Augmented Lagrangian

Name: Edward Rusu
RIN: 660878003

7 April 2016

1 Problem

In our previous report, we explored the solution of the ROF denoising model

$$\min_u \mathcal{F}(u) = \int_\Omega |\nabla u| dx + \frac{\lambda}{2} \int_\Omega (u - u_0)^2 dx$$

using gradient descent, lagged diffusivity, and dual projection. Now, we consider the Augmented Lagrangian (Split-Bregman) approach. We will apply Augmented Lagrangian to ROF denoising, image in-painting, and compressed sensing.

1.1 ROF

To start, consider the ROF model

$$\min_u \mathcal{F}(u) = \int_\Omega |Du| dx + \frac{\lambda}{2} \int_\Omega (u - u_0)^2 dx$$

for image denoising. If we implement the splitting $p = \nabla u$, then our problem becomes

$$\min_{u,p} \mathcal{L}(u, p) = \int_\Omega |p| dx + \frac{\lambda}{2} \int_\Omega (u - u_0)^2 dx.$$

We introduce a penalty term that acts against the distance between u and ∇p, which gives us the Lagrangian

$$\mathcal{L}(u, p; B) = \int_\Omega |p| dx + \frac{\lambda}{2} \int_\Omega (u - u_0)^2 dx + \frac{r}{2} \int_\Omega (\nabla u - p + B)^2 dx. \tag{1}$$

We will implement Augmented Lagrangian to denoise the image shown in Figure (1).

1.2 Image In-Painting

Image in-painting is very similar to image denoising. Consider a subdomain $D \subset \Omega$ that has been corrupted. We will implement a type of harmonic extension to fill in the corrupted image. The harmonic extension PDE is

$$\begin{cases} \Delta u = 0 \\ u|_{\partial D} = u_0 \end{cases}.$$

This PDE is the variational derivative of

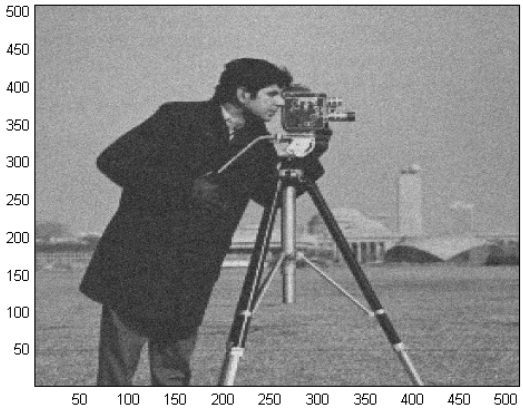

Figure 1: Noisy Image.

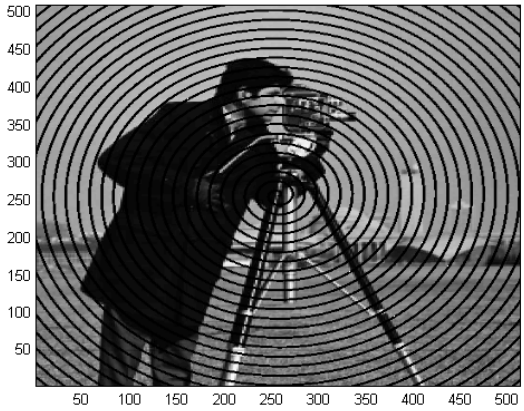

Figure 2: Corrupted Image.

$$\min_u \int_\Omega |\nabla u|^2 dx + \frac{\lambda}{2} \int_{\Omega/D} (u - u_0)^2 dx.$$

We don't want to work with integrals over different domains. We define a characteristic function

$$\chi_D(x) = \begin{cases} 1 & \chi \in D \\ 0 & \text{Otherwise.} \end{cases}$$

so that we can rewrite the functional as

$$\min_u \int_\Omega |\nabla u|^2 dx + \frac{\lambda}{2} \int_\Omega (1 - \chi_D)(u - u_0)^2 dx.$$

As we saw in our first paper, the L_2 space induces blur, so we will adjust the regularity term so that we minimize in the L_1 space

$$\min_u \int_\Omega |\nabla u| dx + \frac{\lambda}{2} \int_\Omega (1 - \chi_D)(u - u_0)^2 dx. \tag{2}$$

Note that this functional is nearly identical to Equation (1) for the ROF model. Thus, total variation appears in image in-painting. The image we will in-paint is shown in Figure (2).

