Pocket Guide to Solve Inverse Problems with GlobalBioIm

Emmanuel Soubies[1], Ferréol Soulez[2], Michael T. McCann[1], Thanh-an Pham[1], Laurène Donati[1], Thomas Debarre[1], Daniel Sage[1], and Michael Unser[1]

[1]Biomedical Imaging Group, EPFL, Lausanne, Switzerland.
[2]Univ Lyon, Univ Lyon1, Ens de Lyon, CNRS, Centre de Recherche Astrophysique de Lyon UMR5574, F-69230, Saint-Genis-Laval, France.

E-mail: `michael.unser@epfl.ch`

Abstract. `GlobalBioIm` is an open-source MATLAB® library for solving inverse problems. The library capitalizes on the strong commonalities between forward models to standardize the resolution of a wide range of imaging inverse problems. Endowed with an operator-algebra mechanism, `GlobalBioIm` allows one to easily solve inverse problems by combining elementary modules in a lego-like fashion. This user-friendly toolbox gives access to cutting-edge reconstruction algorithms, while its high modularity makes it easily extensible to new modalities and novel reconstruction methods. We expect `GlobalBioIm` to respond to the needs of imaging scientists looking for reliable and easy-to-use computational tools for solving their inverse problems. In this paper, we present in detail the structure and main features of the library. We also illustrate its flexibility with examples from multichannel deconvolution microscopy.

1. Introduction

1.1. Inverse Problems in Imaging

Imaging is a fundamental tool for biological research, medicine, and astrophysics. Medical imaging systems are essential for modern diagnosis, while the latest generation of microscopes and telescopes provide images with unprecedented resolution. This imaging revolution is driven, in part, by the current shift towards computational imaging that sees optics and computing combine to bypass many limitations of conventional systems.

These computational imaging techniques rely on the deployment of sophisticated algorithms to reconstruct a d-dimensional continuously defined object of interest $f \in L_2(\mathbb{R}^d)$ from discrete measurements $\mathbf{g} \in \mathbb{R}^M$ recorded by a given imaging system. These quantities are linked according to

$$\mathbf{g} = \mathcal{H}\{f\} + \mathbf{n}, \tag{1}$$

where $\mathcal{H} : L_2(\mathbb{R}^d) \to \mathbb{R}^M$ is an operator that models the imaging system. This operator, which might be linear or not, maps the continuously defined object to

Table 1: Broad class of imaging models defined as the composition of elementary operators. Here, the basic constituents include weighting, windowing, or modulation ($\mathcal{W}$), convolution ($\mathcal{C}$), Fourier transform ($\mathcal{F}$), integration (Σ), rotation ($\mathcal{R}_\theta$), and sampling ($\mathcal{S}$). The Radon transform (for CT and cryo-EM) is written as the composition $\Sigma \circ \mathcal{R}_\theta$ of a rotation and an integration. Similarly, the Laplace transform (for TIRF) is expressed as the composition $\Sigma \circ \mathcal{W}$ of a weighting (decaying exponential) and an integration. Note that these elementary operators might differ for each modality (*e.g.,* using different kernels for the convolution operators), but their construction stays identical.

Imaging modality	Forward model $\mathcal{H}$
X-ray computed tomography (CT)	$\mathcal{S} \circ \Sigma \circ \mathcal{R}_\theta$
Conventional fluorescent microscopy	$\mathcal{S} \circ \mathcal{C}$
Structured-illumination microscopy (SIM)	$\mathcal{S} \circ \mathcal{C} \circ \mathcal{W}$
Total internal reflection fluorescence (TIRF)	$\mathcal{S} \circ \Sigma \circ \mathcal{W}$
Optical diffraction tomography (ODT, first Born)	$\mathcal{S} \circ \mathcal{C} \circ \mathcal{W}$
Cryo-electron tomography (Cryo-EM)	$\mathcal{S} \circ \mathcal{C} \circ \Sigma \circ \mathcal{R}_\theta$
Magnetic resonance imaging (MRI)	$\mathcal{S} \circ \mathcal{F} \circ \mathcal{W}$

discrete noiseless measurements. Finally, $\mathbf{n} \in \mathbb{R}^M$ is an error term, which is often considered to be random.

To numerically solve the inverse problem and recover f, it is necessary to discretize both f and the operator $\mathcal{H}$. This leads to the discrete imaging model $\mathbf{g} = \mathbf{H}\{\mathbf{f}\} + \mathbf{n}$, with $\mathbf{f} \in \mathbb{R}^N$ and $\mathbf{H}\{\,\cdot\,\} : \mathbb{R}^N \to \mathbb{R}^M$.

The classical approach to address this inverse problem and recover an estimated solution $\hat{\mathbf{f}}$ consists in solving

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f} \in \mathbb{R}^N} \big(\mathcal{D}(\mathbf{H}\{\mathbf{f}\}, \mathbf{g}) + \lambda \mathcal{R}(\mathbf{f}) \big). \tag{2}$$

There, $\mathcal{D} : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ measures the discrepancy between the forward model $\mathbf{H}\{\mathbf{f}\}$ and the measurements $\mathbf{g}$ (*i.e.*, the data fidelity), while $\mathcal{R} : \mathbb{R}^N \to \mathbb{R}$ enforces specific regularity constraints on the solution (*e.g.,* spatial smoothness, or nonnegativity). The balance between the data fidelity and regularization terms is controlled by the scalar parameter $\lambda > 0$.

1.2. Unifying Framework for Solving Inverse Problems

The forward models associated with most of the commonly used imaging modalities share important structural properties. This similarity is not surprising since many imaging systems are governed by the same physical principles (*e.g.,* the wave equation). We express in Table 1 the forward models of a wide range of imaging modalities in terms of a limited number of elementary constituents.

By capitalizing on these strong commonalities, the open-source MATLAB® library `GlobalBioIm` simplifies, unifies, and standardizes the resolution of inverse problems given by (2). Hence, the `GlobalBioIm` toolbox gives access to state-of-the-art reconstruction algorithms usable in a wide range of imaging applications. Its design is modular, with three main types of entities: forward models, cost functions, and solvers.

This permits the user to modify each component independently, which is crucial for the handling of a variety of imaging models and solvers within a common framework. This modularity also makes `GlobalBioIm` easily extensible to new modalities and novel reconstruction methods.

`GlobalBioIm` is distributed as an open-source MATLAB® software. We expect it to respond to the needs of imaging scientists looking for reliable and easy-to-use computational tools for the reconstruction of their images. We also believe that `GlobalBioIm` will be of interest to developers of algorithms who focus on the mathematical and algorithmic details of the reconstruction methods.

The present paper provides a functional description of the structure and the key components of the `GlobalBioIm` library. It completes and extends our previous brief communication [38]. For a detailed technical documentation, we refer the reader to an online documentation (`http://bigwww.epfl.ch/algorithms/globalbioim/`).

1.3. Related Work

The development of open-source libraries/toolboxes in imaging sciences has received considerable attention during the past two decades. The majority of existing softwares for solving inverse problems are dedicated to specific modalities, with various degrees of sophistication. Moreover, they cover the whole panel of programming languages.

There exists a large number of toolboxes dedicated to tomographic reconstruction for x-ray computed tomography, positron-emission tomography, single-photon-emission computed tomography, or (scanning) transmission electron microscopy. These include among others ASTRA [39], CASToR [24], CONRAD [22], RTK [29], STIR [37], or TIGRE [6].

For fluorescence microscopy, DeconvolutionLab [31] provides a set of deconvolution methods that range from naive inverse filters to more sophisticated iterative approaches. The emergence of superresolution fluorescence microscopy techniques has also promoted the development of toolboxes tailored for their specific inverse problems. For instance, FairSIM [26] and Simtoolbox [18] are dedicated to the reconstruction of structured-illumination microscopy data. For single-molecule localization microscopy, one can find dedicated localization plugins such as SMAP [21] and ThunderSTORM [27].

Although dedicated to specific physical models, the aforementioned toolboxes generally rely on similar reconstruction methods, ranging from Wiener filtering to advanced regularized iterative algorithms. Conversely, libraries that are generic have recently also been designed to handle multiple imaging modalities. The LazyAlgebra toolbox [35] provides an operator-algebra mechanism in Julia that can be combined with optimization packages for solving inverse problems. More complete libraries such as AIR Tools (MATLAB®) [17], IR Tools (MATLAB®) [15], or TiPi (Java™) [36] provide elements for the implementation of forward models, as well as iterative solvers to tackle the associated inverse problems. The disadvantage of these toolboxes is that the optimization algorithms they provide are generally implemented to minimize a specific functional, thus limiting their modularity.

