The Bregman CookBook

Jerome Gilles
jegilles@math.ucla.edu
http://www.math.ucla.edu/∼jegilles

Version 2.0

October, 30, 2011

Chapter 1

Generalities

1.1 Introduction

In these notes, we review the different Bregman Iteration methods and give some examples of implementation. The reference paper is the work of Goldstein and Osher [1, 3]

1.2 Bregman Iteration

The original Bregman iteration was designed to solve problem of the form

$$\hat{u} = \arg\min J(u) \qquad \text{subject to} \quad H(u) = 0 \tag{1.1}$$

where u is a vector, $H(u)$ is differentiable. Bregman Iterations are given in Algorithm 1.

Algorithm 1 Bregman Iteration

$p^0 = 0$
while "Not converged" **do**
  $u^{k+1} = \arg\min J(u) - \langle p^k, u\rangle + \lambda H(u)$ where $p^k \in \partial J(u^k)$
  $p^{k+1} = p^k - \lambda \nabla H(u^{k+1})$
end while

In the case $H(u, f) = \frac{1}{2}\|Au - f\|_2^2$, where A is a linear operator, this algorithm becomes Algorithm 2.

Algorithm 2 Bregman Iteration: the quadratic regularization case

$p^0 = 0, f^0 = 0$
while "Not converged" **do**
  $u^{k+1} = \arg\min J(u) + \lambda H(u, f^k)$
  $f^{k+1} = f^k + (f - Au^{k+1})$
end while

1.3 Linearized Bregman Iteration

When $J(u)$ is separable (like $J(u) = \|u\|_1$), we can use a first order approximation of $H(u)$ and then the algorithm becomes Algorithm 3.

1.4 Split Bregman Iteration

The Split Bregman Iteration is really dedicated for L^1 minimization problems in the form

$$\hat{u} = \arg\min \|\phi(u)\|_1 + E(u) \tag{1.2}$$

Algorithm 3 Linearized Bregman Iteration

$p^0 = 0$
while "Not converged" **do**
$u^{k+1} = \arg\min J(u) - \langle \lambda \nabla H(u^k) - p^k, u \rangle + \frac{1}{2\delta}\|u - u^k\|_2^2$
$p^{k+1} = p^k - \lambda \nabla H(u^{k+1})$
end while

where E is convex and ϕ is convex and differentiable. The idea is to split the problem into subproblem which are more easy to solve. Then problem 1.2 can be rewritten as

$$(\hat{u}, \hat{d}) = \arg\min \|d\|_1 + E(u) \qquad \text{subject to} \quad \phi(u) = d \tag{1.3}$$

If we write $H(u,d) = \frac{1}{2}\|d - \phi(u)\|_2^2$, we can use the Bregman Iteration formalism to solve this problem:

$$\begin{cases} (u^{k+1}, d^{k+1}) = \arg\min J(u,d) - \langle p_u^k, u - u^k \rangle - \langle p_d^k, d - d^k \rangle + \lambda H(u,d) \\ p_u^{k+1} = p_u^k - \lambda \nabla_u H(u^{k+1}, d^{k+1}) \\ p_d^{k+1} = p_d^k - \lambda \nabla_d H(u^{k+1}, d^{k+1}) \end{cases} \tag{1.4}$$

where $\nabla_u H(u,d) = (\nabla\phi(u))^*(\phi(u) - d)$ and $\nabla_d H(u,d) = d - \phi(u)$. Like in the first section, these iterations can be simplified by

$$\begin{cases} (u^{k+1}, d^{k+1}) = \arg\min \|d\|_1 + E(u) + \frac{\lambda}{2}\|d - \phi(u) - b^k\|_2^2 \\ b^{k+1} = b^k + \phi(u^{k+1}) - d^{k+1} \end{cases} \tag{1.5}$$

The first line can be decoupled into two subproblems, minimizing over u with d^k fixed and next minimizing over d with u^{k+1} fixed. We finally get the Split Bregman Iteration algorithm is given in Algorithm 4.

