AIR Tools II: algebraic iterative reconstruction methods, improved implementation

Hansen, Per Christian; Jørgensen, Jakob Sauer

Published in:
Numerical Algorithms

Link to article, DOI:
10.1007/s11075-017-0430-x

Publication date:
2017

Document Version
Peer reviewed version

Citation (APA):
Hansen, P. C., & Jørgensen, J. S. (2017). AIR Tools II: algebraic iterative reconstruction methods, improved implementation. *Numerical Algorithms*, 1-31. https://doi.org/10.1007/s11075-017-0430-x

Numer Algor
https://doi.org/10.1007/s11075-017-0430-x

ORIGINAL PAPER

AIR Tools II: algebraic iterative reconstruction methods, improved implementation

Per Christian Hansen[1] · **Jakob Sauer Jørgensen**[2]

Received: 19 June 2017 / Accepted: 19 October 2017

Abstract We present a MATLAB software package with efficient, robust, and flexible implementations of algebraic iterative reconstruction (AIR) methods for computing regularized solutions to discretizations of inverse problems. These methods are of particular interest in computed tomography and similar problems where they easily adapt to the particular geometry of the problem. All our methods are equipped with stopping rules as well as heuristics for computing a good relaxation parameter, and we also provide several test problems from tomography. The package is intended for users who want to experiment with algebraic iterative methods and their convergence properties. The present software is a much expanded and improved version of the package AIR TOOLS from 2012, based on a new modular design. In addition to improved performance and memory use, we provide more flexible iterative methods, a column-action method, new test problems, new demo functions, and—perhaps most important—the ability to use function handles instead of (sparse) matrices, allowing larger problems to be handled.

This work is a part of the project HD-Tomo funded by Advanced Grant No. 291405 from the European Research Council. Networking support was provided by the EXTREMA COST Action MP1207.

The work was carried out while the author Jakob Sauer Jørgensen was with the Department of Applied Mathematics and Computer Science, Technical University of Denmark.

✉ Per Christian Hansen
pcha@dtu.dk

Jakob Sauer Jørgensen
jakob.jorgensen@manchester.ac.uk

[1] Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

[2] Present address: School of Mathematics, University of Manchester, Manchester M13 9PL, UK

Keywords Algebraic iterative reconstruction · ART methods · SIRT methods · Column-action methods · Semi-convergence · Stopping rules · Tomographic imaging

Mathematics Subject Classification (2010) 65F10 · 65F22

1 Introduction

Algebraic iterative reconstruction (AIR) methods are popular methods for computing regularized solutions to discretizations of linear inverse problems,

$$A\,x \approx b, \qquad A \in \mathbb{R}^{m\times n}, \tag{1}$$

in particular those arising in computed tomography (CT) [5, Chapter 6]. These methods rely on semi-convergence where the number of iterations acts as regularization parameter. A MATLAB package AIR TOOLS with a collection of such methods has been available from the first author since 2012; the package was outlined in [24] but the software was never officially published.

Based on our experience and feedback from the users, the time is ripe to officially publish this updated and expanded version that also improves on the performance and efficiency of the methods. A particularly powerful expansion is that our iterative methods can now be used in matrix-free mode where the matrix-vector multiplications with A and its transpose are done in functions. This feature also allows us to use optimized software from other packages.

Our package is written solely in MATLAB with the main purpose of providing easy-to-use implementations of the most important AIR methods, along with relevant 2D test problems. Each method is equipped with several stopping rules as well as heuristics for choosing the relaxation parameter. The software is simple to install and does not rely on other software or toolboxes—apart obviously from the case where the user explicitly wishes to use other packages in conjunction with AIR TOOLS II.

The software is distributed as a compressed archive; uncompressing the file will create a directory which contains the code. More information can be found in the `README.txt` file contained in the package. The software is available from Netlib http://www.netlib.org/numeralgo/ as the `na47` package. Maintenance of the code is available from GitHub: https://github.com/jakobsj/AIRToolsII.

There are a few other public-domain software packages that include AIR methods: ASTRA [41], SNARK09 [30], and TIGRE [40]. These packages focus on large-scale CT problems; they lack the generality, flexibility, and completeness of our package, and they provide neither stopping rules nor parameter-choice methods. Our package is not a competitor to dedicated CT packages; it is written for numerical analysts and application specialists who want to experiment with the iterative methods and their convergence properties. At the same time, AIR TOOLS II can handle large-scale problems by interfacing to (GPU-accelerated) third-party software implementations of the computationally expensive applications of the matrix A and its transpose.

Algebraic iterative reconstruction methods take the overall form of either sequential and simultaneous versions—plus their block variants. We use the following acronyms (see, e.g., the survey in [25]):

- *ART:* algebraic reconstruction technique. Methods that sequentially involve one row at a time. The original algorithm is due to Kaczmarz [29]; the name ART originates from the seminal paper [20] by Gordon, Bender, and Herman.
- *CART:* column-action reconstruction technique. Methods that sequentially involve one column at a time [13]; these methods are sometimes referred to as column-relaxation methods [44].
- *SIRT:* simultaneous iterative reconstruction technique. Methods that simultaneously involve all the rows at a time and, therefore, are based on matrix multiplications. This name[1] originates from the paper [19] by Gilbert; see also [42].

All the iterative methods can include constraints that can be formulated as a projection on a convex set; we provide box constraints which include, as a special case, nonnegativity constraints. Our package does not include implementations of any block versions [39] of the above methods; such methods are highly suitable for large-scale problems but MATLAB is not the right platform for such software.

Where possible, we have striven for backward compatibility with AIR TOOLS, but in certain cases, we found it necessary to deviate from this principle, e.g., when improved functionality is provided or when the old input or output parameters were sub-optimal. The revisions and modifications, compared to AIR TOOLS, take the following forms.

1. The code itself is now written in a modular fashion, which is better suited for test, maintenance, and future expansion of the code.
2. All iterative methods now accept either a (sparse) matrix or a function handle as input for the matrix A, allowing larger problems to be handled and also allowing for handling problems where a matrix is not explicitly available.
3. The iterative methods can easily be customized by the user: in the ART and CART methods, the user can specify a row ordering, and in the SIRT methods, the user can specify two weight matrices that define the methods.
4. We added options for showing a “waitbar” and printing on-screen information during the iterations (“verbose”).
5. The SIRT methods exhibit a fully deterministic behavior in spite of the use of random numbers for computing the relaxation parameter via MATLAB’s `eigs`.
6. It is now possible to add a damping parameter in the ART and CART methods, e.g., to robustly handle small row or column norms.
7. It is now possible to use a diminishing relaxation parameter in the ART methods, given by a user-defined function.

[1]In the CT community, the acronym “SIRT” is often associated with a specific simultaneous method called “SART” in the numerical linear algebra community and in our package.

8. Some stopping rules did not work well, and we changed these rules and their implementation such that they are more robust.
9. We provide three more test problems (including the spherical Radon transform), as well as a collection of test phantoms that can be used for all the test problems.
10. We provide a function which, given the matrix A, illustrates the geometry of the underlying test problem.
11. The code is, as much as possible, optimized for speed and memory use—e.g., by avoiding duplication of variables and by blocking matrix operations.
12. Some *input parameters*, or their names, were not well chosen and we have revised these choices. For example, instead of simple box constraints $x_i \in [0, u]$, we now allow more general box constraints $x_i \in [\ell_i, u_i]$ for all pixels.
13. The format of the *output parameters* was changed and we now return more parameters, such as the relaxation parameter chosen by the software.
14. A few *default parameters* were not optimal, and we have changed them to more intuitive values. For example, in the `fancurvedtomo` test problem (previously called `fanbeamtomo`), the default number of projection angles is now identical to that in `paralleltomo`, such that the two functions—with default parameters—produce coefficient matrices of the same size.
15. We added an auxiliary function `fbp` that computes the filtered back projection solution to a parallel-beam tomography problem, using the corresponding matrix A.

Users who upgrade from an older version of the code should note the following changes in the fields of the `options` input struct:

– The name of the relaxation parameter is changed from `lambda` to `relaxpar`.
– The logical `nonneg` is replaced by a scalar or vector `lbound`.
– The scalar `ubound` is replaced by a scalar or vector with the same name.
– The struct `restart` is removed.
– The weighting vector `w` in `cav`, `cimmino`, and `drop` is removed.

There are also changes in the output:

– The vector `info` is changed to a struct with the same name and which contains more information about the computations.
– The struct `restart` is changed to a struct `ext_info` with additional information.

This paper focuses on descriptions of the improvements and additions to the original software AIR TOOLS, and we refer to [24] for more details about the original design of the package. Our paper is organized as follows. Section 2 gives an overview of the package, including descriptions of the iterative methods and their convergence, the stopping rules, and the relaxation-parameter choices. Section 3 describes the test problems, and in Section 4, we list the main considerations about our improved implementation. Finally, in Sections 5 and 6, we present some of the main new features of the software.

2 Overview of the package

The functions fall into the seven categories listed in Table 1. A number of changes were made in these functions, compared to the original paper [24].

- There are four new functions with iterative methods: `art`, `cart`, `columnaction`, `sirt`.
- There are five new test problems: `fanlineartomo`, `phantomgallery`, `seismicwavetomo`, `show_tomo`, `sphericaltomo`.
- The new function `train_relaxpar` collects the previous functions `trainLambdaART` and `trainLambdaSIRT`, and adds new functionality.
- The functions `fancurvedtomo`, `purge_rows`, and `train_dpme` are new names for `fanbeamtomo`, `rzr`, and `trainDPME`, respectively.
- The demo scripts are renamed and updated, and we added several more demos that illustrate the new features; see Table 1 for details.
- There are seven new auxiliary functions as listed in Table 1.

Below we give a brief overview of the methods underlying the functions; more details can be found in the respective references.

2.1 ART, CART, and SIRT methods

All iterative methods in this package take as input the coefficient matrix A (which is typically sparse) and the right-hand side vector b. Instead of the explicit matrix A, our software accepts also a function handle to a function that performs multiplication with A and its transpose, which allows our software to handle larger problems and utilize such functions from external software.

It is possible, in all the iterative methods, to include an orthogonal projection $P_{\mathcal{C}}$ on the convex set $\mathcal{C}$. We provide general box constraints described by the set

$$\mathcal{C} = [\ell_1, u_1] \times [\ell_2, u_2] \times \cdots \times [\ell_n, u_n], \tag{2}$$

where the lower and upper bounds are allowed to be `-Inf` and `Inf`, respectively. This includes, as a special case, the common case of nonnegativity bounds $\mathcal{C} = \mathbb{R}^n_+$.