In contrast, `GlobalBioIm` provides a fully modular environment where one can not only easily combine functionals and operators to define the loss to be minimized in (2), but also benefit from a variety of solvers. This philosophy is shared by a few other toolboxes with different programming languages, such as the Operator Discretization Library (in Python) [1] and the Rice Vector Library (in C++) [28].

2. General Philosophy and Organization

When tasked with the design of a reconstruction algorithm for a new imaging problem, the common practice follows a three steps process.

(i) Modelization of the acquisition system ⇒ Implementation of $\mathbf{H}$.

(ii) Formulation of the reconstruction as an optimization problem (*i.e.,* the cost function) ⇒ Choice of $\mathcal{D}$ and $\mathcal{R}$ in (2).

(iii) Deployment of an optimization method ⇒ Choice of a solver for (2).

This standard pipeline motivates the organization of `GlobalBioIm` around three dedicated main abstract classes: `LinOp`, `Cost`, and `Opti`. Because linear operators and cost functions both belong to the larger mathematical class of maps, the `LinOp` and `Cost` classes are defined as particular instances of a generic abstract class `Map`. The latter also allows for a proper inclusion of nonlinear operators. The organization of the library is illustrated in Figure 1. It is guided by five general principles.

- **Modularity.** All objects are defined as individual modules that can be combined to generate a particular reconstruction workflow. Each building block can thus be easily changed to define new reconstruction pipelines.
- **Flexibility.** The constraints to fulfill during implementation are few. New objects can easily be plugged into the framework of `GlobalBioIm`.
- **Abstraction.** The four abstract classes (`LinOp`, `Cost`, `Opti`, `Map`) define a limited set of attributes and methods that are shared by their derived classes (*i.e.,* subclasses). This constitutes a common guideline for the implementation of subclasses. Moreover, generic concepts—basically, interactions between classes—are implemented at the level of the abstract classes and benefit directly to all subclasses.
- **Readability.** Reconstruction scripts are written in a way that mimics equations in scientific papers, hence keeping a simple connection between theory and implementation.
- **User-friendliness.** The definition (or update) of a new subclass only requires one to create (or edit) a single file. Moreover, the usage of `GlobalBioIm` does not require one to understand advanced computing concepts.

3. Abstract Classes

We now present the four abstract classes that build up the skeleton of the `GlobalBioIm` library. The methods within these abstract classes are prototypes that have to be implemented in derived classes. There are exceptions for some generic concepts (*e.g.,* the chain rule) that are directly implemented in the abstract classes. For the sake of conciseness, we only review here the key attributes and methods of those classes. An exhaustive list of those features can be found in the online documentation (`http://bigwww.epfl.ch/algorithms/globalbioim/`) within the sections "List of Methods" and "List of Properties".

3.1. `Map` *Class*

The abstract `Map` class defines the basic attributes and methods of an operator $\mathbf{H} : \mathbb{R}^N \to \mathbb{R}^M$. These include, at the very minimum, the input size N, the output size M, and the method `apply` that computes $\mathbf{g} = \mathbf{H}\{\mathbf{f}\}$ for a given $\mathbf{f} \in \mathbb{R}^N$.

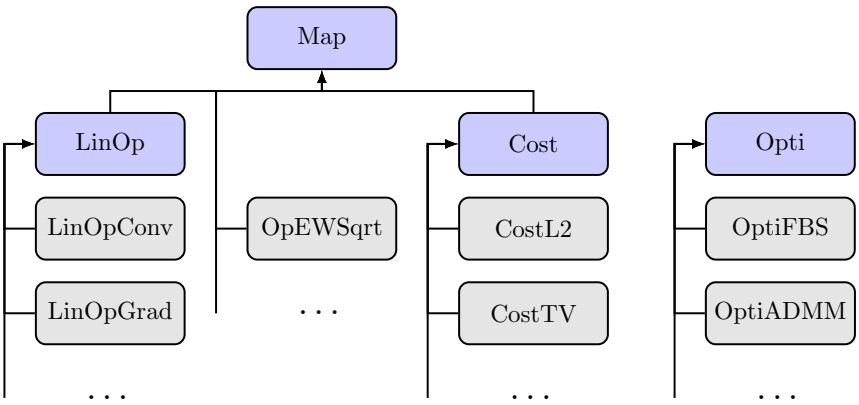

Figure 1: Hierarchy of classes in `GlobalBioIm`. Abstract classes are represented in blue and derived classes in gray. Nonlinear operators (such as the element-wise square-root `OpEWSqrt`) directly inherit from the abstract `Map` class.

In addition, because optimization algorithms may require the differentiation of the objective function in (2), the `Map` class defines the method `applyJacobianT`. Given $\mathbf{v} \in \mathbb{R}^M$ and $\mathbf{f} \in \mathbb{R}^N$, this method computes $\mathbf{u} = [\mathbf{J}_{\mathbf{H}}\{\mathbf{f}\}]^T\mathbf{v}$, where $\mathbf{J}_{\mathbf{H}}\{\mathbf{f}\} \in \mathbb{R}^{M\times N}$ is the Jacobian matrix of $\mathbf{H}$ (assuming that the latter is differentiable). It is formed out of the first-order partial derivatives of the operator $\mathbf{H}$, with

$$[\mathbf{J}_{\mathbf{H}}\{\mathbf{f}\}]_{m,n} = \frac{\partial \mathbf{H}_m}{\partial \mathbf{f}_n}, \tag{3}$$

where $\mathbf{H}_m : \mathbb{R}^N \to \mathbb{R}$ is such that $\mathbf{H} = [\mathbf{H}_1, \dots, \mathbf{H}_M]^T$. Similarly, for invertible maps, the method `applyInverse` allows one to compute $\mathbf{f} = \mathbf{H}^{-1}\{\mathbf{g}\}$ for $\mathbf{g} \in \mathbb{R}^M$.

In addition to the "apply"-type methods, the `Map` class provides prototype methods prefixed by "make". These can be implemented in derived classes to create new instances of `Map` objects that are related to $\mathbf{H}$. For instance, the method `makeInversion` returns a `Map` object that corresponds to $\mathbf{H}^{-1}$.

The prototype methods are also used to overload the MATLAB® operators "$*$" (`mtimes`), "$+$" (`plus`), and "$-$" (`minus`). Hence, the composition between two `Map` objects can be specified easily as `H = H1 * H2`. This will execute the method `makeComposition` of `H1` with `H2` as its argument. By default, the resulting `H` will be a `MapComposition` object that benefits from the generic implementations (*e.g.,* successive calls for `apply`, chain rule for `applyJacobianT`) provided in the `MapComposition` class.

Similarly, the operators "$+$" and "$-$" are associated to the `MapSummation` class. It is noteworthy to mention that this default behavior can be specialized in derived classes with a proper implementation of the "make" methods. This results in automatic simplifications, as described in Section 4.2.

3.2. `LinOp` *Class*

A particular class of map objects contains linear operators $\mathbf{H} : \mathbb{R}^N \to \mathbb{R}^M$ that satisfy

$$\mathbf{H}\{\alpha\mathbf{f} + \mathbf{g}\} = \alpha\mathbf{H}\{\mathbf{f}\} + \mathbf{H}\{\mathbf{g}\} \tag{4}$$

for all scalar α and vectors $\mathbf{f} \in \mathbb{R}^N$ and $\mathbf{g} \in \mathbb{R}^N$. Linear operators are generally represented as a matrix $\mathbf{H} \in \mathbb{R}^{M\times N}$. They are widely used in practice to model imaging systems. In addition to being good approximators, they lead to convex optimization problems for which there exist efficient solvers. An important subclass is formed by the convolution operators that are implemented very efficiently using FFTs. All this motivates the definition of the `LinOp` class, which inherits from the attributes and the methods of `Map` while defining novel ones.

