**A&A manuscript no.**
(will be inserted by hand later)

**Your thesaurus codes are:**
**missing; you have not inserted them**

ASTRONOMY
AND
ASTROPHYSICS
6.2.1997

# Efficient implementations of the SOLA mollifier method

**R. M. Larsen**[1] **and P. C. Hansen**[2]

[1] Department of Computer Science, University of Aarhus, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark
[2] UNI•C, Danish Computing Center for Research and Education, Building 304, Technical University of Denmark, DK-2800 Lyngby, Denmark

Received ____________; accepted ____________

**Abstract.** We describe efficient implementations of the Subtractive Optimally Localized Averages (SOLA) mollifier method for solving linear inverse problems in, e.g., inverse helioseismology. We show that the SOLA method can be regarded as a constrained least squares problem, which can be solved by means of standard "building blocks" from numerical linear algebra. We compare the standard implementation of the SOLA algorithm with our new approaches based on bidiagonalization of the kernel matrix, which allow fast re-computation of the solution when the regularization parameter or the target function are changed. We also illustrate our methods with an example from helioseismology.

**Key words:** inverse problems – numerical methods – implementation – efficiency – bidiagonalization – Lanczos methods

## 1. Introduction

The Backus-Gilbert (BG) method, Backus & Gilbert (1968), also known as the method of optimally localized averages, has been used successfully in seismology, Parker (1977), helioseismology, Christensen-Dalsgaard et al. (1990) and other areas that involve inverse problems, Craig & Brown (1986). The BG method belongs to a class of methods known as *mollifier methods*. The underlying idea in these methods is that if we are given $m$ measurements $b_i$ which represent (noisy) values of $m$ functionals $k_i$ on an unknown function $f$, i.e.,

$$\int_0^1 k_i(x)\, f(x)\, dx + \epsilon_i = b_i \ , \qquad i = 1, \ldots, m \ , \tag{1}$$

*Send offprint requests to*: R. M. Larsen

where $\epsilon_i$ represent the noise, then we estimate $f$ in a given point $x_0$ by a linear combination of the measurements:

$$f_{\rm est}(x_0) = \sum_{i=1}^{m} q_i(x_0)\, b_i = \mathbf{q}(x_0)^T \mathbf{b} \ . \tag{2}$$

Here, we have introduced the two $m$-vectors

$$\mathbf{q}(x_0) = (q_1(x_0), \ldots, q_m(x_0))^T \ , \quad \mathbf{b} = (b_1, \ldots, b_m)^T \ .$$

If the noise is unbiased and has covariance matrix $\mathbf{E}$, then the variance $\sigma^2(x_0)$ of the error in $f_{\rm est}(x_0)$ due to the noise in $\mathbf{b}$ is given by

$$\sigma^2(x_0) = \mathbf{q}(x_0)^T\, \mathbf{E}\, \mathbf{q}(x_0)\,. \tag{3}$$

The BG method is a particular technique for computing the coefficients $q_i(x_0)$ explicitly. If we compare this approach with Tikhonov regularization of a discrete ill-posed problem $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, whose solution is formally given by $(\mathbf{A}^T\mathbf{A} + \lambda^2\,\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}$ for some $\lambda$, then we notice that $\mathbf{q}(x_0)^T$ in a mollifier method conceptually corresponds to a row in the matrix $(\mathbf{A}^T\mathbf{A} + \lambda^2\,\mathbf{I})^{-1}\mathbf{A}^T$.

If $f_{\rm exact}$ denotes the exact solution, then for any regularization method it is easy to see from equation (1) that $f_{\rm est}(x_0)$ is given by

$$f_{\rm est}(x_0) = \int_0^1 \sum_{i=1}^{m} q_i(x_0)\, k_i(x)\, f_{\rm exact}(x)\, dx + \sum_{i=1}^{m} q_i(x_0)\, \epsilon_i \ .$$

The function

$$\mathcal{K}(x_0, x) \equiv \sum_{i=1}^{m} q_i(x_0)\, k_i(x) \tag{4}$$

is called the *averaging kernel* at $x_0$, and it shows how the noise-free part of the regularized estimate $f_{\rm est}(x_0)$ is obtained as an average of the exact solution $f_{\rm exact}$. Thus, $\mathcal{K}(x_0, x)$ determines the resolving power of the particular regularization method at $x_0$, and for $f_{\rm est}(x_0)$ to be meaningful the function $\mathcal{K}(x_0, x)$ should be peaked around $x = x_0$ and be normalized such that

$$\int_0^1 \mathcal{K}(x_0, x)\, dx = 1 \ . \tag{5}$$

Hence, the averaging kernel $\mathcal{K}(x_0, x)$ often plays a role which is at least as important as the computed estimate $f_{\rm est}(x_0)$.

In the BG method the peakedness of $\mathcal{K}(x_0, x)$ is controlled by specifying a criteria function $J(x_0, x)$ and requiring that the quantity

$$\int_0^1 J(x_0, x)\, \mathcal{K}(x_0, x)^2\, dx \tag{6}$$

be sufficiently small, subject to the constraint in equation (5). Often, a regularization term is also added in equation (6). A commonly used criteria function is $J(x_0, x) = 12(x - x_0)^2$. Various other criteria functions are discussed in the original paper by Backus & Gilbert (1968), while the additional regularization term is not used there. See Hansen (1994) for an SVD-analysis and computational details.

The BG method is computationally very expensive: If $n$ is the number of quadrature points used to evaluate the integrals then $\mathcal{O}(mn^2)$ operations are required for each $x_0$. This is prohibitively expensive in large-scale applications. In, e.g., 2D rotational inversion in helioseismology $m$ is of the order $10^5$–$10^6$ and $n$ is of the order $10^3$–$10^4$.

Therefore, there is a need for computationally more efficient methods. One such method is *subtractive optimally localized averages* (SOLA), which has been discovered independently by several researchers; see, e.g., Louis & Maass (1990) and Pijpers & Thompson (1992). The idea in the SOLA method is, instead of minimizing (6), to minimize the squared difference between the averaging kernel and some chosen target function $\mathcal{T}(x_0, x)$,

$$\int_0^1 [\mathcal{K}(x_0, x) - \mathcal{T}(x_0, x)]^2\, dx\,, \tag{7}$$

and it is common practice to retain the constraint in equation (5), see Pijpers & Thompson (1992, 1994). The peakedness of $\mathcal{K}(x_0, x)$ is controlled by the choice of $\mathcal{T}(x_0, x)$, and therefore to compute $f_{\rm est}(x_0)$ one would choose a target function peaked around $x_0$. In practice it is often necessary to add a regularizing term, i.e., to replace equation (7) with

$$\int_0^1 [\mathcal{K}(x_0, x) - \mathcal{T}(x_0, x)]^2 dx \ + \ \lambda^2 \mathbf{q}(x_0)^T \mathbf{E} \mathbf{q}(x_0)\,. \tag{8}$$

Here $\mathbf{E}$ is the error covariance matrix of the measured data points $b_i$ and $\lambda$ is a trade-off or regularization parameter whose value determines the relative importance of the two terms in equation (8). Thus, $\lambda$ controls the propagation of errors from the measurements to $f_{\rm est}(x_0)$.

The great advantage of the SOLA method compared to the BG method is that the $m \times n$ coefficient matrix involved does not depend on $x_0$ and therefore its factorization, costing $\mathcal{O}(mn^2)$ operations, is performed only once. The additional cost to compute $f_{\rm est}(x_0)$ is $\mathcal{O}(n^2)$ for each $x_0$.

In this paper we discuss two different approaches to implementing the SOLA method: The standard approach using a Lagrange multiplier, and a new approach that uses an orthogonal transformation to incorporate the constraint in equation (5) and bidiagonalization algorithms to solve the resulting least squares problem. We demonstrate that the latter approach is more efficient when many values of the regularization parameter $\lambda$ must be tried in order to find the optimal one. This is important, since the optimal $\lambda$ is very rarely known in advance – instead it must be computed by means of parameter-choice methods, such as *generalized cross validation* or the *L-curve criterion*, that involve the optimization of a function that depends on $\lambda$; see Hanke & Hansen (1993) .

Our paper is organized as follows. In Section 2 we introduce our notation and present the algorithms. These algorithms are based on standard linear algebra routines which are available in most scientific software libraries, and in Section 3 we point to available Fortran software. The computational load in the algorithms, and the efficiency of our new approach, is discussed in Section 4. Finally, in Section 5 we illustrate our algorithms with an example from helioseismology.

## 2. Implementations of the SOLA Method

In the following we summarize the SOLA method. The first step is to introduce a proper matrix formulation of the problem. Assume that the data-errors $\epsilon_i$ are described by the covariance matrix $\mathbf{E}$ and that $\mathbf{E}$ has a lower triangular Cholesky factor $\mathbf{C}$, such that $\mathbf{E} = \mathbf{C}\,\mathbf{C}^T$. In case the data are uncorrelated with equal variance, $\mathbf{E}$ will be a scalar times the identity matrix. If the statistics of the data are unknown it is common to set $\mathbf{E}$ equal to the identity. Also assume that we use a quadrature rule with weights $w_j$ on the grid $x_j$, $j = 1, \ldots, n$ to compute the necessary integrals, i.e.,

$$\int_0^1 k_i(x)\, dx \approx \sum_{j=1}^{n} w_j k_i(x_j)\,.$$

We now define the $m \times n$ matrix $\mathbf{K}$,

$$(\mathbf{K})_{ij} = k_i(x_j)\ , \qquad i = 1, \ldots, m\ , \quad j = 1, \ldots, n\ .$$

To express the integrals involved we define the $n \times n$ diagonal matrix $\mathbf{W}$ with diagonal elements

$$(\mathbf{W})_{jj} = w_j\ , \qquad j = 1, \ldots, n\ .$$

Finally we introduce the three $n$-vectors $\mathbf{e}_n = (1, \ldots, 1)^T$, $\mathbf{t}(x_0)$ with elements $(\mathbf{t}(x_0))_j = \mathcal{T}(x_0, x_j)$, $j = 1, \ldots, n$ and $\mathbf{k}(x_0)$ with elements $(\mathbf{k}(x_0))_j = \mathcal{K}(x_0, x_j)$, $j = 1, \ldots, n$.