The three *row-action* ART methods in this package involve an update of the iteration vector x of the form

$$x \leftarrow P_{\mathcal{C}}\left(x + \omega \, \frac{b_i - a_i^T x}{\|a_i\|_2^2 + \alpha} \, a_i \right), \qquad 1 \le i \le m, \tag{3}$$

where b_i is the ith component of the right-hand side, a_i^T is the ith row of the coefficient matrix, and ω is a relaxation parameter which is either constant or decreases with the iterations. Moreover, α is a damping parameter introduced in [2] that stabilizes the iterations for small row norms; we choose

$$\alpha = \mathtt{damp} \cdot \max_i \|a_i\|_2^2, \tag{4}$$

Table 1 Overview of the software in AIR TOOLS II

Category	Functions	
ART methods	`art`	General interface for all the ART methods
	`kaczmarz`	Kaczmarz's method with cyclic row sweep
	`randkaczmarz`	Kaczmarz's method with random row selection
	`symkaczmarz`	Kaczmarz's method with "symmetric" (top $\rightarrow$ bottom $\rightarrow$ top) row sweep
CART methods	`cart`	General interface for the CART methods
	`columnaction`	Column-action method with cyclic col. sweep
SIRT methods	`sirt`	General interface for all the SIRT methods
	`cav`	Component averaging (CAV) method
	`cimmino`	Cimmino's method
	`drop`	Diagonally relaxed orthogonal projection (DROP) method
	`landweber`	Landweber's method
	`sart`	Simultaneous algebraic reconstruction technique (SART)
Training methods	`train_dpme`	Use training to compute a good multiplicative factor in the `DP` and `ME` stopping rules
	`train_relaxpar`	Use training to compute a good relaxation parameter for the ART/CART/SIRT methods
Test problems	`fancurvedtomo`	2D fan-beam CT with a curved detector (equal angles between rays)
	`fanlineartomo`	2D fan-beam CT with a linear detector
	`paralleltomo`	2D parallel-beam CT
	`phantomgallery`	A collection of different 2D phantoms
	`purge_rows`	Remove empty or very sparse rows of A and the corresponding elements of b
	`seismictomo`	2D seismic travel-time tomography
	`seismicwavetomo`	Similar to `seismictomo` but without the ray assumption
	`show_tomo`	Illustrate the geometry of a tomographic test problem using the rows of the matrix A
	`sphericaltomo`	2D spherical Radon transform tomography
Demo scripts	`demo_art`	How to use the ART methods
	`demo_astra_2d`	How to use external code
	`demo_cart`	How to use the CART method
	`demo_constraints`	How to use constraints
	`demo_custom_all`	Customization of the ART and SIRT methods
	`demo_custom_sirt`	A specific customized SIRT method
	`demo_matrixfree`	How to use the matrix-free mode
	`demo_relaxpar`	How to set the relax. parameter strategy
	`demo_show_tomo`	How to use the `show_tomo` function
	`demo_sirt`	How to use the SIRT methods
	`demo_stoprules`	How to set the stopping rule
	`demo_training`	How to use the training methods
Auxiliary	`afun_astra_2d_gpu`	Wrap ASTRA forward and back projectors into a function handle
	`afun_matrix`	Wrap a matrix into a function handle
	`calc_relaxpar`	Compute the relaxation parameter
	`check_inputs`	Check inputs and set default values
	`check_stoprules`	Check if stopping rule criteria are met
	`fbp`	Filtered back projection that explicitly uses the matrix A^T for the back projection
	`get_mfun_dfun`	Compute M and D for the SIRT methods

where damp is a user-specified parameter (default to zero). What distinguishes the different methods is the strategy for selecting the ith row at the kth sweep over the rows, as listed in Table 2.

We provide one *column-action* CART method that uses an update of the form

$$x_j \leftarrow P_{\mathcal{C}}\left(x_j + \omega \frac{c_j^T (b - A\,x)}{\|c_j\|_2^2 + \alpha} \right), \qquad 1 \le j \le n, \tag{5}$$

where x_j is the jth component of x, and c_j is the jth column of the coefficient matrix. Similar to before, we use $\alpha = \texttt{damp} \cdot \max_j \|c_j\|_2^2$. The columns are selected in a cyclic fashion, cf. Table 2. In our implementation of CART, it is possible to use the "flagging" strategy from [13] where update steps (5) are skipped if the update is small; this can save a substantial amount of computational work.

As a new feature that gives more flexibility, the user can now define *customized* ART and CART methods by specifying a fixed, alternative, ordering of the rows and columns, respectively. An example of this is given in the new demo script demo_custom_all.

For all ART and CART methods, to guarantee asymptotic convergence (for $k \to \infty$), the relaxation parameter must satisfy

$$0 < \omega < 2 \tag{6}$$

and our software checks if a user-supplied ω satisfies this criterion. The default value is $\omega = 1$ for ART and $\omega = 0.25$ for CART.

Table 2 Overview of ART and CART methods, where i or j is the choice of row/column index and k counts the number of sweeps over the rows/columns

Method	Choice of i and j	One iteration	Ref.
Custom ART art	The user specifies a row ordering via an integer vector I	m updates: $i = \mathtt{I}(1), \mathtt{I}(2), \ldots, \mathtt{I}(m)$	
Also generic interface to kaczmarz, symkaczmarz, and randkaczmarz			
Kaczmarz kaczmarz	$i = k \pmod m$	m updates: $i = 1, 2, \ldots, m$	[29]
Symmetric Kaczmarz symkaczmarz	$i = \begin{cases} i_0, & 1 \le i_0 \le m{-}1 \\ m{-}i_0{+}1, & m \le i_0 \le m{-}2 \end{cases}$ where $i_0 = k \pmod{2m}$	$m - 1$ updates: $i = 1, 2, \ldots, m{-}1$ or $i = m, m{-}1, \ldots, 2$	[4]
Randomized Kaczmarz randkaczmarz	i = random choice in $\{1, 2, \ldots, m\}$ with probability proportional to $\|a_i\|_2$	m random updates	[38]
Custom CART cart	The user specifies a column ordering via an integer vector J	n updates: $j = \mathtt{J}(1), \mathtt{J}(2), \ldots, \mathtt{J}(n)$	
Also generic interface to columnaction			
Column action columnaction	$j = k \pmod n$	n updates: $j = 1, 2, \ldots, n$	[13, 44]

If the system (1) is inconsistent, then a constant relaxation parameter ω leads to cyclic asymptotic convergence for Kaczmarz's method, and to avoid this, one must use a diminishing step size [3]. Therefore, in the ART methods, the user can supply a function (instead of a constant) with a diminishing parameter, e.g., $\omega_\ell = 1/\sqrt{\ell}$ where ℓ counts the total number of row updates.

All five SIRT methods in this package involve updates of the form

$$x \leftarrow P_{\mathcal{C}}\Big(x + \omega_k\, D\, A^T M\, (b - A\, x)\Big)\,, \qquad k = 1, 2, 3, \ldots, \tag{7}$$

where ω_k is the relaxation parameter in iteration k while D and M are diagonal matrices with positive diagonal elements and, hence, symmetric and positive (SPD), as specified in Table 3. This table also lists the bounds on the relaxation parameter ω_k to ensure asymptotic convergence; these bounds are derived from the general criterion

$$\omega_k < 2/\rho(D\, A^T M A), \qquad \rho(\cdot) = \text{ spectral radius }, \tag{8}$$

and using that $\rho(A^T A) = \|A\|_2^2$ and $\rho(A^T M A) = \|A^T M A\|_2$ when M is SPD. For SART, it proved in [6] that $\rho(D\, A^T M A) \leq 1$, meaning that the criterion $\omega_k < 2$ is sufficient (and easy to check) but not necessary. Our software checks if a user-supplied constant relaxation parameter $\omega_k = \omega$ satisfies these conditions. The default value is $\omega = 1.9/\rho(D\, A^T M A)$.

A new feature, similar to ART and CART, is that the user can now define a *customized* SIRT method by specifying the two matrices D and M, either as matrices or as vectors representing the diagonal of diagonal matrices of appropriate size. To ensure convergence, these matrices must be SPD; but we do not check this. An application of this feature is presented in Section 5.2.

Table 3 Overview of the SIRT methods; I_m and I_n denote identity matrices of size $m \times m$ and $n \times n$

Method	Choice of diagonal matrices	Relax. parameter	Ref.
Custom SIRT `sirt`	The user specifies symmetric and positive definite D and M	$0 < \omega_k < 2/\rho(D\, A^T M A)$	
Also interface to `landweber`, `cimmino`, `cav`, `drop`, and `sart`			
Landweber `landweber`	$D = I_n,\ M = I_m$	$0 < \omega_k < 2/\|A\|_2^2$	[32]
Cimmino `cimmino`	$D = I_n,\ M_{ii} = m^{-1}\,\|a_i\|_2^{-2}$	$0 < \omega_k < 2/\|A^T M A\|_2$	[9]
CAV `cav`	$D = I_n,\ M_{ii} = \|a_i\|_S^{-2}$ where $\|z\|_S^2 \equiv \sum_{j=1}^n z_j^2 \mathrm{nnz}(c_j)$	$0 < \omega_k < 2/\|A^T M A\|_2$	[8]
DROP `drop`	$D_{jj} = \mathrm{nnz}(c_j)^{-1},\ M_{ii} = \|a_i\|_2^{-2}$	$0 < \omega_k < 2/\rho(D\, A^T M A)$	[7]
SART `sart`	$D_{jj} = \|c_j\|_1^{-1},\ M_{ii} = \|a_i\|_1^{-1}$	$0 < \omega_k < 2$	[1]

By defining the iterations as shown in Tables 2 and 3, we ensure that the amount of computational work per iteration is almost identical for all the methods. One iteration in the ART and CART methods, as implemented in this package, is sometimes called a sweep in other works. For consistency with the other methods, we define one `symkaczmarz` iteration as either a "down" or an "up" sweep of the rows, and for this method, the number of iterations must be an even number.

It is interesting to note that all the AIR methods discussed here can be interpreted as optimization methods for (weighted) least squares problems. The ART methods are special instances of projected incremental gradient methods [2], CART is a cyclic coordinate descent algorithm [45], and the SIRT methods are (projected) gradient methods with different scalings.

2.2 Convergence and semi-convergence

For the unconstrained SIRT methods, the convergence analysis can be cast in terms of the singular value decomposition; see, e.g., [18]. For all other methods, the convergence aspects are surprisingly complex, as they depend on both the rank of the matrix and the consistency of the system $A\,x \approx b$ in (1).