For linear operators, the transposed Jacobian matrix $[\mathbf{J}_{\mathbf{H}}\{\mathbf{f}\}]^T$ is independent of $\mathbf{f}$ and is equal to the adjoint operator $\mathbf{H}^T$. Thus, the `LinOp` class provides the method `applyAdjoint` that computes $\mathbf{u} = \mathbf{H}^T\mathbf{v}$ for $\mathbf{v} \in \mathbb{R}^M$ and is directly used to implement the method `applyJacobianT`. Hence, to allow a `LinOp` to be differentiated only requires implementation of `applyAdjoint`, rather than `applyJacobianT`. In keeping with the aforementioned philosophy, the companion method `makeAdjoint` allows one to instantiate a new `LinOp` corresponding to the adjoint $\mathbf{H}^T$.

For least-squares minimization, the normal operators $\mathbf{H}^T\mathbf{H}$ and $\mathbf{H}\mathbf{H}^T$ are at the core of many optimization algorithms. Hence, the methods `applyHtH` and `applyHHt` as well as their "make" counterparts are defined in the `LinOp` class. They can be implemented in derived classes to provide implementations that are faster than the default successive application of $\mathbf{H}$ and $\mathbf{H}^T$. This is particularly useful when $\mathbf{H}^T\mathbf{H}$ turns out to be a convolution, as is the case for deconvolution, cryo-electron microscopy [13, 41], and x-ray computed tomography [23].

3.3. `Cost` *Class*

Cost functions are mappings for which $M = 1$ (*i.e.,* $\mathcal{J} : \mathbb{R}^N \to \mathbb{R}$). Hence, the abstract `Cost` class inherits from all the attributes and methods defined by the `Map` class. However, the `Cost` class also defines new attributes and methods that are specific to cost functions. For instance, the method `applyProx` is dedicated to the computation of the proximity operator of $\mathcal{J}$, which is required for a broad range of optimization algorithms. It is defined by [25] as

$$\text{prox}_{\mathcal{J}}(\mathbf{z}) = \arg\min_{\mathbf{f}\in\mathbb{R}^N} \left(\frac{1}{2}\|\mathbf{f} - \mathbf{z}\|_2^2 + \mathcal{J}(\mathbf{f})\right). \tag{5}$$

The method `applyGrad` computes the gradient $\boldsymbol{\nabla}\mathcal{J}$ of the functional $\mathcal{J}$. Similarly to the method `applyAdjoint` for `LinOp`, `applyGrad` can be seen as an alias for the method `applyJacobianT`. This ensures consistency with the standard terminology employed in scientific publications.

3.4. `Opti` *Class*

The last abstract class `Opti` is a prototype for optimization algorithms. Given a cost function $\mathcal{J}$ resulting from the composition/addition of `Map`, `LinOp`, and `Cost` objects, the `run` method of the `Opti` class implements a general iterative scheme to minimize

$\mathcal{J}$. It includes calls to the methods `initialize`, `doIteration`, and `updateParams`, which are implemented in every derived class.

The method `initialize` performs the computations required prior to starting the main loop of the iterative scheme. Then, `doIteration` is executed at each iteration, preceded by a call to `updateParams` that modifies the parameters of the algorithm (*e.g.,* by modifying the step size in a descent method).

The convergence of the algorithm is monitored during the optimization using a `TestCvg` object (set as an attribute of the `Opti` object). `GlobalBioIm` contains various `TestCvg` classes that implement different convergence criteria (*e.g.,* `TestCvgStepRelative`, `TestCvgCostRelative`). They can also be combined using the class `TestCvgCombine`. Finally, the verbose output is controlled by an `OutputOpti` object (again, set as an attribute of the `Opti` object). Hence, one can easily tune the information displayed and saved during iteration by defining a custom `OutputOpti` class.

4. Key Features of the Library

In this section, we highlight some of the most remarkable features of `GlobalBioIm`, which are intended to simplify the development process.

4.1. Interface and Core Methods

`Map`, `LinOp`, and `Cost` classes contain two types of methods, which come in pairs. *Interface methods* are only implemented in abstract classes and cannot be overridden in derived classes (sealed methods). However, they can be executed by an instantiated object of the class. On the other hand, *core methods* are not implemented in abstract classes, but in derived classes only. In addition, they cannot be executed by an instantiated object (private methods).

This scheme allows for the separation of preprocessing computations, which are common to all derived classes, from the core computations of the method, which are class-dependent. When executed, an interface method checks that inputs are the correct size prior to executing the associated core method. Interface methods are also used to manage the memoize mechanism (see Section 4.3).

From a user viewpoint, only core methods matter. They have to be implemented in derived classes without having to deal with input checking and the memoize mechanism. In the library, the core methods differ from the interface methods by the suffix "_" (*e.g.,* `apply_` versus `apply`).

4.2. Composition of Operators and Automatic Simplification

Up to now, the reader may wonder why it is useful to allow "make"-type methods to be overloaded in derived classes. Actually, these methods are the key ingredients for the automatic simplification mechanism deployed by `GlobalBioIm`. When compositions between maps occur, they allow for the instantiation of specific classes instead of the default generic classes such as `MapComposition`, `MapInversion`, or `MapSummation`. Consequently, the resulting object generally enjoys faster implementations.

To illustrate this feature, let us consider a convolution operator $\mathbf{H}$. Its adjoint $\mathbf{H}^T$ and the normal operator $\mathbf{H}^T\mathbf{H}$ turn out to be convolution operators as well, whose kernels can be precomputed from that of $\mathbf{H}$. Hence, the `LinOpConv` class implements

the `makeAdjoint_` and `makeHtH_` methods by instantiating a new `LinOpConv` with the adequate kernel.

```
function M = makeAdjoint_(this)
    % Reimplemented from parent class :class:'LinOp'.
    M=LinOpConv(conj(this.mtf),this.isReal,this.index);
end
function M = makeHtH_(this)
    % Reimplemented from parent class :class:'LinOp'.
    M=LinOpConv(abs(this.mtf).^2,this.index);
end
```

Because `makeAdjoint_` and `makeHtH_` are used to overload the operators "'" and "∗", respectively, we obtain the following automatic simplification.

```
>> H=LinOpConv(fftn(psf));
>> L=H'*H
L =
  LinOpConv with attributes:
       mtf: [256x256 double]
       ...
```

There are many more examples of "make" methods in `GlobalBioIm` (see also the methods `plus_` and `mpower_`). For instance, the `LinOpConv` class provides the following implementation for the method `plus_` (which is used to overload "+").

```
function M = plus_(this,G)
    % Reimplemented from parent class :class:'LinOp'.
    if isa(G,'LinOpDiag') && G.isScaledIdentity     % If sum with a constant diagonal operator
        M=LinOpConv(G.diag+this.mtf,this.isReal,this.index);
    elseif isa(G,'LinOpConv')                       % If sum with a convolution operator
        M=LinOpConv(this.mtf+G.mtf,this.isReal,this.index);
    else                                            % Otherwise call superclass plus_ method
        M=plus_@LinOp(this,G);
    end
end
```

This allows the following simplification.

```
>> H=LinOpConv(fftn(psf));
>> I=LinOpIdentity(H.sizein);
>> L=H'*H + I
L =
  LinOpConv with attributes:
       mtf: [256x256 double]
       ...
```

We conclude this section with two examples that demonstrate the relevance of this automatic simplification mechanism.

Example 4.1 (Proximity operator with semiorthogonal linear transform). Let $\mathcal{J}$ be a lower semicontinuous convex functional and $\mathbf{L}$ be a semiorthogonal linear operator (*i.e.,* $\mathbf{L}\mathbf{L}^T = \nu\mathbf{I}$ for $\nu > 0$). Then, as demonstrated in [11, Lemma 2.4], the proximity operator of $\alpha\mathcal{J}(\mathbf{L}\,\cdot\,)$ is given by

$$\mathrm{prox}_{\alpha\mathcal{J}(\mathbf{L}\,\cdot\,)}(\mathbf{z}) = \mathbf{z} + \nu^{-1}\mathbf{L}^T\left(\mathrm{prox}_{\nu\alpha\mathcal{J}}(\mathbf{L}\mathbf{z}) - \mathbf{L}\mathbf{z}\right). \tag{6}$$

Due to the automatic simplification mechanism of `GlobalBioIm`, one can easily verify whether $\mathbf{L}$ is semiorthogonal in the constructor of the `CostComposition` class ($\mathbf{L} \rightarrow$ `this.H2`).

```
T=this.H2*this.H2';                                          % Build L'*L
if isa(T,'LinOpDiag') && T.isScaledIdentity && T.diag>0      % Check if scaled identity
    this.isH2SemiOrtho=true;
    this.nu=T.diag;
end
```

Then, (6) is exploited in the implementation of the `applyProx_` method of the `CostComposition` class ($\mathcal{J} \rightarrow$ `this.H1`).

```
function x=applyProx_(this,z,alpha)
    if this.isConvex && this.isH2LinOp && this.isH2SemiOrtho
        H2z=this.H2*z;
        x = z + 1/this.nu*this.H2.applyAdjoint(this.H1.applyProx(H2z,alpha*this.nu)-H2z);
    else
        x = applyProx_@Cost(this,z,alpha);
    end
end
```

As a result, the composition of a `Cost` object that has an implementation of its proximity operator with a semi-orthogonal `LinOp` object automatically leads to a `CostComposition` object that has an implementation of the `applyProx_` method.

