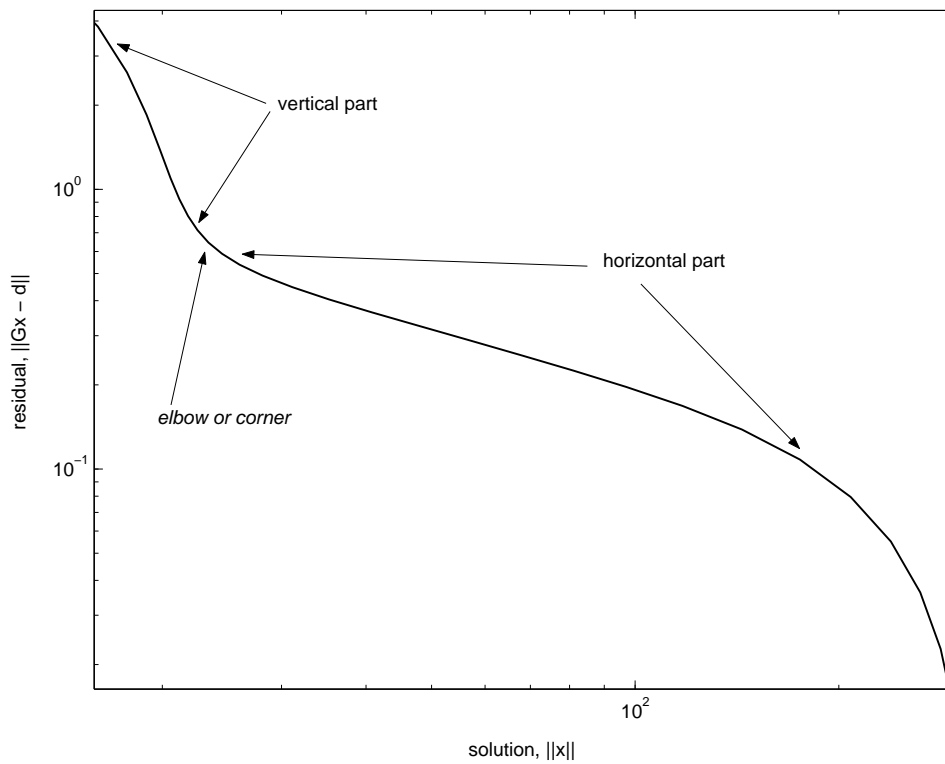
Figure 2.5: The L-curve for the deblurring problem

Figure 2.6 presents the results. Comparing it to the Picard plot on Figure 2.4, it is easy to notice that once the noise vector starts including singular vectors for which the Picard condition fails, the norm of the least-squares solution starts growing and it is no longer a good approximation.
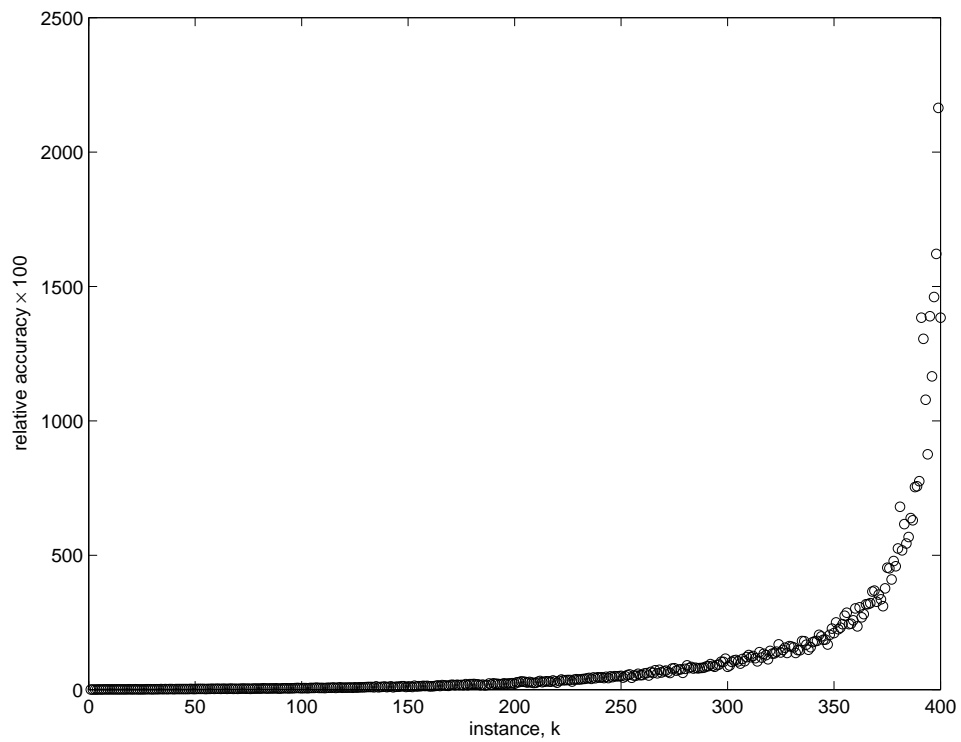
Figure 2.6: Relative accuracy for different noise vectors

Chapter 3

Regularization Using TRS

In this chapter, we discuss several results from TRS theory, that help in analyzing the L-curve behaviour. We consider the optimality conditions for TRS and the behaviour of the optimal objective value under small changes of the trust region radius. We further derive the results developed in the Rendl-Wolkowicz TRS algorithm and apply them to formulate the regularization as a one-dimensional parameterized problem. We show that the curvature of the L-curve can be efficiently computed for each point visited by the TRS solver.

3.1 The Optimality Conditions

It is known ([11, 30]) that x^* is a solution to TRS if and only if:

$$
\begin{array}{ll}
\left.\begin{array}{l}
(A - \lambda^* I)x^* = a, \\
A - \lambda^* I \succeq 0, \lambda^* \leq 0
\end{array}\right\} & \text{dual feasibility} \\
\quad \|x^*\|^2 \leq \varepsilon^2 & \text{primal feasibility} \\
\lambda^*(\|x^*\|^2 - \varepsilon^2) = 0 & \text{complementary slackness}
\end{array}
\tag{3.1}
$$

for some (Lagrange multiplier) λ^*. As shown in Section 2.1, using the above conditions allows us to relate the regularization in the sense of Tikhonov with TRS . Also note that in the scope of this thesis, i.e. applied to the regularization, we may restrict $\lambda^* < 0$, which corresponds to the restriction on the Tikhonov regularization parameter $\alpha^2 > 0$. This leads

to two very important consequences. First of all, the optimal solution always lies on the boundary, i.e. $\|x^*\|_2 = \varepsilon$. Secondly, the so-called *easy case* holds for TRS . The *easy case* corresponds to $a \not\perp \mathcal{N}(A - \lambda_1(A)I)$, where $\lambda_1(A)$ is the smallest eigenvalue of the matrix A. And this condition is implied by $\lambda^* < 0 \leq \lambda_1(A)$.

Remark 3.1.1. *The optimality conditions* (3.1) *imply that solving* (2.2) *with a particular value of the regularization parameter* α *is equivalent to solving* (1.2) *with a corresponding value of* ε. *This can be seen from a Lagrange multiplier argument. Since* $A = G^T G$ *and* $a = G^T d$, *we have* $(G^T G - \lambda I)x = G^T d$. *This, however, is equivalent to* (2.2) *with* $\lambda = -\alpha^2$. *Fixing* λ *yields a solution* x *such that* $\varepsilon = \|x\|_2$, *due to* $\lambda(\|x\|_2^2 - \varepsilon^2) = 0$. *Hence, for every choice of* $\alpha^2 > 0$ *we can find a corresponding value of* ε, *such that the solutions to both* (2.2) *and* (1.2) *coincide.*

3.2   Perturbations $\Delta\varepsilon$ for μ

We keep the data A, a fixed and consider the optimal value as a function of $\varepsilon > 0$. By abuse of notation, we write $\mu_\varepsilon = \mu(A, a, \varepsilon)$. We now derive the expressions for first- and second-order derivatives of μ_ε with respect to ε.

We assume (as in Section 3.1) that the *easy case* holds, and that the optimum point lies on the boundary of the feasible region, i.e. $\|x^*\| = \varepsilon$.

$$
\begin{aligned}
\mu_\varepsilon &= (x^*)^T A x^* - 2a^T x^* \\
&= (x^*)^T A x^* - 2a^T x^* - \lambda^*(\|x^*\|^2 - \varepsilon^2) \\
&= (x^*)^T (A - \lambda^* I) x^* - 2a^T x^* + \lambda^* \varepsilon^2 \\
&= a^T (A - \lambda^* I)^{-1} a - 2a^T (A - \lambda^* I)^{-1} a + \lambda^* \varepsilon^2 \\
&= -a^T (A - \lambda^* I)^{-1} a + \lambda^* \varepsilon^2.
\end{aligned}
$$

Then, using $a^T (A - \lambda^* I)^{-2} a - \varepsilon^2 = \|x^*\|^2 - \varepsilon^2 = 0$, we get

$$
\begin{aligned}
\frac{\partial \mu_\varepsilon}{\partial \varepsilon} &= a^T (A - \lambda^* I)^{-2} a \left(-\frac{\partial \lambda^*}{\partial \varepsilon}\right) + \left(\frac{\partial \lambda^*}{\partial \varepsilon}\right) \varepsilon^2 + 2\lambda^* \varepsilon \\
&= \left(-\frac{\partial \lambda^*}{\partial \varepsilon}\right) \left(a^T (A - \lambda^* I)^{-2} a - \varepsilon^2\right) + 2\lambda^* \varepsilon \\
&= 2\lambda^* \varepsilon
\end{aligned}
\tag{3.2}
$$

and

$$\frac{\partial^2 \mu_\varepsilon}{\partial \varepsilon^2} = 2 \left(\lambda^* + \varepsilon \frac{\partial \lambda^*}{\partial \varepsilon} \right). \tag{3.3}$$

The derivative $\frac{\partial \lambda^*}{\partial \varepsilon}$ can be found using implicit differentiation in $\left\| (A - \lambda^* I)^{-1} a \right\|^2 - \varepsilon^2 = 0$, which is obtained after substituting $x^* = (A - \lambda^* I)^{-1} a$. Namely:

$$a^T (A - \lambda^* I)^{-2} a = \varepsilon^2 \ ;$$
$$2 \left(\tfrac{\partial \lambda^*}{\partial \varepsilon} \right) a^T (A - \lambda^* I)^{-3} a = 2\varepsilon \ ;$$

and

$$\frac{\partial \lambda^*}{\partial \varepsilon} = \frac{\varepsilon}{a^T (A - \lambda^* I)^{-3} a}. \tag{3.4}$$

More details on these and other perturbation results can also be found in [32].

3.3   Applying TRS in L-curve Analysis

As Section 2.3 suggests, one can obtain a good regularization parameter α by looking at the point of the maximum curvature on the L-curve. One way to locate this point is to sequentially solve a number of *trust region subproblems*, while gradually changing the trust region radius. This approach, however, is not very efficient and does not fully exploit the nature of the problem. To see why this is the case, we need to go into more details here.

As we have seen in Section 2.1, a solution to the regularization problem coincides with the optimal solution to TRS . The former is identified by the value of the trust region radius ε, which poses the bound on the norm of the solution. Moreover, as we will see later, the inequality holds with equality for a certain interval of $\varepsilon \in (0, \left\| G^T d \right\|_2)$. Hence, for the L-curve analysis, the trust region radius specifies one of the coordinates of an L-curve point, and another coordinate is given by the objective value of TRS . Thus, every solution to TRS can be associated with a unique point on the L-curve and vice versa. However, as we have already noted, this correspondence only holds inside a certain interval of ε.

Solving TRS usually involves going through an iterative procedure which at each step produces a solution x that is optimal to TRS with different, but close, trust region radius
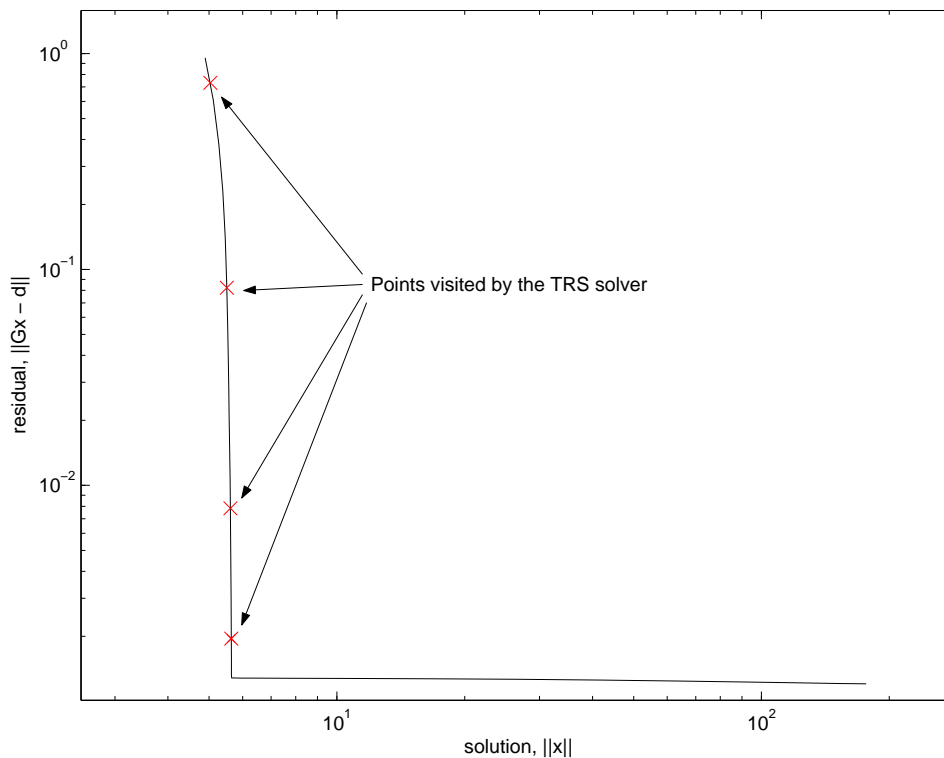
Figure 3.1: Points encountered while solving TRS

ε. It means that at each step of such a procedure we encounter an L-curve point which is, however, thrown away if the TRS solver is used as a blackbox.