For example, consider Kaczmarz's method with a fixed relaxation parameter. If the system is consistent (i.e., b lies in the range of A), then this method converges to the unique solution of minimum 2-norm, independently of the rank of A, provided that x^0 lies in the range of A, e.g., when $x^0 = 0$. But if the system is inconsistent and overdetermined ($m > n$), then, independently of the rank of A, this method does not exhibit asymptotic converge for $k \to \infty$ (we have cyclic convergence [16]).

When we face noisy data—which is typical for CT applications—we rely on the *semi-convergence* properties of the iterative methods, and therefore, we do not elaborate on the asymptotic convergence aspects. To set the stage, we write the noisy data as

$$b = \bar{b} + e, \qquad \bar{b} = A\,\bar{x} = \text{exact data}, \qquad \bar{x} = \text{exact solution}, \qquad e = \text{noise}. \quad (9)$$

Moreover, we let x^k denote the kth iterate of any of the iterative methods, with or without constraints. Semi-convergence—as coined by Natterer [33]—denotes the following scenario:

– During the initial iterations, x^k tends to approach the desired but un-obtainable exact solution $\bar{x}$ to the noise-free problem.
– During later iterations, x^k converges, as specified by the asymptotic convergence theory, to an undesired solution associated with the noisy data (e.g., $A^{-1}b$ if the system matrix is invertible).
– If we can stop the iterations just when the convergence behavior changes from the former to the latter, then we achieve a regularized solution—an approximation to the noise-free solution which is not too perturbed by the noise in the data [21].

The mechanism underlying this semi-convergence can be explained by splitting the reconstruction error $\bar{x} - x^k$ for the kth iteration vector x^k into two components:

$$\bar{x} - x^k = \left(\bar{x} - \bar{x}^k\right) + \left(\bar{x}^k - x^k\right),$$

where the "clean" iteration vector $\bar{x}^k$ corresponds to the noise-free data $\bar{b} = A\,\bar{x}$.

- The first component $\bar{x} - \bar{x}^k$ is the *iteration error* which is independent of the noise in the data. This component decreases as described by the asymptotic convergence theory for the particular method.
- The second component $\bar{x}^k - x^k$ is the *noise error* which is due to the presence of the data error e. This component tends to increase with the number of iterations, and the key to demonstrate semi-convergence for a particular method is to show how fast this component increases.

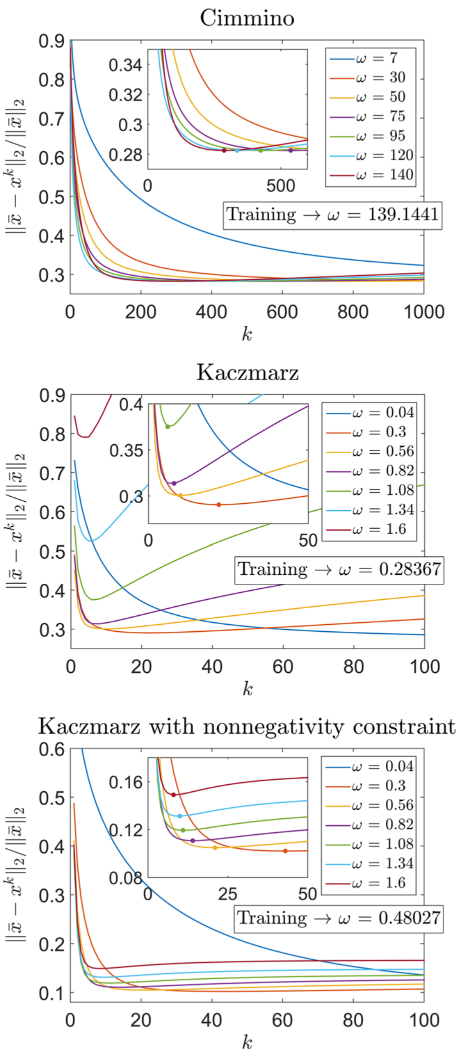


Fig. 1 Illustration of the influence of a constant relaxation parameter $\omega_k = \omega$ on the semi-convergence; the insets show relevant zooms on the minima. The top plot shows that for SIRT methods, the minimum reconstruction error is almost independent on ω (except when it is close to its maximum value). The middle and bottom plots show that the situation is different for the ART methods, both without and with nonnegativity constraints; here, a smaller ω gives a smaller reconstruction error. We also show the values of ω found by the function `train_relaxpar` described in Section 2.5

Figure 1 shows numerical examples of semi-convergence and, in particular, how the relaxation parameter ω influences this behavior. The figure also shows the specific value of ω chosen by the training method discussed in Section 2.5. For the unconstrained SIRT methods, it is fairly straightforward to show semi-convergence in terms of an SVD analysis [18]. Otherwise, the analysis is more involved; see, e.g., [13–15].

2.3 Stopping rules

Stopping rules for iterative methods that solve well-conditioned systems $A\,x = b$ or $\min_x \|A\,x - b\|_2$ typically terminate the iterations when the residual norm $\|b - A\,x^k\|_2$ or normal-equations residual norm $\|A^T(b - A\,x)\|_2$ is sufficiently small. However, for discretizations of ill-posed problems, it is well known that such a small residual does not imply a good approximate solution [21], and hence, these traditional stopping rules can not be used here.

Instead, we must rely on the semi-convergence of the iterative methods, where the number of iterations plays the role of a regularization parameter [21]. Therefore, many of the parameter-choice rules developed for inverse problems can be directly applied as stopping rules for the AIR methods.

Specifically, we need stopping rules—often based on heuristic methods—that seek to identify the change of convergence behavior mentioned above. There is a rich literature on such methods, but not all of them are robust enough to be included in a package like this. We have carefully selected the stopping rules listed in Table 4, with some revisions as listed below.

There are several other good stopping rules that we intentionally did not implement, such as the UPRE and GCV methods [43, Chapter 7]. These methods require an estimate of the trace of the matrix $A A_k^{\#}$, where $A_k^{\#}$ is a method-specific matrix

Table 4 Overview of the stopping rules, where $r^k = b - A\,x^k$, δ is an estimate of $\|e\|_2$, and τ is a "safety factor" slightly larger than 1

Name	Rule	Used in	Ref.
Maximum number of iterations	$k = k_{\max}$	All methods	
DP—discrepancy principle	$\|r^k\|_2 < \tau\,\delta$	All methods	[17]
ME—monotone error	$1/2\,(r^k)^T (r^{k-1} + r^k)/\|r^k\|_2 < \tau\,\delta$	All SIRT methods	[17, 26]
NCP—normalized cumulative periodogram	1D: $\min_k \|v(r^k) - v_{\mathrm{white}}\|_2$ 2D: $\min_k p^{-1} \sum_{\ell=1}^{p} \|v(r_\ell^k) - v_{\mathrm{white}}\|_2$ see (10), (11) for defs. of $v(\cdot)$ and r_ℓ^k	All methods	[22, 36]

such that $x^k = A_k^{\#} b$. The computation of this trace estimate doubles the amount of work in each iteration, and hence, we did not find such stopping rules suited for this package.

2.3.1 *Implementation of the DP and ME rules*

The discrepancy principle (DP) and monotone error (ME) stopping rules are special cases of a more general rule described in [17]. Ideally, they terminate the iterations at the smallest k such that

$$\mathrm{DP}: \big(r^k\big)^T M\, r^k < \tau\, \|Me\|_2, \qquad \mathrm{ME}: \big(r^k\big)^T M\big(r^k + r^{k+1}\big)/\|r_k\|_2 < \tau \|Me\|_2,$$

where $r^k = b - A\,x^k$ and τ is a suitable "safety factor" slightly larger than 1. We modified these rules in two ways, leading to the rules shown in Table 4.

For the case $M \neq I_m$, in AIR TOOLS, we followed the recommendation in [17] to replace $\|Me\|_2$ with the upper bound $\|M\|_2\|e\|_2$. Unfortunately, extensive experiments showed that this bound is always a very large overestimate and hence the iterations terminated much too early. Hence, we decided to ignore the matrix M on both sides of the inequality signs, which gives stopping rules that work much better in practice.

When the ME rules are formulated in the above forward-looking fashion, we always need to take one additional iteration because we need the residual r^{k+1}, associated with iteration vector x^{k+1}, to terminate at iteration k. This is not practical. Since the SIRT methods converge rather slowly, we have chosen instead to use the more practical backward-looking version with $r^k + r^{k+1}$ replaced by $r^{k-1} + r^k$. Also, we found that adding the factor 1/2 improved the performance.

When using these stopping rules, the user is assumed to know a good estimate δ of the norm $\|e\|_2$ of the noise, as well as a suitable value of the "safety factor" τ—and the user only needs to specify the product $\tau\delta$. To aid the user in choosing the τ parameter, we provide a function `train_dpme` that, given an iterative method, a test problem with exact data, and the parameter $\delta = \|e\|_2$, computes the value of τ that leads to the smallest reconstruction error (for the test problem) in the smallest number of iterations. This function implements the strategy from [17] and averages over a user-specified number of noise realizations.

2.3.2 *Implementation of the NCP rule*

The principle underlying this rule is to terminate the iterations when the residual vector r^k resembles white noise. Following [22] and [36], the NCP stopping rule implemented in AIR TOOLS was derived for the case where b represents a 1D signal (e.g., a time series). Let $\hat{r}^k$ denote the discrete Fourier transform of the residual r^k, and let $\overline{m}$ denote the largest integer such that $\overline{m} \leq m/2$. Then, using MATLAB's colon-notation, we define the normalized cumulative periodogram (NCP) for r^k as the vector v with elements

$$v\big(r^k\big)_i = \big\|\hat{r}^k(2:i{+}1)\big\|_2^2 / \big\|\hat{r}^k(2:\overline{m}{+}1)\big\|_2^2, \qquad i = 1, 2, \ldots, \overline{m}. \tag{10}$$

If r^k consists of white noise, then its power spectrum is flat and the elements of the corresponding $v(r^k)$ will approximately be equal to $i/\overline{m}$ for $i = 1, 2, \ldots, \overline{m}$. Hence, we terminate the iterations at that k for which $v(r^k)$ is closest to the vector $v_{\text{white}} = (1/\overline{m}, 2/\overline{m}, \ldots, 1)$.

