Iterative algorithms for two-dimensional helioseismic inversion

Rasmus Munk Larsen, Department of Computer Science, University of Aarhus, Ny Munkegade, building 540, DK-8000 Aarhus C, Denmark.

Abstract

In two-dimensional helioseismic inversion the observed frequencies of the Suns normal modes are used to make inferences about the solar rotation rate as a function of radius and latitude. The inversion methods currently used are either Subtractive Optimally Localized Averages (SOLA) and Regularized Least Squares (RLS), where the main computational burden lies in solving a regularized linear least-squares problem. Standard algorithms are usually based on some orthogonal matrix-factorization, e.g. a singular value decomposition (SVD), but for the two-dimensional helioseismic inversion, where the dimensions of the coefficient matrix in the discretized problem are potentially $10^6 \times 10^4$, this approach is prohibitively expensive. To develop more efficient algorithms we have to incorporate some special knowledge about the underlying physical problem.

In this paper we show how insight into the properties of the continuous problem can be used to dramatically reduce the computational complexity and the amount of storage needed. It is shown that the coefficient matrix has a *block Kronecker product* structure, and we describe two iterative algorithms based on this observation that implement the SOLA inversion method. For large-scale helioseismic datasets the Kronecker product formulation typically reduces the computing time by two orders of magnitude. Finally we discuss the results and performance obtained with the new algorithms on data from the LOWL, GONG and MDI projects and compare it to that obtained using the algorithm described by Larsen and Hansen (1997).

1 Introduction

The Sun oscillates in thousands or even millions of normal modes, excited by the convection in the Sun's outer parts. The behavior of each mode with colatitude θ, longitude ϕ and radius r is characterized by the degree l, azimuthal order m and the radial order n. Measurements of the frequencies ω_{nlm} of these oscillations lead to inferences about the structure and dynamics of the solar interior. In particular, the variation of the frequencies with m can be used to determine the solar rotation $\Omega(r,\theta)$ as a function of colatitude θ and radius r. This is possible because the measured frequencies and the rotation are related via a Fredholm integral equation of the first kind in the following way:

$$d_i = \int_0^\pi \int_0^R K_i(r,\theta)\, \Omega(r,\theta)\, r \mathrm{d}r \mathrm{d}\theta + \epsilon_i \,, \qquad i = 1,\ldots,M \tag{1}$$

where ϵ_i represent measurement errors and a single index i has been used to enumerate all M combinations of (n,l,m). The kernels K_i depend only on the solar structure, and hence are assumed to be known from solar model calculations. Helioseismic data d_i are provided as either individual *frequency splittings*

$$\frac{\Delta\omega_{nlm}}{m} \equiv \frac{\omega_{nlm} - \omega_{nl0}}{m}$$

or as a set of *splitting coefficients* $a_j(n,l)$ obtained from a least-squares fit of an expansion on the form

$$\frac{\Delta\omega_{nlm}}{2\pi} = \sum_{j=1}^{j_{max}} a_j(n,l)\mathcal{P}_j^{(l)}(m) \ ,$$

where $\mathcal{P}_j^{(l)}(m)$ is a polynomial of degree j, to the frequencies for modes belonging to a given (n,l)-combination (a so-called *multiplet*); see Schou, Christensen-Dalsgaard and Thompson 1994 for further details (In the following we will not distinguish between individual splittings and a-coefficient, and 'm' will represent either order m or a-coefficient index $(j+1)/2$.).

With current observational techniques, such as the MDI instrument on board the SOHO satellite, the amount of available data has become very large; it is now possible to observe modes with degrees l ranging from 1 up to 300, and the total number of modes with $1 \leq m \leq l$ is potentially of the order 10^6. In practice, however, individual frequency splittings $\Delta\omega_{nlm}$ cannot (yet!) be determined reliably, and instead data is represented as coefficients $a_j(n,l)$ using the expansion above with $j = 1, 3, 5, \ldots, \min(35, l)$. With this representation, the current MDI datasets comprises 30648 coefficients.

The computational effort required to solve the inverse problem in Equation (1) (in the following referred to as 2dHELIO) is very large, and as the amount of data continues to increase, it becomes necessary to find more efficient algorithms.

2 Linear inversion methods

In this paper we consider linear inversion methods where the computed solution $\bar{\Omega}$, say, at a given radius r_0 and latitude θ_0 can be related to the original data d_i by *inversion coefficients* c_i:

$$\bar{\Omega}(r_0,\theta_0) = \sum_{i=1}^{M} c_i(r_0,\theta_0) d_i \ .$$

Combining this with Equation (1) we get

$$\begin{aligned} \bar{\Omega}(r_0,\theta_0) &= \int_0^\pi \int_0^R \sum_{i=1}^{M} c_i(r_0,\theta_0)\, K_i(r,\theta)\, \Omega(r,\theta)\, r \mathrm{d}r \mathrm{d}\theta \\ &\equiv \int_0^\pi \int_0^R \mathcal{K}(r_0,\theta_0;r,\theta)\, \Omega(r,\theta)\, r \mathrm{d}r \mathrm{d}\theta \ . \end{aligned} \tag{2}$$

The *averaging kernels* $\mathcal{K}(r_0,\theta_0;r,\theta)$ show how the solution $\bar{\Omega}(r_0,\theta_0)$ is a weighted average of the true solution, and provides a measure of the resolution in the inversion. The linear inversion methods fall in two categories

1. **Optimally Localized Averages** (OLA) methods (Backus & Gilbert (1968)) where inversion coefficients $c_i(r_0, \theta_0)$ are explicitly chosen to make $\mathcal{K}$ peaked around (r_0, θ_0). In the so-called Subtractive Optimally Localized Averages (SOLA) variant (see Pijpers and Thompson 1992), which is commonly used in helioseismology, the c_i's are chosen by minimizing the functional

$$\int_0^\pi \int_0^R \left[\mathcal{K}(r_0, \theta_0; r, \theta) - \mathcal{T}(r_0, \theta_0; r, \theta)\right]^2 r \, \mathrm{d}r \, \mathrm{d}\theta \; + \; \lambda^2 \sum_{i=1}^{M} \left[\sigma_i \, c_i(r_0, \theta_0)\right]^2 \qquad (3)$$

subject to the constraint

$$\int_0^\pi \int_0^R \mathcal{K}(r_0, \theta_0; r, \theta) \, r \, \mathrm{d}r \, \mathrm{d}\theta = 1 \; . \qquad (4)$$

The *target function* $\mathcal{T}(r_0, \theta_0; r, \theta)$ is used to specify the desired form of the averaging kernel. The last term in (3) is a measure of the standard deviation of the solution $\bar{\Omega}(r_0, \theta_0)$, and hence the *regularization parameter* λ is used to trade off between the influence of measurement errors on the solution and the localization of the averaging kernel. In the following we will refer to this as the 2dSOLA method.