Figure 3.1 illustrates what happens while solving TRS by the Rendl-Wolkowicz algorithm. The figure presents an L-curve for a sample Shaw problem created using the Hansen MAT-LAB package (see [19]). The TRS algorithm was then executed with the generated data, a fixed trust region radius $\varepsilon = 6$ and a desired tolerance $\delta = 10^{-8}$ (this point lies somewhere near the *elbow*). It took 8 iterations to complete, i.e. to solve this trust region subproblem. An L-curve point was analyzed at every iteration. On the figure, one may see 4 such points marked by x (the other 4 are located outside the interval of interest). It is not hard to notice that these 4 points give enough information to approximate the vertical part of the L-curve to the left of the *elbow*. To make use of this information efficiently, we modify the TRS solver to be able to vary and control the trust region radius as we iterate towards the

solution. We want each point, that we find on the L-curve, to be important in locating the *elbow*.

We now present the Rendl-Wolkowicz TRS algorithm along with techniques developed and discussed in [26].

Exploiting the strong Lagrangian duality of TRS (see [31]), we can show that it (TRS) can be reformulated as an unconstrained concave maximization problem. As shown in [31] strong duality holds for TRS with no duality gap, i.e.

$$\mu_\varepsilon = \min_x max_\lambda L(x, \lambda) = \max_\lambda \min_x L(x, \lambda),$$

where $L(x, \lambda)$ denotes the *Lagrangian* of TRS ,

$$L(x, \lambda) = x^T A x - 2a^T x + \lambda(\|x\|^2 - \varepsilon^2).$$

Then

$$
\begin{aligned}
\mu_\varepsilon &= \min_{\|x\|=\varepsilon, y_0^2=1} x^T A x - 2y_0 a^T x \\
&= \max_t \min_{\|x\|=\varepsilon, y_0^2=1} x^T A x - 2y_0 a^T x + ty_0^2 - t \\
&\geq \max_t \min_{\|x\|^2+y_0^2=\varepsilon^2+1} x^T A x - 2y_0 a^T x + ty_0^2 - t \\
&\geq \max_{t,\lambda} \min_{x,y_0} x^T A x - 2y_0 a^T x + ty_0^2 - t + \lambda(\|x\|^2 + y_0^2 - \varepsilon^2 - 1) \\
&= \max_{r=t+\lambda,\lambda} \min_{x,y_0} x^T A x - 2y_0 a^T x + r_0^2 - r + \lambda(\|x\|^2 - \varepsilon^2) \\
&= \max_\lambda \big(\max_r \min_{x,y_0} x^T A x - 2y_0 a^T x + r_0^2 - r + \lambda(\|x\|^2 - \varepsilon^2)\big) \\
&= \max_\lambda \min_{x,y_0^2=1} x^T A x - 2y_0 a^T x + \lambda(\|x\|^2 - \varepsilon^2) \\
&= \mu_\varepsilon,
\end{aligned}
$$

where the strong duality and the symmetry of the function are used for the last two equalities.

We define

$$k(t) = (\varepsilon^2 + 1)\lambda_1(D(t)) - t, \ t \in \mathbb{R}, \tag{3.5}$$

where $D(t)$ is the symmetric $(n+1) \times (n+1)$ matrix

$$D(t) = \begin{bmatrix} t & -a^T \\ -a & A \end{bmatrix},$$ (3.6)

and λ_1 denotes the smallest eigenvalue. Then the third expression in the above chain can be written as

$$\min_{\|x\|^2+y_0^2=\varepsilon^2+1} x^T A x - 2y_0 a^T x + t y_0^2 - t =$$

$$= \min_{\|x\|^2+y_0^2=\varepsilon^2+1} \begin{bmatrix} y_0 \\ x \end{bmatrix}^T \begin{bmatrix} t & -a^T \\ -a & A \end{bmatrix} \begin{bmatrix} y_0 \\ x \end{bmatrix} - t =$$

$$= (\varepsilon^2+1)\lambda_1(D(t)) - t,$$

where the last equality is obtained by using the Rayleigh quotient for the matrix $D(t)$ and the vector $[y_0 \ x]^T$.

Finally, this implies

$$\mu_\varepsilon = \max_t k(t).$$ (3.7)

Therefore, the Trust Region Subproblem can be transformed to an unconstrained concave maximization problem. Furthermore, under assumptions of the *easy case*, $\lambda_1(D(t))$ is a singleton eigenvalue, and the derivative of $k(t)$ satisfies

$$k'(t) = (\varepsilon^2+1)y_0^2 - 1,$$ (3.8)

where $\begin{pmatrix} y_0 \\ x \end{pmatrix}$ is the normalized eigenvector for $\lambda_1(D(t))$.

We focus on the function $k(t)$, instead of looking directly at μ. Furthermore, we show that the regularization problem can be expressed as a one-dimensional parameterized problem and derive bounds and relations between various controlling parameters.

3.4 Regularization as a one-dimensional parameterized problem

Consider the following parameters:

t – control parameter in $k(t), D(t)$

ε – trust-region radius, norm of the solution $\|x\|_2$

α – Tikhonov regularization parameter

λ – optimal Lagrange multiplier for TRS

As was shown in Section 2.1, there is one-to-one correspondence between α and λ providing $\lambda < 0$, namely $\lambda = -\alpha^2$. However, changing between λ, t and ε is computationally expensive and the following lemmas describe how to achieve this. The upper bounds imposed on these parameters correspond to the bound on the Tikhonov regularization parameter, $\alpha^2 > 0$, and are not crucial for the proofs. The details are discussed in Section 3.5.

Lemma 3.4.1. *Given the parameter $\lambda < 0$, the corresponding values of t and ε can be obtained so that*

$$
\begin{aligned}
t &= \lambda + d^T G (G^T G - \lambda I)^{-1} G^T d \\
\lambda_1(D(t)) &= \lambda \\
\varepsilon^2 &= d^T G (G^T G - \lambda I)^{-2} G^T d
\end{aligned}
\tag{3.9}
$$

Proof: The formula for t follows from Proposition 3.1 and Corollary 3.4 in [26].

The formula for ε follows from the optimality conditions (3.1). The optimal solution x^* to TRS , that corresponds to the Lagrange multiplier $\lambda^* = \lambda$, lies on the boundary, i.e. $\varepsilon^2 = \|x^*\|_2^2$, and satisfies

$$
x^* = (A - \lambda^* I)^{-1} a = (G^T G - \lambda I)^{-1} G^T d,
$$

since $a = G^T d$, $A = G^T G \succeq 0$ and $\lambda < 0$. ■

Lemma 3.4.2. *Given the parameter $t < d^T d$ the corresponding values of λ and ε can be obtained as:*

$$
\begin{aligned}
\lambda &= \lambda_1(D(t)) \\
\varepsilon &= \frac{\sqrt{1 - y_0(t)^2}}{y_0(t)}
\end{aligned}
\tag{3.10}
$$

where $y(t)$ is the eigenvector corresponding to $\lambda_1(D(t))$ and $y_0(t)$ is its first component.

Proof: See Theorem 3.7 in [26]. ∎

Lemma 3.4.3. *Given the parameter $\varepsilon < \left\| G^T d \right\|_2$ the corresponding values of t and λ can be obtained by solving* TRS *by Rendl-Wolkowicz algorithm and the corresponding optimal solution stays on a boundary.*

Proof: The Rendl-Wolkowicz algorithm solves TRS with a fixed trust region radius ε producing the optimal solution x^*, the optimal Lagrange multiplier λ, and the corresponding parameter t. ∎

Combining the above lemmas we can conlude that every one of t, λ, ε, α can be interchangeably used to parameterize the regularization problem.

3.5   Intervals of interest for t, λ and ε

As mentioned in the previous section, the upper bounds on t, λ and ε correspond to the bound on $\alpha^2 > 0$. We show now, that when the parameters are equal to their corresponding upper bounds, the optimal solution to TRS is a naive least-squares solution. Consequently, it is a Tikhonov regularized solution with $\alpha^2 = 0$. This explains the choice for the bounds and indicates that the true regularized solution should be sought strictly inside the interval.

By [26] the expressions for t and ε in Lemma 3.4.1 also hold for $\lambda = 0$ yielding $t = d^T d$ and $\varepsilon^2 = \left\| G^{-1} d \right\|_2^2$ respectively. Then

$$
\begin{aligned}
D(t)y &= \lambda y \\
D(d^T d)y &= 0,
\end{aligned}
$$

where $y = [y_0 \ z]^T$ is the eigenvector corresponding to $\lambda = 0$, and, by Theorem 3.7 in [26], $x^* = \frac{z}{y_0}$. This gives

$$
\begin{bmatrix} d^T d & -dG^T \\ -G^T d & G^T G \end{bmatrix} \begin{bmatrix} y_0 \\ z \end{bmatrix} = 0.
$$

The second row can be written as

$$G^T G x^* = G^T d.$$

These are the normal equations for the problem $\min_x \|Gx - d\|_2^2$ and x^* is indeed a least-squares solution.

Note that if the largest singular value σ_n of the matrix G is known, the results of Section 2.2 imply that $-\sigma_n^2$ specifies a lower bound on λ.

3.6   Curvature of the L-curve

Following [21], see also [17, 22, 18, 17], let

$$\eta := \|x_\varepsilon\|_2^2 \qquad \rho := \mu_\varepsilon + d^T d$$

and

$$\hat\eta := \log \eta \qquad \hat\rho := \log \rho,$$

so that the L-curve is a plot of $\hat\eta/2$ versus $\hat\rho/2$. Then the curvature κ of the L-curve, as a function of ε, is given by

$$\kappa_\varepsilon = 2 \frac{\hat\rho' \hat\eta'' - \hat\rho'' \hat\eta'}{((\hat\rho')^2 + (\hat\eta')^2)^{3/2}}. \tag{3.11}$$

Note, that under assumptions made in Section 3.1, $\eta = \varepsilon^2$, and therefore,

$$\hat\eta' = \frac{\eta'}{\eta} = \frac{2}{\varepsilon} \quad \text{and} \quad \hat\eta'' = -\frac{2}{\varepsilon^2}.$$

Furthermore,

$$\hat\rho' = \frac{\rho'}{\rho} = \frac{\mu_\varepsilon'}{\mu_\varepsilon} \quad \text{and} \quad \hat\rho'' = \frac{\mu_\varepsilon'' \mu_\varepsilon - (\mu_\varepsilon')^2}{\mu_\varepsilon^2}.$$

Substituting these expressions into (3.11) we get

$$
\begin{aligned}
\kappa_\varepsilon &= 2\left(-\frac{\mu'_\varepsilon}{\mu_\varepsilon}\frac{2}{\varepsilon^2} - \frac{\mu''_\varepsilon \mu_\varepsilon - (\mu'_\varepsilon)^2}{\mu_\varepsilon^2}\frac{2}{\varepsilon}\right)\left(\left(\frac{\mu'_\varepsilon}{\mu_\varepsilon}\right)^2 + \left(\frac{2}{\varepsilon}\right)^2\right)^{-3/2}\\
&= 4\varepsilon\mu_\varepsilon\left(\varepsilon(\mu'_\varepsilon)^2 - \mu_\varepsilon\mu'_\varepsilon - \varepsilon\mu_\varepsilon\mu''_\varepsilon\right)\left(\varepsilon^2(\mu'_\varepsilon)^2 + 4\mu_\varepsilon^2\right)^{-3/2} \\
&= \varepsilon^2\mu_\varepsilon\left(2\varepsilon^2\lambda^{*2} - 2\mu_\varepsilon\lambda^* - \varepsilon\mu_\varepsilon\left(\frac{\partial\lambda^*}{\partial\varepsilon}\right)\right)\left(\varepsilon^4\lambda^{*2} + \mu_\varepsilon^2\right)^{-3/2}.
\end{aligned}
\tag{3.12}
$$

3.7 Curvature Estimation and Gauss Quadrature

Numerical evaluation of the expression (3.12) requires calculation of (3.4), which becomes more and more expensive to obtain by direct methods as the dimension of problems increases. This issue, however, is addressed in [12, 13, 2, 14]. A proposed approach lies in obtaining both upper and lower bounds on the expression of the form

$$
\nu_p(\alpha) = d^T G(G^T G + \alpha I)^p G^T d,
$$

where α is a positive scalar and p is a negative integer ($p = -3$ in (3.4)). These bounds are obtained using an iterative procedure and become tighter as the number of iterations increases.

We do not want to reproduce the papers referenced above, but we briefly illustrate the idea and the notation. Note that we may rewrite $\nu_p(\alpha)$ as a quadratic form

$$
s := g^T \varphi(M) g, \text{ with } \varphi(M) := (M + \alpha I)^p , \ g \in \mathbb{R}^n.
$$

For the following analysis it is enough to require that φ is an analytic function and M is a symmetric n-by-n matrix. In our case $M = G^T G$ and $g = G^T d$. Consider an eigenvalue decomposition $U\Lambda U^T$ of the matrix M with $\lambda_1 \leq \cdots \leq \lambda_n$ as eigenvalues. Then s can be expressed as a Stieltjes integral with a staircase measure function $\omega(x)$ that has steps of size $(U^T g)_i^2$ at the corresponding eigenvalues λ_i.

$$
s = \int_a^b \varphi(\lambda)\, d\omega(\lambda)
$$

Here the limits of integration are the lower and upper bounds on the spectrum of M, i.e. $a \leq \lambda_1 \leq \cdots \leq \lambda_n \leq b$. Having s represented as an integral we may further use numerical integration to approximate it. We use Gauss quadrature for this purpose

$$s = \int_a^b \varphi(\lambda) \, d\omega(\lambda) \approx \sum_{i=1}^k \varphi(x_i)\omega_i \ ,$$

where the quantities $x_i \leq \cdots \leq x_k$ denote the abscissas of the quadrature rule, ω_i's are the corresponding weights and k specifies the degree. The larger the degree is used the more and more accurate an approximation $\nu_p(\alpha)$ becomes. Prescribing an abscissa $x_1 = a$ or $x_2 = b$ will give us a Gauss-Radau quadrature rule.