1.3 Compressed Sensing

The problem of compressed sensing is

$$\min_x |x| \text{ such that } Ax = b,$$

for which the Lagrangian is

$$\mathcal{L}(x; \lambda) = |x| + \langle \lambda, Ax - b \rangle.$$

We add a penalty for the distance between Ax and b

$$\mathcal{L}(x;\lambda) = |x| + \langle \lambda, Ax - b\rangle + \frac{r}{2}||Ax-b||^2$$
$$\mathcal{L}(x;\lambda) = |x| + \frac{r}{2}||Ax - b + \lambda||^2.$$

Lastly, we apply splitting $p = x$, so that our Augmented Lagrangian is

$$\mathcal{L}(x,p;\lambda,\mu) = |p| + \frac{r_1}{2}||Ax - b + \lambda||^2 + \frac{r_2}{2}||x - p + \mu||^2. \tag{3}$$

2 Methodology

2.1 ROF

The Augmented Lagrangian procedure for Equation (1) is

$$u^k \leftarrow argmin_u \ \frac{\lambda}{2}||u^{k-1} - u_0||^2 + \frac{r}{2}||\nabla u^{k-1} - p^{k-1} + B^{k-1}||^2$$
$$p^k \leftarrow argmin_p \int_\Omega |p^{k-1}|dx + \frac{r}{2}||\nabla u^k - p^{k-1} + B^{k-1}||^2$$
$$B^k \leftarrow B^{k-1} + \nabla u^k - p^k.$$

The actual steps of the algorithm are written below

1. Solve for u
$$(-\Delta + \lambda)u^k = \lambda u_0 + r\nabla \cdot (B^{k-1} - p^{k-1}) \tag{4}$$

2. The update for p is just the shrink function
$$p^k = \max\left\{0, 1 - \frac{1}{r|\nabla u^k + B^{k-1}|}\right\}(\nabla u^k + B^{k-1})$$

3. Update B
$$B^k = B^{k-1} + \nabla u^k - p^k$$

The RHS in Equation (4) contains a divergence operator and a gradient $p = \nabla u$. We discretize these operators according to [1]. The Laplacian operator in the LHS is discretized using the standard five-point stencil, which results in a block tri-diagonalal matrix. We will use Matlab's conjugate gradient to solve Equation (4).

2.2 In-painting

Recall the in-painting model Equation (2). We can minimize this functional with the Augmented Lagrangian iteration

1. Solve for u
$$(-r\Delta + \lambda(1 - \chi_D))u^k = r\nabla \cdot (B^{k-1} - p^{k-1}) + \lambda(1 - \chi_D)u_0$$

2. Update p

$$p^k = shrink(\nabla u^k + B^{k-1}, r)$$

3. Update B

$$B^k = B^{k-1} + \nabla u^k - p^k$$

This procedure is very similar to that in the ROF model, but with a slightly different equation to update u.

2.3 Compressed Sensing

Recall the compressed sensing model Equation (3). We can minimize this functional with the Augmented Lagrangian iteration

1. Solve for x

$$(r_1 A^T A + r_2)x^k = r_1 A^T (b - \lambda^{k-1}) + r_2(p^{k-1} - \mu^{k-1})$$

2. Update p

$$p^k = shrink(x^k + \mu^{k-1}, r_2)$$

3. Update λ

$$\lambda^k = \lambda^{k-1} + Ax^k - b$$

4. Update μ

$$\mu^k = \mu^{k-1} + x^k - p^k$$

3 Results

The Augmented Lagrangian for ROF produced somewhat disappointing results, as shown in Figure (4). Parameters here were $\lambda = 0.5$ and $r = 0.1$. As we saw with the Tikhonov model, we find ourselves balancing between noise and blur. This is unexpected, since we used total variation as a regularity term.

