arXiv:submit/5434755 [math.NA] 27 Feb 2024

TRIPs-Py: Techniques for Regularization of Inverse Problems in Python

Mirjeta Pasha * Silvia Gazzola † Connor Sanderford ‡
Ugochukwu O. Ugwu §

February 27, 2024

Abstract

Abstract: In this paper, we describe TRIPs-Py, a new Python package of linear discrete inverse problems solvers and test problems. The goal of the package is two-fold: 1) to provide tools for solving small and large-scale inverse problems, and 2) to introduce test problems arising from a wide range of applications. The solvers available in TRIPs-Py include direct regularization methods (such as truncated singular value decomposition and Tikhonov) and iterative regularization techniques (such as Krylov subspace methods and recent solvers for ℓ_p-ℓ_q formulations, which enforce sparse or edge-preserving solutions and handle different noise types). All our solvers have built-in strategies to define the regularization parameter(s). Some of the test problems in TRIPs-Py arise from simulated image deblurring and computerized tomography, while other test problems model realistic problems in dynamic computerized tomography. Numerical examples are included to illustrate the usage as well as the performance of the described methods on the provided test problems. To the best of our knowledge, TRIPs-Py is the first Python software package of this kind, which may serve both research and didactical purposes.

1 Introduction

Inverse problems arise whenever one wants to recover information about a hidden quantity from measurements acquired via a physical process (forward problem). In this paper, we are interested in linear discrete inverse problems, which can be formulated as linear systems of equations or linear least squares problems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \qquad \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a suitable discretization of the forward operator, $\mathbf{x} \in \mathbb{R}^n$ represents a quantity of interest, and $\mathbf{b} \in \mathbb{R}^m$ is the available data, which is typically corrupted by some unknown perturbations (noise) $\mathbf{e}$, i.e.,

$$\mathbf{b} = \mathbf{b}_{\text{true}} + \mathbf{e} = \mathbf{A}\mathbf{x}_{\text{true}} + \mathbf{e}\,. \tag{2}$$

To keep the following derivations simple, we assume $m \geq n$; extensions to the $m < n$ case are often straightforward. Many important applications, such as medical, seismic, and satellite imaging require the solution of inverse problems; see, for instance, [1, 2, 3]. Test problems in TRIPs-Py model deblurring (or deconvolution problems) and computerized tomography problems; while many realistic instances of these problems can be generated using the provided synthetic data, also instances of dynamic tomography problems that use real data are included. For this kind of problems, the matrix $\mathbf{A}$ is typically ill-conditioned, with singular values that gradually decay and cluster at zero. The implication is that the solution of (1) is very sensitive to the noise in $\mathbf{b}$ and some regularization should be applied to recover a meaningful approximation of $\mathbf{x}_{\text{true}}$. All the regularization methods considered in this paper and

*Department of Mathematics, Tufts University, Medford, MA, USA and Laboratory for Decision and Information System, Massachusetts Institute of Technology, Cambridge, MA, USA. (mpasha@mit.edu)
†Department of Mathematical Sciences, University of Bath, Bath, UK. (sg968@bath.ac.uk)
‡Department of Mathematical Sciences, Arizona State University, Tempe, AZ, USA. (csanderf@asu.edu)
§Department of Electrical and Computer Engineering, Tufts University, Medford, MA, USA.(Ugochukwu.Ugwu@tufts.edu)

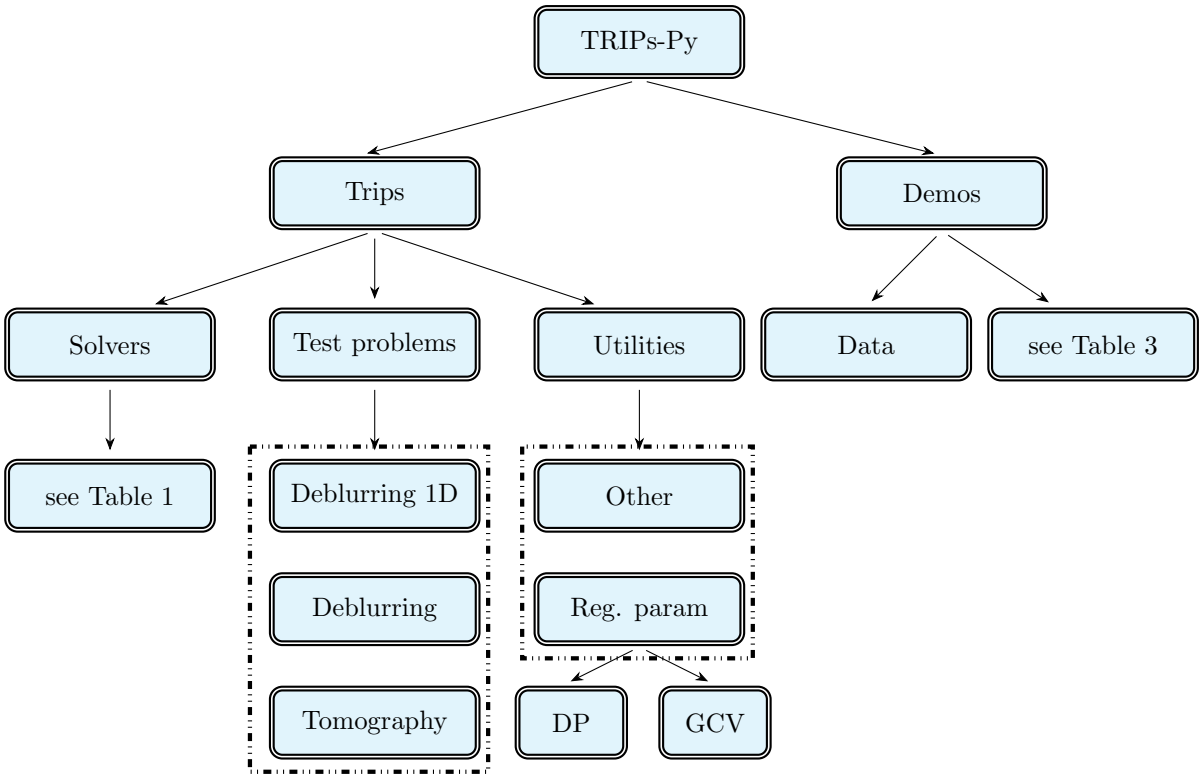


Figure 1: Overview of the TRIPs-Py's structure and contents. Most of the files available in the 'Utilities' directory are auxiliary functions that can be used by the TRIPs-Py solvers, such as functions to set the regularization operators, or to display data and reconstructions.

available within TRIPs-Py compute a regularized solution $\mathbf{x}_{\text{reg}}$ by (approximately) solving the following optimization problem

$$\mathbf{x}_{\text{reg}} = \underset{\mathbf{x}\in\mathcal{D}\subseteq\mathbb{R}^n}{\arg\min}\, \mathcal{F}(\mathbf{x}) + \alpha\mathcal{R}(\mathbf{x}), \quad \alpha > 0, \tag{3}$$

where $\mathcal{F}(\mathbf{x})$ is a *fit-to-data term* that typically involves the matrix $\mathbf{A}$ (or a modification thereof) and the data $\mathbf{b}$, α is a *regularization parameter*, $\mathcal{R}$ is a *regularization term*, and $\mathcal{D}$ is a set of *constraints*. Most of the methods in this paper take $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_p^p$, with $p > 0$, and $\mathcal{R}(\mathbf{x}) = \|\Psi\mathbf{x}\|_q^q$, with $\Psi \in \mathbb{R}^{k\times n}$ and $q > 0$. The choice of p is dictated by the distribution of the noise $\mathbf{e}$ (e.g., $p = 2$ for Gaussian white noise, $p = 1$ for impulse noise), while the choices of q and of the regularization matrix Ψ are dictated by prior information available on $\mathbf{x}_{\text{true}}$ (e.g., if the gradient of $\mathbf{x}_{\text{true}}$ is smooth, one typically takes $q = 2$ and $\Psi = \nabla$, i.e., a discretization of the gradient operator; if $\mathbf{x}_{\text{true}}$ is sparse one takes Ψ as the identity and $0 < q \leq 1$). Approaches for solving (3) depend on the particular choices of p, q, Ψ and the features of the problem (e.g., if $\mathbf{A}$ is small or large scale). A summary of the possible choices of the functionals $\mathcal{F}$ and $\mathcal{R}$ supported within TRIPs-Py is provided in Table 1, and more details are provided in Section 2. Within the TRIPs-Py solvers, the set of constraints $\mathcal{D}$ is taken to be the whole domain $\mathbb{R}^n$ for small-scale problems, and an appropriate linear subspace of $\mathbb{R}^n$ for large-scale problems; choices for the latter are detailed in Section 2.2.

TRIPs-Py serves two interrelated purposes: 1) to provide model implementations of solvers for a variety of both small and large-scale linear inverse problems (including some recent methods that are not available in other packages), and 2) to provide a range of test problems (dealing with both synthetic and real data) for the users to test the TRIPs-Py solvers or, possibly, their own solvers. TRIPs-Py is an open source package that is available through GitHub at `https://github.com/trips-py/trips-py`, where the users can also find installation instructions and requirements. Figure 1 shows an overview of the TRIPs-Py structure and its modules.

When designing TRIPs-Py we aimed to create Python software that is easy to use, with calls to all the solvers that are very basic and similar. More precisely, default values are provided for all the options and parameters needed by the solvers (including automatic strategies to choose the regularization parameter in (3) or the stopping iteration for iterative solvers); however, experienced users can easily set such parameters by themselves. The test problems in TRIPs-Py include 1D and 2D deconvolution problems

(deblurring), and a variety of computed tomography problems that employ both synthetic and real data. All the test problems are generated with very similar instructions, and default values for the test problem generators are provided for users who are not familiar with the associated applications; as for the solvers, experienced users can easily set such problem-specific parameters by themselves.

Although other packages are already available in MATLAB and Python for solving, and numerically experimenting with, inverse problems (and TRIPs-Py shares some features with them), some specifics make TRIPs-Py unique. For instance, solely the solvers and test problems for small-scale inverse problems in TRIPs-Py are closely modeled on the ones available in Regularization Tools [4], a pioneering MATLAB package that has proven to be very popular with the wider community of researchers in algorithms for linear inverse problems since the Nineties. TRIPs-Py shares many design objectives, solvers and test problems with IR Tools [5], a recent MATLAB package of iterative regularization methods and test problems for large-scale linear inverse problems. However, when compared to IR Tools, TRIPs-Py also features the first publicly available implementation of some recent methods for ℓ_p-ℓ_q regularization and some test problems in TRIPs-Py also employ real data. Other popular MATLAB toolboxes such as Restore Tools [6], AIR Tools II [7]) and TIGRE [8] focus only on specific applications (the former is for image deblurring, the latter for computerized tomography), while TRIPs-Py encompasses a range of small and large-scale linear inverse problems. Nowadays, as Python is very popular with researchers both inside and outside of academia, and students at universities are increasingly exposed to Python as a programming language in their courses, we envision that TRIPs-Py could be useful for the community of users of these MATLAB toolboxes that may have to switch to Python for collaborating on research projects and for didactical purposes. Since many TRIPs-Py functionalities are similar to the ones underlying Regularization Tools and IR Tools, we are confident that such users will find transitioning to Python through TRIPs-Py natural. Even in the setting of the many powerful and popular Python packages for the solution of inverse problems, we believe that TRIPs-Py provides some valuable additions, for a variety of reasons. First, as mentioned above, TRIPs-Py's test problems model diverse applications, while many Python packages for inverse problems are focused on a particular application. This differentiates TRIPs-Py from recent packages like CIL [9], ASTRA [10], and the Python version of TIGRE, which all target computerized tomography. Second, TRIPs-Py's solvers for large-scale problems are based on standard or generalized Krylov subspace methods, with some of the former and the latter not being available anywhere else or being used to solve non-convex non-smooth instances of problem (3). This distinguishes TRIPs-Py from other general-purpose libraries for inverse problems like ODL [11], whose majority of solvers are based on optimization methods such as proximal gradient algorithms, primal-dual hybrid gradient algorithms, and ADMM. We anticipate that the users of these Python packages would consider using some of the TRIPs-Py solvers to tackle their applications, and to compare available or newly developed solvers with the ones in TRIPs-Py.

The remaining part of this paper is organized as follows: Section 2 gives an overview of the solvers available in TRIPs-Py, starting with filtering methods for small-scale problems and including many projection methods for large-scale, smooth and non-smooth regularized problems. Section 3 gives an overview of the test problems available in TRIPs-Py, together with some illustrations of the usage of solvers and test problems. Conclusions and future directions are discussed in Section 4.

2 Overview of the TRIPs-Py solvers

We start this section by discussing the TRIPs-Py solvers for regularization methods expressed in the 2-norm, i.e., to solve problem (3) with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ and $\mathcal{R}(\mathbf{x}) = \|\Psi\mathbf{x}\|_2^2$. This is the main part of the section, and most of the TRIPs-Py solvers are tailored to this case. The last part of this section describes a solver that can be employed for regularization methods expressed in the ℓ_p-ℓ_q norm, i.e., to solve problem (3) with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_p^p$ and $\mathcal{R}(\mathbf{x}) = \|\Psi\mathbf{x}\|_q^q$, $p, q > 0$. An overview of the solvers available within TRIPs-Py is given in Table 1. We will not give too many details about the methods in this section, but rather provide extensive references for the reader. All the solvers in TRIPs-Py can be called in a consistent way that, at the very least, should include the forward operator (which could be a matrix or a function that acts on vectors) and the measured data. Additional optional inputs, such as the exact solution $\mathbf{x}_{\text{true}}$ for synthetic test problems, the maximum number of iterations to be performed and information about the stopping criterion for iterative solvers, or other specific features about specific methods, can be included. Most of these inputs are otherwise assigned default values. All functions return the computed approximation of the solution of the inverse problem, together with a dictionary that contains additional information about the solver and typically depends on the additional optional inputs assigned when calling the function.

Table 1: List of the solvers available in TRIPs-Py.

Solver	**Description**	**Specifics of problem** (3)		**Ref.**
		$\mathcal{F}(\mathbf{x})$	$\mathcal{R}(\mathbf{x})$	
`TSVD`	Truncated SVD	$\lVert\mathbf{A}_h\mathbf{x}-\mathbf{b}\rVert_2^2$	0	[12]
`TGSVD`	Truncated GSVD	$\lVert(\mathbf{A}\Psi_{\mathbf{A}}^{\dagger})_h\mathbf{x}-\mathbf{b}\rVert_2^2$	0	[13]
`Tikhonov`	Tikhonov regularization	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\Psi\mathbf{x}\rVert_2^2$	[12]
`CGLS`	Conjugate Gradient Least Squares	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	0	[14]
`GMRES`	Generalized Minimal Residual	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$, $\mathbf{A}\in\mathbb{R}^{n\times n}$	0	[14]
`Hybrid_LSQR`	Hybrid LSQR	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\mathbf{x}\rVert_2^2$	[15]
`Hybrid_GMRES`	Hybrid GMRES	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$, $\mathbf{A}\in\mathbb{R}^{n\times n}$	$\lVert\mathbf{x}\rVert_2^2$	[15]
`GK_Tikhonov`	Golub-Kahan-Tikhonov	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\mathbf{x}\rVert_2^2$	[16]
`A_Tikhonov`	Arnoldi-Tikhonov	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\mathbf{x}\rVert_2^2$	[17]
`GKS`	Generalized Krylov Subspace (GKS)	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\Psi\mathbf{x}\rVert_2^2$	[18]
`MMGKS`	Majorization minimization with GKS	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_p^p$	$\lVert\Psi\mathbf{x}\rVert_q^q$	[19]
`AnisoTV`	Anisotropic Total Variation	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\nabla\mathbf{x}\rVert_1$	[20]
`IsoTV`	Isotropic Total Variation	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\mathrm{TV}(\mathbf{x})$	[20]
`GS`	Group Sparsity	$\lVert\mathbf{A}\mathbf{x}-\mathbf{b}\rVert_2^2$	$\lVert\Psi\mathbf{x}\rVert_{2,1}$	[20]

We first survey methods that are suited for small-scale problems, followed by methods that are suited for large-scale problems; both the cases $\Psi = \mathbf{I}$ and $\Psi \neq \mathbf{I}$ will be covered.