Algorithm 3.1: Lanczos Bidiagonalization II

input : matrix G, starting vector d, optional arguments: γ, δ, p, q, k

output: γ, δ, p, q

1   $\sharp$ *initialization*

2   **if** *optional arguments are NOT specified* **then**

3      $k_{max} = \sqrt{\text{minimal of the dimensions of } G}$

4      $k = 1$

5      $p = d/\left\|d\right\|_2$

6   set γ, δ to be zero vectors of size k_{max}

7   **else**

8      $k_{max} = 2k$

9   expand γ, δ vectors to the size k_{max}

10   **end**

11   $\sharp$ *main loop*

12   **while** $k \leq k_{max}$ **do**

13   **if** $k \leq 1$ **then** $q = G^T p$

14   **else** $q = G^T p - \delta_{k-1} q$

15      $\gamma_k = \left\|q\right\|_2$

16      $q = q/\gamma_k$

17      $p = Gq - \gamma_k p$

18      $\delta_k = \left\|p\right\|_2$

19      $p = p/\delta_k$

20      $k = k + 1$

21   **end**

After k iterations Lanczos Bidiagonalization II algorithm produces $(k+1)$-by-k lower bidiagonal matrix

$$B_k = \begin{bmatrix} \gamma_1 & & & \\ \delta_1 & \ddots & & \\ & \ddots & \ddots & \\ & & \ddots & \gamma_k \\ & & & \delta_k \end{bmatrix},$$

so that the Gauss and Gauss-Radau quadrature rules for $\nu_p(\alpha)$ will be defined as follows:

$$G_p(\alpha) = \left\|G^T d\right\|_2^2 e_1^T (B_k^T B_k + \alpha I)^p e_1 = \|d\|_2^2 e_1^T B_k (B_k^T B_k + \alpha I)^p B_k^T e_1, \qquad (3.13)$$

$$R_p(\alpha) = \left\|G^T d\right\|_2^2 e_1^T (\tilde{U}_k^T \tilde{U}_k + \alpha I)^p e_1, \qquad (3.14)$$

where $\tilde{U}_k$ is $(k+1)$-by-k upper bidiagonal matrix obtained from B_k by a sequence of Givens rotations and by setting the main diagonal to zero.

Functions $G_p(\alpha)$ and $R_p(\alpha)$ provide lower and upper bounds on $\nu_p(\alpha)$ when $\alpha > 0$:

$$G_p(\alpha) \leq \nu_p(\alpha) \leq R_p(\alpha).$$

These bounds depend on the iteration index k and become tighter as it increases. Our implementation of Lanczos Bidiagonalization algorithm allows restarting from the specified (usually last) iteration if optional parameters are supplied. This enables one to increase the precision when necessary. This feature is exploited by the main algorithm that iterates by gradually decreasing α. Since for $p < 0$ and non-singular B_k we have that:

$$\lim_{\alpha \searrow 0} G_p(\alpha) < \inf, \quad \lim_{\alpha \searrow 0} R_p(\alpha) = \inf,$$

it is natural that bounds will become loose as $\alpha \searrow 0$.

Note that evaluating the expressions $G_p(\alpha)$ and $R_p(\alpha)$ implies solving linear systems:

$$(\tilde{U}_k^T \tilde{U}_k + \alpha I)x = e_1,$$
$$(B_k^T B_k + \alpha I)x = B_k^T e_1.$$

It is easy to see that the above equations are normal equations for the linear least-squares problem:

$$\min \left\| \left[\begin{smallmatrix} \tilde{U}_k \\ \sqrt{\alpha}I \end{smallmatrix} \right] x - \left[\begin{smallmatrix} 0 \\ e_1/\sqrt{\alpha} \end{smallmatrix} \right] \right\|_2$$

$$\min \left\| \left[\begin{smallmatrix} B_k \\ \sqrt{\alpha}I \end{smallmatrix} \right] x - \left[\begin{smallmatrix} e_1 \\ 0 \end{smallmatrix} \right] \right\|_2.$$

This means that the solution x for the linear least-squares problem satisfies the original linear system as well. We may, however, exploit the structure of LLS problems and solve them efficiently by a sequence of Givens rotations that produces the QR factorization. This approach is described in [6, 10, 36].

Chapter 4

Regularization Algorithm

Before presenting the details of the algorithm we state our assumptions and present some geometry and relations among the various parameters. The key assumption is that values of parameters are in a bounded interval, as described in Section 3.5. Making this assumption does not restrict our ability to locate a good regularized solution, since it is always located in the interval of interest.

First, we observe that the regularized solution is a monotonic function in t and λ.

Lemma 4.0.1. $\|x(t)\|_2$ and $\|x(\lambda)\|_2$ are monotonically increasing functions in t and λ, respectively.

Proof: (This lemma follows from Theorem 3.7 in [26].) Under our assumptions, we have $\lambda(t) < 0$. Therefore, complementary slackness for TRS in (3.1), and the construction of $y_0(t)$ in [26] implies that

$$\|x(t)\|_2 = \varepsilon(t)^2 = \frac{1 - y_0(t)^2}{y_0(t)^2}.$$

Here $y_0(t)$ is the first component of the normalized eigenvector $y(t)$ corresponding to the smallest eigenvalue λ_1 of the matrix $D(t)$. Then, by Lemma 3.6 in [26], we have $y_0(t)$ is strictly monotonically decreasing, which in turn implies that $\varepsilon(t)$ is strictly monotonically increasing in t.

Using the correspondence between t and λ described in Section 3.4, we can deduce the monotonicity result for λ. $\blacksquare$

Throughout the rest of the paper we will interchangeably use parameters ε, t and λ when describing points on the L-curve. Hence, if a statement is true for *smaller or larger values of ε*, it is also true for respectively smaller or larger values of t and λ.

From a given t, we can calculate the corresponding ε and the value of the objective function $\mu_\varepsilon = k(t)$, thus obtaining a point on the L-curve. To analyze the location of a given pair $(\varepsilon, \mu_\varepsilon)$ on the L-curve, we need the derivative of $l_r := l_r(\varepsilon) := \log(\|Gx(\varepsilon) - d\|_2)$ with respect to $l_x := l_x(\varepsilon) := \log(\|x(\varepsilon)\|_2)$, i.e.

$$
\begin{aligned}
\frac{d(l_r(\varepsilon))/d(\varepsilon)}{d(l_x(\varepsilon))/d(\varepsilon)} = \frac{d(\log(\|Gx - d\|_2))/d(\varepsilon)}{d(\log(\|x\|_2))/d(\varepsilon)} &= \frac{1}{2}\frac{d(\log(\mu_\varepsilon + d^T d))/d(\varepsilon)}{d(\log(\varepsilon))/d(\varepsilon)} \\
&= \frac{1}{2}\frac{\mu_\varepsilon' \varepsilon}{\mu_\varepsilon + d^T d} \qquad (4.1) \\
&= \frac{\varepsilon^2 \lambda_\varepsilon}{\mu_\varepsilon + d^T d},
\end{aligned}
$$

where μ_ε' is found using (3.2).

To distinguish whether a point lies before (left) or after (right) the *elbow*, one can test the value of the derivative. It should be (negative) close to zero if we are at the plateau after the *elbow*. Alternatively, the value tends to a large negative number as we approach the *elbow* from the left.

4.1   Initial L-curve point

Our algorithm iterates by steadily increasing the value of parameter t. Then each subsequent point is located to the right of the previous one, i.e. corresponds to a larger value of ε (and t). Hence, locating the *elbow* of the L-curve is only possible when we start to the left of the *elbow*. We need a value of λ or equivalently, by Lemma 3.4.1 or 3.4.2, t, to locate a point. We can employ different strategies to achieve this task. One way is to start with the point corresponding to $\lambda = -\sigma_n(G)^2$, see Sections 3.5 and 2.2.

In the case we do not have the *largest* singular value of the matrix G, we can start with a point associated with small enough value of $t = \frac{d^T d}{2}$. This value does not have sound

theoretical basis, yet empirically, we note that it works in most cases. We will see that taking this value is good enough to be on the safe side. As we discussed in Section 2.3, a "well-shaped" L-curve plot can be viewed as a linear plateau to the right of the *elbow* and a linear vertical part to the left of the *elbow*. For well shaped L-curve plots, tiny changes in t would result in huge changes in ε when we are on the horizontal part. Vice versa, large changes in t have little affect on ε when we are on the vertical part. This is explained by the structure of the singular value decomposition of the matrix G (see Section 2.2). The behaviour remains true for less well-behaved L-shaped plots. This tells us that points that lie on the plateau region correspond to the values of t that are very close to $d^T d$. Thus, taking half of this value will put us onto the vertical part to the left of the *elbow*.

4.2   Outline of the algorithm

Algorithm 4.1: Trust-Region Based Regularization [**overview**]

    input : operator matrix G, observed data vector d

    output: solution vector x, norm of the residual *res* and the corresponding Tikhonov
                            regularization parameter α

    $\rightarrow$ *initialization* (see Algorithm 5.3)

    $\rightarrow$ *main loop* (see Algorithm 5.4)

    $\rightarrow$ *final solution refinement* (see Algorithm 5.5)

Algorithm 4.2: Helper Functions

---

    function $[t, x, k] = \textbf{l2t}\ (\lambda)$

    begin

        solves for x in $(G^T G - \lambda I)x = a$

        $t = \lambda + d^T G x$

        $k = (x^T x + 1)\lambda - t$

    end

    function $[\lambda, x, k] = \textbf{t2l}\ (t,\ \lambda)$

    begin

        run **eigs** to compute the smallest eigenpair (λ, y) of the matrix D.

        use eigenvalue calculated at the previous step as the initial guess, this **greatly**

        improves convergence rate.

        change the first component y_1 of the eigenvector y to have positive sign.

        $\varepsilon^2 = (1 - y_1^2)/y_1^2$

        $x = (\varepsilon^2 + 1)\lambda - t$

        $k = y_{2\ldots n}/y_1$

    end

    function $[\kappa_{low}, \kappa_{up}] = \textbf{curvature}\ (\varepsilon,\ res,\ \lambda)$

    begin

        compute lower and upper bounds on the curvature using current Lanczos

        bidiagonalized approximation.

    end

General idea of the algorithm is presented in the beginning of Section 4. Below we will describe the details behind the implementation. It can be divided into three large parts: *initialization*, *main loop* and *final solution refinement*.

Algorithm 4.3: Trust-Region Based Regularization [**initialization**]

---

1 compute the *largest* singluar value σ_n of the matrix G

2 compute the initial bidiagonalization (γ, δ) of the matrix G using Lanczos
Bidiagonalization II algorithm, use d as the starting vector.

3 $t_{low} = 0$

4 $t_{up} = d^T d$

5 $\lambda_{low} = -\sigma_n^2$

6 $\lambda_{up} = 0$

7 $\varepsilon_{up} = -1$

8 $\kappa_{low}^{previous} = \inf$

9 $\kappa_{up}^{previous} = \inf$

10 $\lambda = \lambda_{low}$

11 find starting L-curve point, $[t, x, k] = \mathbf{l2t}(\lambda)$

We start by computing several things that are going to be used throughout the algorithm. We also initialize the variables. Computing the *largest* singular value of G is not absolutely necessary, but is relatively cheap and yields a lower bound on the eigenvalue λ. If it is undesirable to compute the *largest* singular value of G, this step can be omitted and a reasonable value for the parameter t, e.g. $\frac{d^T d}{2}$ computed instead. This also places the lower bound on the eigenvalue by Lemma 3.4.2.

The more important step is to compute the initial bidiagonalization of the matrix G. This data is used to estimate the curvature of the L-curve every time a point is obtained. The details are covered in Section 3.7.

We then proceed by getting an initial point on the L-curve. The discussion on getting a good estimate is in Section 4.1. We assume that we know the *largest* singular value of G and thus start with a value on a parameter λ. Hence, to locate a point on the L-curve, we solve for values t, x and k.

Algorithm 4.4: Trust-Region Based Regularization [**main loop**]

---

1 **while** $\lambda < \lambda_{up} - 10^{-10}$ **do**

2   ♯ *calculate the slope of the L-curve and $\frac{d\lambda}{dt}$*

3 $\varepsilon^2 = x^T x,\ res^2 = k + d^T d$

4 $L_{slope} = \lambda \varepsilon^2 / res^2$

5 $\frac{d\lambda}{dt} = (1 + \varepsilon^2)^{-1}$

6   save current point to the solutions history

7 $t_{low} = t,\ \lambda_{low} = \lambda$

8 $[\kappa_{low}, \kappa_{up}] = \mathbf{curvature}(\varepsilon, res^2, \lambda)$

9   ♯ *termination criteria*

10   **while** *curvature value is not certain* **do**

11   **if** $\kappa_{low} > \kappa_{up}^{previous}$ **then**

12   **DONE**, proceed to the final solution refinement

13   **end**

14   **if** $\kappa_{up} < \kappa_{low}^{previous}$ **then**

15 $\kappa_{low}^{previous} = \kappa_{low}$

16 $\kappa_{up}^{previous} = \kappa_{up}$

17   curvature value is now specified, **break**

18   **else**

19   update bidiagonalization (γ, δ) of G to improve precision

20 $[\kappa_{low}, \kappa_{up}] = \mathbf{curvature}(\varepsilon, res^2, \lambda)$

21   recalculate bounds on $\kappa^{previous}$

22   **end**

23   **end**

24   ♯ *update t*

25 $\varepsilon_{target} = \varepsilon$

26   perform triangle interpolation on the $k(t)$ to get an estimated t for ε_{target}

27 $[\lambda, x, k] = \mathbf{t2l}(t, \lambda)$ ♯ *find next L-curve point*

28 **end**

At each iteration the algorithm takes the current point and produces the next one strictly to the right on the L-curve. There are several possible strategies to achieve this goal. As Lemma 4.0.1 suggests, increasing either one of the parameters t, λ or ε will move us further to the right. Hence, we can take a step by changing any one of them. The hard part, though, is the strategy on choosing the step length. In the current implementation we do the following.