Example 4.2 (Woodbury matrix identity). Let $\mathcal{J}(\mathbf{f}) = \frac{1}{2}\|\mathbf{SHf} - \mathbf{g}\|_2^2$, where $\mathbf{S}$ is a downsampling operator and $\mathbf{H}$ is a convolution operator. It follows from (5) that

$$\begin{aligned}\operatorname{prox}_{\alpha\mathcal{J}}(\mathbf{u}) &= \left(\alpha\mathbf{H}^T\mathbf{S}^T\mathbf{SH} + \mathbf{I}\right)^{-1}\left(\alpha\mathbf{H}^T\mathbf{S}^T\mathbf{g} + \mathbf{u}\right) \\ &= \left(\mathbf{I} - \alpha\mathbf{H}^T\mathbf{S}^T\left(\mathbf{I} + \alpha\mathbf{SHH}^T\mathbf{S}^T\right)^{-1}\mathbf{SH}\right)\left(\alpha\mathbf{H}^T\mathbf{S}^T\mathbf{g} + \mathbf{u}\right), \qquad (7)\end{aligned}$$

where the Woodbury matrix identity [16] is used to get (7). Moreover, it turns out that $\alpha\mathbf{SHH}^T\mathbf{S}^T$ is a convolution operator [33, Lemma A.3] and, thus, that $\left(\mathbf{I} + \alpha\mathbf{SHH}^T\mathbf{S}^T\right)$ can easily be inverted in the Fourier domain. In order to apply (7), the specification of $\alpha\mathbf{SHH}^T\mathbf{S}^T$ as a convolution is implemented in `GlobalBioIm`. This is done in the `makeComposition_` method of `LinOpDownsample`, which returns a `LinOpConv` object when appropriate.

```
function G = makeComposition_(this, H)
    % Reimplemented from parent class :class:`LinOp`
    % ...
    if isa(H, 'LinOpComposition')
        if isa(H.H2,'LinOpAdjoint') && isequal(H.H2.TLinOp,this)
            if isa(H.H1, 'LinOpConv')
                P=LinOpSumPatches(this.sizein,this.sizein./this.df);
                G = LinOpConv(P*H.H1.mtf/prod(this.df),H.H1.isReal);
    % ...
```

As a result, $\left(\mathbf{I} + \alpha\mathbf{SHH}^T\mathbf{S}^T\right)$ is identified as being an invertible operator, and (7) can be directly implemented as follows.

```
H=LinOpConv(fftn(psf)); S=LinOpDownsample(H.sizein,[2,2]); fwd=S*H; % Forward model
In=LinOpIdentity(S.sizein); Im=LinOpIdentity(S.sizeout);          % Identity operators
prox=(In - alpha*fwd'*(Im + alpha*(fwd*fwd'))^(-1)*fwd)*(alpha*fwd'*g+u);
```

A complete script (TestProxL2DownSampledConv) in which this example is implemented can be found in the folder Cost/Tests/ of the `GlobalBioIm` library.

4.3. The Memory versus Computational Cost Dilemma

The computation and storage of fixed quantities can significantly accelerate iterative reconstruction methods. For instance, let us consider the least-squares functional $\mathcal{J}(\mathbf{f}) = \frac{1}{2}\|\mathbf{H}\mathbf{f} - \mathbf{g}\|_2^2$, where $\mathbf{H}$ is a convolution operator. Then, the minimization of $\mathcal{J}$ by a gradient-descent algorithm requires the evaluation of

$$\boldsymbol{\nabla}\mathcal{J}(\mathbf{f}) = \mathbf{H}^T(\mathbf{H}\mathbf{f} - \mathbf{g}) \tag{8}$$

$$= \mathbf{H}^T\mathbf{H}\mathbf{f} - \mathbf{H}^T\mathbf{g} \tag{9}$$

at each iteration. The computational burden of this operation is directly related to the implementation strategy. The formulation in (8) requires the evaluation of both $\mathbf{H}$ and $\mathbf{H}^T$, leading to the overall cost of two FFTs plus two iFFTs. Instead, since $\mathbf{H}^T\mathbf{H}$ turns out to be a convolution in this example, the formulation in (9) opens the door to a faster computation of $\boldsymbol{\nabla}\mathcal{J}$. The price to pay, however, is storage for the quantity $\mathbf{H}^T\mathbf{g}$. Imposing one of the above implementations to users could lead to severe memory issues or extremely slow computations, depending on the considered problem and the available hardware resources. Therefore, in `GlobalBioIm`, the choice between speed and memory consumption is left to the user by means of the Boolean attribute `doPrecomputation` of the abstract class `Map`. When activated, the instanciated object is allowed to store relevant quantities for acceleration purposes, at the expense of larger memory consumption. For instance, consider the `Cost` object corresponding to $\mathcal{J} = \frac{1}{2}\|\mathbf{H}\cdot - \mathbf{g}\|_2^2$ for which the `doPrecomputation` option is activated.

```
>> H=LinOpConv(fftn(psf));      % Convolution operator with a 256x256x256 kernel
>> L2=CostL2([],y);             % L2 cost function
>> J=L2*H;                      % Composition of L2 with H
>> J.doPrecomputation=true;     % Activation of the doPrecomputation option
```

Then, we evaluate the gradient of $\mathcal{J}$ at the two random points $\mathbf{f}_1$ and $\mathbf{f}_2$.

```
>> f1=rand(J.sizein);f2=rand(J.sizein);   % Generation of the random points f1 and f2
>> tic; g=J.applyGrad(f1); toc;          % Computation of the gradient at f1
Elapsed time is 1.870925 seconds.
>> tic; g=J.applyGrad(f2); toc;          % Computation of the gradient at f2
Elapsed time is 0.942075 seconds.
```

We observe that the second gradient computation is twice as fast as the first one. This is because the quantity $\mathbf{H}^T\mathbf{g}$ is computed and stored at the first call of the `applyGrad` method. For all subsequent calls, the computational burden is reduced to the application of $\mathbf{H}^T\mathbf{H}$ in (9).

Another feature that allows for faster computations at the expense of larger memory consumption is provided by the structure attribute `memoizeOpts` of the abstract class `Map`. When this attribute is activated, both the input and the result of the evaluation are stored whenever the corresponding `Map` object is evaluated. Hence, if the object is subsequently evaluated with the same input, the stored result is returned without any computation.

```
>> H=LinOpConv(fftn(psf));                % Convolution operator with a 256x256x256 kernel
>> H.memoizeOpts.apply=true;              % Activation of memoize for the apply method
>> f1=rand(H.sizein);f2=rand(H.sizein);   % Generation of the random points f1 and f2
>> tic; g=H*f1; toc;                      % First H*f1
Elapsed time is 0.822508 seconds.
>> tic; g=H*f1; toc;                      % Second (consecutive) H*f1. Returns the stored result
```

```
Elapsed time is 0.020624 seconds.
>> tic; g=H*f2; toc;                % Computation for the new input f2
Elapsed time is 0.823498 seconds.
```

This option proves to be particularly useful to avoid multiple computations within iterative methods. For instance, at each iteration of `OptiVMLMB`, both the cost $\mathcal{J}$ and its gradient $\boldsymbol{\nabla}\mathcal{J}$ need to be evaluated at the same point $\mathbf{f}$. For least-squares minimization, this involves computing $\mathcal{J}(\mathbf{f}) = \frac{1}{2}\|\mathbf{H}\mathbf{f}-\mathbf{g}\|_2^2$ and $\boldsymbol{\nabla}\mathcal{J}(\mathbf{f}) = \mathbf{H}^T(\mathbf{H}\mathbf{f}-\mathbf{g})$, which both call for the quantity $\mathbf{H}\mathbf{f}$. Hence, activating the `memoize` option for the `apply` method of $\mathbf{H}$ allows for the savings of one evaluation of $\mathbf{H}\mathbf{f}$ per iteration.

