PyEPRI: A CPU & GPU compatible Python package for Electron Paramagnetic Resonance Imaging

Rémy Abergel, Sylvain Durand, Yves-Michel Frapart

PyEPRI: A CPU & GPU compatible Python package for Electron Paramagnetic Resonance Imaging

Rémy Abergel[a], Sylvain Durand[a], Yves-Michel Frapart[b]

[a] *Université Paris Cité, CNRS, MAP5, F-75006 Paris, France*
[b] *Université Paris Cité, CNRS, LCBPT, F-75006 Paris, France*

Abstract

This work presents the PyEPRI package, an open-source Python package for Electron Paramagnetic Resonance Imaging. The PyEPRI package implements low-level operators, like projection and backprojection, involved in Electron Paramagnetic Resonance (EPR) and also high-level advanced algorithms, like total variation based EPR image reconstruction, for end-users. The package is fully implemented in Python and provides both CPU and GPU computation capabilities, through the libraries Numpy, PyTorch and Cupy. This package comes with a detailed documentation, including precise mathematical definitions and many reproducible demonstration examples and tutorials, making it easy for users with no particular expertise on coding image processing algorithms to get started. This package is also highly modular and only relies on standard data types, as such, it can also be easily used by advanced users to develop new algorithms while benefiting from an optimized computing environment and some rigorously tested operators. The PyEPRI package has been developed by researchers with the hope that it will be useful to the EPR community.

Keywords: electron paramagnetic resonance imaging, image processing, open-source software, reproducible research

1. Motivations

The reconstruction of EPR images has been the subject of extensive research, leading to various methods and algorithms with significant impact in many fields like biomedical sciences or biology [1–10], batteries conception [11–14], detection of defects on material surface [15, 16]. Over the past decade, significant efforts have been made to perform this reconstruction using modern digital image processing techniques based on variational models, resulting in more efficient algorithms than those implemented in commercial softwares [17–25].

Despite the promising prospects offered by variational methods, they are not accessible to all due to the technical complexity involved in their implementation. The lack of sufficiently robust, generic and well documented open-source software means that these methods are practically only usable by those being able to develop their own codes. Moreover, the efficiency of the code is highly dependent on the choices made regarding the modeling of operators, the minimization algorithms used, and whether or not some mathematical tricks are incorporated. In parallel, commercial software such as Xepr [26] only offers fairly basic reconstruction methods (primarily based on filtered backprojection) with few details about the precise algorithm design, which limits their use as individual components in a more complex processing pipeline.

The PyEPRI package was developed to provide the EPR community with robust, modern, and fast tools for EPR imaging. The package is compatible with both CPU and GPU via the Numpy, Cupy, and PyTorch libraries. This package contains numerous unit tests that verify key mathematical properties of the projection and backprojection operators it includes, ensuring a rigorous and reliable implementation of the reconstruction algorithms and facilitating the package's stability in future releases.

The package is available on the Python Package Index (PyPi) (`https://pypi.org/project/pyepri/`), making installation easier, and its source code is fully available on GitHub under an MIT license (`https://github.com/remy-abergel/pyepri/`). A comprehensive documentation, including numerous tutorials and demonstration examples, is provided with the package and serves as the entry point for users wishing to get started (`https://pyepri.math.cnrs.fr/`).

2. Package overview

2.1. Backend system for CPU & GPU compatibility

The PyEPRI package can perform intensive calculations on CPU or GPU using the popular scientific computing libraries in Python: NumPy, CuPy, and PyTorch. Concretely, at the beginning of their script, the user needs to instantiate a "backend" that specifies which library and computation unit (CPU or GPU) will be used. The backend instance must be passed as input of the PyEPRI functions and is internally used to remap the data and calculations to the appropriate functions. This way, changing the backend instance at the beginning of the script allows to change the computational unit or the scientific library used to perform all the calculations, keeping the rest of the script unchanged. Although the instantiation of a backend is in fact optional and can be skipped for writing library-specific scripts, its use is strongly recommended. It is therefore systematically implemented in all the demonstration scripts provided in the online documentation.

2.2. Data import

The PyEPRI package supports the reading of datasets stored in BES3T[1] format, which is a proprietary format used on Bruker ELEXSYS and EMX machines[2]. Example datasets in BES3T format are embedded in the PyEPRI package. These datasets are used in the various demonstration example scripts included in the online documentation. Users with data in other formats will need to export their datasets in any format supported by Python (for instance in ASCII format) or by one of its many available libraries. Many demonstration examples related to loading and displaying datasets in either BES3T or ASCII format are presented in the online documentation of the PyEPRI package.

2.3. Projection operators

The modeling of EPR imaging is intrinsically linked to the mathematical concept of image projection using the Radon transform, as we shall briefly describe now. Let us consider a paramagnetic species

[1] Bruker EPR Standard for Spectrum Storage and Transfer
[2] the related functions were adapted from those available in the DIVE package [27]

X with reference spectrum $h_X : \mathbb{R} \to \mathbb{R}$ and concentration mapping $U_X : \mathbb{R}^d \to \mathbb{R}$ (where $d = 2$ or 3 denotes the image dimension). An EPR acquisition is obtained by superimposing two magnetic fields into the resonance cavity: an homogeneous magnetic field (with spatially constant intensity), and a so-called magnetic field gradient (with intensity linearly varying along a given direction) characterized by a field gradient vector $\gamma \in \mathbb{R}^d$. An EPR acquisition corresponds to the signal $\mathcal{P}_{X,\gamma} : \mathbb{R} \to \mathbb{R}$ obtained by ramping up the amplitude $B \in \mathbb{R}$ of the homogeneous magnetic field in the resonator. It is called *a projection* and it is defined by

$$\forall B \in \mathbb{R}, \quad \mathcal{P}_{X,\gamma}(B) = \Big(h_X * \mathcal{G}_\gamma(U_X)\Big)(B), \tag{1}$$

where $*$ denotes the convolution product and $\mathcal{G}_\gamma(U_X)$ corresponds to the dilatation with factor $-\|\gamma\|$ of the Radon transform of U_X in the direction γ, that is, $\mathcal{G}_\gamma(U_X)(B) = \frac{1}{\|\gamma\|}\mathcal{R}_\gamma(U_X)(-B/\|\gamma\|)$ where

$$\forall r \in \mathbb{R}, \quad \mathcal{R}_\gamma(U_X)(r) = \int_{\mathbb{R}^d} U_X(x)\, \delta_0(\langle x, \gamma\rangle - r)\, dx \tag{2}$$

and where δ_0 denotes the Dirac impulse centered at 0. In practice, multiple projections are acquired sequentially, for various values of field gradient vectors γ in $(\gamma_1, \gamma_2, \ldots, \gamma_N)$, for instance changing the direction of the field gradient vector from one acquisition to another, or its amplitude, or even both. Besides, practical measurements are sampled for a finite number of values $B \in (B_1, B_2, \ldots, B_{N_B})$ that we assume to be arranged in ascending order and regularly spaced [3].

The PyEPRI package implements a projection operator, able to generate a sequence of projections given

(i) u: a discrete image corresponding to the sampling of U_X with step δ along all directions;

(ii) δ: the spatial sampling step (or *pixel size*) associated to u;

(iii) $\mathcal{B} := (B_1, B_2, \ldots, B_{N_B})$: the sequence of sampling nodes for the homogeneous magnetic field amplitudes;

(iv) h: the reference spectrum sampled over the grid $\mathcal{B}$;

(v) $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_N)$: the sequence of field gradient vectors.

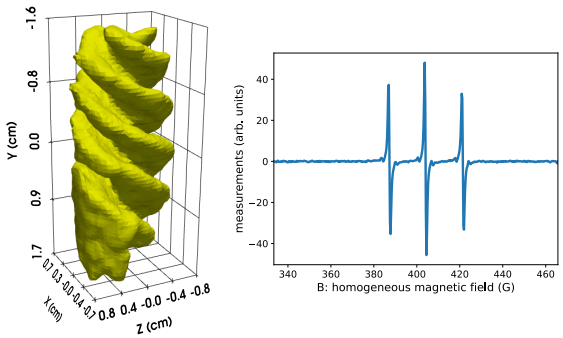
This operator will be denoted by $A_{X,\Gamma}$ below and its mathematical definition (including details about its derivation from (1) and (2)) is provided in the documentation of the PyEPRI package. The projection operator $A_{X,\Gamma}$ changes the d-dimensional discrete image u into a sequence of discrete projections, as illustrated in Figure 1 (in the 3D setting).