Suppose we are given the target value for ε. Then we can potentially solve for t and λ (see Lemma 3.4.3). However, this involves solving TRS that we are trying to avoid. Instead, we try to estimate the value for t. We do not care if that value does not correspond well to the target ε. But we require that a new value for t is larger than the previous one, so that we can take a step.

The key idea behind this estimation lies in observing the properties of the function $k(t)$. Recall from Section 3.3, that $k(t) = (\varepsilon^2 + 1)\lambda - t$ and $\mu_\varepsilon = \max_t k(t)$. The function $k(t)$ is also concave. Now, let ε_0 and t_0 represent the optimal pair (ε_0, t_0) and, hence, a point on the L-curve. Take also $\varepsilon_{\text{tgt}} \geq \varepsilon_0$. Denote t_{tgt} as the optimal value of t corresponding to ε_{tgt}, so that $\mu(\varepsilon_{\text{tgt}}) = k(t_{\text{tgt}})$. Consider the function $k(t)$ with $\varepsilon = \varepsilon_{\text{tgt}}$. Clearly, it attains the maximum at the point $t = t_{\text{tgt}}$. By monotonicity we have $t_0 \leq t_{\text{tgt}}$. We can also take the point $t_1 = d^T d$. From Section 3.5 we know that $k(t_1) = -t_1 = -d^T d$. Moreover, the derivative is $k'(t_1) = -1$. Hence, we get $t_0 \leq t_{\text{tgt}} \leq t_1$.

Now we can take the triangle interpolation at the points t_0 and t_1 to get an estimate for t_{tgt}. This technique gives quite good results due to the fact that $k(t)$ can be approximated by linear models on each of the sides. It is also cheap to compute the tangent at the point t_0, since $k'(t) = (\varepsilon^2 + 1)y_0^2(t) - 1$, where $y_0(t)$ does not depend on ε. Hence, we can use its value from the previous step. Figure 4.1 illustrates this approach.

In the algorithm, we use the most simple case of choosing the target value for ε – just taking the current one. An assumption of the *easy case* ensures that we have a non-zero step in t, since $k(t)$ is differentiable everywhere and $\lim_{t \to 0} k(t) = -t$.

With the new t we can compute both λ and ε as Lemma 3.4.2 suggests. Particular details on the eigenvalue computation are given in Section 5.1.

The termination condition is based on the L-curve maximum curvature criterion, i.e. we
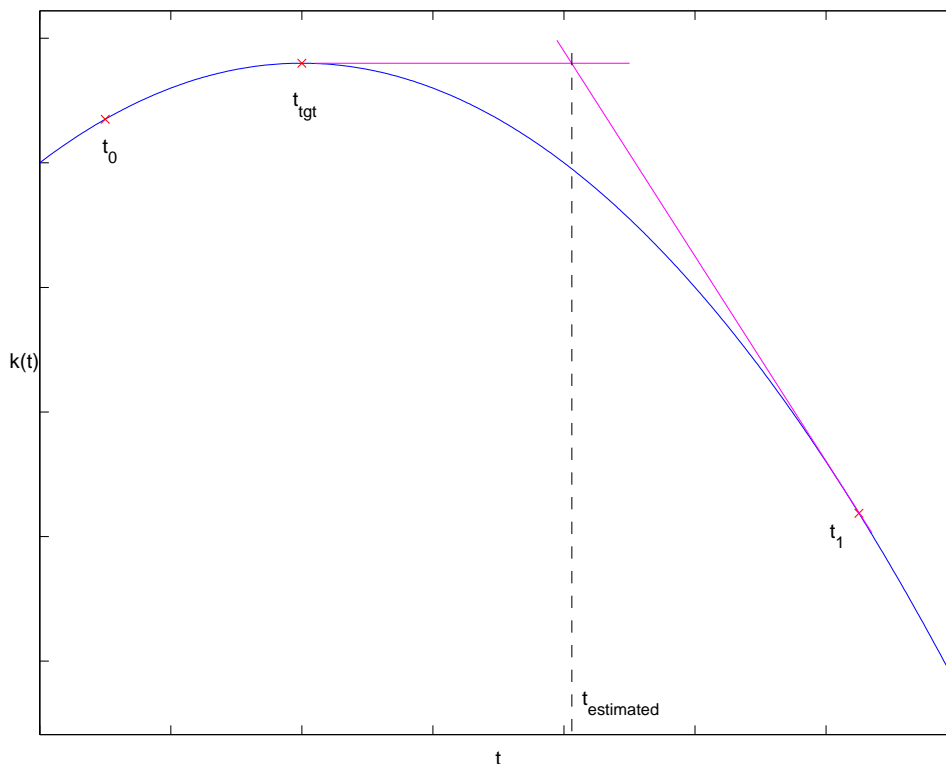
Figure 4.1: k(t) and triangle interpolation

look for a point on the L-curve that has the maximum negative curvature. To locate such a point we compute the curvature at each step. This computation uses Gauss Quadrature approach which is described in Section 3.7. Since we are only getting lower and upper bounds on the real value of the curvature, it can be a problem to compare between two values for the consecutive points, e.g. if the corresponding intervals overlap. If such situation is detected, we improve the bidiagonalization of the matrix G. This increases the precision in the curvature estimation and, eventually, allows to safely compare the curvature at these two points. Since L-curve is a convex function near the *elbow*, we can determine the area of interest by keeping track of the curvature. Once we get the point with a smaller curvature than the previous one, we know we have gone too far. The main loop of the algorithm terminates once we have 3 points, such that the middle one has a larger curvature value than the other two. From the convexity we deduce that the *elbow*

should lie somewhere between the endpoints. The final refinement is then performed to estimate the *elbow* location.

Algorithm 4.5: TRS Based Regularization [**final solution refinement**]

---

1 $\sharp$ *observe interval of last three points* left, center, right

2 **while** *point of maximum curvature still can be improved* **do**

3   set λ as bisection of either left or right interval

4   find corresponding L-curve point, $[t, x, k] = \mathbf{l2t}(\lambda)$

5 $\sharp$ *calculate norm of the residual and* ε

6 $\varepsilon^2 = x^T x$

7 $res^2 = k + d^T d$

8 $[\kappa_{low}, \kappa_{up}] = \mathbf{curvature}(\varepsilon, res^2, \lambda)$

9   **if** *located point has larger curvature* **then**

10   set current point as a solution

11   **DONE**

12   **end**

13   shrink interval of intereset

14 **end**

To estimate the *elbow* location, we proceed with a simple bisection of the left and right intervals trying to find a point with the maximum curvature value. We stop once we have got a point with a curvature value larger than the one we have seen so far.

4.3   Future improvements

The algorithm presented above is not polished enough for commercial use. Rather, it illustrates the concepts and helps in understanding the regularization process better. It would be natural to view it as a base for more robust algorithms that can be built upon the techniques herein. We now outline major improvements and some limitations that apply to the current implementation.

One major limitation is that no proof of convergence to the *elbow* is provided.

The inability to prove convergence follows from the difficulty of proving correctness of the optimum. We leave this question open for future research. However, given a proper L-curve, the method generally finds the solution.

The mathematical reasoning behind the method relies on the L-curve maximum curvature criterion, a heuristic. Given no *apriori* information on the error, e.g. either norm or distribution, there is no way to prove anything definite about the solution. Indeed, it is proved (see [7]) that, in the absence of the error level information, any method will fail on a specially constructed input data. It is important to realize that we can only trust the results if we assume some reasonable constraints on the error level. For example, if the norm of the error is much larger than the norm of the data or, in a physical sense, the energy of the noise is larger than that of the signal, the reconstructed solution will be meaningless. The key difference of our approach to that of the Conjugate Gradient method is that, though we require some constraints on the error, we never ask for them explicitly. Using the L-curve criteria we implicity extract the inherent characteristics of the problem. We believe that many real-world engineering problems possess these characteristics and can be tackled by our method.

In Section 5.3 we outline some possible directions for future research. These include merging with Conjugate Gradient type methods. Here we will concentrate on the improvements that can be done to the presented method.

There are two possible ways to improve the algorithm. One way is to choose a step length more effectively and another is improvement of the termination criteria. Following considerations might be helpful for deciding on the step length and *elbow* location. At every point on the L-curve we are given the slope of the tangent line. If it is known that we are sitting either at the vertical or horizontal part, then it is possible to make a larger step by taking a linear approximation. For instance, we can determine a target value of ε by taking an intersection of the tangent line with the X-axis. This strategy may help to climb down the vertical part faster. Same information can help to locate the corner by taking linear approximations at two points – one on the vertical part and another on the horizontal.

The final refinement step can also be improved by utilizing the data from Lanczos bidiagonalization of the matrix G. Once the *elbow* position is locked, one may construct the so called L- and Curvature-Ribbons to better approximate the *elbow*. This approach is discussed in [2].

Chapter 5

Numerics/Computations

5.1 Eigensolver issues

As shown above, obtaining a new L-curve point means solving for the smallest eigenpair of the matrix $D(t)$. In the case G is large and sparse, the same is true for $D(t)$, so one should use matrix-free iterative algorithms to compute the eigenpairs, e.g. Lanczos methods. As t increases, the smallest eigenvalue may become numerically closer to the second one. This impacts the convergence rate, substantially slowing down the eigensolver.

Under such numerical degeneracy an algorithm may converge to a wrong eigenpair, giving an incorrect eigenvector and an incorrect regularized solution. One way to control the eigensolution is to start with an initial eigenvalue smaller than the estimated one and, at the same time, relatively close to it. For iterative algorithms, it is possible to store previous eigenvalue results to re-use on the next step as an initial guess. This works only if the eigenvalue is about to increase at every subsequent iteration.

We have employed this method in our Regularization Algorithm and it proved to be very efficient. We have used the MATLAB **eigs** routine which uses a Lanczos-type matrix-free algorithm. With a good initial eigenvalue guess, this method computes eigenvalues in time independent of the gap between the first and second eigenvalues.

Another approach that can be used is to apply a spectral transformation to separate the first and second eigenvalues, i.e. preconditioning. In particular, a Tchebyshev polynomial

transformation is discussed in [27] and [28].

A bug in the MATLAB (version 6) **eigs** routine was also discovered. This routine behaves incorrectly when called with numeric initial guess for the eigenvalue and a matrix supplied via an external program file. Under such calling conditions the computed eigenvalue is completely wrong and differs by several orders from the correct one. The bug was reported to the MATLAB technical support group. To provide a work around, we force the algorithm to form the matrix explicitly. This, however, results in larger memory requirements and should be removed once the bug is fixed.

5.2   Image deblurring example

We demonstrate how the algorithm works by considering a sample problem of deblurring an image. Problems of this nature often occur in the real world. For instance, one might need to deblur a photo taken by a space telescope or a satellite.

For this particular example we take an image generated by the Hansen MATLAB package ([19]). This package provides an excellent set of regularization tools that can be used for demonstration purposes. Figure 5.1 shows the image generated by the **blur** command. This command also produces the blurring matrix G and the right-hand side d, i.e. observed data, computed as $d = Gx_{\text{true}} + \eta$. Where η represents the noise. Figure 5.2 shows the observed image.

The generated image is 40-by-40 grayscale picture, which is stored as a vector x_{true} of size 1600. This vector is formed by stretching the image matrix into a single column. Every component x_{true}^i represents the brightness of the pixel, measured from 0 for the white to 3 for the black (see the colorbar). The matrix G stands for the operator that represents degradation of the image caused by atmospheric turbulence blur, modelled by a Gaussian point-spread function,

$$h(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$
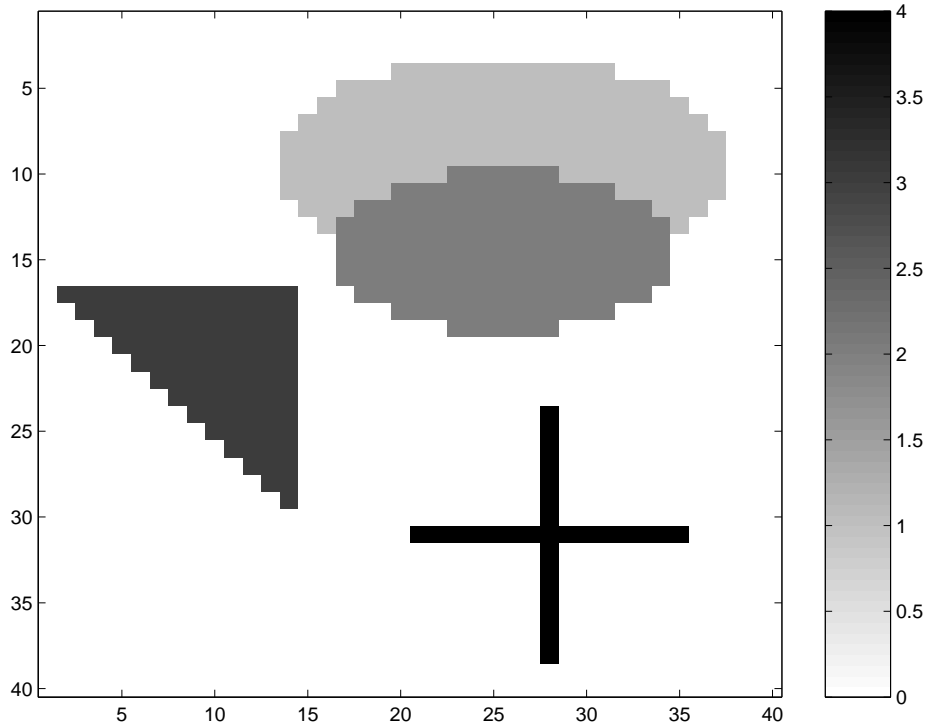
Figure 5.1: Image deblurring example: original picture

Taking a symmetric banded Toeplitz matrix T with the first row:

$$z_i = \begin{cases} e^{-(i-1)^2/2\sigma^2} & 1 \le i \le b, \\ 0 & b < i \le 40 \end{cases}$$

$$T = \begin{bmatrix} z_1 & z_2 & \dots\dots\dots & z_{40} \\ z_2 & z_1 & z_2 & \dots & z_{39} \\ \vdots & \vdots & \vdots & \vdots\ \vdots & \vdots \\ z_{40} & z_{39} & z_{38} & \dots & z_1 \end{bmatrix},$$

matrix G is constructed as $G = (2\pi\sigma^2)^{-1}T \otimes T$, where $\otimes$ denotes the Kronecker product.