2.1 Direct methods for small-scale problems

When problem (1) is small scale, solvers for problems (3) expressed in the 2-norm with $\mathcal{R}(\mathbf{x}) = \lVert\mathbf{x}\rVert_2^2$ typically rely on the Singular Value Decomposition (SVD) of $\mathbf{A}$, i.e.,

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T,$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are the orthogonal matrices of the left and right singular vectors, respectively, and $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ is the matrix formed by the diagonal matrix of the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ on top, and an $(m-n)\times n$ matrix of zeros at the bottom. The SVD of $\mathbf{A}$ is also a useful tool to analyze the ill-posedness of problem (1). While the SVD exists for every matrix, algorithms for its computation have a computational cost of order $O(mn^2)$, and are therefore prohibitive for large-scale problems; see [21, §8.6] for more details.

When $\mathcal{R}(\mathbf{x}) = \lVert\Psi\mathbf{x}\rVert_2^2$, with $\mathbf{I} \neq \Psi \in \mathbb{R}^{k\times n}$, problems (3) are naturally handled by considering the generalized singular value decomposition (GSVD) of the matrix pair $(\mathbf{A}, \Psi)$. Assume that $m \geq n \geq k$, that $\mathrm{rank}(\Psi) = k$ and that the null spaces of $\mathbf{A}$ and Ψ intersect trivially. Then the GSVD of $(\mathbf{A},\Psi)$ is given by

$$\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\Sigma}\mathbf{Y}^T, \qquad \Psi = \widetilde{\mathbf{V}}\widetilde{\Lambda}\mathbf{Y}^T, \tag{4}$$

where $\widetilde{\mathbf{U}} \in \mathbb{R}^{m\times n}$ and $\widetilde{\mathbf{V}} \in \mathbb{R}^{k\times n}$ have orthonormal columns, $\mathbf{Y} \in \mathbb{R}^{n\times n}$ is nonsingular; $\widetilde{\Sigma} \in \mathbb{R}^{n\times n}$ is the diagonal matrix with diagonal entries $0 \leq \widetilde{\sigma}_1 \leq \cdots \leq \widetilde{\sigma}_n \leq 1$, and $\widetilde{\Lambda} \in \mathbb{R}^{k\times n}$ is the matrix formed by the diagonal matrix of the values $1 \geq \widetilde{\lambda}_1 \geq \cdots \geq \widetilde{\lambda}_k \geq 0$ on the left and $k \times (n-k)$ matrix of zeros on the right. The diagonal entries of $\widetilde{\Sigma}$ and $\widetilde{\Lambda}$ are such that, for $1 \leq i \leq k$, $\widetilde{\sigma}_i^2 + \widetilde{\lambda}_i^2 = 1$; the quantities $\widetilde{\sigma}_i/\widetilde{\lambda}_i$ are commonly referred to as the *generalized singular values* of $(\mathbf{A}, \Psi)$. Similarly to the SVD of $\mathbf{A}$, the cost of computing the GSVD of $(\mathbf{A}, \Psi)$ is prohibitive for large-scale problems, unless some structure of $\mathbf{A}$ and Ψ can be exploited; see [21, §6.1.6 and §8.7.4] for properties of the GSVD and its computation.

Next, we describe three TRIPs-Py basic regularization methods that are directly based on the SVD of $\mathbf{A}$ or the GSVD of $(\mathbf{A}, \Psi)$. Such methods are specific instances of the general class of (G)SVD filtering methods, whose solutions $\mathbf{x}_\mu$ can be expressed as

$$\mathbf{x}_\mu = \sum_{i=1}^{n} \phi_i(\mu)\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i \quad (\text{for } \Psi = \mathbf{I}) \quad \text{and} \quad \mathbf{x}_\mu = \sum_{i=1}^{k} \phi_i(\mu)\frac{\widetilde{\mathbf{u}}_i^T\mathbf{b}}{\widetilde{\sigma}_i}\widetilde{\mathbf{y}}_i + \sum_{i=k+1}^{n} (\widetilde{\mathbf{u}}_i^T\mathbf{b})\widetilde{\mathbf{y}}_i \quad (\text{for } \Psi \neq \mathbf{I}), \tag{5}$$

where $\widetilde{\mathbf{y}}_i$, $i = 1, \ldots, n$ are the columns of $\mathbf{Y}^{-T}$. The scalars $\phi_i(\mu)$, $0 \leq \phi_i(\mu) \leq 1$, appearing in the above sums are called *filter factors*. The functional expressions of $\phi_i(\mu)$ determine different filtering methods, which all depend on the parameter μ. The basic principles underlying filtering methods can be understood referring to the so-called 'discrete Picard condition', which offers insight into the relative behavior of the magnitude of $\mathbf{u}_i^T\mathbf{b}$ and σ_i (for $\Psi = \mathbf{I}$), and $\widetilde{\mathbf{u}}_i^T\mathbf{b}$ and $\widetilde{\sigma}_i$ (for $\Psi \neq \mathbf{I}$), which appear in the expression of the filtered solutions. Namely, assuming that the quantities $|\mathbf{u}_i^T\mathbf{b}_{\text{true}}|$ and σ_i, and $|\widetilde{\mathbf{u}}_i^T\mathbf{b}_{\text{true}}|$ and $\widetilde{\sigma}_i$, decay at the same rate, then the noise in $\mathbf{b}$ dominates the unregularized solution for small σ_i's, and $\widetilde{\sigma}_i$'s. Therefore, filtering methods successfully compute regularized solutions when the filter factors $\phi_i(\mu)$ are close to 0 for small σ_i's and $\widetilde{\sigma}_i$'s, and close to 1 for large σ_i's and $\widetilde{\sigma}_i$'s (these are the meaningful components of the solution that we wish to retain). Having too many filter factors close to 1 results in under-regularized solutions (with $\phi_i(\mu) = 1$ for every i resulting in the (unregularized) solution of problem (1)); having too many filter factors close to 0 results in over-regularized solutions. The second sum in the second equality in equation (5) expresses the components of $\mathbf{x}_\mu$ in the null space of Ψ, which are unaffected by regularization. Many strategies for choosing the regularization parameter μ are available in the literature, and Section 2.3 provides more details about the ones implemented in TRIPs-Py. In the following, we specify how the three filtering methods in TRIPs-Py relate to the general framework (5), specifying an expression for the filter factors $\phi_i(\mu)$.

Truncanted SVD (TSVD) The truncated SVD (TSVD) method regularizes (1) by computing

$$\mathbf{x}_h = \sum_{i=1}^{h} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i\,, \quad h \in \{1, 2, \ldots, n\}\,. \tag{6}$$

TSVD is clearly a filtering method, with filter factors $\phi_i(h) = 1$ for $1 \leq i \leq h$ and $\phi_i(h) = 0$ otherwise. Therefore, the integer h plays the role of the regularization parameter, which can be set by applying the Generalized Cross-Validation (GCV) method or the discrepancy principle (if an estimate of the noise magnitude $\|\mathbf{e}\|_2$ is provided); see Section 2.3 for more details. Let $\mathbf{A}_h$ denote the best rank-h approximation of $\mathbf{A}$ in the 2-norm, i.e.,

$$\mathbf{A}_h = \mathbf{U}_h\Sigma_h\mathbf{V}_h^T,$$

where $\mathbf{U}_h$ and $\mathbf{V}_h$ are obtained by taking the first h columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, and Σ_h is the diagonal matrix with the first h singular values on its main diagonal. Then $\mathbf{x}_h$ can be regarded as the solution of the following variational problem

$$\mathbf{x}_h = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}_h\mathbf{x} - \mathbf{b}\|_2^2,$$

which belongs to the framework (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}_h\mathbf{x} - \mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = 0$, $\mathcal{D} = \mathbb{R}^n$.

Tikhonov regularization Tikhonov regularization replaces (1) with the problem of computing

$$\mathbf{x}_\alpha = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \alpha\|\Psi\mathbf{x}\|_2^2, \quad \text{where } \Psi \in \mathbb{R}^{k\times n}\,. \tag{7}$$

When $\Psi = \mathbf{I}$, problem (7) is said to be in the *standard form*, and when $\Psi \neq \mathbf{I}$ it is said to be the *general form*. Problem (7) clearly belongs to the framework (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = \|\Psi\mathbf{x}\|_2^2$, $\mathcal{D} = \mathbb{R}^n$. Problem (7) can be equivalently expressed as a damped least squares problem, with associated normal equations

$$(\mathbf{A}^T\mathbf{A} + \alpha\Psi^T\Psi)\mathbf{x}_\alpha = \mathbf{A}^T\mathbf{b}. \tag{8}$$

If the null spaces of $\mathbf{A}$ and Ψ intersect trivially, the Tikhonov regularized solution $\mathbf{x}_\alpha$ is unique. By plugging the SVD of $\mathbf{A}$ (if $\Psi = \mathbf{I}$) or the GSVD of $(\mathbf{A}, \Psi)$ (if $\Psi \neq \mathbf{I}$) into the above equation, one can see that $\mathbf{x}_\alpha$ can be expressed in the framework of (5) with

$$\phi_i(\alpha) = \frac{\sigma_i^2}{\sigma_i^2 + \alpha} \quad (\text{for } \Psi = \mathbf{I}) \quad \text{and} \quad \phi_i(\alpha) = \frac{(\widetilde{\sigma}_i/\widetilde{\lambda}_i)^2}{(\widetilde{\sigma}_i/\widetilde{\lambda}_i)^2 + \alpha} \quad (\text{for } \Psi \neq \mathbf{I})\,.$$

From the above expression and looking at (7), it is clear that more regularization is imposed for larger values of α, as more weight is put on the regularization term and more filter factors approach to 1 (depending on the location of α within the range of the singular values of $\mathbf{A}$ or generalized singular values of $(\mathbf{A}, \Psi)$). Conversely, smaller values of α lead to under-regularized solutions. When $\Psi \neq \mathbf{I}$ has a

nontrivial null space, it is important to stress (again, from the second equation in (5) or from (7)) that vectors in the null space of Ψ are unaffected by regularization. The regularization parameter α can be set by applying the GCV method or the discrepancy principle (if an estimate of the noise magnitude $\|\mathbf{e}\|_2$ is provided); see Section 2.3 for more details. Finally, we remark that any Tikhonov regularized problem in general form can be equivalently transformed into a Tikhonov regularized problem in standard form. The specific transformation depends on the properties of Ψ. Generically, one expresses the equivalent standard form solution as

$$\mathbf{x}_\alpha = \Psi_{\mathbf{A}}^\dagger \mathbf{z}_\alpha + \bar{\mathbf{x}}_0, \quad \text{where} \quad \mathbf{z}_\alpha = \arg\min_{\mathbf{z}\in\mathbb{R}^k} \|\mathbf{A}\Psi_{\mathbf{A}}^\dagger \mathbf{z} - \bar{\mathbf{b}}\|_2^2 + \alpha\|\mathbf{z}\|_2^2. \tag{9}$$

In the above expression, $\Psi_{\mathbf{A}}^\dagger = (\mathbf{I}-(\mathbf{A}(\mathbf{I}-\Psi^\dagger\Psi))^\dagger\mathbf{A})\Psi^\dagger$ denotes the $\mathbf{A}$-weighted generalized pseudoinverse $\Psi_{\mathbf{A}}^\dagger$ of the operator Ψ, $\bar{\mathbf{x}}_0$ denotes the component of $\mathbf{x}_\alpha$ in the null space of Ψ and $\bar{\mathbf{b}} = \mathbf{b} - \mathbf{A}\bar{\mathbf{x}}_0$; see [22] for more details.

Truncated GSVD (TGSVD) The truncated GSVD (TGSVD) method regularizes (1) by computing

$$\mathbf{x}_h = \sum_{i=0}^{h-1} \frac{\widetilde{\mathbf{u}}_{k-i}^T \mathbf{b}}{\widetilde{\sigma}_{k-i}} \widetilde{\mathbf{y}}_{k-i} + \sum_{i=k+1}^{n} (\widetilde{\mathbf{u}}_i^T \mathbf{b})\widetilde{\mathbf{y}}_i\,, \quad h \in \{1, 2, \ldots, k\}\,. \tag{10}$$

TGSVD is a filtering method that can be expressed as in (5) (second equation), with filter factors $\phi_i(h) = 1$ for $k - h + 1 \le i \le k$ and $\phi_i(h) = 0$ otherwise. The TGSVD solution can be linked to both TSVD and Tikhonov regularization in general form, in that the TGSVD solution can also be expressed as

$$\mathbf{x}_h = \Psi_{\mathbf{A}}^\dagger \arg\min_{\mathbf{z}\in\mathbb{R}^k} \|(\mathbf{A}\Psi_{\mathbf{A}}^\dagger)_h \mathbf{z} - \mathbf{b}\|_2^2 + \bar{\mathbf{x}}_0\,, \tag{11}$$

where $\Psi_{\mathbf{A}}^\dagger$ and $\bar{\mathbf{x}}_0$ are as in equation (9), and $(\mathbf{A}\Psi_{\mathbf{A}}^\dagger)_h$ is the optimal rank-h approximation of $\mathbf{A}\Psi_{\mathbf{A}}^\dagger$ in the 2-norm, which can be either computed using the SVD of $\mathbf{A}\Psi_{\mathbf{A}}^\dagger$ or the GSVD of $(\mathbf{A}, \Psi_{\mathbf{A}})$. The variational problem (11) belongs to the framework (3), as it is essentially a transformed least squares problem. As for TSVD, the truncation parameter h plays the role of the regularization parameter, and similar strategies are used to automatically set it.

2.2 Projection methods for large-scale problems

The methods described in the previous section can be properly applied only when it is feasible to compute some factorizations (e.g., the SVD) of the coefficient matrix $\mathbf{A}$, and possibly joint factorizations of the regularization matrix Ψ and $\mathbf{A}$ (e.g., the GSVD), making them suitable only for small scale problems or for problems where $\mathbf{A}$, and possibly Ψ, have a special structure that can be exploited for storage and computations; see [12, 23]. In general, when solving large-scale problems, only matrix-free methods, which do not require storage nor factorizations of $\mathbf{A}$ but rather computations of matrix-vector products with $\mathbf{A}$ and, often, with $\mathbf{A}^T$, are viable options. This section is devoted to projection methods, which are iterative methods that determine a sequence of approximate regularized solutions of the original problem (1) in a sequence of subspaces of $\mathbb{R}^n$ of dimensions up to $d^* \ll n$.

Remark 1. *All the projection methods available within TRIPs-Py compute a regularized solution by one of the following strategies:*

1. *Applying a projection method directly to problems (1) and stopping before the noise corrupts the approximate solution (i.e., exploiting semiconvergence); see [12, Chapter 6].*
2. *Applying a projection method to approximate the solution of a Tikhonov-like regularized problem.*
3. *Adopting a 'hybrid' approach, whereby Tikhonov regularization is applied to a sequence of projected problems [15].*

As far as regularized problems in the 2-norm are concerned, all the approaches listed above can be employed. Moreover, when considering standard form Tikhonov regularization with a given regularization parameter, the second and third approaches are equivalent. When considering regularization problems formulated in some ℓ_p norm, only the second approach will be considered within a computationally convenient majorization-minimization strategy.