Now we can write the discretized version of expression (8) subject to the constraint in equation (5): $\mathbf{q}(x_0)$ is given as the solution to

$$\left(\mathbf{KWK}^T + \lambda^2\mathbf{E}\right)\mathbf{q}(x_0) = \mathbf{KWt}(x_0)$$
$$\text{subject to} \quad \mathbf{e}_n^T\mathbf{WK}^T\mathbf{q}(x_0) = 1 \; ; \tag{9}$$

and we obtain the discretized version of the averaging kernel as

$$\mathbf{k}(x_0) = \mathbf{K}^T\mathbf{q}(x_0) \; . \tag{10}$$

The kernels $k_i(x)$ are often nearly linearly dependent, and consequently the same holds for the rows of $\mathbf{K}$ which are "samples" of the kernels. Therefore the matrix $\mathbf{K}$ is usually very ill-conditioned, and this motivates the additional regularization term $\lambda^2\mathbf{E}$ in equation (9).

Typically the number of kernels $m$ is much larger than the number of points $n$ in the discretization. This means that the system of equations in equation (9) is underdetermined if $\lambda = 0$. Only when we include the statistical information about the data, represented in the covariance matrix $\mathbf{E}$, to regularize the problem, do we obtain a unique solution. A convenient way to incorporate the statistical information into the least squares problem is to work with the transformed matrix $\mathbf{C}^{-1}\,\mathbf{K}$ and right-hand side $\mathbf{C}^{-1}\,\mathbf{b}$, where $\mathbf{C}$ is the Cholesky factor of $\mathbf{E}$, and the covariance for the errors in $\mathbf{C}^{-1}\,\mathbf{b}$ is now the identity matrix and the variance of the error in $f_{\text{est}}(x_0)$ due to the noise is simply $\|\mathbf{q}(x_0)\|_2^2$. To simplify our presentation, we assume that this transformation to unit covariance form is applied to the data before the SOLA method is applied.

In the following we will describe different algorithms for solving equation (9). In Section 2.1 we describe the algorithm suggested in Pijpers & Thompson (1992, 1994). As we show in Section 2.2, another possibility is to reduce equation (9) to a standard unconstrained regularized least squares problem, using the method proposed in Lawson & Hanson (1974), Chap. 20. For the unconstrained least squares problem a variety of efficient regularization methods are available; see, e.g., the survey in Hanke & Hansen (1993). As we will demonstrate, the latter approach is superior in terms of computational efficiency, especially when many values of $\lambda$ are needed. We also discuss an iterative algorithm which is useful when $\mathbf{K}$ is either sparse or structured.

In the derivation of the algorithms below we have restricted ourselves to the case where the estimate of $f$ is computed for a single point $x_0$. The extension of the algorithms to multiple points is trivial and is given in the summaries at the end of each section. Finally we note that we measure the work associated with the algorithms by the number of floating point operations (flops) used. A "flop" is a simple arithmetic operation such as a floating point multiplication or addition.

### *2.1. SOLA algorithm using a Lagrange multiplier*

The standard solution procedure, which is the one used in Pijpers & Thompson (1992), incorporates the constraint using a Lagrange multiplier and solves the augmented normal equations of (9). This means solving the system

$$\begin{pmatrix} \mathbf{KWK}^T + \lambda^2\mathbf{I} & \mathbf{KWe}_n \\ \mathbf{e}_n^T\mathbf{WK}^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{q}(x_0) \\ \mu \end{pmatrix} = \begin{pmatrix} \mathbf{KWt}(x_0) \\ 1 \end{pmatrix} \tag{11}$$

where $\mu$ is the Lagrange multiplier. This approach has several disadvantages:

- There is a large potential for loss of accuracy in forming $\mathbf{KWK}^T$.
- The augmented system is very large: $(m+1)\times(m+1)$.
- Although the augmented normal equations are symmetric they are not necessarily positive definite. Therefore the Cholesky factorization, which is normally used to solve the normal equations, cannot be applied to equation (11), neither can an iterative method such as conjugate gradients. In Pijpers & Thompson (1994) it is suggested that equation (11) is solved using Gaussian elimination. This approach doubles the operation count compared to a Cholesky factorization.

A solution to the last problem is to apply the Bunch-Kaufman algorithm for symmetric indefinite systems, which computes an $\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T$ factorization using diagonal pivoting (see, e.g., Sections 4.4.4–5 in Golub & Van Loan (1989)), requiring $m^3/3$ flops, the same as the usual Cholesky factorization.

To avoid forming the large system of normal equations, it has been suggested by Christensen-Dalsgaard & Thompson (1993) to reduce the dimension of the system by first computing an SVD of the kernel matrix $\mathbf{K}$:

$$\mathbf{K} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T \; , \tag{12}$$

where $\mathbf{U} \in \mathbb{R}^{m\times n}$ and $\mathbf{\Sigma}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$. Then one switches to a system that uses the transformed quantities:

$$\hat{\mathbf{K}} \;=\; \mathbf{U}^T\mathbf{K} \;=\; \mathbf{\Sigma}\mathbf{V}^T \;\in\; \mathbb{R}^{n\times n},$$
$$\hat{\mathbf{b}} \;=\; \mathbf{U}^T\mathbf{b} \in \mathbb{R}^n \; .$$

Working with $\hat{\mathbf{K}}$ and $\hat{\mathbf{b}}$ instead of $\mathbf{K}$ and $\mathbf{b}$ reduces the computational effort by a factor of $\mathcal{O}(m/n)^2$. Now the most expensive part is the computation of $\mathbf{\Sigma}$ and $\mathbf{V}$ of the SVD which requires $2mn^2 + 11n^3$ flops (note that $\mathbf{U}$ is not computed explicitly; $\mathbf{U}^T\mathbf{b}$ is computed "on the fly"). Details on how to compute the SVD are given in, e.g., Golub & Van Loan (1989).

The algorithm, which is summarized below, describes the general case where $f_{\text{est}}$ is computed at points $x_1, x_2, \ldots, x_q$. The matrix $\mathbf{T}$ below is the matrix whose columns contain "samples" of the $q$ target function: $(\mathbf{T})_{ij} = (\mathbf{t}(x_j))_i = \mathcal{T}(x_j, x_i)$, $i = 1, \ldots, n$, $j = 1, \ldots, q$.

**Algorithm 2.1 SOLA using a Lagrange multiplier and normal equations.**

1. Transform to standard form
$$\mathbf{E} = \mathbf{C}\,\mathbf{C}^T$$
$$\mathbf{K} \leftarrow \mathbf{C}^{-1}\,\mathbf{K}$$
$$\mathbf{b} \leftarrow \mathbf{C}^{-1}\,\mathbf{b}$$
2. Perform SVD reduction (optional)
$$\mathbf{K} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T$$
$$\mathbf{K} \leftarrow \boldsymbol{\Sigma}\,\mathbf{V}^T = \mathbf{U}^T\,\mathbf{K}$$
$$\mathbf{b} \leftarrow \mathbf{U}^T\mathbf{b}$$
3. Set up the augmented normal equations and compute $\mathbf{LDL}^T$ factorization
$$\mathbf{A}_{11} \leftarrow \mathbf{K}\,\mathbf{W}\,\mathbf{K}^T + \lambda^2\,\mathbf{I}$$
$$\mathbf{A}_{12} \leftarrow \mathbf{K}\,\mathbf{W}\,\mathbf{e}_n$$
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{0} \end{pmatrix} = \mathbf{L}\,\mathbf{D}\,\mathbf{L}^T$$
4. Compute the right-hand side
$$\mathbf{T} \leftarrow \begin{pmatrix} \mathbf{K}\,\mathbf{W}\,(\mathbf{t}(x_1), \mathbf{t}(x_2), \ldots, \mathbf{t}(x_q)) \\ \mathbf{e}_q^T \end{pmatrix}$$
5. Solve for coefficients
$$\begin{pmatrix} \mathbf{Q} \\ \mu_1\,\mu_2\,\ldots\,\mu_q \end{pmatrix} = \left( \begin{pmatrix} \mathbf{q}(x_1) \\ \mu_1 \end{pmatrix}, \begin{pmatrix} \mathbf{q}(x_2) \\ \mu_2 \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{q}(x_q) \\ \mu_q \end{pmatrix} \right)$$
$$\leftarrow \mathbf{L}^{-T}(\mathbf{D}^{-1}(\mathbf{L}^{-1}\mathbf{T})) = (\mathbf{LDL}^T)^{-1}\mathbf{T}$$
6. Compute the estimates of $f$ and the corresponding variances at $x_1, x_2, \ldots, x_q$
$$(f_{\mathrm{est}}(x_1), f_{\mathrm{est}}(x_2), \ldots, f_{\mathrm{est}}(x_q))^T \leftarrow \mathbf{Q}^T\,\mathbf{b}$$
$$\left(\sigma^2(x_1), \sigma^2(x_2), \ldots, \sigma^2(x_q)\right)^T \leftarrow \left(\|\mathbf{q}(x_1)\|_2^2, \|\mathbf{q}(x_2)\|_2^2, \ldots, \|\mathbf{q}(x_q)\|_2^2\right)^T$$
7. If needed, compute the corresponding averaging kernels
$$(\mathbf{k}(x_1), \mathbf{k}(x_2), \ldots, \mathbf{k}(x_q))^T \leftarrow \mathbf{K}^T\mathbf{Q}$$

In practice one may wish to repeat the algorithm for different shapes of the target function $\mathcal{T}(x_0, x)$ and/or varying values of the regularization parameter $\lambda$. Changing the target functions requires re-computing steps 4, 5, 6 and 7. Changing $\lambda$ requires re-computing steps 3, 5, 6 and 7. In Section 4 we will give a detailed analysis of the number of operations used by the algorithm.