The above approach for 1D signals is not strictly correct for CT problems where the right-hand side b corresponds to p individual projections—one for each projection angle in the measurements. Here, we describe a new variant of NCP for such problems. We assume that data are organized such that we can partition b and the residual r^k into p conforming sub-vectors,

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}, \qquad r^k = \begin{pmatrix} r_1^k \\ r_2^k \\ \vdots \\ r_p^k \end{pmatrix}, \tag{11}$$

with each sub-vector corresponding to a single projection. This is indeed the case for the 2D test problems generated by `paralleltomo`, `fancurvedtomo`, and `fanlineartomo`. Then we measure the kth residual's deviation from being white noise by averaging the sub-vectors' deviations from being white,

$$\Delta^k = \frac{1}{p} \sum_{\ell=1}^{p} \left\| v\left(r_\ell^k\right) - v_{\text{white}} \right\|_2, \tag{12}$$

and we want to terminate the iterations at the minimal Δ^k. The user must always specify whether the 1D or 2D version, listed in Table 4, should be used.

Our experiments show that Δ^k does not always vary smoothly with k; rather, it may have a zigzag behavior. A similar observation was made in [23] and [35] for the Monte Carlo GCV approach. Their solution, which we replicate in our software, is to apply a short filter that tracks the envelope of the NCP function Δ^k.

2.4 Illustration of the stopping rules

We illustrate the issues described above by means of the 2D X-ray CT problem generated by means of `paralleltomo` (see below), with the parameters

```
N = 50,    theta = 0:3:177,    p = 75
```

which produces a coefficient matrix of dimensions 4500×2500. We added white Gaussian noise with $\|e\|_2/\|\bar{b}\|_2 = 0.03$ and solved the problem with the Cimmino and Kaczmarz methods. The results are shown in Fig. 2.

– The top plots show the error histories, i.e., $\|\bar{x} - x^k\|_2/\|\bar{x}\|_2$ versus k; the minimum reconstruction error is marked by the bullets.
– The middle plots show the DP and ME functions $\|r^k\|_2$ and $1/2 \left(r^k\right)^T \left(r^{k-1} + r^k\right)/\|r^k\|_2$, respectively; their intersections with the horizontal line at $\delta = \|e\|_2$ (marked by the bullets) define the number of iterations (we used $\tau = 1$). The ME rule does not apply to the ART methods.

- The middle left plot also shows the function corresponding to the ME rule as originally proposed in [17]: $\left(r^k\right)^T M\left(r^{k-1}+r^k\right)/(\|M\|_2\,\|r^k\|_2)$. This function is strictly smaller than δ, showing that this version of the ME rule does not work for the present problem.
- The bottom plots show the behavior of the filtered NCP function Δ^k in both its 1D and 2D variants, as well as the un-filtered NCP function in its 2D variant. The latter shows the need for the filtering. For Cimmino, both variants happen to give approximately the same number of iterations, while the 1D variant stops the Kaczmarz iterations too early.

The behavior of the stopping rules observed here is quite general: DP and ME tend to give the same results, and they may terminate the iterations before the minimum reconstruction error is reached; NCP tends to terminate the iterations even earlier. But these early terminations are acceptable because the reconstruction error is not much larger than its minimum. (One could argue that they are desirable because they trade a slightly sub-optimal solution for a shorter computing time.)

To illustrate the robustness of the DP, ME, and NCP stopping rules when applied to the same test problem as above, we generated 500 instances of white Gaussian noise with $\|e\|_2/\|\bar{b}\|_2 = 0.03$. For each instance, we used Cimmino's method and computed the ratio $k_{\text{rule}}/k_{\text{opt}}$ between the number of iterations chosen by the stopping rule and the optimal number of iterations for which the error is minimum, as well as the ratio $\|\bar{x} - x^{k_{\text{rule}}}\|_2/\|\bar{x} - x^{k_{\text{opt}}}\|_2$. Ideally, both ratios should be close to 1. If $k_{\text{rule}}/k_{\text{opt}} < 1$, we stop too early; otherwise, we stop too late, and in both cases,

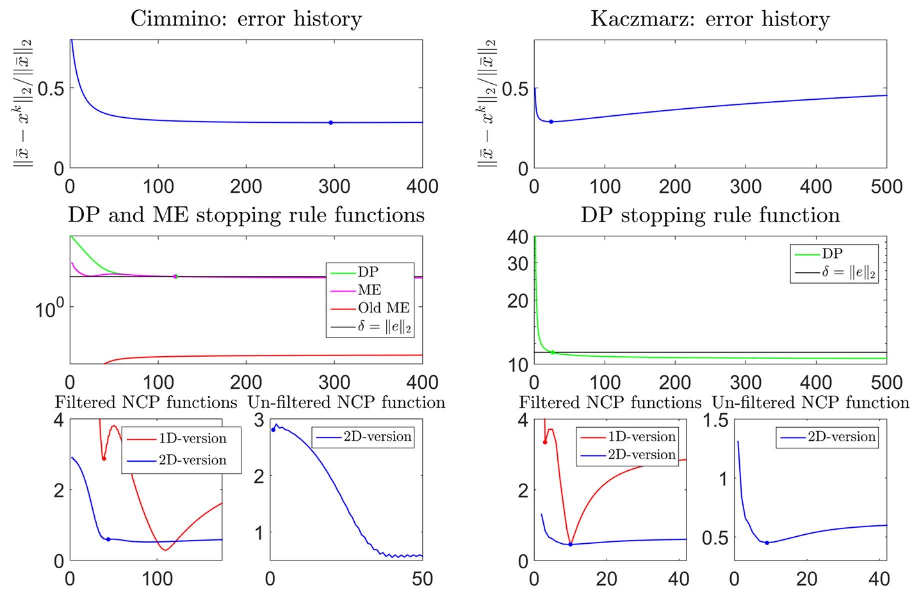


Fig. 2 Illustration of stopping rules for the Cimmino and Kaczmarz methods; see the text for details

$\|\bar{x} - x^{k_{\text{rule}}}\|_2 / \|\bar{x} - x^{k_{\text{opt}}}\|_2 > 1$. Moreover, if the latter ratio is large, it means that the stopping rule did not work well—the iterations stopped either too early or too late.

Scatter plots of all the pairs of these ratios are shown in Fig. 3, for two different choices of the parameter τ in DP and ME. The inserted plots show all the results, and we see that in most of the cases, the ratio $k_{\text{rule}}/k_{\text{opt}}$ is smaller than 1, i.e., we stop too early. The large plots zoom in on the interval $0 \leq k_{\text{rule}}/k_{\text{opt}} \leq 0.8$ where most of the data points are located. From these plots, we see that

– For all three stopping rules, when the iterations are stopped *too early*, then the ratio $\|\bar{x} - x^{k_{\text{rule}}}\|_2 / \|\bar{x} - x^{k_{\text{opt}}}\|_2$ can be as large as 1.4 or 1.8, for $\tau = 1.2$ and 1.3, respectively.

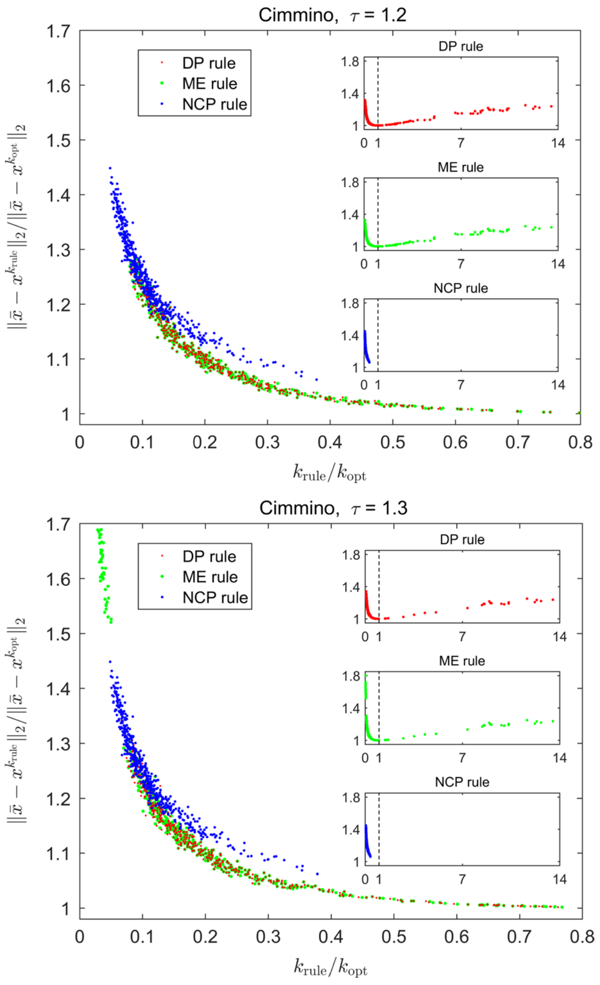


Fig. 3 Illustration of the robustness of the stopping rules for Cimmino's method; see the text for details. The top and bottom plots correspond to using $\tau = 1.2$ and 1.3, respectively, in the DP and ME stopping rules

- For $\tau = 1.3$, the ME rule occasionally terminates the iterations *much too early*, as indicated by the cluster of green points in the upper left corner.
- The NCP rule *consistently* stops the iterations *too early*, and there is no risk of taking too many iterations. On average, for $k_{\text{rule}}/k_{\text{opt}} < 1$, the obtained reconstruction error for the NCP rule is a bit larger than that for the DP and ME rules.
- For both the DP and ME rules, the iterations are occasionally stopped *too late*, especially for $\tau = 1.2$ (63 out of 500 cases); for $\tau = 1.3$, this happens less often (23 out of 500 cases).

In conclusion, the NCP stopping rule never takes too many iterations but the average reconstruction error is a bit larger than for the DP and ME rules. On the other hand, the latter rules may occasionally stop the iterations much too early or much too late—and they require that the noise level is known.

2.5 Choice of relaxation parameter

The relaxation parameter in (3), (5), and (7) is either a constant or it depends on the iterations. The user can specify a *constant* relaxation parameter ω; if this is not specified, then the default values mentioned above are used.

The choice of ω is important, in the sense that we want to compute a good reconstruction in as few iterations as possible. For the SIRT methods, the smallest reconstruction error is almost independent on ω [15], and this means that we prefer an ω towards its maximum allowed value (cf. Table 3)—although not too close to this value. For the ART methods, on the other hand, there is a more complicated relationship between ω and the smallest reconstruction error for this parameter. The dilemma is that a smaller ω (which gives a slower convergence) gives a smaller error—and hence we want to balance the number of iterations and the reconstruction error. Figure 1 in Section 2.2 illustrates these aspects, using the same test problem as in Section 2.4.