2. **Regularized Least Squares** (RLS) methods (Tikhonov (1963)) where the explicit goal is to obtain a solution that fits the data rather than one that corresponds to a well localized average. This is done by minimizing the expression

$$\sum_{i=1}^{M} \sigma_i^{-2} \left[\int_0^\pi \int_0^R K_i \, \bar{\Omega} \, r \, \mathrm{d}r \, \mathrm{d}\theta - d_i\right]^2 \; + \; \lambda^2 \int_0^\pi \int_0^R \left[\mathrm{D}^2 \, \bar{\Omega}\right]^2 r \mathrm{d}r \mathrm{d}\theta \; , \qquad (5)$$

where the second term is included to reduce the norm of the second derivative of the solution and hence force it to be smooth. The regularization parameter λ is used to control the amount of smoothing applied to the solution. In the following we will refer to this as the 2dRLS method.

We will now outline the algorithms used to implement the 2dSOLA method; for further details see Larsen & Hansen 1997.

To minimize the expression in (3) numerically, the solution $\bar{\Omega}$ and the kernels K_i are discretized on a regular $N_r \times N_\theta$ mesh in the region $\left\{ (r, \theta) \in [0\colon R] \times [0\colon \frac{\pi}{2}] \right\}$, see Schou, Christensen-Dalsgaard and Thompson (1994). In the following $N = N_r N_\theta$ denotes the total number of meshpoints, and we assume that $N < M$. The integrals involved are computed using a quadrature rule with weights w_j defined on the meshpoints (r_j, θ_j), $j = 1, \ldots, N$, i.e.:

$$\int_0^{\pi/2} \int_0^R f(r, \theta) \, r \, \mathrm{d}r \, \mathrm{d}\theta \approx \sum_{j=1}^{N} w_j \, f(r_j, \theta_j) \; .$$

We introduce the (dense) matrix $\mathbf{K} \in \mathbb{R}^{M \times N}$ defined as

$$(\mathbf{K})_{ij} = w_j^{1/2} \sigma_i^{-1} K_i(r_j, \theta_j) \; , \qquad i = 1, \ldots, M \; , \quad j = 1, \ldots, N$$

and the N-vector $\mathbf{t}$ with elements $(\mathbf{t})_j = w_j^{1/2}\,\mathcal{T}(r_0,\theta_0;r_j,\theta_j),\ j = 1,\ldots,N$ and the M-vector $\mathbf{c}$ with elements $(\mathbf{c})_i = c_i(r_0,\theta_0),\ i = 1,\ldots,M$.

After eliminating the constraint in Equation (4), the discretized version of (3) can be written as the linear least-squares problem

$$\min \left\| \begin{pmatrix} \mathbf{K}^T \\ \lambda \mathbf{I}_M \end{pmatrix} \mathbf{c} - \begin{pmatrix} \mathbf{t} \\ \mathbf{0} \end{pmatrix} \right\|_2 , \tag{6}$$

(The overhead involved in dealing with the constraint in is negligible compared to the overall cost and therefore we will ignore this step.).

When $\mathbf{K}$ is not sparse or structured and the optimal value of the regularization parameter λ is not known in advance, the most efficient algorithm for solving (6) is that described by Eldén 1977. This procedure is based on a Householder bidiagonalization

$$\mathbf{K}^T = \mathbf{U}\,(\,\mathbf{B}\quad \mathbf{0}\,)\,\mathbf{V}^T , \tag{7}$$

where $\mathbf{U} \in \mathbb{R}^{N\times N}$, $\mathbf{V} \in \mathbb{R}^{M\times M}$, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{B}$ is an $N\times N$ lower bidiagonal matrix (see, e.g., Golub & Van Loan (1989)). The solution to (6) is found as $\mathbf{c} = \mathbf{V}(:,1\!:\!N)\,\mathbf{y}$ where $\mathbf{y}$ is the solution to the banded problem

$$\min \left\| \begin{pmatrix} \mathbf{B} \\ \lambda \mathbf{I}_N \end{pmatrix} \mathbf{y} - \begin{pmatrix} \mathbf{U}^T\mathbf{t} \\ \mathbf{0} \end{pmatrix} \right\|_2 . \tag{8}$$

In the following we will refer to this algorithm as HBD.

The advantage of algorithm HBD is that the bidiagonalization is only computed once and subsequently stored on disk. Once this is done, the solution to Equation (8) can be computed cheaply using only $15N$ floating point operations (a few milliseconds in practice). This computation, which also produces the solution- and residual norms which describe the standard deviation in the solution at (r_0,θ_0) and the departure of the averaging kernel from the target function respectively, makes it feasible to the compute a large number of solutions corresponding to different values of λ in order to find the optimal one, e.g., using the L-curve criterion (Hansen & O'Leary (1993)). This algorithm also facilitates experimenting with different target functions, since the solution for P new target locations, say, can be computed in $P(2N^2 + 18N)$ operations (typically about a minute in practice) once the bidiagonalization is known.

The disadvantage of this method is the $2MN^2$ floating point operations required to compute the bidiagonalization in Equation (7). Although this represents a major improvement over the $M^3/3$ operation needed by the implementation in the original SOLA paper (Pijpers & Thompson 1992), the method still requires tens of hours of cpu-time on a large supercomputer. The most serious restriction of the method is the need to store the full $M \times N$ kernel matrix. This is the limiting factor in practice, where $\mathbf{K}$ is approximately 31000×5200 and hence requires 1.3Gb of memory on the computer. With future datasets potentially comprising some 10^6 frequency splittings, the need for more efficient algorithms is evident.