### *2.2. SOLA by elimination of the constraint*

We can identify equation (9) as an instance of the well known *equality constrained least squares* problem (LSE); see Lawson & Hanson (1974), Chap. 20. To realize this, let us introduce the following quantities:

$$\tilde{\mathbf{K}} = \mathbf{W}^{1/2}\,\mathbf{K}^T \in \mathbb{R}^{n\times m}$$
$$\tilde{\mathbf{t}}(x_0) = \mathbf{W}^{1/2}\,\mathbf{t}(x_0) \in \mathbb{R}^n$$
$$\mathbf{c} = \mathbf{K}\,\mathbf{W}\,\mathbf{e}_n \in \mathbb{R}^m.$$

Using this notation we can write (9) in the following form:

$$\text{Solve } \left(\tilde{\mathbf{K}}^T\tilde{\mathbf{K}} + \lambda^2\mathbf{I}\right)\mathbf{q}(x_0) = \tilde{\mathbf{K}}^T\tilde{\mathbf{t}}(x_0) \quad \text{subject to } \mathbf{c}^T\mathbf{q}(x_0) = 1\ . \tag{13}$$

These are in fact the normal equations for a constrained least squares problem, for which we have the equivalent expression:

$$\min \left\{ \|\tilde{\mathbf{K}}\mathbf{q}(x_0) - \tilde{\mathbf{t}}(x_0)\|_2^2 + \lambda^2\|\mathbf{q}(x_0)\|_2^2 \right\} \quad \text{subject to } \mathbf{c}^T\mathbf{q}(x_0) = 1\ . \tag{14}$$

We can reformulate this as

$$\min \left\| \begin{pmatrix} \tilde{\mathbf{K}} \\ \lambda\mathbf{I}_m \end{pmatrix} \mathbf{q}(x_0) - \begin{pmatrix} \tilde{\mathbf{t}}(x_0) \\ \mathbf{0} \end{pmatrix} \right\|_2 \quad \text{subject to } \mathbf{c}^T\mathbf{q}(x_0) = 1\ . \tag{15}$$

We now have an equality constrained least squares problem in unit covariance form. Several methods have been developed for solving problem LSE. Here we use the one described in Lawson & Hanson (1974), Chap. 20 and Golub & Van Loan (1989), Sect. 12.1.4. The method has three stages:

1. Derive a lower-dimensional unconstrained least squares problem from the given problem.
2. Compute the solution to the lower-dimensional problem.
3. Transform this solution back to the original setting.

Below we describe in detail how each of these three steps should be implemented to give an efficient SOLA algorithm.

#### 2.2.1. Derivation of a lower-dimensional problem

We can determine a parameterization of the set $\mathcal{Q}$ of vectors in $\mathbb{R}^m$ that satisfy the constraint in equation (15):

$$\mathcal{Q} = \{\, \mathbf{q} \in \mathbb{R}^m \mid \mathbf{c}^T\,\mathbf{q} = 1\}.$$

To do this we compute an $m \times m$ Householder transformation $\mathbf{H} = \mathbf{I} - \beta\mathbf{v}\mathbf{v}^T$ (see Golub & Van Loan (1989), Sect. 5.2) such that

$$\mathbf{c} = \mathbf{H}\begin{pmatrix} \|\mathbf{c}\|_2 \\ \mathbf{0} \end{pmatrix}\ .$$

In practice $\mathbf{H}$ is not computed explicitly, but is represented in the form $\mathbf{I} - \beta\mathbf{v}\mathbf{v}^T$. The columns of $\mathbf{H}$ form a basis for $\mathbb{R}^m$ and conceptually we can partition $\mathbf{H}$ as

$$\mathbf{H} = (\mathbf{h}_1, \mathbf{H}_2)\ , \quad \mathbf{h}_1 = \alpha\,\mathbf{c} \in \mathbb{R}^m\ , \quad \mathbf{H}_2 \in \mathbb{R}^{m\times(m-1)}\ ,$$

where $\alpha = \|\mathbf{c}\|_2^{-1}$. It follows that $\mathcal{Q}$ can be represented as

$$\mathcal{Q} = \{\, \mathbf{q} \in \mathbb{R}^m \mid \mathbf{q} = \alpha\mathbf{h}_1 + \mathbf{H}_2\hat{\mathbf{q}}\}\ , \tag{16}$$

where $\hat{\mathbf{q}}$ is an arbitrary $(m-1)$-vector (since all the columns of $\mathbf{H}_2$ are orthogonal to $\mathbf{c}$). We can exploit this fact to transform problem LSE into the following form, where we have inserted equation (16) into equation (15) and collected terms:

$$\min \left\| \begin{pmatrix} \tilde{\mathbf{K}}\,\mathbf{H}_2 \\ \lambda \mathbf{H}_2 \end{pmatrix} \hat{\mathbf{q}} - \begin{pmatrix} \tilde{\mathbf{t}}(x_0) - \alpha\tilde{\mathbf{K}}\mathbf{h}_1 \\ -\lambda\alpha\mathbf{h}_1 \end{pmatrix} \right\|_2 .$$

We now multiply from the left under the norm-sign with the orthogonal matrix

$$\begin{pmatrix} \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{pmatrix}$$

without changing the solution. Using the orthogonality of $\mathbf{H}$ we finally arrive at the reduced problem

$$\min \left\| \begin{pmatrix} \tilde{\mathbf{K}}\,\mathbf{H}_2 \\ \lambda \mathbf{I}_{m-1} \end{pmatrix} \hat{\mathbf{q}} - \begin{pmatrix} \tilde{\mathbf{t}}(x_0) - \alpha\tilde{\mathbf{K}}\mathbf{h}_1 \\ \mathbf{0} \end{pmatrix} \right\|_2 . \tag{17}$$

This is an ordinary unconstrained least squares problem. Having solved this for $\hat{\mathbf{q}}$ we obtain the solution to the original constrained problem as

$$\mathbf{q}(x_0) = \alpha\mathbf{h}_1 + \mathbf{H}_2\hat{\mathbf{q}} = \mathbf{H}\begin{pmatrix} \alpha \\ \hat{\mathbf{q}} \end{pmatrix} . \tag{18}$$

### 2.2.2. Solution via full bidiagonalization

We now describe an algorithm for solving (17) which is efficient when many values of $\lambda$ are required. Following Eldén (1977) we first bring $\tilde{\mathbf{K}}\,\mathbf{H}_2$ into bidiagonal form:

$$\tilde{\mathbf{K}}\,\mathbf{H}_2 = \bar{\mathbf{U}}\,\bar{\mathbf{B}}\,\bar{\mathbf{V}}^T , \tag{19}$$

where

$$\bar{\mathbf{B}} \in \mathbb{R}^{n\times n} , \quad \bar{\mathbf{U}} \in \mathbb{R}^{n\times n} , \quad \bar{\mathbf{V}} \in \mathbb{R}^{(m-1)\times n} . \tag{20}$$

Here $\bar{\mathbf{U}}^T\bar{\mathbf{U}} = \mathbf{I}_n$, $\bar{\mathbf{V}}^T\bar{\mathbf{V}} = \mathbf{I}_n$, and $\bar{\mathbf{B}}$ is lower bidiagonal. This is the most costly part of the computation, it requires $2mn^2 + 2n^3$ flops. The remaining part of the algorithm only involves $\bar{\mathbf{B}}$, which means that the dimension of the problem is effectively reduced from $m \times n$ to $n \times n$. As in the SVD transformation, $\bar{\mathbf{V}}$ is not explicitly computed, but $\bar{\mathbf{V}}^T\,\mathbf{H}_2^T\,\mathbf{b}$ is computed "on the fly". If $\bar{\mathbf{U}}$ is required for computing the discretized kernel $\mathbf{k}(x_0)$, then the orthogonal transformations that $\bar{\mathbf{U}}$ consists of are stored for later use. For algorithmic details on bidiagonalization see, e.g., Golub & Van Loan (1989), pp. 236–238.

