Georg-August-UniversitÃd't GÃűttingen

Documentation

PRwSC Matlab Toolbox

Phase Retrieval with Sparsity Constraints

Stefan Loock[*]

March 9, 2017

Contents

1 Installation **1**
1.1 Acknowledgements 2

2 Introduction **2**
2.1 Solving the Problem: Feasibility Formulation 2
2.2 Solving the Problem: Algorithms 3
2.3 Solving the Problem: Constraints 4

3 Design **5**
3.1 The input_data structure 6
3.2 Algorithms 8
3.3 Drivers 9

4 Examples **11**
4.1 Simulated Data: Phase Object with Poisson Noise 11

1 Installation

To use the PRwSC toolbox simply make sure that the folder which contains the toolbox is on the MATLAB path. In order to be able to use the examples which use the shearlet transform, you need to download ShearLab 3D from `http://www.shearlab.org`. Currently, the toolbox is tested to work with ShearLab 3D version 1.1 which currently is available under `http://shearlab.org/files/software/ShearLab3Dv11.zip`. An extensive description about the shearlet transform and its implementation in ShearLab 3D is given in [6].

[*]s.loock@math.uni-goettingen.de

Make sure that the shearlet toolbox is on your MATLAB path as well in order to function properly.

1.1 Acknowledgements

This work has been funded by the Deutsche Forschungsgesellschaft (DFG) under the Sonderforschungsbereich 755 *Nanoscale Photonic Imaging*.

The file `fresnel_prob.m` which implements the discrete Fresnel transform used in the toolbox is courtesy of Martin Krenkel. Please refer to [5] for more information.

The file `cell256.m` which is used for the simulation examples is courtesy of Klaus Giewekemeyer, see [4, 3].

If you use this toolbox, please cite the publications [7, 8].

2 Introduction

The phase retrieval problem in the general sense means to recover a function or a vector from its modulus and it occurs in several disciplines such as astronomy, crystallography, and x-ray imaging. While the problems in the different disciplines differ slightly and this toolbox is designed for phase retrieval in the context of x-ray imaging, the toolbox should be general enough to also cover different applications such as astronomy.

Most applications furthermore involve the propagation of a wave between the object plane and a measurement plane. This document will only be concerned with the discrete version of the problem and hence the quantities we are dealing with will be matrices and vectors. For sake of simplicity we assume that the measurements and objects to be recovered are vectors in $\mathbb{C}^d$ or $\mathbb{R}^d$. The linear mappings modeling the propagation of a wave will then be unitary matrices in $\mathbb{C}^{d\times d}$. Note that this setting can easily be generalized for when the measured quantities are of a higher dimension, e.g. when they are matrices which relates to two-dimensional images.

Let us denote by $\mathbb{R}_+$ the set of all positive real numbers including zero. We denote the measurements by $m \in \mathbb{R}_+^d$, i.e., the right-hand-side of the non-linear equation

$$|Ux| = m \tag{1}$$

where $|\cdot|$ denotes the point-wise modulus and U a unitary mapping modeling the wave propagation. In general, $x \in \mathbb{C}^d$ and hence, $U \in \mathbb{C}^{d\times d}$ with $U^{-1} = U^*$ since we assume that U is unitary. For a more detailed introduction of the problem, we refer to [7] and the references therein.

2.1 Solving the Problem: Feasibility Formulation

This toolbox is a framework to solve problems of the form (1) using projection algorithms or more generally speaking, algorithms which use proximity algorithms in some alternating fashion to solve an optimization problem where the solution is a special solution of (1).