Algorithm 4 Split Bregman Iteration

$d^0 = 0, b^0 = 0$
while "Not converged" **do**
$u^{k+1} = \arg\min E(u) + \frac{\lambda}{2}\|d^k - \phi(u) - b^k\|_2^2$
$d^{k+1} = \arg\min \|d\|_1 + \frac{\lambda}{2}\|d - \phi(u^{k+1}) - b^k\|_2^2$
$b^{k+1} = b^k + \phi(u^{k+1}) - d^{k+1}$
end while

Chapter 2

Reconstruction of sparse signal

In this application, we want to recover a sparse signal u from its altered version f. We assume that u is altered by a known linear operator A. Then a useful way to recover u is by using an L^1 minimizing scheme:

$$\hat{u} = \arg\min \|u\|_1 + \frac{\mu}{2}\|Au - f\|_2^2 \tag{2.1}$$

By setting $\phi = I$ (identity) and $E(u) = \frac{\mu}{2}\|Au - f\|_2^2$, we can use the split Bregman iteration algorithm:

$$\begin{cases} u^{k+1} = \arg\min \frac{\mu}{2}\|Au - f\|_2^2 + \frac{\lambda}{2}\|d^k - u - b^k\|_2^2 \\ d^{k+1} = \arg\min \|d\|_1 + \frac{\lambda}{2}\|d - u^{k+1} - b^k\|_2^2 \\ b^{k+1} = b^k + u^{k+1} - d^{k+1} \end{cases} \tag{2.2}$$

To solve u^{k+1}:

$$\partial_u \left(\frac{\mu}{2}\|Au - f\|_2^2 + \frac{\lambda}{2}\|d^k - u - b^k\|_2^2 \right) = 0 \tag{2.3}$$

$$\Leftrightarrow \mu A^*(Au^{k+1} - f) - \lambda(d^k - u^{k+1} - b^k) = 0 \tag{2.4}$$

$$\Leftrightarrow u^{k+1} = (\mu A^* A + \lambda I)^{-1}(\mu A^* f + \lambda(d^k - b^k)) \tag{2.5}$$

where A^* is the adjoint operator of A. Solving for d^{k+1} corresponds to a shrinkage:

$$d^{k+1} = shrink(u^{k+1} + b^k, 1/\lambda) \tag{2.6}$$

where the shrinkage operator is applied on each component i of the vector:

$$shrink(u_i, \delta) = sign(u_i)\max(0, |u_i| - \delta) \tag{2.7}$$

Then the corresponding algorithm is given in Algorithm 5 (see corresponding Matlab code in Appendix A).

Algorithm 5 Sparse recovering by Split Bregman Iteration

$d^0 = 0, b^0 = 0$
while "Not converged" **do**
  $u^{k+1} = (\mu A^* A + \lambda I)^{-1}(\mu A^* f + \lambda(d^k - b^k))$
  $d^{k+1} = shrink(u^{k+1} + b^k, 1/\lambda)$
  $b^{k+1} = b^k + u^{k+1} - d^{k+1}$
end while

Chapter 3

ROF denoising

3.1 Anisotropic case

In this example, we consider the anisotropic denoising Rudin-Osher-Fatemi model: we want to retrieve the denoised image u from the acquired image f (see [3]). The proposed model is

$$\hat{u} = \arg\min \|\nabla_x u\|_1 + \|\nabla_y u\|_1 + \frac{\mu}{2}\|u - f\|_2^2 \tag{3.1}$$

Following the Split Bregman Iteration notations, we denote $d_x = \phi_x(u) = \nabla_x u$, $dy = \phi_y(u) = \nabla_y u$, $E(u) = \frac{\mu}{2}\|u - f\|_2^2$, then the corresponding model is

$$\begin{cases} (u^{k+1}, d_x^{k+1}, d_y^{k+1}) = \arg\min \|d_x\|_1 + \|d_y\|_1 + E(u) + \frac{\lambda}{2}\|d_x - \phi_x(u) - b_x^k\|_2^2 + \frac{\lambda}{2}\|d_y - \phi_y(u) - b_y^k\|_2^2 \\ b_x^{k+1} = b_x^k + \phi_x(u^{k+1}) - d_x^{k+1} \\ b_y^{k+1} = b_y^k + \phi_y(u^{k+1}) - d_y^{k+1} \end{cases} \tag{3.2}$$

Solving the first equation with respect to d_x and d_y is like in the previous example given by shrinkage:

$$\begin{cases} d_x^{k+1} = shrink(\nabla_x u^{k+1} + b_x^k, 1/\lambda) \\ d_y^{k+1} = shrink(\nabla_y u^{k+1} + b_y^k, 1/\lambda) \end{cases} \tag{3.3}$$