In general, a projection method for computing an approximation $\mathbf{x}_d$ of a solution of problems (1) is defined by the two conditions

$$\mathbf{x}_d \in \mathcal{S}_d, \quad \mathbf{r}_d := \mathbf{b} - \mathbf{A}\mathbf{x}_d \perp \mathcal{C}_d, \tag{12}$$

Table 2: List of decompositions used by some TRIPs-Py solvers.

Decomposition	Description	Ref.
`arnoldi`	Arnoldi decomposition	[25]
`arnoldi_update`	updated Arnoldi decomposition	[25]
`gsvd`	Generalized Singular Value Decomposition	[21]
`golub_kahan`	Golub-Kahan decomposition	[21]
`golub_kahan_update`	updated Golub-Kahan decomposition	[21]

where $\mathcal{S}_d$ and $\mathcal{C}_d$ are subspaces of $\mathbb{R}^n$ and $\mathbb{R}^m$ of dimension $d \ll \min\{m, n\}$, commonly referred to as the approximation subspace and the constraint subspace, respectively. If $\mathbf{S}_d \in \mathbb{R}^{n\times d}$ and $\mathbf{C}_d \in \mathbb{R}^{m\times d}$ are matrices whose columns span $\mathcal{S}_d$ and $\mathcal{C}_d$, respectively, then the above conditions can be equivalently expressed as

$$\text{find} \quad \mathbf{t}_d \in \mathbb{R}^d \quad \text{such that} \quad \mathbf{C}_d^T(\mathbf{b} - \mathbf{A}\mathbf{S}_d\mathbf{t}_d) = \mathbf{0}\,, \tag{13}$$

so that, since $d \ll n$, one can solve

$$\underbrace{\mathbf{C}_d^T\mathbf{A}\mathbf{S}_d}_{=:\mathbf{G}_d\in\mathbb{R}^{d\times d}}\ \mathbf{t}_d = \underbrace{\mathbf{C}_d^T\mathbf{b}}_{=:\mathbf{g}_d\in\mathbb{R}^d} \tag{14}$$

by any direct method and form the approximate solution of (1) by taking $\mathbf{x}_d = \mathbf{S}_d\mathbf{t}_d$. In practice, starting without loss of generality from the initial guess $\mathbf{x}_0 = \mathbf{0}$, a projection method for the solution of (1) is an iterative method that computes a sequence of approximate solutions $\{\mathbf{x}_d\}_{d=1,2,\dots}$ satisfying conditions (12) by building a sequence of nested subspaces

$$\mathcal{S}_d = \text{range}(\mathbf{S}_d) = \text{range}([\mathbf{S}_{d-1}, \mathbf{s}_d]), \quad \mathcal{C}_d = \text{range}(\mathbf{C}_d) = \text{range}([\mathbf{C}_{d-1}, \mathbf{c}_d]), \quad d = 1, 2, \dots,$$

with $\mathbf{S}_0, \mathbf{C}_0$ being empty matrices; see [24] for more details.

The majority of the solvers in TRIPs-Py are projection methods onto Krylov subspaces [14], i.e., both $\mathcal{S}_d$ and $\mathcal{C}_d$ are Krylov subspaces. In general, given $\mathbf{M} \in \mathbb{R}^{n\times n}$ and $\mathbf{n} \in \mathbb{R}^n$, a Krylov subspace is defined as

$$\mathcal{K}_d(\mathbf{M}, \mathbf{n}) = \text{span}\{\mathbf{n}, \mathbf{M}\mathbf{n}, \mathbf{M}^2\mathbf{n}, \dots, \mathbf{M}^{(d-1)}\mathbf{n}\}\,. \tag{15}$$

Here and in the following, we assume that the dimension of $\mathcal{K}_d(\mathbf{M}, \mathbf{n})$ is d; appropriate checks are incorporated into TRIPs-Py to make sure that this assumption is satisfied. Different Krylov methods differ in the choices of their approximation and constraint Krylov subspaces, and in their implementation: when different Krylov subspace methods share the same $\mathcal{S}_d$ and $\mathcal{C}_d$, $d = 1, 2, \dots$, we say that they are mathematically equivalent. When Krylov methods are used to solve problems (1), the quantities $\mathbf{M}$ and $\mathbf{n}$ appearing in (15) are typically defined in terms of $\mathbf{A}$ and $\mathbf{b}$ (for square matrices), or $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$, or $\mathbf{A}\mathbf{A}^T$ and $\mathbf{b}$ (for rectangular matrices). In the following, we will refer to (15) as standard Krylov subspace, to distinguish it from other Krylov-like subspaces that are generated by computing matrix-vector products with an iteration-dependent modification of the matrix $\mathbf{M}$, and therefore cannot be expressed taking increasing powers of $\mathbf{M}$ as in (15).

2.2.1 Methods based on standard Krylov subspaces

In this section, we briefly describe the Krylov subspace methods available within TRIPs-Py: namely, Generalized Minimal Residual (GMRES)(which can be applied when the coefficient matrix in (1) is square), Least Squares QR (LSQR) and Conjugate Gradient Least Squares (CGLS), along with their available hybrid variants. We will emphasize how such methods relate to the framework in Remark 1.

Methods based on the Arnoldi algorithm: GMRES and its hybrid variant Given $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$, d iterations of the Arnoldi algorithm initialized with $\mathbf{v}_1 = \mathbf{b}/\|\mathbf{b}\|_2$ compute the partial factorization

$$\mathbf{A}\mathbf{V}_d = \mathbf{V}_{d+1}\mathbf{H}_d, \tag{16}$$

where $\mathbf{V}_{d+1} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{d+1}] = [\mathbf{V}_d, \mathbf{v}_{d+1}] \in \mathbb{R}^{n\times(d+1)}$ has orthonormal columns that span the Krylov subspace $\mathcal{K}_{d+1}(\mathbf{A}, \mathbf{b})$, and $\mathbf{H}_d \in \mathbb{R}^{(d+1)\times d}$ is upper Hessenberg. TRIPs-Py provides two versions of the

Arnoldi algorithm, both based on the modified Gram-Schmidt orthogonalization process (see Table 2): one computes a given number of Arnoldi steps, while the other only computes one step, updating an already available partial Arnoldi decomposition. If the matrix $\mathbf{A}$ is symmetric, the Arnoldi algorithm simplifies to the symmetric Lanczos algorithms [26], although only the former is provided in TRIPs-Py.

GMRES is a Krylov method for the solution of the linear system appearing in (1). At the dth iteration, GMRES chooses $\mathcal{S}_d = \mathcal{K}_d(\mathbf{A}, \mathbf{b})$, $\mathcal{C}_d = \mathbf{A}\mathcal{K}_d(\mathbf{A}, \mathbf{b})$ in (12). Because of this choice, the associated residual $\mathbf{r}_d$ at the dth iteration is minimized in $\mathcal{K}_d(\mathbf{A}, \mathbf{b})$; see [14]. In practice, exploiting the Arnoldi factorization (16),

$$\mathbf{x}_d = \arg\min_{\mathbf{x}\in\mathcal{K}_d(\mathbf{A},\mathbf{b})} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{A}\mathbf{V}_d\mathbf{t} - \mathbf{b}\|_2 = \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{H}_d\mathbf{t} - \|\mathbf{b}\|_2\mathbf{e}_1\|_2, \tag{17}$$

where $\mathbf{e}_1$ denotes the first canonical basis vector of $\mathbb{R}^{d+1}$. Referring to the quantities introduced in equation (14), GMRES takes $\mathbf{G}_d = \mathbf{H}_d^T\mathbf{H}_d$ and $\mathbf{g}_d = \mathbf{H}_d^T(\|\mathbf{b}\|_2\mathbf{e}_1)$. It has been theoretically proven that GMRES is an iterative regularization method, so that the number of GMRES iterations serves as a regularization parameter; in other words, GMRES adopts the first strategy listed in Remark 1. If an estimate of the norm of the noise $\mathbf{e}$ affecting the data $\mathbf{b}$ in (2) is provided, then TRIPs-Py automatically uses the discrepancy principle as stopping criterion for the GMRES iterations; see Section 2.3 for more details. Otherwise, the user may assign a maximum number of iterations to be performed, being aware that too many iterations may result in an under-regularized solution. From the first equality in equation (17), we can see that GMRES can be expressed in the general framework of problem (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = 0$ and $\mathcal{D} = \mathcal{K}_{d^*}(\mathbf{A}, \mathbf{b})$, where d^* is the stopping iteration for GMRES.

The *hybrid GMRES* method applies some additional, iteration-dependent Tikhonov regularization in standard form to the projected problem appearing in (17), following the third principle described in Remark 1. Namely, the dth iteration of the hybrid GMRES method computes the regularized solution

$$\mathbf{x}_d = \mathbf{V}_d \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{H}_d\mathbf{t} - \|\mathbf{b}\|_2\mathbf{e}_1\|_2^2 + \alpha_d\|\mathbf{t}\|_2^2, \tag{18}$$

where α_d is an iteration-dependent regularization parameter that can be automatically chosen applying the generalized cross validation method or the discrepancy principle (if an estimate of the noise magnitude $\|\mathbf{e}\|_2$ is provided) to the projected problem; we refer to Section 2.3 for more details. Assuming that a suitable value of α_d is chosen at each iteration, thanks to the additional regularization imposed in (18) to the projected problem, hybrid GMRES is not affected by semiconvergence and setting a stopping criterion is less crucial than for GMRES; see [15]. Therefore hybrid GMRES is stopped when a maximum number d^* of iterations is performed. By exploiting decomposition (16) within (18) and the definition of the matrix $\mathbf{V}_{d+1}$, it can be shown that the hybrid GMRES method belongs to the general framework stated in (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_2^2$, $\alpha = \alpha_{d^*}$ and $\mathcal{D} = \mathcal{K}_{d^*}(\mathbf{A}, \mathbf{b})$. The *Arnoldi-Tikhonov* method is mathematically equivalent to hybrid GMRES, the only difference between the two being that the former performs a given number of Arnoldi steps (which, if an estimate of $\|\mathbf{e}\|_2$ is available, is determined by the discrepancy principle), and regularizes the projected problem only at the last computed iteration.

Methods based on Golub-Kahan bidiagonalization (LSQR and its hybrid variant) and CGLS Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$, d iterations of the Golub–Kahan bidiagonalization algorithm initialized with $\mathbf{v}_1 = \mathbf{A}^T\mathbf{b}/\|\mathbf{A}^T\mathbf{b}\|_2$ and $\mathbf{u}_1 = \mathbf{b}/\|\mathbf{b}\|_2$ compute the partial factorizations

$$\mathbf{A}\mathbf{V}_d = \mathbf{U}_{d+1}\mathbf{B}_d, \quad \mathbf{A}^T\mathbf{U}_{d+1} = \mathbf{V}_{d+1}\bar{\mathbf{B}}_{d+1}^T, \tag{19}$$

where $\mathbf{V}_{d+1} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{d+1}] = [\mathbf{V}_d, \mathbf{v}_{d+1}] \in \mathbb{R}^{n\times(d+1)}$ has orthonormal columns that span the Krylov subspace $\mathcal{K}_{d+1}(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$, $\mathbf{U}_{d+1} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{d+1}] = [\mathbf{U}_d, \mathbf{u}_{d+1}] \in \mathbb{R}^{m\times(d+1)}$ has orthonormal columns that span the Krylov subspace $\mathcal{K}_{d+1}(\mathbf{A}\mathbf{A}^T, \mathbf{b})$, $\bar{\mathbf{B}}_{d+1} \in \mathbb{R}^{(d+1)\times(d+1)}$ is a lower bidiagonal matrix, and $\mathbf{B}_d$ is obtained by taking the first d columns of $\bar{\mathbf{B}}_{d+1}$. As for the Arnoldi algorithm, TRIPs-Py provides two versions of the Golub–Kahan bidiagonalization algorithm (see Table 2); both versions are implemented without reorthogonalization.

LSQR is a Krylov method for the solution of the problems appearing in (1) that, at the dth iteration, chooses $\mathcal{S}_d = \mathcal{K}_d(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$, $\mathcal{C}_d = \mathbf{A}\mathcal{K}_d(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$ in (12). Because of this choice, at the dth iteration the residual $\mathbf{r}_d$ is minimized in $\mathcal{K}_d(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$; see [14]. In practice, exploiting Golub-Kahan bigiagonalization (19),

$$\mathbf{x}_d = \arg\min_{\mathbf{x}\in\mathcal{K}_d(\mathbf{A}^T\mathbf{A},\mathbf{A}^T\mathbf{b})} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \mathbf{V}_d \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{A}\mathbf{V}_d\mathbf{t} - \mathbf{b}\|_2 = \mathbf{V}_d \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{B}_d\mathbf{t} - \|\mathbf{b}\|_2\mathbf{e}_1\|_2. \tag{20}$$

Referring to the quantities introduced in equation (14), LSQR takes $\mathbf{G}_d = \mathbf{B}_d^T \mathbf{B}_d$ and $\mathbf{g}_d = \mathbf{B}_d^T(\|\mathbf{b}\|_2 \mathbf{e}_1)$. It is well-known that LSQR is an iterative regularization method, so that the number of LSQR iterations serves as a regularization parameter; in other words, LSQR adopts the first strategy listed in Remark 1. Similarly to GMRES, the LSQR iterations are stopped when the discrepancy principle is satisfied (if an estimate of $\|\mathbf{e}\|_2$ is provided by the user) or if a maximum number of iterations is performed. From the first equality in equation (20), we can see that LSQR can be expressed in the general framework of problem (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = 0$ and $\mathcal{D} = \mathcal{K}_{d^*}(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$, where d^* is the stopping iteration for LSQR.

Similarly to hybrid GMRES, *hybrid LSQR* applies some additional, iteration-dependent Tikhonov regularization in standard form to the projected problem appearing in (20), following the third principle described in Remark 1. Namely, the dth iteration of the hybrid LSQR method computes the regularized solution

$$\mathbf{x}_d = \mathbf{V}_d \arg\min_{\mathbf{t}\in\mathbb{R}^d} \|\mathbf{B}_d\mathbf{t} - \|\mathbf{A}^T\mathbf{b}\|_2\mathbf{e}_1\|_2^2 + \alpha_d\|\mathbf{t}\|_2^2, \tag{21}$$

where α_d is an iteration-dependent regularization parameter that can be automatically chosen by applying the same approaches listed for hybrid GMRES. A proper choice of α_d mitigates the LSQR semiconvergence and the hybrid LSQR iterations are stopped when a maximum number d^* of iterations is performed. Similarly to hybrid GMRES, hybrid LSQR belongs to the general framework stated in (3), with $\mathcal{F}(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_2^2$, $\alpha = \alpha_{d^*}$ and $\mathcal{D} = \mathcal{K}_{d^*}(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{b})$. Similarly to Arnoldi-Tikhonov, *Golub-Kahan-Tikhonov* computes a fixed number of Golub-Kahan iterations and applies Tikhonov regularization only to the last computed projected problem.