If we make the transformation of variables

$$\begin{aligned} \boldsymbol{\xi} &= \bar{\mathbf{V}}^T\hat{\mathbf{q}}, \\ \bar{\mathbf{t}} &= \bar{\mathbf{U}}^T(\tilde{\mathbf{t}}(x_0) - \alpha\tilde{\mathbf{K}}\mathbf{h}_1) \end{aligned} \tag{21}$$

then we easily see that (17) is equivalent to

$$\min \left\| \begin{pmatrix} \bar{\mathbf{B}} \\ \lambda\mathbf{I}_n \end{pmatrix} \boldsymbol{\xi} - \begin{pmatrix} \bar{\mathbf{t}} \\ \mathbf{0} \end{pmatrix} \right\|_2 . \tag{22}$$

This least squares problem is now solved by determining a series of $2n-1$ Givens rotations $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_{2n-1}$ (see Golub & Van Loan (1989), Sect. 5.2) such that

$$\mathbf{G}_{2n-1}\ldots\mathbf{G}_2\mathbf{G}_1 \left(\begin{array}{c|c} \bar{\mathbf{B}} & \bar{\mathbf{t}} \\ \lambda\mathbf{I}_n & \mathbf{0} \end{array}\right) = \left(\begin{array}{c|c} \bar{\mathbf{B}}_\lambda & \bar{\mathbf{t}}_\lambda \\ \mathbf{0} & * \end{array}\right),$$

where $\bar{\mathbf{B}}_\lambda$ is $n \times n$ lower bidiagonal. The part of the right-hand side denoted by '$*$' consist of $n$ non-zero elements introduced along the way, but these are not computed in practice. We illustrate this procedure on a small example with $n = 3$. First a rotation $\mathbf{G}_1$ applied to rows $n$ and $2n$ annihilates the $(2n, n)$-element, and a nonzero element is created in position $(2n, n-1)$; then we annihilate this element by a rotation $\mathbf{G}_2$ applied to rows $2n-1$ and $2n$:

$$\left(\begin{array}{ccc|c} a_1 & & & t_1 \\ b_1 & a_2 & & t_2 \\ & b_2 & a_3 & t_3 \\ \lambda & & & \\ & \lambda & & \\ & & \lambda & \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} a_1 & & & t_1 \\ b_1 & a_2 & & t_2 \\ & b_2' & a_3' & t_3' \\ \lambda & & & \\ & \lambda & & \\ & \gamma & 0 & * \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} a_1 & & & t_1 \\ b_1 & a_2 & & t_2 \\ & b_2' & a_3' & t_3' \\ \lambda & & & \\ & \lambda' & & * \\ & 0 & 0 & * \end{array}\right)$$

where

$$a_3' = \sqrt{a_3^2 + \lambda^2} , \quad b_2' = b_2 a_3/a_3' , \quad t_3' = t_3 a_3/a_3'$$

$$\gamma = -b_2\lambda/a_3' , \quad \lambda' = \sqrt{\lambda^2 + \gamma^2} .$$

In the next step the same procedure is applied to the leading $(2n-1) \times (n-1)$ sub-matrix to annihilate the element in position $(2n-1, n-1)$, and so on. During the reduction we only need to store the diagonals of $\bar{\mathbf{B}}$ in a pair of vectors, and the current value of $\lambda'$. Now the solution $\boldsymbol{\xi}_\lambda$ can be computed as

$$\boldsymbol{\xi}_\lambda = \bar{\mathbf{B}}_\lambda^{-1}\bar{\mathbf{t}}_\lambda . \tag{23}$$

### 2.2.3. Solution via Lanczos bidiagonalization

For a large sparse or structured (e.g., Hankel or Toeplitz) matrix $\mathbf{K}$ it is prohibitive to compute explicitly the full bidiagonalization in equation (19). Instead we would prefer to keep $\tilde{\mathbf{K}}\,\mathbf{H}_2 = \mathbf{W}^{1/2}\,\mathbf{K}^T\,\mathbf{H}_2$ in factored form and use an iterative algorithm to solve the system in (17). An efficient and stable algorithm for doing this, based on Lanczos bidiagonalization (Golub & Van Loan 1989, Sect. 9.3.3), is the algorithm LSQR, Paige & Saunders (1982a). This algorithm only accesses the coefficient matrix $\tilde{\mathbf{K}}\,\mathbf{H}_2$ via matrix-vector multiplications with $\tilde{\mathbf{K}}\,\mathbf{H}_2$ and $(\tilde{\mathbf{K}}\,\mathbf{H}_2)^T$.

An even more efficient approach is to apply the LSQR algorithm with $\lambda = 0$ and use the fact that the $k$th iterate $\hat{\mathbf{q}}^{(k)}$ is a regularized solution. To be more specific, when we apply LSQR to equation (17) with $\lambda = 0$, i.e., to

$$\min \|(\tilde{\mathbf{K}}\,\mathbf{H}_2)\,\hat{\mathbf{q}} - (\tilde{\mathbf{t}}(x_0) - \alpha\tilde{\mathbf{K}}\mathbf{h}_1)\|_2 ,$$

then the iteration number $k$ plays the role of the regularization parameter; small $k$ correspond to large $\lambda$, and vice versa. The reason is that LSQR captures the spectral components of the solution in order of increasing frequency, hence the initial iterates are smoother than the iterations in later stages, and as a consequence the initial iterates are more regularized.

The LSQR algorithm with $\lambda = 0$ is equivalent to Lanczos bidiagonalization of $\tilde{\mathbf{K}}\,\mathbf{H}_2$. Specifically, after $k$ steps, the Lanczos bidiagonalization algorithm has produced three matrices $\bar{\mathbf{U}}^{(k)} \in \mathbb{R}^{m\times(k+1)}$, $\bar{\mathbf{B}}^{(k)} \in \mathbb{R}^{(k+1)\times k}$, and $\bar{\mathbf{V}}^{(k)} \in \mathbb{R}^{m\times k}$ such that

$$(\tilde{\mathbf{K}}\,\mathbf{H}_2)\,\bar{\mathbf{V}}^{(k)} = \bar{\mathbf{U}}^{(k)}\,\bar{\mathbf{B}}^{(k)}\,, \tag{24}$$

where both $\bar{\mathbf{U}}^{(k)}$ and $\bar{\mathbf{V}}^{(k)}$ have orthonormal columns, and $\bar{\mathbf{B}}^{(k)}$ is bidiagonal. Moreover, the first $k$ columns of $\bar{\mathbf{U}}^{(k)}$, the first $k-1$ columns of $\bar{\mathbf{V}}^{(k)}$, and the leading $k\times(k-1)$ sub-matrix of $\bar{\mathbf{B}}^{(k)}$ are identical to the corresponding quantities in the previous step of the algorithm. The $k$th iterate $\hat{\mathbf{q}}^{(k)}$ is then computed as

$$\hat{\mathbf{q}}^{(k)} = \bar{\mathbf{V}}^{(k)}\,\boldsymbol{\xi}^{(k)}\,,$$

where $\boldsymbol{\xi}^{(k)}$ solves the problem

$$\min \|\bar{\mathbf{B}}^{(k)}\,\boldsymbol{\xi} - (\bar{\mathbf{U}}^{(k)})^T(\tilde{\mathbf{t}}(x_0) - \alpha\tilde{\mathbf{K}}\mathbf{h}_1)\|_2\,.$$

The key to the efficiency of LSQR lies in the way that $\boldsymbol{\xi}^{(k-1)}$ is updated to $\boldsymbol{\xi}^{(k)}$, while $\bar{\mathbf{U}}^{(k)}$ and $\bar{\mathbf{V}}^{(k)}$ are never stored; we refer to Paige & Saunders (1982a) for more details.

We emphasize the difference between the two bidiagonalization procedures. The full bidiagonalization (19) is explicitly computed once, and it is independent of the regularization parameter; $\lambda$ enters when solving the augmented bidiagonal system in equation (22). The Lanczos bidiagonalization appears implicitly in computing the LSQR iterates $\hat{\mathbf{q}}^{(k)}$, and the sequence of iterates $\hat{\mathbf{q}}^{(1)}, \hat{\mathbf{q}}^{(2)}, \ldots$ corresponds to sweeping through a range of regularization parameters.

#### 2.2.4. Transformation back to the original setting

To compute the solution to the original problem in (15) it is necessary to transform the solution $\hat{\mathbf{q}}_\lambda = \bar{\mathbf{V}}\boldsymbol{\xi}_\lambda$ or $\hat{\mathbf{q}}^{(k)} = \bar{\mathbf{V}}^{(k)}\boldsymbol{\xi}^{(k)}$ to the original variable $\mathbf{q}(x_0)$. Below $\boldsymbol{\xi}$ and $\bar{\mathbf{V}}$ denote either $\boldsymbol{\xi}_\lambda$ and $\bar{\mathbf{V}}$ from Section 2.2.2 or $\boldsymbol{\xi}^{(k)}$ and $\bar{\mathbf{V}}^{(k)}$ from Section 2.2.3. Inserting equations (18) and (21) in equation (2) we get

$$\begin{aligned} f_{\rm est}(x_0) &= \mathbf{q}(x_0)^T\mathbf{b} \\ &= \begin{pmatrix}\alpha\\ \hat{\mathbf{q}}\end{pmatrix}^T \mathbf{H}^T\,\mathbf{b} = \begin{pmatrix}\alpha & \boldsymbol{\xi}^T\bar{\mathbf{V}}^T\end{pmatrix}\mathbf{H}^T\mathbf{b} \\ &= \alpha\mathbf{h}_1^T\mathbf{b} + \boldsymbol{\xi}^T\bar{\mathbf{V}}^T\mathbf{H}_2^T\mathbf{b}\,. \end{aligned} \tag{25}$$

