Image inpainting

Name: Chendi Guan , Kexin Zhu
RIN: 661609844 , RIN : 661547433

11 May, 2017

Abstract

This program implements the process of image inpainting.

By solving the underdetermined system problem to get the optimization $|wX|_1$, then we can get X as the final inpainting image. We chose the wavelet tight frame transformation to set the constraint [1]. During the process of minimization, our algorithm can make sure that it satisfies the constraint, which is the equivalence of non-contaminated part of both images (original one and the optimization one). The whole idea for this project is to apply the ADMM method, to get the contaminated pixels similar with its surrounding pixels [1]. As a result, it can reconstruct the image by getting the pixel at the location of characters. The general process is that we optimize $Q = wX$, and apply FONC for X and Q on Lagrange equation [1].

1 Problems

We get one contaminated image. First, we need to separate two parts: contaminated part and uncontaminated part. Second, we need to get rid of contaminated part, and reconstruct this part. Third, to reconstruct contaminated part, we need to optimize this pixels approach to the uncontaminated part.

To make this work mathematically. First, we need to load the pixels of two parts(Ω and Ω^C), we can read this two parts of one image directly. Second, the image is a matrix which size is 256×256, so the combination of contaminated part and uncontaminated part should also have the size 256×256. Third, after apply function swt2 to X, it will convert the image to wX three dimensions, $256 \times 256 \times 9$, we let $Q = wX$. After minimizing Q, we can get X by reducing the dimensions from Q using iwst2 (mathematically, through multipling w^T).

Given the data read from the image, we can easily get a coefficient matrix wX. We need to minimize $|wX|_1$ under the constraint that the new image of uncontaminated

part is always equal to the original image of uncontaminated part. By solving the optimization problem, we need to apply the ADMM method [1]. First write down the Lagrange equation, by introducing λ and μ, through the first order necessary condition (FONC), to solve the optimal X and Q [1].

During the iterations, we can get the optimal value of wX, when X approaches to the tolerance or the program exceeds the maximum iteration, the answer is almost convergent, so we can get the reconstructed image. According to the equation to update X and Q, it is easy to get X through FONC. To update three dimensional Q, we just need to shrink Q in channel 2-9 (3-D matrix in respect to height, from 2 to 9) [1]. Through updating X and Q, make the contaminated part approach to the optimization, and make sure uncontaminated part unchanged. After hundreds times of update, we can get the optimized X and Q, using the data to show image, then we can get the inpainting image.

Chendi worte the pseudocode for shrink and customNorm, code the whole program, test initialization value of project and modify the report. Kexin wrote the pseudocode for the Lagrange equation and wrote the report sketch and helping with coding and test initialization value of the project.

2 Methodology

2.1 Model

(Here Ω represents the image not contaminated, f represents the original image)

$$R_\Omega(X) = \begin{cases} X_{ij} & ij \in \Omega \\ 0 & ij \notin \Omega \end{cases} \tag{1}$$

$$\min_X |wX|_1, \quad \text{s.t.} \quad R_\Omega(X) = R_\Omega(f). \tag{2}$$

$$\min_X |Q|_1, \quad \text{s.t.} \quad R_\Omega(X) = R_\Omega(f) \quad \& \quad Q = wX. \tag{3}$$

2.2 Algorithm

$$\mathcal{L}(X,Q) = |Q|_1 + < R_\Omega(f) - R_\Omega(X), \lambda > + \frac{r_1}{2}\|\mathbf{R_\Omega(X)} - \mathbf{R_\Omega(f)}\|^2 + < wX - Q, \mu > + \frac{r_2}{2}\|\mathbf{wX} - \mathbf{Q}\|^2$$

$$X^k \leftarrow argmin\mathcal{L}(X, Q^{k-1}; \lambda^{k-1}, \mu^{k-1})$$

$$Q^k \leftarrow argmin\mathcal{L}(X^k, Q; \lambda^{k-1}, \mu^{k-1})$$

$$\lambda^k = \lambda^{k-1} + R_\Omega(f) - R_\Omega(X)$$

$$\mu^k = \mu^{k-1} + wX - Q$$

Using first order condition to solve X and Q.