Solve for u:

$$\partial_u \left(\|d_x\|_1 + \|d_y\|_1 + \frac{\mu}{2}\|u - f\|_2^2 + \frac{\lambda}{2}\|d_x - \phi_x(u) - b_x^k\|_2^2 + \frac{\lambda}{2}\|d_y - \phi_y(u) - b_y^k\|_2^2 \right) = 0 \tag{3.4}$$

$$\Leftrightarrow \mu(u - f) - \lambda\nabla_x^T(d_x^k - \nabla_x u - b_x^k) - \lambda\nabla_y^T(d_y^k - \nabla_y - b_y^k) = 0 \tag{3.5}$$

$$\Leftrightarrow (\mu I - \lambda\Delta)u = \mu f - \lambda \text{div}\,(d^k - b^k) \tag{3.6}$$

where $d^k = (d_x^k, d_y^k)^T$ and $b^k = (b_x^k, b_y^k)^T$. We can solve this equation in the Fourier domain (in [3], the authors use a discretized version of the Laplacian and a Gauss-Seidel scheme):

$$(\mu\hat{I} - \lambda\Re(\hat{\Delta}))\hat{U} = \mu\hat{F} - \lambda\widehat{\text{div}\,(d^k - b^k)} \tag{3.7}$$

$$\Leftrightarrow \hat{U} = (\mu\hat{I} - \lambda\Re(\hat{\Delta}))^{-1}\left[\mu\hat{F} - \lambda\widehat{\text{div}\,(d^k - b^k)}\right] \tag{3.8}$$

where the hat symbol stands for the Fourier transform, $\Re$ is the real part and inverse and multiplication are pointwise.

Finally the global algorithm is given in Algorithm 6 (see corresponding Matlab code in Appendix B).

Algorithm 6 Anisotropic ROF denoising

$u^0 = f, d_x^0 = 0, d_y^0 = 0, b_x^0 = 0, b_y^0 = 0$
while "Not converged" **do**
  Update u^{k+1} by using equation (3.8)
  $d_x^{k+1} = shrink(\nabla_x u^{k+1} + b_x^k, 1/\lambda)$
  $d_y^{k+1} = shrink(\nabla_y u^{k+1} + b_y^k, 1/\lambda)$
  $b_x^{k+1} = b_x^k + \nabla_x u^{k+1} - d_x^{k+1}$
  $b_y^{k+1} = b_y^k + \nabla_y u^{k+1} - d_y^{k+1}$
end while

3.2 Isotropic case

This case is similar to the previous one except that we consider the isotropic total variation (see [3] for more details).

$$\hat{u} = \arg\min \sqrt{|\nabla_x u|^2 + |\nabla_y u|^2} + \frac{\mu}{2}\|u - f\|_2^2 \tag{3.9}$$

Like in the previous section, we denote $d_x = \nabla_x u$ and $d_y = \nabla_y u$, then the only difference with the anisotropic case is concerning the minimization with respect with d_x and d_y. If we denote

$$s^k = \sqrt{|\nabla_x u^k + b_x^k|^2 + |\nabla_y u^k + b_y^k|^2} \tag{3.10}$$

then d_x and d_y are updated by

$$d_x^{k+1} = max(s^k - 1/\lambda), 0)\frac{\nabla_x u^k + b_x^k}{s^k} \tag{3.11}$$

$$d_y^{k+1} = max(s^k - 1/\lambda), 0)\frac{\nabla_y u^k + b_y^k}{s^k} \tag{3.12}$$

and finally the algorithm is the one depicted in Algorithm 7.

Algorithm 7 Isotropic ROF denoising

$u^0 = f, d_x^0 = 0, d_y^0 = 0, b_x^0 = 0, b_y^0 = 0$
while "Not converged" **do**
  Update u^{k+1} by using equation (3.8)
  Compute $s^k = \sqrt{|\nabla_x u^k + b_x^k|^2 + |\nabla_y u^k + b_y^k|^2}$
  $d_x^{k+1} = max(s^k - 1/\lambda), 0)\frac{\nabla_x u^k + b_x^k}{s^k}$
  $d_y^{k+1} = max(s^k - 1/\lambda), 0)\frac{\nabla_y u^k + b_y^k}{s^k}$
  $b_x^{k+1} = b_x^k + \nabla_x u^{k+1} - d_x^{k+1}$
  $b_y^{k+1} = b_y^k + \nabla_y u^{k+1} - d_y^{k+1}$
end while

Chapter 4

Non-Blind TV Deconvolution

The model is the same as previously except that a convolution kernel appears (see [2, 3]).

$$\hat{u} = \arg\min \|\nabla_x u\|_1 + \|\nabla_y u\|_1 + \frac{\mu}{2}\|K \star u - f\|_2^2 \tag{4.1}$$

By using the same notations and considering that we can write the convolution operator as a linear operator A: $d_x = \phi_x(u) = \nabla_x u$, $dy = \phi_y(u) = \nabla_y u$, $E(u) = \frac{\mu}{2}\|Au - f\|_2^2$, then the corresponding model is

$$\begin{cases} (u^{k+1}, d_x^{k+1}, d_y^{k+1}) = \arg\min \|d_x\|_1 + \|d_y\|_1 + E(u) + \frac{\lambda}{2}\|d_x - \phi_x(u) - b_x^k\|_2^2 + \frac{\lambda}{2}\|d_y - \phi_y(u) - b_y^k\|_2^2 \\ b_x^{k+1} = b_x^k + \phi_x(u^{k+1}) - d_x^{k+1} \\ b_y^{k+1} = b_y^k + \phi_y(u^{k+1}) - d_y^{k+1} \end{cases} \tag{4.2}$$

