Comparison of ADMM and Kaczmarz Reconstruction Algorithms

Aslı Alpman, Beril Alyüz, Berfin Kavşut

Video Link:
https://drive.google.com/file/d/1CUSOFVX_KoYtx1o3jj-2oVP02PPEwFQy/view?usp=sharing

1. Abstract

System Function Reconstruction for MPI, Magnetic Particle Imaging, leads to an inverse problem which is mostly ill-conditioned. In this project, we implemented Kaczmarz and ADMM, Alternating Direction Method of Multipliers, algorithms to solve this inverse problem. With ADMM, we also employed total variation and l_1 norm regularization to utilize slowly changing nature and the sparsity of the images used in the project. We compared the performance of Kaczmarz and ADMM by looking at the quality of the resulted images and convergence properties.

2. Introduction

MPI images are typically sparse by nature, e.g.angiographic images, therefore $l1$ norm regularization techniques can be used for MPI image reconstructions. Alternating Direction Method of Multipliers (ADMM) is one of iterative solvers can be used for least-squares problem with penalty term of $l1$ norm of solution, total variation, and sparsity constraint.Likewise, Kaczmarz is an iterative solver for weighted, regularized least-squares problem using $l2$ norm regularization. In addition to least-squares problem, $l2$ norm is penalized with regularization parameter of $l2$ norm of solution. ADMM is an iterative solver using $l1$ norm regularization.

Kaczmarz and ADMM reconstructions are compared with small-sized phantom and two real MPI datasets. The first one is a simulated phantom provided to us in our homework assignment shapes as λ. The second and thirds ones, which are concentration phantom and reconstruction phantom, are obtained from the website Open MPI Data [1]. In Methods part, alternative Kaczmarz method for least-squares problem is explained. In ADMM part, theoretical parts for LASSO problem, TV problem, TV L1-L2 problem and non negativity constraint. The results are compared according to PSNR, SSIM values as IQA measurements, and convergence speed with regard to the change in NMSE error and objective functions, and computation times for constant iteration numbers.

3. Methods

3.1 Dataset

The datasets are taken from the website called Open MPI Data. MPI data was stored as MDF (MPI Data Format). Dataset of concentration phantom is received from the provided reconstruction code. The reconstruction code results in ART implementation for the slice of concentration phantom. We have taken our system matrix and measurement vector of concentration phantom from this reconstruction code. System matrix and measurement vectors are preprocessed, and preprocessing code is in Appendix. Receive coils in x and y directions are used for dataset. For each coil, real and imaginary parts are taken separately and concatenated at the end for convenience in our algorithms since working with real-valued matrices is easier for our reconstruction algorithms.

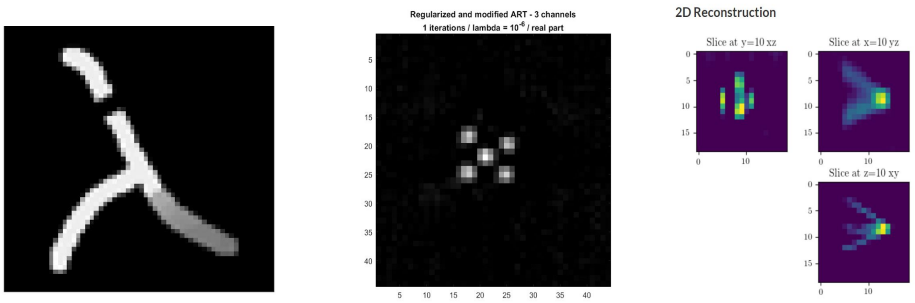

Figure 1: λ. Phantom, Concentration Phantom [1], and Resolution Phantom [2]

The dataset for resolution phantom is acquired from provided datasets in the same website. Similarly, preprocessing code is partially taken from reconstruction code provided in the website. Calibration is obtained with 2D Lissajous Sequence. The grid size of measurement is 19x19x19. The measurements are received slice-by-slice with step-wise movement in z-direction. There are 19 patches each having 2D positions. [3].

3.2 Kaczmarz Method

Kaczmarz inherently implements row normalization in the system matrix and measurement vector. When the system is inconsistent with high condition number, traditional Kaczmarz algorithm becomes unsuccessful for least-squares problem. As a remedy of this problem, there is an alternative Kaczmarz method for least-squares problem, which consists of weighting system matrix and measurement vector by row normalization with respect to row energies. This is known that condition number decreases when there is an ill-conditioned system in weighted least-squares problem. [4]

The weighted least squares problem is as follows:

$$\left\| W^{\frac{1}{2}}(Sc-u) \right\|_2^2 + \lambda \|c\|_2^2 \quad where\ \lambda > 0$$

Weighted Kaczmarz method is altered in this form:

$$\begin{bmatrix} W^{\frac{1}{2}} S & \lambda^{\frac{1}{2}} I \end{bmatrix} \begin{bmatrix} c \\ v \end{bmatrix} = W^{\frac{1}{2}} u \quad where\ v = -\lambda^{-\frac{1}{2}} W^{\frac{1}{2}} (Sc-u)$$

Algorithm (2)

$$1-)\ \alpha_k = \frac{u_i - \langle s_i^H, c^k \rangle - \left(\frac{\lambda}{w_i}\right)^{\frac{1}{2}} v_i^k}{\|s_i\|_2^2 + \frac{\lambda}{w_i}}$$

$$2-)\ c^{k+1} = c^k + \alpha_k s_i^H$$

$$3-)\ v^{k+1} = v^k + \alpha_k \left(\frac{\lambda}{w_i}\right)^{\frac{1}{2}} e_i$$

e_i : *unit vector along* i^{th} *dimension*

λ = *regularization parameter*

3.3 ADMM

3.3.1 LASSO problem

We reconstructed x as a solution of the following problem, known as LASSO -least absolute shrinkage and selection operator-

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \tau . \| x \|_1 \right] \quad (1)$$

(1) can be reformulated as a constrained optimization problem by splitting x into two variables which are x and w.

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \tau . \| w \|_1 \right]$$

$$subject\ to\ w = x$$

Constraint can be added up to the objective by using augmented Lagrangian method.

$$L(x,z,y) = \frac{\mu}{2} . \| Ax - b \|^2 + \tau . \| w \|_1 + y^T (x - w) + \frac{\rho}{2} . \| x - w \|^2$$

Algorithm (2)

While (NOT (primal residual < primal tolerance) OR NOT (dual residual < dual tolerance))

1- $x^{k+1} = argmin_x L(x, z^k, y^k)$
2- $z^{k+1} = argmin_z L(x^{k+1}, z, y^k)$
3- $y^{k+1} = argmin_y L(x^{k+1}, z^{k+1}, y)$

End

Primal residuals and dual residuals are calculated according to the formula in [5]. Primal residual can be interpreted as how far we are from satisfying the constraint. Further, dual residual can be interpreted as the how much dual variable changes in one iteration.

It is possible to write solution of the above steps in closed forms. $L(x,z,y)$ is convex and differentiable in x and thus closed form can be obtained by simply differentiating $L(x,z,y)$ with respect to x. For the second step, the closed form expression utilizes soft threshold function [5]. Lastly, $L(x,z,y)$ is also differentiable in y and the closed form expression for y-update is also obtained by differentiation of $L(x,z,y)$.

3.3.2 TV Problem

We reconstructed x as a solution of the following problem

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \lambda . TV(x) \right] \quad (2)$$

Total variation is defined as follows

$$TV(x) = \sum_{k,j} \sqrt{(x_{k+1,j} - x_{i,j})^2 + (x_{k,j+1} - x_{i,j})^2}$$

where $k = 1, \ldots, n$ is the row index and $j = 1, \ldots, m$ is the column index. Total variation of vectorized image x can also be written with the help of the finite difference operator. Multiplication of the image

with D_i returns vector with 2 elements which are horizontal and vertical difference for i^{th} element of the image.

$$TV(x) = \sum_i \| D_i x \|$$

(2) can be reformulated as a constrained optimization problem by splitting x into two variables which are x and z.

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \lambda . \sum_i \| z_i \| \right]$$

$$subject\ to\ z_i = D_i x$$

We can add constraint to the objective by using augmented Lagrangian method.

$$L(x,z,y) = \frac{\mu}{2} . \| Ax - b \|^2 + \lambda . \sum_i \| z_i \| + y^T (Dx - z) + \frac{\rho}{2} . \| Dx - z \|^2$$

where y is the dual variable and $D = \left[D_{hor}^T , D_{vert}^T \right]^T$. D_{hor} operator calculates horizontal finite and D_{vert} is the operator calculates vertical finite difference for difference for vectorized image multiplied with.

Algorithm is similar to the algorithm 2. The main difference is that two dimensional soft-thresholding is used during update of z vector. Details of the implementation follows the similar pattern with the reference [6].

3.3.3 TVL1-L2 Problem

We reconstructed x as a solution of the following problem, known as TVL1-L2

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \lambda . TV(x) + \tau . \| x \|_1 \right] \quad (3)$$

Total variation is defined as in the TV problem. (3) can be reformulated as a constrained optimization problem x into three variables which are x, z and w.

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \lambda . \sum_i \| z_i \| + \tau . \| w \|_1 \right]$$

$$subject\ to\ z_i = D_i x\ and\ w = x$$

Constraints can be added up to the objective by using augmented Lagrangian method.

$$L(x,w,z,y_1,y_2) = \frac{\mu}{2} . \| Ax - b \|^2 + \tau . \| w \|_1 + y_1^T (x - w) + \frac{\rho}{2} . \| x - w \|^2$$

$$+ \lambda . \sum_i \| z_i \| + y_2^T (Dx - z) + \frac{\rho}{2} . \| Dx - z \|^2 \text{ where } D = \left[D_{hor}^T , D_{vert}^T \right]^T$$

The main difference of this problem is that it uses two auxiliary and dual variables for TV and l_1 norm constraints. Luckily, we already have closed form expressions for both constraints. The integration of those two is done by following the reference paper [7]. The sketch of the algorithm is as follows. Primal and dual residuals are again calculated according to the formula in [5].