Here parameter σ controls the smoothness (by defining the shape of the Gaussian point spread), and b stands for the bandwidth. Since only non-zero elements are within a distance
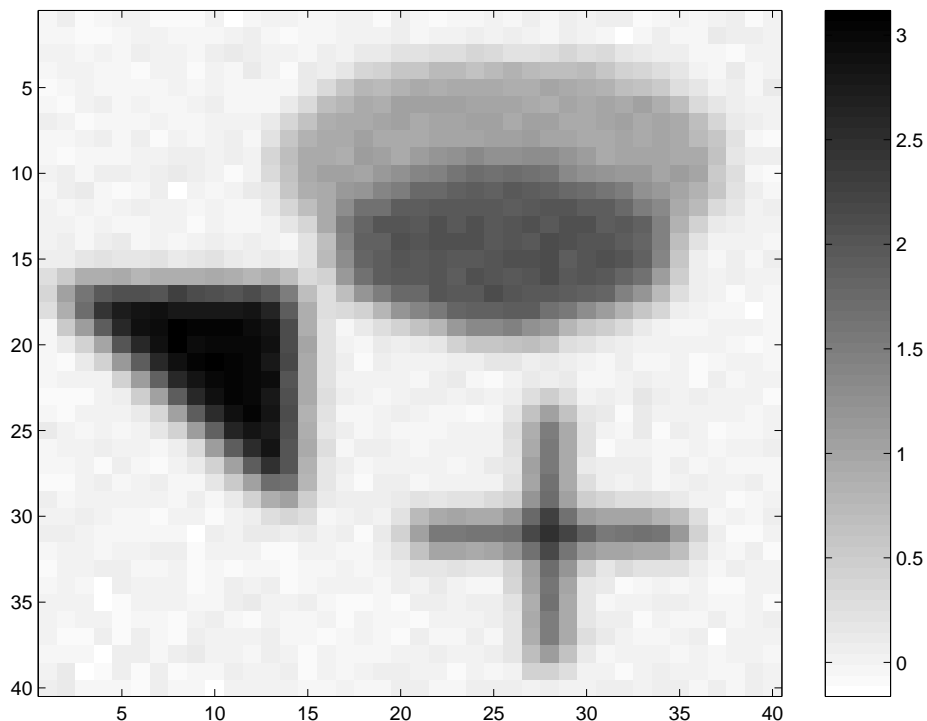
Figure 5.2: Image deblurring example: observed data, blurred with added noise

$b-1$ from the diagonal of the matrix T, it can be stored in a sparse format. It also follows that matrix G is sparse. Hence, we have an example of the large sparse problem.

For our example we fix the parameters to be $\sigma = 1$, $b = 5$. Noise η has a normal distribution with the mean of 0 and the standard deviation of 0.05.

Before running the algorithm, we construct the L-curve to get an idea where the solution is located. We can see that the curve is not strongly L-shaped, but we can still distinguish both vertical and horizontal parts. We also build a plot (dashed line) that shows how well the points on the L-curve approximates the true solution. For every point x we determine the quantity $\dfrac{\|x_{\text{true}} - x\|_2}{\|x_{\text{true}}\|_2}$ which we treat as the relative accuracy; the smaller the value, the better the approximation we obtain. The minimum corresponds to the best possible solution that can be obtained using the Tikhonov regularization approach.

Then we run the RPTRS algorithm. For each point it visits, we present the associated

43
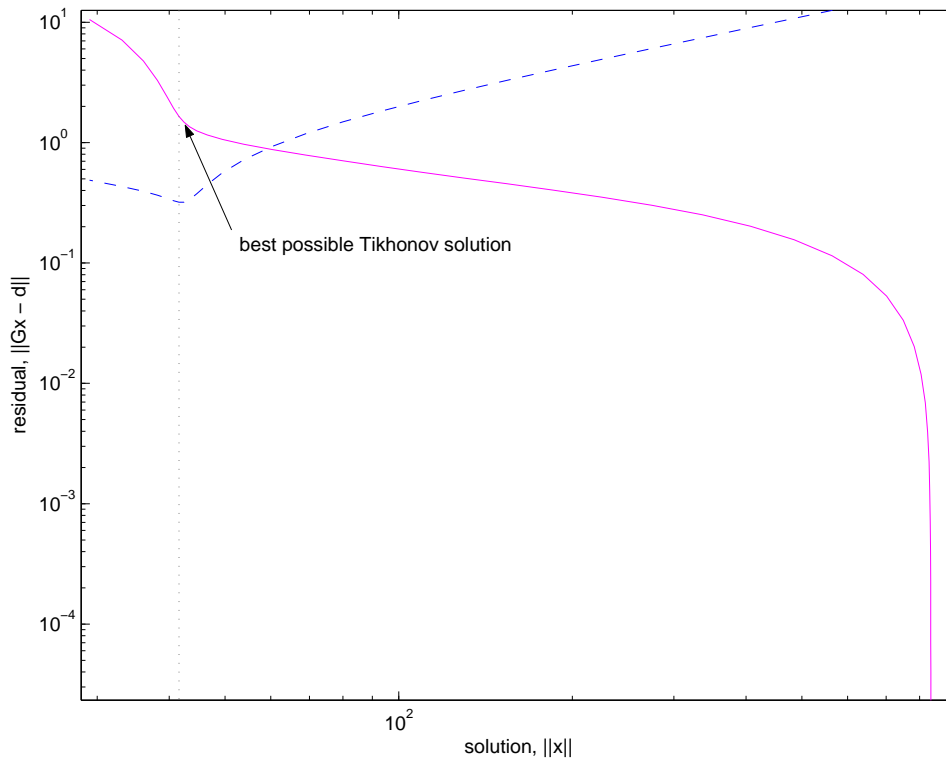
Figure 5.3: Image deblurring example: corresponding L-curve

solution image. We can follow how the solution transforms as we go along the curve. For smaller values of the paramter t the solution appears to be very smooth. The noise components are almost eliminated for these solutions. However, as we increase the regularization parameter, the noise starts to evolve. At the same time, pictures become sharper and represent a better approximation to the true solution. This behaviour continues until we hit the point #5 (see Figure 5.13). Suddenly, the noise components overcome the real signal and the solution becomes less distinguishable. Finally, the situation becomes even worse at the last point. The least-squares solution consists mostly of the noise components and contains practically no signal information.

The algorithm, however, observes the changes in the curvature value and backtracks, trying to locate the *elbow*. Figure 5.4 demonstrates the steps that are taken. Points marked with x (cross) are those visited during the main loop, and circles denote the final refinement

44
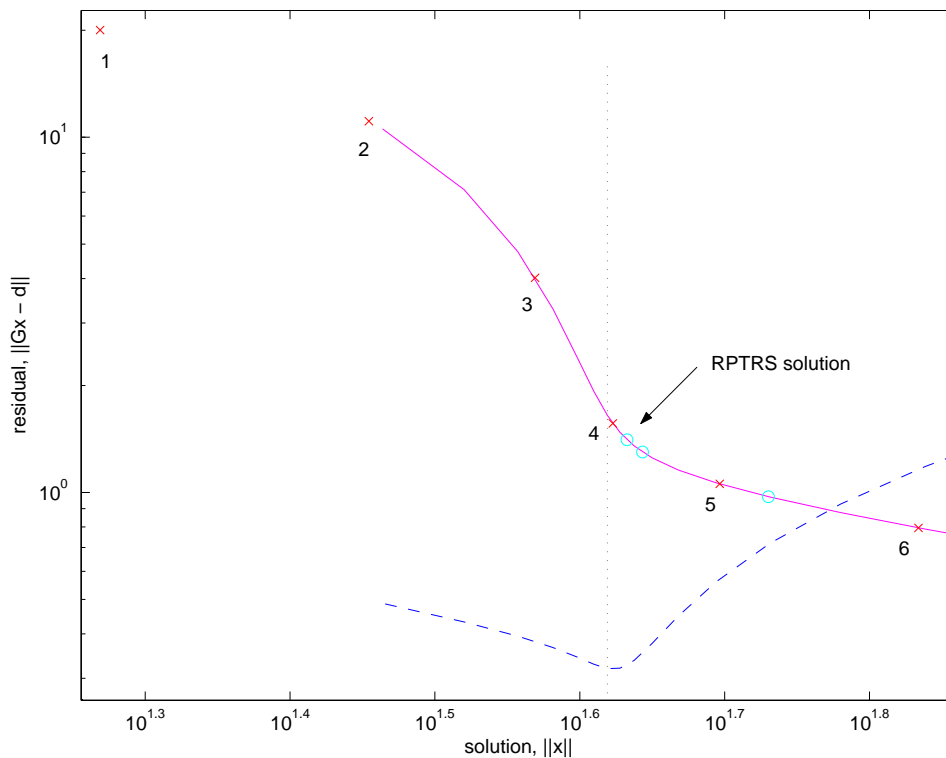
Figure 5.4: Image deblurring example: corresponding L-curve with RPTRS points

steps. The algorithm terminates after locating the point closest to the point of the largest curvature (on the convex part). This point is returned as the solution. Note the proximity to the best possible Tikhonov solution. The final RPTRS solution is shown on Figure 5.5.

For more information and techniques on image de-blurring problems, we note the ongoing research based on wavelets (see e.g. [3, 4, 5]). We do not perform any comparison with these techniques in this thesis.

5.3   Open Questions

We compare our approach to the conjugate gradients based method for solving the least-squares problems CGLS. CGLS is one of the most robust regularization techniques that can handle very large problem instances. This method, described in [24] (see also [17]),
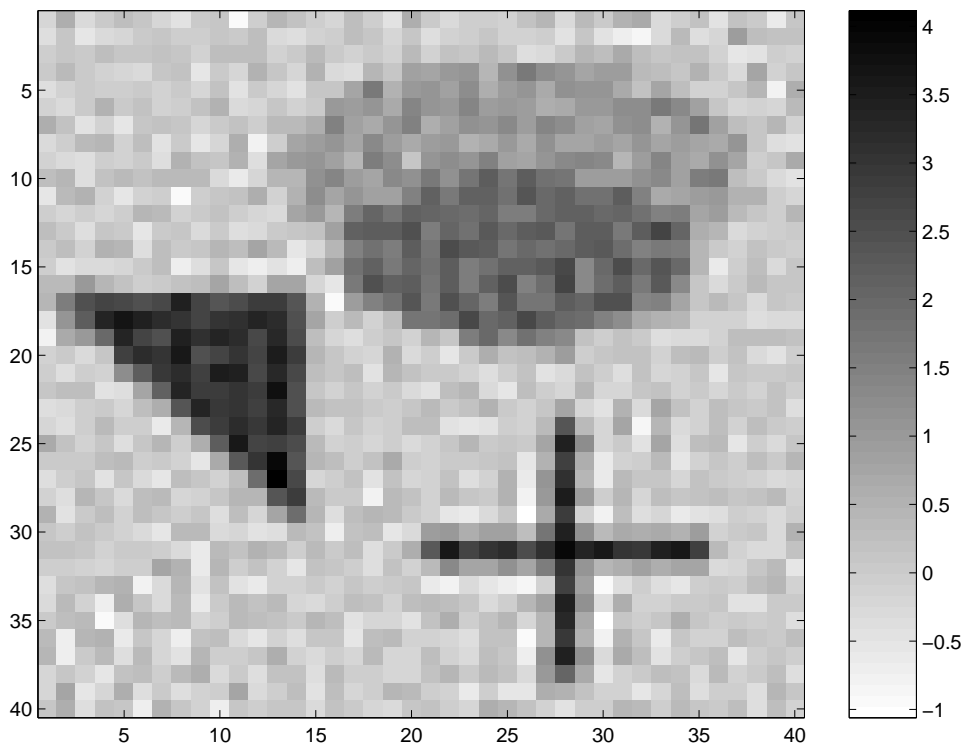
Figure 5.5: Image deblurring example: RPTRS solution picture

applies conjugate gradients (CG) to the normal equations $T^*Tx = T^*d$ along with an early termination criteria to obtain the regularized solution. The stopping condition is based on the discrepancy principle, i.e. the method terminates once the residual is smaller than some prescribed bound δ. Typically, δ is chosen basing on the knowledge of the norm of the noise.

We applied the CGLS algorithm on the data from the previous example supplying δ to be precisely the norm of the noise, i.e. $\delta = \|\eta\|_2$. In some sense this corresponds to the best case for CGLS. The results are presented in Table 5.2 and Figure 5.6. The CGLS points are shown as circles above the L-curve. The CGLS solution is almost as good as the best Tikhonov solution. This result is not unusual and emphasizes the fact that the method was applied with exact knowledge of the noise. However, comparing both CGLS and RPTRS solutions to the true one (see Figure 5.7), we see that both methods achieve

46
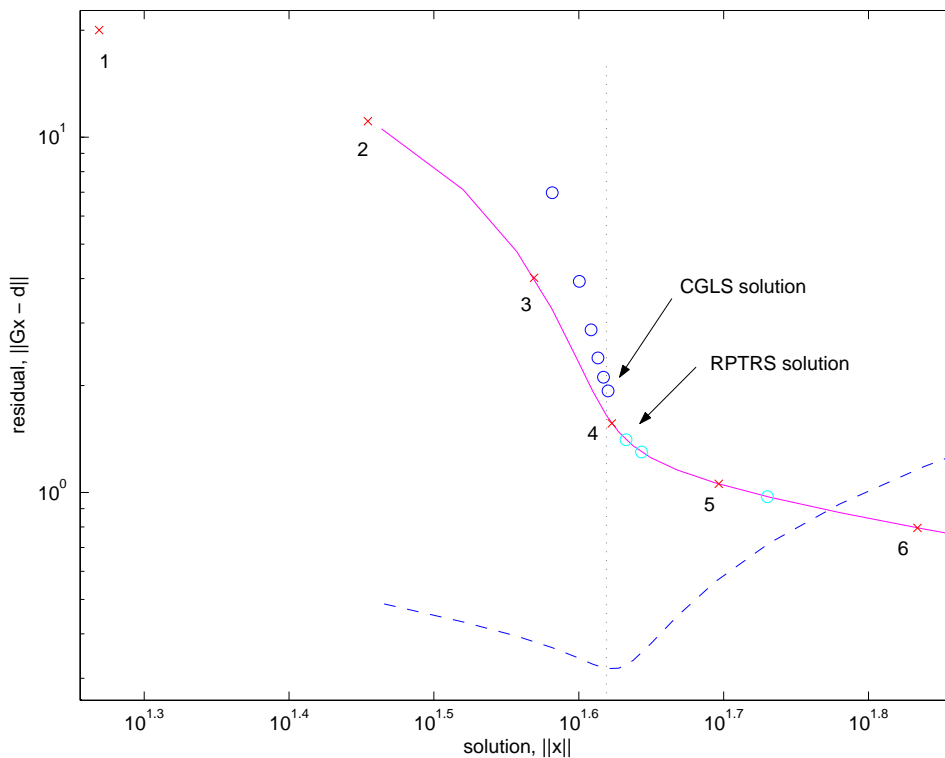
Figure 5.6: Image deblurring example: corresponding L-curve with CGLS points

practically the same accuracy.