Solving the first equation with respect to d_x and d_y is like in the previous example given by shrinkage:

$$\begin{cases} d_x^{k+1} = shrink(\nabla_x u^{k+1} + b_x^k, 1/\lambda) \\ d_y^{k+1} = shrink(\nabla_y u^{k+1} + b_y^k, 1/\lambda) \end{cases} \tag{4.3}$$

Solve for u:

$$\partial_u \left(\|d_x\|_1 + \|d_y\|_1 + \frac{\mu}{2}\|Au - f\|_2^2 + \frac{\lambda}{2}\|d_x - \phi_x(u) - b_x^k\|_2^2 + \frac{\lambda}{2}\|d_y - \phi_y(u) - b_y^k\|_2^2 \right) = 0 \tag{4.4}$$

$$\Leftrightarrow \mu A^*(Au - f) - \lambda\nabla_x^T(d_x^k - \nabla_x u - b_x^k) - \lambda\nabla_y^T(d_y^k - \nabla_y u - b_y^k) = 0 \tag{4.5}$$

$$\Leftrightarrow (\mu A^* A - \lambda\Delta)u = \mu A^* f - \lambda \mathrm{div}\,(d^k - b^k) \tag{4.6}$$

Then we can use the Fourier tranform to solve this system and the equivalent problem is

$$\left(\frac{\mu}{\lambda}|\hat{A}|^2 - \Re(\hat{\Delta})\right)\hat{U} = \frac{\mu}{\lambda}\bar{\hat{A}}\hat{F} - \mathrm{div}\,\widehat{(d^k - b^k)} \tag{4.7}$$

and then

$$\hat{U} = \left(\frac{\mu}{\lambda}|\hat{A}|^2 - \Re(\hat{\Delta})\right)^{-1}\left[\frac{\mu}{\lambda}\bar{\hat{A}}\hat{F} - \mathrm{div}\,\widehat{(d^k - b^k)}\right] \tag{4.8}$$

where the hat symbol stands for the Fourier transform, $\Re$ is the real part and inverse and multiplication are pointwise.

Then the complete algorithm is given in Algorithm.8 (see corresponding Matlab code in Appendix C).

Algorithm 8 Non-blind TV deconvolution

$u^0 = f, d_x^0 = 0, d_y^0 = 0, b_x^0 = 0, b_y^0 = 0$
while "Not converged" **do**
  Update u^{k+1} by using the inverse Fourier transform of equation (4.8)
  $d_x^{k+1} = shrink(\nabla_x u^{k+1} + b_x^k, 1/\lambda)$
  $d_y^{k+1} = shrink(\nabla_y u^{k+1} + b_y^k, 1/\lambda)$
  $b_x^{k+1} = b_x^k + \nabla_x u^{k+1} - d_x^{k+1}$
  $b_y^{k+1} = b_y^k + \nabla_y u^{k+1} - d_y^{k+1}$
end while

Chapter 5

Non-Blind sparse Frame deconvolution

Let D and D^T be respectively a frame decomposition and frame reconstruction operators, see [1]. We assume that we have $D^T D = I$ (we have a tight frame). The provided source code use both Framelet or Curvelet frames but others can be used. Then we are interested to find the restored image that have a sparse frame decomposition.

5.1 Analysis approach

The corresponding model is

$$\hat{u} = \arg\min \|Du\|_1 + \frac{\mu}{2}\|Au - f\|_2^2 \tag{5.1}$$

By setting $d = Du$, we can use the split Bregman iteration to solve this problem:

$$\begin{cases} u^{k+1} = \arg\min \frac{\mu}{2}\|Au - f\|_2^2 + \frac{\lambda}{2}\|d^k - Du - b^k\|_2^2 \\ d^{k+1} = \arg\min \|d\|_1 + \frac{\lambda}{2}\|d - Du^{k+1} - b^k\|_2^2 \\ b^{k+1} = b^k + Du^{k+1} - d^{k+1} \end{cases} \tag{5.2}$$