Here are the derivations for X and Q at iteration k.

$$X^k = argmin|Q|_1 + < R_\Omega(f) - R_\Omega(X), \lambda > + \frac{r_1}{2}\|\mathbf{R_\Omega(X)} - \mathbf{R_\Omega(f)}\|^2 + < wX - Q, \mu > + \frac{r_2}{2}\|\mathbf{wX} - \mathbf{Q}\|^2$$

$$-R_\Omega(\lambda) + r_1(R_\Omega(X) - R_\Omega(f)) + w^T\mu + r_2 w^T(wX - Q) = 0$$

let i, j be index of entries in X

$$ij \in \Omega \quad 0 = -\lambda + r_1(X - f) + w^T\mu + r_2 w^T(wX - Q)$$

$$(r_1 Id + r_2 w^T w)X = \lambda + r_1 f + w^T(r_2 Q - \mu) \quad (since\ w^T w = Id)$$

$$(r_1 + r_2)X = \lambda + r_1 f + w^T(r_2 Q - \mu)$$

$$X = \frac{\lambda + r_1 f + w^T(r_2 Q - \mu)}{r_1 + r_2}$$

$$ij \notin \Omega \quad 0 = w^T\mu + r_2 w^T(wX - Q)$$

$$(r_2 w^T w)X = w^T(r_2 Q - \mu) \quad (since\ w^T w = Id)$$

$$X = \frac{w^T(r_2 Q - \mu^T)}{r_2}$$

Since the non-contaminated part can't be changed because of the constraint. Which means we do not need to change the part in Ω. Thus, we only have r in the real program in stead of r_1, r_2 here.

$$Q^k = argmin|Q|_1 + < R_\Omega(f) - R_\Omega(X), \lambda > + \frac{r_1}{2}\|\mathbf{R_\Omega(X)} - \mathbf{R_\Omega(f)}\|^2 + < wX - Q, \mu > + \frac{r_2}{2}\|\mathbf{wX} - \mathbf{Q}\|^2$$

$$Q^k = argmin|Q|_1 + < -Q, \mu > + \frac{r_2}{2}\|\mathbf{wX} - \mathbf{Q}\|^2$$

$$= shrink(wX^k + \frac{1}{r_2}\mu, \frac{1}{r_2})$$

Algorithm 1 ADMM algorithm

Step 0.
λ is a matrix 256x256 ones
μ is a matrix 256x256x9 ones
$r_2 = 0.75$
(Although we need both r_1, r_2 in the Lagrange method, but after calculation, we do not need r_1 any more, so we just need to initialize r_2.)
while not converge **do**
Step 1. $X^k = \frac{w^T(r_2 Q^{k-1} - \mu^T)}{r_2}$ (Updating formula for X in contaminated range)
Step 2. $Q^k = shrink(wX^k + \frac{1}{r_2}\mu, \frac{1}{r_2})$ (Updating formula for Q).
Step 3. $\lambda^k = \lambda^{k-1} + R_\Omega(f) - R_\Omega(X^k)$(Updating formula for λ).
Step 4. $\mu^k = \mu^{k-1} + wX^k - Q^k$ (Updating formula for μ).
end while

Algorithm 2 CustomNorm algorithm

Step 1. loop through every entry in the matrix
Step 2. Compute the summation of all entries' absolute value

Algorithm 3 Shrink algorithm

Step 1. loop through channel 2-9 in the matrix (z direction in 3-D matrix)
Step 2. loop through x in the matrix
Step 3. loop through y in the matrix
Step 4. shrink that entry with $\frac{1}{r_2}$

3 Results

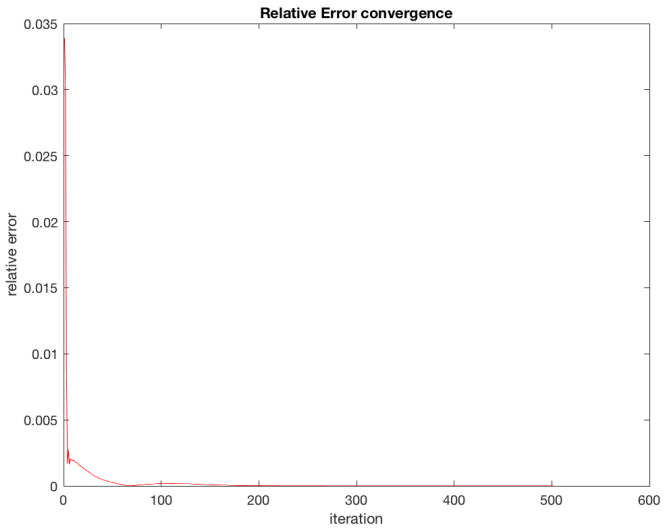
Barbara

After 502 iterations, which means let the optimization process going for 502 times, the relative error of L-1 Norm of Q is less than 1×10^{-8}.