The RPTRS algorithm, though, does not require a specific value of the norm of the noise. This is a big advantage in a sense that CGLS might perform very poorly if supplied with slightly smaller (or larger) value of δ. Figure 5.8 illustrates this situation. Running CGLS with $\delta = 0.6 \|\eta\|_2$ results in a larger number of iterations (31 comparing to 6 with $\delta = \|\eta\|_2$) and the computed solution is much worse now. This shows the importance of a robust stopping criteria that does not rely on the possibly uncertain data.

The main advantage of the CGLS method is its speed. Each iteration of the algorithm requires only several matrix-vector multiplications, where only the original matrix G is used. This allows for solutions of problems that involve large sparse matrices which are never formed explicitly. At the same time, the RPTRS algorithm can be viewed as a matrix-free iterative algorithm based on the Lanczos method that features conjugate gradients
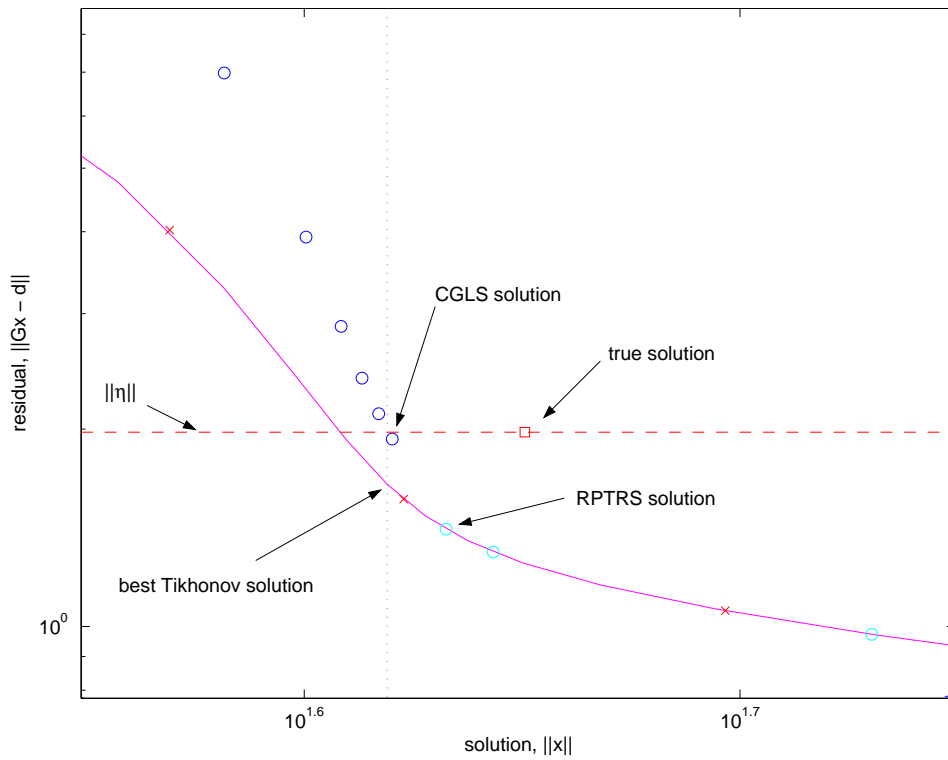
Figure 5.7: Image deblurring example: CGLS, RPTRS, x_{true}, best Tikhonov solutions

steps as well. This leads to a conclusion that combining both approaches may result in a better algorithm that can provide a reliable and a fast way to locate a regularized solution in the absence of any certain knowledge about the noise.
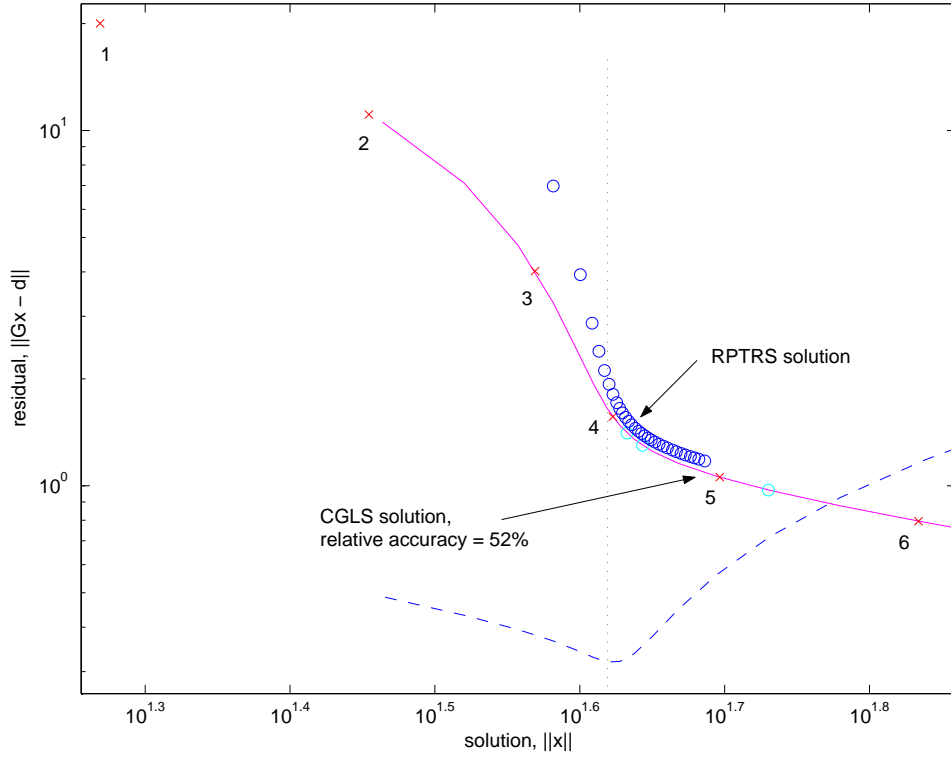
Figure 5.8: Image deblurring example: CGLS with $\delta = 0.6 \, \|\eta\|_2$, rel.acc. $= 52\%$

##	$\|x\|_2$	$\|Gx - d\|_2$	accuracy [%]
1	3.8162e+001	6.9804e+000	47.59
2	3.9849e+001	3.9256e+000	41.83
3	4.0593e+001	2.8676e+000	38.99
4	4.1045e+001	2.3920e+000	36.97
5	4.1406e+001	2.1105e+000	35.51
6	4.1706e+001	1.9309e+000	34.41

Table 5.1: Data for points visited by the CGLS algorithm with $\delta = \|\eta\|_2$

##	$\|x\|_2$	$\|Gx - d\|_2$	accuracy [%]	time	t	λ
1	1.8573e+001	2.0010e+001	65.39	2.794	652.166	-9.8851e-001
2	2.8472e+001	1.1095e+001	49.63	3.054	994.155	-3.4166e-001
3	3.7079e+001	4.0222e+000	38.07	3.014	1271.46	-7.7717e-002
4	4.1957e+001	1.5642e+000	31.82	3.695	1378.38	-7.7959e-003
5	4.9732e+001	1.0570e+000	57.14	6.509	1392.12	-5.3731e-004
6	6.8218e+001	7.9497e-001	116.29	5.558	1393.45	-1.0426e-004
+1	4.2910e+001	1.4078e+000	32.63	2.834	1384.90	-4.1666e-003
+2	5.3732e+001	9.7305e-001	71.49	2.794	1392.69	-3.2078e-004
+3	4.3991e+001	1.2993e+000	35.36	2.824	1388.32	-2.3520e-003

Table 5.2: Data for points visited by the RPTRS algorithm
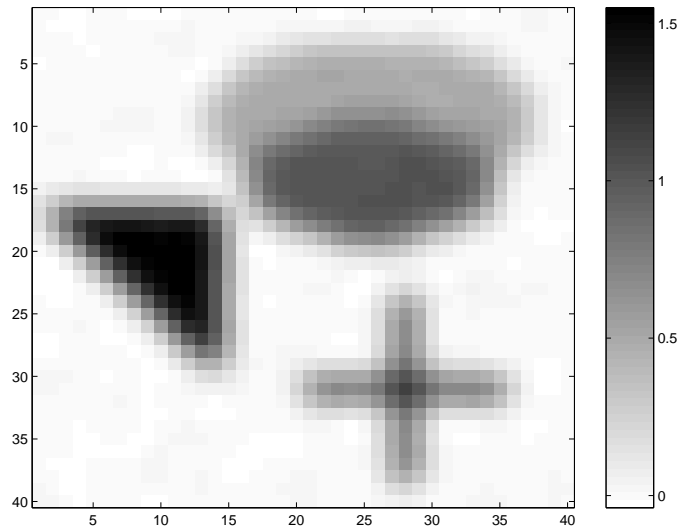
Figure 5.9: Image deblurring example: point #1, t = 652.166, rel.acc. = 65.39%
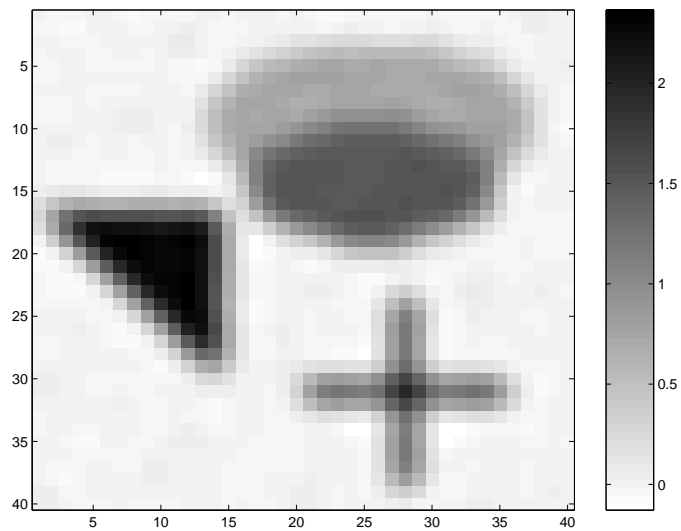
Figure 5.10: Image deblurring example: point #2, t = 994.155, rel.acc. = 49.63%
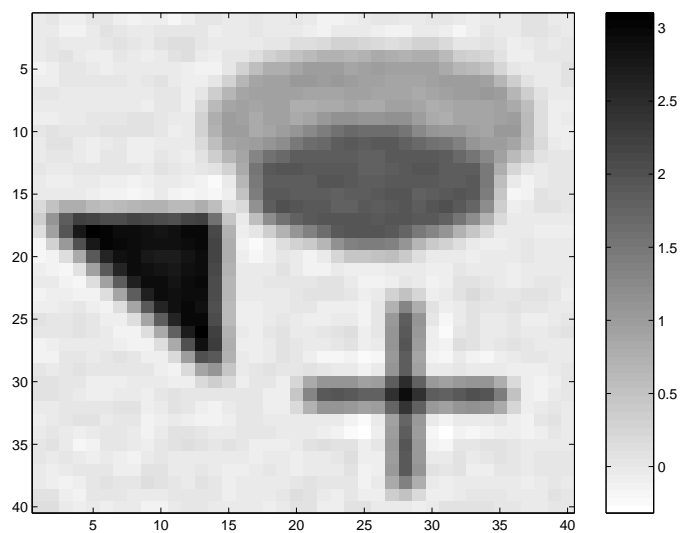
Figure 5.11: Image deblurring example: point #3, t = 1271.46, rel.acc. = 38.07%
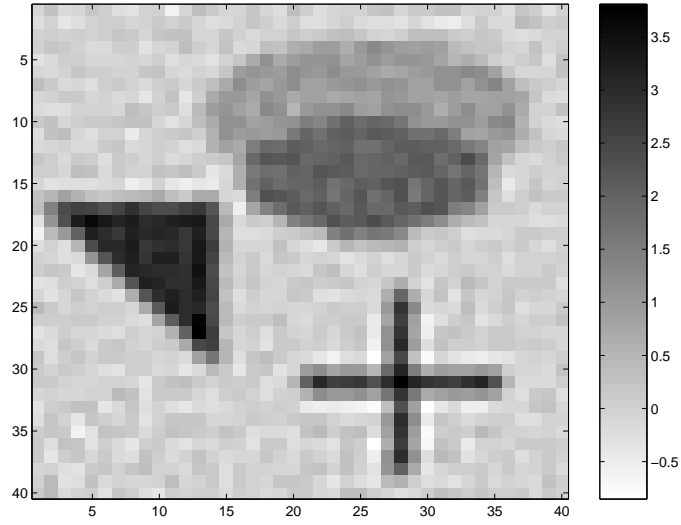
Figure 5.12: Image deblurring example: point #4, t = 1378.38, rel.acc. = 31.82%
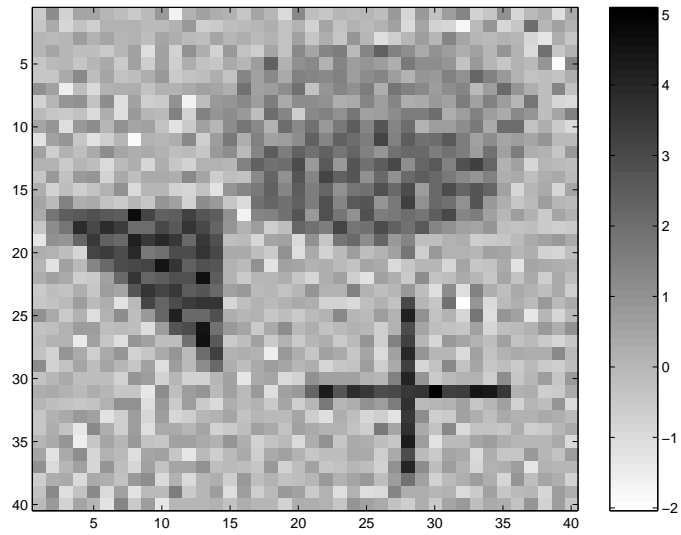
Figure 5.13: Image deblurring example: point #5, t = 1392.12, rel.acc. = 57.14%
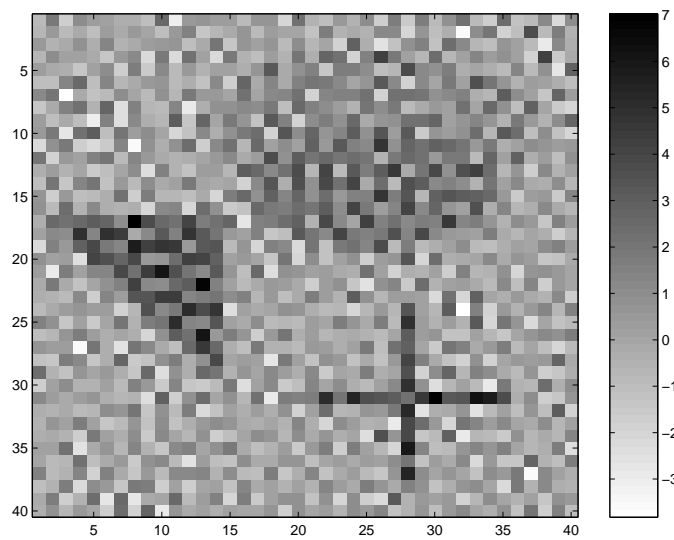
Figure 5.14: Image deblurring example: point #6, t = 1393.45, rel.acc. = 116.29%

