Image Recovery by A Three-Operator Splitting Scheme

WANG, Chuning

December 11, 2017

Abstract

In this research, to recover a image from less than half of information, we apply a convex optimization algorithm, which is a particular application of a three-operator splitting scheme. Our code worked quite well on testing data. However, the real-world data were too big so we encountered some problems. Although the original image of real-world data has not yet been recovered to be clear enough, our analysis shows that iterating the code for more times may finally lead to a satisfactory result. Apart from that, there are also several improvements and we are still making efforts to apply them.

1 Introduction

As technology improves, cameras are designed to detect the world in a more and more subtle and precise way. Many interesting problems related are also emerging, such as image recovery, which is our aim in this study. Image recovery attempts to recover a whole image from reduced information by solving an optimization problem including not only a loss term, but also regularization terms representing properties that the original image should possess. Usually the formula has regularization terms that are not trivial at all so that it could hardly be solved directly. Therefore, we introduce algorithms to make solving the optimization problem possible. In this report, we could find that our algorithm requires calculating an operator related to only one term in each step. The optimization problem would therefore be simplified and solved more easily by iterating the algorithm.

2 Original Problem

As shown in the work of Liu et al. [1], GISC spectral camera is an imaging technique to detect a 3D spectral image data-cube by scanning one of its 2D section at a time. The recovering process of image taken by GISC spectral camera could be treated as solving the following optimization problem:

$$\min_{\mathbf{x}} ||\mathbf{y} - \mathbf{A}\mathbf{x}||_2^2 + \mu_1||\tilde{\mathbf{X}}||_{\mathrm{TV}} + \mu_2||\mathbf{X}||_*, \text{ s.t. } \mathbf{x} \geq \mathbf{0}.$$

Here $\mathbf{y}$ and $\mathbf{A}$ are given, where $\mathbf{y} \in \mathbb{R}^m$ is the observed signal, and $\mathbf{A} \in \mathbb{R}^{m\times n}(m << n)$ is the measurement matrix. $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n_1 n_2 \times n_3}$, and $\tilde{\mathbf{X}} \in \mathbb{R}^{n_1\times n_2\times n_3}$ are the vector form, matrix form, and cube form of unknown spectral images, where GISC spectral camera detects the $n_1 \times n_2$ image under n_3 different wavelengths and $n = n_1 n_2 n_3$. Moreover, the first term in this formula is a loss term, else are regularization terms with μ_1,$\mu_2 > 0$ being their weights respectively. More specifically, TV term is the sum of 2D TV-norm of images of different wavelengths:

$$\begin{aligned} ||\tilde{\mathbf{X}}||_{\mathrm{TV}} &= \sum_{\lambda=1}^{n_3} ||\tilde{\mathbf{X}}(:,:,\lambda)||_{\mathrm{TV}} \\ &= \sum_{\lambda=1}^{n_3}\sum_{j=1}^{n_2}\sum_{i=1}^{n_1} \sqrt{(\tilde{\mathbf{X}}_{i+1,j,\lambda} - \tilde{\mathbf{X}}_{i,j,\lambda})^2 + (\tilde{\mathbf{X}}_{i,j+1,\lambda} - \tilde{\mathbf{X}}_{i,j,\lambda})^2}. \end{aligned}$$

And $||\mathbf{X}_*||$ is the nuclear norm of spectral image matrix:

$$||\mathbf{X}_*|| = \sum_{i=1}^{n_3} \sigma_i,$$

where σ_i denotes the singular value of $\mathbf{X}$.

3 Algorithm

3.1 Three-Operator Splitting Scheme

The three-operator splitting scheme [2] was originally put forward by David and Yin to solve all the problems finding $x \in \mathcal{H}$ such that $0 \in Ax + Bx + Cx$, for three maximal monotone operators A, B, C defined on a Hilbert Space $\mathcal{H}$, where the operator C is cocoersive. Here we define an operator C to be β-cocoercive ($\beta > 0$) if

$$\langle Cx - Cy, x - y \rangle \geq \beta ||Cx - Cy||^2, \forall x, y \in \mathcal{H}.$$