Similarly, if we insert equations (18), (19) and (21) in equation (10) and use the definition of $\tilde{\mathbf{K}}$ then we get

$$\begin{aligned} \mathbf{k}(x_0) &= \mathbf{K}^T\mathbf{q}(x_0) \\ &= \mathbf{W}^{-1/2}\tilde{\mathbf{K}}\mathbf{H}\begin{pmatrix}\alpha\\ \bar{\mathbf{V}}\boldsymbol{\xi}\end{pmatrix} \\ &= \mathbf{W}^{-1/2}\left(\alpha\,\tilde{\mathbf{K}}\,\mathbf{h}_1 + \bar{\mathbf{U}}\,\bar{\mathbf{B}}\,\boldsymbol{\xi}\right)\,. \end{aligned} \tag{26}$$

Finally, due to the orthogonality of $\mathbf{H}$ and $\bar{\mathbf{V}}$ the variance of the error in $f_{\rm est}(x_0)$ due to the noise takes the form

$$\begin{aligned} \sigma^2(x_0) &= \|\mathbf{q}(x_0)\|_2^2 \\ &= \left(\mathbf{H}\begin{pmatrix}\alpha\\ \bar{\mathbf{V}}\boldsymbol{\xi}\end{pmatrix}\right)^T\mathbf{H}\begin{pmatrix}\alpha\\ \bar{\mathbf{V}}\boldsymbol{\xi}\end{pmatrix} \\ &= \alpha^2 + \|\boldsymbol{\xi}\|_2^2\,. \end{aligned} \tag{27}$$

Consider first full bidiagonalization from Section 2.2.2. We notice that $\boldsymbol{\xi}_\lambda$ is the only quantity in the above expressions that depends on the regularization parameter $\lambda$ and the target function $\tilde{\mathbf{t}}(x_0)$. Therefore $\alpha\mathbf{h}_1^T\mathbf{b}$ and $\bar{\mathbf{V}}^T\mathbf{H}_2^T\mathbf{b}$ only need to be computed once (in fact, they are already computed in earlier stages of the algorithm). This means that re-computing the solution $f_{\rm est}(x_0)$ when changing $\lambda$ or $\tilde{\mathbf{t}}(x_0)$ is very cheap. This is a great improvement compared with the Lagrange-multiplier method from Section 2.1, where a matrix factorization must re-computed each time.

Although we do not explicitly compute $\mathbf{q}(x_0)$, it is still quite inexpensive to compute the discretized averaging kernels $\mathbf{k}(x_0)$ for the different solutions. Again, one of the terms, namely $\tilde{\mathbf{K}}\,\mathbf{h}_1$, is already computed in earlier steps of the algorithm and the term $\bar{\mathbf{U}}\,\bar{\mathbf{B}}\,\boldsymbol{\xi}_\lambda$ requires only approximately $2n^2$ flops.

Consider next the case from Section 2.2.3 where we use Lanczos bidiagonalization. Again, only $\boldsymbol{\xi}^{(k)}$ depends on the target function $\tilde{\mathbf{t}}(x_0)$, but now both $\boldsymbol{\xi}^{(k)}$ and $\bar{\mathbf{V}}^{(k)}$ depend on the regularization parameter, i.e., the iteration number $k$. Moreover, they both appear only implicitly in the algorithm, while it is the iteration vector $\hat{\mathbf{q}}^{(k)}$ that is explicitly computed. It is now more efficient to use the formulation $f_{\rm est}(x_0) = \alpha\mathbf{h}_1^T\mathbf{b} + (\hat{\mathbf{q}}^{(k)})^T\mathbf{H}_2^T\mathbf{b}$ to compute the estimated solution, and the variance of the error in $f_{\rm est}(x_0)$ is given by $\sigma^2(x_0) = \alpha^2 + \|\hat{\mathbf{q}}^{(k)}\|_2^2$. Similarly, the discretized averaging kernel should be computed by means of $\mathbf{k}(x_0) = \mathbf{W}^{-1/2}(\alpha\,\tilde{\mathbf{K}}\,\mathbf{h}_1 + \tilde{\mathbf{K}}\,\mathbf{H}_2\,\hat{\mathbf{q}}^{(k)})$.

### *2.3. Summary of the new algorithms*

Our new algorithms, based on bidiagonalization, are summarized below. They describe the general case where $f_{\rm est}$ is computed at points $x_1, x_2, \ldots, x_q$. The matrix $\mathbf{T}$ below is again the matrix whose columns contain "samples" of the $q$ target function: $(\mathbf{T})_{ij} = (\mathbf{t}(x_j))_i = \mathcal{T}(x_j, x_i)$, $i = 1, \ldots, n$, $j = 1, \ldots, q$.

**Algorithm 2.2 SOLA by elimination of the constraint (full bidiagonalization)**

1. Transform to standard form
$\mathbf{E} \; = \; \mathbf{C}\,\mathbf{C}^T$
$\mathbf{K} \; \leftarrow \; \mathbf{C}^{-1}\,\mathbf{K}$
$\mathbf{b} \; \leftarrow \; \mathbf{C}^{-1}\,\mathbf{b}$
2. Set up LSE problem
$\tilde{\mathbf{K}} \; \leftarrow \; \mathbf{W}^{1/2}\,\mathbf{K}^T$
$\mathbf{c} \; \leftarrow \; \tilde{\mathbf{K}}\,\mathbf{W}\,\mathbf{e}_n$
$\tilde{\mathbf{T}} \; \leftarrow \; \mathbf{W}^{1/2}\,(\mathbf{t}(x_1), \mathbf{t}(x_2), \ldots, \mathbf{t}(x_q))$
3. Generate Householder transformation $\mathbf{H} = \mathbf{I} - \beta \mathbf{v}\mathbf{v}^T$ and apply it to $\tilde{\mathbf{K}}$ to get the reduced problem
$(\mathbf{a}_1, \mathbf{A}_2) \; = \; \tilde{\mathbf{K}} \; \leftarrow \; \tilde{\mathbf{K}}\,\mathbf{H}$
$\begin{pmatrix} \beta_1 \\ \boldsymbol{\beta}_2 \end{pmatrix} \; = \; \mathbf{b} \; \leftarrow \; \mathbf{H}^T\,\mathbf{b}$
4. Compute the bidiagonalization of $\mathbf{A}_2$ and change variables.
$\mathbf{A}_2 \; = \; \bar{\mathbf{U}}\,\bar{\mathbf{B}}\,\bar{\mathbf{V}}^T$
$\boldsymbol{\beta}_2 \; \leftarrow \; \bar{\mathbf{B}}^{-1}\bar{\mathbf{U}}^T\mathbf{A}_2\boldsymbol{\beta}_2 \; = \; \bar{\mathbf{V}}^T\boldsymbol{\beta}_2$
5. Modify the right-hand sides.
$$\tilde{\mathbf{T}} \leftarrow \bar{\mathbf{U}}^T \left(\tilde{\mathbf{T}} - \alpha \mathbf{a}_1 \mathbf{e}_q^T\right)$$
6. Apply Givens rotations to get lower bidiagonal and solve for $\boldsymbol{\Xi}$
$$\mathbf{G}_{2n-1} \ldots \mathbf{G}_2 \mathbf{G}_1 \left( \begin{array}{c|c} \bar{\mathbf{B}} & \tilde{\mathbf{T}} \\ \lambda \mathbf{I}_n & \mathbf{0} \end{array} \right) \; = \; \left( \begin{array}{c|c} \bar{\mathbf{B}}_\lambda & \tilde{\mathbf{T}}_\lambda \\ \mathbf{0} & * \end{array} \right)$$
$$\boldsymbol{\Xi} = (\boldsymbol{\xi}(x_1), \ldots, \boldsymbol{\xi}(x_q)) \; \leftarrow \; \bar{\mathbf{B}}_\lambda^{-1}\tilde{\mathbf{T}}_\lambda$$
7. Compute the estimates of $f$ and the corresponding variances at $x_1, x_2, \ldots, x_q$
$$(f_{\rm est}(x_1), f_{\rm est}(x_2), \ldots, f_{\rm est}(x_q))^T \leftarrow \alpha \beta_1 \mathbf{e}_q^T \, + \, \boldsymbol{\Xi}^T \boldsymbol{\beta}_2$$
$$\left(\sigma^2(x_1), \ldots, \sigma^2(x_q)\right)^T \leftarrow$$
$$\left(\alpha^2 + \|\boldsymbol{\xi}(x_1)\|_2^2, \ldots, \alpha^2 + \|\boldsymbol{\xi}(x_3)\|_2^2\right)^T \; .$$
8. If needed, compute the corresponding averaging kernels
$$(\mathbf{k}(x_1), \mathbf{k}(x_2), \ldots, \mathbf{k}(x_q))^T \leftarrow \mathbf{W}^{-1/2} \, \left(\alpha \mathbf{a}_1 \mathbf{e}_q^T \, + \bar{\mathbf{U}}\,\bar{\mathbf{B}}\,\boldsymbol{\Xi}\right) \; .$$

Here step 4 completely dominates the computation time. Changing the target functions requires re-computing the last part of step 2 plus steps 5 through 8. Changing $\lambda$ requires re-computing steps 6 through 8. In Section 4 we will give a detailed analysis of the number of operations used by the algorithm.