Figure 1 illustrates projection synthesis in the somehow standard situation where the sample is made of a single paramagnetic species. The projection operator implemented in PyEPRI is generalized to handle projection synthesis in the more complex situations listed below.

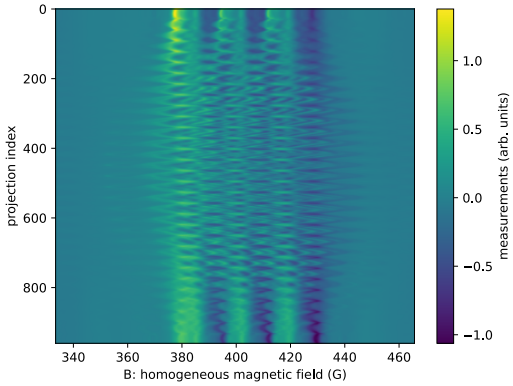
(I) Multiple EPR species (see [25]): the sample of interest contains several distinct paramagnetic species $\mathcal{X} = (X_1, X_2, \ldots, X_K)$ with associated concentration mappings $u := (u_1, u_2, \ldots, u_K)$ and reference spectra $h := (h_1, h_2, \ldots, h_K)$.

(II) Multiple experiments: the acquisition aggregates several experiments with different parameter settings that can potentially affect the shape of the reference spectra (for instance the microwave power can be changed from one experiment to another). PyEPRI supports projection synthesis in this situation given the sequence of concentration mappings of the different EPR species $u := (u_1, u_2, \ldots, u_K)$, the EPR spectrum $h_j^{(i)}$ of the j-th EPR species X_j in the i-th experimental setting (for $1 \le i \le L$ and $1 \le j \le K$), and the field gradient vector sequences associated to each experiment $\Gamma := (\Gamma_i)_{1 \le i \le L}$.

[3] irregular sampling schemes for the projections is currently not supported by the PyEPRI package



(a) input 3D image u (isosurface) and input reference spectrum h



(b) output projections $A_{X,\Gamma}(u)$

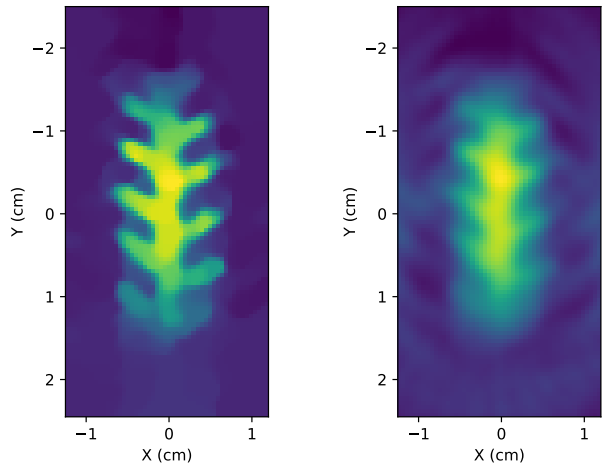
Figure 1: **Projection operation**. The projection operator $A_{X,\Gamma}$ is able to generate a sequence of projections from the set of inputs (i)–(v) described above and partially represented here. We display in (a) an isosurface of a discrete 3D image u (left-hand side) and the graph of a discrete reference spectrum h sampled over a grid $\mathcal{B}$ containing $N_B = 500$ sampling nodes regularly spaced in the range $[333.45, 465.69]$ G (right-hand side). Given a sequence $\Gamma = (\gamma_1, \ldots, \gamma_N) \in (\mathbb{R}^3)^N$ containing $N = 961$ field gradient vectors (not represented here), we used the $A_{X,\Gamma}$ operator to generate the sequence of projections displayed in (b). Those projections are sampled over the grid $\mathcal{B}$ and stored line by line into a 2D array containing N rows and N_B columns. The i-th row of this array corresponds to the projection of the image u with the field gradient vector γ_i.

The generalization of the $A_{X,\Gamma}$ operator described in (II) will be denoted as $\mathcal{A}_{X,\Gamma}$ in the rest of this paper. The interest of this generalization lies in the possibility to address the inversion of $\mathcal{A}_{X,\Gamma}$ using standard inverse problems, opening door for the reconstruction of one or more EPR image(s) from a sequence of projections acquired in one or more experimental condition(s) using an unified mathematical framework. Indeed, most standard methods used to solve inverse problems require, first and foremost, the ability to evaluate, as efficiently as possible, the so-called "direct" operator, which simulates the measurements (i.e., the projections) from the quantities of interest (i.e., the concentration mappings).

The practical implementation of the projection operators provided in the PyEPRI package strongly relies on the use of the FINUFFT package for the fast and accurate computation of nonuniform discrete Fourier transforms [28–30].

2.4. Backprojection operators

The relation (1) that maps an image to a projection is linear and continuous. The mathematical adjoint of the projection operator is called *backprojection*. The discretized operator $\mathcal{A}_{X,\Gamma}$ is also a linear operator and we will refer to the adjoint operator as the backprojection operator, which will be denoted as $\mathcal{A}^*_{X,\Gamma}$. The backprojection operator $\mathcal{A}^*_{X,\Gamma}$ maps a sequence of projections to one or several image(s), as illustrated in Figure 2, in the standard case of projections simulated from a single EPR source image (i.e., when the generalized operator $\mathcal{A}_{X,\Gamma}$ reduces to the initial and more usual one $A_{X,\Gamma}$).



(a) reference volume (slice $Z = 0$) (b) backprojected volume (slice $Z = 0$)

Figure 2: **Backprojection of a sequence of projections generated from a 3D volume**. We display in (a) the central slice ($Z = 0$ cm) of the 3D volume image represented in the left-hand side of Figure 1 (a). We display in (b) the central slice ($Z = 0$ cm) of the backprojected volume obtained by applying the backprojection operator $A^*_{X,\gamma}$ to the sequence of projections displayed in Figure 1 (b). We can see that the backprojected volume (b) looks like a blurry version of the original one (a). This illustrates in fact an interesting property of the composed operation $A^*_{X,\Gamma} \circ A_{X,\Gamma}(u)$ which turns out to be the convolution between the input image u and a fixed (blurring) kernel. We will explain in the next sections how the PyEPRI package takes advantage from this property to implement fast reconstruction algorithms.

Remark. The projection operator $\mathcal{A}_{X,\Gamma}$ being linear, it can be represented using a matrix form. In this context, the adjoint corresponds to the transposed matrix. Although this formulation is mathematically correct, it is not really helpful from a practical standpoint due to the massive size of the matrices involved, which are generally far too large to be stored and manipulated on a computer.

2.5. Projection–backprojection operation using Toeplitz kernels

One can show that performing the *projection–backprojection* of a given image u (which means, computing $A^*_{X,\Gamma} \circ A_{X,\Gamma}(u)$) is equivalent to carrying out the (discrete) convolution between u and a fixed convolution kernel φ, i.e., for any image u,

$$A^*_{X,\Gamma} \circ A_{X,\Gamma}(u) = \varphi * u \,. \tag{3}$$

Note that (3) formally arises from the fact that the composed operator $A^*_{X,\Gamma} \circ A_{X,\Gamma}$ has a Toeplitz structure (see [24]). For that reason, the kernel φ is referred as a Toeplitz kernel. The PyEPRI package implements functions dedicated to the fast and accurate evaluation of the Toeplitz kernel φ, as well as that of $A^*_{X,\Gamma} \circ A_{X,\Gamma}(u)$ using (3).