3 Exploiting kernel structure

The main problem in algorithm HBD described in the previous section is that it builds on standard algorithms from numerical linear algebra, which do not take advantage of any possible structure in the coefficient matrix. To develop more efficient algorithms we have to incorporate special knowledge about the underlying physical problem. In the following we will demonstrate that the kernel matrix $\mathbf{K}$ arising in 2dSOLA and 2dRLS has a special structure, and show how this can be exploited. When combined with the use of iterative regularization methods based on the Lanczos algorithm (see O'Leary & Simmons 1981, Paige & Saunders 1982a, Björck 1988), this observation leads to very efficient implementations of both inversion methods. In our presentation we concentrate on the 2dSOLA methods, but it should be emphasized that the iterative algorithms developed here apply equally well to the 2dRLS method, if the problem is first transformed into normal form by the methods described by Hanke & Hansen (1993).

We note that both Sekii (1993) and Pijpers & Thompson (1996) describe other attempts to exploit the almost separable nature of the kernels. In fact, both methods solve the problem by approximating the kernel with an exactly separable one and therefore do not solve problem 2dHELIO exactly. Both methods can be analyzed in detail using the results developed here, but this is beyond the scope of this paper.

The key to obtaining a more efficient algorithm is the observation that the kernels $K_i(r,\theta)$ in Equation (1) can be written on the form

$$K_i = F_{nl}(r)\,G_{lm}(\theta) \;+\; H_{nl}(r)\,J_{lm}(\theta)\ , \tag{9}$$

see Schou, Christensen-Dalsgaard and Thompson (1994). When we discretize (3), this functional form of K_i gives rise to a kernel matrix with block *Kronecker product* structure, see Appendix A for a formal definition of the Kronecker product. The resulting formulation has two important properties:

- It is possible to represent $\mathbf{K}$ in terms of discretizations of F, G, H and J, which significantly reduces the amount of storage required by the algorithm.
- The structure of $\mathbf{K}$ greatly reduces the cost of multiplying $\mathbf{K}$ or $\mathbf{K}^T$ by a vector. This can be effeciently exploited by solving Equation (6) using an iterative linear least-squares solver, which accesses the coefficient matrix only through matrix-vector multiplications with $\mathbf{K}$ and $\mathbf{K}^T$.

Below we state the results suggested in the discussion above in a more rigorous form. For this purpose the following definitions will prove useful for characterizing the modeset and hence the structure of $\mathbf{K}$:

Definition 3.1

$$\begin{aligned} L &= \text{number of different degrees } l \text{ of the modes in the data set,}\\ m_n^l &= \text{number of different } (n,l)\text{-combinations with degree } l, \end{aligned}$$

$$
\begin{aligned}
M_{nl} &= \sum_{l=1}^{L} m_n^l = \text{ total number of } (n,l)\text{-combinations (multiplets) in the data set,} \\
m_m^l &= \text{number of different } (l,m)\text{-combinations with degree } l, \\
M_{lm} &= \sum_{l=1}^{L} m_m^l = \text{ total number of } (l,m)\text{-combinations in the data set,} \\
M &= \sum_{l=1}^{L} m_n^l m_m^l = \text{ total number of modes in the data set.}
\end{aligned}
$$

The last equation suggests that we have made a rather crude assumption about the dataset: To simplify the notation we assume that if we divide the modeset into l-blocks of (n,l)-combinations (multiplets) with the same value of l', then all (n,l)-combinations belonging to the same l-block are represented in the dataset with same number of different (n,l,m)-combinations (splittings or a-coefficients). In practice, however, this is typically not the case, because certain modes are missing, e.g., because they could not be determined reliably. In this case each l-block is subdivided into smaller blocks, ultimately into a block holding only the modes of a single multiplet, until the assumption holds. This means that the number of blocks L is typically larger than the number of different degrees l. In theory the advantage of using the Kronecker decreases as the number of blocks increase but for all the datasets considered so far, this effect is negligible.
The block Kronecker structure of the kernel matrix is defined in the following lemma:

Lemma 3.1 *The matrix $\mathbf{K} \in \mathbb{R}^{M \times N}$ containing the discretized kernels for the 2dHELIO problem can be written as*

$$
\mathbf{K} = \begin{pmatrix} \mathbf{F}_1 \otimes \mathbf{G}_1 \\ \mathbf{F}_2 \otimes \mathbf{G}_2 \\ \vdots \\ \mathbf{F}_L \otimes \mathbf{G}_L \end{pmatrix} + \begin{pmatrix} \mathbf{H}_1 \otimes \mathbf{J}_1 \\ \mathbf{H}_2 \otimes \mathbf{J}_2 \\ \vdots \\ \mathbf{H}_L \otimes \mathbf{J}_L \end{pmatrix} , \tag{10}
$$

where '$\otimes$' denotes the Kronecker product.

Proof. Without loss of generality we shall ignore the second term in Equation (10) since the structure of the two terms is identical. We define the $M_{nl} \times N_r$ matrix $\mathbf{F}$ and the $M_{lm} \times N_\theta$ matrix $\mathbf{G}$,

$$
\begin{aligned}
(\mathbf{F})_{ij} &= F_i(r_j) \ , \quad i = 1,\ldots,M_{nl} \quad j = 1,\ldots,N_r \\
(\mathbf{G})_{ij} &= G_i(\theta_j) \ , \quad i = 1,\ldots,M_{lm} \quad j = 1,\ldots,N_\theta \ .
\end{aligned}
$$