Then solving for d^{k+1} is equivalent to use the shrinkage operator:

$$d^{k+1} = shrink(Du^{k+1} + b^k, 1/\lambda) \tag{5.3}$$

Now solving for u comes from

$$\partial_u \left(\frac{\mu}{2}\|Au - f\|_2^2 + \frac{\lambda}{2}\|d^k - Du - b^k\|_2^2 \right) = 0 \tag{5.4}$$

$$\Leftrightarrow \mu A^*(Au - f) - \lambda D^T(d^k - Du - b^k) = 0 \tag{5.5}$$

$$\Leftrightarrow (\mu A^* A + \lambda I)u - \mu A^* f - \lambda D^T(d^k - b^k) = 0 \tag{5.6}$$

Like in the previous chapter, we use a Fourier transform to solve this problem:

$$\Leftrightarrow \hat{U}^{k+1} = \left(\mu|\hat{A}|^2 + \lambda\right)^{-1} (\mu \bar{\hat{A}}\hat{F} + \lambda \widehat{D^T(d^k - b^k)}) \tag{5.7}$$

We remind that the hat symbol stands for the Fourier transform and inverse and multiplication are pointwise.

Then the complete algorithm is given in Algorithm.9 (see corresponding Matlab code in Appendix D).

5.2 Synthesis approach

Here the model is

$$\hat{d} = \arg\min \|d\|_1 + \frac{\mu}{2}\|AD^T d - f\|_2^2 \tag{5.8}$$

Algorithm 9 Non-blind Frame deconvolution (Analysis Approach)

$u^0 = 0, d^0 = 0, b^0 = 0$
while "Not converged" **do**
  Update u^{k+1} by using the inverse Fourier transform of equation (5.7)
  $d^{k+1} = shrink(Du^{k+1} + b^k, 1/\lambda)$
  $b^{k+1} = b^k + Du^{k+1} - d^{k+1}$
end while

The restored image at the end is $u = D^T d$. We can directly use the linearized Bregman iteration to solve this model.

$$\begin{cases} c^{k+1} = c^k - \delta D A^T (A D^T d^k - f) \\ d^{k+1} = \arg\min \|d\|_1 + \frac{\mu}{2}\|d - c^{k+1}\|^2 \end{cases} \tag{5.9}$$

or a modified version by adding a preconditioner term $(AA^T - \lambda I)^{-1}$ to accelerate the convergence:

$$\begin{cases} c^{k+1} = c^k - \delta D A^T (AA^T + \lambda I)^{-1} (A D^T d^k - f) \\ d^{k+1} = \arg\min \|d\|_1 + \frac{\mu}{2}\|d - c^{k+1}\|^2 \end{cases} \tag{5.10}$$

Thus, the Fourier transform can be used to solve c^{k+1}. Denote $P = (AA^T + \lambda I)^{-1}$ the preconditioner, then

$$\hat{P} = (|\hat{A}|^2 + \lambda)^{-1} \tag{5.11}$$

Denote $T_d^k = D^T d^k$, $F_W = A^T P A$, $F_F = A^T P f$, we have $\widehat{F_W} = \bar{\hat{A}} \hat{P} \hat{A}$ and $\widehat{F_F} = \bar{\hat{A}} \hat{P} \hat{F}$. Then

$$c^{k+1} = c^k - \delta D(\widehat{F_W} \widetilde{\widehat{T_d^k}} - \widehat{F_F}) \tag{5.12}$$

Then the complete algorithm is given in Algorithm.10 (see corresponding Matlab code in Appendix D).

Algorithm 10 Non-blind Frame deconvolution (Synthesis Approach)

$c^0 = 0, d^0 = 0$
while "Not converged" **do**
  Update c^{k+1} by using equation 5.12
  $d^{k+1} = shrink(c^{k+1}, 1/\mu)$
end while
$u = D^T d^N$

Appendix A

Matlab source code for L^1-sparse recovery by Split Bregman Iteration

```
function u=L1_SplitBregmanIteration(f,A,mu,lambda,err)

%=======================================
%
% L1 Split Bregman Iteration
% Version:
% - v1.0 - 03/31/2011
%
% This function compute the solution
% u = arg min |u|+0.5*mu||Au-f||_2^2
%
% by using the Split Bregman Iteration
%
% In this version we consider only
% the case where A is a square matrix
%
% f: measured data
% A: some linear operator in its matrix
%    form
% mu: regularization coefficient
% lambda: "spliting" regularization
%         coefficient
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
% Note: typically mu=10, lambda=1,
%       eps=0.01 work well
%=======================================
N=size(f,1)

d=zeros(N,1);
b=zeros(N,1);
u=zeros(N,1);

Z=zeros(N,1);
Ft=mu*A'*f;
IV=inv(mu*A'*A+lambda*eye(N));
```

```
up=ones(N,1);

while ((u-up)'*(u-up))>eps,
   up=u; %store the previous iteration
   u=IV*(Ft+lambda*(d-b));   %update u
   tmp=u+b;
   d=sign(tmp).*max(Z,abs(tmp)-1/lambda);   %update d
   b=tmp-d;  %update b
end
```