The Toeplitz kernel φ explicitly depends on the sequence of field gradient vectors Γ and the reference spectrum of the species X (we refer to the online documentation for its explicit definition) but not on the image u involved in (3). Consequently, for a given reference spectrum and a given sequence of field gradient vectors Γ, the Toeplitz kernel φ can be computed once and for all and used *as is* whenever the evaluation of $A^*_{X,\Gamma} \circ A_{X,\Gamma}$ is needed. This feature is of crucial importance in applications requiring many evaluations of the composite operator $A^*_{X,\Gamma} \circ A_{X,\Gamma}$. Indeed, once the kernel φ is computed, the evaluation of $A^*_{X,\Gamma} \circ A_{X,\Gamma}(u)$ using (3) is usually much more faster than the successive evaluation of the $A_{X,\Gamma}$ and $A^*_{X,\Gamma}$ operators taken separately [4]. The ability to rapidly perform the projection–backprojection operation is particularly valuable when implementing iterative algorithms for EPR image reconstruction, as we shall discuss in the next section. As regards the generalized projection operator $\mathcal{A}_{X,\Gamma}$, the rapid evaluation of $\mathcal{A}^*_{X,\Gamma} \circ \mathcal{A}_{X,\Gamma}$ using Toeplitz kernels is also possible and this feature is implemented and documented in the PyEPRI package.

[4] This can be explained by the fact that evaluating the operator $A_{X,\Gamma}$ and its adjoint $A^*_{X,\Gamma}$ requires the computation of non-uniform discrete Fourier transforms, which demand significant computational resources. In contrast, the evaluation of $A^*_{X,\Gamma} \circ A_{X,\Gamma}$ using (3) can be performed with standard discrete Fourier transforms, which are easier to compute.

2.6. Optimization algorithms

Modern EPR image reconstruction methods rely on variational models and consist in formulating the image reconstruction task as the minimization problem of finding

$$\widetilde{u} \in \operatorname*{argmin}_{u \in \mathcal{E}} E(u) \,, \tag{4}$$

where $\mathcal{E}$ represents the space in which the reconstruction lies and $E : \mathcal{E} \to \mathbb{R} \cup \{+\infty\}$ is the energy that we need to minimize to obtain the reconstruction $\widetilde{u}$. When imaging a sample containing a single paramagnetic species, the space $\mathcal{E}$ is expressed as $\mathcal{E} = \mathbb{R}^\Omega$ where [5] $\Omega = \{0, 1, \ldots, N_1 - 1\} \times \ldots \times \{0, 1, \ldots, N_d - 1\}$ represents the discrete image domain and $N_1, \ldots, N_d$ represent the number of pixels of the image along each one of its d dimensions. Otherwise, in the situation where multiple EPR species are involved, the space $\mathcal{E}$ is expressed as $\mathcal{E} = \mathbb{R}^{\Omega_1} \times \cdots \times \mathbb{R}^{\Omega_K}$, where $\Omega_j = \{0, 1, \ldots, N_1^{(j)} - 1\} \times \ldots \times \{0, 1, \ldots, N_d^{(j)} - 1\}$ represents the discrete image domain of the j-th image and $N_1^{(j)}, \ldots, N_d^{(j)}$ its dimensions. In both situations, the energy E is usually decomposed into the sum of two terms,

$$\forall u \in \mathcal{E}, \quad E(u) = F(u) + \lambda \cdot G(u) \,, \tag{5}$$

where F is referred as *data-fidelity* term, G is referred as *regularity term* and $\lambda > 0$ is referred as *regularity parameter*. The data-fidelity term $F(u)$ reflects the plausibility of the image (or the sequence of images) u with respect to the measured data, this term is therefore small for choices of u being compliant with the measured data and large otherwise. The regularity term $G(u)$ reflects some prior knowledge about the image (or the sequence of images) u that we are looking for, it usually promotes a form of spatial regularity by being correspondingly larger when u is considered as spatially irregular. This term can also be designed to impose some constraints (for instance positivity, unitary norm, ...) to the reconstruction model (in this case, one sets $G(u) = +\infty$ when u does not satisfies the constraint). The regularity parameter λ controls the relative importance of the data-fidelity term with respect to the regularity term in the minimization process. In many situations, the energy E involved in (4) is convex but nondifferentiable and its minimization can be efficiently handled using modern proximal algorithms which come with strong convergence guarantees (a general review about this topic can be found for instance in [31]).

2.6.1. TV-regularized least-squares

At the present time, the PyEPRI package implements numerical schemes for addressing the minimization (4) when the energy E decomposes as (5) using a least-squares data-fidelity term and a Total Variation (TV) regularity term, as we shall describe now.

Given a sequence s of measured projections, the least-squares data-fidelity term corresponds to the setting

$$\forall u \in \mathcal{E}, \quad F(u) = \frac{1}{2} \|A(u) - s\|_2^2 \tag{6}$$

where $A = A_{X,\Gamma}$ (respectively $A = \mathcal{A}_{X,\Gamma}$) is the direct operator that simulates a sequence of projections from an image u (respectively a sequence of images $u = (u_1, u_2, \ldots, u_K)$). Such a data-fidelity term enforces, in the minimization process (4), the choice of images (or sequences of images) u leading to simulated projections that are close to the observed ones (i.e., leading to $A(u) \approx s$).

[5] The notation $\mathbb{R}^\Omega$ formally refers to the mappings from Ω to $\mathbb{R}$

The TV regularity term is defined as follows. When $u \in \mathbb{R}^{\Omega}$ (i.e., when u is a single image with domain Ω), we take

$$G(u) = \mathrm{TV}(u) := \sum_{k\in\Omega} \|\nabla_{\mathrm{FD}}(u)(k)\|_2 \,, \tag{7}$$

where ∇_{FD} denotes the classical forward finite differences scheme operator (its explicit definition for 2D and 3D images can be found in Appendix A). Otherwise, in the more general situation where $u = (u_1, u_2, \ldots, u_K)$ is a sequence of images, we shall consider the sum of the TV of the individual images,

$$G(u) = \sum_{j=1}^{K} \mathrm{TV}(u_j)\,,$$

as a choice of generalized regularity term, as done in [25]. The use of the TV as a regularizer promotes the choice of piecewise constant images which is a kind of regularity promotion that allows for producing images with sharp discontinuities (and thus, sharp edges) and also relatively free of oscillations caused by noise [32]. Another, somehow more modern, explanation for the popularity of TV in image processing consists in interpreting (7) as the ℓ^1 norm of $\nabla_{\mathrm{FD}}u$, which can be seen as a sparsity promoting term for $\nabla_{\mathrm{FD}}(u)$. This term indeed promotes images u with sparse gradient $\nabla_{\mathrm{FD}}(u)$, reducing drastically the number of unknowns of the problem and allowing for accurate reconstruction even when the number of measurements is significantly smaller than the number pixels (or voxels) present in u. This principle underlies the research on compressed sensing. It should be noted, however, that the concept of compressed sensing also strongly relies on the usage of random sampling during the acquisition process, which is generally not considered in EPR imaging. Therefore, using the term "compressed sensing" to refer to TV regularized EPR image reconstruction is somewhat inaccurate.

2.6.2. *Condat-Vũ solver*

The PyEPRI package implements a generic solver dedicated to the minimization of TV-regularized energies. This solver boils down to a numerical scheme that generates a sequence $(u^{(n)})_{n\geq 0}$ that is proved to converge towards a minimizer of the targeted energy E. This numerical scheme is described in Appendix B and was proposed simultaneously by Condat and Vũ in [33, 34] before being further studied in a more general framework in [35]. Compared to other classical proximal algorithms, such as the celebrated Chambolle-Pock algorithm [36], proximal ADMM [37, 38], FISTA [39], or many others [31, 40], the Condat-Vũ algorithm takes advantage of the Lipschitz-differentiability of the data-fidelity term F defined in (6). More importantly, in the context of TV-regularized least-squares presented in Section 2.6.1, the Condat-Vũ solver involves the computation of

$$\nabla F(u^{(n)}) = A^* \circ A(u^{(n)}) - A^*(s)\,. \tag{8}$$

at each iteration $n \geq 0$ of the scheme. The computation of (8) corresponds in practice to the most computationally intensive part of the scheme. However, when $A = A_{X,\Gamma}$ or $A = \mathcal{A}_{X,\Gamma}$, the term $A^*\circ A(u^{(n)})$ can be computed efficiently using convolutions with precomputed Toeplitz kernels, as explained in Section 2.5. Besides, the backprojected term $A^*(s)$ involved in (8) never changes and can be computed once and for all at the beginning of the iterative optimization process. This means that we can evaluate (8) efficiently at each iteration of the scheme, as pointed out in [21, 24]. In comparison, other standard proximal algorithms would require the distinct evaluation of the projection and backprojection operators A and A^* at each iteration of the scheme, leading to significantly higher computational times per iteration.