Let $\mathbf{F}$ and $\mathbf{G}$ be partitioned such that

$$
\mathbf{F} = \begin{pmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_L \end{pmatrix} , \qquad \mathbf{G} = \begin{pmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_L \end{pmatrix} , \tag{11}
$$

where the block matrices $\mathbf{F}_l \in \mathbb{R}^{m_n^l \times N_r}$, $\mathbf{G}_l \in \mathbb{R}^{m_m^l \times N_\theta}$, $\quad l = 1, \ldots, L$ contain discretizations of the respective functions corresponding to modes of degree l. It now follows immediately from Equation (9) and the definition of the Kronecker product that we can choose a permutation of the rows of $\mathbf{K}$, such that it can be written in the form of Equation (10). $\square$

Example: If the given data corresponds to the modeset

$$\{(n,l,m) \mid (n,l) \in \{(1,2),(2,2),(1,3),(2,3)\},\ 1 \le m \le l\},$$

we get the following discretization

$$\mathbf{K} = \begin{pmatrix} \mathbf{F}_1 \otimes \mathbf{G}_1 \\ \mathbf{F}_2 \otimes \mathbf{G}_2 \end{pmatrix} + \begin{pmatrix} \mathbf{H}_1 \otimes \mathbf{J}_1 \\ \mathbf{H}_2 \otimes \mathbf{J}_2 \end{pmatrix}$$

with blocks defined by

$$\mathbf{F}_1 = \begin{pmatrix} F_{1\,2}(r_1) & F_{1\,2}(r_2) & \cdots & F_{1\,2}(r_{N_r}) \\ F_{2\,2}(r_1) & F_{2\,2}(r_2) & \cdots & F_{2\,2}(r_{N_r}) \end{pmatrix}, \qquad \mathbf{F}_2 = \begin{pmatrix} F_{1\,3}(r_1) & F_{1\,2}(r_2) & \cdots & F_{1\,3}(r_{N_r}) \\ F_{2\,3}(r_1) & F_{2\,3}(r_2) & \cdots & F_{2\,3}(r_{N_r}) \end{pmatrix},$$

$$\mathbf{G}_1 = \begin{pmatrix} G_{2\,1}(\theta_1) & G_{2\,1}(\theta_2) & \cdots & G_{2\,1}(\theta_{N_\theta}) \\ G_{2\,2}(\theta_1) & G_{2\,2}(\theta_2) & \cdots & G_{2\,2}(\theta_{N_\theta}) \end{pmatrix}, \quad \mathbf{G}_2 = \begin{pmatrix} G_{3\,1}(\theta_1) & G_{3\,1}(\theta_2) & \cdots & G_{3\,1}(\theta_{N_\theta}) \\ G_{3\,2}(\theta_1) & G_{3\,2}(\theta_2) & \cdots & G_{3\,2}(\theta_{N_\theta}) \\ G_{3\,3}(\theta_1) & G_{3\,3}(\theta_2) & \cdots & G_{3\,3}(\theta_{N_\theta}) \end{pmatrix},$$

and $N = N_\theta N_r$, $M_{nl} = 4$, $M_{lm} = 6$, $M = 10$, $m_n^2 = 2$, $m_m^2 = 2$, $m_n^3 = 2$, $m_m^3 = 3$.

The derivation of the fast matrix-vector multiplication algorithm provides the proof for the following theorem (see Appendix A for a definition of $\mathrm{vec}(\cdot)$):

Theorem 3.1 *Let $\mathbf{K} \in \mathbb{R}^{M \times N}$ be on the form given in Lemma 3.1, and let $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{d} \in \mathbb{R}^M$ then the matrix-vector multiplications $\mathbf{Kx}$ and $\mathbf{K}^T\mathbf{d}$ can be computed using $4N_\theta(M + N_r M_{lm})$ floating point operations and $2(M_{nl}N_r + M_{lm}N_\theta + M_{lm}N_r)$ words of storage or by using $4N_r(M + N_\theta M_{nl})$ floating point operations and $2(M_{nl}N_r + M_{lm}N_\theta + M_{nl}N_\theta)$ words of storage.*

Proof. Without loss of generality we shall ignore the second term in Equation (10) since the structure of the two terms is identical. Let $\mathbf{X} \in \mathbb{R}^{N_\theta \times N_r}$ and $\mathbf{D}_l \in \mathbb{R}^{m_m^l \times m_n^l}$, $l = 1, \ldots, L$ be defined such that

$$\mathrm{vec}\,(\mathbf{X}) = \mathbf{x}\ , \qquad \mathrm{vec}\,(\,[\,\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_L\,]\,) = \mathbf{d}\ .$$

First consider the case $\mathbf{K}\,\mathbf{x}$: By combining the results from Lemma 3.1 and Lemma A.2 we obtain:

$$\mathbf{K}\,\mathbf{x} = \begin{pmatrix} (\mathbf{F}_1 \otimes \mathbf{G}_1)\,\mathbf{x} \\ (\mathbf{F}_2 \otimes \mathbf{G}_2)\,\mathbf{x} \\ \vdots \\ (\mathbf{F}_L \otimes \mathbf{G}_L)\,\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathrm{vec}(\,\mathbf{G}_1\,\mathbf{X}\,\mathbf{F}_1^T\,) \\ \mathrm{vec}(\,\mathbf{G}_2\,\mathbf{X}\,\mathbf{F}_2^T\,) \\ \vdots \\ \mathrm{vec}(\,\mathbf{G}_L\,\mathbf{X}\,\mathbf{F}_L^T\,) \end{pmatrix}$$

We are free to choose the order in which the matrices in each block are multiplied together. If $(\mathbf{G}_l\,\mathbf{X}\,\mathbf{F}_l^T),\ l=1,\dots,L$ is computed as $((\mathbf{G}_l\,\mathbf{X})\,\mathbf{F}_l^T)$ we get

$$\mathbf{K}\,\mathbf{x} = \begin{pmatrix} \mathrm{vec}(\,\mathbf{Z}_1\,\mathbf{F}_1^T\,) \\ \mathrm{vec}(\,\mathbf{Z}_2\,\mathbf{F}_2^T\,) \\ \vdots \\ \mathrm{vec}(\,\mathbf{Z}_L\,\mathbf{F}_L^T\,) \end{pmatrix}, \qquad \mathbf{Z} = \begin{pmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \\ \vdots \\ \mathbf{Z}_L \end{pmatrix} = \mathbf{G}\,\mathbf{X} \in \mathbb{R}^{M_{lm}\times N_r}\,, \tag{12}$$

where $\mathbf{G}$ is defined in Equation (11). Assuming that the number of floating point operations required to multiply an $m\times k$ by an $k\times n$ matrix is $2mnk$, the total cost of computing $\mathbf{K}\,\mathbf{x}$ as in Equation (12) is

$$2N_rN_\theta M_{lm} + \sum_{l=1}^{L} 2m_n^l m_m^l N_r = 2N_r(N_\theta M_{lm} + M)\ .$$

To store $\mathbf{F}$, $\mathbf{G}$ and the temporary results in $\mathbf{Z}$, a total of $M_{nl}N_r + M_{lm}N_\theta + M_{lm}N_r$ words are required.