To solve this problem, we introduce an operator:

$$T := I_{\mathcal{H}} - J_{\gamma B} + J_{\gamma A} \circ (2J_{\gamma B} - I_{\mathcal{H}} - \gamma C \circ J_{\gamma B}),$$

where $\gamma \in (0, 2\beta)$, $I_{\mathcal{H}}$ is the identify map in $\mathcal{H}$, and $J_S := (I + S)^{-1}$ denotes the resolvent of a monotone operator S. Then we iterates the following sentence:

$$z^{k+1} := (1 - \lambda_k) z^k + \lambda_k T z^k,$$

where we start from an arbitrary point z^0. Moreover, in this equation, $\lambda_k \in (0, (4\beta - \gamma)/2\beta)$ is a relaxation parameter. And we could fix $\gamma < 2\beta$ and $\lambda_k \equiv 1$.

3.2 Application on Optimization

Taking $f(x)$ to be TV norm term, $g(x)$ to be nuclear norm term, and $h(Lx)$ to be the loss term, our original problem could be view as finding x that could minimize

$$f(x) + g(x) + h(Lx),$$

where f, g, and h are proper,closed, and convex functions. Moreover, h is $(1/\beta)$-Lipschitz-differentiable, and L is a linear mapping. It is worth noticing that we have a constraint $\mathbf{x} \geq \mathbf{0}$. However, any constraint $\mathbf{x} \in \mathcal{C}$ could be handled as adding an indicator function

$$i_{\mathcal{C}}(x) = \begin{cases} 0 & x \in \mathcal{C} \\ +\infty & x \notin \mathcal{C} \end{cases}$$

to either f or g since it would not influence their convex property.

In order to solve the optimization problem by three-operator splitting scheme, we may take

$$A = \partial f, B = \partial g, C = \nabla(h \circ L) = L^* \circ \nabla h \circ L,$$

where ∂f and ∂g are subdifferentials of f and g respectively. It is obvious that in our settings A, B, and C are all maximal monotone operators. And we may also notice that ∇h of the $(1/\beta)$-Lipschitz-differentiable convex function h is β-cocoercive. Therefore, such minimization problem would become a particular case that could be solved by the splitting scheme we introduced. When $S = \partial f$, $J_S(x)$ would become the proximal map:

$$\text{argmin}_y f(y) + \frac{1}{2}||x - y||^2.$$

So we could begin with any z^0, set stepsize $\gamma \in (0, 2/(\beta||L||^2))$. And we would finally get z^i converge to x minimizing

$$f(x) + g(x) + h(Lx)$$

by Algorithm 1.

Algorithm 1

for $k = 0, 1, ...$ **do**
    $x^k = \mathbf{prox}_{\gamma g}(z^k)$
    $y^k = Lx^k$
    $z^{k+\frac{1}{2}} = 2x^k - z^k - \gamma L^* \nabla h(y^k)$
    $z^{k+1} = z^k + \lambda_k(\mathbf{prox}_{\gamma f}(z^k + \frac{1}{2}) - x^k)$
end for

3.3 Coordinate Descent

According to the provider of real-world data, at least 40% of the information should be preserved. It means that for m, n such that the measurement matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we must have $m \geq 0.4 \times n$. For the real-world data provided, it means that $\mathbf{A}$ would occupy at least 36 GB. But the memory of my computer was only 16 GB. Therefore, it was impossible for it to read the whole $\mathbf{A}$ into memory at a time.

To solve this problem, we adopted coordinate descent, equally dividing $\mathbf{A}$ and $\mathbf{x}$ into n_3 corresponding parts (A_i *and* SOL_i) according to n_3 different wavelengths. And we updated SOL_i, an $n_1 \times n_2$ section of x, under a wavelength at a time. In this way, we need only to read a corresponding A_i, an $m \times n_1 n_2$ part of $\mathbf{A}$, and its transpose A_i^T into memory at a time. Substituting $f(x)$ by TV norm term, $g(x)$ by nuclear norm term, and $h(x)$ by the loss term, the detailed procedure is shown in Algorithm 2 below. To avoid any confusion that may be caused by complex notations, we first list them and clarify their meaning:
$A_{n_1 \times n_2 \times n_3}$: Measurement matrix.
A_i: The measurement matrix A was equally cut into n_3 parts horizontally, A_i is the i-th part.
SOL_i: Accordingly, the unknown image x was also cut into n_3 parts, and each time the i-th part SOL_i was updated.
epoch: In each epoch, every SOL_i would be updated once in the order of $i = 1, 2, 3, ..., n_3$.
m: Number of epochs that we will go through.
μ_1: Weights of the TV term.
μ_2: Weights of the nuclear term.
γ: Stepsize.
λ: A sequence of relaxation parameters.
maxit: Number of iterations in Algorithm 1 in order to update a SOL_i in every epoch.