In-painting turned out much better. As we can see in Figure (6), the corruption is completely removed. Parameters here were $\lambda = 0.5$ and $r = 0.075$.

Compressed Sensing worked very well. With parameters $r_1 = 100$ and $r_2 = 0.1$, we see convergence with 10^{-12} in only six iterations. A plot of the error with each iteration is shown in Figure (7). The number of zeros is around 500, giving us a somewhat sparse result.

4 Conclusion

In conclusion, we see that Augmented Lagrangian for the ROF model does not compare as well as previous iterators, such as lagged diffusivity and Chambolle's dual projection. However, it works very well with image in-painting and compressed sensing.

References

[1] Antonin Chambolle. *An algorithm for total variation minimization and applications.* CEREMADE–CNRS UMR 7534, Université de Paris-Dauphine, 75775 Paris Cedex 16, France.

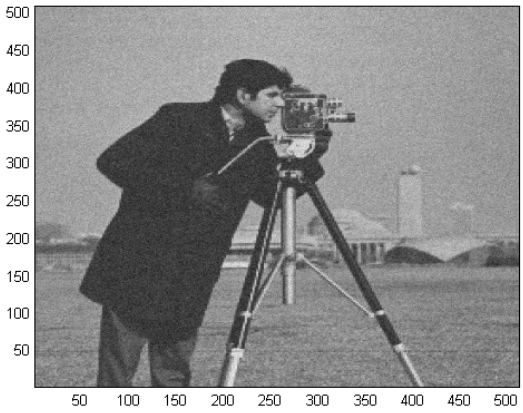

Figure 3: Original noisy image.

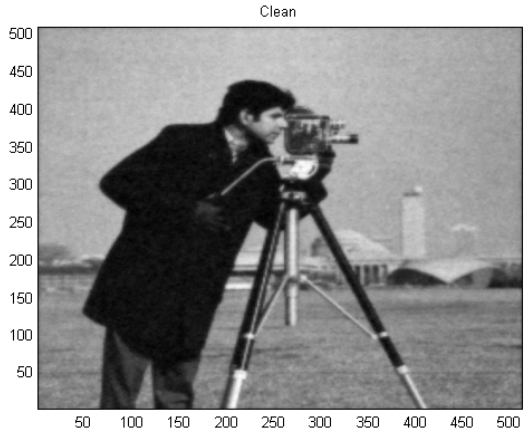

Figure 4: 20 iterations of Augmented Lagrangian for ROF denoising.

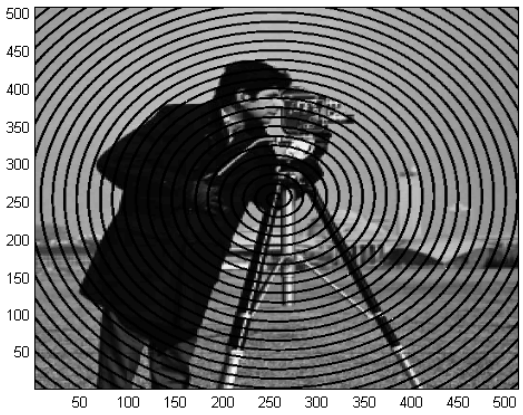

Figure 5: Original corrupted image.

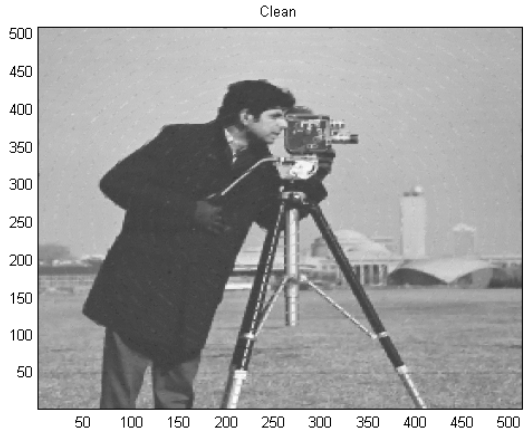

Figure 6: 20 iterations of Augmented Lagrangian for in-painting.