We also solved the problem (3) with an alternative method without using any inversion in the algorithm. The implementation of the alternative method is completed by referencing the paper [8]. Paper introduces a proximal term G in order to avoid matrix inversion during the x update [8]. G is defined as:

$$G = \frac{1}{\delta} I - \rho I - \rho D^T D - \mu A^T A$$

The primal variable x is obtained by solving the following problem:

$$x^{k+1} = \arg\min_x L_p(x, w^k, z^k, {y_1}^k, {y_2}^k) + \frac{1}{2}\left\| x - x^k \right\|_G^2$$

Paper provides solutions for this problem. In order to avoid using inverse, the paper also introduces the minimization of the w as:

$$w^{k+1} = \arg\min_w \tau \left\| w \right\|_1 + \frac{\rho}{2}\left\| w - x^{k+1} + \frac{{y_1}^k}{\rho} \right\|_2^2$$

Variable w is updated with a one dimensional shrinkage operator as it was done in the previous implementation. Similarly, variable z is found by solving the following problem:

$$z^{k+1} = \arg\min_z \lambda \left\| z \right\|_1 + \frac{\rho}{2}\left\| z - D \cdot x^{k+1} + \frac{{y_2}^k}{\rho} \right\|_2^2$$

Variable z is updated by using multidimensional shrinkage operator as it was done in the previous implementation. Dual variables y_1 and y_2 are updated in the same manner as it was done in the previous implementation. Paper introduces a new variable δ which should be chosen in the interval:

$$\delta \in (0, \frac{1}{\rho + \rho\left\| D \right\|^2 + \mu\left\| A \right\|^2}]$$

In our implementation, δ is chosen as the upper boundary value.

3.3.4 Non-negativity Constraint

Since we are reconstructing MPI images, the values we obtained are the particle concentrations. In the previous algorithms, we forced the reconstructed image to be nonnegative by simply assigning zeros to the negative values at the resulted image. However, we can also add up this constraint into the objective function that we are trying to minimize with the following indicator function.

$$i_c(x) = \{0 \qquad x \in R_+ \qquad \infty \qquad otherwise$$

The objective function for LASSO becomes,

$$\left[\tfrac{\mu}{2} . \left\| Ax - b \right\|^2 + \tau . \left\| x \right\|_1 + i_c(x) \right]$$

The objective function for TV regularized problem becomes,

$$\left[\frac{\mu}{2} . \| Ax - b \|^2 + \lambda . TV(x) + i_c(x) \right]$$

We applied this constraint to the LASSO and TV problem by splitting another variable w. TVL1-L2 implementations use variable s for the non-negativity constraint. The update of this term yields non negative projection [9]. In each iteration w is updated by assigning zeros to the negative values of the $\left[w + \frac{1}{\rho} y_3 \right]$ where y_3 is the dual variable of w and ρ is the penalty parameter in the Lagrangian, Likewise, s is updated by assigning zeros to the negative values of the $\left[s + \frac{1}{\rho} y_3 \right]$ where y_3 is the dual variable of s.

3.4. Evaluation Criteria

For the λ phantom, there was a reference image to compare with our reconstructions, therefore PSNR and SSIM value were prefered as quantitative assessments. Separate visual results are also displayed in Results part. NMRS error and objective functions are used for the assessment of convergence speed of iterative solvers. Objective functions are minimized values inside iterative solver. NMRS error is defined as follows:

$$\text{nrmse} = \sqrt{immse(x, ref)/(max(x) - min(x))} \quad [4]$$

4. Results

4.1. Reconstruction Results for λ. Phantom

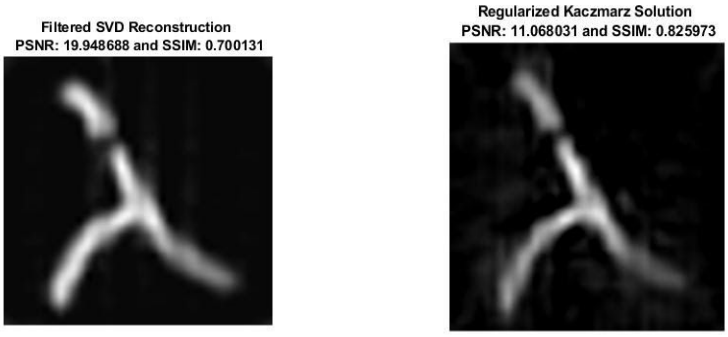

Figure 2: Filtered SVD and Regularized Kaczmarz Reconstruction for $\boldsymbol{\lambda}$ = 6400

The reconstruction with Filtered SVD is applied to system matrix and measurement vector without row norm thresholding and row normalization. However, system matrix and measurement vector is thresholded according to a certain threshold that is chosen to be 30 in order to avoid noise amplification problems. Regularized Kaczmarz method works for 100 iterations and it takes 8.171875 seconds to complete 100 iterations. The algorithm converges around 2-3. iteration, we have checked objective values and NRMSE values for each iteration and the values stabilizes after 3.iteration. The PSNR and SSIM values for Filtered SVD and Regularized Kaczmarz is given in the respective figures' titles.

For all ADMM methods (lasso, total variation and combinations), the parameters are chosen to be same. Step length is 1, μ is 1, ρ which is the penalty parameter is 5, $\boldsymbol{\tau}$ that controls the l_1 norm and sparsity of the solution is 0.1 and $\boldsymbol{\lambda}$ which controls the total variation parameter is also 0.1. Maximum iteration for each method is chosen as 50 such that the convergence of the methods can be observed in NRMS and Objective Function Values.

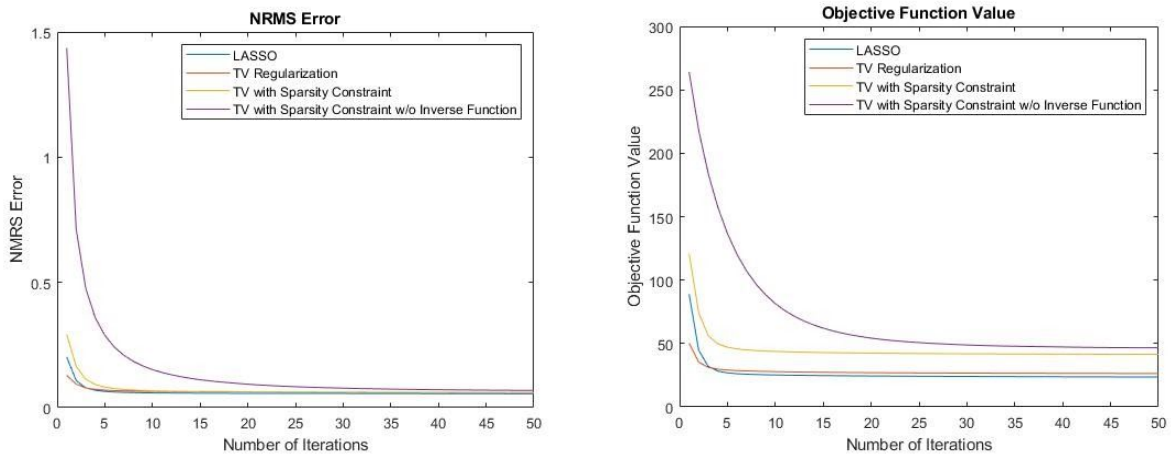

Figure 3 : NRMS Error and Objective Function Value for ADMM Reconstruction with Optimal Values

The reconstruction for each ADMM method and their PSNR and SSIM values is as follows:

Figure 4: ADMM Reconstruction for Optimal Parameters (left) and ADMM Reconstruction with Nonnegativity Constraint for Optimal Parameters (right)

We have also included non negativity as a dual variable to each method to enforce non negativity criteria within each method, rather than enforcing it after the reconstruction (setting all negative values to zero after reconstruction). CPU Time for each method is given in the Table 1 in the Appendix. We have used the same parameters that is used in previous reconstructions. This produces the following NRMSE and Objective Function plots.

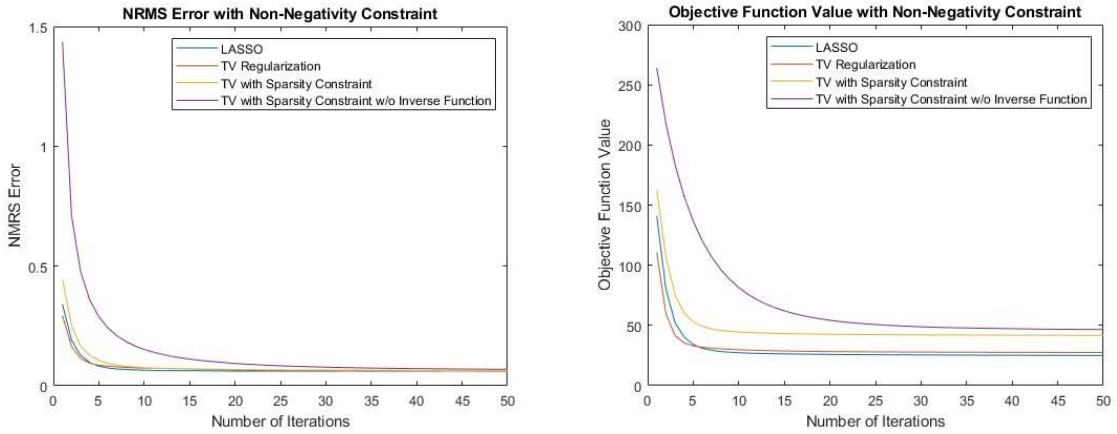

Figure 5: NRMS Error and Objective Function Value for ADMM Reconstruction for Optimal Values and with Nonnegativity Constraint

4.2. Reconstruction Results for Concentration Phantom

Firstly, we implemented Kaczmarz algorithm and ADMM algorithm for unweighted systems. It was observed that result images were noisy. Then, row normalization was applied and noise was eliminated. For the rest of Kaczmarz and ADMM results, we applied row norm thresholding with respect to row energies and row normalization for concentration phantom. Row normalization also worked better for ADMM results due to decrease in condition number. The improvement can be seen in Figure 6.

Concentration phantom image is 44x44 2D image. For all reconstructions of concentration phantom, the rows having lower *l*2-norm than 50 are thresholded. Before row normalization, condition number was 2795614, and condition number became 7347.1 after row normalization. Optimum lambda acquired by L-curve is 11000 and used for filtered SVD and weighted regularized Kaczmarz solutions.