Algorithm 2

for $epoch = 1 : m$ **do**
  for $i = 1 : n_3$ **do**
    $other\ terms \leftarrow \sum_{k=1,k\neq i}^{n_3} A_k * SOL_k$
    $x \leftarrow SOL_i$
    procedure OPTM($x, y, A_i, A_i^T, other\ terms, n1, n2, \mu_1, \mu_2, \gamma, \lambda, maxit$) ▷ Update x to minimize the given formula.
      $z_0 \leftarrow x$
      for $j = 1 : maxit$ **do**
        $x_j \leftarrow \mathbf{prox}_{2\gamma\mu_2||\ ||_*}(z_{j-1})$
        $m_j \leftarrow 2 * x_j - z_{j-1} - \gamma * 2 * A_i^T * (A_i * x_j + other\ terms - y)$
        $z_j \leftarrow z_{j-1} + \lambda_j * (\mathbf{prox}_{2\gamma\mu_1(||\ ||_{TV} + I_{\{\mathbf{x}\geq\mathbf{0}\}})}(m_j) - x_j)$
      end for
      return z_{maxit}
    end procedure
    $SOL_i \leftarrow z_{maxit}$
  end for
end for

I initialized $\lambda = \text{ones}([maxit, 1])$, $\gamma = 1/(2 \times \max(\text{eigenvalue}(A^T \cdot A)))$, $SOL(1 : n_3) = \{zeros(n_1 \times n_2, 1)\}$. And I used UNLocBoX[3] to calculate the two proximals. It is also worth mentioning that while calculating proximal we use TV norm and nuclear norm of the matrix form of $SOL_i \in \mathbb{R}^{n_1 \times n_2}$

4 Experiments on Testing Data

4.1 Data Processing

I tested the effect of Algorithm 2 with a $40 \times 90 \times 7$ image, which looks like Figure 1 as seven 40×90 pictures. I separately adopted two matrices $\mathbf{A}$ to process the original $\mathbf{x}$. It is also worth mentioning that since I adopted coordinate descent to save memory, both matrix would be cut into seven matrices A_i, each with 40×90 columns to update SOL_i, corresponding part divided from $\mathbf{x}$ with $n_1 \times n_2$ entries. Unfortunately, I have not found the best way to set parameters including γ, μ_1, μ_2 yet.

Figure 1: This is the image of original testing data.