CGLS is a Krylov method for the solution of (1), which is mathematically equivalent to LSQR. That is, at the dth iteration, $\mathcal{S}_d$ and $\mathcal{C}_d$ are as in LSQR, but the CGLS solution is computed using a three-term recurrence formula, rather than by (implicitly) computing the factorization (19) and solving the projected problem (20) as in LSQR. Both CGLS and LSQR can be naturally used to approximate the solution of the Tikhonov-regularized problem (7) (both in standard and general form, with a fixed regularization parameter α), following the second strategy in Remark 1. This is achieved by applying such solvers to the equivalent formulation of (7) as a damped least squares problem, and stopping when we reach high accuracy in the associated normal equation residual. However, since CGLS does not compute the factorization (19), it is not possible to state a hybrid version of CGLS as in (21).

2.2.2 Methods based on generalized Krylov subspaces

TRIPs-Py contains methods that rely on generalized Krylov subspaces, whose definition will be made clear in the next paragraph. Such methods include GKS, MMGKS, AnisoTV, IsoTV, and GS (see Table 1 for a few more details). First we briefly describe the generalized Krylov subspace method (GKS) that approximates the solution of (7), with a general regularization matrix $\Psi \in \mathbb{R}^{k\times n}$; see [18]. Then we describe the majorization-minimization (MM) technique that solves (3) for a broad selection of $0 < p, q \leq 2$ and Ψ, and we explain how GKS can be used to approximate the solution of the resulting reweighted problem: the resulting method is referred to as MMGKS.

Generalized Krylov subspace (GKS) method The GKS method computes an approximation to (7) with a general $\Psi \in \mathbb{R}^{k\times n}$ by first determining an initial approximation subspace for the solution through $1 \leq d \ll \min\{m, n\}$ steps of the Golub-Kahan bidiagonalization algorithm applied to $\mathbf{A}$, with initial vector $\mathbf{b}$. This gives a decomposition like the first one appearing in (19). As far as d is relatively small, it is inexpensive to compute the skinny QR factorizations

$$\begin{aligned} \mathbf{A}\mathbf{V}_d &= \mathbf{Q}_\mathbf{A}\mathbf{R}_\mathbf{A} \quad \text{with} \quad \mathbf{Q}_\mathbf{A} \in \mathbb{R}^{m\times d}, \quad \mathbf{R}_\mathbf{A} \in \mathbb{R}^{d\times d}, \\ \Psi\mathbf{V}_d &= \mathbf{Q}_\Psi\mathbf{R}_\Psi \quad \text{with} \quad \mathbf{Q}_\Psi \in \mathbb{R}^{r\times d}, \quad \mathbf{R}_\Psi \in \mathbb{R}^{d\times d}, \end{aligned} \tag{22}$$

where the matrices $\mathbf{Q}_\mathbf{A}$ and $\mathbf{Q}_\Psi$ have orthonormal columns and the matrices $\mathbf{R}_\mathbf{A}$ and $\mathbf{R}_\Psi$ are upper triangular. Taking $\mathbf{x}_d = \mathbf{V}_d\mathbf{t}$ in (7) and using the factorizations in (22), we obtain the d-dimensional linear system of equations

$$(\mathbf{R}_\mathbf{A}^T\mathbf{R}_\mathbf{A} + \alpha\mathbf{R}_\Psi^T\mathbf{R}_\Psi)\mathbf{t} = \mathbf{R}_\mathbf{A}^T\mathbf{Q}_\mathbf{A}^T\mathbf{b}, \tag{23}$$

to be solved to get $\mathbf{t} = \mathbf{t}_d$. We then expand the approximation subspace for the solution of (7) by taking

$$\mathbf{V}_{d+1} = [\mathbf{V}_d, \mathbf{v}_{\text{new}}] \in \mathbb{R}^{n\times(d+1)}, \quad \text{where} \quad \mathbf{v}_{\text{new}} = \widetilde{\mathbf{r}}_d/\|\widetilde{\mathbf{r}}_d\|_2, \quad \widetilde{\mathbf{r}}_d = \mathbf{A}^T(\mathbf{A}\mathbf{V}_d\mathbf{t}_d - \mathbf{b}) + \alpha\Psi^T\Psi\mathbf{V}_d\mathbf{t}_d, \tag{24}$$

i.e., by adding the current normalised residual of the (full) normal equations. This concludes the first GKS iteration.

Next, the QR factorization in (22) is updated for the matrices $\mathbf{AV}_{d+1}$ and $\Psi\mathbf{V}_{d+1}$ and the process in steps (22) - (24) is repeated, expanding the solution space during subsequent GKS iterations until a sufficiently accurate solution is reached. Since a suitable value of the regularization parameter α in (7) is usually not know and applying some parameter choice strategy to (7) is computationally unfeasible for large-scale problems, the GKS method allows an iteration-dependent choice of α. Specifically, in TRIPs-Py one can adopt the discrepancy principle (if a good estimate of the noise magnitude $\|\mathbf{e}\|_2$ is provided by the user) or generalized cross validation to compute $\alpha = \alpha_d$ for problem (23); see Section 2.3 for more details. Since, in general, α may vary at each iteration and $\Psi \neq \mathbf{I}$, the range of $\mathbf{V}_{d+1}$ (approximation subspace for the solution) is not a standard Krylov subspace anymore, and it is called generalized Krylov subspace. If α is fixed or $\Psi = \mathbf{I}$, the GKS method is mathematically equivalent to LSQR or CGLS applied to the damped least squares formulation of the Tikhonov problem (7).

Majorization-minimization Generalized Krylov subspace (MMGKS) method Let us consider problem (3) with $\mathcal{F}(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|_p^p$ and $\mathcal{R}(\mathbf{x}) = \|\Psi\mathbf{x}\|_q^q$. Due to the non-differentiability and possibly nonconvexity for certain choices of p and q, it is common to replace the data fidelity and the regularization terms with differentiable approximations thereof, and solve the following problem

$$\min_{\mathbf{x}} \mathcal{J}_{\epsilon,\alpha}(\mathbf{x}) = \min_{\mathbf{x}} \sum_{j=1}^{m} \phi_{p,\epsilon}((\mathbf{Ax} - \mathbf{b})_j) + \alpha \sum_{j=1}^{k} \phi_{q,\epsilon}((\Psi\mathbf{x})_j)\,, \tag{25}$$

where

$$\phi_{r,\epsilon}(t) = \left(t^2 + \epsilon^2\right)^{r/2} \quad \text{with} \quad \begin{cases} \epsilon > 0 & \text{for} \quad 0 < r \leq 1, \\ \epsilon = 0 & \text{for} \quad r > 1, \end{cases} \tag{26}$$

for a small positive constant $\epsilon > 0$. A well-known approach to solve (25) is majorization-minimization (MM) [27, 28], which solves a sequence of regularized least squares problems. Namely, at the $(\ell+1)$th MM iteration, one computes

$$\mathbf{x}^{(\ell+1)} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{P}_{\text{fid}}^{(\ell)}\,(\mathbf{Ax} - \mathbf{b})\,\|_2^2 + \alpha\|\mathbf{P}_{\text{reg}}^{(\ell)}\Psi\mathbf{x}\|_2^2\,, \tag{27}$$

where $\mathbf{P}_s^{(\ell)}$ ($s =$ fid, reg) denotes a diagonal weighting matrix, whose entries are determined from the current solution $\mathbf{x}^{(\ell)}$, $\ell = 0, 1, \ldots$.

Classical methods for MM involve solving (27) with a fixed regularization parameter by, e.g., applying CGLS, which results in time-consuming inner-outer iterative strategies. The MMGKS method implemented in TRIPs-Py simultaneously computes a new approximation $\mathbf{x}^{(\ell+1)}$ and updates the weights $\mathbf{P}_s^{(\ell+1)}$ ($s = \text{fid}, \text{reg}$) by projecting the current problem (27) onto a generalized Krylov subspace, whereby an iteration-dependent suitable value for the regularization parameter $\alpha = \alpha_\ell$ can be computed (similarly to the method described in the previous paragraph) and the quantities $\mathbf{P}_{\text{fid}}^{(\ell)}\mathbf{A}$, $\mathbf{P}_{\text{fid}}^{(\ell)}\mathbf{b}$, and $\mathbf{P}_{\text{reg}}^{(\ell)}\Psi$ can be incorporated. In this way, for the MMKS methods, no inner-outer iteration scheme is needed and an MM iterate corresponds to a GKS iterate. Namely, the generalized Krylov subspace is expanded by 1 vector at each iteration, so that the $(\ell+1)$th approximate solution $\mathbf{x}^{(\ell+1)}$ belongs to a generalized Krylov subspace of dimension $d = d_0 + \ell$, where d_0 denotes the dimension of the initial approximation subspace for building the generalized Krylov subspace. We refer to [29, 19] for further details.

A variety of regularized formulations can be handled more or less straightforwardly by the MMGKS methods, provided that the parameters p and q (for the ℓ_p and ℓ_q (quasi) norms), the regularization matrix Ψ and the weights $\mathbf{P}_s^{(\ell)}$ ($s =$ fid, reg) are properly defined. Some of the solvers listed in Table 1 are indeed drivers for MMGKS, whereby MMGKS is called with specific inputs, to conveniently handle particularly relevant regularization terms. Namely,

- `AnisoTV` implements anisotropic total variation (TV) regularization. In this instance, $p = 2$ and the matrix $\Psi \in \mathbb{R}^{k\times n}$ is a rescaled finite difference discretization of the first derivative operator in one or two spatial dimensions, or in three spatio-temporal dimensions; k depends on the dimensionality of the problem. The weights are given by the diagonal matrix with entries

$$\left((\Psi\mathbf{x}^{(\ell)})_j^2 + \epsilon^2\right)^{(q-2)/4}, \quad \text{where} \quad 0 < q \leq 1\,.$$

- `IsoTV` implements isotropic TV regularization. In one spatial dimension this coincides with anisotropic TV, so that isotropic TV is more meaningful in two spatial dimensions or in three spatio-temporal dimensions.

- `GS` implements a regularizer enforcing group sparsity (possibly under transform), for solutions that are that are naturally partitioned in subsets; see [30].

We refer to Section 3.3 for more details about (an)isotropic TV and group sparsity within MMGKS; see also [20].

2.3 Strategies to choose the regularization parameter

In this section we discuss methods available within TRIPs-Py to choose the regularization parameter for the solvers described in the previous two subsections (and summarized in Table 1). Properly choosing regularization parameters is a pivotal task for any regularization method, as the success of the latter crucially depends on the former. In this section, when considering projection methods, it is convenient to use the following compact notation for the regularized solution associated to the dth projected problem

$$\mathbf{x}_d = \mathbf{V}_d \underbrace{\mathbf{F}_{\alpha_d}^{\dagger} \mathbf{f}_d}_{=:\mathbf{t}_d}, \tag{28}$$

where $\mathbf{V}_d$ is the basis for the dth approximation subspace, $\mathbf{F}_{\alpha_d}^{\dagger}$ is the so-called dth regularized inverse for the projected problem (depending on the dth Tikhonov regularization parameter α_d), and $\mathbf{f}_d$ is the dth projected right-hand side vector. All quantities appearing in (28) are specific for each projection method. In particular, for projections methods that do not involve any additional Tikhonov regularization, $\mathbf{x}_d = \mathbf{V}_d \mathbf{F}_0^{\dagger} \mathbf{f}_d$.

Discrepancy principle Let us assume that an estimate δ for the norm of the noise $\mathbf{e}$ appearing in (2) is available. Then the discrepancy principle computes the regularization parameter for the regularized solution of the generic regularization problem (3) by imposing

$$\mathcal{D}(\mathbf{x}_{\text{reg}}) := \|\mathbf{A}\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2 = \eta\delta, \tag{29}$$

where $\eta > 1$ ($\eta \simeq 1$) is a safety factor. Since the specific expression and properties of the functional $\mathcal{D}(\mathbf{x}_{\text{reg}})$ depend on the considered regularization method, TRIPs-Py includes different versions of (29). In general, for the discrepancy principle to be satisfied, one should make the natural assumption that

$$\|\mathbf{b}_0\|_2 \leq \|\mathbf{e}\|_2 \leq \|\mathbf{b}\|_2 \,, \tag{30}$$

where $\mathbf{b}_0$ denotes the orthogonal projection of $\mathbf{b}$ onto the null space of $\mathbf{A}\mathbf{A}^T$; see [31].

For T(G)SVD, $\mathcal{D}(\mathbf{x}_h)$ is a decreasing functional of the discrete truncation parameter h: therefore one chooses the truncation parameter h^* such that

$$\mathcal{D}(\mathbf{x}_{h^*}) \geq \eta\delta > \mathcal{D}(\mathbf{x}_{h^*+1}) \,.$$

Similarly, for purely iterative regularizing methods, $\mathcal{D}(\mathbf{x}_{\text{reg}})$ is evaluated at discrete points (i.e., the number of iterations or, equivalently, the dimension of the projection subspace). For the dth projected problem, $\mathcal{D}(\mathbf{x}_d)$ is essentially the norm of the residual associated to the dth solution. For GMRES and LSQR, $\mathcal{D}(\mathbf{x}_d)$ is a decreasing functional of d: this is a consequence of the optimality properties mentioned in Section 2.2.1. Therefore, to satisfy the discrepancy principle, one stops at the iteration d^* such that

$$\mathcal{D}(\mathbf{x}_{d^*-1}) \geq \eta\delta > \mathcal{D}(\mathbf{x}_{d^*}) \,.$$

Note that, for GMRES and LSQR, the discrepancy principle is computationally cheap to apply, as the functional $\mathcal{D}(\mathbf{x}_d)$ can be computed with respect to the projected coefficient matrix and right-hand side vector; see the last two equalities in equations (17) and (20), respectively, for a justification.

When the discrepancy principle is applied to Tikhonov regularization (for either the full-dimensional problem or at each iteration of a projected problem), (29) amounts to a nonlinear equation to be solved with respect to the regularization parameter α. Note that, with the change of variable $\hat{\alpha} = 1/\alpha$ in (7), (23), (27), and $\hat{\alpha}_d = 1/\alpha_d$ in (18), (21), $\mathcal{D}(\mathbf{x}_{\text{reg}})$ is a decreasing convex functional of $\hat{\alpha}$ and $\hat{\alpha}_d$, respectively, so that Newton's method is guaranteed to converge when started on the left of the zero of $\mathcal{D}(\mathbf{x}_{\text{reg}}) - \eta\delta$. For hybrid GMRES and hybrid LSQR, i.e., when considering the projected problems appearing in (18) and (21), the discrepancies computed with respect to the full-dimensional and the projected problems coincide, i.e.,

$$\mathcal{D}(\mathbf{x}_{\text{reg}}) = \|\mathbf{b} - \mathbf{A}\mathbf{V}_d \mathbf{F}_{\alpha_d}^{\dagger} \mathbf{f}_d\|_2 = \|\mathbf{f}_d - \mathbf{F}_{\alpha_d}^{\dagger} \mathbf{f}_d\|_2 \,; \tag{31}$$

see, again, the last two equalities in equations (17) and (20). Therefore, in order for $\mathcal{D}(\mathbf{x}_{\text{reg}}) - \eta\delta$ to have a zero, one should also assume that

$$\|(\mathbf{I} - \mathbf{F}_0^\dagger)\mathbf{f}_d\|_2 \leq \|\mathbf{e}\|_2 \leq \|\mathbf{b}\|_2 \,, \tag{32}$$

(it should probably be something like $(\mathbf{I} - \mathbf{F}\mathbf{F}_0^\dagger)$) which is essentially condition (30) applied to the projected problems appearing in (18) and (21). Since the quantity on the left of the first inequality is the norm of the GMRES or the LSQR residuals (recall that, with $\alpha_d = 0$ hybrid GMRES or hybrid LSQR are equivalent to GMRES or LSQR, respectively), and since such residuals decrease with d, the above condition is satisfied when d is sufficiently large (typically after only a few iterations); see also [32]. For methods based on generalized Krylov subspaces, the discrepancies computed with respect to the full-dimensional and the projected problems are different. Namely,

$$\mathcal{D}(\mathbf{x}_{\text{reg}}) = \left(\|\mathbf{f}_d - \mathbf{R}_\mathbf{A}\mathbf{F}_\alpha^\dagger\mathbf{f}_d\|_2^2 + \|(\mathbf{I} - \mathbf{Q}_\mathbf{A}\mathbf{Q}_\mathbf{A}^T)\mathbf{b}\|_2^2\right)^{1/2} \tag{33}$$

when computed for the full-dimensional problem. The second term in the above sum is dropped when $\mathcal{D}(\mathbf{x}_{\text{reg}})$ is computed with respect to the projected problem; see also [33]. In this setting, in order for $\mathcal{D}(\mathbf{x}_{\text{reg}}) - \eta\delta$ to have a zero, a condition similar to (32) should be satisfied; for the full-dimensional problem, the leftmost quantity in (32) should be replaced by $(\|(\mathbf{I} - \mathbf{F}_0^\dagger)\mathbf{f}_d\|_2 + \|(\mathbf{I} - \mathbf{Q}_\mathbf{A}\mathbf{Q}_\mathbf{A}^T)\mathbf{b}\|_2^2)^{1/2}$. By default, for the GKS-based solvers, TRIPs-Py applies the discrepancy principle with respect to the full-dimensional problem.