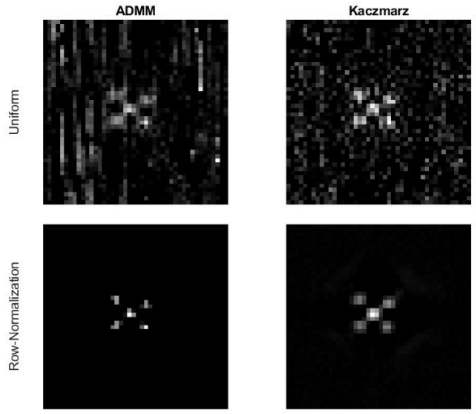

Figure 6: Reconstructions of ADMM and Kaczmarz with Uniform Weight and Row Normalized Weight

To analyze how total variation and l1 regularization affect the image, we implemented TVL1-L2 algorithm for different l1 and total variation regularization parameter combinations. Here, step length, and are given 1 and kept constant. Absolute tolerance, which is linked with stopping criteria, is selected as for TV and TVL1-L2 implementation. For LASSO and alternative TVL1-L2, it is . Relative tolerance is given

as for all methods. Stopping criteria are linear combination of absolute tolerance and relative tolerance where the multiplier of relative tolerance changes with the state of the current iteration.

Figure 7 shows the results of the TVL1-L2 for the given and λ values. Figure 8 shows the results of the alternative TVL1-L2 for the given and λ values.

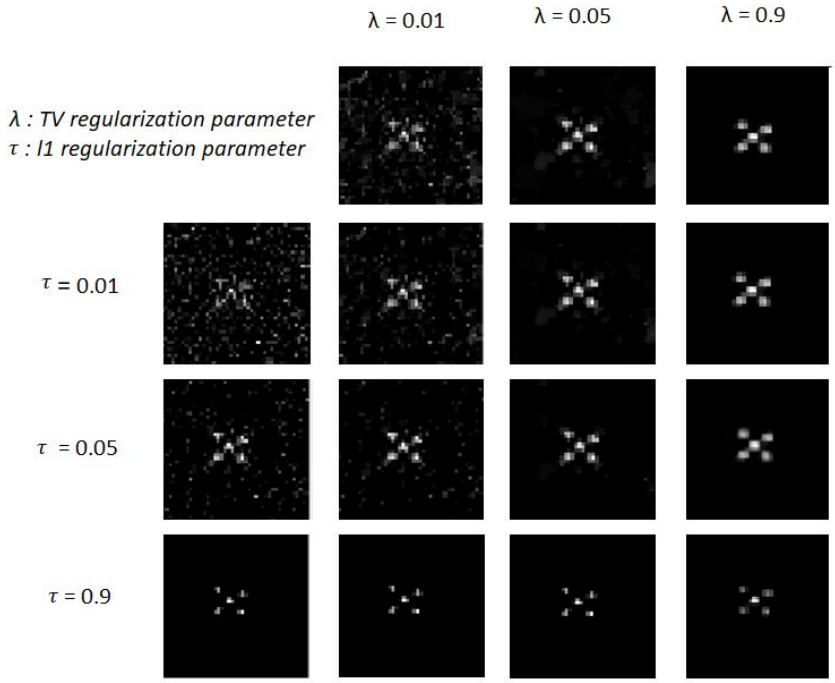

Figure 7: Combination of different TV and l1 norm regularization parameters for TVL1-L2 implementation

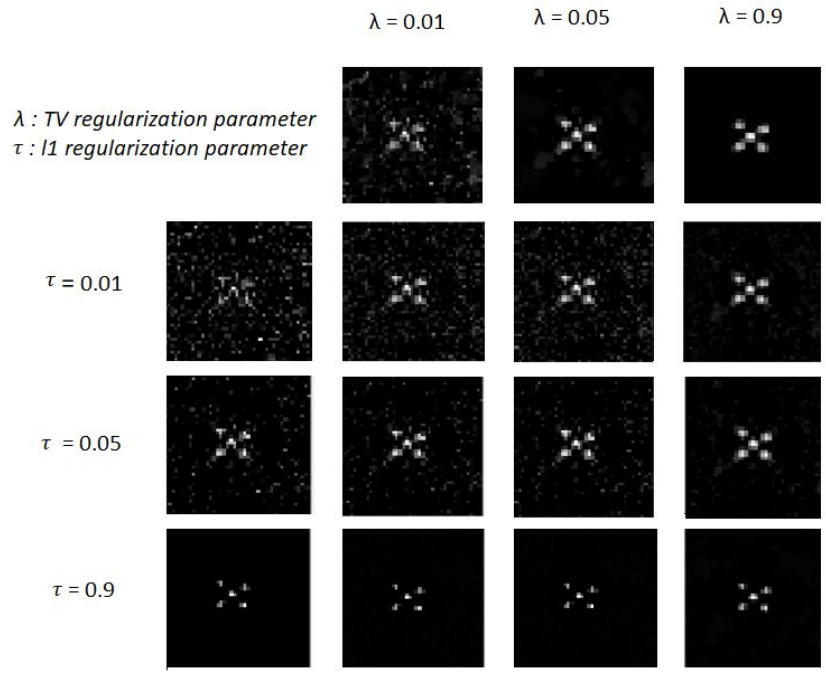

Figure 8: Combination of different TV and l1 norm regularization parameters for alternative TVL1-L2 implementation

4.3. Reconstruction Results for Resolution Phantom

We have also reconstructed images using more realistic dataset [3]. We have taken a measurement of a volume that is sampled with 2D Lissajous trajectory for each slice and we tried to reconstruct each slice with our ADMM methods. Step length is 1, μ is 1, ρ which is the penalty parameter is 5, $\boldsymbol{\tau}$ that controls the l1 norm and sparsity of the solution is 0.5 and $\boldsymbol{\lambda}$ which controls the total variation parameter is also 0.5. We have chosen maximum iteration as 10 since the realistic matrices have large sizes and it takes long time to perform one iteration. We have applied row norm thresholding to the system matrix and the measurement vector for the realistic dataset. Following two figure shows the reconstructions:

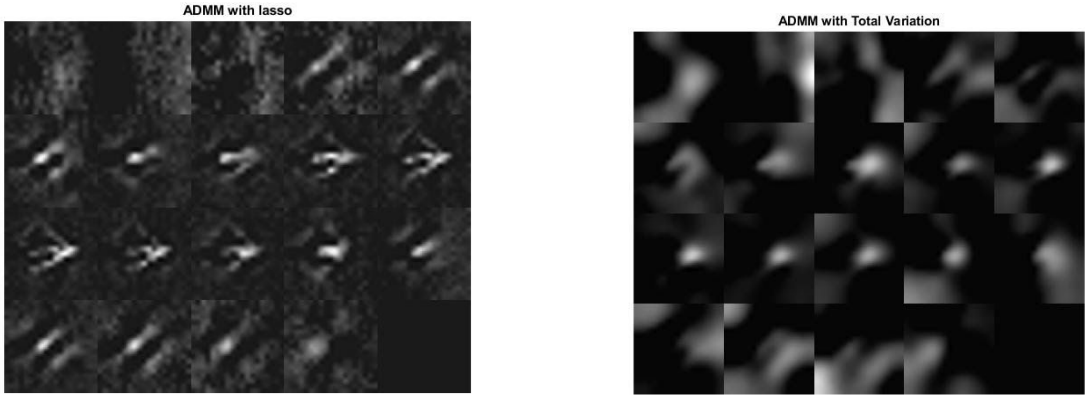

Figure 9: ADMM Reconstructions with Realistic Dataset (Resolution Phantom)

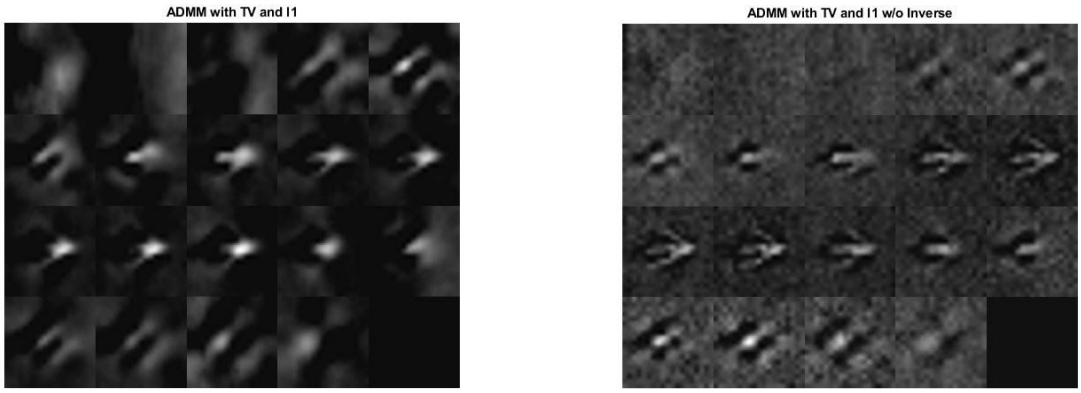

Figure 10: ADMM Reconstructions with Realistic Dataset (Resolution Phantom) Cont'd

4. Discussion

We have used the dataset that is given in Homework 3 to evaluate the performance of the ADMM reconstruction techniques in terms of SSIM and PSNR since other datasets that we have used do not have reference images. We have used row normalized versions of the system matrices for ADMM due to the improvement in the condition number and better results obtained as in the Figure 6.

According to PSNR and SSIM values in Figure 4, the best reconstruction is ADMM with TV, then TVL1-L2 combined, alternative method for TVL1-L2 combined and LASSO. However, all values of PSNR and SSIM are in their acceptable ranges and very close to one another. In addition, convergence for the alternative method of TVL1-L2 takes longer. Alternative method for ADMM with TVL1-L2 have

seems to converge in latter iterations compared to other ADMM methods among methods without non-negativity constraint, this can be seen from Figure 3. ADMM with l_1 , TV and TVL1-L2 seems to converge around 5th iteration whereas the alternative method for TV-l_1 seems to converge around 15th iteration.