The set of all admissible solutions to (1) will be denoted by

$$M := \left\{ x \in \mathbb{C}^d \mid |Ux| = m \right\}. \tag{2}$$

In the presence of noisy measurements one may want to enlarge this set to

$$M_\varepsilon := \left\{ x \in \mathbb{C}^d \mid d\left(|Ux|, m\right) \leq \varepsilon \right\} \tag{3}$$

where $d(\cdot,\cdot) : \mathbb{C}^d \times \mathbb{C}^d \to \mathbb{R}_+$ denotes an appropriate distance function. If additional information about x is available, this can be incorporated into a constraint set C. A prominent example of such a set is

$$C := \left\{ x \in \mathbb{C}^d \mid \operatorname{supp} x \subset \Lambda \right\} \tag{4}$$

with a known index-set $\Lambda \subset \{1, \ldots, d\}$.[1]

Following this notation, the task is to find $x \in M \cap C$ or, if $M \cap C = \emptyset$, a solution x which is simultaneously close to both sets. This, of course, translates to the case where noise is present and M is replaced by M_ε.

2.2 Solving the Problem: Algorithms

There is a wide range of algorithms which solves phase retrieval problems using projection algorithms onto the sets M and C. For an overview, we refer to [1] and the references therein. In the following, P_A denotes the projection operator onto a set A. If A is non-convex and P_A therefore set-valued, with slight abuse of notation, we donate by P_A any selection out of the projection onto A. The following algorithms are part of the PRwSC toolbox:

- Method of alternating projections (MAP[2]):

$$x^{k+1} = P_M P_C x^k \tag{5}$$

- Relaxed averaged alternating reflections algorithm (RAAR[3]):

$$x^{k+1} = \frac{\beta_k}{2} \left(R_M R_C + I\right) x^k + (1 - \beta_k) x^k \tag{6}$$

  where $\beta_k \in (0, 1)$ and $R_M = 2P_M - I$, $R_C = 2P_C - I$.

Following the template given by `MAP.m`, one can easily implement further algorithms. For more details, we refer to section 3.

[1] The support of x, or $\operatorname{supp} x$, describes the set of all indices j such that $x_j \neq 0$.

[2] Implemented in `MAP.m`.

[3] Implemented in `RAAR.m`.

2.3 Solving the Problem: Constraints

The PRwSC toolbox contains several proximity / projections operators which are used as constraints. Depending on the type of objects, those constraints are

- support constraint
- range constraints:
  - positivity constraint
  - negativity constraint
  - amplitude constraint
- sparsity constraints:
  - shearlet soft-thresholding
  - shearlet smooth-hard shrinkage

The problem types that are available are *phase*, *amplitude*, and *mixed*. The problem type is specified in the field `input_data.problem.type`. The constraint type is specified in the field `input_data.problem.constraint.name`.

3 Design

The toolbox is structured in the following subdirectories:

- `Algorithms`: Contains implementations of the different algorithms which itself use the appropriate projection and proximity operators as defined in the `input_data` object. In each iteration, an error is computed which will be returned in the `output_data` object together with the reconstructed object.

- `Drivers`: This directory contains problem specific driver files. Here, for each problem, an `input_data` object is constructed which contains all necessary information needed to solve the problem.

- `InputData`: Contains the corresponding data for each problem. For simulation purposes, this will be the exact solution of the problem from which an appropriate right-hand-side m will be constructed using the driver file or the experimental data obtained in an experiment.

- `OutputData`: The folder to store reconstructions and plots calculated using the algorithms.

- `Processors`: Methods handling the input data stored in the `InputData` directory which will be called from the driver file. This is the place to write your own methods to handle your experimental data in a suitable way and make it accessible to the toolbox.

- `ProxOperators`: Contains all necessary projection and proximity operators which will be used by the algorithms such as the projection onto the set M, the set C and variants of it but also proximity operators which are used as sparsity constraints.

- `Utilities`: The folder containing all little helper functions such as the wave propagators, the thresholding functions, methods to compute an error such as the distance function mentioned above and others.

There are two main (global) variables (or structures) used in the toolbox. The `input_data` structure contains all information and data necessary to use the toolbox. The `output_data` structure contains all information and objects computed by the toolbox. In the following, we describe what information these two structures contain and which of them are optional.