Remark. The Condat-Vũ solver implemented in the PyEPRI package is generic and takes as input a function dedicated to the evaluation of the gradient of the data-fidelity term F, allowing for instance to consider data-fidelity terms different from (6). Some *higher level* functions specifically designed for EPR image reconstruction, both in the monosource and multisources frameworks, are also provided in the package and utilize some appropriate instances of this generic solver. These higher level functions enable non-expert users to more easily use the package for image reconstruction without needing to deal themselves with details related to the underlying optimization scheme. They are presented in sections 2.7.2 and 2.7.3.

2.7. *EPR imaging features*

In this section, we will describe the main EPR imaging features currently available in the PyEPRI package. Sections 2.7.1 and 2.7.2 focus on standard EPR imaging of a sample containing a single EPR species using filtered backprojection and TV-regularized least-squares. The example sample that we shall use to illustrate those EPR imaging features is made of tubes with various diameters filled with a solution of TAM, it is described in Figure 3. Imaging of multiple EPR sources will be presented in Section 2.7.3 and illustrated using a sample containing two distinct EPR species (TAM and TEMPO).

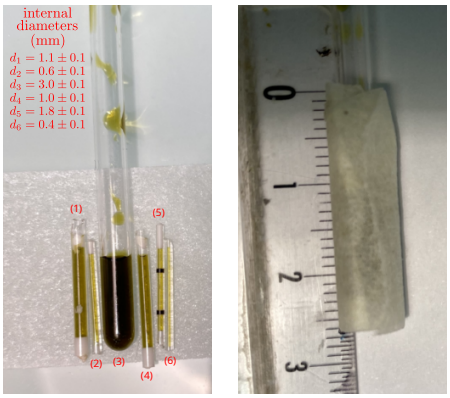


Figure 3: **Tubes filled with a TAM solution**. The sample is made of six tubes with various diameters filled with a 1 mM TAM solution (see left-hand side image). The tubes were wrapped together using masking tape (see right-hand side image) and a sequence containing $94 \times 94 = 8836$ three-dimensional projections (not displayed here) has been acquired using a L-band Bruker imager, using a constant field gradient magnitude of 20 G/cm. It should be noted that tubes (1) and (6) were badly sealed and leaked (partially for tube (1) and totally for tube (6)) during the experiment. This will affect the the upcoming image reconstructions.

2.7.1. *Single source imaging using filtered backprojection*

In the continuous setting and in the presence of a single EPR species X, one can derive an exact inversion formula that allows the reconstruction of U_X from the measurements $\mathcal{P}_{X,\gamma}(B)$ taken for any $B \in \mathbb{R}$ and for any γ on the unit sphere of $\mathbb{R}^d$. This formula is generally discretized to obtain an approximate inversion formula in the discrete setting. This method involves interpolating each projection and integrating the contributions of each interpolated projection over the sphere to obtain an image. This image reconstruction method is referred to as the filtered backprojection (FBP) as it can be interpreted as a modified backprojection of the input sequence of projections [6]. We refer to the documentation of the PyEPRI package for a mathematically detailed presentation of the FBP method it implements.

The FBP method, like many direct inversion techniques, tends to amplify noise present in the measured projections. In practical applications, this issue is addressed by preprocessing the projections to attenuate their high-frequency components, using for instance frequency

[6] as we mentioned earlier, computing the backprojection of a sequence of projections yields a blurry version of the underlying image, the filtered backprojection integrates an appropriate inverse filter into the backprojection process in order to get rid of this blurring operation.

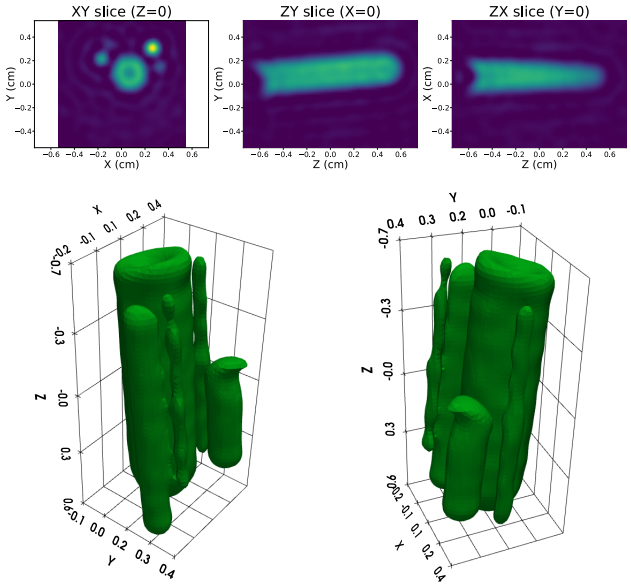


Figure 4: **3D reconstruction using filtered backprojection**. We display the FBP reconstruction (with pixel size = 200 µm) of the TAM solution depicted in Figure 3. In order to avoid dramatic noise amplification, a frequency cutoff was used to remove 90% of the highest frequency content of the projections, leading to a poorly resolved, but still exploitable, image. In essence, the trade-off between noise suppression and spatial resolution is a significant challenge when applying FBP in real-world scenarios.

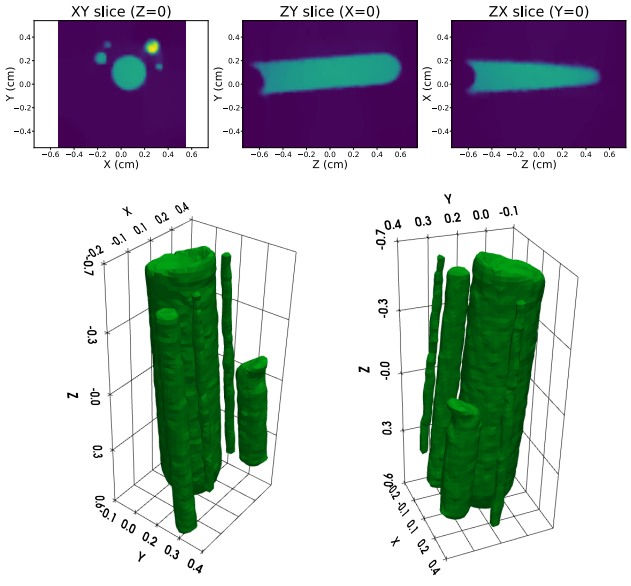


Figure 5: **3D reconstruction using TV-regularized least-squares**. We processed the same dataset as in Figure 4 but using the TV-regularized least-squares reconstruction model instead of the FBP. The reconstructed image has the same pixel-size (200 µm) as that reconstructed using FBP, but we can see that it exhibits a much higher perceptual resolution, with in particular sharp edges allowing a clear separation of the content of the thin tubes. In the general case, using total variation as a regularization term allows for the generation of smooth images (without oscillations caused by noise), while still preserving discontinuities to represent edges. For the same number of projections, this variational framework provides a reconstruction of higher quality than FBP. Besides, in favorable conditions, where the sample of interest is minimally textured (as is the case here), high-quality reconstructions can be achieved with a very limited number of measured projections.

apodization or cut-band filters. However, this filtering process inevitably leads to a loss of information, resulting in images that are overly smooth and poorly resolved, as illustrated in Figure 4. The advantage of this method lies in its simplicity and ease of implementation, which makes it widely used in practice, particularly when computational resources are limited or when a quick reconstruction is required.

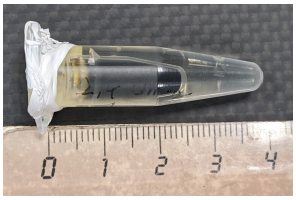
2.7.2. Single source imaging using TV-regularized least-squares

We display in Figure 5 the TV-regularized least-squares reconstruction of the TAM sample depicted in Figure 3. The reconstructed image is obtained by computing a minimizer of the TV-regularized energy described in Section 2.6.1, by taking $A = A_{\mathcal{X},\Gamma}$ as the direct operator. The energy minimization is handled using the Condat-Vũ solver presented in Section 2.6.2.