Conversely, if $(\mathbf{G}_l\,\mathbf{X}\,\mathbf{F}_l^T),\ l=1,\dots,L$ is computed as $(\mathbf{G}_l\,(\mathbf{X}\,\mathbf{F}_l^T))$ the resulting fomulas are

$$\mathbf{K}\,\mathbf{x} = \begin{pmatrix} \mathrm{vec}(\,\mathbf{G}_1\,\mathbf{Z}_1\,) \\ \mathrm{vec}(\,\mathbf{G}_2\,\mathbf{Z}_2\,) \\ \vdots \\ \mathrm{vec}(\,\mathbf{G}_L\,\mathbf{Z}_L\,) \end{pmatrix}, \qquad \mathbf{Z} = (\,\mathbf{Z}_1, \mathbf{Z}_2, \dots, \mathbf{Z}_L\,) = \mathbf{X}\,\mathbf{F}^T \in \mathbb{R}^{N_\theta\times M_{nl}}\,, \tag{13}$$

and this formulation requires

$$2N_rN_\theta M_{nl} + \sum_{l=1}^{L} 2m_n^l m_m^l N_\theta = 2N_\theta(N_r M_{nl} + M)$$

floating point operations and $M_{nl}N_r + M_{lm}N_\theta + M_{nl}N_\theta$ words of storage. For a given $\mathbf{K}$ we are free to choose either the formulation (12) or the one in (13).

Next consider the case $\mathbf{K}^T\mathbf{d}$: By applying Equation (A.4), inserting the definition of $\mathbf{d}$ and once again using the result from Lemma A.2 we obtain

$$\begin{aligned} \mathbf{K}^T\,\mathbf{d} &= \left(\mathbf{F}_1^T\otimes\mathbf{G}_1^T, \mathbf{F}_2^T\otimes\mathbf{G}_2^T, \dots, \mathbf{F}_L^T\otimes\mathbf{G}_L^T\right)\,\mathbf{d} \\ &= (\mathbf{F}_1^T\otimes\mathbf{G}_1^T)\,\mathrm{vec}(\mathbf{D}_1) + (\mathbf{F}_2^T\otimes\mathbf{G}_2^T)\,\mathrm{vec}(\mathbf{D}_2) + \cdots + (\mathbf{F}_L^T\otimes\mathbf{G}_L^T)\,\mathrm{vec}(\mathbf{D}_L) \\ &= \sum_{l=1}^{L} \mathrm{vec}(\mathbf{G}_l^T\,\mathbf{D}_l\,\mathbf{F}_l) \end{aligned}$$

Again we have the freedom to compute each term in the sum as either $((\mathbf{G}_l^T\,\mathbf{D}_l)\,\mathbf{F}_l)$ or $(\mathbf{G}_l^T\,(\mathbf{D}_l\,\mathbf{F}_l))$. The two choices lead to the following two computational procedures

$$\mathbf{K}^T\,\mathbf{d} = \mathrm{vec}(\mathbf{Z}\,\mathbf{F}), \qquad \mathbf{Z} = \left(\mathbf{G}_1^T\mathbf{D}_1, \mathbf{G}_2^T\mathbf{D}_2, \dots, \mathbf{G}_L^T\mathbf{D}_L\right) \in \mathbb{R}^{N_\theta\times M_{nl}}\,, \tag{14}$$

Table 1: Reduction in storage end execution time resulting from use of the Kronecker formulation, when $N_r = 101$ and $N_\theta = 51$.

	M	M_{nl}	M_{lm}	Storage	Operations	Measured
LOWL	3279	1102	294	33	4.6	29
GONG	4704	1556	793	43	2.7	18
MDI	30648	2036	5255	136	6.9	49
MDIm	150372	2029	31375	202	21.9	—

$$\mathbf{K}^T \mathbf{d} = \mathrm{vec}(\mathbf{G}^T \mathbf{Z}), \qquad \mathbf{Z} = \begin{pmatrix} \mathbf{D}_1\mathbf{F}_1 \\ \mathbf{D}_2\mathbf{F}_2 \\ \vdots \\ \mathbf{D}_L\mathbf{F}_L \end{pmatrix} \in \mathbb{R}^{M_{lm} \times N_r} , \tag{15}$$

With respect to the number of floating point operations and amount of storage required Equation (14) is identical to Equation (13) and Equation (15) is identical to Equation (12). The theorem follows by multiplying these numbers by 2 to account for the cost of computing the product with the second term in Equation (10). □

This result should be compared with the $2MN$ flops and MN words of storage required when $\mathbf{Kx}$ and $\mathbf{K}^T\mathbf{d}$ is computed in a straightforward way. We have implemented a fast matrix-vector multiplication routine KRONMV based on Equations (12)–(15) in FORTRAN. A few practical remarks about the implementation:

- If matrices are stored columnwise (as in FORTRAN) then the elements of $\mathbf{X}$ and those of the vector $\mathbf{x} = \mathrm{vec}(\mathbf{X})$ are layed out in memory in exactly the same way. Hence the vec(·) function simply corresponds to interpreting the two-dimensional array holding the $N_\theta \times N_r$ matrix $\mathbf{X}$ as a one-dimensional array holding the $N_\theta N_r \times 1$ vector $\mathbf{x}$ – no reshuffling of the elements is necessary.