For the methods with non-negativity constraint,there are similar patterns in convergence. Alternative method for TVL1-L2 converges later than other methods and objective function for methods non-negativity constraint is very similar to objective function plot of the methods without non-negativity constraint. These indicate that enforcing non-negativity as an additional constraint gives similar results to correcting negativity after reconstruction. ADMM reconstructions with non-negativity constraint have similar PSNR and SSIM values to ADMM reconstructions without the constraint. Therefore, we did not employ non negativity constraint for our next reconstructions.

To analyze how total variation and l_1 regularization affect the image, we implemented TVL1-L2 algorithm for different l_1 and total variation regularization parameter combinations. In figure 7 and 8, values increases from top to the bottom. This means that the price we will pay for the l_1 norm of the reconstructed image increases. Since our image is sparse in the image domain, we improved the reconstructed image by including this information in our objective with l_1 norm regularization in the image domain. The leftist column which consists of three images have the results of LASSO problems with the shown parameters. As gets bigger, only a few points remain and others are suppressed. Suppressed points can correspond to the noise or particle. The sweet is the one which suppresses noise and keeps the values coming from the particles. Luckily, it seems like the data have high SNR and thus l_1 regularization leads successful noise suppression.

Further, λ values increases from left to the right which means that the penalty for the rapid changes in the image increases. Since it is likely for noise to lead rapid changes in the image, total variation regularization also suppresses the noise. In the uppermost row which consists of three images, we observe that the background noise is suppressed however, the pixels which have high values are smeared around. If we look at the combination of the TV and l_1 norm in figure 7 and 8, we observe that the combined image quality is mostly determined by the "sweetest" parameter. Even though one of the parameters leads poor reconstruction by its own, other regularization can greatly improve the reconstructed image. To illustrate, the image which corresponds to and in figure 7 and 8, clearly shows that the better parameter compensates for the worse parameter. It may be due to that both regularization have denoising effect and the noisy background which can not be suppressed by one of them is compensated by the other.

Lastly, we also test our algorithms with the resolution phantom that is taken from the dataset [3]. Lasso and alternative method for TVL1-L2 seems to reconstruct somewhat meaningful images, with identifiable features of the resolution phantom (such as three lines). However, TV and original TVL1-L2 method seem to suffer in reconstruction for this dataset. Reconstruction with TV doesn't have identifiable features from the phantom, original TVL1-L2 method amplify certain parts of the resolution phantom while suppressing the lines. This may be due to that the finite difference is looked slice by slice. There could be an improvement if the variations are taken in more directions rather than two (right and down). Also, the reason that alternative TVL1-L2 method performed better in this dataset than the original TVL1-L2 method could be the inversion also alternative TVL1-L2 method sometimes updates the dual variable for TV with increments smaller than the threshold forcing the algorithm to converge prematurely. This could be the reason the effects of l_1 reconstruction seems to be seen more in alternative method for TVL1-L2 .

In addition, since it is an actual measurement from a realistic phantom, we could expect some noise in the measurements which may corrupt the reconstruction from these measurements.

5. References

[1] MagneticParticleImaging, “MagneticParticleImaging/MDF,” GitHub. [Online]. Available: https://github.com/MagneticParticleImaging/MDF/tree/master/matlab. [Accessed: 14-Jun-2020].

[2] “Open MPI Data,” Reconstruction · Open MPI Data. [Online]. Available: https://magneticparticleimaging.github.io/OpenMPIData.jl/latest/reconstructions.html. [Accessed: 14-Jun-2020].

[3] T. Knopp, P. Szwargulski, F. Griese, and M. Gräser, “OpenMPIData: An initiative for freely accessible magnetic particle imaging data,” Data in Brief, vol. 28, p. 104971, 2020.

[4] Knopp, T., Rahmer, J., Sattel, T.F., Biederer, S., Weizenecker, J., Gleich, B., Borgert, J., Buzug, T.M.: Weighted iterative reconstruction for magnetic particle imaging. Phys. Med. Biol. 55(8), 1577–1589 (2010).

[5] S. Boyd, “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers,” Foundations and Trends® in Machine Learning, vol. 3, no. 1, pp. 1–122, 2010.

[6] Tao, Min, J. Yang. “Alternating Direction Algorithms for Total Variation, Deconvolution in Image Reconstruction,” http://people.fas.harvard.edu/~cs278/papers/adm.pdf. [Accessed: 14- Jun- 2020].

[7] J. Yang, Y. Zhang and W. Yin, "A Fast Alternating Direction Method for TVL1-L2 Signal Reconstruction From Partial Fourier Data," in IEEE Journal of Selected Topics in Signal Processing, vol. 4, no. 2, pp. 288-297, April 2010, doi: 10.1109/JSTSP.2010.2042333.

[8] R. Gao, F. Tronarp and S. Särkkä, "Combined Analysis-L1 and Total Variation ADMM with Applications to MEG Brain Imaging and Signal Reconstruction," 2018 26th European Signal Processing Conference (EUSIPCO), Rome, 2018, pp. 1930-1934, doi: 10.23919/EUSIPCO.2018.8553122.

[9] *Angms.science*, 2020. [Online]. Available: https://angms.science/doc/NMF/nnls_admm.pdf. [Accessed: 14- Jun- 2020].

[10] MATLAB scripts for alternating direction method of multipliers. [Online]. Available: https://web.stanford.edu/~boyd/papers/admm/. [Accessed: 14-Jun-2020].

6. Appendix

1. TABLES

	CPU Time	CPU Time (Non-negativity constraint)
Lasso	3.9218750	4.8125
TV	4.0937500	4.765625
TV-L1	5.2187500	5.375
TV-L1 w/o inverse	10.265625	10.375

Table 1: CPU Times of Each ADMM Reconstruction Method for 50 Iteration

2. CODES

Algorithm of Kaczmarz function is implemented according to [4] and ADMM function for Lasso problem is partially implemented according to [10]. ADMM with TV regularization, ADMM for TVL1-L2 problem and alternative ADMM for for TVL1-L2 is implemented according to the following papers respectively [6,7,8].

Preprocessing Data Code for Concentration Phantom

```
%% 1. Loading the required external functions

clear all
close all

%% 2. Download measurement and systemMatrix from http://media.tuhh.de/ibi/mdf/

filenameSM = 'systemMatrix.mdf';
filenameMeas = 'measurement.mdf';

websave(filenameSM,'http://media.tuhh.de/ibi/mdfv2/systemMatrix_V2.mdf')
websave(filenameMeas,'http://media.tuhh.de/ibi/mdfv2/measurement_V2.mdf')

%% 3. Loading the data
% For the System matrix (later named SM)
% to obtain infos on the file, use the command: infoSM = h5info(filename_SM);
% or read the format documentation

% read the data, saved as real numbers
S = h5read(filenameSM, '/measurement/data');

% reinterpret as complex numbers
S = complex(S.r,S.i);
% get rid of background frames
isBG = h5read(filenameSM, '/measurement/isBackgroundFrame');
S = S(isBG == 0,:,:,:);

% For the measurements
% read and convert the data as complex numbers
% note that these data contain 500 measurements
u = h5read(filenameMeas, '/measurement/data');
%u = squeeze(u(1,:,:,:) + 1i*u(2,:,:,:));
u = fft(cast(u,'double'));
u = u(1:(size(u,1)/2+1),:,:,:);

%% 4. Pre-process - Remove the frequencies which are lower than 30 kHz, as they are unreliable due to the
anologue filter in the scanner

% generate frequency vector
numFreq = h5read(filenameMeas, '/acquisition/receiver/numSamplingPoints')/2+1;
rxBandwidth = h5read(filenameMeas, '/acquisition/receiver/bandwidth');
freq = linspace(0,1,numFreq) .* rxBandwidth;

% we supose that the same frequencies are measured on all channel for
% the SM and the measurements. use only x/y receive channels
```

```
idxFreq = freq > 80e3;
S_truncated = S(:,idxFreq,1:2);
u_truncated = u(idxFreq,1:2,:);

% take calibration measurement from each coil
% take transpose of them to have frequency components
% in columns, corresponding to each grid position
SystemMatrix.S1 = transpose(S_truncated(:,:,1)); %764x6859
SystemMatrix.S2 = transpose(S_truncated(:,:,2)); %764x6859

% prepare the system matrix
% take real and imaginary parts of complex system matrices
% concetanate the system matrices from two receive coils
S1 = [real(SystemMatrix.S1);imag(SystemMatrix.S1)]; % 1528x6859
S2 = [real(SystemMatrix.S2);imag(SystemMatrix.S2)]; % 1528x6859
S = [S1;S2]; % 3056x6859

% take measurement data of resolution phantom from each coil
u1 = u_truncated(:,1,:); u1 = reshape(u1,size(u1,1),size(u1,3)); %764x38000
u2 = u_truncated(:,2,:); u2 = reshape(u2,size(u2,1),size(u2,3)); %764x38000

% take average of measurements
measurement.u1 = mean(u1,2); %764x1
measurement.u2 = mean(u2,2); %764x1

% prepare the measurement vector
% take real and imaginary parts of complex system matrices
% concetanate the measurements from two receive coils
u1_real = [real(measurement.u1);imag(measurement.u1)]; %1528x1
u2_real = [real(measurement.u2);imag(measurement.u2)]; %1528x1
u = [u1_real;u2_real]; %3056x1

save('SM_website.mat','S');
save('meas_website.mat','u');

% %% 5. Merge frequency and receive channel dimensions
% S_truncated = reshape(S_truncated, size(S_truncated,1), size(S_truncated,2)*size(S_truncated,3));
% u_truncated = reshape(u_truncated, size(u_truncated,1)*size(u_truncated,2), size(u_truncated,3));
%
% %% 6. Averaged the measurement used for the reconstruction over all temporal frames
% u_mean_truncated = mean(u_truncated,2);
%
% %% 7. Make two simple reconstructions
% % a normalized regularized kaczmarz approach
% c_normReguArt = kaczmarzReg(S_truncated(:,:),...
%                     u_mean_truncated(:),...
%                     1,1*10^-6,0,1,1);
%
% % and an regularized pseudoinverse approach
% [U,Sigma,V] = svd(S_truncated(:,:).','econ');
% Sigma2 = diag(Sigma);
% c_pseudoInverse = pseudoinverse(U,Sigma2,V,u_mean_truncated,5*10^2,1,1);
%
```

```
% %% 8. Display an image
% % read the original size of an image
% number_Position = h5read(filenameSM, '/calibration/size');
%
% figure
% subplot(1,2,1)
% imagesc(real(reshape(c_normReguArt(:),number_Position(1),number_Position(2))));
% colormap(gray); axis square
% title({'Regularized and modified ART - 3 channels';'1 iterations / lambda = 10^{-6} / real part'})
```