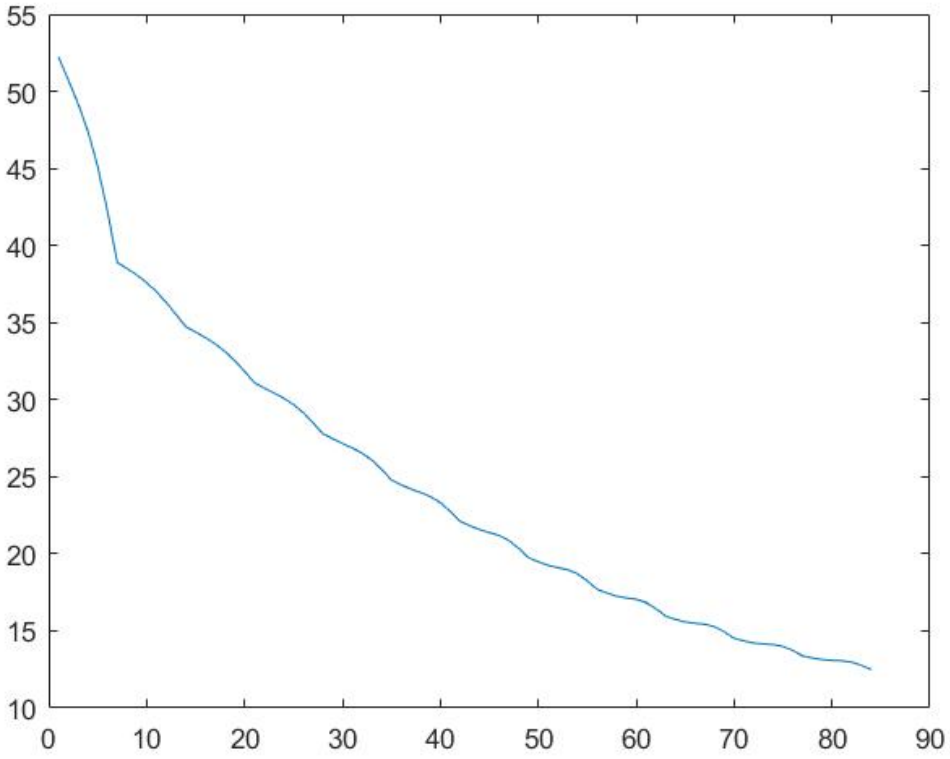
4.2 A with down-sample effect

In this experiment A is a $(40 * 90 * 7/3) \times (40 * 90 * 7)$ matrix with down-sample effect. It means that if $\mathbf{x} = (x_1, x_2, x_3, ...)$, we would have $\mathbf{Ax} = (x_1, x_4, x_7, ...)$. I also set both μ_1, μ_2 to be 0.048, and set γ to be 0.5. I set epoch $m = 100$ and $maxit = 3$ for both cases, which means the whole $\mathbf{x}$ (i.e. SOL_1 to SOL_7) would be updated for 100 times, and every SOL_i would update itself for 3 times each epoch. Running the program takes about 1630s in total. I found that x_j converged well. The recovered image is shown in Figure 2, and the l_2-norm distance of my recovered $\mathbf{x}$ and the original $\mathbf{x}$ after updating each SOL_i for $maxit$ times is in Figure 3.

Figure 2: This is the normalized image of **x** recovered after processing by our down-sample **A**.

	1	2	3	4	5	6	7
1	52.2305	50.7373	49.1977	47.4246	45.1474	42.2327	38.9020
2	38.4991	38.0843	37.5920	36.9990	36.3063	35.5305	34.7349
3	34.3884	34.0254	33.6174	33.1270	32.5321	31.8405	31.0981
4	30.7614	30.4319	30.0911	29.6769	29.1489	28.5016	27.7817
5	27.4561	27.1515	26.8717	26.5362	26.0774	25.4749	24.7859
6	24.4770	24.1955	23.9670	23.7079	23.3151	22.7572	22.1129
7	21.8239	21.5661	21.3790	21.1919	20.8602	20.3518	19.7503
8	19.4835	19.2508	19.1003	18.9725	18.7014	18.2468	17.7006
9	17.4594	17.2546	17.1387	17.0525	16.8399	16.4455	15.9579
10	15.7492	15.5765	15.4913	15.4318	15.2720	14.9433	14.5238
11	14.3506	14.2125	14.1510	14.1082	13.9902	13.7268	13.3800
12	13.2395	13.1346	13.0892	13.0579	12.9719	12.7682	12.4919

(a) Table.



(b) Graph.

Figure 3: (Part of) the $l_2 - norm$ distance of my recovered **x** and the original **x** for down-sample **A**.

4.3 Randomly Generated A

In this experiment, I set measurement matrix $\mathbf{A}$ to be a randomly generated $(40*90)\times(40*90*7)$ matrix with each entry between 0 and 10^{-4}. I set both μ_1, μ_2 to be 2×10^{-6}, and set γ to be $1/2$, while all eigenvalues of $A_i^T A_i$ are in $[0,1]$. Moreover, I set epoch $m = 1000$ and $maxit = 3$, which means the whole $\mathbf{x}$ (i.e. SOL_1 to SOL_7) would be updated for one thousand times, and every SOL_i would update itself for 3 times each epoch. Running the program takes about 12000s in total, most of which are spent on reading and calculation involving A_i^T and A_i. Moreover, by measuring $||x_j - x_{j-1}||_\infty$, I found that x_j converged well.