- The block Kronecker formulation has the advantage that it consists mainly of matrix-matrix multiplications (BLAS 3) – two for each block in $\mathbf{K}$. This ensures good performance on modern pipelined CPU's with a hierarchical (cache-based) memory system. Furthermore, a highly optimized matrix-matrix multiplication subroutine (GEMM) is available on most platforms as part of the standard software libraries. For this reason, the actual reduction in execution time is often much larger than the one predicted by the decrease in the number of floating point operations (cf. the last two columns in Table 1).

- The block Kronecker formulation is well suited for implementation on a parallel computer since the computations associated with each block can be carried out independently.

In Table 1 we have listed the improvements achieved, when applying the Kronecker formulation to a number of datasets: LOWL (Tomczyk et. al. (1995)), GONG (for observation months 4–10) (Harvey et. al. (1996)) and MDI (144 days of observations) (Kosovichev et. al. (1997)). These are all real helioseismic datasets consisting of a-coefficients, while MDIm is an artificial set consisting of the individual frequency splittings for the modes in MDI, and is included to show that the Kronecker formulation becomes even more attractive when such datasets become available. The columns 'Operations' and 'Storage' list the factor by which the number of floating point operations and the amount of storage are reduced for a number of real datasets, when KRONMV is used instead of the normal matrix-vector multiplication routine GEMV to compute $\mathbf{K}\mathbf{x}$ or $\mathbf{K}^T\mathbf{d}$. Kernels were discretized on a 101×51 grid. The last column lists the actual reduction in execution time measured on a single processor of a Silicon Graphics PowerChallenge with 8 75MHz MIPS R8000 processors. The large discrepancy between the reduction in the number of operations and the speedup, is explained by the poor cache-utilization of GEMV, which is a BLAS-2 operation. As an example, GEMV only achieved a performance of 16.2 and 24.9 Mflops on the MDI and GONG data respectively, while KRONMV achieves 120 Mflops and 166 Mflops respectively on the two datasets.

4 Two iterative SOLA implementations

The fast matrix-vector multiplication algorithm described in the previous section has been used in two iterative implementations of the 2dSOLA method. Both implementations are based on the Lanczos bidiagonalization (LBD) (Golub & Van Loan (1989), Section 9.3), which accesses the coefficient matrix $\mathbf{K}$ only via matrix multiplication with $\mathbf{K}$ and $\mathbf{K}^T$. Given a unit starting-vector $\mathbf{u}_1$, the LBD algorithm computes two sets of orthogonal *Lanczos vectors* $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ such that after k iteration and with

$$\mathbf{U}_k = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{k+1}) \in \mathbb{R}^{N\times(k+1)} \,, \qquad \mathbf{V}_k = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k), \in \mathbb{R}^{M\times k}$$

we obtain a $(k+1)\times k$ bidiagonal matrix $\mathbf{B}_k$ satisfying

$$\mathbf{K}^T = \mathbf{U}_k \mathbf{B}_k \mathbf{V}_k^T \,. \tag{16}$$

In each step two matrix-vector multiplications are performed, one with $\mathbf{K}$ and one with $\mathbf{K}^T$. We can compute an approximate solution to Equation (6) using avariant of algorithm HBD where the full Householder bidiagonalization (7) has been replaced by the LBD (16), i.e. Equation (8) becomes

$$\min \left\| \begin{pmatrix} \mathbf{B}_k \\ \lambda \mathbf{I}_k \end{pmatrix} \mathbf{y}^{(k)} - \begin{pmatrix} \mathbf{U}_k^T \mathbf{t} \\ \mathbf{0} \end{pmatrix} \right\|_2 \,, \tag{17}$$

from which we get the kth solution as $\mathbf{c}^{(k)} = \mathbf{V}_k \mathbf{y}^{(k)}$.

An important property of the LBD is, that the singular values of $\mathbf{B}_k$ are usually good approximations to the k largest singular values of $\mathbf{K}$ or to put differently: The solution components in $\mathbf{c}$ corresponding to the large singular values tend to converge first in $\mathbf{c}^{(k)}$. Since these constitute the stable part of the solution in the presence of noise or measurement errors, this means that the Lanczos algorithm has a stabilizing or regularizing effect in itself. Consequently, there are two levels of regularization involved here: An “outer” regularization controlled by the number of iterations k and an “inner” regularization controlled by λ. In our implementation we have chosen not to use the outer regularization, but instead we continue the iteration until a solution, which is close to the one computed by the direct algorithm for the same value of λ, is found; this is discussed in further detail by Björck (1988).

In the 2dSOLA method we wish to solve Equation (6) for P, say, different right-hand sides each corresponding to a different target location (r_0, θ_0). The number of right-hand sides has a great impact on the efficiency of different implementations based on Equation (17). This is our main motivation for studying the two fundamentally different approaches described below.

4.1 Algorithm LBFR

One way to apply the LBD is by substituting it directly into Equation (8) as suggested above and solve the resulting equation (with multible right-hand sides). This approach was presented by Björck (1988) and O'Leary & Simmons (1982). To make this procedure numerically stable in finite precision arithmetic it is necessary to enforce the orthogonality among the Lanczos vectors, which is otherwise lost due to roundoff-errors. This is done by reorthogonalizing $\mathbf{u}_{k+1}$, which is generated in the kth step, against all the previous vectors $\mathbf{u}_i,\ i = 1, \ldots, k$ (and $\mathbf{v}_k$ against $\mathbf{v}_i,\ i = 1, \ldots, k-1$) using, e.g., the modified Gram-Schmidt procedure (Golub & Van Loan (1989), pp. 218). To do this we need to store the large $\mathbf{U}_k$ and $\mathbf{V}_k$ matrices in fast memory using a total of $k(M+N)$ words of storage. Moreover, the modified Gram-Schmidt reorthogonalization requires an extra $\frac{k^2}{2}(M+N)$ floating point operations. The advantage in the presence of many right-hand sides is that the LBD is computed only once, requiring only $2k$ matrix-vector multiplications. Since the number of iterations needed in practice is relatively small, the reduction in execution time compared to computing the full Householder bidiagonalization is still rather dramatic, while the amount of storage needed is only reduced by a small factor (which is certainly still very useful!). For the MDI dataset about 1000 iterations are sufficient, and this results in a reduction of the execution time by a factor of 50(!), while the amount of storage is reduced by a factor of 4.3 compared to algorithm HBD. Since the bidiagonalization is computed once and stored on disk for later use, this algorithm has the same advantage as HBD namely that it is cheap to re-compute the solution after changing the regularization parameter λ and/or the target functions. In the following we will refer to this algorithm as LBFR (Lanczos Bidiagonalization with Full Reorhogonalization).