Preprocessing Data Code for Resolution Phantom

```
%% Loading the required external functions

clear all
close all

%% System Matrix

% load system matrix
filenameSM = 'calibration2.mdf';
filenameMeas = 'resolutionPhantom2.mdf';

number_Position = h5read(filenameSM, '/calibration/size'); %[19;19;19]

% read the data, saved as real numbers
S = h5read(filenameSM, '/measurement/data');
% reinterpret as complex numbers
S = complex(S.r,S.i);
% get rid of background frames
isBG = h5read(filenameSM, '/measurement/isBackgroundFrame');
S = S(isBG == 0,:,:,:);
% size of S : 6859x817x3
% # of grid positions: 6859 = 19x19x19
% # of frequency components : 817
% # of receive coils : 3

% pre-process
% remove the frequencies which are lower than 30 kHz,
% as they are unreliable due to the anologue filter in the scanner
% generate frequency vector
numFreq = h5read(filenameMeas, '/acquisition/receiver/numSamplingPoints')/2+1;
rxBandwidth = h5read(filenameMeas, '/acquisition/receiver/bandwidth');
freq = linspace(0,1,numFreq) .* rxBandwidth; %1x817

% we suppose that the same frequencies are measured on all channel for
% the SM and the measurements
% use only x/y receive channels
idxFreq = freq > 80e3;
S_truncated = S(:,idxFreq,1:2); %6859x764x2

% take calibration measurement from each coil
% take transpose of them to have frequency components
```

```
% in columns, corresponding to each grid position
SystemMatrix.S1 = transpose(S_truncated(:,:,1)); %764x6859
SystemMatrix.S2 = transpose(S_truncated(:,:,2)); %764x6859

% prepare the system matrix
% take real and imaginary parts of complex system matrices
% concetanate the system matrices from two receive coils
S1 = [real(SystemMatrix.S1);imag(SystemMatrix.S1)]; % 1528x6859
S2 = [real(SystemMatrix.S2);imag(SystemMatrix.S2)]; % 1528x6859
S = [S1;S2]; % 3056x6859

%take calibration measurement for each slice and gather them in S_slices
N = 19;
grid_pos_no = N*N; %NxNxN 3D image, one slice is NxN 2D image
patch_no = N;

for k = 1:patch_no
   %take calibration measurements of grid positions for one slice
   S_slice_truncated = S_truncated( (k-1)*grid_pos_no+1: k*grid_pos_no,:,:); %361x764x2

   % take calibration measurement from two receive coils for each slice
   % similar to previous steps, for each slice, take transpose of system matrix
   % to have frequency components in columns, corresponding to each grid position
   SystemMatrix.S1_slice = transpose(S_slice_truncated(:,:,1)); %764x361
   SystemMatrix.S2_slice = transpose(S_slice_truncated(:,:,2)); %764x361

   % prepare the system matrix for each slice
   % take real and imaginary parts of complex system matrices
   % concetanate the system matrices from two receive coils
   S1_slice = [real(SystemMatrix.S1_slice);imag(SystemMatrix.S1_slice)];
   S2_slice = [real(SystemMatrix.S2_slice);imag(SystemMatrix.S2_slice)];
   S_slices(:,:,k) = [S1_slice;S2_slice];

end

%save system matrix
save('SM_2D.mat','S');

%save system matrix for each slice
save('SM_2D_slice.mat','S_slices');

%% Measurement of Resolution Phantom

% load measurement
filenameMeas = 'resolutionPhantom2.mdf';

% for the measurements
% read and convert the data as complex numbers
% note that these data contain 500 measurements
u = h5read(filenameMeas, '/measurement/data');
u = fft(cast(u,'double'));
u = u(1:(size(u,1)/2+1),:,:,:);
```

```
% use only x/y receive channels
u_truncated = u(idxFreq,1:2,:); %764x2x38000

% take measurement data of resolution phantom from each coil
u1 = u_truncated(:,1,:); u1 = reshape(u1,size(u1,1),size(u1,3)); %764x38000
u2 = u_truncated(:,2,:); u2 = reshape(u2,size(u2,1),size(u2,3)); %764x38000

% take average of measurements
measurement.u1 = mean(u1,2); %764x1
measurement.u2 = mean(u2,2); %764x1

% prepare the measurement vector
% take real and imaginary parts of complex system matrices
% concetanate the measurements from two receive coils
u1_real = [real(measurement.u1);imag(measurement.u1)]; %1528x1
u2_real = [real(measurement.u2);imag(measurement.u2)]; %1528x1
u_final = [u1_real;u2_real]; %3056x1

%take measurement for each slice and gather them in u_slices
patch_no = 19;
period_per_patch = 1000;

for k = 1:patch_no

   %take measurements of grid positions for one slice
   u_slice = u(:,:,(k-1)*period_per_patch+1 : k*period_per_patch,:);%817x3x1000x2
   u_slice_truncated = u_slice(idxFreq,1:2,:); %764x2x2000

   % take calibration measurement from two receive coils for each slice
   u1_slice = u_slice_truncated(:,1,:); u1_slice = reshape(u1_slice,size(u1_slice,1),size(u1_slice,3)); %764x2000
   u2_slice = u_slice_truncated(:,2,:); u2_slice = reshape(u2_slice,size(u2_slice,1),size(u2_slice,3)); %764x2000

   % take linear combination of measurements for each slice
   measurement.u1_slice = mean(u1_slice,2); %764x1
   measurement.u2_slice = mean(u2_slice,2); %764x1

   % prepare the measurement vector for each slice
   u1_slice_real = [real(measurement.u1_slice);imag(measurement.u1_slice)]; %1528x1
   u2_slice_real = [real(measurement.u2_slice);imag(measurement.u2_slice)]; %1528x1
   u_slices(:,k) = [u1_slice_real; u2_slice_real]; %3056x1
end

% save measurement vector
u= u_final;
save('meas_resolutionPhantom.mat','u');

% save measurement vector for each slice
save('meas_resolutionPhantom_slice.mat','u_slices');
```

row_norm_treshold function

```
function [S,u] = row_norm_treshold(treshold,S,u)
```

```
[row_no,~] = size(S);
norm_S = sqrt(sum(S.^2,2));

index = find(norm_S<treshold);
S(index,:) = [];
u(index,:) = [];

end
```

row_normalization function

```
function [S,u] = row_normalization(S,u)
   %row normalization
   row_energy = sum(S.^2,2);
   w = 1./row_energy;
   W = diag(w);
   S = sqrt(W)*S;
   u = sqrt(W)*u;
end
```

filtered_svd function

```
function [c,history] = filtered_svd (S,u,lambda,opt)

t = cputime;

[U,Sigma,V] = svd(S,'econ');

switch opt
   case 'singular'
      singular_values = diag(Sigma);
      lambda = singular_values(1)* singular_values(end);
   case 'other'
   %do nothing
end

%solution
new_Sigma =  Sigma + (lambda./Sigma);
c = V*((U'*u)./diag(new_Sigma));

history.time = cputime-t;
history.regularization_parameter = lambda;

end
```

regularized_kaczmarz function

```
function [c,history]= regularized_kaczmarz(S,u,lambda,iter,ref,opt)

t = cputime;

[N,M] = size(S);

row_energy = sum(S.^2,2);
```

```
switch opt
   case 'uniform'
      w = ones(N,1);
      W = eye(N);
   case 'normalized'
      %weighting function
      w = 1./row_energy;
      W = diag(w);
end

identity = eye(N);

c = zeros(M,1);
v = zeros(N,1);
alpha = 0;

for k = 1:iter

   %randomized sub-iterations
   iter_order = randperm(N);
   for i = 1:N
      ind = iter_order(i);
      alpha = ( u(ind) - dot(S(ind,:)',c) -  sqrt(lambda/w(ind) ) * v(ind) ) /...
            ( row_energy(ind) + lambda/w(ind));
      c = c + alpha*(S(ind,:)');
      ei = identity(:,ind);
      v = v + alpha * sqrt(lambda/w(ind)) * ei ;
   end

   %enforce positive values on image
   %c(c<0) = 0;

   history.ima(:,k) = c;
   history.nrmse(k) = sqrt(immse(c,ref))/(max(c)-min(c));
   history.weighting_option= opt;
   history.regularization_parameter = lambda;
   history.obj(k) = objective(S,u,c,W,lambda);
end

history.time = cputime-t;

end

function obj = objective(S,u,c,W,lambda)
   obj = norm(sqrt(W)*(S*c-u),2)^2 + lambda*(norm(c,2))^2;
end
```

L-curve code

```
% only shown for dataset of concentration phantom
%this code is used for choosing optimal lambda values for regularized weighted kaczmarz solution
clear all;
```

```
close all;

load('SM_website.mat','S');
load('meas_website.mat','u');

[U,Sigma,V] = svd(S,'econ');
condition_no = cond(Sigma);
[size_original,~] = size(S);

threshold = 50;
[S,u] = row_norm_treshold(threshold,S,u);
[size_threshold,~] = size(S);

[U,Sigma,V] = svd(S,'econ');
condition_no_threshold= cond(Sigma);

opt = 'other';
lambda= 1e3:1000:1e6;
residual_norm = zeros(length(lambda),1);
solution_norm = zeros(length(lambda),1);
for k = 1:length(lambda)
   [c,history] = filtered_svd(S,u,lambda(k),opt);
   residual_norm(k)= norm(S*c-u,2);
   solution_norm(k) = norm(c,2);
end

figure;set(gcf, 'WindowState', 'maximized');
plot(residual_norm,solution_norm);
xlabel('||Sc-u||');ylabel('||c||');
title('L-curve');
saveas(gcf,'Lcurve_website.png');

%solution_norm = 27.21 and residual_norm = 3330 for lambda_optimal
[~,index] = min(abs(solution_norm-27.21));
website.lambda_optimal = lambda(index);
%lambda_optimal = 11000;
```