**Algorithm 2.3 SOLA by elimination of the constraint (Lanczos bidiagonalization)**

1.–3. These steps are similar to Algorithm 2.2, except that $\tilde{\mathbf{K}} = \mathbf{W}^{1/2}\,\mathbf{K}^T\,\mathbf{H}$ is kept in factored form.
4. For each of the $q$ columns of $\tilde{\mathbf{T}}$, apply $k_i$ steps of the LSQR algorithm to produce the solutions
$\hat{\mathbf{Q}} = \left(\hat{\mathbf{q}}^{(k_1)}(x_1), \hat{\mathbf{q}}^{(k_2)}(x_2), \ldots, \hat{\mathbf{q}}^{(k_q)}(x_q)\right) \; .$
5. Compute the estimates of $f$ and the corresponding variances at $x_1, x_2, \ldots, x_q$
$$(f_{\rm est}(x_1), f_{\rm est}(x_2), \ldots, f_{\rm est}(x_q))^T \leftarrow \alpha \beta_1 \mathbf{e}_q^T + \hat{\mathbf{Q}}^T \boldsymbol{\beta}_2$$
$$\left(\sigma^2(x_1), \ldots, \sigma^2(x_q)\right)^T \leftarrow$$
$$\left(\alpha^2 + \|\hat{\mathbf{q}}^{(k_1)}(x_1)\|_2^2, \ldots, \alpha^2 + \|\hat{\mathbf{q}}^{(k_q)}(x_q)\|_2^2\right)^T \; .$$
6. If needed, compute the corresponding averaging kernels
$(\mathbf{k}(x_1), \mathbf{k}(x_2), \ldots, \mathbf{k}(x_q))^T \leftarrow \mathbf{W}^{-1/2} \, \left(\alpha \mathbf{a}_1 \mathbf{e}_q^T + \tilde{\mathbf{K}}\,\mathbf{H}_2\,\hat{\mathbf{Q}}\right) \; .$

Changing the target functions requires re-computing steps 4 through 6. In this variant of the algorithm the $k$ acts as the regularization parameter; increasing $k$ requires additional steps of the LSQR algorithm to be computed in step 4. If $k$ is decreased no extra Lanczos iterates are needed, provided the iteration vectors $\hat{\mathbf{q}}^{(k)}$ are saved. In any case steps 5 through 6 need to be re-computed.

## 3. Available Software

Having presented the algorithm we will briefly discuss how they can be implemented in practice. Any specific implementation will depend on the architecture of the specific platform used. A common approach is to use the Basic Linear Algebra Subroutines (BLAS). The BLAS has become a de facto standard and FORTRAN BLAS routines are supplied by most computer vendors in highly optimized version for their specific machine(s). Most vendors also supply a C interface to the BLAS. Among the BLAS subroutines we find most of the necessary building blocks for the algorithms presented: Solution of triangular systems of equations, Givens rotations, Householder transformations (rank-1 update), etc.

The more complex "building blocks" such as the Cholesky factorization, the Bunch-Kaufmann $\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T$ factorization, the SVD, and the bidiagonalization are not part of the BLAS. These routines are available in most mathematical software libraries, such as LINPACK, Dongarra et al. (1979), LAPACK, Anderson et al. (1992), NAG (1991), and IMSL (1991), and some are also found in Numerical Recipes, Press et al. (1992). In Table 1 we have listed the relevant subroutines in the different libraries. Only LAPACK and NAG contain bidiagonalization routines, in the other libraries this operation is a part of the SVD routine.

In the sparse/structured case, the core of the computations lies in the LSQR algorithm which is available in FORTRAN; see Paige & Saunders (1982b).

## 4. The Computational Efficiency of the Algorithms

In the following we will consider the computational efficiency of the traditional and the new implementations of the SOLA method. We remark that for sparse and structured matrices, for which the iterative LSQR algorithm should be used, the flop counts depend on the sparsity

**Table 1.** Subroutines to be used as "building blocks" for the algorithms.

| Factorization | | LINPACK | LAPACK | NAG | IMSL | Num. Rec. |
|---|---|---|---|---|---|---|
| Cholesky | $\mathbf{C}\,\mathbf{C}^T$ | `_POFA` | `_POTRF` | `F07FDF` | `LFTDS` | `CHOLDC` |
| Cholesky, banded matrix | | — | `_PBTRF` | `F07HDF` | `LFCQS` | — |
| Bunch-Kaufman | $\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T$ | `_SIFA` | `_SYTRF` | `F07MDF` | `LFTSF` | — |
| SVD | $\mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T$ | `_SVDC` | `_GESVD` | `F02WEF` | `LSVRR` | `SVDCMP` |
| Bidiagonalization | $\bar{\mathbf{U}}\,\bar{\mathbf{B}}\bar{\mathbf{V}}^T$ | Part of `_SVDC` | `_GEBRD` | `F01QCF` + `F02SWF` | Part of `LSVRR` | Part of `SVDCMP` |

**Table 2.** Flop counts for transformation to unit covariance form; $p$ is the bandwidth.

| Covariance matrix | Transformation |
|---|---|
| Full | $m^3/3 + m^2 n + 2m^2$ |
| Banded | $m(p^2 + 2np + 7p)$ |
| Diagonal | $mn + 3m$ |

**Table 3.** Flop counts for Algorithm 2.1 with and without SVD preprocessing.

| Step | without SVD prep. | with SVD prep. |
|---|---|---|
| 2 | — | $2mn^2 + 2mn + 11n^3 + 3n^2$ |
| 3 | $m^3/3 + m^2(n+2) + mn$ | $4n^3/3 + 3n^2 + 2nq$ |
| 4 | $2mnq + nq$ | $2n^2 q + nq$ |
| 5 | $2m^2 q + mq$ | $2n^2 q + nq$ |
| 6 | $4mq$ | $4nq$ |
| 7 | $2mnq$ | $2n^2 q$ |
| total | $m^3/3 + m^2(n + 2q + 2) + m(4nq + n + 5q)$ | $2mn^2 + 2mn + 13n^3 + 6n^2(q+1) + 8nq$ |

**Table 4.** Flop counts for Algorithms 2.2–2.3. The quantity $\bar{k}$ is the average number of iterations, and $N$ is the number of flops in a matrix-vector multiplication.

| Step | Algorithm 2.2 | Algorithm 2.3 |
|---|---|---|
| 2 | $2mn$ | $2mn$ |
| 3 | $4mn$ | $4mn$ |
| 4 | $2mn^2 + 2mn + 2n^3$ | $\bar{k}(2N + 12m + 10n)q$ |
| 5 | $2n^2 q + 3nq$ | $2mq$ |
| 6 | $11nq + 13n$ | $2mnq$ |
| 7 | $4nq$ | — |
| 8 | $2n^2 q + 5nq$ | — |
| total | $2mn^2 + 8mn + 2n^3 + 4n^2 q + 23nq$ | $\bar{k}(2N + 12m + 10n)q + 2m(n+1)q + 6mn$ |

and structure of the matrix as well as the necessary number of iterations.

We calculate the number of floating point operations used in the algorithms as a function of the problem size. The "size" of the problem is characterized by three constants: The number $m$ of kernels, the number $n$ of points used in the discretization of the target functions and kernels, and the number $q$ of target functions, i.e., the number of points where we wish to determine $f_{\text{est}}$. The relative sizes of $m$, $n$ and $q$ will depend on the actual application area, but it is often the case that $m \gg n \approx q$.

### *4.1. Reduction to Unit Covariance Form*

Since the reduction to unit covariance form is the only part of the algorithms that depends on the covariance matrix $\mathbf{E}$, we have chosen to analyze this preprocessing step separately, and to analyze the algorithms under the assumption that we have a problem in unit covariance form, i.e., the associated covariance matrix is the identity matrix.

The actual amount of statistical information available may vary a great deal, depending on the application area. In fact, it is often very difficult to estimate the covariance matrix. In helioseismology, for instance, computing the $b_i$'s from the solar data is a complex process where the correlations are not well known. Some experiments have been performed on artificial data, and they seem to indicate that the covariance matrix has rather small off-diagonal elements, suggesting that the data may be described satisfactory by a diagonal or banded covariance matrix with little bandwidth; see Schou et al. (1995).

In Table 2 we list the computational cost of the preprocessing step for a full covariance matrix, a banded covariance matrix with bandwidth $p$, and a diagonal covariance matrix, respectively. For a full $m \times m$ covariance matrix the time spent computing the Cholesky factorization is excessively large. In the more usual case where the variance of the individual errors is known in advance, but no knowledge about correlations between the errors is available, the transformation consists in a simple scaling of the rows in the kernel matrix and a similar scaling of the data and will not contribute significantly to the overall computation time. We emphasize that this transformation is done only once, even if the problem is being solved for several values of $\lambda$ and/or target functions.

### *4.2. Comparison of the Algorithms*

In Tables 3 and 4 we summarize the detailed flop counts for Algorithms 2.1–2.3.

In connection with the iterative algorithm used in Step 4 of Algorithm 2.3, the quantity $N$ denotes the number of flops involved in a matrix-vector multiplication with $\mathbf{K}$, and $\bar{k}$ denotes the average number of iterations (since $k$ will typically depend on $x_0$). If $\mathbf{K}$ is sparse, then $N$ is twice the number of nonzeros in the matrix, and if $\mathbf{K}$ has Toeplitz form then $N$ is proportional to $m \log_2 m$. We shall not comment further on the efficiency of this algorithm, since the flop counts depend dramatically on $N$.