Appendix B

Matlab source code for ROF denoising

B.1 Anisotropic case

```
function u=ATV_ROF(f,mu)

%=============================================
% function u=ATV_ROF(f,mu)
%
% Anisotropic ROF denoising
% Version:
% -v1.0 - 04/04/2011
%
% This function performs the minimization of
% u=arg min |D_x u|+|D_y u|+0.5*mu*||u-f||_2^2
%
% by the Split Bregman Iteration
%
% f = noisy image
% mu = regularization parameter
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
% Note: mu=10 performs well
%
%=============================================

[M,N]=size(f);

f=double(f);
dx=zeros(M,N);
dy=zeros(M,N);
bx=zeros(M,N);
by=zeros(M,N);
u=f;
Z=zeros(M,N);

Mask=zeros(M,N);
Mask(1,1) = 1;
FMask=fft2(Mask);
```

```
lambda=100;

%Fourier Laplacian mask initialization
D = zeros(M,N);
D([end,1,2],[end,1,2]) = [0,1,0;1,-4,1;0,1,0];
FD=fft2(D);

%Fourier constant initialization
FW=((mu/lambda)*abs(FMask).^2-real(FD)).^-1;
FF=(mu/lambda)*conj(FMask).*fft2(f);

err=1;
tol=1e-3*norm(f(:),2);
while err>tol,
    tx=dx-bx;
    ty=dy-by;
    up=u;
    %Update u
    u=real(ifft2(FW.*(FF-fft2(tx-tx(:,[1,1:N-1])+ty-ty([1,1:M-1],:)))));
    ux=u-u(:,[1,1:N-1]);
    uy=u-u([1,1:M-1],:);

    %Update dx
    tmpx=ux+bx;
    dx=sign(tmpx).*max(Z,abs(tmpx)-1/lambda);

    %Update dy
    tmpy=uy+by;
    dy=sign(tmpy).*max(Z,abs(tmpy)-1/lambda);

    %Update bx and by
    bx=tmpx-dx;
    by=tmpy-dy;

    err=sum(sum((up-u).^2));
end
```

B.2 Isotropic case

```
function u=ITV_ROF(f,mu)

%===========================================
% function u=ITV_ROF(f,mu)
%
% Isotropic ROF denoising
% Version:
% -v1.0 - 10/30/2011
%
% This function performs the minimization of
% u=arg min sqrt(|D_x u|^2+|D_y u|^2)+0.5*mu*||u-f||_2^2
%
% by the Split Bregman Iteration
%
% f = noisy image
% mu = regularization parameter
%
```

```
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
% Note: mu=10 performs well
%
%==========================================

[M,N]=size(f);

f=double(f);
dx=zeros(M,N);
dy=zeros(M,N);
bx=zeros(M,N);
by=zeros(M,N);
s=zeros(M,N);
u=f;
Z=zeros(M,N);

lambda=100;

Mask=zeros(M,N);
Mask(1,1) = 1;
FMask=fft2(Mask);

%Fourier Laplacian mask initialization
D = zeros(M,N);
D([end,1,2],[end,1,2]) = [0,1,0;1,-4,1;0,1,0];
FD=fft2(D);

%Fourier constant initialization
FW=((mu/lambda)*abs(FMask).^2-real(FD)).^-1;
FF=(mu/lambda)*conj(FMask).*fft2(f);

err=1;
tol=1e-3*norm(f(:),2);
while err>tol,
%while K<Niter,
    tx=dx-bx;
    ty=dy-by;

    up=u;
    %Update u
    u=real(ifft2(FW.*(FF-fft2(tx-tx(:,[1,1:N-1])+ty-ty([1,1:M-1],:)))));
    ux=u-u(:,[1,1:N-1]);
    uy=u-u([1,1:M-1],:);


    tmpx=ux+bx;
    tmpy=uy+by;

    s=sqrt(tmpx.^2+tmpy.^2);

    thresh=max(Z,s-1/lambda)./max(1e-12,s);

    %Update dx
    dx=thresh.*tmpx;
```

```
    %Update dy
    dy=thresh.*tmpy;

    %Update bx and by
    bx=tmpx-dx;
    by=tmpy-dy;

    err=sum(sum((up-u).^2));
end
```