4.4. GPU Computing

`GlobalBioIm` provides two functions that allow the user to easily run any reconstruction pipeline on the GPU for faster computation. The function `useGPU`, which is typically called at the beginning of the script, allows selection of the computation mode: CPU computation (default), GPU computation with the MATLAB® Parallel Computing Toolbox™, or GPU computation with CudaMat (`https://github.com/RainerHeintzmann/CudaMat`). Next, the function `gpuCpuConverter` converts the input variable to the appropriate data type as specified by `useGPU`. A typical use of these functions is presented below.

```
useGPU(1);              % Set the GPU mode to 1 -> Matlab Parallel Computing Toolbox (TM)

%-- Load data
load('data');           % Variable g
g=gpuCpuConverter(g);   % Convert them to the correct type
% ... load and convert other variables
```

5. An Example with MultiChannel Deconvolution

5.1. Simulation Setting

We consider the sample depicted in Figure 2a. It has been extracted from the neuronal culture acquisition shared by Schmoranzer on the Cell Image Library website (`http://cellimagelibrary.org/images/41649`). It contains three channels that we process independently. Our simulation pipeline is illustrated in Figure 2. It encompasses several steps to account for the fact that, for real-world experiments, (i) the underlying sample is generally not supported within the field-of-view of the microscope; (ii) the sample is not periodic (contrary to what is implicitly assumed when using FFTs to perform the convolution). The three point-spread functions (PSF) have been generated in the Fourier domain. They are related to the classical Airy disk model, which is a radial function with the profile

$$h(\rho) = \begin{cases} \frac{1}{\pi}\left(2\cos^{-1}\left(\frac{\rho}{\rho_c}\right) - \sin\left(2\cos^{-1}\left(\frac{\rho}{\rho_c}\right)\right)\right), & \forall \rho < \rho_c \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

where $\rho_c = 2\mathrm{NA}/\lambda_{\mathrm{exc}}$ is the cutoff frequency which depends on the numerical aperture NA and the excitation wavelength λ_{exc}. Here, we set NA = 1.4 and λ_{exc} to 654nm (CY3 dye, red), 542nm (FITC dye, green), and 477nm (DAPI dye, blue)

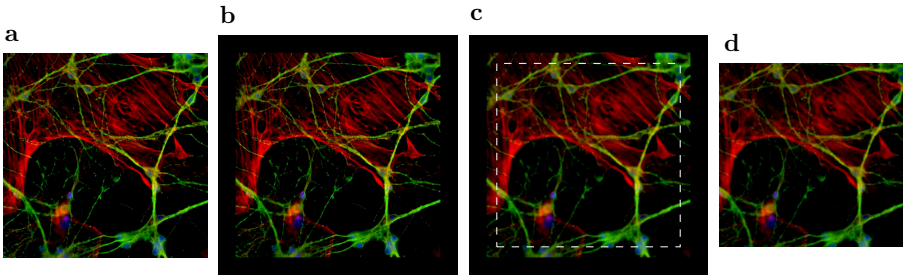

Figure 2: Simulation of multichannel blurred data. a) Input image ($512 \times 512 \times 3$). b) The image is zero-padded. c) Each channel is convolved with its corresponding PSF and a central region of size ($460 \times 460 \times 3$) is extracted (simulated field-of-view). d) Data are corrupted by additive Gaussian noise so that the resulting signal-to-noise ratio (SNR) is equal to 10dB.

for the three channels, respectively. Finally, the spatial sampling step (*i.e.,* the camera pixel size) is set to 64.5nm. Note that the data generated with this pipeline contain contributions from structures that lie outside the field-of-view.

5.2. Deconvolution

Given the blurred and noisy data $\{\mathbf{g}_k \in \mathbb{R}^M\}_{k=1}^3$, we formulate the deconvolution task as the optimization problem

$$\{\hat{\mathbf{f}}_k\}_{k=1}^3 = \arg \min_{\{\mathbf{f}_k \in \mathbb{R}^N\}_{k=1}^3} \left(\sum_{k=1}^3 \frac{1}{2} \|\mathbf{S}\mathbf{H}_k \mathbf{f}_k - \mathbf{g}_k\|_2^2 + \lambda \mathcal{R}(\mathbf{L}\mathbf{f}_k) + i_{\geq 0}(\mathbf{f}_k) \right), \qquad (11)$$

where $\mathbf{H}_k \in \mathbb{R}^{N \times N}$, $k \in \{1, 2, 3\}$, is the convolution operator for the kth channel, and $\mathbf{S} \in \mathbb{R}^{M \times N}$ selects the region of $\mathbf{Hf}$ that corresponds to the field-of-view. Indeed, since the sample is not fully included in the field-of-view, we seek a wider reconstruction that is larger than the field-of-view (*i.e.,* $N > M$) in order to avoid reconstruction artifacts [3, 31]. Finally, $i_{\geq 0}(\mathbf{f}) = \{0 \text{ if } \mathbf{f} \in \mathbb{R}^N_{\geq 0}; +\infty \text{ otherwise}\}$ is a nonnegativity constraint.

With `GlobalBioIm`, the construction of the operators $\mathbf{H}_k$, $k \in \{1, \dots, 3\}$, and $\mathbf{S}$ is done as follows.

```
%-- Load Data --
load('psf'); szin=size(psf);    % Variable psf (512x512x3)
load('ground_truth');           % Variable gt (460x460x3)
load('data'); szout=size(g);    % Variable g (460x460x3)

%-- Forward Model --
H=LinOpConv(fft2(psf),1,[1 2]);
S=LinOpSelectorPatch(szin,[1 1 1],szout);
```

Here, the three PSFs are stacked within the same `LinOpConv` operator which is set by the argument `[1 2]` to apply a convolution only to the first two dimensions. Hence, it performs independent 2D convolutions for each channel. The selector operator $\mathbf{S}$ then extracts a region that has the same size as the data.

Next, both the data fidelity term (*i.e.,* $\frac{1}{2}\|\cdot - \mathbf{g}\|_2^2$) and the nonnegativity constraint (*i.e.,* $i_{\geq 0}$) can be defined with two lines of code.

```
%-- L2 Loss function and nonnegativity constraint
L2=CostL2([ ],g);
P=CostNonNeg(szin);
```

For the regularization term $\mathcal{R}(\mathbf{L}\,\cdot)$, we propose to illustrate the modularity of `GlobalBioIm` by providing a set of examples (see Figures 3 and 4) that include various regularizers.