ADMM Functions

soft_threshold function

```
%used inside admm functions

function [res] = soft_threshold(arg,threshold)
%
% 1D shrinkage operator.
%
res = max(0,abs(arg)-threshold) .* arg./(abs(arg));
res(isnan(res))=0;
end
```

soft_threshold_2D function

```
%used inside admm functions

function res = soft_threshold_2D(arg,threshold)
%
% 2D shrinkage operator.
%
res = max(0,vecnorm(arg,2,2)-threshold) .* arg./(vecnorm(arg,2,2));
res(isnan(res))=0;
end
```

admm_lasso function

```
function [history, res] = admm_lasso(A,b,rho,mu,tau,step_length,MAX_ITER,varargin)

% performs ADMM to solve LASSO problem, returns history struct and resulted
% vector x.
% Original problem:
% minimize (mu/2)*||Ax-b||^2  + tau * ||x||_1
%
% With ADMM :
% minimize tau * ||z||_1 + (mu/2)*||Ax-b||^2
%
% subject to x = z;
%
% rho : penalty parameter, step_length: step size for dual update,
% variable length depends on the whether reference image is given or not.
%
% reference paper: "Distributed Optimization and Statistical Learning via
% the Alternating Direction Method of Multipliers" by Stephen Boyd, Neal
% Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein.

ref_flag = false; % reference image does not exist in default.

if nargin>7 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end

%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-4;

[~, num_pos] = size(A);
x = zeros(num_pos,1); %primal variable.
z = x; %splitted variable for l1 norm constraint.
y = zeros(num_pos,1); %dual variable of z.


t = cputime;
left_multip =  inv(mu*(A'*A) + rho* eye(num_pos)); % compute only once.

for i= 1:MAX_ITER
```

```
    %% x-update.
    right_multip = mu * A'*b + rho*z-y;
    x = left_multip*right_multip;

    %% z-update.
    z_old = z;
    z = soft_threshold(x+(1/rho)*y,tau/rho);

    %% dual update.
    y = y + step_length * rho*(x-z);

    %% history
    history.obj(i)= objective(A,b,x,mu,tau); %calculates the objective
    % function value in each iteration

    if ref_flag
       %calculates nrmse if reference image exists.
       history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
    end

    %% stopping criteria check.
    residual_primal  = norm(x - z);
    residual_dual  = norm(-rho*(z - z_old));

    tolerance_primal = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(z));
    tolerance_dual = sqrt(num_pos)*ABSTOL + RELTOL*norm(y);

    if ((residual_primal < tolerance_primal) && (residual_dual < tolerance_dual))
       break;
    end

end

res = x;
history.cpu_time = cputime-t;

end

function res = objective(A,b,x,mu,tau)
%calculates LASSO objective
res = (mu/2)*norm(A*x-b)^2 + tau * norm(x,1);
end
```

admm_tv function

```
function [history,result] = admm_tv(A,b,rho,mu,lambda,step_length,MAX_ITER,varargin)

% performs ADMM to solve TV/L2 problem, returns history struct and resulted
% vector x.
% Original problem:
% minimize (mu/2)*||Ax-b||^2 + lambda* TV(x)
%
% With ADMM:
```

```
% minimize sum_over_i(||z_i||_2) + (mu/2)*||Ax-b||^2
%
% subject to D_i * x = z_i where i=1,...,num_of_pos
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% variable length depends on the whether reference image is given or not.
%
% reference paper: "ALTERNATING DIRECTION ALGORITHMS FOR TOTAL VARIATION
% DECONVOLUTION IN IMAGE RECONSTRUCTION" by Min TAO and Junfeng YANG

ref_flag = false; % reference image does not exist in default.

if nargin>7 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end

%Global constants and defaults
ABSTOL   = 1e-5;
RELTOL   = 1e-4;

[~, num_pos] = size(A);
size_1D = sqrt(num_pos);

%% construct difference matrix both in horizontal and vertical direction.
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D1 = spdiags([-1*e e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
border_ind = size_1D * (1:size_1D);
D1(border_ind,:) = 0;

%vertical difference matrix.
e = ones(n,1);
D2 = spdiags([-1*e e],[0,size_1D],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
border_ind = size_1D * (size_1D-1);
D2(border_ind+1:end,:) = 0;

D = [D1', D2']';

%% initialization of x,z,y

x = zeros(num_pos,1); %primal variable.
z = zeros(2*num_pos,1); % spliited variable associated with TV term. First
% dimension is 2*num_pos since horizontal and vertical differences are
% stacked in a fashion that the first half contains the horizontal difference
```

```
% info and the second half contains the vertical
% difference info.
y = zeros(2*num_pos,1); %dual variable of z.


t = cputime;
left_multip = inv(rho*(D'*D) + (mu)* (A'*A));%calculate once, use in every iter

for i = 1:MAX_ITER
   %% x-minimization step.
   right_multip = D'*(rho*z-y) + mu * A'*b;
   x = left_multip * right_multip;

   %% z-minimization step.
   z_old = z;
   finitediff_reordered = reshape(D*x,num_pos,2); %each row represents z_i
   y_reordered = reshape(y,num_pos,2);
   temp = soft_threshold_2D(finitediff_reordered+(1/rho)*y_reordered,lambda/rho);
   z = reshape(temp,2*num_pos,1);

   %% dual variable update.
   y = y - step_length*rho*(z-D*x);

   %% history
   history.obj(i)= objective(A,b,x,mu,lambda,D);%calculates the objective
   % function value for current iteration.

   if ref_flag %calculates nrmse if reference exists
      history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
   end

   %% stopping criteria check.
   residual_primal  = norm(D*x - z);
   residual_dual  = norm(-rho*D'*(z - z_old));

   tolerance_primal = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
   tolerance_dual = sqrt(num_pos)*ABSTOL + RELTOL*norm(y);

   if ((residual_primal < tolerance_primal) && (residual_dual < tolerance_dual))
      break;
   end

end

result = x;
history.cpu_time = cputime-t;
end

function res = objective (A,b,x,mu,lambda,D)
% calculates objective function for TV regularized least square.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2));

end
```

admm_tv_sparse function

```
function [history,res] = admm_tv_sparse(A,b,rho,mu,lambda,tau,step_length,MAX_ITER,varargin)

% performs ADMM to find a sparse solution to the TV/L2 problem. eturns
% history struct and resulted vector x.
%
% Original problem:
% minimize (mu/2)*||Ax-b||^2 + lambda* TV(x) + tau * ||x||_1
%
% With ADMM:
% minimize sum_over_i(||z_i||_2) + tau*||w||_1 + (mu/2)*||Ax-b||^2
%
% subject to D_i * x = z_i where i=1,...,num_of_pos and w = x
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% variable length depends on the whether reference image is given or not.

ref_flag = false; % reference image does not exist in default.

if nargin>8 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end

%Global constants and defaults
ABSTOL   = 1e-5;
RELTOL   = 1e-4;

[~, num_pos] = size(A);
size_1D = sqrt(num_pos);

%% construct difference matrix both in horizontal and vertical direction.
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D1 = spdiags([-1*e e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
border_ind = size_1D * (1:size_1D);
D1(border_ind,:) = 0;

%vertical difference matrix.
e = ones(n,1);
D2 = spdiags([-1*e e],[0,size_1D],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
border_ind = size_1D * (size_1D-1);
D2(border_ind+1:end,:) = 0;
```

```
D = [D1', D2']';

x = zeros(num_pos,1); %primal variable.
w = x; %auxilary variable for sparsity constraint.
y1 = zeros(num_pos,1); %dual variable of sparsity constraint.
z = zeros(2*num_pos,1); % First dimension is 2*num_pos since horizontal and
% vertical differences are stacked in a fashion that the first half contains
% the horizontal difference info and the second half contains the vertical
% difference info.
y2 = z; %dual variable for finite difference constraint.

t = cputime;
left_multip = inv(D'*D + eye(num_pos) + (mu/rho)*(A'*A)); %compute once.

for i=1:MAX_ITER

   %% x-update.
   right_multip = D'*(z-(1/rho)*y2) + (w-(1/rho)*y1) + (mu/rho)* A'*b;
   x = left_multip * right_multip;

   %% w-update. (enforce sparsity)
   w_old = w;
   w = soft_threshold( x + (1/rho)*y1,tau/rho); %1D shrinkage operator

   %% z-update. (TV term)
   z_old = z;
   finitediff_reordered = reshape(D*x,num_pos,2); %each row represents z_i
   y2_reordered = reshape(y2,num_pos,2);
   z = reshape(soft_threshold_2D(finitediff_reordered+(1/rho)*y2_reordered,lambda/rho),2*num_pos,1);

   %% y1 update (dual variable of sparsity constraint i.e w = x)
   y1 = y1 - step_length * rho * (w - x);

   %% y2 update (dual variable of TV constraint i.e z_i = D_i * x)
   y2 = y2 - step_length * rho * (z - D * x);


   %% history
   history.obj(i)= objective(A,b,x,mu,lambda,tau,D);%calculates the
   % objective function value for current iteration

   if ref_flag %calculates nrmse if reference exists
      history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
   end

   %% stopping criteria check.
   residual_primal1  = norm(D*x - z);
   residual_primal2  = norm(x - w);
   residual_dual1  = norm(-rho*D'*(z - z_old));
   residual_dual2  = norm(-rho*(w - w_old));

   tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
```

```
    tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
    tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);
    tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);

    if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && (residual_primal2 <
tolerance_primal2) && (residual_dual2 < tolerance_dual2))
        break;
    end

end

res = x;
history.cpu_time = cputime-t;
end

function res = objective(A,b,x,mu,lambda,tau,D)
%calculates objective least square function with TV and l1 regularization.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2))+ tau*norm(x,1);

end
```