Appendix C

Matlab source code for Non-blind Anisotropic TV deconvolution

```
function u=ATV_NB_Deconvolution(f,A,mu,lambda,Niter)

%=============================================
% function u=ATV_NB_Deconvolution(f,A,mu,lambda,Niter)
%
% Anisotropic TV Non-Blind Deconvolution
% Version:
% -v1.0 - 04/05/2011
%
% This function performs the minimization of
% u=arg min |D_x u|+|D_y u|+0.5*mu*||Au-f||_2^2
%
% by Split Bregman Iteration
%
% f = noisy image
% A = convolution kernel
% mu = regularization parameter
% lambda = fidelity factor for the split variables
% Niter = number of iterations
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
% Note: for a gaussian blur with sigma=1,
%  mu=500, lambda=1 and Niter=5 perform well
%
%=============================================

[M,N]=size(f);

%Structures and constants initialization
f=double(f);
dx=zeros(M,N);
dy=zeros(M,N);
bx=zeros(M,N);
by=zeros(M,N);
u=f;
Z=zeros(M,N);
```

```
up=ones(M,N);

a=lambda/(mu+8*lambda);
b=mu/lambda;
K=0;

%Fourier kernel mask initialization
Mask=zeros(M,N);
[H,L]=size(A);
Mask([end+1-floor(H/2):end,1:ceil(H/2)],[end+1-floor(L/2):end,1:ceil(L/2)]) = A;
FMask=fft2(Mask);

%Fourier Laplacian mask initialization
D = zeros(M,N);
D([end,1,2],[end,1,2]) = [0,1,0;1,-4,1;0,1,0];
FD=fft2(D);

%Fourier constant initialization
FW=((mu/lambda)*abs(FMask).^2-real(FD)).^-1;
FF=(mu/lambda)*conj(FMask).*fft2(f);

%Bregman iterations
while K<Niter,
    K=K+1;
    up=u;
    tx=dx-bx;
    ty=dy-by;

    %Update u
    u=real(ifft2(FW.*(FF-fft2(tx-tx(:,[1,1:N-1])+ty-ty([1,1:M-1],:)))));
    ux=u-u(:,[1,1:N-1]);
    uy=u-u([1,1:M-1],:);

    %Update dx
    tmpx=ux+bx;
    dx=sign(tmpx).*max(Z,abs(tmpx)-1/lambda);

    %Update dy
    tmpy=uy+by;
    dy=sign(tmpy).*max(Z,abs(tmpy)-1/lambda);

    %Update bx and by
    bx=tmpx-dx;
    by=tmpy-dy;
end
```

Appendix D

Matlab source code for Non-blind Frame deconvolution

D.1 Framelet - Analysis Approach

```
function u=Framelet_NB_Deconvolution(f,A,mu,lambda,Niter,frame,NLevel)

%=============================================================
%
% u=Framelet_NB_Deconvolution(f,A,mu,lambda,Niter,frame,NLevel)
%
% This function performs the Nonblind Framelet deconvolution by
% the analysis approach.
% -v 1.0 : 05/05/2011
%
% Parameters:
% f: input blurred image
% A: input blur kernel
% mu,lambda: regularization parameters
% Niter: number of iterations
% frame: type of used Framelet (0=Haar, 1=Piecewise Linear
%        Framelet, 2=Piecewise Cubic Framelet)
% NLevel: number of scale used in the Framelet decomposition
%
% NB: mu=10000,lambda=20,Niter=5 with Haar and NLevel=3
% performs well for a 3x3 disc kernel.
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
%=============================================================

%Generation of Framelet filters
[D,R]=GenerateFrameletFilter(frame);
nD=length(D);

%structures initialization
[M,N]=size(f);
U=FraDecMultiLevel(zeros(M,N),D,NLevel);
d=U;
b=U;
```

```
%Fourier mask + initialization of Fourier constant quantities
Mask=zeros(M,N);
[H,L]=size(A);
Mask([end+1-floor(H/2):end,1:ceil(H/2)],[end+1-floor(L/2):end,1:ceil(L/2)]) = A;
FMask=fft2(Mask);

FW=(mu*abs(FMask).^2+lambda).^-1;
FF=mu*conj(FMask).*fft2(f);

K=0;

while K<Niter,
    K=K+1
    %update u
    tx=FraRecMultiLevel(SubFrameletArray(d,b),R,NLevel);
    u=real(ifft2(FW.*(FF+lambda*fft2(tx))));

    %update d
    U=AddFrameletArray(FraDecMultiLevel(u,D,NLevel),b);
    d=ShrinkFramelet(U,1/lambda);

    %update b
    b=SubFrameletArray(U,d);
end
```

Where `FracDecMultiLevel()`,`FracRecMultiLevel()`,`ShrinkFramelet()`, `AddArray()` and `SubArray()` are functions which respectively perform the Framelet decomposition, reconstruction and shrinkage and the sum and substraction of Framelet coefficients.