4.2 Algorithm LSQR

The LSQR algorithm by Paige & Saunders (1982a,b) solves Equation (17) using a clever iteration scheme that updates a QR factorization of $\mathbf{B}_k$ and thereby allows $\mathbf{c}^{(k)}$ to be computed from $\mathbf{c}^{(k-1)}$ without ever storing $\mathbf{U}_k$ and $\mathbf{V}_k$. Unfortunately, the numerical

Table 2: Storage and execution time used by three different implementations for computing the 2d SOLA solution in every point of the 101×51 discretization grid.

		Storage/Mb			Time/minutes		
	M	Direct	Lanczos	LSQR	Direct	Lanczos	LSQR
LOWL	3279	140	75	11	12.3	2.5	90
GONG	4704	200	90	15	45	3.3	200
MDI	30648	1300	300	26	375	7.5	750
MDIm	150372	6200	1274	73	NA	26	2400

stability of the LSQR algorithm hinges on the fact that the starting vector $\mathbf{u}_1$ is chosen to be a normalized version of the right-hand side, i.e. $\mathbf{u}_1 \equiv \mathbf{t}/\|\mathbf{t}\|_2$. Therefore the LBD has to be computed once for every right-hand side, requiring a total of $2P\bar{k}$ matrix-vector multiplications, where he average number of iterations $\bar{k}$ is used due to the fact, that the number of iterations needed for convergence depends strongly on the target location such that points close to the surface, i.e. with $r/R \approx 1$ converge in only 20-40 iterations while points below $r/R = 0.5$ typically need up to a thousand iteration to converge.

If the number of right-hand sides is very small ($\approx$10–50) this algorithm is faster than LBFR, since the reorthogonalization is avoided, but for large P it is very slow. Also, the solution has to be recomputed from scratch when λ or the target functions are changed. One very attractive property of this algorithm, however, is its very small memory consumption, which makes it usable even on small workstations or PC's (in our implementation only $4N + 18M$ words are used in addition to the storage needed for $\mathbf{K}$). Moreover, the solutions corresponding to different right-hand sides (target location) can be computed completely independently and in parallel using, e.g., a cluster of workstations. Using this algorithm, the MDI dataset can be inverted using only 26Mb of memory, but the execution time is about twice that of algorithm HBD. In the following we will refer to this algorithm as LSQR.

4.3 Results

In Table 2 we have listed the result of inverting the datasets used in Table 1 on 5151 targets locations. For the iterative methods we used the somewhat artificial stopping criterion that the computed solution be equal within the error bars to the solution computed by HBD. In LBFR $k = 1000$ iterations were used. In LSQR an average of $\bar{k} \approx 100$ iterations were used pr. target point and the total execution time was estimated from the time spent solving for a small subset of the target locations.

All experiments were carried out on a Silicon Graphics PowerChallenge with 8 75MHz MIPS R8000 processors. It was not possible to run HDB on the large MDIm dataset due to the large amount of memory needed. Looking at the table, the time/space tradeoff between the two algorithms LBFR and LSQR is striking, and the fact that the large MDIm dataset with 150372 modes can be inverted in 26 minutes is quite a remarkable result.

5 Conclusion

Regarded as a case study in scientific computing, the investigation of algorithms for two-dimensional helioseismic inversion presented in this paper, demonstrates that in many applications it is simply not enough to use standard algorithms from numerical linear algebra, say, even though they may be state-of-the-art methods for the general problem they solve. Instead, it is necessary to include special knowledge about the problem at hand – in this case the structure of the kernels – in order to obtain an efficient algorithm. In two-dimensional helioseismic inversion this approach represents a substantial breakthrough for the treatment of large datasets, and the reduction in complexity that the new methods developed here represent, will allow us to treat even the largest future datasets expected from the combination of MDI and GONG data with relative ease.

Acknowledgements: Most of this work was done during the 7 months that the author spent at JILA at the University of Colorado in Boulder. The author would like to thank the Faculty of Science at the University of Aarhus and the EPOS project for supplying the funds that made this stay possible. Also many thanks to Prof. Juri Toomre, JILA for his great hospitality and to A. G. Kosovichev, Jesper Schou and the SOHO team for inviting me to spend a very inspiring month at Stanford University. Finally many thanks to my advisor Jørgen Christensen-Dalsgaard for countless discussions, suggestions and bug-reports.

References

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

[2] Björck, Å., 1988, *BIT*, **28**, 659.

[3] Björck, Å., 1996, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia.

[4] Christensen-Dalsgaard, J., Schou, J., Thompson, M. J., 1990, *Mon. Not. R. astr. Soc.* **242**, 353.

[5] Eldén, L., 1977, *BIT* **17**, 134.

[6] Fausett, D. W., Fulton, C. T., 1994, *SIAM J. Matrix Anal. Appl.* **15**, 219.

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

[8] Graham, A., 1981, *Kronecker Products and Matrix Calculus: with Applications*, Halsted Press, John Wiley and Sons, Inc., New York.

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

[10] Hansen, P. C., O'Leary, D. P, .1993, *SIAM J. Sci. Comput.* **14**, 1487–1503.

[11] Harvey, J. W., Hill, F., Hubbard, R. P., Kennedy, J. R., Leibacher, J. W., Pintar, J. A., Gilman, P. A., Noyes, R. W., Title, A. M., Toomre, J., Ulrich, R. K., Bhatnagar, A., Kennewell, J. A., Marquette, W., Partrón, J., Saá, O. & Yasukawa, E., 1996, *Science*, **272**, 1284 – 1286.