Generalized Cross Validation (GCV) The GCV criterion prescribes to take the regularization parameter that minimizes the functional

$$\mathcal{G}(\mathbf{x}_{\text{reg}}) = \frac{\|\mathbf{A}\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2^2}{\left(\text{trace}(\mathbf{I} - \mathbf{A}\mathbf{A}_{\text{reg}}^\dagger)\right)^2} \,. \tag{34}$$

Such procedure is derived from statistical techniques, starting from the principle that a good regularized solution $\mathbf{x}_{\text{reg}}$ (defined by a good regularization parameter) should be able to predict the exact data $\mathbf{b}_{\text{true}}$ as well as possible. In equation (34), $\mathbf{A}_{\text{reg}}^\dagger$ is the regularized inverse of $\mathbf{A}$ (specific for each regularization method), i.e., a matrix such that $\mathbf{x}_{\text{reg}} = \mathbf{A}_{\text{reg}}^\dagger\mathbf{b}$, and the quantity $\mathbf{A}\mathbf{A}_{\text{reg}}^\dagger$ is often referred to as 'influence matrix'. Since GCV does not require any information about the magnitude of the noise affecting the data $\mathbf{b}$, it is the default regularization parameter choice method for the TRIPs-Py solvers that involve TSVD or Tikhonov regularization (for either the full-dimensional or the projected problem).

For (G)SVD spectral filtering methods, the functional $\mathcal{G}(\mathbf{x}_{\text{reg}})$ can be conveniently expressed with respect to the filter factors and quantities appearing in the SVD of $\mathbf{A}$ or the GSVD of $(\mathbf{A}, \Psi)$. In particular, for T(G)SVD, the functional $\mathcal{G}(\mathbf{x}_h)$ is evaluated at discrete points and its denominator simplifies to $(m-h)^2$, with $h = 1, \ldots, n$ for TSVD and $h = 1, \ldots, k$ for GSVD (see equations (6) and (10), respectively); we refer to [12, §5.4] for further details.

When GCV is applied to the hybrid-projection methods based on standard Krylov subspace methods (Section 2.2.1) and to the methods based on generalized Krylov subspaces (Section 2.2.2), the values of $\mathcal{G}(\mathbf{x}_{\text{reg}})$ may depend on whether the regularization parameter is selected for the projected problem only, or by linking the projected problem to the corresponding full-dimensional regularized problem. For hybrid methods based on standard Krylov subspaces, the numerator of $\mathcal{G}(\mathbf{x}_{\text{reg}})$ (i.e., the square of the functional $\mathcal{D}(\mathbf{x}_{\text{reg}})$) is the same when computed for both the full-dimensional and the projected problems; see (31). This is not true for GKS-based methods; see (33). Concerning the denominator of $\mathcal{G}(\mathbf{x}_{\text{reg}})$, using the properties of the trace, one can derive the expressions

$$\zeta - \text{trace}(\mathbf{E}_d\mathbf{F}_\alpha^\dagger), \quad \text{where} \quad \mathbf{E}_d = \mathbf{H}_d, \mathbf{B}_d, \mathbf{R}_\mathbf{A}$$

for hybrid GMRES, hybrid LSQR and (MM)GKS, respectively. The constant ζ is m for all the methods when computed for the full-dimensional problem. For the projected problems, $\zeta = d + 1$ for hybrid GMRES and hybrid LSQR (see [34] for more detailed derivations in the hybrid GMRES case), and $\zeta = d$ for (MM)GKS. By default, and in agreement with the common choices made in the literature, TRIPs-Py uses the GCV criterion computed for the full-dimensional problem for hybrid GMRES and hybrid LSQR, and the projected GCV version for the solvers based on generalized Krylov subspaces; see [34, 35].

2.4 Regularization operators

This section describes the regularization matrices implemented in TRIPs-Py. We consider two types of operators: those based on a finite-difference discretization of the first derivative operator and those based on framelet operators. Some more details about the usage of these regularization matrices are discussed in Section 3, and an illustration is provided towards the end of the demo `demo_Tomo_large_scale.ipynb`.

Case 1: Regularization operators based on the first derivative operator Let

$$\Psi_D = \begin{bmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{(n_D-1)\times n_D} \quad \text{and} \quad \mathbf{I}_{n_D} \in \mathbb{R}^{n_D \times n_D} \tag{35}$$

be a rescaled finite-difference discretization of the first derivative operator and the identity matrix of order n_D, respectively. For problems that depend on one or two spatial dimensions (x, y) and, possibly, a time dimension t, the matrix Ψ_D is used to obtain discretizations of the first derivatives in the D-direction, with $D = x$ (vertical direction), $D = y$ (horizontal direction), and $D = t$ (time direction). For a static image represented as a 2D array $\mathbf{X} \in \mathbb{R}^{n_x \times n_y}$, such that $\mathbf{x} = \text{vec}(\mathbf{X}) \in \mathbb{R}^{n_x n_y}$ is obtained by stacking the columns of $\mathbf{X}$, its horizontal and vertical derivatives are given as

$$\begin{array}{rcl} \text{vec}(\Psi_x \mathbf{X}) & = & (\mathbf{I}_{n_y} \otimes \Psi_x)\mathbf{x} \in \mathbb{R}^{n_y(n_x-1)} \\ \text{vec}(\mathbf{X}\Psi_y^T) & = & (\Psi_y \otimes \mathbf{I}_{n_x})\mathbf{x} \in \mathbb{R}^{(n_y-1)n_x} \end{array}, \tag{36}$$

respectively. The discrete gradient is then expressed as

$$\Psi_s = [(\mathbf{I}_{n_y} \otimes \Psi_x)^T, (\Psi_y \otimes \mathbf{I}_{n_x})^T]^T. \tag{37}$$

When modelling dynamic inverse problems with a time-varying solution, let $\mathbf{X} \in \mathbb{R}^{n_s \times n_t}$ be the 2D array whose columns store the quantity of interest at the n_t time instants; note that, if such quantities are 2D images, then the columns of $\mathbf{X}$ are vectorialized images with $n_s = n_x n_y$ pixels. The derivative in the time dimension is then given by $\text{vec}(\mathbf{X}\Psi_t^T) = (\Psi_t \otimes \mathbf{I}_{n_s})\mathbf{x} \in \mathbb{R}^{(n_t-1)n_s}$.

Case 2: Regularization operators based on a two-level framelet analysis operator When regularizing by leveraging sparsity but the desired solution is not sparse in the original domain, a well known technique is to perform a transformation to another domain, where the solution may admit a sparse representation. For this purpose, TRIPs-Py provides a two-level framelet analysis operator, defined as follows. Let $\mathbf{W} \in \mathbb{R}^{r \times n}$ with $1 \leq n \leq r$. The set of the rows of $\mathbf{W}$ is a framelet system for $\mathbb{R}^n$ if, for all $\mathbf{x} \in \mathbb{R}^n$,

$$\|\mathbf{x}\|_2^2 = \sum_{j=1}^{r} (\mathbf{w}_j^T \mathbf{x})^2, \tag{38}$$

where $\mathbf{w}_j \in \mathbb{R}^n$ denotes the jth row of the matrix $\mathbf{W}$ (written as a column vector), i.e., $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_r]^T$. The matrix $\mathbf{W}$ is referred to as an analysis operator and $\mathbf{W}^T$ as a synthesis operator. We use the same tight frames as in [36, 37, 38], i.e., the system of linear B-splines. This system is formed by a low-pass filter $\mathbf{W}_0 \in \mathbb{R}^{n \times n}$ and two high-pass filters $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$, whose corresponding masks are

$$\mathbf{w}^{(0)} = \frac{1}{4}[1, 2, 1], \quad \mathbf{w}^{(1)} = \frac{\sqrt{2}}{4}[1, 0, -1], \quad \mathbf{w}^{(2)} = \frac{1}{4}[-1, 2, -1].$$

The analysis operator $\mathbf{W}$ in one space-dimension is derived from these masks and by imposing reflexive boundary conditions to ensure that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$. The corresponding two-dimensional operator $\mathbf{W}$ is given by

$$\mathbf{W} = \left[[\mathbf{W}_0 \otimes \mathbf{W}_0]^T, [\mathbf{W}_0 \otimes \mathbf{W}_1]^T, [\mathbf{W}_0 \otimes \mathbf{W}_2]^T, [\mathbf{W}_1 \otimes \mathbf{W}_0]^T, \ldots, [\mathbf{W}_2 \otimes \mathbf{W}_2]^T\right]^T, \tag{39}$$

where $\otimes$ denotes the Kronecker product. This matrix is not explicitly formed. We note that the evaluation of matrix-vector products with $\mathbf{W}$ and $\mathbf{W}^T$ is inexpensive, because the matrix $\mathbf{W}$ is sparse. The operator $\mathbf{W}$ can be used for instance as a regularization operator in GKS and MMGKS.

Table 3: List of TRIPs-Py demos.

Demo	**Description**
`demo_dynamic_Emoji.ipynb`	Emoji test problem and illustrations for solving the static and dynamic problems.
`demo_dynamic_CrossPhantom.ipynb`	CrossPhantom test problem and illustrations for solving the static and dynamic problems.
`demo_dynamic_Stempo.ipynb`	STEMPO test problem and illustrations for solving the static and dynamic problems.
`demo_1D_Deblurring.ipynb`	1D Deblurring test problem and illustration of direct and iterative methods to solve it.
`demo_2D_Deblurring_small_scale.ipynb`	2D Deblurring for a small scale test problem whose naive solution can be computed, and illustration of direct and iterative regularization methods.
`demo_2D_Deblurring_large_scale.ipynb`	2D Deblurring for a large scale test problem and illustration of iterative methods to solve the ill-posed problem.
`demo_2D_Deblurring_your_data.ipynb`	Illustration of functions that can be used to upload users' images and define a deblurring problem.
`demo_Tomo_small_scale.ipynb`	Small-scale Tomography test problem and illustration of solution methods.
`demo_Tomo_small_scale.ipynb`	Larg-scale Tomography test problem and illustration of iterative regularization methods.
`demo_Tomo_saved_data.ipynb`	Illustration of how to use functions on the Tomography class that access previously saved tomography data.

3 Overview of the TRIPs-Py test problems

In this section, we consider three main classes of test problems. In the first class we consider both 1D and 2D deblurring, with the latter being used to produce both small-scale and large-scale synthetic test problems. In the second class we consider computed tomography with synthetic data, which can be used to generate both small-scale and large-scale inverse problems. The third class contains dynamic inverse problems with real data. The usage of the classes to generate test problems and of the TRIPs-PY's solvers that can be attempted to compute their solution is illustrated in a number of demos collected in jupyter notebooks. A complete list of demos in TRIPs-Py, and short descriptions thereof, is given in Table 3. For a smooth usage of the package, we recommend the users to first download the data needed for test problems from the google drive [1] and place the folder data inside the folder demos.

3.1 Deblurring (deconvolution)

Deblurring can be formulated as an integral equation of the kind

$$\int \mathcal{B}(s,t)x(t)ds + e = b(s), \tag{40}$$

where $s, t \in \mathbb{R}^D$ represent spatial information (in TRIPs-Py, $D = 1, 2$). The kernel $\mathcal{B}(s,t)$ (also known as the point spread function (PSF)) defines the blur. It is well known that, if the kernel is spatially invariant (as it is in TRIPs-Py), i.e., $\mathcal{B}(s,t) = \mathcal{B}(s-t)$, then (40) is a deconvolution problem. In practical settings, discrete data are collected in finite regions, so that the continuous model (40) is discretized and yields a linear system of equation as in (1). In this particular case, the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ represents the blurring operator, which is defined starting from the PSF and the boundary conditions on the unknown quantity

[1] https://drive.google.com/drive/folders/1VB8LaFewgwNKXq7QSVns4D6Rod8W8Pa1?usp=sharing

of interest. In TRIPs-Py the PSF is given by a rescaled, possibly asymmetric Gaussian of the form

$$\mathcal{B}(s,t) = c\exp\left(-\frac{1}{2}(s-t)^T\mathbf{B}(s-t)\right), \quad \text{where} \quad c>0,\ \mathbf{B} = \text{diag}(\beta_1^2,\ldots,\beta_D^2), \tag{41}$$

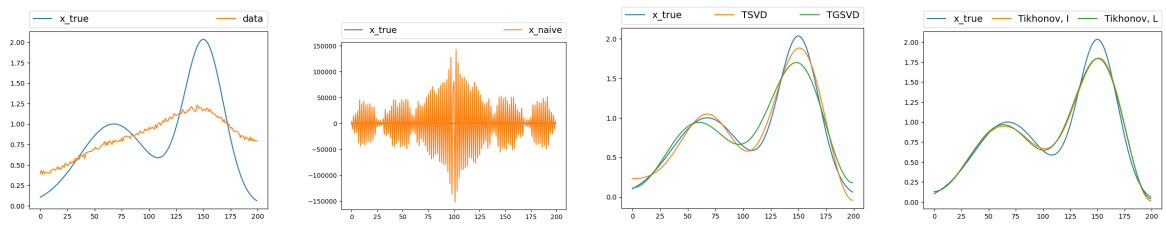
whose spread parameters $\beta_1,\ldots,\beta_D$ $(D=1,2)$ are set by the user; reflective boundary conditions are used. The vector $\mathbf{b}$ contains the vectorized measured blurred and noisy quantity of interest. By default, the synthetic deblurring test problems available within TRIPs-Py avoid inverse crime by allowing a mismatch between the forward operator used to solve the problem, and the forward operator used to generate the data; see [39]. Namely, the latter employs zero boundary conditions. Inverse crimes can be allowed by setting the `CommitCrime` option to `True` (the default being `False`). More details on deblurring can be found in [23].