- The total-variation (TV) [8, 9, 30] combines the gradient operator $\mathbf{L} = [\mathbf{D}_1\, \mathbf{D}_2]^T$ with the (ℓ_2, ℓ_1)-mixed norm $\mathcal{R} = \|\cdot\|_{2,1}$. More precisely, for $\mathbf{f} \in \mathbb{R}^N$, we have that
$$\mathcal{R}(\mathbf{L}\mathbf{f}) = \sum_{n=1}^{N} \sqrt{[\mathbf{D}_1\mathbf{f}]_n^2 + [\mathbf{D}_2\mathbf{f}]_n^2}, \tag{12}$$
where $\mathbf{D}_1$ ($\mathbf{D}_2$, respectively) is the finite-difference operator along the first (second, respectively) dimension.
- The Hessian-Schatten-norm (HS) [19, 20] computes the $(*, \ell_1)$-mixed norm $\mathcal{R} = \|\cdot\|_{*,1}$ of the Hessian operator $\mathbf{L} = [\mathbf{D}_{ij}]_{1\leq i,j\leq 2}$ applied to $\mathbf{f} \in \mathbb{R}^N$ as
$$\mathcal{R}(\mathbf{L}\mathbf{f}) = \sum_{n=1}^{N} \left\| \begin{bmatrix} [\mathbf{D}_{11}\mathbf{f}]_n & [\mathbf{D}_{12}\mathbf{f}]_n \\ [\mathbf{D}_{21}\mathbf{f}]_n & [\mathbf{D}_{22}\mathbf{f}]_n \end{bmatrix} \right\|_*, \tag{13}$$
where $\|\cdot\|_*$ denotes the nuclear norm (*i.e.,* the first-order Schatten norm). It is defined as the ℓ_1-norm of the singular values of its argument. Finally, $\mathbf{D}_{ij}$ denotes the operator of second-order finite difference along the dimensions i and j.
- The smoothed total-variation (S-TV) [4, 8] is defined, for $\varepsilon > 0$, by
$$\mathcal{R}(\mathbf{L}\mathbf{f}) = \sum_{n=1}^{N} \sqrt{[\mathbf{D}_1\mathbf{f}]_n^2 + [\mathbf{D}_2\mathbf{f}]_n^2 + \varepsilon^2}. \tag{14}$$
- The Good's roughness (GR) [40] is defined, for $\varepsilon > 0$, by
$$\mathcal{R}(\mathbf{L}\mathbf{f}) = \sum_{n=1}^{N} \frac{[\mathbf{D}_1\mathbf{f}]_n^2 + [\mathbf{D}_2\mathbf{f}]_n^2}{\sqrt{|\mathbf{f}_n|^2 + \varepsilon^2}}. \tag{15}$$

Since the TV and HS regularizers are not differentiable, gradient-based methods cannot be used to minimize the objective function (11). However, for both these regularizers, the proximity operator of $\mathcal{R}(\,\cdot\,)$ can be efficiently computed (see [11] for $\|\cdot\|_{2,1}$ and [10, 19] for $\|\cdot\|_{\mathcal{S}_1,1}$). Hence, the optimization problem can be tackled using proximal-splitting algorithms such as the alternating direction method of multipliers (ADMM) [2, 7, 14, 32] or the primal-dual method proposed in [12]. These algorithms are designed to minimize cost functions of the form $\mathcal{J} = \sum_{p=1}^{P} \mathcal{J}_p(\mathbf{T}_p\,\cdot)$, where $\{\mathbf{T}_p\}_{p=1}^P$ are linear operators and the $\{\mathcal{J}_p\}_{p=1}^P$ are "simple" functions in the sense that their proximity operator can be evaluated efficiently.

The scripts provided in Figure 3 illustrate how these two algorithms can be implemented within the framework of `GlobalBioIm` to solve Problem (11) with TV or HS regularization. The modified lines of code between each setting have been

Total-variation [8, 9, 30] | Hessian-Schatten [19, 20]

ADMM [2, 7, 14, 32]

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
R=CostMixNorm21(L.sizeout,4);

%-- Optimization Algorithm ---------
Fn={L2*S,lamb*R,P};
Hn={H,L,LinOpIdentity(szin)};
rho_n=[1,1,1]*5e-1;
Opt=OptiADMM([],Fn,Hn,rho_n);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpHess(szin,[ ],[1 2]);
R=CostMixNormSchatt1(L.sizeout,1);

%-- Optimization Algorithm ---------
Fn={L2*S,lamb*R,P};
Hn={H,L,LinOpIdentity(szin)};
rho_n=[1,1,1]*5e-1;
Opt=OptiADMM([],Fn,Hn,rho_n);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

Primal-dual [12]

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
R=CostMixNorm21(L.sizeout,4);

%-- Optimization Algorithm ---------
Fn={L2*S,lamb*R,P};
Hn={H,L,LinOpIdentity(szin)};
Opt=OptiPrimalDualCondat([ ],[ ],Fn,Hn);
T=H'*H+L'*L+LinOpIdentity(szin);
Opt.tau=1;Opt.sig=1/(Opt.tau*T.norm);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpHess(szin,[ ],[1 2]);
R=CostMixNormSchatt1(L.sizeout,1);

%-- Optimization Algorithm ---------
Fn={L2*S,lamb*R,P};
Hn={H,L,LinOpIdentity(szin)};
Opt=OptiPrimalDualCondat([ ],[ ],Fn,Hn);
T=H'*H+L'*L+LinOpIdentity(szin);
Opt.tau=1;Opt.sig=1/(Opt.tau*T.norm);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

Figure 3: `GlobalBioIm` scripts for minimizing (11) with non-differentiable regularizers $\mathcal{R}(\mathbf{L}\,\cdot\,)$. Differences with respect to the script corresponding to ADMM with TV are highlighted.

highlighted—observe that very few modifications are needed. This underlines the simplicity of changing the regularizer and/or the algorithm within the `GlobalBioIm` framework.

For both algorithms, the splitting strategy is specified by the two cell arrays `Fn` and `Hn`. Note that, since $\mathbf{S}$ is a semi-orthogonal linear operator, the composition `L2*S` results in a `Cost` object that has an implementation of the proximity operator (see Example 4.1). Moreover, the two scripts that use the primal-dual method illustrate the relevance of the automatic simplification features described in Section 4.2. In order to ensure the convergence of the algorithm, the two parameters $\sigma > 0$ and $\tau > 0$ have to be chosen so that $\tau\sigma\|\sum_{p=1}^{P}\mathbf{T}_p^T\mathbf{T}_p\| \leq 1$ holds true [12]. The norm which is involved in this inequality is easily obtained with `GlobalBioIm` by building the operator `T=H'*H + L'*L + LinOpIdentity(szin)` explicitly and computing its norm `T.norm`. The key is that the composition used to build `T` is automatically simplified to a convolution operator with the proper kernel. Finally, observe that, as for the convolution operator, the gradient and Hessian operators are defined using the argument `[1 2]`, which implies that these operators are applied independently to each channel.

Smoothed Total Variation [4, 8]

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
RL=CostHyperBolic(L.sizeout,1e-7,4)*L;

%-- Optimization Algorithm -------
C = L2*S*H + lamb*RL;
Opt=OptiVMLMB(C,0.,[]);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

Good's Roughness [40]

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
RL=CostGoodRoughness(L,1e-2);

%-- Optimization Algorithm -------
C = L2*S*H + lamb*RL;
Opt=OptiVMLMB(C,0.,[]);
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

VMLMB [34]

FISTA [5]

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
RL=CostHyperBolic(L.sizeout,1e-7,4)*L;

%-- Optimization Algorithm -------
C = L2*S*H + lamb*RL;
Opt=OptiFBS(C,P);
Opt.fista=true; Opt.gam=5e-2;
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

```
%-- Regularizer ----------------
lamb=5e-3;
L=LinOpGrad(szin,[1 2]);
RL=CostGoodRoughness(L,1e-2);

%-- Optimization Algorithm -------
C = L2*S*H + lamb*RL;
Opt=OptiFBS(C,P);
Opt.fista=true; Opt.gam=5e-2;
Opt.OutOp=OutputOpti(1,S'*gt,50);
Opt.CvOp=TestCvgStepRelative(1e-4);
Opt.maxiter=500; Opt.ItUpOut=50;
Opt.run(S'*y);
```

Figure 4: `GlobalBioIm` scripts for minimizing (11) with differentiable regularizers $\mathcal{R}(\mathbf{L}\,\cdot\,)$. Differences with respect to the script corresponding to VMLMB with S-TV are highlighted.

As opposed to TV and HS, the S-TV and the GR regularizers are differentiable. Hence, the optimization problem in (11) can be addressed through gradient-based methods. In Figure 4, we present scripts in which the objective function in (11) with S-TV or GR regularization is minimized using either the variable-metric limited-memory-bounded (VMLMB) algorithm [34] or the fast iterative shrinkage-thresholding algorithm (FISTA) [5]. For these two minimization algorithms, each iteration requires the evaluation of the gradient of $\sum_{k=1}^{3} \|\mathbf{SH}_k \cdot - \mathbf{g}_k\|_2^2 + \lambda \mathcal{R}(\mathbf{L}\,\cdot\,)$ as well as a projection onto the set of nonnegative vectors. Once again, changing the regularizer or the optimization method only requires the modification of very few lines, as highlighted in Figure 4.

5.3. Numerical Comparisons

The modularity of `GlobalBioIm`, which was demonstrated in the scripts presented in Section 5.2, offers a simple way to compare the effect of regularizers as well as the efficiency of optimization algorithms.