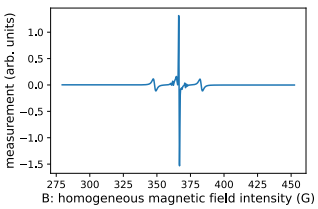
As mentioned earlier, the PyEPRI package implements a specific function dedicated to single EPR image reconstruction from a sequence of EPR projections. This function is built by combining basic components (so-called *low-level* functions) to obtain a *high-level* instance specifically designed for image reconstruction. This approach spares non-expert users from having to deal with generic functions (such as the generic Condat-Vũ solver, the fast implementation of the gradient of the data-fidelity least-squares term using Toeplitz kernels, etc.) being outside from their own fields of expertise, thus simplifying the use of the package for end-users.

2.7.3. Multiple sources imaging using TV-regularized least-squares

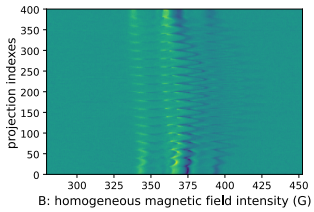
In the multisources framework, taking $A = \mathcal{A}_{\mathcal{X},\Gamma}$ in the TV-regularized least-squares model presented in Section 2.6.1 allows for addressing the source separation problem, that is the reconstruction of the images of each EPR species present in the sample, as done in [25]. This kind of problem can therefore also be numerically handled using the generic Condat-Vũ solver implemented in the PyEPRI package. Again, a *high-level* function specifically dedicated to the source separation problem was implemented in the PyEPRI package to avoid the somehow tricky configuration of the Condat-Vũ solver with fast gradient data-fidelity term computation using Toeplitz kernels. In order to illustrate the source separation feature, we will use a dataset taken from [25] and described in Figure 6. The source separation result obtained using the PyEPRI package is presented in Figure 7 (using slices) and in Figure 8 (using isosurfaces).



(a) picture of the sample

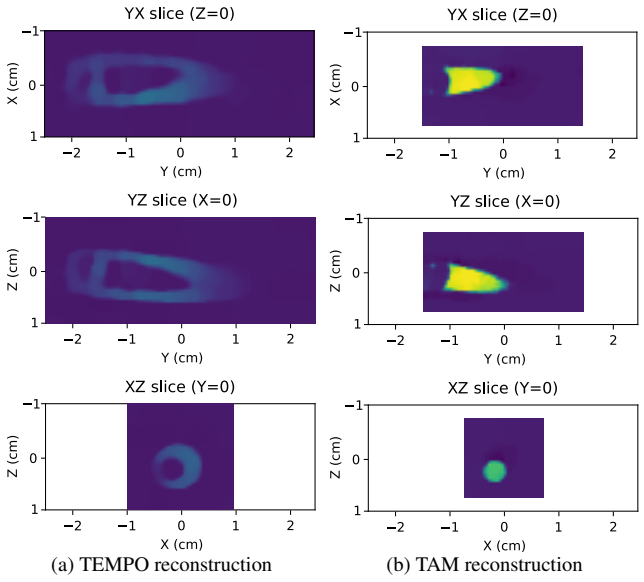


(b) zero-gradient spectrum of the sample



(c) measured projections

Figure 6: **TAM insert in TEMPO solution**. We display in (a) the picture of a sample made of a (small) eppendorf filled with a 12.5 mM solution of TAM immersed into a (large) eppendorf filled with a 14 mM solution of 4OH-TEMPO. We display in (b) and (c) the zero-gradient spectrum and the sequence of projections measured from the sample. The spectrum (b) gathers the contribution of a three-lined spectrum of 4OH-TEMPO and a single thin lined spectrum of TAM. The measured projections (c) were acquired with an L-band Bruker spectrometer using a constant field gradient magnitude of 20 G/cm and $20 \times 20 = 400$ three-dimensional orientations. This dataset was acquired at SFR ICAT University of Angers with the kind help of Dr. Raffaella Soleti.



(a) TEMPO reconstruction (b) TAM reconstruction

Figure 7: **TAM and TEMPO reconstructions (slices)**. The source separation model applied to the sequence of projections presented in Figure 6 (c) yields two distinct images, one represents the TEMPO concentration mapping, the other represents the TAM concentration mapping. The two images were reconstructed using the same pixel size (500 µm) but using different sizes to optimize the computation time (this feature was not present in [25]). We display in (a) the central slices along each axis of the TEMPO reconstruction, and we display in (b) the same slices of the TAM reconstruction. We can see that the shapes of the two eppendorfs are correctly retrieved, and the separation seems effective since no signal is detected in the TEMPO image in the area corresponding to the TAM eppendorf. A three-dimensional display of the two images using isosurfaces is also proposed in Figure 8.

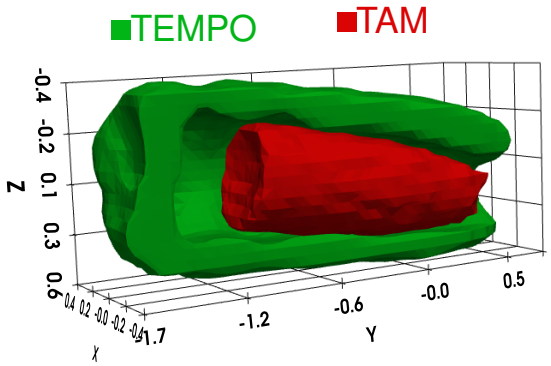


Figure 8: **TAM and TEMPO reconstructions (isosurfaces)**. We display here two isosurfaces, one coming from the TEMPO reconstruction (displayed in green), and the other one coming from the TAM reconstruction (displayed in red). The TEMPO isosurface was restricted to a half-space in order to make visible the hole that it contains and which is due to the TAM eppendorf (including its lid) visible in Figure 6.

3. CPU & GPU benchmarking

In this section we shall present some benchmarks about the execution time monitored for the different image reconstruction features of the PyEPRI package, using either a CPU or a GPU backend (see Section 2.1). The experiments presented here were led using a conventional laptop (Dell Precision 7680 with Ubuntu 24.04) equipped with an Intel Core i9-13950HX processor, 64 GB of RAM, and a NVIDIA RTX 4000 Ada graphics card with 12 GB of memory. The installation of the package and its dependencies was done using the Python Package Installer (PIP) following the instructions of the PyEPRI online documentation. Note that an advanced installation of the package and its dependencies is possible and may lead to reduced execution times, but we did not consider this option further as we believe most users will prefer the installation via PIP for simplicity. Furthermore, our GPU benchmarks include the data transfer from the RAM to the GPU memory. These transfer times are often significant, if not predominant, in the total computation time (as pointed out in [41]). While they are often excluded from benchmarks, we have chosen to include them in order to provide a more accurate representation of the actual computation times for the majority of users.

3.1. *Benchmarking for single source imaging using filtered backprojection*

We display in Figure 9 the computation times related to the FBP reconstruction at different resolutions of the "`fusillo-20091002`" (real) dataset embedded with the package (see the online documentation, and also the 3D surface displayed in Figure 1 (a) which actually corresponds to a reconstruction obtained from this real dataset). The computation times related to FBP reconstruction obtained from the "`tamtubes-20211201`" dataset (see Figure 3) are displayed in Figure 10.

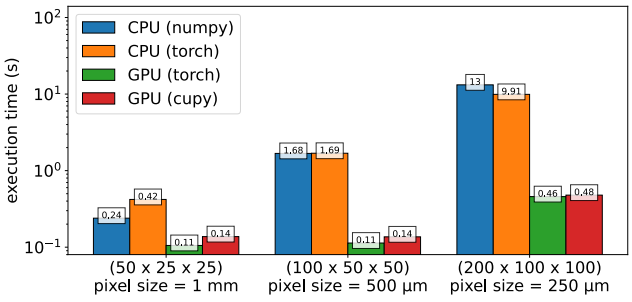


Figure 9: **Execution times for FBP reconstruction of the *fusillo***. The dataset is made of $31 \times 31 = 961$ three-dimensional projections containing $N_B = 500$ sample points each and acquired using magnetic field gradients with 14 G/cm magnitude. We performed the image reconstruction from this dataset using FBP at different resolutions and using the four backends implemented in the PyEPRI Package. We display the execution time for each experiment (the dimensions $N_1 \times N_2 \times N_3$ of the reconstructed image as well as its pixel size δ are displayed below each experiment). We can see that very fast reconstructions are achieved using the PyEPRI package. Besides, the execution time obtained with GPU backends are roughly 1.7 to 20 times faster than those obtained using CPU. The execution time for GPU backend is dominated by data transfer which explains why we observed the same execution time for the GPU reconstructions at $\delta = 1$ mm and $\delta = 500$ µm.