3.1.1 1D Deblurring

Referring to the notations in (2), in the 1D setting we consider a one dimensional true signal $\mathbf{x}_{\text{true}} \in \mathbb{R}^n$ and a convolution forward operator $\mathbf{A} \in \mathbb{R}^{n\times n}$, with associated smoothed and noisy signal $\mathbf{b} \in \mathbb{R}^n$. This problem can be setup in TRIPs-Py by defining an object of the `Deblurring1D()` class. The method `gen_xtrue()` generates the true signal. The required arguments are the dimension of the problem `nx` and the `test` signal; for the latter, the user can choose among saved options `piecewise`, `sigma`, `curve0`, `curve1`, `curve2`, and `curve3`. The user can have acces to the forward operator by calling the method `forward_Op_1D()` with `parameter` storing the spread of the 1D Gaussian blurring function. The data can be generated using the method `gen_data()`. Noise is then added to the data through the method `add_noise()`, which takes as input the noiseless data, the distribution of the random noise and the noise level (for Gaussian and Laplace noise), i.e., the ratio $\|\mathbf{e}\|_2/\|\mathbf{A}\mathbf{x}_{\text{true}}\|_2$. An illustration of the usage of the class `Deblurring1D` is shown in Code 1, and more illustrations are presented in demo `demo_1D_Deblurring.ipynb`. Figure 2 shows the true `'curve2'` signal, its blurred and noisy version with 1% Gaussian noise, and reconstructions with tSVD and tGSVD.

Code 1: Generate Image deblurring 1D

```
# Define an object of Deblurring1D class
Deblur1D = Deblurring1D(CommitCrime = True)
nx = 200
# Generate a true solution
x_true = Deblur1D.gen_xtrue(nx, test = 'curve2')
# Generate the forward operator
A = Deblur1D.forward_Op_1D(parameter = 30, nx = nx)
# Generate the blurred and noisy curve
b_true = Deblur1D.gen_data(x_true)
(d, delta) = Deblur1D.add_noise(b_true, 'Gaussian', noise_level = 0.01)
Deblur1D.plot_data(d)
Deblur1D.plot_rec(x_true)
```



(a) (b) (c) (d)

Figure 2: 1D image deblurring test problem. (a) $\mathbf{x}_{\text{true}}$ and data with 1% Gaussian noise, (b) $\mathbf{x}_{\text{true}}$ and naive, unregularized solution, (c) $\mathbf{x}_{\text{true}}$, TSVD and TGSVD solutions, (d) $\mathbf{x}_{\text{true}}$, standard and general form Tikhonov solutions.

3.1.2 2D deblurring

In this section we show strategies to define and solve a 2D deblurring problem in TRIPs-Py. Similarly to the 1D case, 2D image deblurring test problems are set up as objects of the `Deblurring2D()` class. As a first step, we start with a small scale problem to illustrate the ill-posedness of such inverse problem and the performance of direct regularization methods applied to it (with some more illustrations available through the notebook `demo_Deblurring_small_scale.ipynb`). Then we consider a larger scale problem to illustrate iterative regularization methods (a full investigation is provided in the demo `demo_Deblurring_large_scale.ipynb`). For consistancy, we choose the same image in both examples, but the user can choose from other images provided in the package such as `h_im`, `hubble`, `grain`, and `sky`. Other images (in tiff, tif, jpg, png or mat formats) can be easily used in TRIPs-Py by performing the following operations:

1. Create the folder `my_image_data` under demos/data and place the desired image inside the folder.
2. Run the function

```
convert_image_for_trips(imag = 'image_name', image_type= image_type')
```

where `'image_name'` and `'image_type'` refer to the desired image name and type.

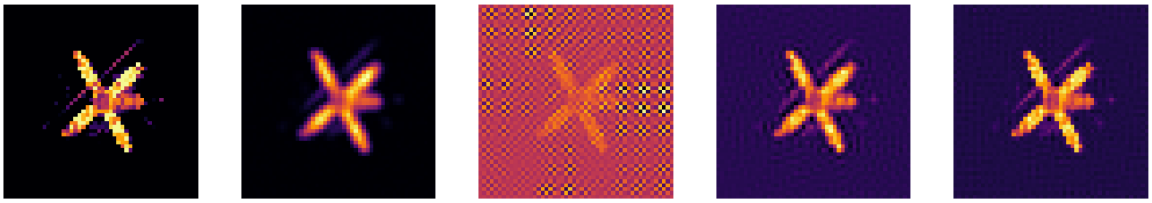
This is shown in the demo `demo_Deblurring_your_data.ipynb`.

Small-scale image deblurring For this illustration we set $n_x = n_y = 50$ and define the forward operator to be a Gaussian PSF (41) with parameters $(\beta_1, \beta_2) = (1, 1)$. The true image and the blurred and noisy image with 1% Gaussian noise are shown in Figure 3 a) and b). We call methods TSVD, Hybrid LSQR, and MMGKS with parameters specified as follows.

```
(x_tsvd, truncation_value) = TruncatedSVD_sol(A.todense(), b_vec, regparam
    = 'dp', delta = delta)
(x_hybrid_lsqr, info_hybrid_lsqr) = Hybrid_LSQR(A, b_vec, n_iter = 100,
    regparam = 'dp', x_true = x_true, delta = delta)
L = first_derivative_operator_2d(nx, ny)
(x_mmgks, info_mmgks) = MMGKS(A, data_vec, L, pnorm=2, qnorm=1,
    projection_dim=2, n_iter = 100, regparam = 'dp', x_true = x_true, delta
     = delta)
```

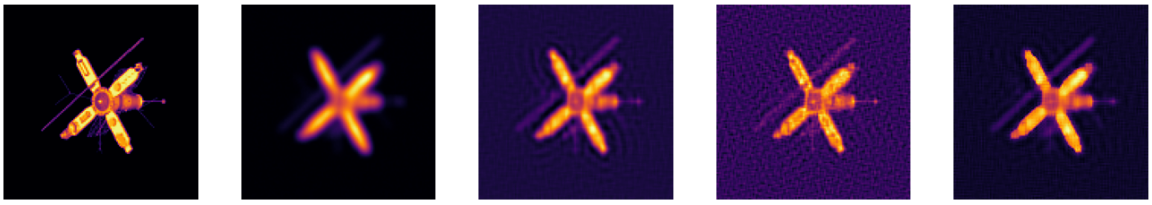
We remark that the user should provide the operator $\mathbf{A}$ as a dense matrix for TSVD and the observed data as a vector for all the methods. The regularization parameter can be a scalar, or chosen by the discrepancy principle (`'dp'`) or generalized cross validation (`'gcv'`). When the regularization parameter is set to `'dp'`, the user must specify the noise level (returned from `add_noise()` function (see Code 1)). For the MMGKS, the user can specify the regularization parameter $\mathbf{L}$ (for this example we choose $\mathbf{L}$ to be 2D discretized derivative operator of the first order). The values of p and q can be set as well through the input parameters `pnorm` and `qnorm`. Approximate solutions obtained from truncated SVD, and iterative methods such as Hybrid LSQR and MMGKS are shown in Figure 3 c), d), and e), respectively. The approximate solution and information collected through the iterations is outputed. Apart from visual inspections provided in Figure 3 for this example, more quantitative measures on the reconstructed solution can be found on the demo jupyter notebook `demo_Deblurring_small_scale.ipynb`.

Large-scale image deblurring For the large-scale version we consider a satellite image of size 128×128 pixels shown in Figure 4 (a). Such image is blurred by a Gaussian PSF with parameters $(\beta_1, \beta_2) = (3, 3)$ and we add 1% Gaussian noise. An illustration of how to define a 2D Deblurring problem is shown in Code 2. The blurred and noisy image is shown in Figure 4 (b). Approximate solutions obtained by hybrid GMRES, hybrid LSQR, and MMGKS are shown in Figure 4 c), d), and e). The calls to these solvers are essentially identical to the ones illustrated for the small-scale example. More details can be found on demo `demo_Deblurring_large_scale.ipynb`.



(a) (b) (c) (d) e)

Figure 3: 2D image deblurring small-scale test problem. (a) True image of 50×50 pixels, (b) Blurred and noisy image with 1% Gaussian noise. Approximate solutions by (c) TSVD, (d) Hybrid LSQR, and (e) MMGKS.



(a) (b) (c) (d) e)

Figure 4: 2D image deblurring large-scale test problem. (a) True image of 128×128 pixels. (b) Blurred and noisy image with 1% Gaussian noise. Approximate solutions obtained by (c) Hybrid GMRES, (d) Hybrid LSQR, and (e) MMGKS.

Code 2: Generate Image deblurring 2D

```
% \begin{minted}[breaklines]{python}
# Create an object of class Deblurring()
Deblur = Deblurring(CommitCrime = False)
nx = 256
ny = 256
# Generate the forward operator
A = Deblur.forward_Op((9,9), (3,3), nx, ny)
choose_image = 'satellite'
# generate the true image
x_true = Deblur.gentrue(choose_image)
# generate data
b_true = Deblur.gen_data(x_true) # generate_matrix)
# add noise
(b, delta) = Deblur.add_noise(b_true, opt= 'Gaussian', noise_level = 0.01)
b_vec = b.reshape((-1,1))
```

3.2 Computerized tomography

Another test problem considered in TRIPs-Py is 2D X-ray computerized tomography (CT), which consists in reconstructing an object (i.e., the attenuation coefficients of an object) from a set of projections along straight lines (i.e., intensities of energy rays recorded by detectors). In the following we briefly describe the physical and mathematical formulation of CT and then illustrate how to define a CT test problem within TRIPs-Py.

Consider $\mathbf{x} = [x_1, x_2]^T$, $\beta \in [-\pi, \pi]$, $\boldsymbol{\beta} = [\cos(\beta), \sin(\beta)]^T$, $\boldsymbol{\beta}^\perp = [-\sin(\beta), \cos(\beta)]^T$, and $c \in \mathbb{R}$. Let

$$\mathcal{L}(\beta, c) = \left\{\mathbf{x} \in \mathbb{R}^2 \,|\, \mathbf{x} \cdot \boldsymbol{\beta} = c\right\} = \left\{\mathbf{x} \in \mathbb{R}^2 \,|\, \mathbf{x} = c\boldsymbol{\beta} + \ell\boldsymbol{\beta}^\perp,\ \ell \in \mathbb{R}\right\}$$

be the perpendicular line to $\boldsymbol{\beta}$ with signed orthogonal distance c from the origin. Then, assuming that absorption dominates potential scattering effects, Lambert-Beer law links the attenuation coefficient $f(\mathbf{x})$ of the object we wish to image to the recorded intensity $I_{\beta,c}$ of the measured X-ray of incoming intensity

I_0 along $\mathcal{L}(\beta, c)$ as follows

$$\underbrace{\int_{\mathcal{L}(\beta,c)} f(\mathbf{x}) d\ell}_{=\mathfrak{R}[f](\beta,c)} = -\log\left(\frac{I_{\beta,s}}{I_0} + e_{\beta,c}\right), \tag{42}$$

where $e_{\beta,c}$ is a random perturbation corrupting the measurements. The left-hand side of the above equation, when computed for all β's and c's, is the Radon transform of the function f. After a discretization process, from (42) we obtain a discrete formulation of the problem, $\mathbf{Ax} = \mathbf{b}$. More details on CT can be found in [40].

Similarly to deblurring, we generate a CT test problem in TRIPs-Py by first defining an object of the class `Tomography()`. By default, also the synthetic CT test problems available within TRIPs-Py avoid inverse crime by using slightly different forward operators to solve the problem and to generate the data. Namely, the set of projection angles for the two operators are affected by a small constant mismatch. Inverse crimes can be allowed by setting the `CommitCrime` option to `True` (the default being `False`). The following demos in TRIPs-Py illustrate how to set up and solve tomography test problems.

- ◇ `demo_Tomo_small_scale.ipynb` This notebook defines a tomography test problem with small dimensions, so that the forward operator can be explicitly formed and stored to compute a naive solution or a regulatized solution from truncated SVD.
- ◇ `demo_Tomo_large_scale.ipynb` This notebook showcases how to generate a large-scale tomography problem along with how to call algorithms to solve the formulated inverse problem. For both demos described above, to setup the forward operator, we use the ASTRA toolbox with 'fanflat' 2D geometry. Other geometries such as 'parallel' and 'fanflat_vec' can be used. More details on the ASTRA toolbox and its documentation can be found in [10] and references therein.
- ◇ `demo_Tomo_saved_data.ipynb` This notebook demonstrates how to generate a tomogrpahy problem where the forward operator and the data are already available. This demo has no dependency on any ASTRA toolbox to generate the forward operator.

Code 3 illustrates how to set up a large scale tomography test problem, where $n_x = n_y = 256$ and the number of view angles is 50; a small scale problem can be defined similarly by reducing n_x and n_y. The angles can be limited by varying the parameter `views`. The true phantom is given in Figure 5(a) and the observed sinogram with 1% Gaussian noise is shown in Figure 5(b). Reconstructed solutions by iterative methods Hybrid LSQR, GKS, and MMGKS are shown in Figure 5 (c), (d), and (e), respectively.

Code 3: Generate X-Ray CT problem

```
# Define an object of the class Tomogrpahy
Tomo = Tomography()
# Specify the dimensions of the phantom and the number of angels
nx = 256
ny = 256
views = 50
# Define the true solution throught the function gen_true().
(x_true, nx, ny) = Tomo.gen_true(testproblem = 'tectonic', nx = nx, ny = ny
    )
# Define the forward operator
A = Tomo.forward_Op(nx, ny, views)
# Generate the data
(A, b_true, p, q, AforMatrixOperation) = Tomo.gen_data(x_true, nx, ny,
    views)
# Add noise in the true simulated sinogram b_true
(b, delta) = Tomo.add_noise(b_true = b_true, opt = 'Gaussian', noise_level
    = 0.001)
```

3.3 Dynamic Computerized Tomography

Recent technological advancements in detector speed and accuracy resulted in a growing interest in X-ray computerized tomography, which in turn prompted the need for new and efficient methods to analyze the collected data. In particular, in this paper and within TRIPs-Py, we are interested in reconstructing a series of images to explore the spatial and temporal properties of data acquired within dynamic CT.

(a) (b) (c) (d) e)

Figure 5: Tomography test problem. (a) True image of 256×256 pixels. (b) Sinogram. Approximate solutions obtained by (c) Hybrid LSQR, (d) GKS, and (e) MMGKS.

The aim of this section is three-folded: 1) to briefly describe several regularization terms $\mathcal{R}(\mathbf{x})$ for use in dynamic inverse problems, 2) to show how to handle dynamic data and generate dynamic CT problems in TRIPs-Py, and 3) to illustrate the need for temporal regularization along with spatial regularization, as well as the usage of TRIPs-Py solvers for dynamic CT.