Figure 4 shows the recovered image. It could be noticed that the first block is very light and the blocks after it become darker and darker. As far as I am concerned, the reason is that while running *line* 5 to *line* 12 of Algorithm 2, we are actually updating SOL_i such that $A_i \cdot SOL_i$ would approach $\mathbf{y}$. So when i increases, the space for each entry of SOL_i to improve becomes smaller and smaller. I also tried to update SOL_i in a random order. However, we still got recovered images that seem only like changing the order of the seven blocks in Figure 4.

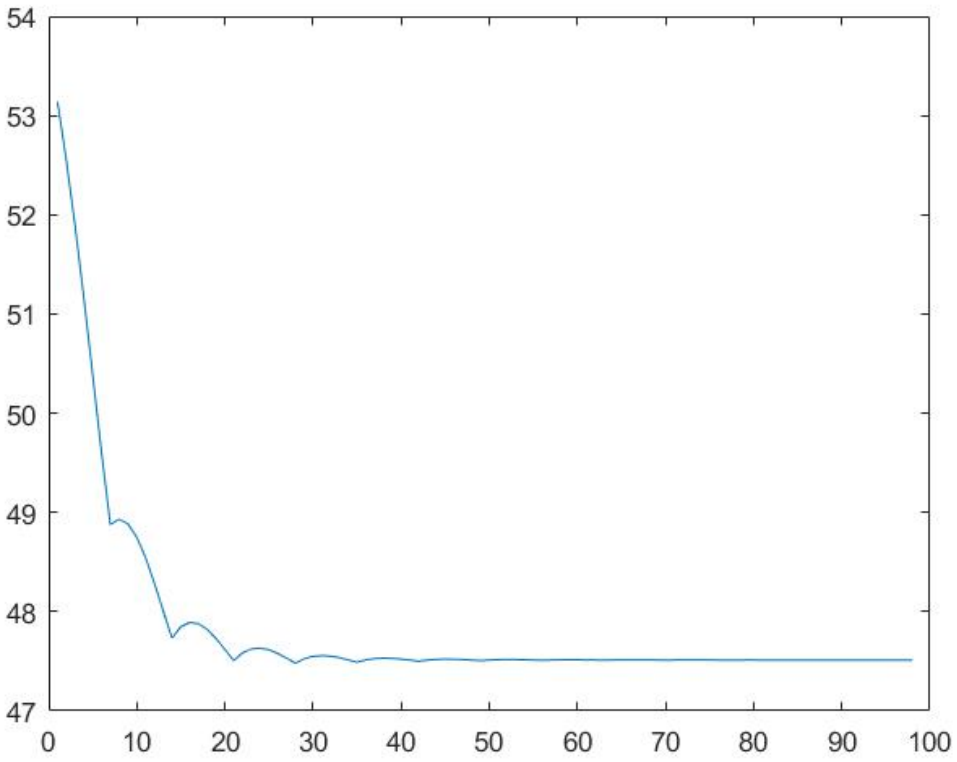
Figure 4: This is the image of $\mathbf{x}$ recovered after processing by our random matrix $\mathbf{A}$.

To see more clearly, I normalized each SOL_i and got the one in Figure 5, which looks closer to our original picture. Moreover, I again measured the $l_2 - norm$ distance of my recovered x and the original x after updating each SOL_i after $maxit$ times. And the result is shown in Figure 6, showing that the recovered image is getting closer and closer to the original one, though in a quite low speed.

Figure 5: This is the normalized image of $\mathbf{x}$ recovered after processing by our random matrix $\mathbf{A}$.

	1	2	3	4	5	6	7
1	53.1391	52.5491	51.8855	51.1564	50.3822	49.6019	48.8752
2	48.9329	48.8828	48.7486	48.5440	48.2875	48.0058	47.7338
3	47.8474	47.8907	47.8787	47.8209	47.7289	47.6178	47.5064
4	47.5848	47.6240	47.6322	47.6148	47.5781	47.5302	47.4808
5	47.5254	47.5493	47.5570	47.5512	47.5352	47.5131	47.4900
6	47.5136	47.5267	47.5314	47.5292	47.5218	47.5112	47.4999
7	47.5121	47.5189	47.5215	47.5205	47.5169	47.5117	47.5061
8	47.5123	47.5157	47.5170	47.5166	47.5148	47.5122	47.5094
9	47.5124	47.5142	47.5148	47.5146	47.5137	47.5123	47.5109
10	47.5124	47.5132	47.5135	47.5134	47.5129	47.5122	47.5115
11	47.5122	47.5126	47.5127	47.5126	47.5124	47.5120	47.5115
12	47.5119	47.5121	47.5121	47.5120	47.5119	47.5116	47.5114
13	47.5115	47.5116	47.5116	47.5115	47.5114	47.5113	47.5111
14	47.5111	47.5111	47.5111	47.5111	47.5110	47.5109	47.5108

(a) Table.