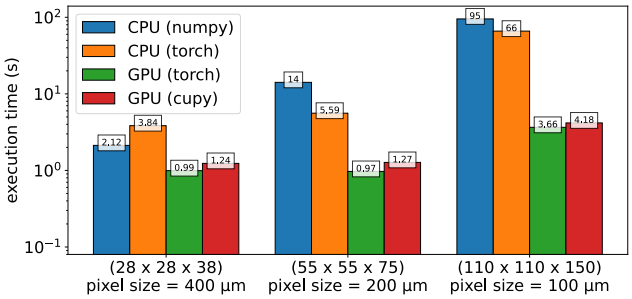


Figure 10: **Execution times for FBP reconstruction of TAM solution in tubes**. Same as in Figure 9 for the FBP reconstruction using the `tamtubes-20211201` dataset made of $94 \times 94 = 8836$ three-dimensional projections containing $N_B = 360$ sample points each. The reconstruction obtained at resolution $\delta = 200$ µm is displayed in Figure 4. This dataset is roughly six times larger than that considered in Figure 9 and leads to increased execution times with roughly the same factor. Even for high-resolution reconstructions, the execution times remain quite small compared to the time necessary to acquire such dataset using the fastest spectrometers. Again, using a GPU backend leads to interesting speed-up factors compared to the CPU runtime.

3.2. *Benchmarking for TV-regularized least-squares single source imaging*

We considered the same datasets as in Section 3.1 and performed the image reconstructions using the TV-regularized least-squares model

implemented in the PyEPRI package with the same targeted resolutions. In our experiments, we used a tolerance parameter $\texttt{tol} = 10^{-3}$ to automatically stop the iterations of the Condat-Vũ solver when the relative distance between two consecutive scheme iterates is below $\texttt{tol}$ (see Appendix Appendix B). This setting enables fast reconstruction of good quality images but it must be noted that achieving accurate convergence towards the solution of the underlying minimization problem (4) involves in practice significantly lower settings of the tolerance parameter (typically $\texttt{tol} \leq 10^{-5}$), which can increase the number of iterations and thus the overall execution time. The observed execution times are presented in Figure 11 and Figure 12.

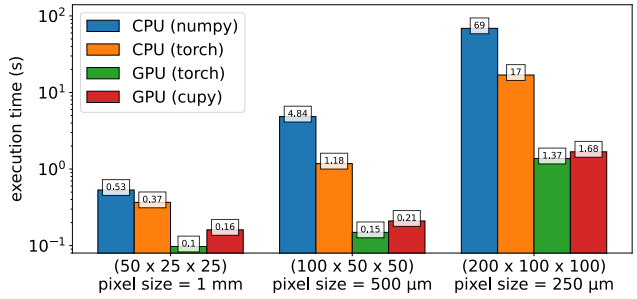


Figure 11: **Execution times for TV-regularized least-squares reconstruction of the *fusillo***. We display here the execution times measured when performing image reconstruction for different resolutions from the *fusillo* dataset using the TV-regularized least-squares model implemented in PyEPRI. Those execution times can be compared to those obtained using FBP reconstruction over the same dataset presented in Figure 9. We can observe that the TV-based reconstruction model can result in slightly higher execution times on this dataset (up to a factor of 5 depending on the selected backend), but it remains very fast nonetheless (especially with GPU backends) and provides a significant improvement in terms of image quality in practice.

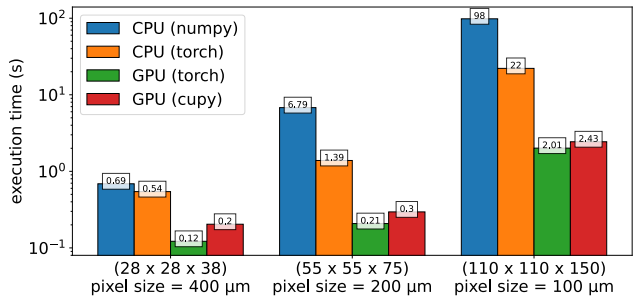


Figure 12: **Execution times for TV-regularized least-squares reconstruction of TAM solution in tubes**. We display the execution times obtained when addressing the reconstruction of the TAM solution contained in tubes (see Figure 3) using the TV-based image reconstruction algorithm of the PyEPRI package. Those execution times can be compared to those presented in Figure 10. The image reconstructed with resolution $\delta = 200\,\mu\text{m}$ is displayed in Figure 5 and can be compared to that obtained using FBP, displayed in Figure 4. We can see that, on this dataset (which is larger than the *fusillo* one), the TV-based image reconstruction model yields faster execution times than FBP. However, this should be taken with caution, as in the case of the TV-based algorithm, a stricter adjustment of the tolerance parameter in order to achieve accurate convergence can increase the execution time.

3.3. Benchmarking for TV-regularized least-squares multiple sources imaging

In this section, we focus on the execution time related to the TAM and TEMPO separation from the sample presented in Figure 6 using the TV-regularized least-squares multisources image reconstruction algorithm implemented in the PyEPRI package. For this dataset, fast and visually acceptable reconstruction results can be obtained by using $\texttt{tol} = 10^{-4}$. However, we observed that the quality of the image separation could benefit from the use of a fine convergence criterion. For that reason, we performed our experiments using a tolerance parameter $\texttt{tol} = 10^{-5}$, leading to longer execution times than in the single image reconstruction framework. The results are presented in Figure 13.

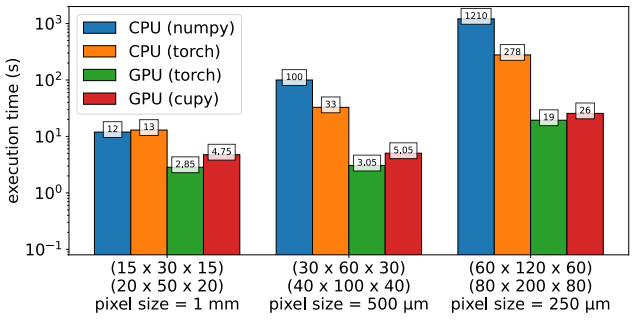


Figure 13: **Execution times for multiple EPR sources imaging**. We performed the joint reconstruction of the TAM and TEMPO images from the sample presented in Figure 6. The reconstruction was done for different targeted resolutions using the multiple EPR imaging algorithm implemented in the PyEPRI package. Below each experiment, we display the size $(N_1^{\text{TAM}} \times N_2^{\text{TAM}} \times N_3^{\text{TAM}})$ of the reconstructed TAM image (top), the size $(N_1^{\text{TEMPO}} \times N_2^{\text{TEMPO}} \times N_3^{\text{TEMPO}})$ of the reconstructed TEMPO image (middle), and the pixel size δ common to both images (bottom). We used $\texttt{tol} = 10^{-5}$ as stopping criterion as we observed that this setting improved the separation result on this dataset. The images displayed in Figure 7 and Figure 8 correspond to the reconstruction obtained with pixel size $\delta = 500\,\mu\text{m}$ and $\texttt{tol} = 10^{-5}$. Note that the setting $\texttt{tol} = 10^{-4}$ leads to roughly twice faster reconstructions and still visually acceptable separation results. We can see that even such challenging image reconstruction task can be done in a relatively low amount of computation time using the PyEPRI package.

4. Conclusion and perspectives

This package has been carefully developed and tested, we sincerely hope it will useful to the EPR community. We strongly encourage interested users to contact us, as we will be happy to assist them if they encounter any difficulties in using the package. In a next release, we plan to add support for 4D spectral-spatial EPR imaging: this involves implementing a fast and accurate 4D projection operator, its adjoint, and use those low-level operators to build some more complex image reconstruction pipelines. Our long-term goal is to maintain the PyEPRI package up-to-date and enrich it with robust and efficient EPR image reconstruction features, keeping the initial spirit of providing both user-friendly and developer-friendly tools. As Python provides a simple and effective framework to build, combine and share features, we think that this package opens room for building and sharing new advanced EPR image processing applications based on the many tools developed and shared by the image processing and machine learning communities.