In the dynamic setting, we are interested in solving the minimization problem

$$\underbrace{\begin{bmatrix} \widehat{\mathbf{A}}^{(1)} & & \\ & \ddots & \\ & & \widehat{\mathbf{A}}^{(n_t)} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \widehat{\mathbf{x}}^{(1)} \\ \vdots \\ \widehat{\mathbf{x}}^{(n_t)} \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} \widehat{\mathbf{e}}^{(1)} \\ \vdots \\ \widehat{\mathbf{e}}^{(n_t)} \end{bmatrix}}_{\mathbf{e}} = \underbrace{\begin{bmatrix} \widehat{\mathbf{b}}^{(1)} \\ \vdots \\ \widehat{\mathbf{b}}^{(n_t)} \end{bmatrix}}_{\mathbf{b}}. \tag{43}$$

The operator $\mathbf{A}$ has typically a block diagonal structure where the blocks $\widehat{\mathbf{A}}^{(t)}$ change in time $t = 1, \ldots, n_t$. If $\widehat{\mathbf{A}}^{(t)} = \widehat{\mathbf{A}}$ for $t = 1, \ldots, n_t$, the forward operator simplifies to $\mathbf{A} = \mathbf{I}_{n_t} \otimes \widehat{\mathbf{A}}$, where $\otimes$ denotes the Kronecker product. Problem (43) may be approached by solving a sequence of 'static' regularized problems of the form

$$\mathbf{x}^{(t)}_{\text{static}} = \underset{\mathbf{x} \in \mathbb{R}^{n_s}}{\arg\min} \|\widehat{\mathbf{A}}^{(t)}\mathbf{x} - \widehat{\mathbf{b}}^{(t)}\|_2^2 + \alpha \|\Psi_s \mathbf{x}\|_q^q, \qquad t = 1, 2, \ldots, n_t, \tag{44}$$

where Ψ_s is defined as in (37) and $q > 0$ (note that $q = 1$ corresponds to anisotropic total variation in space). However, in order to overcome the difficulties associated to often having only limited information per time instance, and to improve the reconstruction quality, it is crucial to incorporate temporal information into the reconstruction process by solving

$$\mathbf{x}_{\text{dynamic}} = \underset{\mathbf{x} \in \mathbb{R}^{n}}{\arg\min} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \alpha \mathcal{R}(\mathbf{x}). \tag{45}$$

Here $\mathcal{R}(\mathbf{x})$ is a regularization term that takes into consideration both spatial and temporal dimensions. In TRIPs-Py, the user can find three regularization terms for (45); more details and a more extended list of such regularizers can be found in [20]. In the Bayesian setting, edge-preserving regularization methods for dynamic problems are proposed in [41].

- ◇ **Anisotropic space-time TV** We take

$$\mathcal{R}(\mathbf{x}) = \sum_{t=1}^{n_t} \|\Psi_s \mathbf{x}^{(t)}\|_1 + \sum_{t=1}^{n_t - 1} \|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\|_1 = \|(\mathbf{I}_{n_t} \otimes \Psi_s)\mathbf{x}\|_1 + \|(\Psi_t \otimes \mathbf{I}_{n_s})\mathbf{x}\|_1 \tag{46}$$

  The anisotropic TV terms $\|\Psi_s \mathbf{x}^{(t)}\|_1$, $t = 1, \ldots, n_t$, ensure that the discrete spatial gradients of the images are sparse at each time step; moreover, we enforce that the images do not change considerably from one time instant to the next one by penalizing the 1-norm of their difference.

- ◇ **Isotropic TV in space, Anisotropic TV in time** Assuming, for simplicity, that $n_x = n_y$, we take

$$\begin{aligned} \mathcal{R}(\mathbf{x}) &= \sum_{\ell=1}^{n} \sqrt{((\mathbf{I}_{n_t} \otimes \mathbf{I}_{n_y} \otimes \bar{\Psi}_x)\mathbf{x})_\ell^2 + ((\mathbf{I}_{n_t} \otimes \bar{\Psi}_y \otimes \mathbf{I}_{n_x})\mathbf{x})_\ell^2} + \sum_{t=1}^{n_t - 1} \|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\|_1 \\ &= \| \left[(\mathbf{I}_{n_t} \otimes \mathbf{I}_{n_y} \otimes \bar{\Psi}_x)\mathbf{x}, (\mathbf{I}_{n_t} \otimes \bar{\Psi}_y \otimes \mathbf{I}_{n_x})\mathbf{x} \right] \|_{2,1} + \|(\Psi_t \otimes \mathbf{I}_{n_s})\mathbf{x}\|_1, \end{aligned} \tag{47}$$

  where $\bar{\Psi}_D$ $(D = x, y)$ denote the square version of the matrix Ψ_D defined in (35), where a zero row has been added at the bottom, and $\|\cdot\|_{2,1}$ denotes the functional defined, for a matrix $\mathbf{Z} \in \mathbb{R}^{m_x \times m_y}$, as $\|\mathbf{Z}\|_{2,1} = \sum_{i=1}^{m_x} \|\mathbf{Z}_{i,:}\|_2$.

◇ **Group sparsity** In dynamic CT, one can naturally group the spatial variables (pixels) at each time instant, i.e., $\{\mathbf{x}^{(t)}\}_{t=1}^{n_t}$, although there are other possible ways of defining groups. In TRIPs-Py we impose group sparsity across the groups defined by the pixels of the gradient images for all the time instants, i.e., we take

$$\mathcal{R}(\mathbf{x}) = \sum_{\ell=1}^{n_s'} \left(\sum_{t=1}^{n_t} (\Psi_s \mathbf{x}^{(t)})_\ell^2 \right)^{1/2} = \|\Psi_s \mathbf{X}\|_{2,1}, \tag{48}$$

where $n_s' = (n_y - 1)n_x + (n_y - 1)n_x$ is the total number of pixels in the gradient images and, referring to the notations in (43), $\mathbf{X} = [\widehat{\mathbf{x}}^{(1)}, \ldots, \widehat{\mathbf{x}}^{(n_t)}] \in \mathbb{R}^{n_s \times n_t}$.

Code 4 gives examples of the usage within TRIPs-Py of the basic regularization operators that appear in the regularization functionals listed above, as well as the framelet operator described in Section 2.4. These are coded in the file `operators.py` under the directory `trips/utilities`. All such regularization functionals can be handled in the general framework of MMGKS, as described at the end of Section 2.2.2.

Code 4: Regularization operators

```
# Discretization of the first derivative operator in 1D
L = gen_first_derivative_operator(n)
# Discretization of the first derivative operator in 2D
L = gen_first_derivative_operator_2D(nx, ny)
# Spacetime derivative operator D_1
L = gen_spacetime_derivative_operator(nx, ny, nt)
# Framelet operator of the second level
W = create_framelet_operator(nx, ny, 2)
```

3.3.1 Emoji test problem

This example considers real data of an "emoji" phantom measured at the University of Helsinki [42]. We modify the data to determine two main limited angle problems as follows:

1. *Problem 1: 10 projection angles.* From the dataset `DataDynamic_128x30.mat` we generate the problems $\widehat{\mathbf{A}}^{(t)}\widehat{\mathbf{x}}^{(t)} + \widehat{\mathbf{e}}^{(t)} = \widehat{\mathbf{b}}^{(t)}$, $t = 1, 2, \ldots, 33$ where $\widehat{\mathbf{A}}^{(t)} \in \mathbb{R}^{2,170 \times 16,384}$ are defined by taking 217 fan-beam projections around only 10 equidistant angles in $[0, 2\pi)$. The forward operator $\mathbf{A}$ has size $71,610 \times 540,672$.
2. *Problem 2: 30 projection angles.* From the dataset `DataDynamic_128x60.mat` we still generate 33 static problems, with $\widehat{\mathbf{A}}^{(t)} \in \mathbb{R}^{6,510 \times 16,384}$ computed from taking 217 fan-beam projections around 30 equidistant angles in $[0, 2\pi)$. Hence the dynamic forward operator $\mathbf{A}$ has size $214,830 \times 540,672$.

For both cases, the ground truth $\mathbf{x}_{\text{true}}$ is not available, but photographs of the shapes scanned [42] are shown in Figure 6.

In TRIPs-Py, the main function to generate emoji data is `generate_emoji(dataset = your_dataset)` which allows the user to select the dataset to be either 30 or 60. Such function automatically downloads the data from the repository [2]. If a noise level is provided as an argument in `generate_emoji()`, then more Gaussian noise (on the top of the unknown noise already present in the recorded data) of the given level is added to the returned data. With `generate_emoji()` we generate data for solving both a sequence of static inverse problems of the form (44), where the operators are saved in $\mathbf{A}_{\text{seq}}$ and the sinograms are the columns of $\mathbf{B}$, and the dynamic problem of the form (45), with the blockdiagonal matrix $\mathbf{A}$ and the stacked sinograms $\mathbf{b}$. These are illustrated in Code 5.

[2] https://zenodo.org/records/1183532

Code 5: Emoji test problem

```
# Generate the emoji test problem with the dataset option 30
(F, d, Aseq, B, nx, ny, nt) = generate_emoji(dataset = 30)
# Generate the emoji test problem with the dataset option 60
(F, d, Aseq, B, nx, ny, nt) = generate_emoji(dataset = 60)
# generate a test problem by adding more noise to the real data
(F, d, Aseq, B, nx, ny, nt) = generate_emoji(dataset = 30, noise_level =
    0.001)
```

We run 100 iterations of MMGKS for solving both the static problems and the dynamic problem, with regularization parameter computed by GCV. The usage of the TRIPs-Py functions to solve the static problems is illustrated in Code 6, and the reconstructions obtained at time steps $t = 6, 14, 20, 26$ are shown in Figure 7, first row. The function `plot_recstructions_series()` can be used to display the reconstructions for both static and dynamic problems with the argument `dynamic = True` or `dynamic = False`, respectively. When displaying the results, we set negative solution entries to 0 as a post-processing step (note that this is different to applying nonnegativity constraints during the reconstructions as done, for instance, in [36, 37]). The usage of the TRIPs-Py functions to solve the dynamic problem is reported in Code 7, where we particularly focus on solvers that, within MMGKS, enforce (An)isotropic TV in space and anisotropic TV in time, as well as Group Sparsity.

More details about this test problem can be found in the demo `demo_dynamic_Emoji.ipynb`.

Code 6: Solve and plot a series of static emoji problems

```
# Define the regularization operator (only for space)
L = gen_first_derivative_operator_2D(nx, ny)
# For all problems, call MMGKS
for i in range(nt):
    b_vec = B[i].reshape((-1,1))
    (x_static_mmgks, info_mmgks) = MMGKS(Aseq[i], b_vec, L, pnorm=2, qnorm
    =1,projection_dim=1, n_iter = 100, regparam= 'gcv' , x_true=None,
    epsilon = 0.001)
    xx[i] = x_static_mmgks
# Plot static reconstructions
plot_recstructions_series(xx, (nx, ny, nt), dynamic = False, testproblem =
    'Emoji', geome_x = 1,geome_x_small = 0,  save_imgs= False, save_path='
    ./reconstruction/Emoji')
```

Figure 6: Emoji test problem. True images [42] at time instances $t = 6, 14, 20, 26$.

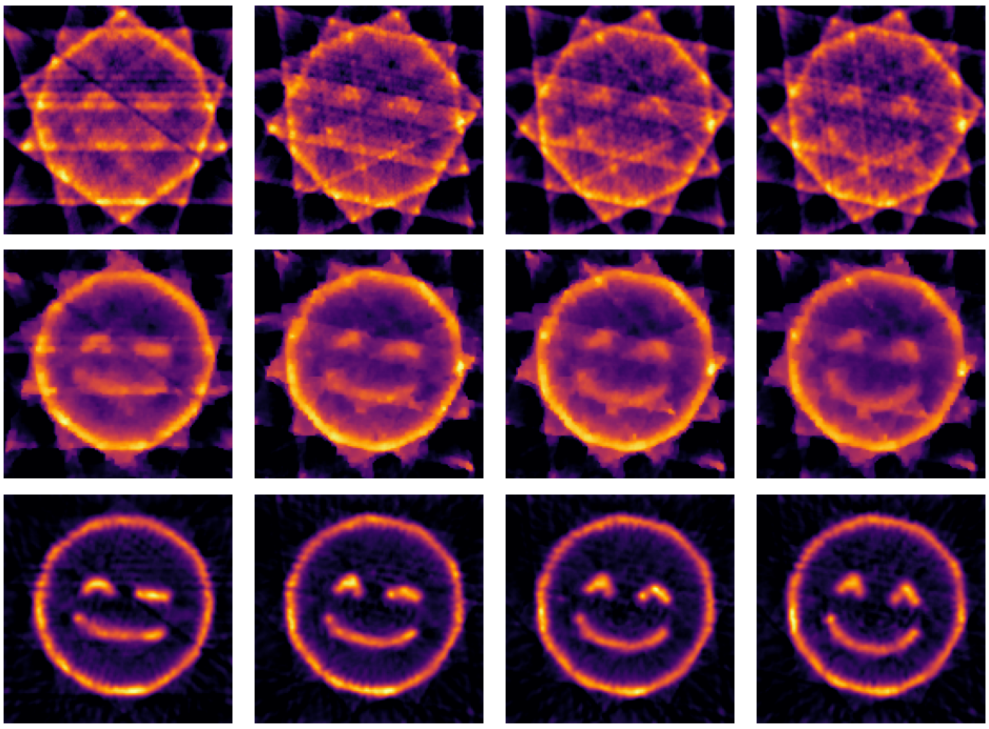


Figure 7: Emoji test problem. Reconstruction results with 10 projection angles. The first row shows the reconstructions with spatial anisotropic TV for the static problem, the second and third rows display the reconstructions for the dynamic problem solved with anisotropic and isotropic TV at time instances $t = 6, 14, 20, 26$, respectively, from left to right.

Code 7: Solve the dynamic Emoji with different regularizers

```
# Define the regularization operator (for space and time)
L = gen_spacetime_derivative_operator(nx, ny, nt)
data_vec = d.reshape((-1,1))
(x_mmgks, info_mmgks) = MMGKS(F, data_vec, L, pnorm=2, qnorm=1,
    projection_dim=1, n_iter = 100, regparam = 'gcv', x_true = None, delta
    = delta, epsilon = 0.001)
# Run TV_iso
(x_mmgks, info_mmgks) = MMGKS(F, data_vec, L, pnorm=2, qnorm=1,
    projection_dim=1, n_iter = 100, regparam = 'gcv', x_true = None, delta
    = delta, epsilon = 0.001, isoTV = 'isoTV', prob_dims = (nx, ny, nt)
# Run GS
(x_mmgks, info_mmgks) = MMGKS(F, data_vec, L, pnorm=2, qnorm=1,
    projection_dim=1, n_iter = 100, regparam = 'gcv', x_true = None, delta
    = delta, epsilon = 0.001, GS = 'GS', prob_dims = (nx, ny, nt)
# Plot dynamic reconstructions
plot_recstructions_series(x_mmgks, (nx, ny, nt), dynamic = True,
    testproblem = 'Emoji', geome_x = 1,geome_x_small = 0,  save_imgs= False
    , save_path='./reconstruction/Emoji')
```

3.3.2 STEMPO

Starting from the Spatio-TEmporal Motor-POwered (STEMPO) ground truth phantom from [43], within TRIPs-Py we generate both static and dynamic inverse problems, with both simulated and real data. The data for this example can be downloaded from the repository [3].

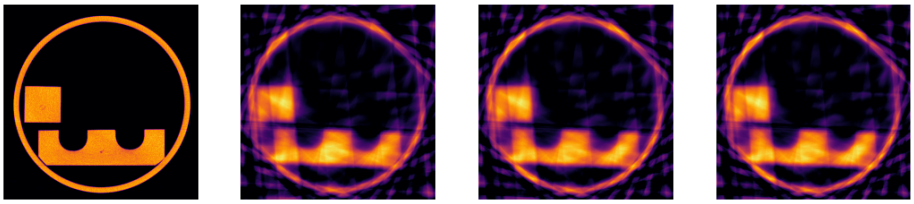
For the simulated data we consider the STEMPO ground truth phantom `stempo_ground_truth_2d_b4.mat`, which contains 360 images of size 560×560. From this dataset

[3] https://zenodo.org/records/8239013

we retain n_t images, chosen uniformly from 1 to 360 with a factor of 8, i.e., we choose the 1st, the 8th, ... up to the 360th image: these represent the ground truth at n_t time instances; n_t can be given in input by the user. Using the ASTRA toolbox [10] we generate the forward operators $\widehat{\mathbf{A}}^{(t)}$, $t = 1, 2, \ldots, n_t$, each defined with respect to the angles stored in n_t vectors of length 11, with 791 parallel rays departing from each angle. Each angle vector is generated by choosing 11 equispaced degree angles from $(5 * (t-1), 5 * (t-1) + 140)$, for $t = 1, 2, \ldots, n_t$, which are then converted to radian. For instance, for $n_t = 20$, the forward operators at each time instant are of size 8701×313600, while the dynamic forward operator is a block diagonal matrix of size 174020×6272000. We perturb each measured vectorized sinogram $\widehat{\mathbf{d}}^{(t)}_{\text{true}}$ with white Gaussian noise of level given in input by the user. , i.e., the noise vector $\mathbf{e}^{(t)}$ has mean zero and a rescaled identity covariance matrix. The true images $\mathbf{x}^{(t)}$ at time steps $n_t = 1, 10, 20, 30$ are shown in Figure 8. An illustration for both real and simulated gata generation is shown in Code 9. More details about this test problem can be found in the demo `demo_dynamic_Stempo.ipynb`.