To aid the user in finding a good constant relaxation parameter ω, we provide a function `train_relaxpar` that, given a noisy test problem with a known exact solution, determines an optimal ω that leads to semi-convergence in the smallest number of iterations. See [24] for the details of this algorithm. The values of ω found by this function are also shown in Fig. 1.

For the SIRT methods, we can also use a relaxation parameter ω_k that depends on the iterations, and we provide two strategies for choosing ω_k. The *line search* method [12], [15] minimizes the error $\|D^{-1/2}(x^* - x^k)\|_2$ in each iteration; here, x^* is the solution to $A\,x = b$ which is assumed to be consistent. This leads to the choice

$$\omega_k = \frac{(r^k)^T M r^k}{\|D^{1/2} A^T M r^k\|_2^2}. \tag{13}$$

Since this strategy assumes a consistent system, it can lead to oscillations in the error history for non-consistent problems, cf. [15].

An alternative is to use a *diminishing step-size* strategy for ω_k in which the iterations "slow down" as they reach the point of semi-convergence. We implement the "Ψ-strategies" from [18] in which $\omega_0 = \omega_1 = \sqrt{2}/\rho$ and

$$\omega_k = \begin{cases} \frac{2\left(1-\xi_k\right)}{\varrho}, & \Psi_1\text{-strategy} \\ \frac{2\left(1-\xi_k\right)}{\varrho\left(1-\xi_k^k\right)^2}, & \Psi_2\text{-strategy} \end{cases} \qquad \text{for } k = 2, 3, \ldots \tag{14}$$

where $\varrho = \|A^T M A\|_2$ and ξ_k is the unique root in the interval (0, 1) of the polynomial $(2k-1)\xi^{k-1} - (\xi^{k-1} + \cdots + \xi + 1)$. Modified versions are obtained by scaling ω_k by 2 and 1.5, respectively. We refer to [18] for more details.

3 Test problems

We provide three types of test problems: 2D X-ray tomography, 2D spherical Radon tomography, and 2D seismic travel-time tomography. In all cases, we use a pixel basis; the image is $N \times N$ and it is represented by the vector $x \in \mathbb{R}^n$ with $n = N^2$. The user must specify N and parameters for the measurement geometry. We return the image $\bar{x}$, the corresponding data $b \in \mathbb{R}^m$, and the sparse matrix A that represents the underlying mathematical model: $b = A\,x$.

A new feature in AIR TOOLS II is that the test problem generators can return a *function handle* `@A` instead of the sparse matrix, such that the multiplications Ax and $A^T y$ are replaced by the calls `A(x,'notransp')` and `A(y,'transp')`. The updated versions of all the iterative methods accept such a function handle instead of the matrix.

For all test problems, it is possible to display the geometries in an animation by specifying an input parameter `isDisp`; we recommend to do this only for small problems. An additional new feature to illustrate the measurement geometries is by means of the function `show_tomo` which, given a test problem matrix or function handle, performs a loop where it displays the rows of the matrix reshaped into an $N \times N$ image.

3.1 X-ray CT problems

The underlying model in these test problems consists of straight X-rays that penetrate the object, after which we record the damping. According to Lambert-Beer's law [5, §2.3.1], and after taking the logarithm of the recorded data, the damping is given as a line integral along the X-ray of the object's attenuation coefficient. Discretization of the object then leads the following model, for the ith ray:

$$b_i = \sum_{j \in \mathcal{S}_i} L_{ij}\, x_j,$$

where $\mathcal{S}_i$ is the set of indices to those pixels that are penetrated by the ith ray, L_{ij} is the length of the ith ray through the jth pixel, and x_j is the attenuation coefficient in pixel j. Defining the elements of the sparse matrix A as

$$a_{ij} = \begin{cases} L_{ij}, & j \in \mathcal{S}_i \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

we arrive at the linear algebraic system $A\,x = b$. This classical discretization scheme is often referred to as the "line model" (or "Siddon's method," although his contribution [37] was a fast algorithm for implementing this scheme).

The detector has p pixels, and the assumption is that each detector pixel is hit by a single X-ray. Moreover, the source-detector pair is rotated around the object, and measurements are recorded for N_θ angles $\theta_1, \theta_2, \ldots, \theta_{N_\theta}$. Hence, the number of data is $m = p\,N_\theta$. The default values are $p = \texttt{round}(\sqrt{2}N)$ and $N_\theta = 180$. The measurement configurations are as follows—see Fig. 4 for illustrations of the three configurations:

– The object domain is the square $[-N/2\,,\ N/2] \times [-N/2\,,\ N/2]$.
– `paralleltomo`. The source is placed infinitely far from a flat detector with equal spacing between the pixels (which is a very good approximation to the situation at synchrotron facilities) and the p rays are therefore parallel. The distance between the first and the last ray is d, with default value $d = p - 1$, and the default projection angles are $0°, 1°, 2°, \ldots, 179°$.
– `fancurvedtomo`. The source is located at distance $R\,N$ from the object's center (default $R = 2$). The detector is curved such that there is an equal angular span between each ray, and the total angular span is d degrees; the default value of d is such that the first and last rays originating from $(0, RN)$ hit the two domain corners $(-N/2\,,\ N/2)$ and $(N/2\,,\ N/2)$. The default projection angles are $0°, 2°, 4°, \ldots, 358°$.
– `fanlineartomo` (new test problem). Again, the source is located at distance $R\,N$ from the object's center (default $R = 2$). The detector is linear such that

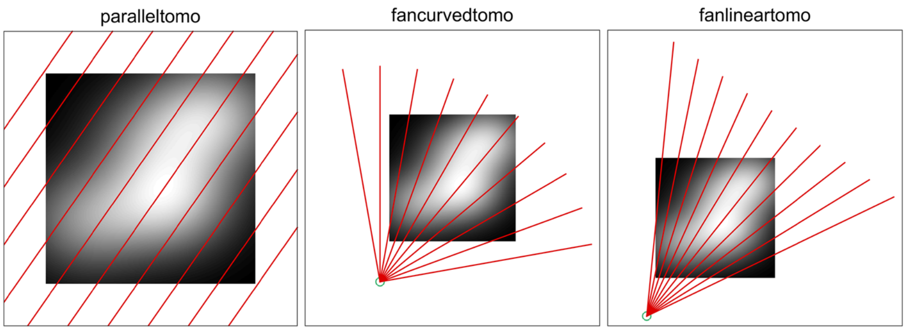


Fig. 4 Illustration of the measurement geometries in the three X-ray CT test problems. The parameters (which are not the default values) were chosen for illustration purposes

there is an uneven angular span between the rays; it is located at distance `sd` N from the source (default `sd` $= 3$) and the distance between the centers of the end-point pixels is `dw` N (default `dw` $= 2.5$). The default projection angles are $0°, 2°, 4°, \ldots, 358°$.

3.2 Spherical Radon transform tomography

The spherical Radon transform arises, e.g., in photo-acoustic tomography [31] where a laser pulse focused inside the object generates a thermoelastic expansion that produces a pressure wave which is measured by transducers outside the object. The data consist of integrals over spheres (or circles in the 2D case)—and hence the name "spherical means" is also used. The measurement configuration for the test problem `sphericaltomo` is as follows—see Fig. 5 for an illustration:

- The object domain is a square centered at the origin.
- The centers for the integration circles are placed on a circle just outside the image domain, and the user specifies the angles to these circle centers. The default angles are $[0°, 2°, 4°, \ldots, 358°]$.
- For each circle center, we integrate along a user-specified number of concentric circles with equidistant radii, using the periodic trapezoidal rule. The default number of circles is round$(\sqrt{2}N)$.

3.3 Seismic travel-time tomography

These test problems simulate geophysical problems where one records the travel time of seismic waves between sub-surface sources and detectors located either at or below the surface. The goal is to determine the sub-surface attenuation, or slowness, in the

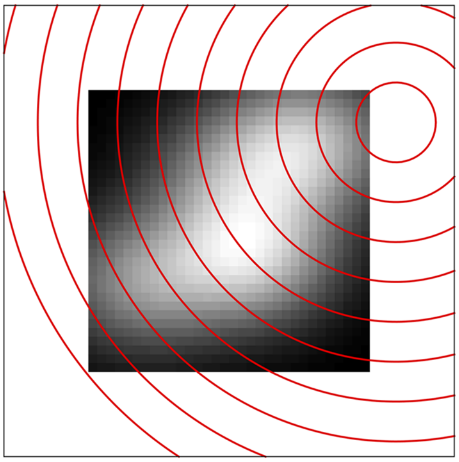


Fig. 5 Illustration of the measurement geometry in the spherical Radon transform test problems `sphericaltomo`. We show 10 concentric integration circles centered outside the image domain

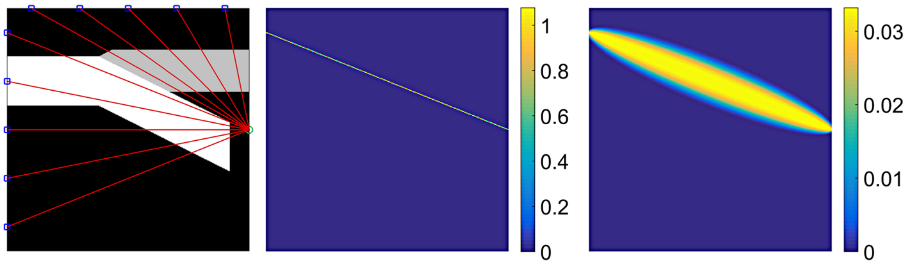


Fig. 6 Left: illustration of the measurement geometry in the two seismic travel-time test problems. Middle and right: a single row of A displayed as an image, showing a single ray in `seismictomo` and a single Fresnel zone in `seismicwavetomo`. The parameters (which are not the default values) were chosen for illustration purposes

domain represented by x. The measurement configurations are as follows—see Fig. 6 for illustrations:

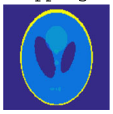
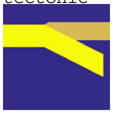
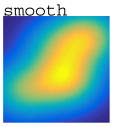
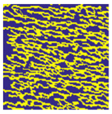
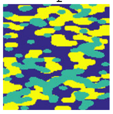
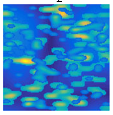
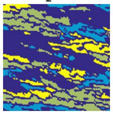
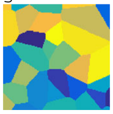
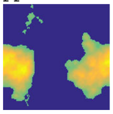
- The domain is the square and of size $[0, N] \times [0, N]$.
- There are s equally spaced sources located at the right side of the domain; the default is $s = N$.
- There are p receivers equally spaced on the top (the surface) and the left side of the domain; the default is $p = 2N$.
- `seismictomo`. The recorded travel time is modeled by a line integral of the attenuation coefficient along a straight line between the source and the detector, and when discretized, this leads to the same linear model $b = A\,x$ as before with the matrix elements given by (15).
- `seismicwavetomo` (new test problem). Here, the wave between the source and the receiver is assumed to travel within the first Fresnel zone [27] shaped as an ellipse with focal points at the source and the receiver. The width of the ellipse depends on the dominating frequency ν of the propagating wave (default $\nu = 10$); the smaller the ν, the wider the ellipse. The recorded travel time is modeled as an integral of the attenuation coefficient over the Fresnel zone. This also leads to a sparse matrix A but with more nonzeros than for `seismictomo`; the smaller the ν, the more nonzeros in A.