Appendix A

MATLAB Code

A.1 RPTRS Regularization Algorithm

```
0001 function [x, res, alpha] = RPTRS (G, d, x_bar)
0002
0003 % ----------------------------------------------------------------------
0004 % Parameterized Trust Region Subproblem Regularization Algorithm
0005 %
0006 % solves the problem min ||Gx - d||
0007 %
0008 % INPUT:
0009 %   G     -- operator matrix
0010 %   d     -- rhs (observed data)
0011 %   x_bar -- true solution (optional), this parameter is used for
0012 %            accuracy calculations
0013 %
0014 % OUTPUT:
0015 %   x     -- regularized solution
0016 %   res   -- norm of the residual
0017 %   alpha -- Tikhonov regularization parameter
0018 %
0019 % ----------------------------------------------------------------------
0020 % Developed by Oleg Grodzevich as a part of Master of Mathematics Thesis,
0021 % University of Waterloo, Combinatorics and Optimization department.
```

```
0022 %
0023 % E-mail: illinar@mindon.net
0024 % ----------------------------------------------------------------------
0025
0026 global A a gamma delta bidiag_p bidiag_q
0027
0028 % ----------------------------------------------------------------------
0029 % Initialization
0030 % ----------------------------------------------------------------------
0031
0032 % x_bar is optional
0033 if nargin < 3, x_bar = ones(size(d,1),1); end
0034
0035 % in fact we do not need A matrix explicitly, this is only needed to work
0036 % around a bug in the implementation of eigs()
0037 A  = G'*G;
0038 a  = G'*d;
0039 dd = d'*d;
0040
0041 % configuration options
0042 lslope_tol1 = 1;
0043
0044 % compute the largest singular value of G
0045 time = cputime; sigmaLA = svds(G,1); time = cputime - time;
0046
0047 % initial interval [low,up) for t and lambda
0048 t_low   = 0;
0049 t_up    = dd;
0050 l_low   = -sigmaLA^2;
0051 l_up    = 0;
0052 itcount = 1;                    % iterations counter
0053 phist   = [];                   % points history
0054 pkappaU = 1e10;                 % previous curvature
0055 pkappaL = 1e10;
0056 nx_bar  = norm(x_bar);
0057
0058 % largest singular value computation time
0059 disp([' ']);
```

```matlab
0060 disp(['   +-------------------------------------', ...
0061       '-------------------------------------+']);
0062 disp([sprintf('   | initial time for sigmaLA = %7g %48c|', time, ' ')]);
0063
0064 % bidiagonalize matrix G
0065 time = cputime;
0066 [gamma, delta, bidiag_p, bidiag_q] = lbidiag2 (G, d);
0067 time = cputime - time;
0068 disp([sprintf('   | initial time for bidiagG = %7g %48c|', time, ' ')]);
0069
0070 % solve for the initial point: lambda -> t
0071 lambda = l_low;
0072 [t,x,k,time] = l2t(lambda);
0073
0074 % info header
0075 disp(['   +----+-------------+-------------+-------------+', ...
0076       '---------+---------+-------------+']);
0077
0078 disp(['   | ## |      norm(x) |   norm(Gx-d) | accuracy [%] |', ...
0079       '     time |        t |       lambda |']);
0080
0081 disp(['   +----+-------------+-------------+-------------+', ...
0082       '---------+---------+-------------+']);
0083
0084 % ----------------------------------------------------------------
0085 % Loop
0086 % ----------------------------------------------------------------
0087 l_damp =  1e-11;
0088 while lambda < l_up - l_damp
0089
0090   % calculate the slope of L-curve and d(lambda)/dt = y(1)
0091   eps2   = x'*x;                % norm of the solution squared
0092   r2     = k+dd;                % norm of the residual squared
0093   lslope = lambda*eps2/r2;
0094   dldt   = 1/(1+eps2);
0095
0096   % save current point
0097   pt     = [t lambda sqrt(eps2) sqrt(r2) lslope dldt];
```

```
0098   phist  = [pt ; phist];
0099
0100   % update lower bounds on t and lambda
0101   t_low  = pt(1);
0102   l_low  = pt(2);
0103
0104   % relative accuracy
0105   acc    = norm(x - x_bar)/nx_bar;
0106
0107   % plot point on the L-curve
0108   hold on; loglog(pt(3),pt(4),'rx'); hold off;
0109
0110   % information
0111   disp([ ...
0112   sprintf('   | %2d | %12.4e | %12.4e | %12.2f | %7g | %7g | %12.4e |',...
0113   itcount, pt(3), pt(4), acc*100, time, pt(1), pt(2))]);
0114
0115   disp(['   +----+-------------+-------------+--------------+', ...
0116           '---------+--------+--------------+']);
0117
0118   % calculate curvature
0119   [kappaL, kappaU, time] = curvature(G, d, pt(3), r2, pt(2));
0120
0121   disp([ ...
0122     sprintf('   | curvature  = (%12.4e,%12.4e)  time = %7g %26c|', ...
0123     kappaL, kappaU, time, ' ')]);
0124
0125   % --- termination
0126   stop = false;
0127   done = false;
0128   while ~stop
0129     if kappaL > pkappaU
0130       % return previous solution
0131       res   = phist(2,4);
0132       alpha = sqrt(-phist(2,2));
0133       done  = true;
0134       break;
0135     end
```

```
0136
0137     if kappaU < pkappaL, pkappaL = kappaL; pkappaU = kappaU; stop = true;
0138     else
0139       time = cputime;
0140       [gamma, delta, bidiag_p, bidiag_q] = ...
0141         lbidiag2(G, d, gamma, delta, bidiag_p, bidiag_q, length(gamma)+1);
0142       [kappaL, kappaU, time0] = curvature(G, d, pt(3), r2, pt(2));
0143       time = cputime - time;
0144
0145       disp([ ...
0146         sprintf('   | curvature  = (%12.4e,%12.4e)  time = %7g %26c|', ...
0147         kappaL, kappaU, time, ' ')]);
0148
0149       % recalculate previous kappa bounds
0150       [pkappaL, pkappaU, time] = ...
0151         curvature(G, d, phist(2,3), phist(2,4)^2, phist(2,2));
0152
0153       disp([ ...
0154         sprintf('   | previous   = (%12.4e,%12.4e)  time = %7g %26c|', ...
0155         pkappaL, pkappaU, time, ' ')]);
0156     end
0157   end
0158
0159   % true solution should sit between two last points
0160   if done, break; end
0161
0162   disp([sprintf('   | slope      = %12.4e %57c|', pt(5), ' ')]);
0163
0164   % possible pause before next iteration
0165   % keyboard
0166
0167   % target epsilon is the current one
0168   tgt_eps = pt(3);
0169
0170   % do triangle interpolation on k(t) to obtain the estimate for t
0171   e2p1 = tgt_eps^2 + 1;
0172
0173   % we choose the point t=dd,k(t)=-dd for the second pivot, as we
```

```matlab
0174   % know this point is always there
0175   k1    = e2p1*pt(2)-pt(1);
0176   l1    = e2p1*pt(6)-1;
0177   k2    = -dd;
0178   l2    = -1;                      % slope ~ -1 there, since y(1)->0
0179   [t,u] = tinterpl (pt(1), dd, k1, k2, l1, l2);
0180
0181   % display target epsilon
0182   disp([sprintf('   | target eps = %12.4e %57c|', tgt_eps, ' ')]);
0183
0184   % information about projected point
0185   disp([sprintf('   | target t   = %12.4e %57c|', t, ' ')]);
0186   disp(['   +----+-------------+-------------+-------------+', ...
0187         '---------+---------+-------------+']);
0188
0189   % now tgt_t is the projected point, check bounds, however both
0190   % MUST be good
0191   if t <= t_low || t >= t_up
0192     disp('>> PANIC: this should never happen!');
0193     return;
0194   end
0195
0196   % save previous solution
0197   soln = x;
0198
0199   % need previous lambda as an initial guess
0200   if itcount > 1, [lambda,x,k,time,y] = t2l(t,lambda,y);
0201   else,           [lambda,x,k,time,y] = t2l(t,lambda); end
0202
0203   itcount = itcount + 1;
0204 end
0205
0206 % info footer
0207 disp(['   +----------------------------------------', ...
0208      '----------------------------------------+']);
0209
0210 % --------------------------------------------------------------------
0211 % Refine between the last two points
```

```
0212 % --------------------------------------------------------------------
0213 ptR     = phist(1,:);           % right  point
0214 ptC     = phist(2,:);           % center point
0215 ptL     = phist(3,:);           % left   point
0216
0217 % largest known curvature so far
0218 crvC    = (pkappaL + pkappaU)/2;
0219
0220 disp([ ...
0221   sprintf('   | curvature [largest] = %12.4e                %36c|', ...
0222   crvC, ' ')]);
0223
0224 disp(['    +------------------+-------------+-------------+', ...
0225           '---------+---------+-------------+']);
0226
0227 while ~stop
0228
0229   done     = false;
0230   interval = -1;
0231
0232   while ~done
0233     % left/right intervals
0234     if interval < 0, lambda = (ptC(2)+ptL(2))/2;
0235     else             lambda = (ptC(2)+ptR(2))/2; end
0236
0237     [t,x,k,time] = l2t(lambda);
0238
0239     eps2    = x'*x;                % norm of the solution squared
0240     r2      = k+dd;                % norm of the residual squared
0241
0242     % relative accuracy
0243     acc     = norm(x - x_bar)/nx_bar;
0244
0245     % plot point on the L-curve
0246     hold on; loglog(sqrt(eps2),sqrt(r2),'co'); hold off;
0247
0248     % information
0249     disp([ ...
```

```
0250       sprintf('    |     %12.4e | %12.4e | %12.2f | %7g | %7g | %12.4e |',...
0251       sqrt(eps2), sqrt(r2), acc*100, time, t, lambda)]);
0252
0253       disp(['   +----------------+------------+-------------+', ...
0254               '--------+--------+-------------+']);
0255
0256       % calculate curvature
0257       [kappaL, kappaU, time] = curvature(G, d, sqrt(eps2), r2, lambda);
0258       disp([ ...
0259         sprintf('    | curvature  = (%12.4e,%12.4e)  time = %7g %26c|', ...
0260         kappaL, kappaU, time, ' ')]);
0261
0262       disp(['   +----------------+------------+-------------+', ...
0263               '--------+--------+-------------+']);
0264
0265       % curvature
0266       crv = (kappaL+kappaU)/2;
0267
0268       if crv < crvC
0269           % solution
0270           soln  = x;
0271           res   = sqrt(r2);
0272           alpha = sqrt(-lambda);
0273
0274           done = true;
0275           stop = true;
0276       end
0277
0278       if interval < 0, ptL = [t lambda sqrt(eps2) sqrt(r2)];
0279       else              ptR = [t lambda sqrt(eps2) sqrt(r2)]; end
0280
0281       if interval < 0, interval = 1;
0282       else              done  = true; end
0283     end
0284 end
0285
0286 % info footer
0287 disp([ ...
```

```
0288 sprintf('   | norm(x) = %12.4e, alpha = %12.4e, accuracy = %5.2f %20c|',...
0289 norm(soln), alpha, acc*100, ' ')]);
0290
0291 disp(['   +---------------------------------------',...
0292      '---------------------------------------+']);
0293
0294 % ----------------------------------------------------------------------
0295 % --- END of main function
0296 % ----------------------------------------------------------------------
0297
0298 % ----------------------------------------------------------------------
0299 % --- conversion lambda -> t
0300 % ----------------------------------------------------------------------
0301 function [t,x,k,time] = l2t(l)
0302 global A a
0303
0304 time = cputime; x = (A-l*speye(size(A)))\a; time = cputime - time;
0305 t = l + a'*x;
0306 k = (x'*x+1)*l - t;
0307
0308 % ----------------------------------------------------------------------
0309 % --- conversion t -> lambda
0310 % ----------------------------------------------------------------------
0311 function [l,x,k,time,y] = t2l(t, lambda, y)
0312 global A a
0313
0314 % options for eigs
0315 opts.tol   = 1e6*eps;
0316 opts.issym = 1;
0317 opts.disp  = 0;
0318
0319 % starting eigenvector (seems that this doesn't help much)
0320 if nargin > 2, opts.v0 = y; end
0321
0322 % construct matrix explicitly or eigs is doing crazy things
0323 % note: this is confirmed to be a bug
0324 time = cputime;
0325 D = [t -a' ; -a A]; [y,l,inf] = eigs(D, 1, lambda, opts);
```

```
0326 time = cputime-time;
0327
0328 if inf > 0
0329   disp('>> WARNING: EIGS did not converge!');
0330 end
0331
0332 % normalize the sign of the eigenvector
0333 if y(1) < 0, y = -y; end
0334
0335 eps2 = (1 - y(1)^2)/y(1)^2;
0336 k    = (eps2+1)*l - t;
0337 x    = y(2:end)/y(1);
0338
0339 % ----------------------------------------------------------------------
0340 % --- triangle interpolation
0341 % ----------------------------------------------------------------------
0342 function [t,y] = tinterpl(t1, t2, k1, k2, slope1, slope2)
0343
0344 tt = pinv([-slope1 1 ; -slope2 1])*[k1 - t1 * slope1 ; k2 - t2 * slope2];
0345 t  = tt(1);
0346 y  = tt(2);
0347
0348 % ----------------------------------------------------------------------
0349 % --- compute a'(A-l*I)^-3a: naive
0350 % ----------------------------------------------------------------------
0351 function [v,time] = est3naive (l)
0352 global A a
0353
0354 time  = cputime;
0355 v     = (A-l*speye(size(A)))\a;
0356 w     = (A-l*speye(size(A)))\v;
0357 v     = v'*w;
0358 time  = cputime - time;
0359
0360 % ----------------------------------------------------------------------
0361 % --- compute a'(A-l*I)^-3a: robust
0362 % ----------------------------------------------------------------------
0363 function [Gp,Rp,time] = est3robust (G, d, l)
```