admm_tv_sparse_wo_inverse function

```
function [history,res] = admm_tv_sparse_wo_inverse(A,b,rho,mu,lambda,tau,step_length,MAX_ITER,varargin)
%admm_tv_sparse(A,b,rho,mu,lambda,step_length,tolerance)

% ADMM Function with Total Variation and L1 norm regularization
% performs ADMM to find a sparse solution to the TV/L2 problem. e:time
% spent, res: resulted x vector.
%
% Original problem:
% minimize (mu/2)*||y - Mx||^2 + tau*||x||_1 + lambda*||x||_TV
%
% With ADMM:
% minimize (mu/2)*||y - Mx||^2 + tau*||v||_1 + lambda*||w||_2
% such that v = x, w_i = D_i*x for i = 1...n_y
% D_i*x  Rd represents the first-order finite difference of x at i:th component in d different directions.
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% tolerance: value for relative change in x and used as stopping criteria.
%
% reference paper: "Combined Analysis-L1 and Total Variation ADMM with
% Applications to MEG Brain Imaging and Signal Reconstruction" by Rui Gao,
% Filip Tronarp, and Simo Särkkä

ref_flag = false; % reference image does not exist in default.

if nargin>8 %checks whether reference image is given as input.
    ref_flag = true;
    ref_im = varargin{1};
end
```

```
%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-4;

t_start = cputime;

[~, num_pos] = size(A);
grid_size = sqrt(num_pos);
%% compute the difference matrix D_i for each i
% For the point (i,j), take the difference between (i+1,j) and (i,j+1)
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D_horizontal = spdiags([e -e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
%border_ind = grid_size * (1:grid_size);
D_horizontal(1,:) = 0;
D_horizontal(end,:) = 0;
%vertical difference matrix.
e = ones(n,1);
D_vertical = spdiags([-1*e e],[0,grid_size],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
%border_ind = grid_size * (grid_size-1);
D_vertical(:,1) = 0;
D_vertical(:,end) = 0;

D = [D_horizontal', D_vertical']';

%% ADMM Steps
delta = 1/((rho + rho * normest(D)^2 + mu * norm(A,2)^2));
x = zeros(num_pos,1);
w = zeros(num_pos,1); %sparsity
z = zeros(2*num_pos,1); %TV
y1 = zeros(num_pos,1);
y2 = zeros(2*num_pos,1);


for i = 1:MAX_ITER
   %% x-update
   x = x - delta*(y1 + D'*y2)+delta*rho*(x-w)+delta*rho*D'*(D*x-z)+delta*mu*A'*(b-A*x);
   %% w-update
   w_old = w;
   e = x + y1/rho;
   w = e./abs(e).*max(abs(e)-tau/rho,0); %Enforce sparsity
   %% z-update
   z_old = z;
   t = reshape(D*x,num_pos,2) + reshape(y2,num_pos,2)/rho;
   z = reshape(max(norm(t,2)-lambda/rho,0).*t/norm(t,2),2*num_pos,1); %TV term update
   %% dual variable for sparsity constraint update (eta)
```

```
    y1 = y1 + step_length*rho*(x - w);
    %% dual variable for TV update (ksi)
    y2 = y2 + step_length*rho*(D*x - z); %ksi update
    %% history
    history.obj(i)= objective(A,b,x,mu,lambda,tau,D);%calculates the
    % objective function value for current iteration

    if ref_flag %calculates nrmse if reference exists
       history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
    end

    %% stopping criteria check.
    residual_primal1  = norm(D*x - z);
    residual_primal2  = norm(x - w);
    residual_dual1  = norm(-rho*D'*(z - z_old));
    residual_dual2  = norm(-rho*(w - w_old));

    tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
    tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
    tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);
    tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);

    if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && (residual_primal2 <
tolerance_primal2) && (residual_dual2 < tolerance_dual2))
         break;
    end

end
res = x;
history.cpu_time = cputime-t_start;
end
function res = objective(A,b,x,mu,lambda,tau,D)
%calculates objective least square function with TV and l1 regularization.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2))+ tau*norm(x,1);
end
```

ADMM Functions with Non-Negativity Constraint

admm_lasso_nonnegative function

```
function [history, res] = admm_lasso_nonnegative(A,b,rho,mu,tau,step_length,MAX_ITER,varargin)

% performs ADMM to solve nonnegative LASSO problem, returns history struct
% and resulted vector x.
% Original problem:
% minimize (mu/2)*||Ax-b||^2  + tau * ||x||_1 such that x_i>0 where x_i^'s
% are i^th element of x vector
%
% With ADMM :
% minimize tau * ||z||_1 + (mu/2)*||Ax-b||^2 + i_c(w)
% subject to x = z and x = w;
% i_c is the indicator function of whether its argument is in set C or not.
%
```

```
% rho : penalty parameter, step_length: step size for dual update.
% variable length depends on the whether reference image is given or not.


ref_flag = false; % reference image does not exist in default.

if nargin>7 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end

%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-4;

[~, num_pos] = size(A);

x = zeros(num_pos,1); %primal variable.
z = x; %splitted variable associated with l1 norm
w = x; %splitted variable associated with nonnegativity constraint.
y1 = zeros(num_pos,1); %dual variable of z.
y2 = zeros(num_pos,1); %dual variable of w.

t = cputime;
left_multip =  inv(mu*(A'*A) + 2*rho* eye(num_pos)); % compute only once.

for i = 1:MAX_ITER
   %% x-update.
   right_multip = mu * A'*b + rho*(z+w)-y1-y2;
   temp = x;
   x = left_multip*right_multip;

   %% z-update. (term associated with l1 norm)
   z_old = z;
   z = soft_threshold(x+(1/rho)*y1,tau/rho);

   %% dual update.
   y1 = y1+ step_length * rho*(x-z);

   %% w-update. (term associated with nonnegative constraint)
   w_old = w;
   w = x+(1/rho)*y2;
   w(w<0) = 0;

   %% dual update.
   y2 = y2+step_length*rho*(x-w);

   %% history
   history.obj(i)= objective(A,b,x,mu,tau); %calculates the objective
   % function value in each iteration.

   if ref_flag
      %calculates nrmse if reference image exists.
```

```
        history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
    end

    %% stopping criteria check.
    residual_primal1  = norm(x - z);
    residual_primal2  = norm(x - w);
    residual_dual1  = norm(-rho*(z - z_old));
    residual_dual2  = norm(-rho*(w - w_old));

    tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(z));
    tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
    tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);
    tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);

    if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && ...
            (residual_primal2 < tolerance_primal2) && (residual_dual2 < tolerance_dual2))
        break;
    end

end
res = x;
history.cpu_time  = cputime-t;
end

function res = objective(A,b,x,mu,tau)
%calculates LASSO objective
res = (mu/2)*norm(A*x-b)^2 + tau * norm(x,1);
end
```

admm_tv_nonnegative function

```
function [history,result] = admm_tv_nonnegative(A,b,rho,mu,lambda,step_length,MAX_ITER,varargin)

% performs ADMM to solve TV/L2 problem, e: time spent, result: resulted x.
% Original problem:
% minimize (mu/2)*||Ax-b||^2 + lambda* TV(x) such that x_i>0 where x_i^'s
% are i^th element of x vector
%
% With ADMM:
% minimize sum_over_i(||z_i||_2) + (mu/2)*||Ax-b||^2 + i_c(w)
%
% subject to x = w and D_i * x = z_i where i=1,...,num_of_pos
% i_c is the indicator function of whether its argument is in set C or not.
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% reference paper: "ALTERNATING DIRECTION ALGORITHMS FOR TOTAL VARIATION
% DECONVOLUTION IN IMAGE RECONSTRUCTION" by Min TAO and Junfeng YANG

ref_flag = false; % reference image does not exist in default.

if nargin>7 %checks whether reference image is given as input.
    ref_flag = true;
```

```
    ref_im = varargin{1};
end


%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-4;


t = cputime;
[~, num_pos] = size(A);
size_1D = sqrt(num_pos);

%% construct difference matrix both in horizontal and vertical direction.
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D1 = spdiags([-1*e e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
border_ind = size_1D * (1:size_1D);
D1(border_ind,:) = 0;

%vertical difference matrix.
e = ones(n,1);
D2 = spdiags([-1*e e],[0,size_1D],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
border_ind = size_1D * (size_1D-1);
D2(border_ind+1:end,:) = 0;

D = [D1', D2']';

%% initialization of x,z,y

x = zeros(num_pos,1); %primal variable.
z = zeros(2*num_pos,1); % First dimension is 2*num_pos since horizontal and
% vertical differences are stacked in a fashion that the first half contains
% the horizontal difference info and the second half contains the vertical
% difference info.
w = x;
y1 = zeros(2*num_pos,1); %dual variable for TV term.
y2 = zeros(num_pos,1); %dual variable for nonnegativity constraint.

%% iteration until relative change in x is small enough.

left_multip = inv(rho*(D'*D) + (mu)* (A'*A)+ rho*eye(num_pos));%calculate once, use in every iter

for i = 1:MAX_ITER
    %% x-minimization step.
    right_multip = D'*(rho*z-y1) + mu * A'*b- y2 + rho * w;
```

```
    x = left_multip * right_multip;

    %% z-minimization step.
    z_old = z;
    finitediff_reordered = reshape(D*x,num_pos,2); %each row represents z_i
    y_reordered = reshape(y1,num_pos,2);
    temp = soft_threshold_2D(finitediff_reordered+(1/rho)*y_reordered,lambda/rho);
    z = reshape(temp,2*num_pos,1);

    %% dual variable update.
    y1 = y1 - step_length*rho*(z-D*x);

    %% w-minimization step.
    w_old = w;
    w = x + (1/rho)*y2;
    w(w<0) = 0;

    %% dual variable update.
    y2 = y2 + step_length*rho*(x-w);


    %% history
    history.obj(i)= objective(A,b,x,mu,lambda,D);%calculates the objective
    % function value for current iteration.

    if ref_flag
        %calculates nrmse if reference exists
        history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
    end

    %% stopping criteria check.
    residual_primal1  = norm(D*x - z);
    residual_primal2  = norm(x - w);
    residual_dual1  = norm(-rho*D'*(z - z_old));
    residual_dual2  = norm(-rho*(w - w_old));

    tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
    tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
    tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);
    tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);

    if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && (residual_primal2 <
tolerance_primal2) && (residual_dual2 < tolerance_dual2))
        break;
    end

end
result = x;
history.cpu_time = cputime-t;

end

function res = objective (A,b,x,mu,lambda,D)
```

```
% calculates objective function for TV regularized least square.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2));

end
```