3.4 Collection of test phantoms

We provide a total of nine test images, also known as phantoms, as listed and shown in Table 5. In addition to the well-known Shepp-Logan medical phantom, we provide phantoms with various characteristics and features that make them suited for testing different reconstruction capabilities (such as edges and small details).

The phantom is always an $N \times N$ image with pixel values between 0 and 1, and the user must reshape this image to obtain the vector x. The user can specify a number of additional input parameters that define the appearance of the phantom. All phantoms except `shepplogan` are created by us.

Table 5 An overview of the nine test phantoms generated by the function `phantomgallery`

Name	Description	Parameters
`shepplogan` 	The Shepp-Logan "medical" phantom. Default in the X-ray CT and spherical Radon transform test problems	None
`tectonic` 	Two convergent tectonic plates creating a subduction zone. Default in the seismic test problems	None
`smooth` 	A smooth image consisting of a superposition of four Gaussian functions	`P1` = 1, 2, 3, 4 (default 4) defines four different images
`binary` 	A random image with binary pixel values arranged in domains dominated by horizontal structures	`P1` is the seed for the random number generator
`threephases` 	A random image with pixel values 0, 0.5, and 1 arranged in domains	`P1` controls the number of domains (default = 100) `P2` is the seed for the random number generator
`threephasessmooth` 	Similar to `threephases`, but the domains have smoothly varying pixel values and there is a smooth background	`P1` controls the number of domains (default = 100) `P2` controls the intensity variation within each domain (default = 1.8) `P3` is the seed for the random number generator
`fourphases` 	A random image similar to `binary` but with three phases separated by (thin) structures that form the fourth phase	`P1` is the seed for the random number generator
`grains` 	A random image with Voronoi cells, simulating grains in a crystalline material	`P1` is the number of cells (default = `round(3*sqrt(N))`) `P2` is the seed for the random number generator
`ppower` 	A random image with patterns of nonzero pixels [28], suited for sparse reconstructions	`P1` is the fraction of nonzero pixels, between 0 and 1 (default = 0.3) `P2` > 0 is the smoothness of the image (default = 2) `P3` is the seed for the random number generator

The images were generated with $N = 128$ and default parameters

4 Improving the implementation of AIR TOOLS II

The original software AIR TOOLS was not designed in a modular fashion, leading to a large amount of duplicated code. Several additional features have been added over the years, making it increasingly difficult to maintain and ensure correctness and consistency due to the non-modular design. Moreover, the original software was not designed with large-scale problems in mind; hence, the explicit use of the matrix A was required and the code was not optimized for performance. These reasons motivated a major re-design of the software, which has led to this improved implementation which incorporates a reorganizing of the code in a modular fashion with emphasis on computational speed and memory use and with expanded functionality.

One of the major new features of the software is the ability to use functions that compute matrix-vector products, instead of requiring the coefficient matrix A to be explicitly stored. This allows us to solve much bigger reconstruction problems—at the cost of somewhat larger execution times due to the computation of the matrix elements "on the fly," and we illustrate this feature in the next section. The following considerations are in order:

- Computation of the matrix norm $\|A^T M A\|_2$ and the spectral radius $\rho(D\,A^T M A)$, used for checking and/or selecting the constant relaxation parameter in the SIRT methods, is done by means of the MATLAB function `eigs` (MATLAB's `svds` does not accept a function handle as input). We set a fairly large accuracy threshold because we only need the result with a few significant digits.
- In doing this, we ensure that the computed results are fully reproducible. The MATLAB function `eigs` uses a random starting vector causing minor variations in the computed spectral radius. Therefore, we make the computation of the spectral radius and, thus, the relaxation parameter, deterministic by using fixed random number generating settings before calling `eigs`, and then restoring the previous settings afterwards.
- Computation of the row and column 2-norms must be done by extracting the rows or columns individually, by applying A or A^T to all the canonical unit vectors. This is time-consuming for large problems.
- Computation of the row and column 1-norms is simple when the matrix has nonnegative elements (making the SART method attractive for CT problems where this is always true). If $\mathbf{1}$ denotes the vector of all ones (of appropriate length), then the vectors $A\,\mathbf{1}$ and $A^T\mathbf{1}$ hold the 1-norms of the rows and columns, respectively.

Another major update consists of collecting all the iterative methods in three general functions `art`, `cart`, and `sirt`, and adding a small number of auxiliary functions (e.g., for checking the input parameters and imposing the stopping criteria). Previously, all the iterative methods were implemented in separate functions with similar structure; collecting them in three general functions with common auxiliary functions ensures consistency and makes it much easier to maintain and update the software (e.g., if more stopping rules are added). For backward consistency, we provide interface functions with all the old function names, such as `kaczmarz` and `cimmino`, that call these general functions.

An additional advantage of this design is that we can now allow the user to custom-define variants of the iterative methods. In particular, the user can specify the row or column ordering for the ART and CART methods, and specify the matrices D and M (either as matrices or vectors representing the diagonal of diagonal matrices) in the SIRT methods.

The original software included a possibility for continuing the iterations after they were terminated, by returning the necessary "internal" matrices and parameters. This feature was surprisingly difficult to use, so it was removed. We return relevant "internal" quantities such as D and M, as well as the constant relaxation parameter ω if this strategy is used.

5 Demonstrations of new features

Here, we present three examples that demonstrate some of the new features of the software. More examples can be found in the original paper [24], as well as in the demo scripts provided in the package.

5.1 Use of constraints

In this example, we show that the use of constraints can lead to an improved reconstruction, and we demonstrate how to specify these constraints in our software. We generate a noisy parallel-beam CT test problem and solve it by Cimmino's method with four instances of the constraints:

1. No constraints.
2. Nonnegativity constraints $\mathcal{C} = \mathbb{R}^n_+$.
3. Box constraints $\mathcal{C} = [0, 1]^n$,
4. Box constraints combined with tight constraints for those pixels in $\bar{x}$ whose values are known to be close to 0.3.

For each instance, we compute the 2-norm of the reconstruction error in two different ways: i) over all the pixels or ii) only in those pixels where $\bar{x}_i \neq 0.3$, giving the results:

Case	No constr.	$\mathcal{C} = \mathbb{R}^n_+$	$\mathcal{C} = [0, 1]^n$	Box + equality
i)	1.896	0.879	0.866	0.769
ii)	1.825	0.832	0.819	0.769

Clearly, each added constraint improves the reconstruction. In particular, note that the tight constraints not only affect the relevant pixels but also improve the reconstruction elsewhere. This example is available in the script `demo_constraints`.

```
N = 50;             % The image is N-by-N.
theta = 0:2:178;    % No. of used angles.
p = 75;             % No. of parallel rays.
eta = 0.02;         % Relative noise level.
k = 5000;           % Number of iterations.

% Create the noisy test problem.
[A,b_ex,x_ex] = paralleltomo(N,theta,p);
rng(0);
e = randn(size(b_ex));
b = b_ex + eta*norm(b_ex)*e/norm(e);

% Perform Cimmino without constraints.
Xcim = cimmino(A,b,k);

% Perform Cimmino iterations with nonnegativity constraints.
options.lbound = 0;
Xcimn = cimmino(A,b,k,[],options);

% Perform Cimmino iterations with box constraints.
options.lbound = 0;
options.ubound = 1;
Xcimb = cimmino(A,b,k,[],options);

% Determine the indices to those regions in the image that are known to
% have exact pixel values equal to 0.3, and enforce very tight bounds
% (simulation equality constraints) on these elements.
I = find(abs(x_ex-0.3)<1e-10);
L = zeros(N^2,1);  L(I) = 0.299;
U = ones(N^2,1);   U(I) = 0.301;
options.lbound = L;
options.ubound = U;
Xcime = cimmino(A,b,k,[],options);

% Show that enforcing constraints improves the reconstruction, by
% computing the errors in those pixels where the equality constraint
% is not active
z_ex = x_ex; z_ex(I) = [];
Zcim = Xcim; Zcim(I) = [];
Zcimn = Xcimn; Zcimn(I) = [];
Zcimb = Xcimb; Zcimb(I) = [];
Zcime = Xcime; Zcime(I) = [];
[norm(z_ex-Zcim),norm(z_ex-Zcimn),norm(z_ex-Zcimb),norm(z_ex-Zcime)]
```

5.2 Use of the custom SIRT function `sirt`

It is shown in [16] that a down-up "double sweep" of the symmetric Kaczmarz algorithm is mathematically identical to one iteration of the SIRT algorithm with $D = I$ and a symmetric M given by

$$M = (2\omega - 1)\,(\omega\varDelta + L)^{-T}\,\varDelta\,(\omega\varDelta + L)^{-1}, \tag{16}$$

where $\varDelta$ is diagonal and L is strictly lower triangular such that $L + \varDelta + L^T = AA^T$. To verify this experimentally, we generate a small test problem, run a few iterations of both methods, and compute the 2-norm of the difference between respective iteration vectors – see Fig. 7. To guarantee full rank of AA^T we ensure that the matrix A used here has no zero rows. This example is available in the script `demo_custom_sirt`.