[12] Kosovichev, A. G., Schou, J., Scherrer, P. H., Bogart, R. S., Bush, R. I., Hoeksema, J. T., Aloise, J., Bacon, L., Burnette, A., de Forest, C., Giles, P. M., Leibrand, K., Nigam, R., Rubin, M., Scott, K., Williams, S. D., Basu, S., Christensen-Dalsgaard, J., Däppen, W., Rhodes Jr, E. J., Duvall Jr, T. L., Howe, R., Thompson, M. J., Gough, D. O., Sekii, T., Toomre, J., Tarbell, T. D., Title, A. M., Mathur, D., Morrison, M., Saba, J. L. R., Wolfson, C. J., Zayer, I. & Milford, P. N., 1997, *Solar Phys.*, **170**, 43 – 61.

[13] Larsen, R. M., Hansen, P. C., 1997, *Astron. Astrophys. Suppl.* **121** , 587 – 598.

[14] O'Leary, D. P., Simmons, J. A., 1981, *SIAM J. Sci. Statist. Comput.*, **2**, 474–489.

[15] Paige, C. C., Saunders, M. A., 1982, *ACM Trans. Math. Software* **8**, 43.

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

[17] Pijpers, F. P., Thompson, M. J., 1992, *Astron. Astrophys.* , **262**, L33.

[18] Pijpers, F. P., Thompson, M. J., 1996, *Mon. Not. R. astr. Soc.*, **279**, 498 – 510.

[19] Schou, J., Christensen-Dalsgaard, J. & Thompson, M. J., 1994, *Astrophys. J.*, **433**, 389 – 416.

[20] Sekii, T., 1993, *Mon. Not. R. astr. Soc.*, **264**, 1018 – 1024.

[21] Tikhonov, A. N., 1963, *Soviet Math. Dokl.* **4**, 1035–1038.

[22] Tomczyk, S., Streander, K., Card, G., Elmore, D., Hull, H. & Caccani, A., 1995, *Solar Phys.*, **159**, 1–21.

[23] Thompson, M. J., Toomre, J., Anderson, E. R., Antia, H. M., Berthomieu, G., Burton-clay, D., Chitre, S. M., Christensen-Dalsgaard, J., Corbard, T., DeRosa, M., Genovese, C. R., Gough, D. O., Haber, D. A., Harvey, J. W., Hill, F., Howe, R., Korzennik, S. G., Kosovichev, A. G., Leibacher, J. W., Pijpers, F. P., Provost, J., Rhodes Jr., E. J., Schou, J., Sekii, T., Stark, P. B., Wilson, P. R., 1996, *Science*, **272**, 1300 – 1305.

A The Kronecker product

Linear least squares problems on Kronecker product form have a very regular block structure which allows them to be solved with great savings in computing time and storage compared to general problems of the same dimensions. In this appendix we will define the Kronecker product (also known as tensor or direct product) and list a number of its useful properties. More detailed descriptions of Kronecker products, their properties and algorithms can be found in, e.g., Björck (1996), Fausett & Fulton (1994), Graham (1981) and references therein.

Definition A.1 *If* $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{B} \in \mathbb{R}^{p\times q}$ *then* $\mathbf{A}\otimes\mathbf{B}$ *is the* **Kronecker product** *of* $\mathbf{A}$ *and* $\mathbf{B}$*, which is the* $mp \times nq$ *block matrix*

$$\mathbf{A}\otimes\mathbf{B} \equiv \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{12}\mathbf{B} & a_{22}\mathbf{B} & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & \cdots & a_{mn}\mathbf{B} \end{pmatrix}$$

Hence $\mathbf{A}\otimes\mathbf{B}$ is the matrix which contains all the possible products of elements from $\mathbf{A}$ and $\mathbf{B}$. Lemma A.1 below lists some fundamental properties of the Kronecker product.

Lemma A.1 *If matrices* $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ *and* $\mathbf{D}$ *are given such that the additions and multiplications below are defined, then*

$$(\mathbf{A}+\mathbf{B})\otimes\mathbf{C} = (\mathbf{A}\otimes\mathbf{C}) + (\mathbf{B}\otimes\mathbf{C}) \tag{A.1}$$

$$\mathbf{A}\otimes(\mathbf{B}+\mathbf{C}) = (\mathbf{A}\otimes\mathbf{B}) + (\mathbf{A}\otimes\mathbf{C}) \tag{A.2}$$

$$\mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}) = (\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C} \tag{A.3}$$

$$(\mathbf{A}\otimes\mathbf{B})^T = \mathbf{A}^T\otimes\mathbf{B}^T \tag{A.4}$$

$$(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = \mathbf{AC}\otimes\mathbf{BD} \tag{A.5}$$

Proof. Equations (A.1) to (A.4) follow immediately from the definition. The proof for Equation (A.5) is straightforward and can be found in, e.g., Graham [8]. □

The highly regular structure can be exploited when multiplying a vector by a Kronecker product. To show this we first introduce the function $\text{vec}(\cdot) : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ which stacks the columns of a matrix into a long vector. If $\mathbf{A} = (a_1, a_2, \ldots, a_n) \in \mathbb{R}^{m\times n}$ then we define

$$\text{vec}(\mathbf{A}) = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} . \tag{A.6}$$

We can now state the following important result:

Lemma A.2 *If* $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$*, and* $\mathbf{x} = \text{vec}(\mathbf{X})$*, where* $\mathbf{X} \in \mathbb{R}^{q\times n}$*, then*

$$(\mathbf{A}\otimes\mathbf{B})\,\mathbf{x} = \text{vec}(\mathbf{B}\,\mathbf{X}\,\mathbf{A}^T) . \tag{A.7}$$

Proof. See Graham [8]. □

Consequently, if $\mathbf{A}$ and $\mathbf{B}$ are $n\times n$ matrices the cost of computing $(\mathbf{A}\otimes\mathbf{B})\,\mathbf{x}$ is $2n^4$ while $\text{vec}(\mathbf{B}\,\mathbf{X}\,\mathbf{A}^T)$ requires only $4n^3$ floating point operations.