63

```matlab
0364 global gamma delta bidiag_p bidiag_q
0365
0366 time    = cputime;
0367 [Gp,Rp] = estgr(gamma, delta, G, d, -1, -3);
0368 time    = cputime - time;
0369
0370 % ---------------------------------------------------------------------------
0371 % --- curvature
0372 % ---------------------------------------------------------------------------
0373 function [kappaL,kappaU,time] = curvature(G, d, e, m, l)
0374
0375 time   = cputime;
0376 [Gp,Rp,time0] = est3robust(G,d,l);
0377
0378 const0 = (e^2)*m;
0379 const1 = const0*((e^4)*(l^2)+(m^2))^(-3/2);
0380 const2 = 2*(e^2)*(l^2)-2*m*l;
0381
0382 kappaL = const1*(const2-const0/Gp);
0383 kappaU = const1*(const2-const0/Rp);
0384 time   = cputime - time;
```

A.2   Lanczos Bidiagonalization II Algorithm

```matlab
0001 function [gamma, delta, p, q] = lbidiag2 (G, d, gamma, delta, p, q, k)
0002
0003 % ----------------------------------------------------------------------
0004 % Lanczos Bidiagonalization II Algorithm
0005 %
0006 % References:
0007 % [1] G. H. GOLUB and U. von MATT,
0008 %     "Generalized Cross-Validation for Large-Scale Problems", 1996,
0009 %     TR-96-28
0010 %
0011 % [2] G. H. GOLUB and U. von MATT,
0012 %     "Tikhonov Regularization for Large Scale Problems", 1997, SCCM-97-03
0013 %
0014 % ----------------------------------------------------------------------
0015 % Developed by Oleg Grodzevich as a part of Master of Mathematics Thesis,
0016 % University of Waterloo, Combinatorics and Optimization department.
0017 %
0018 % E-mail: illinar@mindon.net
0019 % ----------------------------------------------------------------------
0020
0021 % ----------------------------------------------------------------------
0022 % Initialization
0023 % ----------------------------------------------------------------------
0024 [m,n] = size(G);              % dimension of matrix G
0025 normG = normest(G);          % estimated norm of G
0026 kmax  = sqrt(min(m,n));      % upperbound on number of iterations
0027 kmin  = ceil(3*log(min(m,n))); % lowerbound on number of iterations
0028 tol   = max(m,n)*10*eps*normG; % stopping tolerance
0029
0030 if nargin <= 2
0031   gamma = [];                % result: above diagonal
0032   delta = [];                % result: diagonal
0033   k     = 1;                 % iteration index
0034   p     = d/norm(d, 2);      % starting vector
0035 else
0036   kmax  = 2*k;
```

```
0037 end
0038
0039 % ----------------------------------------------------------------------
0040 % Loop
0041 % ----------------------------------------------------------------------
0042 while k <= kmax % stopping criteria can be improved
0043
0044   if (k <= 1), q = G'*p;
0045   else,        q = G'*p - delta(k-1)*q; end
0046
0047   gamma(k) = norm(q, 2);
0048
0049   % termination criteria
0050   if k > kmin && abs (gamma(k)) <= tol, gamma = gamma(1:k-1); break; end
0051
0052   q       = q / gamma(k);
0053   p       = G*q - gamma(k)*p;
0054   delta(k) = norm(p, 2);
0055
0056   % termination criteria
0057   if k > kmin && abs (delta(k)) <= tol, break; end
0058
0059   p       = p / delta(k);
0060   k       = k+1;
0061
0062 end
```

A.3 Estimating curvature using Gauss/Gauss-Radau Quadrature

```
0001 function [Gp, Rp] = estgr (gamma, delta, G, d, alpha, p)
0002
0003 % ----------------------------------------------------------------------
0004 % Estimate lower/upper bounds for the expression d'G (G'G + alpha I)^p G'd
0005 % using Gauss/Gauss-Radau quadrature rules.
0006 %
0007 % References:
0008 % [1] G. H. GOLUB and U. von MATT,
0009 %     "Generalized Cross-Validation for Large-Scale Problems", 1996,
0010 %     TR-96-28
0011 %
0012 % [2] G. H. GOLUB and U. von MATT,
0013 %     "Tikhonov Regularization for Large Scale Problems", 1997, SCCM-97-03
0014 %
0015 % ----------------------------------------------------------------------
0016 % Developed by Oleg Grodzevich as a part of Master of Mathematics Thesis,
0017 % University of Waterloo, Combinatorics and Optimization department.
0018 %
0019 % E-mail: illinar@mindon.net
0020 % ----------------------------------------------------------------------
0021
0022 n         = length       (gamma);
0023 normd     = norm         (d,2);
0024 [Q, u, v] = lbidiagqr    (gamma, delta, sqrt(alpha));
0025 Bk        = sparse       (1:n,   1:n, gamma, n+1, n) + ...
0026             sparse       (2:n+1, 1:n, delta, n+1, n);
0027
0028 b         = lbidiagqtx   (Q, eye(2*n+1, 1)*normd);
0029 xi        = ubidiagsolve (u, v, b(1:n));
0030
0031 if     p == -1, Gp = Bk *xi; Gp = Gp(1)*normd;
0032 elseif p == -2, Gp = xi'*xi;
0033 elseif p == -3
0034   b   = lbidiagqtx   (Q, [zeros(n+1,1); xi/sqrt(alpha)]);
```

```
0035   eta = ubidiagsolve (u, v, b(1:n));
0036   Gp  = xi'*eta;
0037 end
0038
0039 % compute \tilde{U}_k
0040 [Q, u, v] = lbidiagqr (gamma, delta, 0) ;
0041 u(n) = 0;
0042
0043 [Q, u, v] = ubidiagqr    (u, v, sqrt(alpha));
0044 b         = ubidiagqtx   (Q, [zeros(n,1); eye(n,1)]);
0045 eta       = ubidiagsolve (u, v, b(1:n));
0046 zeta      = lbidiagsolve (u, v, eta);
0047 Rp        = norm(G'*d)^2*norm (zeta, 2)^2/alpha;
```

Note, functions **lbidiagqr, ubidiagqr, lbidiagqtx, ubidiagqtx, ubidiagsolve, lbidiagsolve** are external C functions that compute and operate with QR-decomposition. They are available from author upon request.

Bibliography

[1] R. ASTER, B. BORCHERS, and C. THURBER. *Parameter Estimation and Inverse Problems*. Academic Press, 2004.

[2] D. CALVETTI, P.C. HANSEN, and L. REICHEL. L-curve curvature bounds via Lanczos bidiagonalization. *Electron. Trans. Numer. Anal.*, 14:135–150 (electronic), 2002. Orthogonal polynomials, approximation theory, and harmonic analysis (Inzel, 2000).

[3] D. L. DONOHO and I. M. JOHNSTONE. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[4] D. L. DONOHO and I. M. JOHNSTONE. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

[5] D. L. DONOHO, I. M. JOHNSTONE, G. KERKYACHARIAN, and D. PICARD. Wavelet shrinkage: asymptopia? *Journal of the Royal Statistical Society, Ser. B*, 57:301–337, 1995.

[6] L. ELDÉN. Algorithms for the regularization of ill-conditioned least squares problems. *BIT*, 17:134–145, 1977.

[7] H. W. ENGL, M. HANKE, and A. NEUBAUER. *Regularization of Inverse Problems*. Kluwer Academic Publishers Group, Dordrecht, 1996.

[8] C. FORTIN and H. WOLKOWICZ. The trust region subproblem and semidefinite programming. *Optimization Methods and Software*, 19(1):41–67, 2004. special issue

dedicated to Jochem Zowes 60th birthday, Guest Editors: Florian Jarre and Michal Kocvara.

[9] M. FROH. Trust region subproblems and linear least-squares regularization. Master's thesis, University of Waterloo, 2003.

[10] W. GANDER. On the linear least squares problem with a quadratic constraint. Technical Report STAN-CS-78-697, Department of Computer Science, Stanford University, Stanford, CA, 1978.

[11] D.M. GAY. Computing optimal locally constrained steps. *SIAM J. Sci. Statist. Comput.*, 2:186–197, 1981.

[12] G. H. GOLUB and U. von MATT. Generalized cross-validation for large-scale problems. *Journal of Computational and Graphical Statistics*, 6(1):1–34, March 1997.

[13] G. H. GOLUB and U. von MATT. Tikhonov regularization for large scale problems. Technical Report SCCM-97-03, Stanford University, 1997.

[14] G. H. GOLUB and U. von MATT. Tikhonov regularization for large scale problems. In *Scientific computing (Hong Kong, 1997)*, pages 3–26. Springer, Singapore, 1997.

[15] J. HADAMARD. Sur les problhmes aux dirivies partielles et leur signification physique. *Princeton University Bulletin*, pages 49–52, 1902.

[16] J. HADAMARD. *Lectures on Cauchy's problem in linear partial differential equations.* Dover Publications, New York, 1953.

[17] M. HANKE and P.C. HANSEN. Regularization methods for large-scale problems. *Surveys Math. Indust.*, 3(4):253–315, 1993.

[18] P.C. HANSEN. Analysis of discrete ill-posed problems by means of the L-curve. *SIAM Rev.*, 34(4):561–580, 1992.

[19] P.C. HANSEN. Regularization tools: a Matlab package for analysis and solution of discrete ill-posed problems. *Numer. Algorithms*, 6(1-2):1–35, 1994.

[20] P.C. HANSEN. *Rank-deficient and discrete ill-posed problems.* SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998. Numerical aspects of linear inversion.

[21] P.C. HANSEN. The l-curve and its use in the numerical treatment of inverse problems. Technical report, Technical University of Denmark, 1999.

[22] P.C. HANSEN and D.P. O'LEARY. The use of the *L*-curve in the regularization of discrete ill-posed problems. *SIAM J. Sci. Comput.*, 14(6):1487–1503, 1993.

[23] F. NATTERER. *The Mathematics of Computerized Tomography.* Wiley, New York, 1986.

[24] A. NEMIROVSKII. The regularization properties of the adjoint gradient method in ill-posed problems. *USSR Comput. Math. and Math. Phys.*, 26(2):7–16, 1986.

[25] A. NEUMAIER. Solving ill-conditioned and singular linear systems: a tutorial on regularization. *SIAM Rev.*, 40(3):636–666 (electronic), 1998.

[26] F. RENDL and H. WOLKOWICZ. A semidefinite framework for trust region subproblems with applications to large scale minimization. *Math. Programming*, 77(2, Ser. B):273–299, 1997.

[27] M. ROJAS, S.A. SANTOS, and D.C. SORENSEN. A new matrix-free algorithm for the large-scale trust-region subproblem. Technical Report TR99-19, Rice University, Houston, TX, 1999.

[28] M. ROJAS and D.C. SORENSEN. A trust-region approach to the regularization of large-scale discrete forms of ill-posed problems. *SIAM J. Sci. Comput.*, 23(6):1842–1860 (electronic), 2002.

[29] C.B. SHAW, Jr. Improvement of the resolution of an instrument by numerical solution of an integral equation. *J. Math. Anal. Appl.*, 37:83–112, 1972.

[30] D.C. SORENSEN. Minimization of a large-scale quadratic function subject to a spherical constraint. *SIAM Journal on Optimization*, 7(1):141–161, 1997.

[31] R. STERN and H. WOLKOWICZ. Indefinite trust region subproblems and nonsymmetric eigenvalue perturbations. *SIAM J. Optim.*, 5(2):286–313, 1995.

[32] R.J. STERN and J.J. YE. Variational analysis of an extended eigenvalue problem. *Linear Algebra Appl.*, 220:391–417, 1995.

[33] A. N. TIKHONOV. Regularization of incorrectly posed problems. *Soviet Math.*, 4:1624–1627, 1963.

[34] A.N. TIKHONOV and V.Y. ARSENIN. *Solutions of Ill-Posed Problems*. V.H. Winston & Sons, John Wiley & Sons, Washington D.C., 1977. Translation editor Fritz John.

[35] R.J. VANDERBEI. *The Amateur Astrophotographer*. Forthcoming.

[36] U. von MATT. *Large Constrained Quadratic Problems*. Ph. D. thesis, Institute for Scientific Computing, ETH, Zürich, Switzerland, 1993.