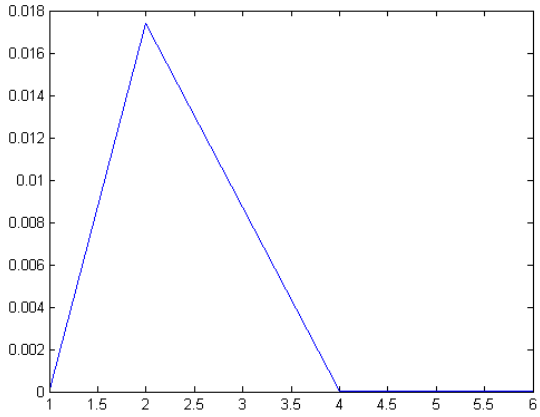

Figure 7: Max error in $Ax - b$ in each iteration. After six iterations, the error converges to less than 10^{-12}.

Appendix A

AL for ROF

```
%% Initialize Program

[u_xact, u0, ~, ~] = LoadData;
[y,x] = size(u0); % Get the dimensions important for differencing
u0 = u0(:); % Reshape the image into a vector: L-R columns stack T-B

% Pick some parameters here
maxIter = 20;
lmda = 0.5;
r = 0.1;
B_nxt = zeros(y*x,2);

%% Construct Operators
% y = 3; x = 4;
n = y*x;
e = ones(n,1);
e1 = ones(n,1);
for i = y:y:n
    e1(i) = 0;
end
e2 = ones(n,1);
for i = y+1:y:n
    e2(i) = 0;
end

% Lapalcian
L = spdiags([e e1 -4*e e2 e],[-y -1 0 1 y],n,n);
```

```
% x-Gradient
Dx = spdiags([-e e],[0 y],n,n);

% y-Gradient
Dy = spdiags([e1 -e],[-1 0],n,n);

% x-Divergence
e3 = ones(n,1);
e3(end-y+1:end) = 0;
Divx = spdiags([-e e3],[-y 0],n,n);

% y-Divergence
e4 = ones(n,1);
for i = y:y:n
    e4(i) = 0;
end
e5 = ones(n,1);
for i = 1:y:n
    e5(i) = 0;
end
Divy = (spdiags([-e5 e4],[-1 0],n,n)).';

%% Augmented Lagrangian Iteration
p_nxt = [Dx*u0 Dy*u0];
u_nxt = u0;
for j = 1:maxIter
    u = u_nxt;
    B = B_nxt;
    p = p_nxt;

%     divr = Divx*(B(:,1)-p(:,1)) + Divy*(B(:,2)-p(:,2));
    divr = Divx*(B(:,1)-p(:,1)) + Divy*(B(:,2)-p(:,2));
    [u_nxt,flag] = cgs(-L + lmda*speye(n),lmda*u0 + r*divr,[],20); % Just do 1 iteration of cgs

    u_grad = [Dx*u_nxt Dy*u_nxt];
    p_nxt = shrnk(r,u_grad+B,2);

    B_nxt = B + u_grad - p_nxt;

end

%% Plot results
u = reshape(u_nxt,y,x);
u0 = reshape(u0,y,x);

figure(1), clf
pcolor(flipud(u0)), title('Noisy'), shading interp, colormap gray
```

```
figure(2), clf
pcolor(flipud(u)), title('Clean'), shading interp, colormap gray
```