Code 9: Stempo test problem

```
# Generate a simulated STEMPO test problem
(F, d, Aseq, B, nx, ny, nt, savedelta, truth) = generate_stempo(data_set =
    'simulation', data_thinning = 2, nt = 10)
# Generate a STEMPO test problem with real data
(F, d, Aseq, B, nx, ny, nt, savedelta, truth) = generate_stempo(data_set =
    'real', data_thinning = 2)
```



(a) (b) (c) (d)

Figure 8: STEMPO test problem. (a) True image, and reconstructions at time instance $t = 10$ by (b) static anisotropic TV, (c) Hybrid LSQR, (d) dynamic isotropic TV.

3.3.3 Cross Phantom

The last example considers real data of the cross phantom measures at the University of Helsinki [44]. We obtain the forward operator and the data from the file `DataDynamic_128x15.mat`, which can be downloaded from the repository [4]. The spatial resolution is 128×128 pixels, while the time resolution is 16. The CT sinogram that represents the available data consists of 16 time frames that are generated by measuring a 2D cross-section of the dynamic phantom that is built from an aluminum stick, a graphite stick, and candle wax. The measurements were collected by considering 15 projection angles at each time instance, with angles shifted by one degree from a given time instance to the next one, i.e., if the angles for time step t_1 are $[1, 15, 29, ..]$, then for time step t_2 they are $[2, 16, 30, ..]$. The measurement matrix that represents the forward operator obtained with a cone-beam geometry is a sparse matrix of size 33600×262144, and the sinogram matrix is represented as $\mathbf{B} \in \mathbb{R}^{140 \times 240}$, i.e., 15 projections on 16 images. The instance of the Cross Phantom test problem just described can be easily generated in TRIPs-Py as shown in Code 1Reconstructed images with MMGKS for the static inverse problems at time instances $t = 1, 5, 10, 15$ are shown in the first row of Figure 9. The second row of Figure 9 shows reconstructed images by solving the dynamic inverse problem with anisotropic TV at time instances $t = 1, 5, 10, 15$ from left to right, respectively. More details about this test problem can be found in the demo `demo_dynamic_CrossPhantom.ipynb`.

[4] https://zenodo.org/records/1446516

Code 10: CrossPhantom test problem

```
# Generate the CrossPhantom test problem
(F, d, Aseq, B, nx, ny, nt) = generate_crossPhantom(dataset = 15)
```

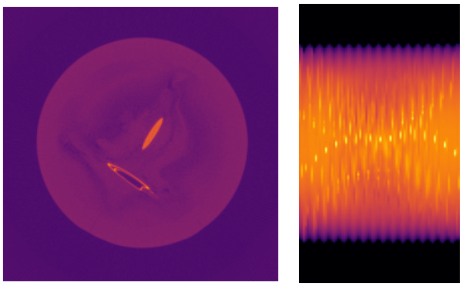


Figure 9: CrossPhantom test problem. Left frame: the high-resolution filtered back-projection reconstruction of the time-dependent cross phantom computed from 360 projections. Right frame: sinogram that contains measurements from 80 angels for 16 time instances together.

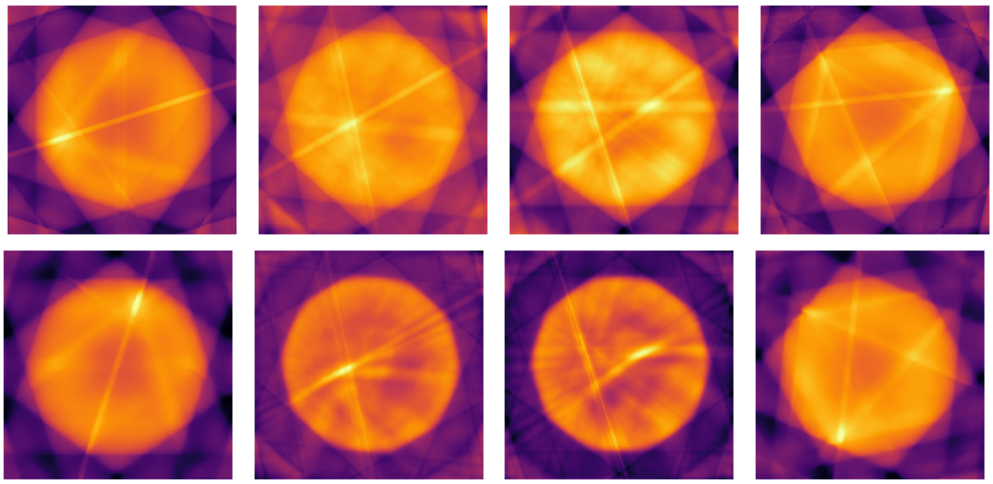


Figure 10: CrossPhantom test problem. Reconstructions with MMGKS for the static test problems (first row) and anisotropic TV for the dynamic inverse problems (second row) at time steps $t = 1, 5, 10, 15$, from left to right.

4 Conclusions and outlook

In this paper we describe TRIPs-Py, a Python package collecting solvers for small and large-scale discrete ill-posed inverse problems, and test problems. The package allows the user to easily set-up a test problem, with both simulated and real data, and employ built-in regularization methods to solve the inverse problem. Among solvers for small-scale problems, TRIPs-Py includes direct regularization methods such as truncated (G)SVD and Tikhonov. Among iterative regularization, TRIPs-Py includes methods based on both standard and generalized Krylov subspace methods. Among the test problems, special emphasis is given to a framework for handling dynamic simulated and real data. We envision that the user can provide their own data and use our solution methods within TRIPs-Py, as well as test their solution methods with the TRIPs-Py test problems.

This is a first public version of TRIPs-Py. Future developments will focus on adding new solution methods, functionalities, and improve existing data generation and visualisation tools. Specifically, in future versions of TRIPs-Py, we plan to:

- ⋄ Include solvers based on flexible Krylov subspace methods to, e.g., enforce sparsity on the desired solution; see [45].

- ◇ Include methods that can enforce constraints into the solutions; see, e.g., [46, 36, 37].
- ◇ Implement memory-aware iterative methods for edge-preserving and sparsity for solving large-scale and massive inverse problems; see [47].
- ◇ Add other strategies to automatically determine the regularization parameter(s).
- ◇ Add other instances of the existing deblurring test problems (e.g., a variety of PSFs) and tomography test problems (e.g., different scanning geometries).
- ◇ Add other test problems, such as hyperspectral imaging and spectral CT.
- ◇ Develop tools for high-order representation of dynamic imaging data.

Users are welcomed to contribute to TRIPs-Py by solution methods, functionalities, and/or test problems and data.

Acknowledgments

MP gratefully acknowledges support from the NSF under award No. 2202846. MP further acknowledges partial support from the NSF-AWM Mentoring Travel award. Both MP and SG acknowledge the Isaac Newton Institute for Mathematical Sciences, Cambridge, for the support and hospitality during the programme "Rich and Nonlinear Tomography - a multidisciplinary approach" (supported by EPSRC grant no EP/R014604/) where partial work on this paper was undertaken. SG would like to thank Ludovico Carozza for his advice on many Python functionalities. We would like to thank Tatiana Bubba for discussions about avoiding 'inverse crimes' in inverse problems.

References

[1] Boas DA, Brooks DH, Miller EL, DiMarzio CA, Kilmer M, Gaudette RJ, et al. Imaging the body with diffuse optical tomography. IEEE signal processing magazine. 2001;18(6):57-75.

[2] Miller EL, Abriola LM, Aghasi A. Environmental remediation and restoration: hydrological and geophysical processing methods. IEEE Signal Processing Magazine. 2012;29(4):16-26.

[3] Bennett AF, Chua BS, Leslie L. Generalized inversion of a global numerical weather prediction model. Meteorology and Atmospheric Physics. 1996;60(1):165-78.

[4] Hansen PC. Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems. Numer Algo. 1994;6(3):1-35.

[5] Gazzola S, Hansen PC, Nagy JG. IR Tools: a MATLAB package of iterative regularization methods and large-scale test problems. Numerical Algorithms. 2019;81(3):773-811.

[6] Nagy J, Palmer K, Perrone L. Iterative Methods for Image Deblurring: A MATLAB Object Oriented Approach. Numerical Algorithms. 2004;36(1):73-93.

[7] Hansen PC, Jørgensen JS. AIR Tools II: Algebraic iterative reconstruction methods, improved implementation. Numer Algo. 2018;79:107-37.

[8] Biguri A, Dosanjh M, Hancock S, Soleimani M. TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction. Biomed Phys Eng Express. 2016;2:055010.

[9] Jørgensen JS, Ametova E, Burca G, Fardell G, Papoutsellis E, Pasca E, et al. Core Imaging Library - Part I: a versatile Python framework for tomographic imaging. Phil Trans R Soc. 2021;A:3792020019220200192.

[10] Van Aarle W, Palenstijn WJ, De Beenhouwer J, Altantzis T, Bals S, Batenburg KJ, et al. The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography. Ultramicroscopy. 2015;157:35-47.

[11] et al AJ. odlgroup/odl: ODL 0.7.0. Zenodo; 2018. doi:10.5281/zenodo.1442734.

[12] Hansen PC. Discrete inverse problems: Insight and algorithms. SIAM; 2010.

[13] Hansen PC. Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion. SIAM; 1998.

[14] Saad Y. Iterative methods for sparse linear systems. SIAM; 2003.

[15] Chung J, Gazzola S. Computational methods for large-scale inverse problems: a survey on hybrid projection methods. arXiv preprint arXiv:210507221. 2023.

[16] Fenu C, Reichel L, Rodriguez G. GCV for Tikhonov regularization via global Golub–Kahan decomposition. Numerical Linear Algebra with Applications. 2016;23(3):467-84.

[17] Lewis B, Reichel L. Arnoldi–Tikhonov regularization methods. Journal of Computational and Applied Mathematics. 2009;226(1):92-102.

[18] Lampe J, Reichel L, Voss H. Large-scale Tikhonov regularization via reduction by orthogonal projection. Linear algebra and its applications. 2012;436(8):2845-65.

[19] Lanza A, Morigi S, Reichel L, Sgallari F. A generalized Krylov subspace method for $\ell_p - \ell_q$ minimization. SIAM Journal on Scientific Computing. 2015;37(5):S30-50.

[20] Pasha M, Saibaba AK, Gazzola S, Espanol MI, de Sturler E. Efficient edge-preserving methods for dynamic inverse problems. arXiv preprint arXiv:210705727. 2021.

[21] Golub GH, Van Loan CF. Matrix Computations, 4th ed. Johns Hopkins University Press, Baltimore; 2013.

[22] Eldén L. A weighted pseudoinverse, generalized singular values, and constrained least squares problems. BIT Numerical Mathematics. 1982;22(4):487-502.

[23] Hansen PC, Nagy JG, O'leary DP. Deblurring images: matrices, spectra, and filtering. SIAM; 2006.

[24] Björck Å. Numerical Methods in Matrix Computations. Texts in Applied Mathematics. Springer International Publishing; 2014.

[25] Arnoldi WE. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of applied mathematics. 1951;9(1):17-29.

[26] Lanczos C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. United States Governm. Press Office Los Angeles, CA; 1950.

[27] Lange K. MM optimization algorithms. SIAM; 2016.

[28] Rodriguez P, Wohlberg B. An efficient algorithm for sparse representations with ℓ_p data fidelity term. In: Proceedings of 4th IEEE Andean Technical Conference (ANDESCON); 2008. .

[29] Huang G, Lanza A, Morigi S, Reichel L, Sgallari F. Majorization–minimization generalized Krylov subspace methods for $\ell_p - \ell_q$ optimization applied to image restoration. BIT Numerical Mathematics. 2017;57(2):351-78.

[30] Bach F, Jenatton R, Mairal J, Obozinski G. Optimization with Sparsity-Inducing Penalties. Foundations and Trends in Machine Learning. 2012;4(1):1-106.

[31] Reichel L, Shyshkov A. A new zero-finder for Tikhonov regularization. BIT Numerical Mathematics. 2008;48(3):627-43.

[32] Gazzola S, Landman MS. Krylov methods for inverse problems: Surveying classical, and introducing new, algorithmic approaches. Mitteilungen der Gesellschaft für Angewandte Mathematik und Mechanik. 2020;43(4).

[33] Buccini A, Reichel L. An ℓ^2-ℓ^q Regularization Method for Large Discrete Ill-Posed Problems. Journal of Scientific Computing. 2019;78:1526-49.

[34] Novati P, Russo MR. A GCV-based Arnoldi-Tikhonov regularization method. BIT. 2014;54:501-21.

[35] Buccini A, Reichel L. Generalized cross validation for $\ell_p - \ell_q$ minimization. Numer Algor. 2021;88:1595–1616.

[36] Buccini A, Pasha M, Reichel L. Modulus-based iterative methods for constrained $\ell_p - \ell_q$ minimization. Inverse Problems. 2020;36(8):084001.

[37] Buccini A, Pasha M, Reichel L. Linearized Krylov subspace Bregman iteration with nonnegativity constraint. Numerical Algorithms. 2020:1-24.

[38] Cai JF, Osher S, Shen Z. Linearized Bregman iterations for frame-based image deblurring. SIAM Journal on Imaging Sciences. 2009;2:226-52.

[39] Mueller JL, Siltanen S. Linear and nonlinear inverse problems with practical applications. SIAM; 2012.

[40] Hansen PC, Jørgensen JS, Lionheart WRB. Computed Tomography: Algorithms, Insight, and Just Enough Theory. SIAM, Philadelphia; 2021.

[41] Lan S, Pasha M, Li S. Spatiotemporal Besov Priors for Bayesian Inverse Problems. arXiv preprint arXiv:230616378. 2023.

[42] Meaney A, Purisha Z, Siltanen S. Tomographic X-ray data of 3D emoji. arXiv preprint arXiv:180209397. 2018.

[43] Heikkilä T. STEMPO–dynamic X-ray tomography phantom. arXiv preprint arXiv:220912471. 2022.

[44] Latva-Äijö S, Meaney A, Siltanen S. Tomographic X-ray data of 3D cross phantom. arXiv preprint arXiv:180900166. 2018.

[45] Gazzola S, Nagy JG, Sabaté Landman M. Iteratively Reweighted FGMRES and FLSQR for Sparse Reconstruction. SIAM Journal on Scientific Computing. 2021;(0):S47-69.

[46] Gazzola S, Wiaux Y. Fast nonnegative least squares through flexible Krylov subspaces. SIAM J Sci Comput. 2017;39:A655-79.

[47] Pasha M, de Sturler E, Kilmer ME. Recycling MMGKS for large-scale dynamic and streaming data. arXiv preprint arXiv:230915759. 2023.