D.2 Framelet - Synthesis Approach

```
function u=Framelet_NB_Deconvolution2(f,A,mu,lambda,delta,Niter,frame,NLevel)

%===============================================================
%
% u=Framelet_NB_Deconvolution2(f,A,mu,lambda,Niter,frame,NLevel)
%
% This function performs the Nonblind Framelet deconvolution by
% the synthesis approach.
% -v 1.0: 05/05/2011
%
% Parameters:
% f: input blurred image
% A: input blur kernel
% mu,lambda: regularization parameters
% delta: gradient descent speed
% Niter: number of iterations
% frame: type of used Framelet (0=Haar, 1=Piecewise Linear
%        Framelet, 2=Piecewise Cubic Framelet)
% NLevel: number of scale used in the Framelet decomposition
%
% NB: mu=1000,lambda=10,delta=20,Niter=50 with Haar and NLevel=3
% perform well for a 3x3 disc kernel.
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
```

```
% email: jegilles@math.ucla.edu
%
%=============================================================

%Generation of Framelet filters
[D,R]=GenerateFrameletFilter(frame);
nD=length(D);

%structures initialization
[M,N]=size(f);
U=FraDecMultiLevel(zeros(M,N),D,NLevel);
d=U;
c=U;

%Fourier mask + initialization of Fourier constant quantities
Mask=zeros(M,N);
[H,L]=size(A);
Mask([end+1-floor(H/2):end,1:ceil(H/2)],[end+1-floor(L/2):end,1:ceil(L/2)]) = A;
FMask=fft2(Mask);

P=(abs(FMask).^2+lambda).^-1;
FW=delta*conj(FMask).*P.*FMask;
FF=delta*conj(FMask).*P.*fft2(f);

K=0;

while K<Niter,
    K=K+1
    %update c
    tx=FW.*fft2(FraRecMultiLevel(d,R,NLevel))-FF;
    c=SubFrameletArray(c,FraDecMultiLevel(real(ifft2(tx)),D,NLevel));

    %update d
    d=ShrinkFramelet(c,1/mu);
end

tu=FraRecMultiLevel(d,R,NLevel);
u=bilateralFilter(tu);
```

D.3 Curvelet - Analysis Approach

```
function u=Curvelet_NB_Deconvolution(f,A,mu,lambda,Niter,NLevel)

%=====================================================================
%
% function u=Curvelet_NB_Deconvolution(f,A,mu,lambda,Niter,NLevel)
%
% This function performs the Nonblind Curvelet deconvolution by
% the analysis approach.
% -v1.0 - 05/04/2011
%
% Parameters:
% f: input blurred image
% A: input blur kernel
% mu,lambda: regularization parameters
% Niter: number of iterations
```

```
% NLevel: number of scale used in the Curvelet decomposition
%
% NB: mu=10000,lambda=10,Niter=10 and NLevel=3 perform well.
%
% Author: Jerome Gilles
% Institution: UCLA - Math Department
% email: jegilles@math.ucla.edu
%
%=====================================================================


%structures initialization
[M,N]=size(f);
U=fdct_wrapping(zeros(M,N),1,1,NLevel);
d=U;
b=U;

%Fourier mask + initialization of Fourier constant quantities
Mask=zeros(M,N);
[H,L]=size(A);
Mask([end+1-floor(H/2):end,1:ceil(H/2)],[end+1-floor(L/2):end,1:ceil(L/2)]) = A;
FMask=fft2(Mask);

FW=(mu*abs(FMask).^2+lambda).^-1;
FF=mu*conj(FMask).*fft2(f);

K=0;
%Bregman iterations
while K<Niter,
    K=K+1
    %update u
    tx=ifdct_wrapping(SubCurveletArray(d,b),1,M,N);
    u=real(ifft2(FW.*(FF+lambda*fft2(tx))));

    %update d
    U=AddCurveletArray(fdct_wrapping(u,1,1,NLevel),b);
    d=ShrinkCurvelet(U,1/lambda);

    %update b
    b=SubCurveletArray(U,d);
end
```

Bibliography

[1] Jian-Feng Cai, Stanley Osher, and Zuowei Shen. Split bregman methods and frame based image restoration. *Multiscale Modeling and Simulation*, 8(2):337–369, 2009.

[2] Pascal Getreuer. *tvreg: Variational Imaging Methods for Denoising, Deconvolution, Inpainting, and Segmentation*. UCLA Department of Mathematics.

[3] Tom Goldstein and Stanley Osher. The split bregman method for L1 regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009. UCLA CAM Report 08-29.