Appendix A. Forward finite differences scheme

Let $u \in \mathbb{R}^{\Omega}$ be a d-dimensional discrete image with discrete domain $\Omega \subset \mathbb{Z}^d$ (in practice, $d = 2$ or 3). The forward finite differences of u correspond to the signal $\nabla_{\text{FD}}(u) = \left(\nabla_{\text{FD}}^{(1)}(u), \ldots, \nabla_{\text{FD}}^{(d)}(u)\right) \in (\mathbb{R}^{\Omega})^d$ defined by, for all $i \in \{1, 2, \ldots, d\}$,

$$\forall k \in \Omega, \quad \nabla_{\text{FD}}^{(i)}(u)(k) = \begin{cases} u(k+\delta_i) - u(k) & \text{if } k + \delta_i \in \Omega \\ 0 & \text{otherwise,} \end{cases}$$

where $\delta_i \in \mathbb{R}^d$ is the vector containing zero everywhere excepting the i-th entry that takes the value 1.

Appendix B. Generic Condat-Vũ solver

Let $K \geq 1$ and let $\mathcal{E} = \mathbb{R}^{\Omega_1} \times \cdots \times \mathbb{R}^{\Omega_K}$ denote the set of sequence of images $u = (u_1, u_2, \ldots, u_K)$ with domains Ω_1, Ω_2, …, Ω_K. The

generic Condat-Vũ solver implemented in the PyEPRI package aims to compute a minimizer of an energy $E : \mathcal{E} \to \mathbb{R}$ of the type

$$\forall u = (u_1, \dots, u_K) \in \mathcal{E}, \quad E(u) = F(u) + \lambda \sum_{j=1}^{K} \mathrm{TV}(u_j),$$

where $F : \mathcal{E} \to \mathbb{R}$ denotes a differentiable function with Lipschitz continuous gradient ∇F. The single image framework corresponds to the case $K = 1$ but we will describe the solver in the more general case, where $K \geq 1$. Notice that, in this framework, the gradient of the data-fidelity term, $\nabla F(u)$, is made of K elements

$$\forall u = (u_1, \dots, u_K) \in \mathcal{E}, \quad \nabla F(u) = \left(\frac{\partial F(u)}{\partial u_1}, \frac{\partial F(u)}{\partial u_2}, \dots, \frac{\partial F(u)}{\partial u_K} \right) \in \mathcal{E}.$$

Given two time step parameters $\tau > 0$, $\sigma > 0$, and some initial variables $u^{(0)} \in \mathcal{E}$, $\overline{u}^{(0)} \in \mathcal{E}$ and $p^{(0)} = (p_1^{(0)}, p_2^{(0)}, \dots, p_K^{(0)}) \in \prod_{j=1}^{K} \left(\mathbb{R}^{\Omega_j}\right)^d$, the scheme consists in iterating for $n \geq 0$,

$$\begin{cases} p_j^{(n+1)} = \Pi_{\mathcal{B}} \left(p_j^{(n)} + \sigma \lambda \nabla_{\mathrm{FD}} \left(\overline{u}_j^{(n)} \right) \right) & \text{for } 1 \leq j \leq K \quad \text{(B.1a)} \\ u_j^{(n+1)} = u_j^n - \tau \left(\dfrac{\partial F(u^{(n)})}{\partial u_j} - \lambda \mathrm{div}_{\mathrm{FD}} \left(p_j^{(n+1)} \right) \right) & \text{for } 1 \leq j \leq K \quad \text{(B.1b)} \\ \overline{u}_j^{(n+1)} = 2\, u_j^{(n+1)} - u_j^{(n)} & \text{for } 1 \leq j \leq K \quad \text{(B.1c)} \end{cases}$$

where we have set

$$\forall p_j \in (\mathbb{R}^{\Omega_j})^d, \ \forall k \in \Omega_j, \quad \Pi_{\mathcal{B}}(p_j)(k) = \frac{p_j(k)}{\max(1, \|p_j(k)\|_2)},$$

and $\mathrm{div}_{\mathrm{FD}} = -\nabla_{\mathrm{FD}}^*$ denotes the opposite adjoint of the forward finite differences operator ∇_{FD}.

The primal and dual steps τ and σ involved in (B.1) are automatically tuned according to [35, Theorem 1] in order to ensure the convergence of the numerical scheme towards a minimizer of E. The PyEPRI implementation of Scheme (B.1) provides an optional tolerance parameter `tol` which can be used to stop the iterations of the scheme when

$$\|u^{(n+1)} - u^{(n)}\|_2 \leq \texttt{tol} \cdot \|u^{(n)}\|_2, \quad \text{(B.2)}$$

i.e., when the relative distance between two consecutive iterates is less than `tol`. The implemented function also enable the evalution of the energy of the iterates $E(u^{(n)})$ which is helpful to monitor the convergence of the scheme towards a minimizer of E.

References

[1] L. J. Berliner, H. Fujii, Magnetic resonance imaging of biological specimens by electron paramagnetic resonance of nitroxide spin labels, Science 227 (4686) (1985) 517–519. doi:10.1126/science.2981437.

[2] P. Kuppusamy, P. Wang, J. L. Zweier, M. C. Krishna, J. B. Mitchell, L. Ma, C. E. Trimble, C. J. C. Hsia, Electron paramagnetic resonance imaging of rat heart with nitroxide and polynitroxyl-albumin, Biochemistry 35 (22) (1996) 7051–7057. doi:10.1021/bi952857s.

[3] E. Martyna, R. Bell, D. Hleihel, E. D. Barth, C. McFaul, C. R. Haney, J. Bielanska, K. Pustelny, K.-H. Ahn, C. A. Pelizzari, M. Kocherginsky, H. J. Halpern, Electron paramagnetic resonance oxygen image hypoxic fraction plus radiation dose strongly correlates with tumor cure in FSa fibrosarcomas, International Journal of Radiation Oncology Biology Physics 71 (2) (2008) 542–549. doi:10.1016/j.ijrobp.2008.02.022.

[4] D. A. Komarov, Y. Ichikawa, K. Yamamoto, N. J. Stewart, S. Matsumoto, H. Yasui, I. A. Kirilyuk, V. V. Khramtsov, O. Inanami, H. Hirata, In vivo extracellular pH mapping of tumors using electron paramagnetic resonance, Analytical Chemistry 90 (23) (2018) 13938–13945, pMID: 30372035. doi:10.1021/acs.analchem.8b03328.

[5] A. Taguchi, S. DeVience, B. Driesschaert, V. V. Khramtsov, H. Hirata, In vitro simultaneous mapping of the partial pressure of oxygen, pH and inorganic phosphate using electron paramagnetic resonance, Analyst 145 (9) (2020) 3236–3244. doi:10.1039/d0an00168f.

[6] B. Gallez, Oxygenation status in normal tissues, pathological tissues and malignant tumors: A pO2 database based on electron paramagnetic resonance (epr) oximetry measurements, Applied Magnetic Resonance 52 (10) (2021) 1395–1450. doi:10.1007/s00723-021-01358-7.

[7] K. Kimura, N. Iguchi, H. Nakano, H. Yasui, S. Matsumoto, O. Inanami, H. Hirata, Redox-sensitive mapping of a mouse tumor model using sparse projection sampling of electron paramagnetic resonance, Antioxidants & Redox Signaling 36 (1-3) (2022) 57–69, pMID: 33847172. doi:10.1089/ars.2021.0003.

[8] M. C. Emoto, H. Sato-Akaba, Y. Matsuoka, K. ichi Yamada, H. G. Fujii, Non-invasive mapping of glutathione levels in mouse brains by in vivo electron paramagnetic resonance (EPR) imaging: Applied to a kindling mouse model, Neuroscience Letters 690 (2019) 6–10. doi:10.1016/j.neulet.2018.10.001.

[9] Y. Takakusagi, R. Kobayashi, K. Saito, S. Kishimoto, M. C. Krishna, R. Murugesan, K.-i. Matsumoto, EPR and related magnetic resonance imaging techniques in cancer research, Metabolites 13 (1) (2023). doi:10.3390/metabo13010069.