3.1 The input_data structure

Field	Values	Description
data	$\mathbb{R}^{N\times M}$	data used for simulation
problem	structure	contains problem type and information
algorithm	structure	information on the algorithm and parameters
physicalParameters	struct	physical parameters of the object
Psi	$\psi \in \mathbb{C}^{N\times M}$	wave function to be reconstructed
noise	structure	information on noise type and intensity
system	structure	shearlet system
initialGuess	$x^{(0)} \in \mathbb{C}^{N\times M}$	initial guess

Table 1: The input_data field and its structure.

Field	Values	Description
type	{'amplitude', 'phase', 'mixed'}	type of the object determines wave function ψ
propagator	{'Fourier', 'Fresnel'}	determines which transform U in (1) is used
Fx	$(0,\infty)$	Fresnel number in horizontal direction
Fy	$(0,\infty)$	Fresnel number in vertical direction
data	$\mathbb{R}^{N\times M}$	measurements, right-hand-side in (1)
constraint	structure	constraint type and information

Table 2: The input_data.problem field and its structure.

Field	Values	Description
name	'support'	simple support constraint on the object
	'support and range'	simple and a range constraint (e.g., positivity)
	'shearlet thresholding'	thresholding of shearlet coefficients
	'shearlet thresholding and range'	thresholding in combination with range constraint
info	'soft thresholding'	type of thresholding: soft-thresholding
	'smooth hard'	type of thresholding: smooth-hard shrinkage
range	'positive', 'negative'	specifies range information
parameter	'constant'	constant thresholding parameter $\gamma_k \equiv \gamma_0$
	'dynamic'	thresholding parameter $\gamma_k = \gamma_0 / k$
gamma0	$(0,\infty)$	initial thresholding parameter
support	$\#\text{supp} \times 1$	index-set of the assumed support of the object

Table 3: The input_data.problem.constraint field and its structure.

Field	Values	Description
name	’RAAR’	uses the RAAR algorithm
	’MAP’	uses the method of alternating projections
parameter	’constant’	uses a constant relaxation parameter $\beta_k \equiv \beta_0$
	’dynamic’	uses a variable relaxation parameter
beta	structure	contains additional parameters used for β_k
maxit	$(0,\infty)$	maximal number of iterations to perform
error	$(0,\infty)$	approximation error, exits algorithms if achieved
metric	’norm’	$\|x^{k+1} - x\|_F$ when x is known
		$\|x^{k+1} - x^k\|_F$ when x is unknown
	’shadow’	$\|P_S x^k - x\|_F$ when x is known
		$\|P_S x^k - P_M x^k\|$ when x is unknown
	’setdistance’	$\|P_M P_S x^k - P_S x^k\|_F^2 / \|P_S x^k\|_F^2$

Table 4: The input_data.problem.algorithm field and its structure.

Field	Values	Description
delta		decrement of real part of refractive index
beta		imaginary part of refractive index
tau		lateral thickness
lambda		wavelength
P		phase distribution
A		amplitude distribution

Table 5: The input_data.problem.physicalParameters field and its structure.

Field	Values	Description
type	’poisson’	Poisson data
	’gaussian’	additive gaussian noise
	’none’	exact data
level		intensity of the noise
error		size of the ε in M_ε

Table 6: The input_data.problem.noise field and its structure.

The shearlet system stored in `input_data.system` is described in [7, Table 3.1].

3.2 Algorithms

As we briefly touched on in subsection 2.2, the problem is solved using projection algorithms (and generalizations thereof). If you want to implement your own algorithm, the `MAP.m` file is a good, minimal template to do so. Every algorithm should only have one input variable (`input_data`) and one output variable (`output_data`). The structure of the `input_data` object is discussed above. At this point, the `output_data` structure has the following fields:

Field	Type	Description
reconstruction	size of the object	reconstruction x^N
error	N	error for each iteration step

Table 7: The input_data.problem.noise field and its structure.