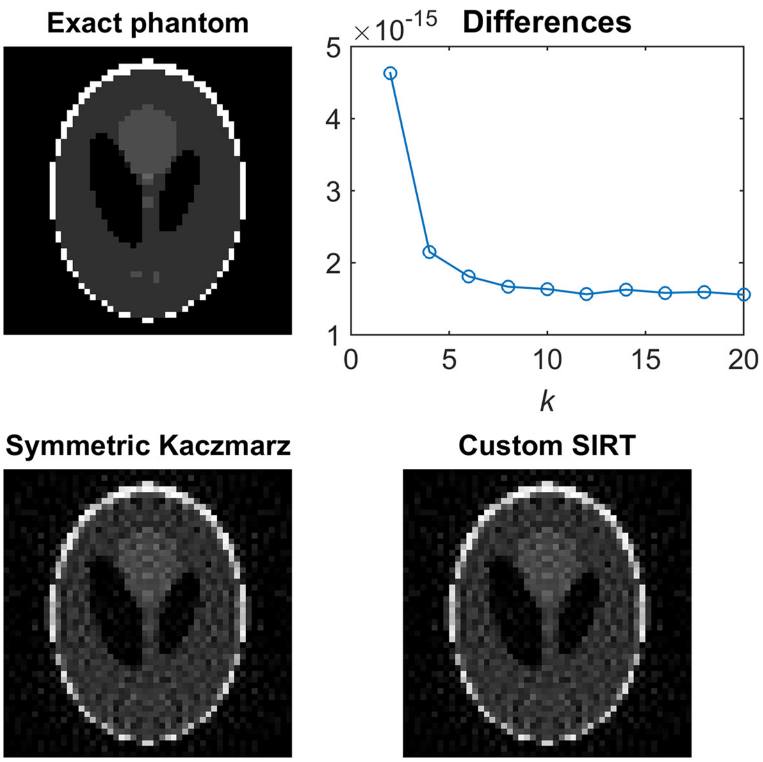


Fig. 7 Experimental verification that the symmetric Kaczmarz method is identical (within rounding errors) to a certain SIRT method with $D = I$ and M given by (16)

```
N = 50;              % The image is N-by-N.
theta = 0:5:175;     % No. of used angles.
p = 75;              % No. of parallel rays.
kmax = 20;           % Number of iterations.
omega = 1;           % Relaxation parameter.

% Create the test problem and purge zero rows from A.
[A,b,x] = paralleltomo(N,theta,p);
[A,b] = purge_rows(A,b);

% Perform the symmetric Kaczmarz iterations.
options.relaxpar = omega;
Xsym = art('symkaczmarz',A,b,2:2:kmax,[],options);

% Compute the "weight" matrix M that corresponds to symmetric Kaczmarz.
AAT = full(A*A');
L = tril(AAT,-1);
Delta = diag(diag(AAT));
M = (omega*Delta+L')\((2*omega-1)*Delta)/(omega*Delta+L);

% Set M and D fields (D is the identity) and run custom SIRT.
sirt_method.M = M;
Xsirt = sirt(sirt_method,A,b,1:1:kmax/2,[],options);
```

5.3 Function handles instead of matrices

The next example illustrates how to use function handles instead of matrices in the iterative methods, thus making it possible to handle larger reconstruction problems for which it is not feasible to store the entire matrix in memory. We show two different cases, both related to the `paralleltomo` test problem:

1. We create a function-handle "wrapper" `Afun_ps` to an existing A matrix, in order to illustrate the general principle. The matrix is generated by means of a standard call to the `paralleltomo` function, and the "wrapper" uses the function `afun_matrix` provided in this package.
2. We use a function handle `Afun_mf` to a fully matrix-free version, generated by calling `paralleltomo` with a sixth input parameter `0`. Note that all test problems come with this feature.

This example is available in the script `demo_matrixfree`.

```
% Create the matrix version of the test problem with no noise.
N = 50;  theta = 0:5:175;  p = 75;
[A,b,x_ex] = paralleltomo(N,theta,p);

% Pseudo matrix-free version by wrapping A matrix into function handle.
Afun_ps = @(XX,TT) afun_matrix(XX,TT,A);

% Fully matrix-free version by matrix-free mode of paralleltomo.
Afun_mf = paralleltomo(N,theta,p,[],[],0);  % Note the 6th parameter.

% Compute SART solutions using all three versions.
k = 50;
Xsart = sart(A,b,k);
Xsart_ps = sart(Afun_ps,b,k);
Xsart_mf = sart(Afun_mf,b,k);
```

6 Interface to other software: the ASTRA tomography toolbox

The possibility of specifying a function handle instead of a matrix as input to the iterative methods (cf. Section 5.3) allows AIR TOOLS II to *interface to other software*. In this way, we can use efficient third-party implementations of the matrix multiplications. The only requirement is that the user writes a function in the same format as `afun_matrix` to implement the necessary operations, namely, multiplication with the matrix and its transpose, and returning the size of the matrix.

This section illustrates the above functionality in a detailed example showing how to use the matrix-free GPU-accelerated forward and back projectors (corresponding to A and A^T) provided in the ASTRA tomography toolbox [34, 41]. This enables much faster execution than using the matrix-free methods of AIR TOOLS II. The corresponding code for the 2D problem is available in the demo `demo_astra_2d` and it uses the "wrapper" function `afun_astra_2d_gpu`. In order to run this example, the user needs to install ASTRA and have access to a suitable GPU, see www.astra-toolbox.com/ for details. All our experiments were done using version 1.8 of

ASTRA, which was the latest release at the time of writing. All our experiments were done using version 1.8 of ASTRA, which was the latest release at the time of writing. To be clear, apart from this example, no other functionality of AIR TOOLS II relies on external software.

ASTRA, among other things, provides an algebraic iterative reconstruction method called "SIRT" which is equivalent to our SART method—although with more restricted options, such as a fixed choice of $\omega_k = 1$ for the relaxation parameter, no stopping rules other than a maximum number of iterations, and no progress output during iterations. Another current limitation in ASTRA is that all data must fit into the GPU's RAM, thus limiting the size of data that can be handled. AIR TOOLS II provides the flexibility to go beyond this limitation.

The script `demo_astra_2d` uses `fanlineartomo` to generate a test problem and compute a reconstruction with the SART algorithm. We consider four cases:

(i) Using `sart` with the matrix A explicitly generated by `fanlineartomo`
(ii) Using `sart`'s matrix-free mode with the function handle from `fanlineartomo`
(iii) Using `sart` with ASTRA's GPU-accelerated forward and back projections
(iv) Using ASTRA's equivalent method called "SIRT" available in `SIRT_CUDA`

The resulting reconstructions along with computing times are shown in Fig. 8. Since we focus on demonstration of the software, rather than on achieving the best reconstruction, only 10 iterations are performed—which explains the blurred reconstructions.

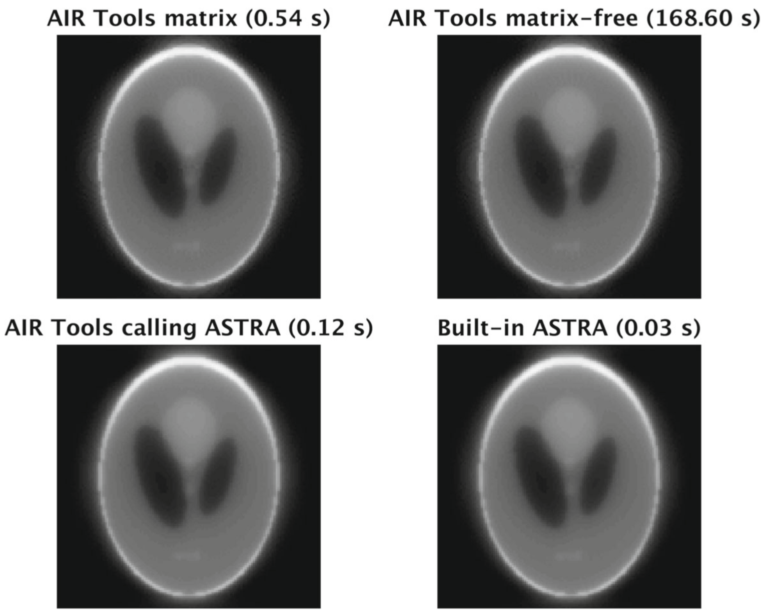

AIR Tools matrix (0.54 s)
AIR Tools matrix-free (168.60 s)
AIR Tools calling ASTRA (0.12 s)
Built-in ASTRA (0.03 s)

Fig. 8 The output from `demo_astra_2d` which demonstrates how to interface to the ASTRA tomography toolbox [41] from AIR TOOLS II. The figures show the reconstruction after 10 iterations, together with the corresponding computing times; see the text for details. All images are shown in gray-scale window [0, 0.6]

The four reconstructions are very similar in appearance, thus verifying that the interface to ASTRA works as expected, with only subtle differences ascribed to the different discretizations used in AIR TOOLS II and ASTRA to compute the matrix A. The computing time reported for case (i) does not include the initial time for setting up the matrix A. Case (ii) takes substantially longer, which is expected since all matrix elements are computed on the fly when needed, i.e., twice per iteration. The advantage of case (ii) is lower memory requirements, since A is never explicitly stored. If the problem considered is too large for the matrix to fit into system memory, one can thus trade memory usage for computing time by using the matrix-free mode in AIR TOOLS II. In case (iii), the GPU acceleration provided by ASTRA reduces the matrix-free computing time, thus offering both low memory usage and fast reconstruction. Even faster is case (iv) where ASTRA's built-in function "SIRT" avoids time-consuming data transfers to and from the GPU.

Using ASTRA's GPU-accelerated implementations of 3D tomography problems, it is now possible to handle large 3D CT problems in AIR TOOLS II via the function-handle interface. To illustrate this, we use the SOPHIABEADS DATASETS [10, 11] consisting of cone-beam micro-CT measurements of glass beads packed in a plastic container. Specifically, we use the dataset `SophiaBeads_1024_averaged` with 1024 projection angles in [0°, 360°]. The projection data for each angle are 2D images of 1564×1564 pixels. We use the ASTRA function `SIRT3D_CUDA` running on a high-end GPU (nVIDIA GeForce GTX TITAN X, 12-GB RAM), where we can compute up to 200 "slices" of the 3D reconstruction before exceeding the available GPU memory. We thus select the central 200 rows of each projection image and compute a 3D reconstruction of $1564 \times 1564 \times 200$ voxels. The reconstruction obtained after 50 iterations (reconstruction time 782 s) is shown in the left part of Fig. 9, which shows the three central orthogonal "slices." The reconstruction is somewhat blurry; it can be improved by running more iterations or by accelerating the iterations with

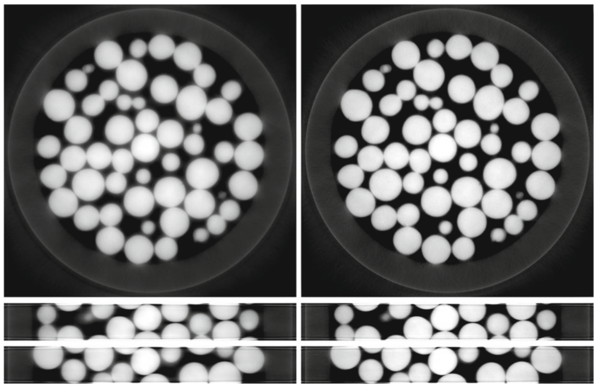


Fig. 9 Orthogonal "slices" of two 3D reconstructions after 50 iterations, using the SOPHIABEADS DATASETS. Top: the 1564×1564 horizontal slice; middle and bottom: the 200×1564 vertical slices. Left: reconstruction by ASTRA's `SIRT3D_CUDA` method with relaxation parameter $\omega_k = 1$. Right: reconstruction by AIR TOOLS's `sart` with relaxation parameter $\omega_k = 1.9$ interfacing to ASTRA's GPU-accelerated forward and back projectors. All images are shown in the gray-scale window [0, 0.004]

a larger relaxation parameter ω_k, but ASTRA's software does not allow other values than 1.