[10] C. Simon, C. Lion, H. Ahouari, H. Vezin, S. Hawkins, C. Biot, EPR imaging of sinapyl alcohol and its application to the study of plant cell wall lignification, Chemical Communications 57 (2021) 387–390. doi:10.1039/D0CC05218C.

[11] M. Sathiya, J.-B. Leriche, E. Salager, D. Gourier, J.-M. Tarascon, H. Vezin, Electron paramagnetic resonance imaging for real-time monitoring of Li-ion batteries, Nature communications 6 (2015). doi:10.1038/ncomms7276.

[12] A. Niemöller, P. Jakes, R.-A. Eichel, J. Granwehr, EPR imaging of metallic lithium and its application to dendrite localisation in battery separators, Scientific reports 8 (1) (2018) 14331. doi:10.1038/s41598-018-32112-y.

[13] F. Geng, Q. Yang, C. Li, B. Hu, C. Zhao, M. Shen, B. Hu, Operando EPR and EPR imaging study on a NaCrO2 cathode: Electronic property and structural degradation with Cr dissolution, The Journal of Physical Chemistry Letters 12 (2) (2021) 781–786, pMID: 33410689. doi:10.1021/acs.jpclett.0c03327.

[14] S. Kang, X. Lou, M. Shen, F. Geng, B. Hu, Visualizing lithium deposition and identifying two types of dendrites in extreme-fast-charging full cells across the entire lifespan by operando EPR and EPR imaging, The Journal of Physical Chemistry Letters 15 (50) (2024) 12248–12256, pMID: 39635912. doi:10.1021/acs.jpclett.4c03022.

[15] M. Abou Fadel, X. Zhang, A. De Juan, R. Tauler, H. Vezin, L. Duponchel, Extraction of pure spectral signatures and corresponding chemical maps from EPR imaging data sets: Identifying defects on a CaF_2 surface due to a laser beam exposure, Analytical chemistry 87 (2015) 3929–3935. doi:10.1021/ac504733u.

[16] L. Binet, D. Gourier, S. Derenne, Potential of epr imaging to detect traces of primitive life in sedimentary rocks, Earth and Planetary Science Letters 273 (2008) 359–366. doi:10.1016/j.epsl.2008.06.052.

[17] C. Johnson, D. McGarry, J. Cook, N. Devashayam, J. Mitchell, S. Subramanian, M. Krishna, Maximum entropy reconstruction methods in electron paramagnetic resonance imaging, Annals of Operations Research 119 (2003) 101–118. doi:10.1023/A:1022978322046.

[18] M. Tseitlin, T. Czechowski, S. S. Eaton, G. R. Eaton, Regularized optimization (RO) reconstruction for oximetric EPR imaging, Journal of Magnetic Resonance 194 (2) (2008) 212–221. doi:10.1016/j.jmr.2008.07.002.

[19] Y. Ikebata, H. Sato-Akaba, T. Aoyama, H. Fujii, K. Itoh, H. Hirata, Resolution-recovery for EPR imaging of free radical molecules in mice, Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 62 (3) (2009) 788–795. doi:10.1002/mrm.22029.

[20] D. Johnson, R. Ahmad, G. He, A. Samouilov, J. Zweier, Compressed sensing of spatial electron paramagnetic resonance imaging, Magnetic Resonance in Medecine 72 (3) (2014) 893–901. doi:10.1002/mrm.24966.

[21] S. Durand, Y.-M. Frapart, M. Kerebel, Electron paramagnetic resonance image reconstruction with total variation and curvelets regularization, Inverse Problems 33 (11) (2017) 114002. doi:10.1088/1361-6420/aa8412.

[22] R. Ahmad, A. Samouilov, J. Zweier, Accelerated dynamic EPR imaging using fast acquisition and compressive recovery, Journal of Magnetic

Resonance 273 (2016) 105–112. doi:10.1016/j.jmr.2016.10.001.
[23] Z. Qiao, G. Redler, B. Epel, H. Halpern, A balanced total-variation-Chambolle-Pock algorithm for EPR imaging, Journal of Magnetic Resonance 328 (2021). doi:10.1016/j.jmr.2021.107009.
[24] R. Abergel, M. Boussâa, S. Durand, Y.-M. Frapart, Electron Paramagnetic Resonance Image Reconstruction with Total Variation Regularization, Image Processing On Line 13 (2023) 90–139. doi:10.5201/ipol.2023.414.
[25] M. Boussâa, R. Abergel, S. Durand, Y.-M. Frapart, Ultrafast multiple paramagnetic species EPR imaging using a total variation based model, Journal of Magnetic Resonance 357 (2023) 107583. doi:10.1016/j.jmr.2023.107583.
[26] Bruker, Xepr version 2.6b.36, https://www.bruker.com/de/products-and-solutions/mr/epr-instruments/epr-research-instruments/xepr-software.html, (2009).
[27] S. R. Sweger, S. Pribitzer, S. Stoll, Bayesian probabilistic analysis of DEER spectroscopy data using parametric distance distribution models, The Journal of Physical Chemistry A 124 (30) (2020) 6193–6202. doi:10.1021/acs.jpca.0c05026.
[28] A. H. Barnett, J. Magland, L. af Klinteberg, A parallel nonuniform fast fourier transform library based on an “exponential of semicircle" kernel, SIAM Journal on Scientific Computing 41 (5) (2019) C479–C504. doi:10.1137/18M120885X.
[29] A. H. Barnett, Aliasing error of the $\exp\left(\beta\sqrt{1-z^2}\right)$ kernel in the nonuniform fast fourier transform, Applied and Computational Harmonic Analysis 51 (2021) 1–16. doi:10.1016/j.acha.2020.10.002.
[30] Y.-h. Shih, G. Wright, J. Andén, J. Blaschke, A. H. Barnett, cuFINUFFT: a load-balanced GPU library for general-purpose nonuniform FFTs, in: 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 688–697. doi:10.1109/IPDPSW52791.2021.00105.
[31] L. Condat, D. Kitahara, A. Contreras, A. Hirabayashi, Proximal splitting algorithms for convex optimization: A tour of recent advances, with new twists, SIAM Review 65 (2) (2023) 375–435. doi:10.1137/20M1379344.
[32] A. Chambolle, T. Pock, An introduction to continuous optimization for imaging, Acta Numerica 25 (2016) 161–319. doi:10.1017/S096249291600009X.
[33] L. Condat, A Primal-Dual Splitting Method for Convex Optimization Involving Lipschitzian, Proximable and Linear Composite Terms, Journal of Optimization Theory and Applications 158 (2) (2013) 460–479. doi:10.1007/s10957-012-0245-9.
[34] B. C. Vũ, A splitting algorithm for dual monotone inclusions involving cocoercive operators, Advances in Computational Mathematics 38 (3) (2013) 667–681. doi:10.1007/s10444-011-9254-8.
[35] A. Chambolle, T. Pock, On the ergodic convergence rates of a first-order primal-dual algorithm, Mathematical Programming 159 (1) (2016) 253–287. doi:10.1007/s10107-015-0957-3.
[36] A. Chambolle, T. Pock, A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging, Journal of Mathematical Imaging and Vision 40 (1) (2011) 120–145. doi:10.1007/s10851-010-0251-1.
[37] D. Gabay, B. Mercier, A dual algorithm for the solution of nonlinear variational problems via finite element approximation, Computers & mathematics with applications 2 (1) (1976) 17–40. doi:10.1016/0898-1221(76)90003-1.
[38] J. Eckstein, Some saddle-function splitting methods for convex programming, Optimization Methods and Software 4 (1) (1994) 75–83. doi:10.1080/10556789408805578.
[39] A. Beck, M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM Journal on Imaging Sciences 2 (1) (2009) 183–202. doi:10.1137/080716542.
[40] N. Parikh, S. Boyd, Proximal algorithms, Foundations and Trends® in Optimization 1 (3) (2014) 127–239. doi:10.1561/2400000003.
[41] Z. Qiao, G. Redler, B. Epel, Y. Qian, H. Halpern, Implementation of GPU-accelerated back projection for EPR imaging, Journal of X-ray science and technology 23 (4) (2015) 423–433. doi:10.3233/XST-150498.