The result form matlab is as below:

```
relativeError =
   1.9684e-09
iteration =
   502
```

Here is the Relative Error convergence graph:



From the graph, we can see the relative error decreases dramatically at first and then the convergence rate become small, but the convergence process made the relative within tolerance eventually.

Which means the difference of Q between two iterations is super small so that it can be ignored and treated as equal.

Since the X which is a matrix with dimension of 256×256, instead of showing 65536 numbers, showing the result image can be the most intuitive.

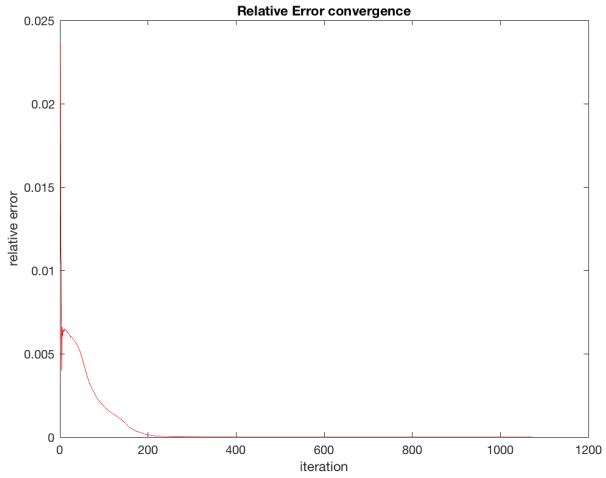
Figure 1: Left:barbara contaminated, Right: barbara inpainting.

Cameraman

The result form matlab is as below:

```
relativeError =
   7.7628e-09
iteration =
        1073
```

Here is the Relative Error convergence graph:



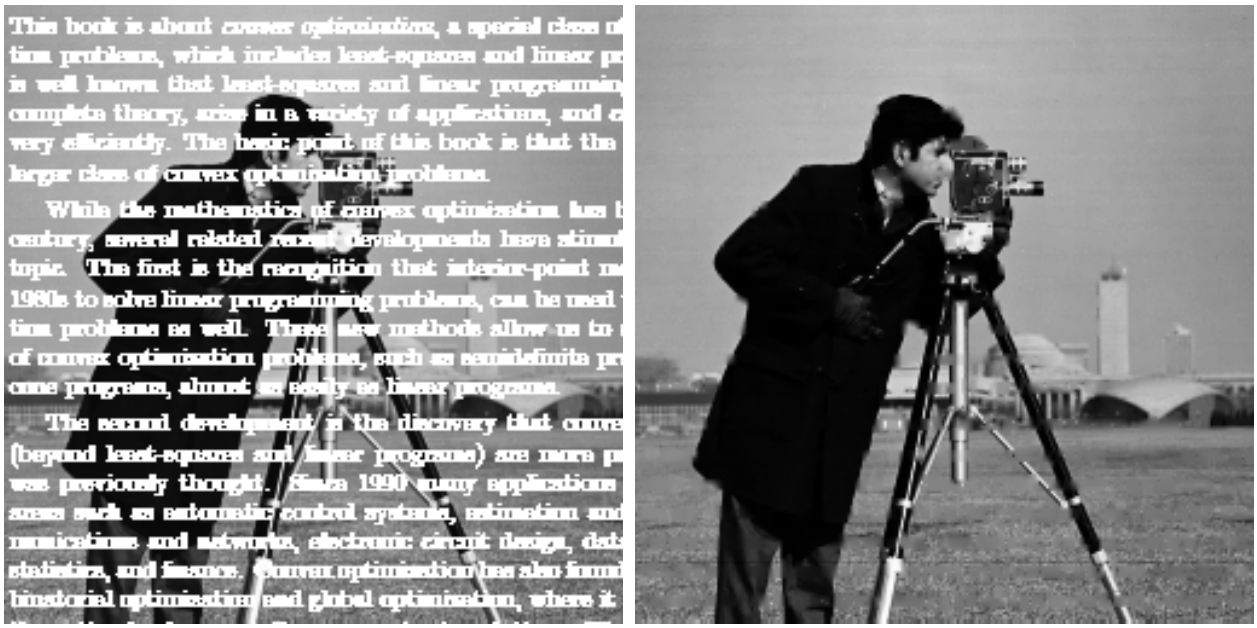


Figure 2: Left:cameraman contaminated, Right: cameraman inpainting.