Your own implementation of algorithms should conform with some simple guidelines:

- The algorithm has one input (`input_data`) and one output (`output_data`) variable. The structure of those variables should conform with those mentioned above.
- The iteration variable should be named `x`, temporary variables `tmpN` starting with $N = 1$ or `x_old` if they are a placeholder for the last iteration of `x`.
- The algorithm should calculate an error of some form and use the `input_data.algorithm.error` variable to terminate if this error is achieved. It furthermore should respect the `input_data.algorithm.maxit` and finish after that number of iterations independently of the error.
- The error for each iteration should be returned in the corresponding field as a vector.

```
function output_data = MAP(input_data)
% Input : input_data structure
% Output: output_data structure

        x    = input_data.initialGuess;
        k = 1;
        output_data.error = zeros(input_data.algorithm.maxit);
        error = inf;

        while ( k < input_data.algorithm.maxit || error < input_data.algorithm.error )

          gamma = computeShrinkageParameter(input_data, k);
          tmp1  = magnitudeProjection( input_data, x);
          x_old = x;
          x     = constraintProximityOperator(input_data, tmp1, gamma);
          error = computeError(input_data, x, x_old, tmp1);
          output_data.error(k) = error;
          k = k+1;
          itMsg = sprintf('Iteration: %d, Gamma: %.6f, Error: %.3f, Constraint: %s',
                        k, gamma, error, input_data.problem.constraint.name);
          disp(itMsg);
        end

        output_data.error = output_data.error(1:k-1);
        output_data.reconstruction = x;

end
```

3.3 Drivers

The driver files are used to create a `input_data` variable according to the guidelines described above. Depending weather you are using experimental or simulated data, the creation of the driver file may differ slightly.