We first present the quality of the deconvolution obtained with the four regularizers TV, HS, S-TV, and GR. Here, we used ADMM to minimize (11) with non-differentiable regularizers (*i.e.,* TV and HS), and FISTA to minimize (11) with differentiable regularizers (*i.e.,* S-TV and GR). The SNR of the deconvolved image

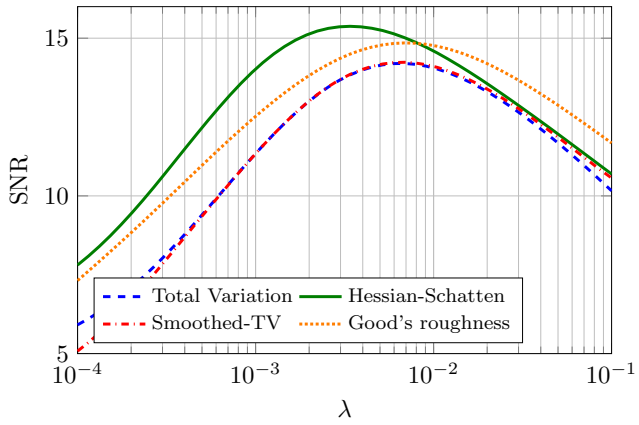

Figure 5: Evolution of the signal-to-noise ratio of the deconvolved image with respect to the regularization parameter λ.

as a function of the regularization parameter λ is depicted in Figure 5, while the deconvolved images that maximize the SNR are presented in Figure 6. As expected, HS and GR lead to better results by avoiding the well-known staircasing effect of TV and S-TV. Although GR is slightly below HS in terms of SNR, it provides comparable qualitative (*i.e.,* visual) results.

We now fix the parameter λ to the value that maximizes the SNR in Figure 5 for TV and S-TV. The convergence curves generated by ADMM and the primal-dual method for the minimization of (11) with TV, as well as those generated by FISTA and VMLMB when the regularizer is set to be S-TV, are presented in Figure 7. We would like to emphasize that the parameters of the algorithms have not been tuned to obtain the fastest convergence. Hence, these results constitute more an illustration of the kind of comparisons that can be easily performed with `GlobalBioIm` rather than an empirical demonstration of the convergence speed of these algorithms. Moreover, both ADMM and the primal-dual method offer alternative splitting strategies that may lead to improved convergence speed. Note that the adaptation of the scripts in Figure 3 to these variations is straightforward with `GlobalBioIm`. We refer the reader to the online documentation of the corresponding two `Opti` classes for more details on how to establish such adaptations.

5.4. Other examples

One can find an example of three-dimensional deconvolution on real data within the Section "Examples" of the online documentation. Moreover, references to papers that use `GlobalBioIm` are listed in the Section "Related Papers" of this documentation. We distinguish between works that provide open-source codes and those which do not. Hence, this list constitutes a growing source of examples of use of `GlobalBioIm` on concrete problems.

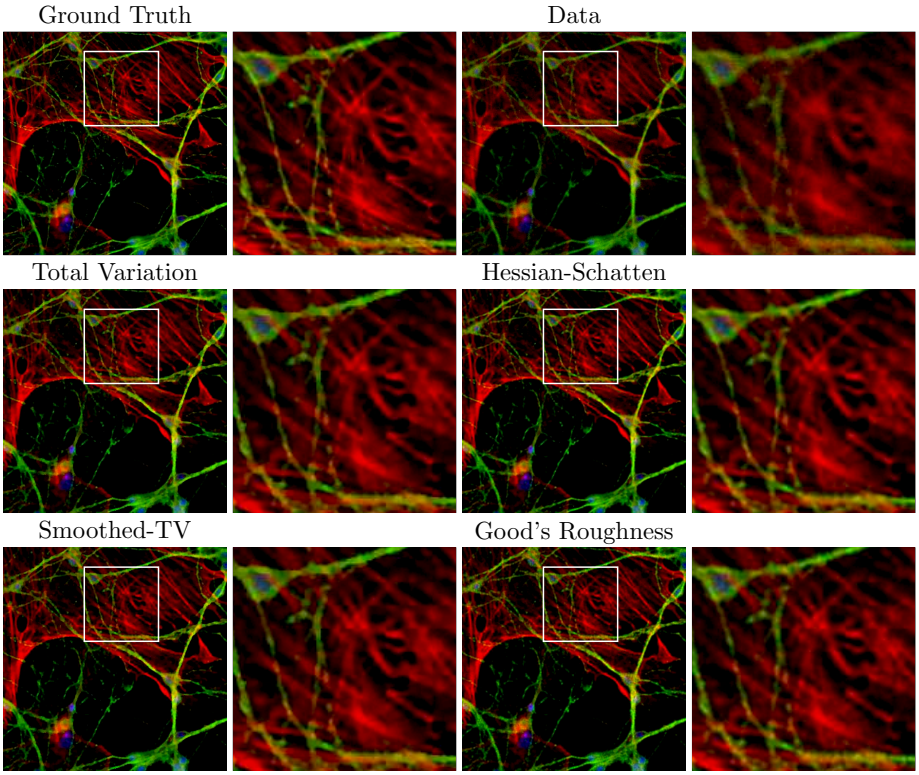

Figure 6: Deconvolution results obtained with different regularizers for the optimal λ extracted from Figure 5. A zoom of the region delimited by the white square is also presented.

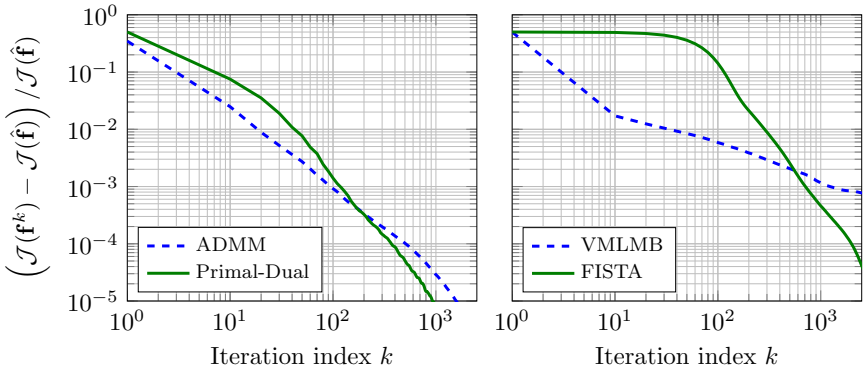

Figure 7: Convergence curves for the minimization of (11) with TV (left) or S-TV (right). The solution $\hat{\mathbf{f}}$ has been computed by performing 10,000 iterations of ADMM (FISTA, respectively).

6. Discussion

Open-source software is an essential component of modern research. Not only does it shape theoretical developments, but it also turns out to be a critical tool to bridge the gap that separates researchers specialized in computer science/mathematics from scientists versed in biophysical sciences/medicine. Moreover, open-source software can act as a catalyst for engaging in new collaborations by promoting external contributions.

Motivated by the observation that the image-formation models of most of the commonly used biomedical imaging systems can be expressed as a composition of a limited number of elementary operators, we developed the open-source MATLAB library `GlobalBioIm`. This library provides a unified and user-friendly framework for the resolution of inverse problems. It is designed around three entities, namely, forward models, cost functions, and optimization algorithms, which constitute the building blocks of any inverse problem. This organization gives `GlobalBioIm` a modularity that greatly facilitates the comparison between regularizers and or solvers, as illustrated in Section 5. Moreover, `GlobalBioIm` enjoys an operator-algebra mechanism able to perform automatic simplification of composed operators. Finally, new modalities, cost functions, or solvers are easily added to the framework of `GlobalBioIm`.

Acknowledgements

The authors would like to thank warmly Rainer Heintzmann for fruitful discussions related to this project as well as Philippe Thévenaz for his useful feedback on the paper. This research was supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme, Grant Agreement No. 692726 GlobalBioIm: Global integrative framework for computational bio-imaging.

References

[1] Jonas Adler, Holger Kohr, and Ozan Öktem. ODL-A Python framework for rapid prototyping in inverse problems. *Royal Institute of Technology*, 2017.

[2] Manya V Afonso, José M Bioucas-Dias, and Mário AT Figueiredo. An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE Transactions on Image Processing*, 20(3):681–695, 2011.