As an alternative, by interfacing to ASTRA's 3D forward and back projectors, we can compute a reconstruction using our `sart` function. We use the default relaxation parameter $\omega_k = 1.9$ and run again 50 iterations, which produces the reconstruction shown in the right part of Fig. 9. The beads are substantially sharper and they have a higher contrast to the background. The computing time for AIR TOOLS II was 1633 s, roughly only a factor 2 longer than that of ASTRA.

These simple examples were chosen to clearly illustrate how to increase the flexibility of AIR TOOLS II—here in conjunction with the ASTRA software for large-scale CT reconstruction. All features of AIR TOOLS II can be employed including stopping rules, relaxation parameter strategies, general constraints, storing of intermediate iteration vectors, a number of other iterative algorithms, and printing of progress per iteration using the `verbose` option. This provides a much more general set of iterative methods, while still utilizing ASTRA for GPU acceleration of the forward and back projection computations at only a moderately longer computation time.

7 Conclusion

We gave an overview of the MATLAB software package AIR TOOLS II with focus on the improvements, compared to the original package from 2012. Perhaps, the main improvement is the ability to use function handles instead of (sparse) matrices, thus allowing larger problems to be solved. Other improvements include a new modular design, better user interface, more flexible iterative methods, a new column-action method, new test problems, and new demo functions. Compared to dedicated tomography software, our package allows the user to easily experiment with a variety of well-documented algebraic iterative reconstruction methods in a flexible and uniform framework, and at the same time, our software can be used efficiently for real-data reconstruction problems.

Acknowledgements We thank Tommy Elfving for his continued help and support during this project.

References

1. Andersen, A.H., Kak, A.C.: Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm. Ultrason. Imaging **6**, 81–94 (1984)
2. Andersen, M.S., Hansen, P.C.: Generalized row-action methods for tomographic imaging. Numer. Algorithms **67**, 121–144 (2014). https://doi.org/10.1007/s11075-013-9778-8
3. Bertsekas, D.P.: Incremental proximal methods for large scale convex optimization. Math. Prog. **129**, 163–195 (2011)
4. Björck, Å., Elfving, T.: Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. BIT **19**, 145–163 (1979)
5. Buzug, T.M.: Computed Tomography. Springer, Berlin (2008)
6. Censor, Y., Elfving, T.: Block-iterative algorithms with diagonally scaled oblique projections for the linear feasibility problem. SIAM J. Matrix Anal. Appl. **24**, 40–58 (2002)

7. Censor, Y., Elfving, T., Herman, G.T., Nikazad, T.: On diagonally relaxed orthogonal projection methods. SIAM J. Sci. Comp. **30**, 473–504 (2007)
8. Censor, Y., Gordon, D., Gordon, R.: Component averaging: an efficient iterative parallel algorithm for large sparse unstructured problems. Parallel Comput. **27**, 777–808 (2001)
9. Cimmino, G.: Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. La Ricerca Scientifica, XVI, Series II, Anno IX **1**, 326–333 (1938)
10. Coban, S.B.: Sophiabeads datasets project codes, May 2017. Available online from: sophilyplum.github.io/sophiabeads-datasets
11. Coban, S.B., McDonald, S.A.: Sophiabeads dataset project, March 2015. Available online from: https://doi.org/10.5281/zenodo.16474
12. Dos Santos, I.T.: A parallel subgradient projections method for the convex feasibility problem. J. Comput. Appl. Math. **18**, 307–320 (1987)
13. Elfving, T., Hansen, P.C., Nikazad, T.: Convergence analysis for column-action methods in image reconstruction. Numerical Algorithms (2016). https://doi.org/10.1007/s11075-016-0176-x. Erratum (Fig. 3 was incorrect), https://doi.org/10.1007/s11075-016-0232-6
14. Elfving, T., Hansen, P.C., Nikazad, T.: Semi-convergence properties of Kaczmarz's method. Inverse Prob. **30** (2014). https://doi.org/10.1088/0266-5611/30/5/055007
15. Elfving, T., Hansen, P.C., Nikazad, T.: Semi-convergence and relaxation parameters for projected SIRT algorithms. SIAM J. Sci. Comput. **34**, A2000–A2017 (2012). https://doi.org/10.1137/110834640
16. Elfving, T., Nikazad, T.: Properties of a class of block-iterative methods. Inverse Prob. **25**, 115011 (2009). https://doi.org/10.1088/0266-5611/25/11/115011
17. Elfving, T., Nikazad, T.: Stopping rules for Landweber-type iteration. Inverse Prob. **23**, 1417–1432 (2007). https://doi.org/10.1088/0266-5611/23/4/004
18. Elfving, T., Nikazad, T., Hansen, P.C.: Semi-convergence and relaxation parameters for a class of SIRT algorithms. Electron. Trans. Numer. Anal. **37**, 321–336 (2010)
19. Gilbert, P.: Iterative methods for the three-dimensional reconstruction of an object from projections. J. Theor. Biol. **36**, 105–117 (1972)
20. Gordon, R., Bender, R., Herman, G.T.: Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. J. Theor. Biol. **29**, 471–481 (1970). https://doi.org/10.1016/0022-5193(70)90109-8
21. Hansen, P.C.: Discrete Inverse Problems: Insight and Algorithms. SIAM, Philadelphia (2010)
22. Hansen, P.C., Kilmer, M.E., Kjeldsen, R.H.: Exploiting residual information in the parameter choice for discrete ill-posed problems. BIT Numer. Math. **46**, 41–59 (2006)
23. Hansen, P.C., Nagy, J.G., Tigkos, K.: Rotational image deblurring with sparse matrices. BIT Numer. Math. **54**, 649–671 (2014). https://doi.org/10.1007/s10543-013-0464-y
24. Hansen, P.C., Saxild-Hansen, M.: AIR Tools—a MATLAB package of algebraic iterative reconstruction methods. J. Comp. Appl. Math. **236**, 2167–2178 (2012). https://doi.org/10.1016/j.cam.2011.09.039
25. Herman, G.T., Lent, A.: Iterative reconstruction algorithms. Comput. Biol. Med. **6**, 273–294 (1976)
26. Hämarik, U., Tautenhahn, U.: On the monotone error rule for parameter choice in iterative and continuous regularization methods. BIT **41**, 1029–1038 (2001)
27. Jensen, J.M., Jacobsen, B.H., Christensen-Dalsgaard, J.: Sensitivity kernels for time-distance inversion. Sol. Phys. **192**, 231–239 (2000)
28. Jørgensen, J.S., Sidky, E.Y., Hansen, P.C., Pan, X.: Empirical average-case relation between undersampling and sparsity in X-ray CT. Inverse Problems and Imaging **9**, 431–446 (2015). https://doi.org/10.3934/ipi.2015.9.431
29. Kaczmarz, S.: Angenäherte auflösung von Systemen linearer Gleichungen. Bulletin de l'Académie Polonaise des Sciences et Lettres **A35**, 355–357 (1937)
30. Klukowska, J., Davidi, R., Herman, G.T.: SNARK09—a software package for reconstruction of 2D images from 1D projections. Comput. Methods Programs Biomed. **110**, 424–440 (2013). https://doi.org/10.1016/j.cmpb.2013.01.003. The software is available from www.dig.cs.gc.cuny.edu/software/snark09/index.php
31. Kuchment, P., Kunyansky, L.: Mathematics of thermoacoustic tomography. Euro. J. Applied Math. **19**, 191–224 (2008)
32. Landweber, L.: An iteration formula for Fredholm integral of the first kind. Am. J. Math. **73**, 615–624 (1951)

33. Natterer, F.: The Mathematics of Computerized Tomography. Reprinted by SIAM, Philadelphia (2001)
34. Palenstijn, W.J., Batenburg, K.J., Sijbers, J.: Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs). J. Structural Biology **176**, 250–253 (2011)
35. Perry, K., Reeves, S.: A practical stopping rule for iterative signal restoration. IEEE Trans. Signal Proces. **42**, 1829–1833 (1994)
36. Rust, B.W., O'Leary, D.P.: Residual periodograms for choosing regularization parameters for ill-posed problems. Inverse Prob. **24** (2008). https://doi.org/10.1088/0266-5611/24/3/034005
37. Siddon, R.L.: Fast calculation of the exact radiological path for a three-dimensional CT array. Med. Phys. **12**, 252–255 (1985)
38. Strohmer, T., Vershynin, R.: A randomized Kaczmarz algorithm for linear systems with exponential convergence. J. Fourier Anal. Appl. **15**, 262–278 (2009)
39. Sørensen, H.H.B., Hansen, P.C.: Multicore performance of block algebraic iterative reconstruction methods. SIAM J. Sci. Comp. **36**, C524–C546 (2014)
40. TIGRE: Tomographic iterative GPU-based reconstruction toolbox, available from github.com/CERN/TIGRE
41. van Aarle, W., Palenstijn, W.J., Cant, J., Janssens, E., Bleichrodt, F., Dabravolski, A., De Beenhouwer, J., Batenburg, K.J., Sijbers, J.: Fast and flexible X-ray tomography using the ASTRA toolbox. Opt. Express **24**, 25129–25147 (2016). https://doi.org/10.1364/OE.24.025129. The software is available from www.astra-toolbox.com
42. van der Sluis, A., van der Vorst, H.A.: SIRT- And CG-type methods for the iterative solution of sparse linear least-squares problems. Linear Algebra Appl. **130**, 257–303 (1990)
43. Vogel, C.R.: Computational Methods for Inverse Problems. SIAM, Philadelphia (2002)
44. Watt, D.W.: Column-relaxed algebraic reconstruction algorithm for tomography with noisy data. Appl. Opt. **33**, 4420–4427 (1994)
45. Wright, S.J.: Coordinate descent algorithms. Math. Program., Ser. B **151**, 3–34 (2015)