If we consider the case where one wants to calculate $f_{\mathrm{est}}$, $\sigma^2$ and the corresponding averaging kernels at $q$ specified points $x_1, x_2, \ldots, x_q$ and $q$ is equal to the number $n$ of discretization points, then the number of flops used by the three algorithms are as follows:

| | | |
|---|---|---|
| 2.1 | : | $m^3/3 + 3m^2 n$ |
| 2.1 (SVD) | : | $2m(n^2+n) + 19n^3$ |
| 2.2 | : | $2m(n^2+4n) + 6n^3$ . |

Obviously Algorithm 2.1 without the SVD preprocessing is prohibitively slow except for very small problems. The other two approaches are comparable, Algorithm 2.2 being faster if $m < 2.1\, n^2$. A question of special interest is the amount of work needed to re-compute the solution when $\lambda$ is changed:

| | | $f_{\mathrm{est}}$ and $\sigma^2$ | $f_{\mathrm{est}}$, $\sigma^2$ and $\mathbf{k}$ |
|---|---|---|---|
| 2.1 | : | $m^3/3 + m^2(n+2q)$ | $m^3/3 + m^2(n+2q)$ |
| 2.1 (SVD) | : | $4n^3/3 + 2n^2 q$ | $4n^3/3 + 4n^2 q$ |
| 2.2 | : | $15nq$ | $2n^2 q + 20nq$ |

Here Algorithm 2.2 is clearly superior. In most parameter-choice methods, only $f_{\mathrm{est}}$ and $\sigma^2$ have to be re-computed – a fact that makes Algorithm 2.2 even more attractive.

A crucial factor in mollifier methods is the shape of the chosen target function $\mathcal{T}(x_0, x)$. Therefore it is likely that solutions corresponding to a series of differently shaped target functions are computed. Below we list the number of flops used when $q$ target functions are changed:

| | | $f_{\mathrm{est}}$ and $\sigma^2$ | $f_{\mathrm{est}}$, $\sigma^2$ and $\mathbf{k}$ |
|---|---|---|---|
| 2.1 | : | $q(2m^2 + 2mn)$ | $q(2m^2 + 4mn)$ |
| 2.1 (SVD) | : | $q(4n^2 + 6n)$ | $q(6n^2 + 6n)$ |
| 2.2 | : | $q(2n^2 + 18n)$ | $q(4n^2 + 23n)$ |

Again, Algorithm 2.2 is the most efficient – although the difference is not as dramatic as above.

Pijpers & Thompson (1994) demonstrated that for helioseismic inversion it is possible to determine an automatic scaling of the width of the (Gaussian) target functions $\mathcal{T}(x_0, x)$. From physical arguments it is possible to derive an expression that relates the smallest length scale by which the rotation can be resolved at radius $x_0$ to certain physical quantities. They demonstrate that once the optimal target width has been computed at $x_0$, this relation can be used to scale the target width at any radius. Using this scaling the optimal value of $\lambda$ for each $x_0$ can be computed efficiently using Algorithm 2.2.

## 5. An Example From Helioseismology

To illustrate our bidiagonalization methods we have included a numerical example from *inverse helioseismology*. The problem consists in inferring the solar rotation rate as a function of radius from the rotational frequency-splittings of the eigen-oscillations observed on the solar surface. We use a subset of mode set 1 from Christensen-Dalsgaard et al. (1990), restricted to the frequency range 2.75–3.25 mHz. This reduced set contains 212 modes of degrees

$$l = 1, 2, \ldots, 29, 30, 32, \ldots, 78, 80, 85, \ldots, 195, 200\ ,$$

see Christensen-Dalsgaard et al. (1990) and references therein for further details on the rotational inversion problem. This example is identical to the 1-D test problem used by Hanke & Hansen (1993) and the dimensions of the kernel-matrix are $m = 212$ and $n = 100$. As target functions we use a Gaussian of width $\Delta$:

$$\mathcal{T}(x_0, x) = \frac{c}{\Delta} \exp\left[ -\left( \frac{x - x_0}{\Delta} \right)^2 \right]\ ,$$

where $c \approx 1/\sqrt{\pi}$ is a normalization factor and $x$ is the fraction of solar radius $r/R_\odot$. The errors in the right-hand side are Gaussian with zero mean, standard deviation $\sigma_0 = 10^{-3}$, and uncorrelated.

All our calculations are carried out in Matlab on an HP9000/819 workstation with machine precision $2.22 \cdot 10^{-16}$ and IEEE arithmetic. For simplicity we use the midpoint quadrature rule to calculate the weights in $\mathbf{W}$.

Figure 1 shows the target function and the computed averaging kernels $\mathbf{k}$ for $x_0 = 0.5$ and $x_0 = 0.9$, computed by means of both full bidiagonalization and Lanczos bidiagonalization. The averaging kernels computed by Algorithm 2.3 (the Lanczos-based algorithm) with $k = 35$ are almost identical to those computed by Algorithm 2.2 with $\lambda = 2.3 \cdot 10^{-3}$.

Figure 2 shows the "trade-off plots" of the Algorithms 2.2 and 2.3 at the two points $x_0 = 0.5$ and $x_0 = 0.9$. Define the error magnification as the ratio of the standard deviation of the error in $f_{\mathrm{est}}(x_0)$ due to the noise, $(\mathbf{q}(x_0)^T \mathbf{E} \mathbf{q}(x_0))^{1/2}$, to the standard deviation $\sigma_0$ of the errors. We plot the error magnification versus the residual norm $\int_0^1 [\mathcal{K}(x_0, x) - \mathcal{T}(x_0, x)]^2\, dx$ for varying values of the regularization parameters $\lambda$ and $k$. The solid line shows the trade-off curve for Algorithm 2.2 using full bidiagonalization with $\lambda$ ranging from $4.5 \cdot 10^{-4}$ (top left) to 1.4 (bottom right). The "optimal" value of $\lambda$, corresponding to the corner of the trade-off curve, is approximately $2.3 \cdot 10^{-3}$. The dots represent the discrete trade-off curve associated with Algorithm 2.3, using Lanczos bidiagonalization. The regularizing effect of the Lanczos-based algorithm is clearly seen: As the iterates are generated the algorithm effectively sweeps through a range of regularization parameters, reaching the optimal solution after approximately 35 iterations.

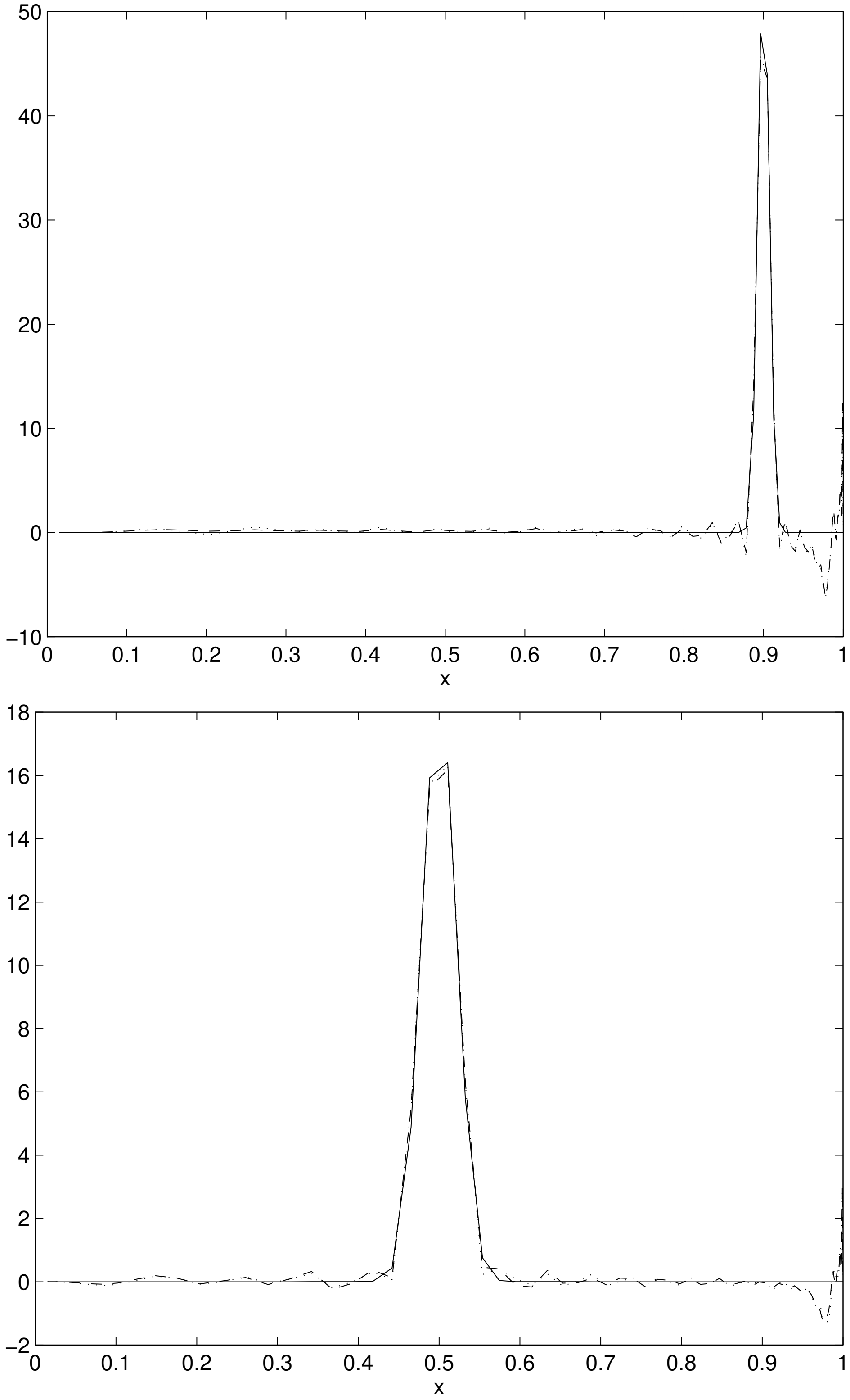


**Fig. 1. Top:** Computed averaging kernels $\mathcal{K}(x_0, x)$ for the estimate at $x_0 = 0.9$. The target width $\Delta$ is $10^{-2}$. The dashed and the dotted lines are the averaging kernels for the almost identical solutions computed by means of full bidiagonalization with $\lambda = 2.3 \cdot 10^{-3}$ and Lanczos bidiagonalization with $k = 35$, respectively. The solid line is the target function $\mathcal{T}(x_0, x)$. **Bottom:** The same plot for $x_0 = 0.5$, $\Delta = 3 \cdot 10^{-2}$, $\lambda = 2.3 \cdot 10^{-3}$ and $k = 35$.