The toolbox contains a demo file named `simulatedDataCellImage.m`. You may follow the naming convention which first asserts the type and later the specific name, e.g. `experimentalDataXY.m` for a driver file using experimental data. The driver file should only have one argument which is the filename containing the data. If you use a container format, you may want to write a routine or extend the existing `readDataFromFile.m` to your needs. The driver file needs to create all necessary fields covered above. If you need to run multiple numerical experiments and want to modify your `input_data`, you can do that still later on. Here is an example file:

```
function  input_data  = simulatedDataCellImage( filename )
%CELL256 Describes problem and creates data structure
%   This driver file is an example file to explain
%   how problems are created in the PRwSC toolbox.
%
%   Input:
%           filename - Name (string) of the file that contains the data.
%                      The input is assumed to be of the type .mat
%                      containing only one variable named 'pattern'.
%
%   Output:
%           input_data - A Matlab structure containing various
%                        information needed in the toolbox like
%                        the problem type, physical parameters,
%                        qualitative constraints and algorithmic
%                        parameters.
% For an explanation of the elements of the created structure,
% we refer to the documentation.
% Get preprocessed data. Right now, the method only returns the data
% itself.

input_data = readDataFromFile(filename);

% Since we don't now at this point which fields the readDataFromFile
% function provides us, we check for necessary information and provide
% some reasonable choices ourselves if they are not set yet.

if ~isfield(input_data, 'problem')
    input_data.problem.type    = 'phase';        % 'phase', 'amplitude', 'mixed'
end

if ~isfield(input_data, 'algorithm')
    input_data.algorithm.name       = 'RAAR';    % currently: RAAR, MAP
    input_data.algorithm.parameter  = 'dynamic'; % 'dynamic', 'constant'
    input_data.algorithm.beta.null  = 0.99;
    input_data.algorithm.beta.max   = 0.55;
    input_data.algorithm.beta.switch = 20;
    % input_data.algorithm.beta.value = 0.55;
end

if ~isfield(input_data, 'physicalParameters')
    if strcmp(input_data.problem.type, 'phase')
    % decrement of real part of defr. idx
        input_data.physicalParameters.delta = 1.6e-6;
    else
        input_data.physicalParameters.delta  = 0;
    end
    if strcmp(input_data.problem.type, 'amplitude')
    % imag. part of defr. idx
        input_data.physicalParameters.beta = 8e-7;
    else
        input_data.physicalParameters.beta   = 0.0;
    end
    % lateral thickness
    input_data.physicalParameters.tau    = 1e-5;
    % Wave length
    input_data.physicalParameters.lambda = 1e-10;
    % Wave number
    input_data.physicalParameters.k = 2*pi/input_data.physicalParameters.lambda;
    % Phase distribution
    input_data.physicalParameters.P = input_data.data * input_data.physicalParameters.tau;
    % Amplitude distribution
    input_data.physicalParameters.A = input_data.data * input_data.physicalParameters.tau;
    % Wave function Psi
    input_data.Psi    = exp(-1i.*input_data.physicalParameters.k....
                *input_data.physicalParameters.delta.*input_data.physicalParameters.P...
                -input_data.physicalParameters.k.*input_data.physicalParameters.beta....
                *input_data.physicalParameters.A);
end

if ~isfield(input_data.problem, 'noise')
    input_data.noise.type = 'poisson';  % poisson or gaussian
    input_data.noise.level = 50;        % poisson: expct. photons/pixel, gaussian: variance
    input_data.noise.error = 1e-2;      % describes the phattening of the set M
end
if ~isfield(input_data.problem, 'propagator')
    input_data.problem.propagator           = 'Fresnel';    % Fresnel or Fourier
    input_data.problem.Fx                   = 4*1e-3;       % Fresnel number x-coordinate
    input_data.problem.Fy                   = 4*1e-3;       % Fresnel number y-coordinate
    % Measurement data are pointwise modulus in fresnel/fourier domain
    input_data.problem.data            = abs(wavePropagator(input_data, input_data.Psi,1));
    input_data.problem.data            = createNoisyMeasurements(input_data);
    % shearlet, support, negativity, amplitude
    input_data.problem.constraint.name      = 'shearlet thresholding and range';
    % optional (depends on constraint name). soft / smooth hard
    input_data.problem.constraint.info      = 'smooth hard';
    input_data.problem.constraint.parameter = 'dynamic';   % dynamic = 1/k right now, const
    input_data.problem.constraint.gamma0    = 0.001;       % initial threshold parameter
    % for the cell and box, we would chose R=0.75.
    input_data.problem.constraint.support   = createSupportIdx(input_data, 0.8, 'box');
end
end
```

4 Examples

The toolbox contains one example file for simulated data. This example should be sufficient to write your own files and run your own reconstruction algorithms.

4.1 Simulated Data: Phase Object with Poisson Noise

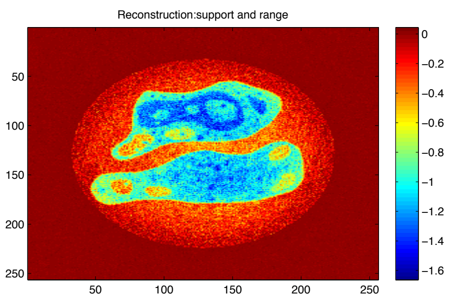
We consider an object $x \in \mathbb{C}^{n_1 \times n_2}$ of the form $x = \mathrm{e}^{-\mathrm{i}k\tau\delta T}$ where $T \in \mathbb{R}^{n_1 \times n_2}$. Objects of this type are called phase objects, τ describes the maximal thickness of the objects and T the relative lateral thickness profile. Furthermore, $k = 2\pi/\lambda$ is the wave number. This object is created using the driver file `simulatedDataCellImage.m`. The resulting configuration file is given in Section 3.3. Since we want to compare the reconstruction using different constraints, we use the `simulationComparison.m` file to run several instances with different constraints:

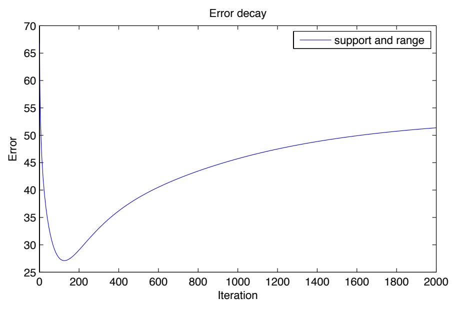
```
clear all
addpath('Algorithms', 'Drivers', 'InputData', 'OutputData',
                                'Processors', 'ProxOperators', 'Utilities')
addpath(genpath('../ShearLab3Dv11'))     % Shearlet Toolbox

input_data  = simulatedDataCellImage( 'cell256.mat' );

input_data.algorithm.maxit = 2000;
input_data.algorithm.error = 1e-10;
input_data.algorithm.metric = 'norm';

[sizex,sizey] = size(input_data.data);
useGPU        = 0;
scales        = 4; % number of scales
input_data.system        = SLgetShearletSystem2D(useGPU,sizex,sizey,scales);

if ~isfield(input_data, 'initialGuess')
        % initial guess is the simple back projection
    input_data.initialGuess = wavePropagator(input_data,input_data.problem.data);
end
```

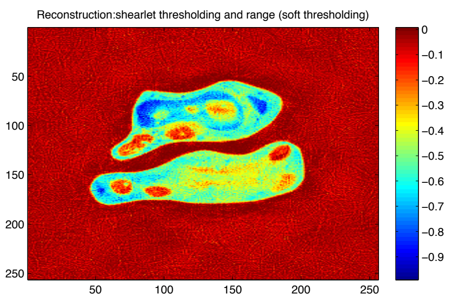
```
%% shearlet soft thresh + range
input_data.problem.constraint.name      = 'shearlet thresholding and range';
% optional (depends on constraint name). soft thresholding/ smooth hard
input_data.problem.constraint.info      = 'soft thresholding';
% dynamic = 1/k. right now, const
input_data.problem.constraint.parameter = 'const';
% initial threshold parameter
input_data.problem.constraint.gamma0    = 0.005;
% for the cell and box, we would chose R=0.75.
input_data.problem.constraint.support   = createSupportIdx(input_data, 0.75, 'ball');

% object that holds the reconstruction and errors
recDataShearletRangeSoft = RAAR(input_data);

%% shearlet smooth hard
input_data.problem.constraint.name      = 'shearlet thresholding';
% optional (depends on constraint name). soft thresholding/ smooth hard
input_data.problem.constraint.info      = 'smooth hard';
% dynamic = 1/k right now, const
input_data.problem.constraint.parameter = 'const';
% initial threshold parameter
input_data.problem.constraint.gamma0    = 0.007;
% for the cell and box, we would chose R=0.75.
input_data.problem.constraint.support   = createSupportIdx(input_data, 0.75, 'ball');

recDataShearletSmoothHard = RAAR(input_data);

%% support and range
input_data.problem.constraint.name      = 'support and range';
if isfield(input_data.problem.constraint, 'info')
    input_data.problem.constraint = rmfield(input_data.problem.constraint, 'info');
end
% for the cell and box, we would chose R=0.75.
input_data.problem.constraint.support   = createSupportIdx(input_data, 0.75, 'ball');

recDataSupportRange = RAAR(input_data);
```

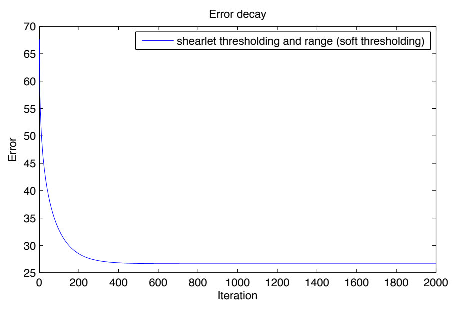
Running this file yields three reconstruction, one where we used the combination of shearlet soft-thresholding and a range constraint (the first one), one where we only used shearlet thresholding but the so-called smooth-hard shrinkage (which is a type of thresholding, see [2]) and the last one were we use the combination of the support and range constraint. The last one is basically the benchmark we compare to since support information combined with a range constraint is a strong a priori information.