admm_tv_sparse_nonnegative function

```
function [history,res] = admm_tv_sparse_nonnegative(A,b,rho,mu,lambda,tau,step_length,MAX_ITER,varargin)

% performs ADMM to find a sparse solution to the TV/L2 problem. returns
% history struct and resulted vector x.
%
% Original problem:
% minimize (mu/2)*||Ax-b||^2 + lambda* TV(x) + tau * ||x||_1 such that x_i>0
% where x_i^'s are i^th element of x vector
%
% With ADMM:
% minimize (mu/2)*||Ax-b||^2 + sum_over_i(||z_i||_2) + tau*||w||_1  + i_c(s)
%
% subject to D_i * x = z_i where i=1,...,num_of_pos and w = x and s = x
% i_c is the indicator function of whether its argument is in set C or not.
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% variable length depends on the whether reference image is given or not.

ref_flag = false; % reference image does not exist in default.

if nargin>8 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end


%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-3;

[~, num_pos] = size(A);
size_1D = sqrt(num_pos);

%% construct difference matrix both in horizontal and vertical direction.
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D1 = spdiags([-1*e e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
border_ind = size_1D * (1:size_1D);
D1(border_ind,:) = 0;

%vertical difference matrix.
```

```
e = ones(n,1);
D2 = spdiags([-1*e e],[0,size_1D],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
border_ind = size_1D * (size_1D-1);
D2(border_ind+1:end,:) = 0;

D = [D1', D2']';

x = zeros(num_pos,1); %primal variable.
w = x; %auxilary variable for sparsity constraint.
y1 = zeros(num_pos,1); %dual variable of w.
z = zeros(2*num_pos,1); % First dimension is 2*num_pos since horizontal and
% vertical differences are stacked in a fashion that the first half contains
% the horizontal difference info and the second half contains the vertical
% difference info.
y2 = zeros(2*num_pos,1); %dual variable of z.
s = x; %primal term associated with nonnegativity constraint.
y3 = zeros(num_pos,1); %dual variable of s.


t = cputime;
left_multip = inv(rho*(D'*D) + 2*rho*eye(num_pos) + (mu)*(A'*A)); %compute once.
for i = 1:MAX_ITER
   %% x-update.
   right_multip = (mu)* A'*b + D'*(rho*z-y2) - y1 - y3 + rho*(w + s);
   x = left_multip * right_multip;

   %% w-update. (enforce sparsity)
   w_old = w;
   w = soft_threshold( x + (1/rho)*y1,tau/rho); %1D shrinkage operator

   %% y1 update (dual variable of sparsity constraint i.e w = x)
   y1 = y1 - step_length * rho * (w - x);

   %% z-update. (TV term)
   z_old = z;
   finitediff_reordered = reshape(D*x,num_pos,2); %each row represents z_i
   y2_reordered = reshape(y2,num_pos,2);
   z = reshape(soft_threshold_2D(finitediff_reordered+(1/rho)*y2_reordered,lambda/rho),2*num_pos,1);

   %% y2 update (dual variable of TV constraint i.e z_i = D_i * x)
   y2 = y2 - step_length * rho * (z - D * x);

   %% s-update (nonnegativity constraint).
   s_old = s;
   s = x + (1/rho)*y3;
   s(s<0) = 0;

   %% y3 dual variable update.
   y3 = y3 + step_length*rho*(x-w);
```

```
   %% history
   history.obj(i)= objective(A,b,x,mu,lambda,tau,D);%calculates the
   % objective function value for current iteration

   if ref_flag %calculates nrmse if reference exists
      history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
   end

   %% stopping criteria check.
   residual_primal1  = norm(D*x - z);
   residual_primal2  = norm(x - w);
   residual_primal3  = norm(x - s);
   residual_dual1  = norm(-rho*D'*(z - z_old));
   residual_dual2  = norm(-rho*(w - w_old));
   residual_dual3  = norm(-rho*(s - s_old));

   tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
   tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
   tolerance_primal3 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(s));
   tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);
   tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);
   tolerance_dual3 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y3);

   if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && (residual_primal2 <
tolerance_primal2) && (residual_dual2 < tolerance_dual2) && (residual_primal3 < tolerance_primal3) &&
(residual_dual3 < tolerance_dual3))
       break;
   end
end

res = x;
history.cpu_time = cputime-t;
end

function res = objective(A,b,x,mu,lambda,tau,D)
%calculates objective least square function with TV and l1 regularization.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2))+ tau*norm(x,1);
end
```

admm_tv_sparse_wo_inverse_nonnegative function

```
function [history,res] =
admm_tv_sparse_wo_inverse_nonnegative(A,b,rho,mu,lambda,tau,step_length,MAX_ITER,varargin)
%admm_tv_sparse(A,b,rho,mu,lambda,step_length,tolerance)

% ADMM Function with Total Variation and L1 norm regularization
% performs ADMM to find a sparse solution to the TV/L2 problem. e:time
% spent, res: resulted x vector.
%
% Original problem:
% minimize (mu/2)*||y - Mx||^2 + tau*||x||_1 + lambda*||x||_TV
%
```

```
% With ADMM:
% minimize (mu/2)*||y - Mx||^2 + tau*||v||_1 + lambda*||w||_2
% such that v = x, w_i = D_i*x for i = 1...n_y
% D_i*x  Rd represents the first-order finite difference of x at i:th component in d different directions.
%
% rho: penalty parameter in augmented Lagrangian.
% step_length:step size for dual update,
% tolerance: value for relative change in x and used as stopping criteria.
%
% reference paper: "Combined Analysis-L1 and Total Variation ADMM with
% Applications to MEG Brain Imaging and Signal Reconstruction" by Rui Gao,
% Filip Tronarp, and Simo Särkkä

ref_flag = false; % reference image does not exist in default.

if nargin>8 %checks whether reference image is given as input.
   ref_flag = true;
   ref_im = varargin{1};
end

%Global constants and defaults
ABSTOL   = 1e-4;
RELTOL   = 1e-3;

t_start = cputime;

[~, num_pos] = size(A);
grid_size = sqrt(num_pos);
%% compute the difference matrix D_i for each i
% For the point (i,j), take the difference between (i+1,j) and (i,j+1)
%horizontal difference matrix.
n = num_pos;
e = ones(n,1);
D_horizontal = spdiags([e -e],0:1,n,n);

% horizontal finite difference can not be calculated for the pixels on the
% right edge. So, make the operator zero for that pixels.
%border_ind = grid_size * (1:grid_size);
D_horizontal(1,:) = 0;
D_horizontal(end,:) = 0;
%vertical difference matrix.
e = ones(n,1);
D_vertical = spdiags([-1*e e],[0,grid_size],n,n);

% vertical finite difference can not be calculated for the pixels on the
% bottom edge. So, make the operator zero for that pixels.
%border_ind = grid_size * (grid_size-1);
D_vertical(:,1) = 0;
D_vertical(:,end) = 0;

D = [D_horizontal', D_vertical']';

%% ADMM Steps
```

```
delta = 1/((rho + rho * normest(D)^2 + mu * norm(A,2)^2));
x = zeros(num_pos,1);
w = zeros(num_pos,1); %sparsity
z = zeros(2*num_pos,1); %TV
y1 = zeros(num_pos,1);
y2 = zeros(2*num_pos,1);
y3 = zeros(num_pos,1);
s =x;

for i = 1:MAX_ITER
   %% x-update
   x = x - delta*(y1 + D'*y2 + y3)+delta*rho*(x-w)+delta*rho*D'*(D*x-z)+delta*mu*A'*(b-A*x) +
delta*rho*(x-s);
   %% w-update
   w_old = w;
   e = x + y1/rho;
   w = e./abs(e).*max(abs(e)-tau/rho,0); %Enforce sparsity
   %% z-update
   z_old = z;
   t = reshape(D*x,num_pos,2) + reshape(y2,num_pos,2)/rho;
   z = reshape(max(norm(t,2)-lambda/rho,0).*t/norm(t,2),2*num_pos,1); %TV term update
   %% dual variable for sparsity constraint update (eta)
   y1 = y1 + step_length*rho*(x - w);
   %% dual variable for TV update (ksi)
   y2 = y2 + step_length*rho*(D*x - z); %ksi update
   %% s-update (nonnegativity constraint).
   s_old = s;
   s = x + (1/rho)*y3;
   s(s<0) = 0;

   %% y3 dual variable update.
   y3 = y3 + step_length*rho*(x-w);
   %% history
   history.obj(i)= objective(A,b,x,mu,lambda,tau,D);%calculates the
   % objective function value for current iteration

   if ref_flag %calculates nrmse if reference exists
      history.nrmse(i) = sqrt(immse(x,ref_im))/(max(x)-min(x));
   end

   %% stopping criteria check.
   residual_primal1  = norm(D*x - z);
   residual_primal2  = norm(x - w);
   residual_primal3  = norm(x - s);
   residual_dual1  = norm(-rho*D'*(z - z_old));
   residual_dual2  = norm(-rho*(w - w_old));
   residual_dual3  = norm(-rho*(s - s_old));

   tolerance_primal1 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(D*x), norm(z));
   tolerance_primal2 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(w));
   tolerance_primal3 = sqrt(num_pos)*ABSTOL + RELTOL*max(norm(x), norm(s));
   tolerance_dual1 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y2);
   tolerance_dual2 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y1);
```

```
    tolerance_dual3 = sqrt(num_pos)*ABSTOL + RELTOL*norm(y3);

    if ((residual_primal1 < tolerance_primal1) && (residual_dual1 < tolerance_dual1) && (residual_primal2 <
tolerance_primal2) && (residual_dual2 < tolerance_dual2) && (residual_primal3 < tolerance_primal3) &&
(residual_dual3 < tolerance_dual3))
        break;
    end
end
res = x;
history.cpu_time = cputime-t_start;
end
function res = objective(A,b,x,mu,lambda,tau,D)
%calculates objective least square function with TV and l1 regularization.
res = (mu/2)*norm(A*x-b)^2 + lambda * sum(vecnorm(D*x,2,2))+ tau*norm(x,1);
end
```