(b) Graph.

Figure 6: (Part of) the table showing the $l2-norm$ distance of my recovered $\mathbf{x}$ and the original $\mathbf{x}$ for randomly generated $\mathbf{A}$.

In order to compare our result with that directly optimizing

$$\min_{\mathbf{x}} ||\mathbf{y} - \mathbf{A}\mathbf{x}||_2^2,$$

I also ran the same code only changing μ_1 and μ_2 to be zero. The corresponding results are shown in Figure 7 and Figure 8.

Figure 7: This is the image of $\mathbf{x}$ recovered after processing by our random matrix $\mathbf{A}$ with $\mu_1 = \mu_2 = 0$.

Figure 8: This is the normalized image of $\mathbf{x}$ recovered after processing by our random matrix $\mathbf{A}$ with $\mu_1 = \mu_2 = 0$.

5 Experiments on Real-World Data

5.1 Data Processing

Real-world data includes a $(140 * 140 * 9) \times (140 * 140 * 9)$ square matrix $\mathbf{R}$ and a $140 \times 140 \times 90$ observed signal $\mathbf{y}$. And our task is to pick a part of $\mathbf{y}$ and its corresponding rows of $\mathbf{R}$ and recover the whole unknown spectral images. As I mentioned before, at least 40% of data should be preserved to recover the original image according to providers of the real-world data. Therefore, we chose 4/9 entries of $\mathbf{y}$, as well as corresponding lines of $\mathbf{R}$ to be our measurement matrix $\mathbf{A}$.

5.2 Results and Analysis

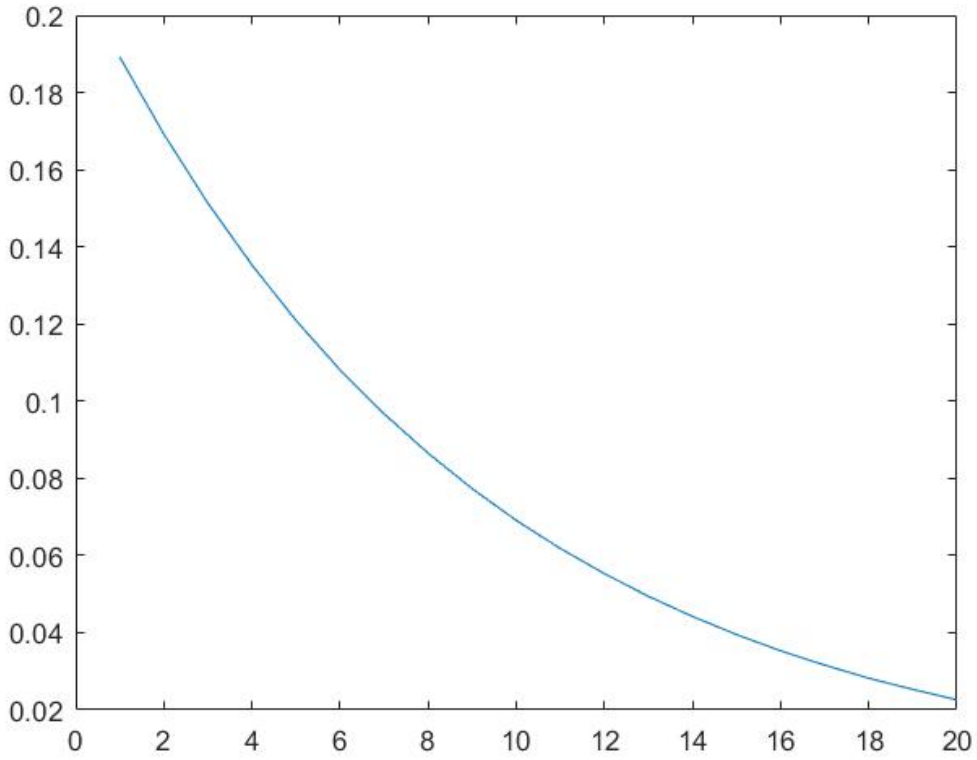
Since the size of $\mathbf{A}$ was too big, it took my program quite a long time to read A_i and calculate line 9 of the algorithm. I ran it for a whole day only to get the recovered image (as shown in Figure 8 & Figure 9) after going through 20 epochs, which seemed not containing too much information, and my computer had already stopped responding. It was so difficult to run enough epochs to reflect informations we need, not to mention adjusting parameters to get a clearer recovered image. But I suggest that it would converge to a better result after running enough epochs. I calculated the $l_2 - norm$ distance between $\mathbf{x}$ recovered after each $i - th$ and $(i + 1) - th$ epoch and the result is shown in Figure 10. It is obvious that as more epochs were run, the distance between $\mathbf{x}$ we recovered was becoming smaller and smaller.