4 Observation and Conclusions

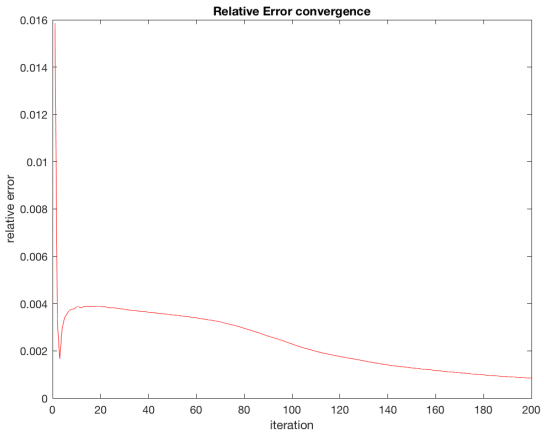
In our algorithm, the result image can be pretty close to inpainting image with good initial value guessing within 200 iterations. But we choose to let the program continue doing process until the relative error reaches tolerance.

After several tests, we get the observation below:

The initialization value will affect the result a lot, it is important to test and get the best initialization value. Since after calculation, there is no need to use r_1 because r_1 is only useful to calculate uncontaminated part, which we just need to preserve the original one. Thus here, we use $r = r2$ as in code for convenience.

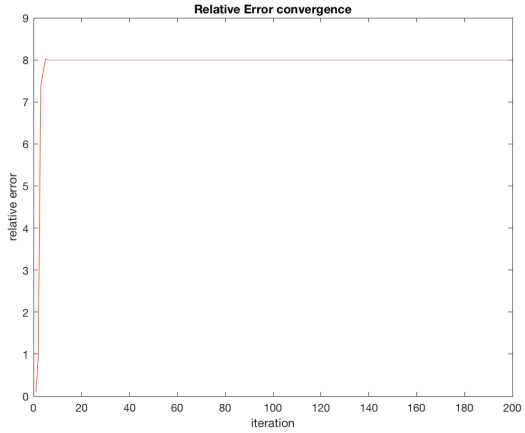
When $r > 1$, after 200 iterations, there is still some characters on the image.

For example $r = 1.25$, after 200 iterations, we can still see most of characters on the image.



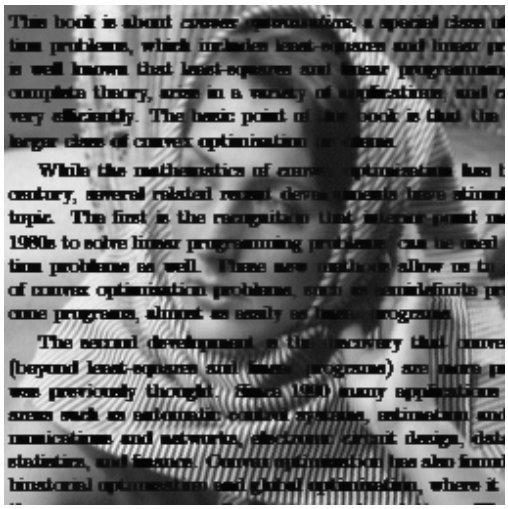
From the convergence graph we can see the relative error is still considerable compared to tolerance.

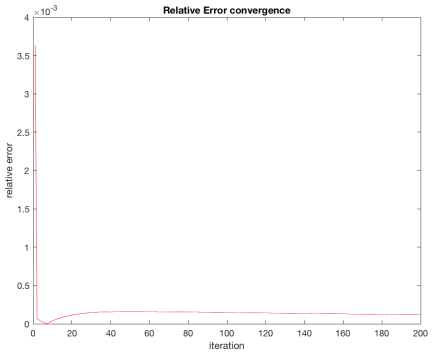
When $r = 0.1$, after two hundreds of iterations, there is only character on the figure, and cannot find the uncontaminated part.



From the convergence graph we can see the relative error becomes quite large in only a few iterations and keep constant. We think it's because the color is black (numerically 0) and it can't be smaller numerically.

When $r > 10$, it is almost indistinguishable for naked eyes after 200 iterations.





From the convergence graph we can see the relative error keeps constant on 10^{-4}. We think it's because r is too big, then $\frac{1}{r}$ is too small, thus shrink process would become inefficiently.

After trying for a lot of initial value for r, we decide that $r = 0.75$

When initialize λ and μ, if all the entry in the matrix is 0, then at the first iterations, the relative error will become bigger, then it will converge to 0 gradually.