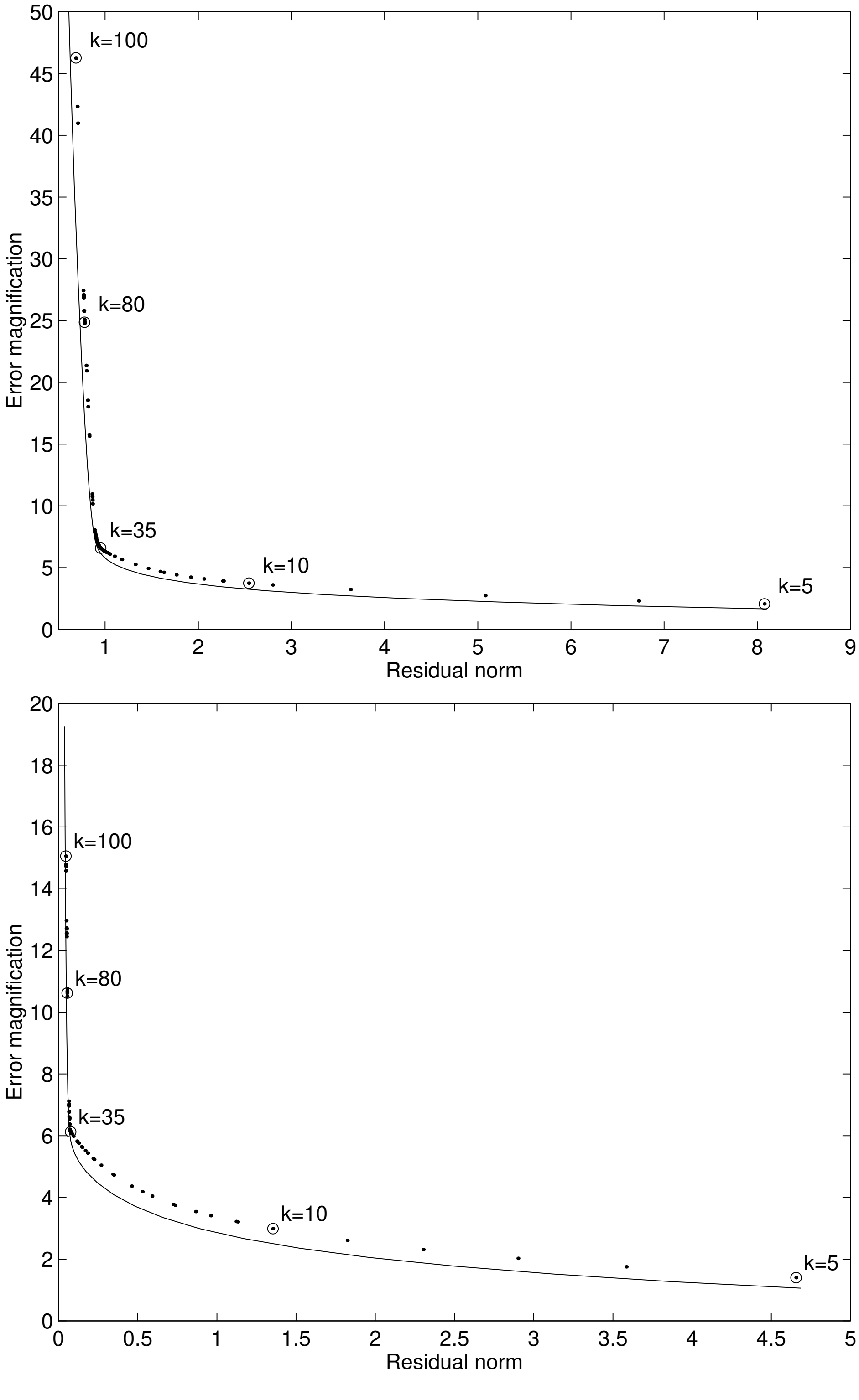


**Fig. 2.** **Top:** Trade-off curves of the error magnification $(\mathbf{q}(x_0)^T\mathbf{E}\mathbf{q}(x_0))^{1/2}/\sigma_0$ versus the residual norm $\int_0^1[\mathcal{K}(x_0,x)-\mathcal{T}(x_0,x)]^2\,dx$, at $x_0 = 0.9$ with $\Delta = 10^{-2}$. The solid line is the trade-off curve for the solutions computed by means of the full bidiagonalization procedure values of $\lambda$ in the interval from $4.5\cdot 10^{-4}$ to 1.4. The dots represent the discrete trade-off curve of the Lanczos-based method. The solutions corresponding to 5, 10, 35, 80 and 100 iterations are marked on the plot. **Bottom:** The same plot for $x_0 = 0.5$ and $\Delta = 3\cdot 10^{-2}$.

## 6. Conclusion

The SOLA mollifier method is an inversion method which, in terms of both computational efficiency and resolution ability, is competitive to other classes of inversion methods. In its original implementation, however, it is inefficient when many regularization parameters must be used, e.g., in connection with parameter-choice methods such as generalized cross-validation. We have demonstrated how this problem can be overcome using standard "building blocks" from numerical linear algebra. The improvement for dense matrices is quite dramatic when using the full bidiagonalization approach in Algorithm 2.2. For sparse or structured matrices it is convenient to use an iterative Lanczos-based method, Algorithm 2.3, in which the amount of regularization is controlled by the number of iterations.

Another advantage of using the standard numerical linear algebra "building blocks" is that these subroutines are usually available in highly vectorized and parallelized versions on today's supercomputers, thus ensuring good performance without the need for a tedious fine-tuning of the software; see, e.g., Dongarra, Duff, Sorensen & van der Vorst (1991).

As illustrated in our numerical example, Algorithm 2.3 computes solutions in relatively few iterations which are very close to the optimal solutions obtained from Algorithm 2.2. It is quite typical when solving large ill-posed problems, e.g., in inverse helioseismology, that the number of iterations needed in a Lanczos-based method is far less than the dimension of the problem. This makes these iterative methods an interesting alternative to the direct methods such as Algorithms 2.1 and 2.2.

*Acknowledgements.* We thank Jørgen Christensen-Dalsgaard for fruitful discussions during this work.

## References

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchow, O., Sorensen, D., 1992, *LAPACK Users' Guide*, SIAM, Philadelphia.

Backus, G. E., Gilbert, J. F., 1968, Geophys. J. R. Astron. Soc. **16** , 169.

Christensen-Dalsgaard, J., Schou, J., Thompson, M. J., 1990, Mon. Not. R. Astr. Soc. **242**, 353.

Christensen-Dalsgaard, J., Thompson, M. J., 1993, A&A, **272**, L1.

Craig, I. J. D., Brown, J. C., 1986, *Inverse Problems in Astronomy*, Adam Hilger, Bristol.

Dongarra, J., Bunch, J. R., Moler, C. B., Stewart, G. W., 1979, *Linpack Users' Guide*, SIAM, Philadelphia.

Dongarra, J., Duff, I. S., Sorensen, D. C., van der Vorst, H. A., 1991, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia.

Eldén, L., 1977, BIT **17** , 134.

Golub, G. H., Van Loan, C. F., 1989, *Matrix Computations*, 2. Ed., Johns Hopkins University Press, Baltimore.

Hanke, M., Hansen, P. C., 1993, Surv. Math. Ind. **3**, 253.

Hansen, P. C., 1994, Inverse Problems **10**, 895.

*IMSL Math/Library*, Version 2.0, 1991.

Lawson, C. L., Hanson, R. J., 1974, *Solving Least Squares Problems*, Prentice-Hall.

Louis, A. K., Maass, P., 1990, Inverse Problems **6** , 427.

*NAG Fortran Library Manual, Mark 15*, 1991, NAG Ltd., Oxford.

Paige, C. C., Saunders, M. A., 1982, ACM Trans. Math. Software **8** (1982), 43.

Paige, C. C., Saunders, M. A., 1982, ACM Trans. Math. Software **8**, 195.

Parker, R. L., 1977, Ann. Rev. Earth Planet. Sci. **5**, 35.

Pijpers, F. P., Thompson, M. J., 1992, A&A, **262**, L33.

Pijpers, F. P., Thompson, M. J., 1994, A&A, **281** , 231.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., 1992, *Numerical Recipes*, 2. Ed., Cambridge University Press.

Schou, J., Christensen-Dalsgaard, J., Thompson, M. J., 1995, In *Proc. GONG'94: Helio- and Astro-seismology from Earth and Space*, eds Ulrich, R. K., Rhodes Jr, E. J., Däppen, W., Astron. Soc. of the Pacific Conference Series, San Francisco, **76** , 528.

This article was processed by the author using Springer-Verlag LaTeX A&A style file *L-AA* version 3.