Figure 9: This is the image of x recovered from part of y and A.

Figure 10: This is the normalized image of x recovered from part of y and A.

1	2	3	4	5	6	7	8	9	10	11	12
0.1892	0.1692	0.1513	0.1353	0.1209	0.1081	0.0967	0.0865	0.0773	0.0691	0.0618	0.0553
13	14	15	16	17	18	19	20	21	22	23	24
0.0494	0.0442	0.0395	0.0353	0.0316	0.0282	0.0253	0.0226	0	0	0	0

(a) Table.



(b) Graph.

Figure 11: This shows the distance between x recovered after each i-th and (i+1)-th epoch.

6 Further Improvements

As we can see, the results are not satisfactory enough. In my opinion, there are two reasons. The first one is that the codes to recover image from real-world data have not run enough epochs to converge due to the limited capability of my computer. According to our experiments on testing data, it could be hard to draw any convincing negative conclusion about the effectiveness of the algorithm on real-world data before running for hundreds of epochs. And the second one is that adopting coordinate descent prevents each SOL_i directly converging to that part of original image, as we have explained in Section 4.3.

We put forward two improvements as an attempt to deal with the problems. The first one is to use a server with larger memory to run more epochs (such as HPC2 Cluster provided by HKUST). In this way, we may even be able to give up coordinate descent and directly use the whole measurement matrix $\mathbf{A}$ to update the whole $\mathbf{x}$ in an epoch. The second potential improvement is to use deep learning ideas to decide the free parameters based on training data [4]. Basically, we may start from an optimization algorithm, reduce it to $5-10$ (or so) iterations and change some components each iteration to free parameters. The training will take time. But after training is done, a $5-10$ iteration neural network would be built, and time for running the code would be significantly reduced.

7 Conclusion

In this study, we attempted to recover an image taken by GISC spectral camera from less than half information provided. The approach we adopted was solving an optimization problem with additional constrictions related to the property of image. Algorithm we introduced to solve this problem was a particular case of a three-operator splitting scheme. Its effectiveness was first examined on test data before we actually apply it on real-world data. And we could see that it basically performed well on testing data, with some problems remaining at the same time. However, when it came to real-world data, the size of data was too big so that we could only run 20 epochs and get a result providing us hardly any valuable information. Considering problems we were facing, two further improvements were put forward, which I am still working on to seek better recovering results.

References

[1] Z. Liu, S. Tan, J. Wu, E. Li, X. Shen, and S. Han, "Spectral camera based on ghost imaging via sparsity constraints," *Scientific reports*, vol. 6, 2016.

[2] D. Davis and W. Yin, "A three-operator splitting scheme and its optimization applications," *arXiv preprint arXiv:1504.01032*, 2015.

[3] N. Perraudin, D. Shuman, G. Puy, and P. Vandergheynst, "UNLocBoX A matlab convex optimization toolbox using proximal splitting methods," *ArXiv e-prints*, Feb. 2014.

[4] Y. Chen, W. Yu, and T. Pock, "On learning optimized reaction diffusion processes for effective image restoration," *CoRR*, vol. abs/1503.05768, 2015.