When initialize λ and μ, if all the entry in the matrix is 1, then at the first iterations, the relative error will become smaller efficiently, then it will converge to 0 gradually. Finally, we choose to make tolerance 10^{-8}, and the maximum iteration 1200, then it will be efficient as well as get a ideal result.

Use Alternative Direction Minimization Method (ADMM) to solve the underdetermined system problem. So we can get the minimization X as the picture without characters on it. Using ADMM method is very efficient, because it takes two related variables, and update them simultaneously. In general, when the tolerance is very small and the max iteration to be very large, then the inpainting image will be much more clear, and approach to the original uncontaminated picture.

It is obvious that apply ADMM method to real-world problems will be very useful. Through doing this project we can see that ADMM method can reconstruct image efficiently. It is also impressive to convert a image problem to a optimization problem.

References

[1] Rongjie Lai
`Computational Optimization,Chapter 9 Convex Optimization and Chapter 10 Topic, Spring 2017.`

Appendix

Main.m

```
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the first image
X = double(imread('barbara_contaminated.png'));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the second image
%X = double(imread('cameraman_contaminated.png'));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Origin = X; % represent the original picture
load Omega  % I1(Omega), I2(Omega) are not contaminated

% omega complement,
% I1(OmegaComplement), I2(OmegaComplement) are contaminated
OmegaComplement = setdiff([1:256^2],Omega);

% show the original picture
figure;
imshow(Origin,[]);

% max iteration number
maxIterations = 1200;
% initialize iteration number to 0
iteration = 1;
% relative error (initialized to infinity to start the while loop)
relativeError = inf;
% set tolenrance to 10^(-8)
tol = 1e-8;

% initial value
r = 0.75;
Q = swt2(X,1,1);
lambda = ones(256,256);
mu= ones(256,256,9);
diff = zeros(maxIterations,1); % use to store the history difference to plot

while relativeError > tol
    oldX = X;
    oldNorm = customNorm(Q);

    %%% update X
```

```
    % tempX is the new temperory image we get, and since we only need to
    % change the part which is contaminated
    % (the restraint is R_omega(X) = R_omega(f)) where f is the original image
    % so we appiled the equation in the second line below
    tempX = (iswt2((r*Q-mu),1,1))/(r);
    X(OmegaComplement) = tempX(OmegaComplement);

    %%% update Q
    Q = updateQ(swt2(X,1,1)+1/(r)*mu, r);

    %%% update lambda
    lambda = lambda + X- oldX;

    %%% update mu
    mu = mu + swt2(X,1,1) - Q;

    % update error
    newNorm = customNorm(Q);
    relativeError = abs( (newNorm-oldNorm)/max(1,oldNorm));
    diff(iteration) = relativeError;

    % reach the maximum iterations, then break the loop
    if iteration == maxIterations
        break
    end
    iteration = iteration + 1;
end

% to show the numerical result
relativeError
iteration

% plot the convergence of relative error
iterations = linspace(1, iteration, iteration);
diffs = diff(1:iteration);
figure;
plot( iterations', diffs, 'r-');
title('Relative Error convergence')
```

```
xlabel('iteration'); ylabel('relative error');

% show the final result image
figure;
imshow(X,[]);
```

customNorm.m

```
% return the sum of each L1-norm of each 2-D child matrix in 3-D matrix X
% also can be used to calculate 2-D matrix since height of 2-D matrix is 1
% also can be used to calculate 1-D matrix since width of 1-D matrix is 1
function temp = customNorm(X)
    [sizeX,sizeY,sizeZ] = size(X);
    temp = 0;
    for i = 1:sizeX
        for j = 1:sizeY
            for k = 1:sizeZ
                temp = temp + abs(X(i,j,k));
            end
        end
    end
end
```

updateQ

```
function newQ = updateQ(Q, r)
    % this function is designed to update Q
    % bascially it just shrink each entry in Q(:,:,2:9)
    [sizeX,sizeY,sizeZ] = size(Q);
    newQ = zeros(sizeX,sizeY,sizeZ);
    newQ(:,:,1) = Q(:,:,1);
    for channel = 2:sizeZ
        for i=1:sizeX
            for j = 1:sizeY
                newQ(i,j,channel) = sign(Q(i,j,channel))*...
                    max(0, abs(Q(i,j,channel))-1/r);
```

```
            end
        end
    end
end
```

swt2.m and iswt2.m are given