AL for In-painting

```
%% Initialize Program
[~,~,u0,D] = LoadData;
[y,x] = size(u0); % Get the dimensions important for differencing
u0 = u0(:); % Reshape the image into a vector: L-R columns stack T-B
D = D(:);

% Pick some parameters here
maxIter = 20;
lmda = 0.5;
% r = 0.1;
r = 0.075;
B_nxt = zeros(y*x,2);

%% Construct Operators
% y = 3; x = 4;
n = y*x;
e = ones(n,1);
e1 = ones(n,1);
for i = y:y:n
    e1(i) = 0;
end
e2 = ones(n,1);
for i = y+1:y:n
    e2(i) = 0;
end

% Lapalcian
L = spdiags([e e1 -4*e e2 e],[-y -1 0 1 y],n,n);

% x-Gradient
Dx = spdiags([-e e],[0 y],n,n);

% y-Gradient
Dy = spdiags([e1 -e],[-1 0],n,n);

% x-Divergence
e3 = ones(n,1);
e3(end-y+1:end) = 0;
Divx = spdiags([-e e3],[-y 0],n,n);

% y-Divergence
e4 = ones(n,1);
for i = y:y:n
```

```
    e4(i) = 0;
end
e5 = ones(n,1);
for i = 1:y:n
    e5(i) = 0;
end
Divy = (spdiags([-e5 e4],[-1 0],n,n)).';

%% Augmented Lagrangian Iteration
D2 = spdiags(1-D,0,n,n);
p_nxt = [Dx*u0 Dy*u0];
u_nxt = u0;
for j = 1:maxIter
    u = u_nxt;
    B = B_nxt;
    p = p_nxt;

%     divr = Divx*(B(:,1)-p(:,1)) + Divy*(B(:,2)-p(:,2));
    divr = Divx*(B(:,1)-p(:,1)) + Divy*(B(:,2)-p(:,2));
    [u_nxt,flag] = cgs(-r*L + lmda*D2, lmda*(1-D).*u0 + r*divr,[],20); % Just do 1 iteration o

    u_grad = [Dx*u_nxt Dy*u_nxt];
    p_nxt = shrnk(r,u_grad+B,2);

    B_nxt = B + u_grad - p_nxt;

end

%% Plot results
u = reshape(u_nxt,y,x);
u0 = reshape(u0,y,x);

figure(1), pcolor(flipud(u0)), title('Noisy'), shading interp, colormap gray
figure(2), pcolor(flipud(u)), title('Clean'), shading interp, colormap gray
```

AL for Compressed Sensing

```
%% Initialize Program
[A,b,u0] = LoadDataForCompressiveSensing;
[y,x] = size(A);

% Pick some parameters here
r1 = 100;
r2 = 0.1;
maxIter = 200;

%% Construct Operators
LHS = r1*(A'*A) + r2*speye(x);
```

```
%% Augmented Lagrangian Iteration
u_nxt = u0;
lmda_nxt = A*u0-b;
p_nxt = u0;
mu_nxt = u0-p_nxt;
data1 = zeros(1,maxIter);
data2 = zeros(1,maxIter);
for j = 1:maxIter
    u = u_nxt;
    p = p_nxt;
    lmda = lmda_nxt;
    mu = mu_nxt;

    [u_nxt,flag] = cgs(LHS, r1*A'*(b-lmda) + r2*(p-mu),[],20);
%     u_nxt = cgs(LHS, r1*A'*(b-lmda) + r2*(p-mu),[],200);
%     p_nxt = shrnk3(u_nxt - mu,r2);
    p_nxt = (1-1./(r2*abs(u_nxt + mu))).*(u_nxt + mu);
    lmda_nxt = lmda + A*u_nxt - b;
    mu_nxt = mu + u_nxt - p_nxt;

    data1(j) = norm(A*u_nxt - b,inf);
    if (norm(A*u_nxt - b,inf) < 1e-10)
        break;
    end
    data2(j) = sum(u_nxt < 1e-12);
end

%% Plot results
data1 = data1(1:j);
plot(1:j,data1);
% plot(1:maxIter,data1,'r',1:maxIter,data2,'b');
```