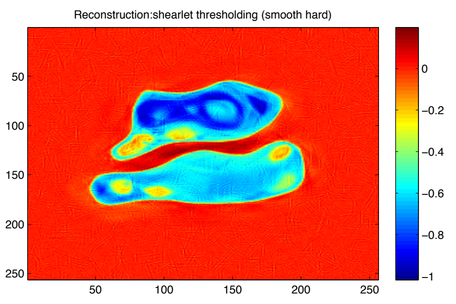
We obtain three output structures which have the structure as described above. We can now use the `plotReconstructionResults.m` file to plot the reconstructions and errors or use other methods to examine and compare the reconstruction quality.


Reconstruction:support and range
50
100
150
200
250
0
−0.2
−0.4
−0.6
−0.8
−1
−1.2
−1.4
−1.6

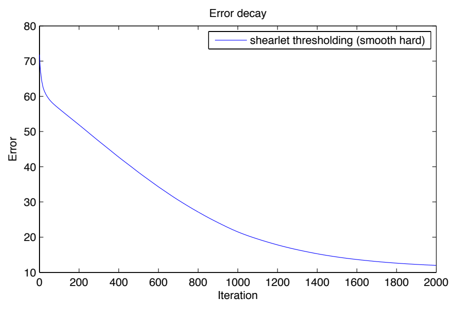

Error decay
support and range
Error
Iteration
0 200 400 600 800 1000 1200 1400 1600 1800 2000
25 30 35 40 45 50 55 60 65 70


Reconstruction:shearlet thresholding and range (soft thresholding)
50
100
150
200
250
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9


Error decay
shearlet thresholding and range (soft thresholding)
Error
Iteration
0 200 400 600 800 1000 1200 1400 1600 1800 2000
25 30 35 40 45 50 55 60 65 70


Reconstruction:shearlet thresholding (smooth hard)
50
100
150
200
250
0
−0.2
−0.4
−0.6
−0.8
−1

e


Error decay
shearlet thresholding (smooth hard)
Error
Iteration
0 200 400 600 800 1000 1200 1400 1600 1800 2000
10 20 30 40 50 60 70 80

References

[1] H. H. Bauschke, P. L. Combettes, and D. R. Luke. Phase retrieval, error reduction algorithm, and Fienup variants: A view from convex optimization. *Journal of the Optical Society of America A*, 19(7):1334–1345, 2002.

[2] R. Chartrand. Shrinkage mappings and their induced penalty functions. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1026–1029. IEEE, 2014.

[3] K. Giewekemeyer. *A study on new approaches in coherent x-ray microscopy of biological specimens*. PhD thesis, Georg-August-Universität Göttingen, 2011.

[4] K. Giewekemeyer, S. P. Krüger, S. Kalbfleisch, M. Bartels, C. Beta, and T. Salditt. X-ray propagation microscopy of biological cells using waveguides as a quasipoint source. *Physical Review A*, 83(2):023804, 2011.

[5] M. Krenkel. *Cone-beam x-ray phase-contrast tomography for the observation of single cells in whole organs*. PhD thesis, Georg-August-Universität Göttingen, 2015.

[6] G. Kutyniok, W.-Q Lim, and R. Reisenhofer. ShearLab 3D: Faithful digital shearlet transforms based on compactly supported shearlets. *ACM Transactions on Mathematical Software*, 42(1):5, 2016.

[7] S. Loock. *Phase Retrieval with Sparsity Constraints*. PhD thesis, Georg-August-Universität Göttingen, 2016.

[8] A. Pein, S. Loock, G. Plonka, and T. Salditt. Using sparsity information for iterative phase retrieval in x-ray propagation imaging. *Optics Express*, 24(8):8332–8343, 2016.