[3] Mariana SC Almeida and Mario Figueiredo. Deconvolving images with unknown boundaries using the alternating direction method of multipliers. *IEEE Transactions on Image Processing*, 22(8):3074–3086, August 2013.

[4] Jean-Franccois Aujol. Some first-order algorithms for total variation based image restoration. *Journal of Mathematical Imaging and Vision*, 34(3):307–327, 2009.

[5] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, March 2009.

[6] Ander Biguri, Manjit Dosanjh, Steven Hancock, and Manuchehr Soleimani. TIGRE: A MATLAB-GPU toolbox for CBCT image reconstruction. *Biomedical Physics & Engineering Express*, 2(5):055010, September 2016.

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, July 2011.

[8] Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

[9] Antonin Chambolle and Pierre-Louis Lions. Image recovery via total variation minimization and related problems. *Numerische Mathematik*, 76(2):167–188, 1997.

[10] Giovanni Chierchia, Nelly Pustelnik, Béatrice Pesquet-Popescu, and Jean-Christophe Pesquet. A nonlocal structure tensor-based approach for multicomponent image recovery problems. *IEEE Transactions on Image Processing*, 23(12):5531–5544, December 2014.

[11] Patrick Louis Combettes and Jean-Christophe Pesquet. A proximal decomposition method for solving convex variational inverse problems. *Inverse Problems*, 24(6):065014, November 2008.

[12] Laurent Condat. A primal–dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms. *Journal of Optimization Theory and Applications*, 158(2):460–479, December 2013.

[13] Laurène Donati, Masih Nilchian, Carlos Óscar Sanchez Sorzano, and Michael Unser. Fast Multiscale Reconstruction for Cryo-EM. *Journal of Structural Biology*, In Press, September 2018.

[14] Michel Fortin and Roland Glowinski. *Augmented Lagrangian methods: applications to the numerical solution of boundary-value problems*, volume 15. Elsevier, 2000.

[15] Silvia Gazzola, Per Christian Hansen, and James G. Nagy. IR Tools: A MATLAB Package of Iterative Regularization Methods and Large-Scale Test Problems. *arXiv preprint arXiv:1712.05602*, 2017.

[16] William W Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.

[17] Per Christian Hansen and Jakob Sauer Jørgensen. AIR Tools II: algebraic iterative reconstruction methods, improved implementation. *Numerical Algorithms*, 79(1):107–137, November 2017.

[18] Pavel Křížek, Prívek, Tomáš Lukeš, Martin Ovesný, Karel Fliegel, and Guy M. Hagen. SIMToolbox: A MATLAB toolbox for structured illumination fluorescence microscopy. *Bioinformatics*, 32(2):318–320, January 2016.

[19] Stamatios Lefkimmiatis and Michael Unser. Poisson image reconstruction with Hessian Schatten-norm regularization. *IEEE Transactions on Image Processing*, 22(11):4314–4327, November 2013.

[20] Stamatios Lefkimmiatis, John Paul Ward, and Michael Unser. Hessian Schatten-norm regularization for linear inverse problems. *IEEE Transactions on Image Processing*, 22(5):1873–1888, January 2013.

[21] Yiming Li, Markus Mund, Philipp Hoess, Joran Deschamps, Ulf Matti, Bianca Nijmeijer, Vilma Jimenez Sabinina, Jan Ellenberg, Ingmar Schoen, and Jonas Ries. Real-time 3D single-molecule localization using experimental point spread functions. *Nature Methods*, 15(5):367, April 2018.

[22] Andreas Maier, Hannes G Hofmann, Martin Berger, Peter Fischer, Chris Schwemmer, Haibo Wu, Kerstin Müller, Joachim Hornegger, Jang-Hwan Choi, Christian Riess, Andreas Keil, and Rebecca Fahrig. CONRAD–A software framework for cone-beam imaging in radiology. *Medical Physics*, 40(11), November 2013.

[23] Michael T. McCann, Masih Nilchian, Marco Stampanoni, and Michael Unser. Fast 3D reconstruction method for differential phase contrast x-ray CT. *Optics Express*, 24(13):14564–14581, June, 2016.

[24] Thibaut Merlin, Simon Stute, Didier Benoit, Julien Bert, Thomas Carlier, Claude Comtat, Marina Filipovic, Frédéric Lamare, and Dimitris Visvikis. CASToR: A generic data organization and processing code framework for multi-modal and multi-dimensional tomographic reconstruction. *Physics in Medicine & Biology*, 63(18):185005, September 2018.

[25] Jean-Jacques Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes Rendus de l'Académie des Sciences Série A Mathématiques*, 255:2897–2899, 1962.

[26] Marcel Müller, Viola Mönkemöller, Simon Hennig, Wolfgang Hübner, and Thomas Huser. Open-source image reconstruction of super-resolution structured illumination microscopy data in ImageJ. *Nature Communications*, 7:10980, March 2016.

[27] Martin Ovesnỳ, Pavel Křížek, Josef Borkovec, Zdeněk Švindrych, and Guy M Hagen. ThunderSTORM: A comprehensive ImageJ plug-in for PALM and STORM data analysis and super-resolution imaging. *Bioinformatics*, 30(16):2389–2390, May 2014.

[28] Anthony D Padula, Shannon D Scott, and William W Symes. A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms. *ACM Transactions on Mathematical Software*, 36(2):8, 2009.

[29] Simon Rit, M Vila Oliva, Sébastien Brousmiche, Rudi Labarbe, David Sarrut, and Gregory C Sharp. The Reconstruction Toolkit (RTK), an open-source cone-beam CT reconstruction toolkit based on the Insight Toolkit (ITK). In *Journal of Physics: Conference Series*, volume 489, page 012079. IOP Publishing, 2014.

[30] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, November 1992.

[31] Daniel Sage, Lauréne Donati, Ferréol Soulez, Denis Fortun, Guillaume Schmit, Arne Seitz, Romain Guiet, Cédric Vonesch, and Michael Unser. DeconvolutionLab2: An open-source software for deconvolution microscopy. *Methods*, 115:28–41, February 2017.

[32] Simon Setzer, Gabriele Steidl, and Tanja Teuber. Deblurring poissonian images by split bregman techniques. *Journal of Visual Communication and Image Representation*, 21(3):193–199, 2010.

[33] Emmanuel Soubies and Michael Unser. Computational super-sectioning for single-slice structured-illumination microscopy. *Preprint*, 2018.

[34] Eric Thiébaut. Optimization issues in blind deconvolution algorithms. In *Proc. SPIE 4847, Astronomical Data Analysis II*, volume 4847, pages 174–184. International Society for Optics and Photonics, December 2002.

[35] Eric Thiébaut. Zenodo emmt/LazyAlgebra.jl: First release of LazyAlgebra, 2018.

[36] Eric Thiébaut, Jonathan, Ferréol Soulez, and Ludo. Zenodo emmt/TiPi: Release 1.0.0 of TiPi, 2018.

[37] Kris Thielemans, Charalampos Tsoumpas, Sanida Mustafovic, Tobias Beisel, Pablo Aguiar, Nikolaos Dikaios, and Matthew W Jacobson. STIR: Software for tomographic image reconstruction Release 2. *Physics in Medicine & Biology*, 57(4):867, February 2012.

[38] Michael Unser, Emmanuel Soubies, Ferréol Soulez, Michael McCann, and Lauréne Donati. GlobalBioIm: A unifying computational framework for solving inverse problems. In *Proceedings of the OSA Imaging and Applied Optics Congress on Computational Optical Sensing and Imaging (COSI'17)*, San Francisco CA, USA, June 26-29, 2017. paper no. CTu1B.

[39] Wim van Aarle, Willem Jan Palenstijn, Jan De Beenhouwer, Thomas Altantzis, Sara Bals, K. Joost Batenburg, and Jan Sijbers. The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography. *Ultramicroscopy*, 157:35 – 47, May 2015.

[40] Peter J Verveer, Mark J Gemkow, and Thomas M Jovin. A comparison of image restoration approaches applied to three-dimensional confocal and wide-field fluorescence microscopy. *Journal of Microscopy*, 193(1):50–61, January 1999.

[41] Cédric Vonesch, Lanhui Wang, Yoel Shkolnisky, and Amit Singer. Fast wavelet-based single-particle reconstruction in Cryo-EM. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1950–1953, 2011.